UNIVERSIDAD
DE LA REPUBLICA
URUGUAY

PEDECIBA Informática
Universidad de la República
Uruguay

Analítica sobre Big Data

Tesis de Maestría en Informática

Juan Francisco Rodriguez Saredo

Directora de Tesis:
Dra. Ing. Regina Motz

Director académico:
Dr. Ing. Javier Baliosián

–

Noviembre de 2018

Dr. Ing. Pablo Rodríguez Bocca

Dra. Ing. Libertad Tansini

Dr. Ing. Cristian Cechinel

–

Noviembre de 2018

Agradecimientos

A mi tutora de tesis de maestría, Regina Motz, por su guía académica, su confianza y apoyo para que pudiera finalizar este trabajo.

A mi supervisor académico, Javier Baliosián, quien me encaminó durante los primeros pasos de la maestría.

A mi abuelo, Doctor en Química Juan Francisco Saredo, quien, con su trabajo académico a lo largo de su vida, fue una inspiración para continuar estudiando.

A mi padre, Ronald, quien no pudo ver el final del esfuerzo y a mi madre, Nélida, que estuvo presente como ejemplo de que, con esfuerzo, todo es posible.

A Claudia y a mis hijas: Leticia, Ana Claudia y Carla por su constante apoyo.

RESUMEN

El análisis de grandes datos ha sustituido a la recolección de datos como nuevo «cuello de botella» en el proceso de toma de decisiones. Para extraer conocimiento de utilidad de grandes, heterogéneos y fluctuantes conjuntos de datos, se necesita de poderosos recursos computacionales y abstracciones de programación, que sean efectivamente utilizados. *Big Data* surgió junto con la aparición en el mercado de computadoras con gran capacidad de cómputo las cuales actúan en forma distribuida, pero coordinadamente, aprovechando su potente capacidad de procesamiento. Para su tratamiento se debe tener en consideración las principales características en *Big Data*: volumen de los datos generados, su variabilidad y la velocidad con que ellos se originan. Su tratamiento implica tener que emplear algoritmos específicos que alternan el uso del disco y de la memoria, reducir las dimensiones de los modelos (para facilitar la interpretabilidad o para llegar a resultados válidos), adaptar algoritmos de propósito general (como el gradiente estocástico), generar nuevos algoritmos para el procesamiento de datos originados por *streaming* y distribuir los datos entre múltiples nodos, utilizando modelos computacionales que organizan los cálculos (el más popular es *MapReduce*). Tal diversidad de abordajes es debido a las diferencias entre el *Data Mining* tradicional y la analítica aplicada a *Big Data*.

Incluso el análisis estadístico debe modificarse debido a que luego del procesamiento de los datos, el análisis predictivo en *Big Data* emplea muestras que representan a la mayoría de la población, por lo que la significación estadística no es tan apreciable como lo es en la analítica tradicional. Este hecho da lugar a nuevos métodos estadísticos para obtener conocimientos de los modelos predictivos.

En el presente documento se describen generalidades del proceso de analítica sobre *Big Data* y se presentan técnicas que pueden ser aplicadas a este tipo de problemas. A lo largo de ellas se exploran y analizan distintos algoritmos y su viabilidad para enfrentarse con datos masivos, presentándose, en algunos casos, sugerencias para su adaptación. Por otro lado se presenta un ordenamiento que incluye una clasificación y una taxonomía de los términos de minería de

datos y modelos computacionales adaptados a *Big Data*. Al estudiarse los algoritmos actuales, se identifican posibles modificaciones planteadas como casos de estudio cuya viabilidad podría ser analizada en el futuro. A su vez se presenta un caso de estudio donde algunas de las técnicas estudiadas son aplicadas al Plan Ceibal, basándose en los datos obtenidos de los usuarios, a través del análisis de una red modelada por un grafo, cuyos nodos son los centros de estudio y sus aristas están representadas por la hora en que esos centros están conectados

Palabras claves:
Big Data, Aprendizaje Automático, Clustering, Arboles de Decisión, Map Reduce.

Tabla de contenidos

Capítulo 1

Introducción

Big Data se refiere a los conjuntos de datos cuyo tamaño va más allá de la capacidad de las herramientas tradicionales de software de datos para capturar, almacenar, gestionar y analizar.

El trabajo que se reporta en esta Tesis de Maestría tiene como objetivo principal la presentación de técnicas aplicables a la analítica sobre *Big Data*, mediante la elaboración del estado del arte de esta disciplina respecto a las aplicaciones de *Data Mining* (Minería de Datos).

En la investigación abordada se dan respuesta a las siguientes interrogantes:

- ¿Los métodos tradicionales utilizados en *Data Mining* pueden aplicarse directamente sobre *datasets* generados como *Big Data*?
- ¿Cuáles son algunas de las nuevas técnicas o adaptaciones de algoritmos ya existentes que solucionan problemas de *Big Data*?
- ¿Cómo se pueden relacionar los conceptos de técnicas y de tareas tradicionales de *Data Mining* con situaciones que generan *Big Data* tales como los sistemas de recomendación o las redes sociales?

Data Mining se puede describir como una actividad interdisciplinaria donde se combinan, entre otros, la inteligencia artificial y los métodos estadísticos con el objetivo principal de descubrir información oculta en los datos [1].

Los sistemas en el mundo real (por ejemplo, sensores del clima, detección de intrusos en redes de computadoras, redes sociales y aplicaciones de comercio electrónico) producen una gran cantidad de datos, provenientes de diversas fuentes, altamente complejos y no estructurados, los cuales son difíciles de procesar. Este desafío es un problema común para organizaciones e industrias y

requiere una nueva dirección de investigación como lo planteó *Michael Stonebraker* en el año 2015 [2]. *Big Data* es un concepto que hace referencia a la recolección, procesamiento, gestión y análisis de grandes cantidades de datos, cuyo objetivo es generar conocimiento o información destinada a contribuir en el proceso de la toma de decisiones. Las tareas nombradas son enfrentadas con limitaciones por los métodos habituales, los cuales están sustentados sobre bases de datos relacionales. El desafío fundamental de las aplicaciones de *Big Data* es la exploración de grandes y heterogéneos volúmenes de datos para extraer conocimiento útil.

Big Data se identificó inicialmente en referencia a los tres principales características que la definen: velocidad (con la cual son originados los datos), variabilidad (debido a los diversos tipos de fuentes de datos que pueden ser integrados o aparición de nuevos atributos en ellos) y volumen (en relación a la enorme cantidad de datos que pueden ser generados por segundo). Actualmente (aunque no son características que definan a *Big Data*) se agregan cuatro nuevas: validez (para determinar qué datos son admitidos), veracidad (vinculado a la consistencia y confiabilidad de los datos), valor (relacionado al capital que se puede generar a partir de los datos) y visualización (la cual es crítica debido a que el uso de gráficos para observar el comportamiento de grandes cantidades de datos complejos es mucho más efectivo para trasmitir significado).

La analítica sobre *Big Data* puede ser vista como una variación de la aplicación de *Data Mining* adaptada a la gestión de grandes volúmenes de datos. El tratamiento de esta gran cantidad de datos, implica tener que distribuirlos entre múltiples nodos, debido a que cargarlos en una sola máquina, podría tomar un tiempo prolongado (existen dos posibles limitaciones: la que la memoria pueda tener en sí y el ancho de banda entre el disco y la memoria). Es necesario entonces adecuar algunos métodos tradicionales de *Data Mining* mediante nuevos algoritmos, para poder aplicar dichas técnicas a dominios exclusivos de *Big Data*, donde los datos crecen de tal forma que se tornan impredecibles, complejos y difíciles de gestionar.

1.1. ¿Por qué los métodos de Data Mining presentan limitaciones al aplicarse a Big Data?

Data Mining posee modelos que son utilizados de manera sistemática. En cambio, al tratarse de *Big Data*, existe un reto importante al intentar generar modelos generales mediante la combinación de los patrones descubiertos localmente e integrarlos en forma unificada. Estas limitaciones son debidas a las *3 v's* que caracterizan a los grandes datos: son generados masivamente y con gran variabilidad y velocidad, por lo que patrones que pueden ser detectados «localmente» pueden no existir en el futuro. Esta situación también puede repetirse en *Data Mining*, pero su evolución es lenta y, en consecuencia, es posible adaptar nuevos modelos con relativa facilidad. En *Big Data* esta evolución es altamente acelerada y si no se dispone de una adecuada solución los modelos dejan de ser eficientes.

La adaptación de las técnicas tradicionales de *Data Mining* a *Big Data* se logra atendiendo las características citadas de las *3 v's* de los grandes datos. El volumen grande de datos se puede solucionar con algoritmos adecuados que utilizan disco y memoria (esto se presenta en la *Sección 5.2*, *Algoritmos de recuperación de ítems*, donde se emplean algoritmos a pasos, como *PCY* y en la *5.8*, *Clustering*, en la cual se mencionan *BFR* y *CURE*). También existen técnicas que distribuyen los cálculos empleando *MapReduce*. Por otro lado, la velocidad con que se generan los datos, o su variabilidad al transcurrir el tiempo, pueden ser atendidas por medio de un análisis adecuado de *streams* de datos o adaptando algoritmos de propósito general (esto se presenta en el *Capítulo 4* donde se analiza el método del gradiente estocástico, resultado de la adaptación del gradiente ascendente, que permite mayor velocidad en tiempo de ejecución).

1.2. Taxonomía de términos en Big Data

La Minería de Datos tradicional diferencia los términos tarea y método. La tarea indica el problema a ser resuelto y los métodos (también llamados técnicas o algoritmos) son las herramientas utilizadas para resolver las tareas.

A los efectos de establecer una base para una adecuada taxonomía, nuestra

propuesta modifica, expandiéndolo, el concepto de «método» e incorpora uno nuevo, «aplicación», los cuales se describen a continuación.

Tareas en Data Mining

Las tareas o modelos en *Data Mining* pueden ser predictivos y descriptivos. Dentro de los predictivos, se encuentran la clasificación y la regresión. En las tareas descriptivas se encuentran el agrupamiento, reglas de asociación, correlaciones y detección de anomalías, entre otros. Por ejemplo, clasificar una población de acuerdo a cierto atributo, se considera como una tarea de clasificación. Los métodos (técnicas o algoritmos) para resolver esta tarea pueden ser: árboles de decisión, redes neuronales y regresión, entre otros.

Técnicas en Data Mining

Las técnicas (también llamados indistintamente métodos o algoritmos en *Data Mining*) son utilizados para resolver las tareas mencionadas anteriormente. Algunos ejemplos son los árboles de decisión, las redes neuronales artificiales, las máquinas de vectores de soporte, el perceptrón, vecinos más cercanos y las redes bayesianas.

Clasificación de los métodos en Big Data

El ordenamiento de *Data Mining* en base a tareas y técnicas es necesario adaptarlo para *Big Data* por las siguientes consideraciones:

- Complejidad que presenta diferenciar las tareas de los métodos (pueden confundirse en sus objetivos particulares al aplicarse a un determinado problema). Una complicación adicional puede ocurrir, por ejemplo, al observar técnicas que utilizan a otras, para su implementación. Por ejemplo, en el *Capítulo 5*, el método *LSH* (funciones de *hash* localmente sensibles), es utilizado por otra técnica, *K-NN* (k vecinos más cercanos).
- La presencia de modelos como *MapReduce* o el análisis de datos generados en un *stream*, diversifican las herramientas disponibles para resolver los problemas de analítica.

Por lo tanto, consideraremos que las tareas son los problemas a resolver y los enfoques que se utilicen para solucionarlo los llamaremos técnicas (a diferencia de *Data Mining* no se usarán indistintamente los términos algoritmos, métodos y técnicas debido a que también se pueden incluir otras estrategias de solución).

Aplicaciones

Con la masiva utilización de algunas aplicaciones de Internet, del tipo de redes sociales, o sitios de ventas, han surgido campos de investigación (sistemas de recomendación. análisis de links, de sentimientos y semántica de redes sociales, entre otros) construidos sobre *Big Data* que, en esta tesis, son denominados aplicaciones.

Denominamos «aplicaciones» a problemas de *Big Data* que se resuelven utilizando los conceptos de tarea (para identificar al problema) y de técnicas (para resolverlo).

En la Tabla 1.1 se presenta una clasificación de las tareas junto con las técnicas o algoritmos para abordarlas. Esta clasificación está basada en una tabla presentada en [3] para *Data Mining*, ampliada en este trabajo con tareas y algoritmos utilizados para *Big Data*.

Tarea / Técnica	Predictivo		Descriptivo				
	Clasificación	Regresión	Agrupamiento (Clústeres)	Reglas de Asociación	Correlaciones/ Factorizaciones	Detección de Anomalías	Sumarización
Redes Neuronales	V	V	V				V
Árboles de decisión	V	V	V	V			V
Redes de Kohonen			V				
Regresión lineal y logarítmica		V			V		
Regresión logística	V			V			
k-means			V				
A priori				V			
Vecinos más próximos	V	V	V				
Análisis factorial y de componentes principales					V		
Algoritmos genéticos	V	V	V	V	V		V
Máquinas de vectores soporte	V	V	V				V
Stream de datos	V	V	V	V	V	V	V
Análisis de grandes grafos	V		V	V	V	V	V
Ítemsets similares	V		V	V			
Ítemsets frecuentes	V	V	V				V
MapReduce	V				V		V
Redes bayesianas	V		V				V
LSH			V				V
Minhashing	V		V				V
Reducción de la dimensión	V	V	V	V	V		V
Maximización Expectación	V		V				V
Perceptrón	V		V				
Ensamble de clasificadores	V						V
Flajolet-Martin							V

Tabla 1.1: Tareas y técnicas para Big Data (Basado en: *Introducción a la Minería de Datos José Hernández, María J. Ramírez y César Ferri - 2004*)

En la Figura 1.1, en base al relevamiento llevado a cabo durante el trabajo,

se categorizan las técnicas de acuerdo a su empleo común en *Data Mining* y *Big Data* o exclusivos a cada uno de ellos. También se consideró una categoría donde algunos algoritmos son aplicados en *Big Data*.

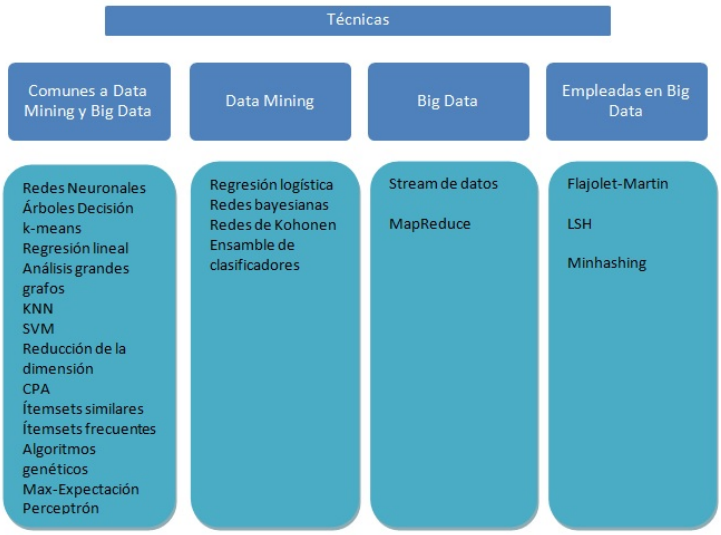

Figura 1.1: Empleo de las técnicas

1.3. Objetivos

A continuación se presentan los objetivos planteados para esta tesis de maestría.

- Presentar, en una primera aproximación, una clasificación de las técnicas de analítica originadas en *Data Mining* y su correspondiente adaptación para *Big Data*, junto con una taxonomía de los métodos disponibles, los cuales, en la literatura existente, resultan difíciles de contextualizar en el marco de una clasificación general, estableciendo la diferencia entre tareas, técnicas y aplicaciones como se expresó en la *Sección 1.2*.
- Proporcionar un exhaustivo análisis, a partir del cual sea posible disponer de la descripción de varias técnicas para el tratamiento de *Big Data*, con la correspondiente fundamentación teórica.
- Presentación de un caso de estudio donde algunas de las técnicas estudiadas son aplicadas al Plan Ceibal, basándose en los datos obtenidos de los usuarios, a través del análisis de una red modelada por un grafo, cuyos nodos son los centros de estudio y sus aristas están representadas por la hora en que esos centros están conectados.

- Identificación de posibles casos de estudio, junto a un relevamiento de posibles áreas de investigación del tema.

1.4. Contribución

La búsqueda de un ordenamiento de las aplicaciones y de los problemas asociados a *Big Data* condujo a la presentación de los conceptos adaptados de tareas, métodos y aplicaciones, que se utilizaron para efectuar una clasificación de las tareas y técnicas de *Big Data*. Esta clasificación está representada en la Tabla 1.1 y es utilizada para contextualizar las aplicaciones tal cual fueron definidas en la *Sección 1.2*.

Al analizar el flujo tradicional en el aprendizaje de máquinas, identificamos la necesidad de modificarlo para adaptarlo a *Big Data*, teniendo en cuenta que pueden originarse nuevas características y considerar que los patrones anómalos pueden ser parte de pequeñas sub-poblaciones. Considerar este escenario podría implicar que se necesite más de un modelo de aprendizaje automático para identificar el comportamiento de las distintas sub poblaciones.

En el estudio de las técnicas empleadas en *Data Mining* o en los modelos de funcionamiento de aprendizaje automático, se presentan los fundamentos teóricos que posibilitan su utilización en *Big Data* o, en caso contrario, para descartarlas. En algunos técnicas se presentan ejemplos de funcionamiento de algoritmos específicos.

A lo largo del trabajo se han identificado posibles casos de estudio relacionados con la analítica sobre *Big Data*, junto a un relevamiento de posibles áreas de investigación del tema, conjuntamente con la presentación de abundantes ejemplos para aclarar algoritmos y conceptos abordados.

En el Caso de Estudio del Plan Ceibal, se aplicaron técnicas de analítica sobre los datos generados diariamente por los usuarios, descubriéndose indicadores útiles destinados al soporte para la toma de decisiones: relación de visitas a sitios con material educativo comparados con visitas a sitios de diverso contenido, detección de horarios de máxima y mínima utilización de los recursos, exploración de clústeres y detección de *outliers* (centros de estudio que difieren en la utilización habitual de la red, en relación a los horarios).

El Caso de Estudio anterior fue presentado en el *VI CBIE (Congresso Brasileiro de Informática na Educação)*, realizado por la *Sociedade Brasileira de Computação (SBC)*, en la ciudad de *Recife, República Federativa del Brasil* en

el período del 30 de Octubre al 02 de Noviembre de 2017, bajo el nombre de «*Application of clustering techniques on data generated by an Online Educational Network*». Posteriormente, a partir de este trabajo, se presentó un segundo artículo en el *workshop LALA (Learning Analytics Latin America 2018)*, en la ciudad de *Guayaquil, República del Ecuador*, denominado *"Dibujando el mapa de tráfico de redes escolares en línea"*

1.5. Organización de este informe

El *Capítulo 2* presenta una propuesta del ciclo de vida del trabajo con *Big Data* y se enfoca en describir sus fases iniciales, desde la recolección de los datos hasta su procesamiento, incluyendo la fase de generación (*stream* de datos), el tratamiento de los datos (datos faltantes, detección de valores anómalos, etc.) y *MapReduce* (fase de procesamiento).

Una profundización en la analítica sobre datos incluyendo el concepto de modelos utilizados se expone en el *Capítulo 3*. También se presenta un ordenamiento del modelado estadístico, el cual es empleado para estructurar el presente trabajo y, debido a la importancia de obtener modelos adecuados, se presentan diversos criterios para evaluarlos, introduciendo la noción de error y costos asociados. En este capítulo, asimismo, se explora la necesidad de modificar el análisis predictivo para *Big Data* dando lugar a nuevos métodos estadísticos para obtener conocimientos de los modelos predictivos. A su vez se enumeran algunos campos de investigación actuales de analítica sobre *Big Data*.

El *Capítulo 4* habla de técnicas generales de *Data Mining* y en ellas se introduce el aprendizaje automático y una idea de su adaptación a *Big Data*. Otros algoritmos presentados son: reducción de la dimensión (método *SVD* y la descomposición *CUR*, entre otros), algoritmos de optimización y regresión. Se realiza una descripción del aprendizaje automático y, en particular, del perceptrón, describiéndose algunas limitaciones que presenta, lo que abre el camino para otras tecnologías más apropiadas como máquinas de vectores de soporte. Estas son estudiadas como una alternativa para resolver las restricciones del perceptrón y para enfrentar problemas de *Big Data* cuyos datos pueden presentar millones de características. Luego se presentan algoritmos para la reducción de la dimensión y de optimización, que son utilizados en varias de las técnicas desarrolladas, y, en particular, algunas modificaciones

que mejoran el escalado. En el mismo capítulo se aborda la regresión como un modelo paramétrico, detallándose los modelos de regresión *RIDGE* y *LASSO*.

En el *Capítulo 5* se describen varias técnicas y tareas de *Data Mining* adaptadas a *Big Data*:

- Búsqueda de artículos similares: Es un problema fundamental en *Big Data*. Se estudian conjuntos de ítems que tienen una intersección común con otros conjuntos. Asimismo se describen los conceptos de distancia y similaridad de *Jaccard, shingles, minhashing* y *LSH*.
- Itemsets frecuentes: Busca determinar el total de canastos que contienen un determinado conjunto de ítems. Se explora el algoritmo *A Priori*, el cual establece un procedimiento para descartar subconjuntos. Debido a la alta exigencia de memoria de dicho algoritmo y para optimizar su uso, se presenta el algoritmo de *PCY* y el algoritmo *Multihash*.
- Clasificación: Se desarrolla bajo la óptica de la regresión lineal y de modelos lineales generalizados empleando la regresión logística.
- Métodos de vecinos más cercanos: Son métodos que permiten que la complejidad del modelo crezca a medida que aumenta la cantidad de los datos observados. Se aborda el problema de *K-NN* en *Big Data* y se presentan técnicas para resolverlo.
- Árboles de decisión: Es apropiado para problemas con muchas variables de entrada, los cuales no pueden ser resueltos por clasificadores lineales.
- Combinación de clasificadores: Para mejorar los resultados de clasificadores individuales, se pueden combinar varios de ellos. Se describen los métodos de *Bagging, Boosting y AdaBoost*.
- Redes neuronales artificiales: Se presentan los algoritmos de propagación hacia adelante y hacia atrás, aplicando dos fases al proceso de aprendizaje, lo que se describe como un ciclo de propagación-adaptación. Al final se abordan los conceptos de redes neuronales profundas y superficiales y, a partir de allí, el concepto del aprendizaje profundo (*deep learning*).
- Métodos de *clustering*: Debido a su importancia, son tratados especialmente los métodos de clustering jerárquicos y por asignación de puntos. Estos métodos presentan un alto costo computacional al tratar grandes conjuntos de datos mencionándose algunos algoritmos que resuelven el problema. Se abordan los clústeres en modelos de *stream* de datos y los clústeres probabilísticos utilizando el modelo de mezcla de gausianas.

A continuación se enumeran algunas *aplicaciones*, que son estudiadas en el *Capítulo 6*, en el contexto planteado en la *Sección 1.2*.

- Análisis de links: Se estudia la tecnología de motores de búsqueda y se introduce el algoritmo de *Page Rank* como una tarea de sumarización.
- Sistemas de recomendación: Los recomendadores están basados en algoritmos que intentan predecir la respuesta de un usuario, frente a un conjunto de opciones. Los dos tipos más frecuentes son: sistemas basados en contenido y sistemas de filtrado colaborativo.
- Análisis de sentimientos: Se presenta un algoritmo de aprendizaje automático para su tratamiento y se describe el caso de estudio denominado análisis de sentimientos colectivo.
- Redes sociales: Se plantea el problema de identificación de comunidades. Se incursiona en el problema de búsqueda de comunidades que se superponen y se estudia el modelo de grafo de afiliación.

En el *Capítulo 7* se presenta el Caso de estudio del Plan Ceibal. Se pretende encontrar patrones o características que indiquen la existencia de horarios en el que se observan aproximadamente la misma cantidad de conexiones en centros de estudio, en determinados días.

Las conclusiones del trabajo se presentan en el *Capítulo 8*, conjuntamente con el resumen de las propuestas que han surgido como posibles líneas de trabajo futuro.

Por último, los métodos estadísticos de aprendizaje son expuestos en el *Anexo 4*. En entornos reales, la incertidumbre es el factor más importante a tener en cuenta, lo que justifica su relevancia. Se describe brevemente la teoría de la decisión bayesiana.

Capítulo 2

Big Data

El paradigma *Big Data* puede ser considerado como el sistema de Big Data y su ambiente (realidad que genera los datos), siendo el objetivo del sistema la recolección de los datos, aprender sus características y tomar decisiones correctas [1]. El paradigma planteado permite definir un ciclo de vida para el proceso completo que se presenta en la próxima sección. Por su naturaleza dinámica, los datos generalmente son originados en forma de *stream* de datos.

2.1. Ciclo de vida

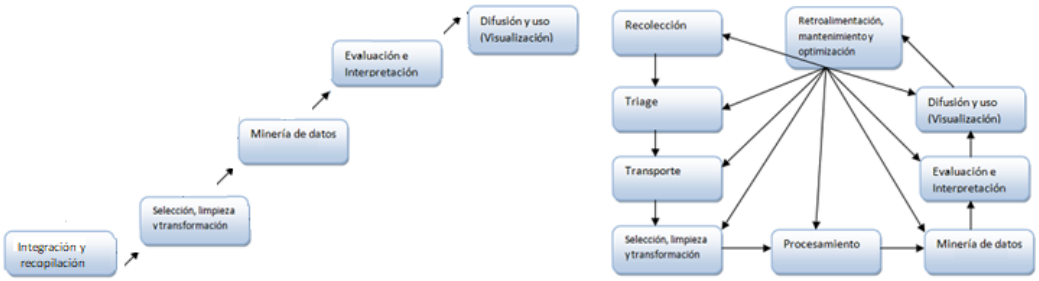

Figura 2.1: Proceso de Data Mining y Ciclo de Big Data

El proceso empleado en *Data Mining* consta, habitualmente, de las fases: Integración y recopilación de las fuentes de datos; Selección, limpieza y transformación; Minería de datos; Evaluación e interpretación; Difusión y uso [3]. Este proceso se puede adaptar a *Big Data*, debiendo ser consideradas las *3 v's*, obteniéndose un ciclo de vida. En las fases iniciales se requiere un tratamiento diferente al utilizado en el *Data Mining* tradicional, considerándose que los

datos son generados a mucha velocidad y en gran volumen. Por lo tanto, hay que seleccionar algunos de ellos y descartar otros, transportarlos y efectuar un procesamiento distribuido. En la Figura 2.1, se aprecia el proceso de *Data Mining* y el ciclo de *Big Data*. En dicho ciclo se observan las siguientes etapas:

1. Recolección: consiste en el procesamiento original de los datos (provenientes de sensores, por ejemplo).
2. *Triage*: se trata de la priorización y selección de los datos útiles en los puntos de generación.
3. Transporte: está relacionado a mejorar la eficiencia, para lograr procesar la mayor cantidad de datos en el menor tiempo posible.
4. Selección, limpieza y transformación: se eliminan o corrigen datos incompletos, y se proyectan y filtran los datos de los atributos que podrían ser de utilidad.
5. Procesamiento: esta etapa tiene la finalidad de aumentar la disponibilidad de procesamiento de los datos para generar productos. El proceso efectuado por *MapReduce* es la técnica más popular en *Big Data*.
6. Minería de datos: está asociado a qué tareas y técnicas de análisis se deben utilizar para entender los datos y encontrar relaciones y patrones ocultos en ellos.
7. Evaluación e interpretación: es la generación de conocimiento a través del análisis de la información, utilizando determinadas herramientas junto al conocimiento aportado por expertos. Esta etapa, si es necesario, puede ser iterada desde cualquier fase del ciclo.
8. Difusión y uso: consiste en la toma de decisiones y proveer servicios de analítica (como la creación de servicios de analítica sobre grandes datos distribuidos).
9. Retroalimentación, mantenimiento y optimización: en caso de cambiar los datos o los objetivos del proyecto, se puede realizar cambios sobre cualquier etapa del proceso.

De acuerdo a la cantidad y/o tipo de datos, y a la forma en que éstos se generen, existen situaciones en las que algunas de las etapas nombradas no son ejecutadas.

Dentro de la etapa de Difusión y Uso, se puede considerar a la visualización como una actividad cuya importancia se torna imprescindible para la interpretación de los resultados (es desarrollada en el *Anexo 1, Visualización*).

Las etapas dos (*triage)* y tres (transporte) no se desarrollan con más detalle ya que su estudio se aparta del objetivo del trabajo. El *triage* es necesario para descartar datos no deseados de las fuentes que los generan. Por ejemplo, en sensores de sonidos originados en el medio ambiente pueden interesar, exclusivamente, los provenientes del transporte público. Con respecto a la fase de transporte de datos, es importante considerar los desafíos que se plantean durante la transferencia entre sistemas (plataformas, bases de datos, etc.) como ser la seguridad y optimización.

2.2. Stream de datos

Considerando la generación de datos en *Big Data* y las *3 v's* características, es usual que datos que arriban en la forma de *stream* no sean almacenados (lo cual implica que deben ser procesados al ser obtenidos, ya que, en caso contrario, se perderían). Por lo tanto, al procesar un *stream*, las primeras fases del ciclo de *Big Data* se deben ejecutar de diferente manera.

A continuación se describe el problema de extraer una muestra de un *stream*, suponiendo que los datos se reciben a una velocidad y volumen tales, que no pueden ser almacenados. En consecuencia, se hace necesario procesarlos en memoria principal para que no se pierdan algunos elementos. Por lo tanto, sería necesario, de alguna forma, realizar sumarizaciones. En primer lugar, se busca tener una muestra de los datos, eliminando los que no son útiles. Otra opción, es tratar un número grande de datos como una ventana y procesarlos, abstrayéndose del resto de los datos. Sin embargo, si la ventana es muy grande, se deben efectuar sumarizaciones [4]. La tasa de arribo de los datos en el *stream* no está bajo el control del sistema y ésta es la diferencia fundamental con una base de datos (sea o no relacional). Los *streams* pueden llegar a ser almacenados en un archivo, con la limitación de que no se pueden realizar consultas (a menos que se utilicen procedimientos de recuperación, los cuales consumen mucho tiempo).

En la práctica, se dispone de un «almacenamiento de trabajo» (en memoria o en disco), que almacena las sumarizaciones, o partes de *streams* y es utilizado para responder a *queries*. Estas pueden existir de dos formas: «*standing queries*» y «*queries ad hoc*». Las primeras se encuentran almacenadas y previamente configuradas, son ejecutadas permanentemente y producen salidas en determinados momentos (por ejemplo calcular la máxima temperatura re-

cibida a través de un sensor del medio ambiente cada media hora). El otro tipo de *query* es *ad hoc*: se trata de una consulta particular sobre el *stream* en su estado actual que puede tomar como entrada a las sumarizaciones que las *standing queries* ya procesaron o sobre datos arribando en determinado momento [4].

El principal problema en el procesamiento es la memoria requerida. Para ilustrar este hecho se presenta un par de ejemplos.

Datos provenientes de cámaras de video

Aunque la resolución de cada imagen sea baja, se producen *streams* de imágenes cada un segundo aproximadamente. Al multiplicarse por muchas cámaras que pudiera haber desplegadas en una ciudad, fácilmente, se llega al orden de los terabytes en un día.

Monitoreo plataforma marítima

Para monitorear el mar correspondiente a la plataforma marítima del Uruguay (350 millas oceánicas), se necesitaría 2,3 millones de sensores, que deberían enviar 10 datos por segundo, los cuales generarían 8 terabytes de datos al día (para la confección de este ejemplo, se obtuvieron datos de la Web y está basado en un ejemplo real de [4]).

En el procesamiento de un *stream*, se tienen en cuenta los siguientes aspectos [4]:

- En ocasiones, es más eficiente tener una respuesta aproximada a un problema que una solución exacta.
- Generalmente, las técnicas basadas en funciones de *hash* resultan útiles ya que introducen aleatoriedad en la conducta del algoritmo, para producir una respuesta aproximada al resultado real.

El alcance de estos dos puntos se puede observar en el Capítulo 5, al estudiarse la similaridad de documentos donde se analiza la técnica de *Minhashing* para la detección de documentos similares. Este algoritmo resuelve en forma aproximada, pero eficientemente, el problema empleando funciones de *hash* con una probabilidad alta de que si dos documentos son similares, sus funciones de *minhash* también lo sean.

Algunos de los problemas que surgen durante la obtención de una muestra de los datos representativa se presentan en el *Anexo 2, Stream de Datos*.

Aprendizaje online

El aprendizaje online es una técnica que trata de hacer predicciones sobre los datos que llegan en línea. En el aprendizaje en línea o entrenamiento de *streaming* los datos no están disponibles al correr el algoritmo. Por lo tanto, se requiere efectuar un aprendizaje automático online (en este trabajo, al *aprendizaje automático*, se le abreviará *ML* por su equivalente en inglés: *Machine Learning*). Un ejemplo es el *ad targeting*. Cuando a un usuario de Internet se le muestra un aviso, cierta información del usuario como la edad, sitios Web que ya haya visitado e información del sitio donde se encuentra en el momento, como el texto y otra actividad que se pueda captar, son procesados por una *ML*. Esta información es utilizada para determinar cuál es el mejor aviso para mostrarle al usuario. Toda esta información origina un conjunto de coeficientes relativos al usuario, w_t. Éstos son empleados por el modelo de *ML* cuyos algoritmos sugieren algunos avisos al usuario. Los sitios de las compañías de ventas de servicios más importantes, utilizan los algoritmos online, para el análisis de los datos generados por los usuarios, que retornan al sitio *Web*. El flujo de los datos y de los usuarios se puede considerar continuo y de su análisis se pueden optimizar las decisiones sobre el sitio. Si un producto es ofrecido por un sitio a un precio p y si éste le es útil al usuario, se puede llegar a realizar una venta *(y=1)*, o no *(y=0)*. Por lo tanto, interesa establecer un algoritmo de aprendizaje para inducir la compra o para optimizar el precio a solicitarle al usuario. Cuando un usuario pulsa sobre uno de los avisos sugeridos, el algoritmo de *ML* capta que dicho usuario aceptó el aviso y se le asigna una etiqueta verdadera. Con dicha información el algoritmo de *ML* actualiza sus coeficientes de w_t a w_{t+1} (el aprendizaje automático es desarrollado en el *Capítulo 4*).

En resumen, los datos arriban cada cierto tiempo t. Se observa el *input* x (información del usuario de la página *Web*). Se hace una predicción y_t. Luego, se observa la verdadera salida, esto es, qué aviso seleccionó el usuario, actualizándose los coeficientes w_t a cada paso. El algoritmo que mejor se adaptaría a una situación donde es necesario mejorar la performance, en forma *online*, es el gradiente estocástico. Este modelo siempre está actualizado y tiene bajo costo computacional (debido a que no tiene que observar, ni almacenar todos los datos). En la *Sección 4.3, Algoritmos de optimización* y en el *Anexo 6* se describe dicha técnica.

2.2.1. Problema de muestreo y conteo en un stream

Para crear una muestra, se puede considerar una función de *hash*. Sea un flujo de datos compuesto de tuplas de n componentes, de los cuales un subconjunto de ellos pueden ser la clave. Luego, para construir la muestra, se toman los elementos del *hash* por medio de la clave.

En ocasiones interesa, antes de procesar el *stream*, seleccionar ciertas tuplas haciendo búsquedas en ellas, realizándose un filtrado, luego del cual, las tuplas aceptadas son las que se procesan.

Filtro de Bloom

El filtro de *Bloom* es una estructura de datos probabilística, concebida por *Burton Bloom*. Es usada para verificar si un elemento pertenece o no a un conjunto. Permite filtrar elementos del *stream*, usando un *array* grande de bits *h* y varias funciones de *hash k*. Inicialmente, se carga el *array h* con un conjunto de miembros aceptados, los cuales son dispersados en las cubetas del *array h*, por medio de *bits* con valor *1*. Dado un nuevo elemento proveniente del *stream* que se busca seleccionar y usando las funciones de *hash k*, se genera un conjunto de posiciones del arreglo *h*. Si los valores de estas posiciones son todos *1*, el elemento es aceptado [4]. Pueden aparecer casos de falsos positivos.

Conteo de elementos diferentes del array

Otro problema lo constituye el conteo de elementos diferentes en el *array*. El algoritmo de *Flajolet-Martin*, cuya fundamentación matemática se puede leer en [5], puede ser aplicado para resolverlo. Dicho algoritmo emplea una función de *hash h* que mapea cada elemento a un *string* de *bits*. Este *string* de *bits* debe ser suficientemente largo como para abarcar los diferentes elementos que pueda generar la función de *hash* cuyo número se supone es mayor que todos los elementos diferentes del *stream*. Por ejemplo si se quisiera procesar a todas las URLs se necesitan 64 *bits* [4]. Al aplicarse *h* a cada elemento del *stream*, el string de *bits* va a terminar con una cierta cantidad de ceros, inclusive *0*. Sea R la mayor cantidad de ceros (al final del *stream*) obtenidos hasta el momento. Se puede estimar como 2^R el número de elementos distintos en el *stream*. A los efectos ilustrativos, en el *Anexo 2, stream de Datos*, presentamos un ejemplo de su funcionamiento.

2.3. Tratamiento de los Datos

La etapa 4 del ciclo de vida de la *Sección 2.1* trata sobre la selección, limpieza y transformación de los datos, donde se eliminan o corrigen datos incompletos. En ocasiones, los datos que se van a recopilar, provienen de fuentes muy disímiles, por lo que se hace necesario una limpieza e integración de ellos. Se busca que los datos sean íntegros, consistentes y completos.

En ambientes de *Big Data* existen tres grandes problemas al momento de obtener los datos: datos incompletos (ausencia de datos de algún atributo), datos no balanceados (si los datos se pueden agrupar en clases, el tamaño de éstas puede ser muy desigual) y datos inexactos (datos que se encuentran en una clase pero en realidad pertenecen a otra) [1]. En esta sección se presentan algunas técnicas básicas para solucionar estos problemas y se introducen los conceptos de centrar, escalar y normalizar datos, los cuales permiten uniformizarlos.

Es conveniente considerar las consecuencias, en situaciones de ausencia de datos, tanto al omitir (purificación) como al ingresar datos faltantes (imputación). Puede ocurrir que un atributo no se genere debido a características particulares de la fuente de origen y, por ejemplo, utilizar la técnica de sustituirlo por su promedio, sea una mala decisión [6].

Los patrones para la omisión de datos pueden ser en un atributo fijo o en varios de ellos (también pueden ocurrir al avanzar el tiempo en forma periódica o aleatoriamente). En ocasiones, se considera que la falta de valores sigue un patrón aleatorio (diseñándose algoritmos para imputar datos bajo esta hipótesis) aunque no siempre es así (puede existir un patrón en los datos omitidos que esté asociado a las características de la población de referencia, lo cual invalida el supuesto de aleatoriedad en el que se sustenta la técnica de sustitución [6].

2.3.1. Purificación de los datos faltantes

Purificación es el proceso de no considerar los registros que contienen datos faltantes. Tal práctica no es aconsejable si la cantidad de datos incompletos es muy grande, por el hecho de que se pierden muchas observaciones. Una alternativa, es ignorar las características que faltan mediante la eliminación de columnas (en general, las columnas representan a los atributos o características de los datos). Además de ignorar características, se puede ignorar observaciones (filas) o combinaciones de ambas. Cualquiera sea el caso, la purificación debe

ser realizada una vez que se tenga la certeza, de que son pocos los registros que se eliminan respecto al total.

2.3.2. Imputación o sustitución de los datos faltantes

La imputación o sustitución es el proceso de completar los datos faltantes en lugar de eliminarlos. Una manera de lograrlo, es considerar el valor más común, si se tratase de variables categóricas.

A continuación se presenta una lista con algunos métodos de imputación [6].

- Imputación utilizando promedios: La imputación por el método de promedios, debido a sus limitaciones teóricas no se considera un procedimiento apropiado. Se ha demostrado que su aplicación afecta la distribución de probabilidad de la variable imputada, atenúa la correlación con el resto de las variables y subestima la varianza, entre otras cosas (la suma de cuadrados de las desviaciones de las observaciones respecto de la media permanece inalterada pero se incrementa el tamaño de muestra, lo cual origina que la varianza de la variable disminuya y se generen, en forma artificial, intervalos de confianza más estrechos) [6].
- Imputación mediante una distribución no condicionada (*hot-deck*): El *hot-deck* tiene como objetivo llenar los registros vacíos con información de campos con registros completos. Los datos faltantes se reemplazan a partir de una selección aleatoria de valores observados, lo cual no introduce sesgos en la varianza del estimador. El algoritmo consiste en ubicar registros completos e incompletos, identificar características comunes entre los datos que serán «donantes» y los datos faltantes y decidir cuáles se utilizarán para imputar. Para este procedimiento es importante generar agrupaciones que garanticen que la imputación se llevará a cabo entre observaciones con características comunes y que la selección de los donantes se realice en forma aleatoria para evitar la introducción de sesgos [6].
- Imputación por regresión: Es posible utilizar modelos de regresión para imputar información en la variable Y, a partir de un grupo de variables $x_1, x_2, ..., x_p$. El procedimiento consiste en eliminar las observaciones con datos incompletos y ajustar una ecuación de regresión para predecir los valores que serán utilizados para sustituir los faltantes [6]. Una variante

a este procedimiento es la imputación por «regresión estocástica», en donde los datos faltantes se obtienen con un modelo de regresión más un valor aleatorio asociado al término de error. Este procedimiento garantiza variabilidad en las imputaciones, contribuyendo a reducir el sesgo en la varianza [6].

- Estimación por máxima verosimilitud (MV): El procedimiento para estimar los parámetros de un modelo utilizando una muestra se describe en las siguientes fases.

  1. Estimar los parámetros del modelo con los datos completos con la función de máxima verosimilitud.
  2. Utilizar los parámetros estimados para predecir los valores omitidos.
  3. Sustituir los datos por las predicciones, y obtener nuevos valores de los parámetros maximizando la verosimilitud de la muestra completa.
  4. El algoritmo se aplica hasta lograr la convergencia, la cual se obtiene cuando el valor de los parámetros no cambia entre dos iteraciones sucesivas [6].

- Imputación múltiple (IM): La imputación múltiple utiliza métodos de simulación de *Monte Carlo*, sustituyendo los datos faltantes a partir de algunas simulaciones. En cada simulación se analiza la matriz de datos completos a partir de métodos estadísticos convencionales y posteriormente se combinan los resultados para generar estimadores robustos, su error estándar e intervalos de confianza. Los estimadores finales se calculan como un promedio de los valores sustituidos, lo cual garantiza que se mantenga la variabilidad en los valores imputados [6].
- Imputación estandarizada para valores con datos erróneos: La imputación estandarizada consiste en sustituir los atributos erróneos por valores que se disponen de antemano (por ejemplo: nombres de departamentos o barrios). Usualmente se puede usar la distancia *edit* para comparar las palabras entradas contra un diccionario de términos.
- Imputación utilizando aprendizaje automático: Existen técnicas de *ML* para efectuar la imputación de datos (en el artículo «*Imputation of missing data using machine learning techniques*» de *Kamakshi Lakshminarayan, Steven A. Harp, Robert Goldman and Tariq Samad* [7], se puede acceder a la lectura de dos técnicas). Una de ellas se fundamenta en los

árboles de decisión *C4.5*, el cual efectúa la división de los atributos de acuerdo a la ganancia de información (Sección 5.5, Arboles de Decisión). La segunda técnica se basa en *Autoclass* que consiste en una clasificación bayesiana, la cual determina la asignación de los datos incompletos, con una determinada probabilidad, a clases (clústeres) , situando los puntos a la clase con mayor probabilidad [7].

A continuación se plantean tres casos de estudio relacionados con la imputación

- Si se consideran los métodos de clustering aptos para grandes *datasets*, se plantea el caso de estudio de adaptación de *ML* para situaciones de datos incompletos siguiendo la idea planteada para *Autoclass*. En tal caso se debería estudiar cuál es el algoritmo de clusterización más conveniente (el algoritmo *CURE*, considerado en la *Sección 5.8*, podría ser adecuado)
- Para el procedimiento de imputación estandarizada, utilizar *k-shingles* para las comparaciones, con un k adecuado de tal forma que si todos los *k-shingles* coinciden con una palabra de un diccionario con los términos estándar, ésta es imputada. Por otro lado se podría verificar la viabilidad de usar la técnica propuesta para descartar rápidamente palabras del diccionario. A tales efectos, debería establecerse el valor de k adecuado para que el descarte de términos realizado por los *k-shingles* funcione más rápidamente que la distancia *edit*.
- En entornos de *Big Data*, analizar la forma de establecer que los atributos que faltan pueden estar correlacionados (en la población o en subpoblaciones de los datos). También sería interesante estudiar estadísticamente como serían los fundamentos si presentaran alguna particular distribución de probabilidad de ocurrencia.

Centrar, escalar y normalizar datos

En ocasiones, las variables que definen un modelo pueden escalar de diferente manera, por lo que se hace necesario efectuar una transformación, denominada normalización, la cual permite aumentar la interpretación de las características.

A continuación se presentan algunas definiciones necesarias para definir los conceptos de centrar, escalar y normalizar. Para ello se consideran n observaciones: $x_1, x_2, ..., x_n$, que constituyen una muestra aleatoria de una población más

grande. En la Tabla 2.1 se presenta una distinción entre los valores muestrales (denominados estadísticos) y los que corresponden a la población (llamados parámetros).

Muestra	Población
n = número de elementos de la muestra	N = número de elementos de la población
$\bar{x}$ = media muestral	μ=media poblacional
s^2=varianza muestral	σ^2=varianza poblacional
s = desviación estándar muestral	σ = desviación estándar poblacional

Tabla 2.1: Distinción entre valores muestrales y poblacionales

Tabla de frecuencias

En general, dada una lista grande de datos, ésta es agrupada en clases o categorías donde interesa conocer la cantidad de elementos de cada una de ellas (frecuencia).

Si el valor de la clase es x_i, su frecuencia se representa por f_i. Si n es el número de elementos de la muestra y k la cantidad de clases, se verifica la ecuación 2.1

$$n = \sum_{i=1}^{n} f_i \tag{2.1}$$

Media muestral

La Ec. (2.2) define la media empírica o promedio [8].

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i \tag{2.2}$$

La varianza muestral se define en la Ec. (2.3).

$$s^2 = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})^2 = \frac{1}{n-1} (\sum_{i=1}^{n} x_i^2) - (\sum_{i=1}^{n} x)^2/n)) \tag{2.3}$$

(la segunda definición no contiene la media)

Centrar una variable consiste en sustraer su media a cada uno de sus valores.

Desviación estándar empírica

La desviación estándar empírica se presenta en la Ec. (2.4).

$$S = \sqrt{S^2} \tag{2.4}$$

La desviación estándar muestral es una medida de la dispersión de los datos. La media poblacional se define en la Ec. 2.5

Media poblacional

$$\mu = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{2.5}$$

Varianza poblacional

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2 \tag{2.6}$$

Desviación estándar poblacional

$$\sigma = \sqrt{\sigma^2} \tag{2.7}$$

Escalar una variable consiste en dividir su valor por la desviación estándar. También puede utilizarse otros métodos.

Normalización

La normalización es un proceso que busca dar a cada atributo igual peso, lo cual puede lograrse si los valores se disponen en un intervalo pequeño. Existen varias métodos. Uno de ellos es el *z-score*, que consiste en restar la media y dividir por la desviación estándar (Eq. (2.8)).

$$Z_i = \frac{(xi - \bar{x})}{S} \tag{2.8}$$

2.3.3. Patrones anómalos

Previo a la búsqueda de valores erróneos, se puede comenzar con la detección de valores anómalos, los cuales no son, necesariamente, erróneos. Estos valores, también llamados *outliers*, pueden representar un dato fidedigno de la realidad, aunque pueden estar alejado de las mediciones habituales. Un valor anómalo es aquel que no se corresponde con la conducta normal esperada [9].

Debido a su importancia, son considerados una tarea en el esquema planteado en el *Capítulo 1.*

Desafíos

Una forma de resolver este problema, teóricamente, puede ser definiendo una región del espacio con los datos, que representan puntos poseedores de *conducta normal* y todo lo que cae fuera de esa región es anómalo. Este enfoque presenta una serie de limitantes [9]:

- Es difícil definir la región indicada.
- Las conductas anómalas, cuando son creadas con fines maliciosos, habitualmente se presentan en forma oculta.
- En muchos ambientes las conductas evolucionan, al igual que lo hacen las anómalas.
- En ocasiones los datos contienen ruido, lo cual provoca que se confundan datos anómalos con datos con ruido.

Los tipos de *outliers* pueden ser puntuales (caso en que existe una instancia individual, la cual se considera anormal respecto al resto de los datos), contextuales (la instancia individual es anómala bajo determinado contexto, pero no en otros) o colectivos (en lugar de existir una sola instancia anómala, se trata de una colección de instancias de este tipo) [10].

Detección de anomalías

Una forma de detectarlas es empleando una medida de distancia. Por ejemplo, se pueden utilizar técnicas de clustering: si un punto tiene baja probabilidad de pertenecer a un grupo, puede ser anómalo [9]. Para la detección de anomalías empleando la tarea de clasificación, es necesario disponer de antemano de un conjunto de entrenamiento con datos normales y anómalos. Luego se efectúa el entrenamiento en base a árboles de decisión, *Naive Bayes,* Máquinas de vectores de soporte y redes neuronales, entre otros [9].

El hecho de no detectar un valor anómalo puede resultar en un problema importante si los datos están normalizados, debido a que podría afectar al resto de las observaciones. Una forma de tratarlos, si se tienen valores numéricos, es sustituirlos por el máximo del conjunto o el mínimo, conforme al valor del dato.

Caso de estudio: Outliers en Big Data

Las características de *Big Data* (como ser el gran volumen de datos que procesa) permiten identificar pequeñas sub-poblaciones que en contextos de minería de datos tradicional serían considerados como datos anómalos. Se plantea desarrollar técnicas para que los *outliers* de *Data Mining* tradicional puedan ser pequeñas poblaciones en *Big Data* y, adicionalmente, desarrollar la teoría y técnicas para detectar datos anómalos en contextos de *Big Data*.

2.3.4. Transformación de los atributos

A continuación, se presentan algunas posibles transformaciones para los atributos en los datos

Reducción de la dimensión

La alta dimensionalidad es un problema a la hora de interpretar los datos, así como también lo es, el problema de la maldición de la dimensión (*Sección 5.8.2*). Por lo tanto, en muchos problemas es importante aplicar técnicas de reducción de la dimensión (*Sección 4.2*).

Aumento de la dimensión

En otras circunstancias, interesa aumentar la dimensión o incluso crear nuevos atributos como función de los ya existentes. El aumento de las características permite utilizar el efecto de la maldición de la dimensionalidad, al introducir nuevas dimensiones. De esta forma, los datos se separan en el espacio, posibilitando detectar fronteras lineales donde antes no las había. Se pueden transformar problemas no lineales o irresolubles, en problemas lineales al aumentar el espacio [11].

Creación de características

Se trata de casos en los cuales se agrega un nuevo atributo, como ser, el cuadrado de otro atributo (en este caso, agregando dicho atributo, si el modelo inicial fuese lineal, podría transformarse en cuadrático). También puede utilizarse la combinación de las características u otras opciones. Normalmente, para facilitar la interpretación, se utilizan combinaciones simples. Para atributos numéricos, se emplean operaciones matemáticas (suma, máximo, mínimo, etc.). Para los atributos que son nominales, se pueden utilizar operadores lógicos (conjunción, disyunción, etc.).

2.4. MapReduce

La quinta etapa del ciclo de vida trata del procesamiento de los datos, en los cuales se busca generar productos de utilidad.

La importancia de considerar el procesamiento de datos en analítica, es debido a que para muchos problemas, no se puede abstraer su gestión de la arquitectura del sistema. Luego de la limpieza de los datos, éstos son procesados. La única manera de poder tratar cantidades de datos tan grandes, es a través de alguna técnica del tipo «*divide and conquer*».

En contextos de *Big Data* la técnica más utilizada es *MapReduce*. Variadas técnicas de analítica se basan en ésta para realizar el procesamiento y aplicación de los métodos sobre los datos.

MapReduce es un modelo computacional para procesar, en forma distribuida y rápida, masivas cantidades de datos. El proceso de *MapReduce* fue inicialmente implementado por *Google* y su propósito original buscaba ejecutar la multiplicación de grandes matrices dividiendo a una de ellas en franjas [12]. En el *Anexo 3*, *MapReduce*, se presenta dicha multiplicación.

A continuación se presentan algunas características de *MapReduce* [4].

- Gestiona grandes volúmenes de datos.
- Administra una gran cantidad de procesadores.
- Efectúa tratamientos paralelos y distribuidos (mediante una agenda de entradas y salidas, administrando la tolerancia a fallos y monitoreando los procesos).
- Realiza una computación centralizada y un *storage* de datos distribuidos. La administración de dichos datos en estos procesos, es llevado a cabo por el sistema de archivos distribuidos situado por «debajo» de *MapReduce*.
- La idea sería, si es viable, no mover los datos, sino llevar la programación a los datos para realizar los cálculos.
- Existen dos actores en el proceso: máster y *worker*. El máster es el programa a cargo de administrar el sistema y es llamado, también, nodo maestro. Se denomina *worker* a una máquina procesando alguna tarea dispuesta por el máster.

El sistema *MapReduce* se puede instalar sobre una base de sistema de archivos, denominado sistema de archivos distribuidos [4].

Bajo el entendido de que cada sub problema es independiente, éstos pueden ser atendidos en forma paralela por un «*worker*» (nodo) distinto (hilos, cores, procesadores, computadoras en clústeres).

En la implementación de *MapReduce* se consideran los siguientes aspectos [4]:

- Descomposición de las tareas para ser ejecutados en forma paralela.
- Asignación de las tareas a los distintos *workers*.
- Asegurarse de que el *worker* reciba los datos que necesita.
- Sincronización de los distintos *workers*.
- Compartir los datos producidos por un *worker* si otro los necesita.
- Tolerancia a fallas: monitoreo de la ejecución de los puntos anteriores considerando errores de software o de hardware.

En relación a los aspectos anteriores, *MapReduce* provee una abstracción que oculta al programador muchos detalles a nivel del sistema. Los desarrolladores sólo tienen que preocuparse por lo que debe calcularse sin considerar cómo se hace, o cómo se llevan los datos a los procesos. Aún así, coordinar y organizar grandes cantidades de cálculos requiere de una planificación cuidadosa [4].

2.4.1. Operaciones Map y Reduce

Map

Un nodo maestro distribuye los datos entre los *workers*. En cada *worker*, una función *Map* recibe un *dataset* organizado en tuplas del tipo (clave, valor). Luego, se aplica un cálculo específico sobre todos los registros de entrada de dicho *dataset* y se devuelve una lista de tuplas, expresado en la Ec. (2.9), también del tipo (clave, valor).

$$Map(c1, v1) \longrightarrow lista(c2, v2) \tag{2.9}$$

Estas operaciones son realizadas paralelamente por los *workers*. El *worker* puede volver a subdividir las tareas, en caso de ser necesario. Las listas de tuplas se reorganizan en base a las diferentes claves, formándose un grupo por cada una de ellas. Una vez que el *worker* concluyó sus tareas, le pasa la respuesta al nodo maestro.

Reduce

Por cada grupo formado por las claves, se aplica la función *Reduce* en paralelo, la cual retorna valores, como se indica la Ec. (2.10).

$$Reduce(c2, list(v2)) \longrightarrow list(v3) \tag{2.10}$$

Puede ocurrir que en un proceso de *MapReduce* no haya necesidad de aplicar la función *Reduce*. En el *Anexo 3*, se desarrolla con más detalle su ejecución, junto a aspectos relacionados con la tolerancia a fallos.

Combinadores

A menudo, la función *Map* de *MapReduce* tiene que procesar muchos pares (clave, valor). A los efectos de ahorrar costos en la transferencia, se puede preprocesar los valores en el *Map*. Una forma de hacerlo es considerando una clave y de alguna manera procesar varios valores que se asocian con dicha clave [4].

El proceso de combinar $(k, lista(v_k))$ devuelve, mediante una función de agregación, un valor v_c, por ejemplo la cantidad de elementos con esa misma clave o el promedio, transfiriéndose por el *Map* la pareja: (k, v_c). Normalmente, la función del combinador es similar a la efectuada por el *Reduce*. Se le debe exigir a la función que cumpla con las propiedades de conmutatividad y asociatividad. Por ejemplo, si se quiere obtener el promedio de una serie de números, *MapReduce* no funcionaría (no es conmutativo ni asociativo). Por ejemplo, si se dispone de dos *mappers* y el primero calcula: $\frac{(a_1+a_2+...+a_p)}{p}$ y el segundo $\frac{(a_{p+1}+\cdots+a_q)}{(q-p)}$, el promedio de ellos no es, necesariamente, igual al que se obtiene de calcular $\frac{(a_1+...+a_q)}{q}$.

Una forma de obtener el promedio podría lograrse pre-calculando en el *Map* la suma de los números y devolver en forma de (clave, valor) a la clave y, como valor, la cantidad de elementos y su suma. Luego, el *Reduce* sólo tiene que sumar para todos los mappers, con igual clave, los valores de las sumas y de las cantidades y calcular el promedio. De esta forma es posible eludir el problema de la conmutatividad y asociatividad, aunque para algunas funciones no es posible realizarlo, como por ejemplo la mediana. En este caso, es necesario transferir todos los valores al *reducer* para que realice los cálculos.

2.4.2. Aplicaciones

Debido a su demostrada eficiencia, esta tecnología es utilizada en el diseño de algoritmos. A continuación se enumeran algunas aplicaciones.

Multiplicación de matriz por vector

La implementación de *MapReduce* por parte de *Google* fue empleada para realizar la multiplicación de una matriz por un vector para calcular el *Page Rank* de los sitios *Web* (desarrollado en la *Sección 6.1, Análisis de Links*).

Se considera una matriz M muy grande de dimensión $n \times n$, cuyo elemento genérico es m_{ij}. Sea v un vector de dimensión n. Se busca calcular $x = Mv$ usando *MapReduce*.

Asumiendo que M y v entran en la memoria, la función *Map* produce el par (clave, valor) dado por $(i, m_{ij}v_j)$, para cada elemento m_{ij}. La función *Reduce* toma todos los pares con igual clave i y los suma originándose el par (i, x_i).

Si el vector v no entra en memoria, se puede dividir la matriz M en franjas verticales y el vector v en la misma cantidad de franjas horizontales. A su vez cada franja de la matriz y su correspondiente parte del vector, deben tener una cantidad de elementos, que posibilite efectuar su multiplicación matricial. El tamaño de las franjas debe ser tal que puedan entrar en memoria.

Operaciones del Algebra Relacional

Estas operaciones pueden implementarse con *MapReduce*. Algunas de ellas se resuelven aplicando únicamente la función *Map*. Por ejemplo, para la selección, $\sigma_C(R)$ (selección de las tuplas en un esquema R que cumplen la condición C), la función *Map* recorre cada tupla t en R y verifica que se cumpla la condición C. Si la tupla verifica la indicada condición, *Map* produce el par (t, t) y la función *Reduce* es la identidad [4].

El resto de las operaciones y su implementación en álgebra relacional con *MapReduce* se puede ver en [4], incluyendo la operación de «agrupación y agregación». En este caso, dado una relación R, la función *Map* hace la agrupación produciendo un par (a, b), donde a representa el grupo y b a los distintos valores asociados a la clave a. Luego, la función *Reduce* aplica una función de agregación sobre los valores de la lista formada con los b.

La multiplicación de matrices implementada con *MapReduce* puede verse en el *Anexo 3*, utilizando dos procesos en serie.

MapReduce como flujo de trabajo

Se puede extender el *workflow Map* y *Reduce* a cualquier colección de funciones, lo cual puede representarse como un grafo acíclico y donde el arco $A \rightarrow B$ indica que la salida de A es la entrada de B. En particular, si algún nodo falla, el administrador distribuye la tarea entre otros nodos. Un ejemplo es un *join* de tres relaciones, el cual se puede implementar considerando hacer en el primer *MapReduce* el join de las dos primeras relaciones y con un segundo *MapReduce* se hace el *join* del resultado del primero con la tercera relación [4].

2.4.3. Costo de las comunicaciones

En muchas aplicaciones, el «cuello de botella» se encuentra al mover los datos entre los nodos. El costo de comunicación de una tarea, depende del tamaño de la entrada, y el costo de comunicación de un algoritmo, es la suma de los costos correspondientes de cada una de las tareas que lo componen. De hecho, la eficiencia de este tipo de algoritmos, es medida por el costo de la comunicación, prescindiendo del costo de las tareas en sí (generalmente dichas tareas son simples y su costo es de orden lineal con la entrada). En cambio, existe competencia entre nodos de un clúster al momento de comunicarse y, aunque los datos residan en un disco local, el hecho de cargarlos en memoria, puede insumir un tiempo mayor al necesario, para realizar los cálculos por parte del procesador.

La razón por la cual no es necesario, para el estudio del tiempo de procesamiento, tomar en consideración el tamaño del *output* de las funciones, es debido a que el input en algunas funciones, es formado por los *outputs* de otras. Además, habitualmente es lógico esperar una salida menor en tamaño que la entrada, ya que son resúmenes de éstas [4].

2.5. Conclusiones

En este capítulo se presentó una posible representación para el ciclo de vida de *Big Data*, adaptado en sus fases iniciales a las *3 v's* de los grandes datos. Esta generación de datos en oportunidades se realiza en forma de *stream* lo que requiere que su analítica se lleve a cabo en tiempo real. La implementación de servicios de analítica sobre un *stream* de datos presenta desafíos adicionales a

las técnicas del procesado directo desde bases de datos. Habitualmente, luego de analizados, los datos se descartan.

El ciclo de vida comienza con la colección de los datos, luego se seleccionan los datos útiles en los puntos de generación (*triage)* y se transportan (implica mejorar la eficiencia donde se evalúa costos vs. procesamiento).

La etapa cuatro del ciclo se refiere al tratamiento de los datos. Las técnicas mencionadas para sanitar y normalizar datos habituales en la minería de datos deben ser adaptadas para *Big Data*.

La búsqueda de datos anómalos es considerada dentro de esta etapa y se presenta como una tarea en la Tabla 1.1 Tareas y técnicas para *Big Data*. En grandes datos los *outliers* detectados en *Data Mining* tradicional pueden definir sub-poblaciones no detectadas.

Por último se describió la etapa cinco del ciclo, relativo al procesamiento de los datos abordado a través de la técnica de *MapReduce*, la cual permite realizar cálculos de datos a gran escala, con una arquitectura distribuida y tolerante a fallos. El principal costo al implementar *MapReduce*, está asociado a la comunicación entre las tareas. No obstante, se pueden desarrollar mejoras al diseño del proceso para minimizar dichos costos.

MapReduce permite el procesamiento de grandes cantidades de datos y su forma de trabajo atiende los problemas que ocasionan las *3 v's* características de *Big Data*. La técnica es usada por casi todos los métodos que trabajan con grandes *datasets*. Para ello es necesario adaptar las funciones de *Map* y *Reduce* a cada situación particular. Ejemplos de su utilización los constituyen: vecinos más cercanos, *SVM*, análisis de links, sitios de venta online y sistemas de recomendación, entre otros.

Capítulo 3

Analítica sobre Datos

En este capítulo se explora el concepto de modelo, errores, sesgo, varianza y flexibilidad y se describe la adaptación del análisis predictivo a *Big Data*.

En la *Sección 1.2*, Taxonomía de términos en *Big Data*, se mencionaron las tareas predictivas y descriptivas.

Tareas predictivas

En este tipo de tarea se define un conjunto E de valores de entrada (en base a atributos nominales o numéricos) $E = A_1 \times A_2 ... \times A_n$. Un ejemplo de entrada es una tupla $E = a_1 \times a_2 ... \times a_n$ tal que $a_i \in A_i$. La predicción requiere, en su etapa de construcción, tener etiquetas para la variable de salida, para un conjunto limitado de datos. Una etiqueta representa alguna información sobre una predicción. El objetivo es desarrollar un modelo en el cual se pueda inferir un único aspecto de los datos (se le denomina la variable predicha, de salida, de respuesta, output o endógena), a partir de alguna combinación de otros aspectos de los propios datos (denominados variables de predicción, predictores, explicativas o exógena) [8].

Tareas descriptivas

Las tareas descriptivas consideran un conjunto sin ordenar ni etiquetar. El objetivo es describir los datos (por ejemplo la existencia de correlaciones). Las tareas aludidas utilizan algunas técnicas cuyo funcionamiento se basa en el empleo de modelos. La analítica sobre datos busca la identificación y representación de patrones en ellos. Un modelo, también llamado hipótesis, es la representación de un patrón. Las características, atributos, o *features*, son las

propiedades que muestran las representaciones. En *Big Data* el modelo a emplear debería posibilitar su utilización como modelo de aprendizaje de manera tal que pueda automatizar las tareas de aprendizaje a partir de los datos [1].

Como un caso particular de modelo se presenta el análisis de correlaciones, el cual permite identificar relaciones estadísticas entre valores de los datos (en el *Capítulo 1*, la identificación de correlaciones, se definió como una tarea descriptiva).

3.1. Modelos

Para algunos modelos, interesa determinar el conjunto de características óptimo. A este proceso, se le denomina «selección de características». Esta selección se suele llevar a cabo mediante técnicas estadísticas y requiere, habitualmente, de un conocimiento profundo de la naturaleza del problema.

Selección y Extracción de Características

Algunas de las motivaciones para seleccionar y extraer las características son:

- Obtener información que puede permitir la discriminación.
- Eliminar información redundante e irrelevante.
- Reducir la dimensionalidad del problema.

Los objetos se describen mediante características las cuales pueden ser cualitativas (que pueden ser ordinales como ser el grado de educación o nominales, por ejemplo profesión) o cuantitativas (que pueden ser a su vez continuas como la temperatura del medio ambiente o discretas, como, por ejemplo, la edad de un paciente) [3].

En la práctica se pretende obtener un modelo que permita extraer las características de una relación (que generalmente no es evidente), a partir de grandes cantidades de datos. Dado que a partir de éstos se obtiene una respuesta de predicción, no se puede establecer una relación causa-efecto entre ellos. A lo máximo, se puede establecer una asociación. Por ejemplo, existe una relación entre el peso y la altura de algunos seres vivientes, pero ello no implica que se pueda variar la altura al modificarse el peso [8]. El modelado puede ser un proceso iterativo en el cual, ocasionalmente, se necesita retroceder a etapas anteriores, debiéndose, en algunos casos, modificar el planteo del problema.

Modelado estadístico

El modelado estadístico, del cual se puede ver una clasificación en la Figura 3.1, considera si las densidades condicionales de las observaciones son conocidas o no.

Dados un modelo predictivo, una variable de respuesta Y y un conjunto de predictores $x_1, x_2, ..., x_p$ (también llamados variables explicativas), la modelización estadística, consiste en descomponer los valores que toma Y, para cada individuo i, en base a dos componentes independientes: uno en función de las variables explicativas y un segundo componente que varía según cada individuo.

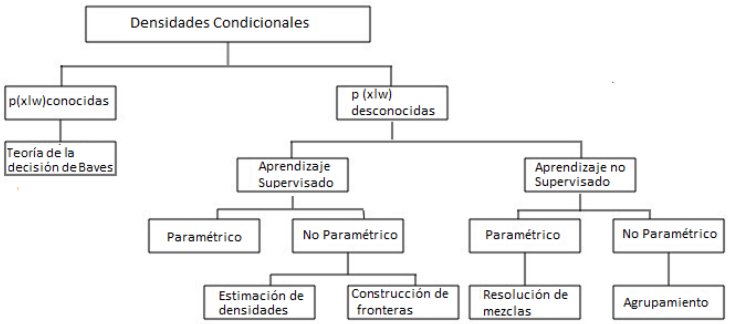

Figura 3.1: Clasificación del modelado estadístico (Origen: *Statistical Pattern Recognition: A Review Anil K. Jain, Robert P.W. Duin, Jianchang Mao - 1999*)

El modelo, por lo tanto, se puede representar por la Ec. (3.1), donde la función f relaciona los valores de la variable de respuesta con los predictores (parte determinista) y ϵ_i la componente específica del individuo llamada error.

$$y_i = f(x_{ij}, ..., x_{ip}) + \epsilon_i \tag{3.1}$$

El término ϵ_i es un valor dependiente del individuo y es caracterizado por una distribución de probabilidad que genera los valores para cada individuo, el cual se supone centrado ($E[\epsilon_i] = 0$) [3]. A su vez, la precisión en la predicción depende de dos valores: el error reducible (el cual se puede mejorar ajustando la función f) y el error irreducible que es el asociado con el valor de Y.

Clasificación del modelado estadístico

La Figura 3.1 muestra una clasificación del modelado a partir de las densidades condicionales [13]. Desde el punto de vista estadístico, las técnicas

presentadas en la *Sección 1.2*, se clasifican en base a las densidades de probabilidad condicional. Si éstas son conocidas, se trata de la Teoría de decisión de *Bayes* (estos métodos se estudian en el *Anexo 4*) y, en caso de tratarse de densidades desconocidas, se definen métodos de agrupamiento y tareas de clasificación, entre otros (que son las descriptas en el *Capítulo 1*). El entrenamiento o aprendizaje de un modelo, puede ser supervisado o no. A la búsqueda de patrones o de asociaciones, sin una variable establecida de salida, se le denomina aprendizaje no supervisado. Se habla de aprendizaje o entrenamiento supervisado, en aquellos casos en que se dispone de un conjunto de entrenamiento con etiquetas de salida conocidas.

A su vez, los modelos pueden ser paramétricos (siendo los parámetros coeficientes o valores que caracterizan al modelo) o no paramétricos. Los métodos paramétricos se deberían adaptar a una función que, en caso de estar alejada de la verdadera función, provoca que los datos no se ajusten bien. En cambio, los modelos no paramétricos no utilizan una función como base del modelo, sino que se manejan con propiedades de distancia de las observaciones a una supuesta función, la cual va mejorando su forma al observarse nuevos datos (por ejemplo, el modelo de clústeres). A continuación, se estudian las posibles combinaciones entre modelos paramétricos y supervisados.

- Aprendizaje supervisado paramétrico. Los modelos predictivos paramétricos asumen que se tiene una determinada función de predicción con coeficientes a determinar y un conjunto de datos de entrenamiento. Con estas observaciones y con métodos adecuados se entrenan los coeficientes, buscando mejorar la predicción mediante métricas de calidad (por ejemplo, la regresión lineal o el método de mínimos cuadrados entre otros). En el aprendizaje supervisado paramétrico se calculan los parámetros que definen al modelo.
- Aprendizaje supervisado no paramétrico. En el aprendizaje supervisado no paramétrico se determina la probabilidad *a posteriori* (representada por la función de densidad de probabilidad) de que un dato pertenezca a una clase, a partir de la información proporcionada por el conjunto de datos con sus respectivas etiquetas.
- Aprendizaje no supervisado paramétrico. No se dispone de un conjunto de entrenamiento, sino que se construye el modelo en base a los datos que se dispone, o datos que se reciben a medida que avanza el tiempo.

Un ejemplo es el modelo de Mezcla de Gausianas.

- Aprendizaje no supervisado no paramétrico. Este tipo de hipótesis no dispone de un conjunto de entrenamiento y el modelo no es caracterizado por parámetros. Incluye a los agrupamientos o técnicas de clustering, las cuales identifican grupos de datos que comparten ciertas características.

En el caso de los clasificadores (aprendizaje supervisado) su performance depende del tamaño del conjunto de entrenamiento, su complejidad y del número de características (el cual depende a su vez del conjunto de entrenamiento). En caso de tener que reducir las dimensiones, se pueden proyectar sobre un espacio con menos dimensiones o haciendo una selección de ellas de acuerdo a algún criterio.

Flexibilidad de los modelos

En el contexto de este trabajo, la flexibilidad es utilizada como sinónimo de complejidad. El modelo lineal para realizar analítica sobre datos más simple podría ser el modelo lineal, aunque puede ocurrir que no sea el más adecuado a los datos disponibles. En este caso se dice que el modelo propuesto no es suficientemente flexible. En el caso de tratarse de modelos paramétricos, mayor flexibilidad implica disponer de más parámetros [14].

En la Figura 3.2 se observan algunas técnicas que, conforme aumenta su flexibilidad, la interpretabilidad disminuye.

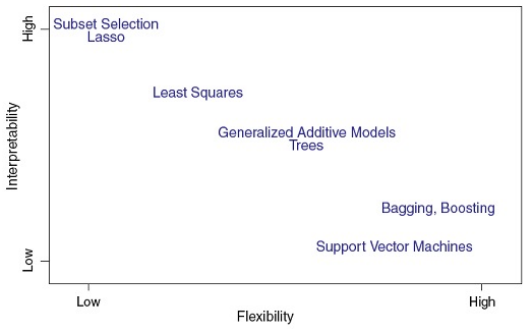

Figura 3.2: Interpretabilidad vs. Flexibilidad de los modelos generados por algunas técnicas. (Origen: *An Introduction to Statistical Learning with Applications in R, James G., Witten D., Hastie T., Tibshirani R. - 2017)*

Cuando la inferencia es el objetivo, es aconsejable la utilización de técnicas, poco flexibles y simples de interpretar. En cambio, cuando la finalidad es la predicción, interesa la precisión pudiéndose optar por hipótesis flexibles

sin interesar la interpretación [14]. Los grados de libertad es la cantidad que indica la flexibilidad del modelo (cuantos más grados de libertad mayor es la flexibilidad) [14].

3.1.1. Errores

En contextos de aprendizaje automático, es importante evaluar distintos modelos y algoritmos para un mismo problema. Los errores, que eventualmente ocurren, dependen de los modelos y tareas utilizadas en el proceso de minería. Sin conocimiento *a priori* sobre la naturaleza del algoritmo de aprendizaje, ninguno de ellos es mejor que los demás, incluyendo modelos aleatorios. La forma de diferenciar la calidad de los algoritmos es estimando los errores que cometen. Para tal fin, en los modelos predictivos se tiene un conjunto de entrenamiento y otro de *testing*. Desde un punto de vista conceptual, existen tres tipos de errores: de entrenamiento, de validación y generalizado. De acuerdo a la técnica que se emplee, existen implementaciones particulares de los errores (por ejemplo, error absoluto y cuadrático, entre otros).

Medida de pérdida

Se usa la notación $L(y, f_{\hat{w}}(x))$ para representar la medida de pérdida o función de pérdida, siendo y el verdadero valor y $f_{\hat{w}}(x)$ el valor $\hat{y}$ predicho por el modelo. La función de pérdida es utilizada para penalizar los errores en la predicción.

Error de entrenamiento

El error de entrenamiento se define como el promedio de la función de pérdida, usando los propios datos de entrenamiento o aprendizaje. Se trata de evaluar la pérdida, en el conjunto de entrenamiento, de la función ajustada, indicando qué tan baja es la evaluación del rendimiento previsto del modelo, Ec. (3.2).

$$\textit{Error de entrenamiento} = \frac{1}{N}\sum_{i=1}^{N} L(y_i, f_{\hat{w}}(x_i)) \tag{3.2}$$

Como ejemplo de una función de pérdida, aunque es generalizable a otros modelos supervisados, en la *Sección 4.4* se presenta la tarea de regresión, la cual utiliza la Ec. (3.3).

$$L(f(x), h(x)) = (f(x) - h(x))^2 \tag{3.3}$$

La función $f(x)$ es el valor verdadero, y $h(x)$ el estimado por la regresión para un determinado modelo.

La Ec. (3.4) se utiliza como función de pérdida para la clasificación.

$$L(f(x), h(x)) = \begin{cases} 0 & \text{si } f(x) = h(x) \\ 1 & \text{sino} \end{cases} \tag{3.4}$$

El error medio cuadrático se define a partir del error cuadrático (Ec. (3.5)) y es designado por *MSE* (*Mean Squared Error*), Ec. (3.6). El error cuadrático tiene la ventaja de que refleja los casos en que si el error es pequeño, la pérdida será pequeña, y si es grande, la pérdida será muy grande [3].

$$\textit{Error cuadrático} : L(y_i, f_{\hat{w}}(x)) = (y - f_{\hat{w}}(x))^2 \tag{3.5}$$

$$MSE = \frac{1}{N} \sum_{i=1}^{N} (y_i - f_{\hat{w}}(x_i)^2 \tag{3.6}$$

Cuanto más próximos estén $\hat{y}_i$ y $f_{\hat{w}}(x_i)$, el error será menor y viceversa.

El *MSE* aplicado al conjunto de entrenamiento se denomina *MSE* de entrenamiento y aplicado a un conjunto de *test* se denomina *MSE* de *test*. El modelo que se busca elegir es el que tiene menor *MSE* de *test* [3].

Otros tipos de error comúnmente utilizados son:

El error medio absoluto se obtiene a partir del error absoluto (Eq. 3.7) y se designa por *MAE (Mean Absolute Error)* (Ec. (3.8)).

$$\textit{Error absoluto} : L(y, f_{\hat{w}}(x)) = |y - f_{\hat{w}}(x)| \tag{3.7}$$

$$MAE = \frac{1}{n} \sum_{i=1}^{n} |y - f_{\hat{w}}(x_i)| \tag{3.8}$$

La raíz del error medio cuadrático se designa por *RMSE* (*Root Mean Squared Error*, raíz cuadrada del promedio de los errores cuadráticos), Ec. (3.9).

$$RMSE = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (y_i - f_{\hat{w}}(x_i))^2} \tag{3.9}$$

Al evaluar el modelo, no es conveniente calcular el error en base a los datos del conjunto de entrenamiento. Esto es debido a que los parámetros $\hat{w}$ fueron calculados utilizando dichos datos y, en caso de ser utilizados en la etapa de *testing*, se puede obtener un modelo demasiado optimista. Si éste fuese el caso, se obtendría un error pequeño sobre los datos de entrenamiento, pero no tendría igual performance sobre nuevos datos, salvo que los datos de entrenamiento sean representativos de todo el conjunto (para grandes datasets o, si éstos son dinámicos, es bastante improbable que la muestra sea representativa) [3]. Para estos casos es necesario definir el error generalizado o error verdadero y el error de *test.*

Sea N_{Test} el número de observaciones de un conjunto disjunto al de entrenamiento denominado conjunto de *test* o de validación. Empleando el conjunto de entrenamiento se construye un modelo representado por $f_{\hat{w}}$ (función ajustada usando los datos de entrenamiento).

Error de testing

Se define una función L de pérdida y, haciendo el promedio sobre el conjunto de validación, se obtiene el error de *test,* Ec. (3.10).

$$Error\ de\ test = \frac{1}{N_{test}} \sum_{i \in test} L(y_i, f_{\hat{w}}(x_i)) \tag{3.10}$$

Error generalizado o error verdadero

El error generalizado o verdadero mide la efectividad con la que predice el modelo para todas las observaciones. Hay situaciones en las que no se puede calcular el error verdadero, por lo que se utiliza el error en el *testing*. El objetivo sería obtener el mínimo valor de la función *MSE*, aunque esto generalmente no es posible, ya que el *MSE*, al igual que el error generalizado, no puede ser calculado, porque están definidos en términos de la función real. Para llevar a cabo este cálculo, se debería hallar el promedio de todos los posibles *datasets*, lo cual, habitualmente, no es posible, dado que «todos los posibles *datasets*» refiere a todo el conjunto de datos del universo considerado. Por este motivo, es que se utilizan mecanismos de optimización.

Ejemplo 3.1

En este ejemplo intervienen los tres errores en una clasificación. Suponer que se tiene un *dataset* de enteros en $[-20, +20]$ el cual contiene 4 puntos: $-3, -2, +3, +4$ y se busca predecir el signo de un nuevo entero.

Se tiene una regla del tipo:

$$Si\ (x > valor)\ predecir + 1;\ sino\ predecir\ -1$$

Como regla de entrenamiento se considera:

$$Si\ (x > 1)\ predecir\ +1;\ sino\ predecir\ -1$$

Esta regla de entrenamiento para los valores -3 y -2 predice -1 y para $+3$ y $+4$ predice $+1$, por lo que el error de entrenamiento es $0\,\%$.

Suponer que ahora se dispone de nuevos puntos a clasificar, provenientes del conjunto de validación: $0, 1, 2, 5$.

Para los valores 0 y 1 la regla predice -1 , lo cual es equivocado. Para 2 y 5 predice $+1$. Se dice que el error de *test* o de validación es $2/4$ $(50\,\%)$.

Para determinar el error verdadero o generalizado, se debería considerar a todos los casos, que serían 41 enteros (desde -20 a $+20$). La regla aplicada a ellos clasificaría mal al 0 y al 1, es decir el error generalizado sería de $2/41$ $(0,05\,\%)$.

3.2. Sesgo y Varianza en un modelo

El error del modelo puede variar en función de la complejidad, por lo tanto, luego de obtenido un modelo, éste es evaluado para que no incurra en errores, especialmente al aumentar su complejidad. Al hacer predicciones existen tres fuentes de error: el ruido, el sesgo y la varianza. La varianza y el sesgo son dos características a considerar al seleccionar un modelo estadístico. Estos elementos se analizan en el contexto de lo que se llama «balance entre la varianza y el sesgo». Este balance indica que modelos simples, no son suficientes para describir una relación entre los datos, pero, por otro lado, los modelos muy complejos, si bien son más flexibles para describir relaciones complicadas, presentan la desventaja de generar comportamientos extraños. De aquí la importancia de estudiar este balance [10].

La varianza de los modelos refiere a la cantidad en que las estimaciones de los parámetros del modelo cambiarían si se utilizaran distintos conjuntos de datos de entrenamiento. Es deseable que la varianza se mantenga estable al cambiar los *datasets*. Si la varianza es alta, pequeños cambios en el conjunto de entrenamiento, cambian considerablemente la estimación de los parámetros. La varianza mide la forma en que el ruido se distribuye en cada característica y cuánto afecta a la predicción.

El sesgo es el error introducido al utilizar un modelo que intenta aproximarse a una situación real, es decir, indica qué tan alejado se encuentra el modelo de los verdaderos valores. En otras palabras, el sesgo es una evaluación que indica lo bien que el modelo se ajusta, a la relación real entre las variables predictoras y la predicción.

En el *Anexo 5, Errores y Evaluación de los Modelos*, se define con mayor detalle esta relación, considerando el balance entre el sesgo y la varianza. Asimismo se presenta la derivación formal de las tres fuentes de error (el ruido, el sesgo y la varianza).

Cantidad de datos

Cuando se dispone de pocos datos en el conjunto de entrenamiento, el modelo puede quedar «pobre» y, si las observaciones son escasas en el conjunto de validación, se puede originar una estimación incorrecta del error generalizado. Si no son suficientes los puntos para los dos conjuntos se puede conducir una validación cruzada (desarrollado en el *Anexo 5*, junto a otras técnicas de validación).

Los investigadores *Michelle Banko y Eric Brill* han efectuado estudios donde compararon la precisión de distintos algoritmos, concluyendo que no importa la calidad del algoritmo, si no el tamaño del conjunto de entrenamiento para obtener mejores resultados [15]. Dicho de otra forma, es preferible canalizar el esfuerzo de investigación en el tratamiento de grandes conjuntos de datos, en lugar de mejorar algoritmos entrenados con pequeños conjuntos de entrenamiento. El hecho de que los datos en *Big Data* sean masivos, posibilitaría el aprendizaje de los modelos, aunque, como se verá en la *Sección 3.4, Análisis Predictivo en Big Data*, se debe atender otras situaciones que surgen por el gran volumen y dimensionalidad.

En conclusión, una forma confiable de obtener una alta performance en *ML* es considerar un algoritmo con bajo sesgo y entrenarlo con un conjunto de

datos de un tamaño grande.

En el *Anexo 5* se presenta una detallada descripción del comportamiento del error verdadero y de entrenamiento en función de la cantidad de datos del conjunto de entrenamiento y de la complejidad del modelo.

3.3. Correlaciones

Una correlación cuantifica la relación estadística entre dos valores de datos. En una correlación fuerte, cuando uno de los valores de los datos es modificado, es altamente probable que se modifique también el otro [16]. Previo a la definición de la correlación se brindarán otras definiciones necesarias.

Variables aleatorias

Una variable aleatoria es un valor probabilístico, resultado de un experimento. Sea Y una variable aleatoria donde todos sus posibles valores son $y_1, y_2, ..., y_n$.

La distribución de probabilidad de una variable Y es la probabilidad $P(Y = y_i)$ para una observación y_i. Estas variables pueden ser categóricas o numéricas y, a su vez, pueden ser discretas o continuas.

Para el caso de variables numéricas discretas se tienen las siguientes definiciones:

Esperanza

$$E[Y] = \sum_{i=1}^{n} y_i P(Y = y_i) \tag{3.11}$$

Varianza

$$Var(Y) = E[Y - E[Y]^2] \tag{3.12}$$

Desviación estándar

$$\sigma = \sqrt{Var(Y)} \tag{3.13}$$

Covarianza empírica

Se consideran pares de puntos (X_i, Y_i). La covarianza empírica se define en la Ec. (3.14). $\bar{X}$,e $\bar{Y}$, indican los valores promedios de las variables X e Y.

$$Cov(X,Y) = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y}) = \frac{1}{n-1}(\sum_{i=1}^{n} x_i y_i - n\bar{x}\bar{y}) \tag{3.14}$$

Coeficiente de Correlación

La correlación se define a partir de la covarianza. Básicamente, proporciona una medida de la relación lineal entre dos variables [8]. Es utilizada, por ejemplo, en los sistemas de recomendación para buscar afinidad en los gustos de dos usuarios (desarrollado en sistemas de recomendación en el *Capítulo 6*).

$$p(X,Y) = \frac{Cov(X,Y)}{\sigma_x \sigma_y} \tag{3.15}$$

La correlación $p(X,Y)$ mide la «fuerza» de la relación lineal entre X e Y [8].

Si $p(X,Y) = 0$, no existe correlación lineal.

Se puede demostrar que $-1 \leq p(X,Y) \leq +1$ [8].

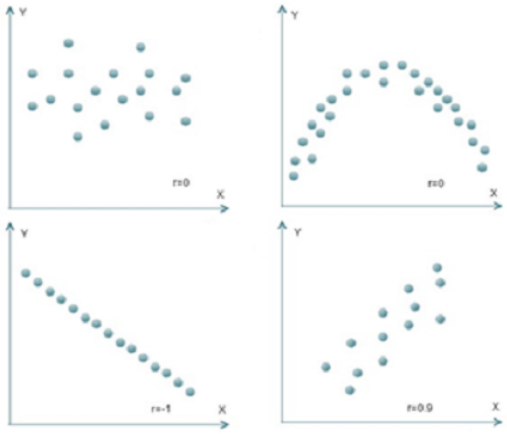

Figura 3.3: Gráficas de dispersión comunes para algunos valores de r (Origen: *Probabilidad y estadística Aplicaciones y métodos George C. Cannavos - 1998*)

La correlación, se puede considerar como un método descriptivo, utilizado para determinar el grado de similitud entre dos variables numéricas a través de un número (r).

- Si $r = 0$, no existe correlación.
- Si $r < 0$, están inversamente correlacionadas, es decir si aumenta una variable la otra disminuye.
- Si $r > 0$, al crecer una, la otra también lo hace.

La Figura 3.3 muestra algunos casos de acuerdo al valor de la correlación.

Desde un punto de vista práctico, se considera que, cuando dichas variables escalan de diferente manera, es necesario normalizarlas.

En contextos de *Big Data* puede surgir correlaciones espurias, cuyo desarrollo forma parte de la próxima sección

3.4. Análisis predictivo en *Big Data*

Las técnicas de análisis predictivo se basan principalmente en métodos estadísticos y *Big Data* requiere el desarrollo de nuevos métodos debido a varios factores. En primer lugar, los métodos estadísticos convencionales están relacionados con la significación estadística: se obtiene una pequeña muestra de la población y el resultado se compara con la posibilidad de examinar la importancia de una determinada relación. La conclusión se generaliza a toda la población. Por el contrario, las muestras de *Big Data* son masivas y representan a la mayoría de la población. Como resultado, la noción de significación estadística no es tan relevante para *Big Data*. En segundo lugar, en términos de eficiencia computacional, muchos métodos convencionales para muestras pequeñas no se amplían a grandes cantidades de datos [17]. El tercer factor corresponde a las características distintivas inherentes a los macro datos: la heterogeneidad, la acumulación de ruido, las correlaciones falsas y la endogeneidad incidental [18], los cuales se describen a continuación.

1. Heterogeneidad: *Big Data* a menudo es originada por diferentes fuentes y representan información de diferentes sub-poblaciones, siendo altamente heterogéneas. Los datos de una sub-población en muestras pequeñas pueden considerarse *outliers* debido a sus pocas ocurrencias. Sin embargo, el gran tamaño en *Big Data* permite modelar la heterogeneidad que surge de los datos sub-poblacionales. Los grandes volúmenes de datos permite a *Big Data* la identificación de pequeñas sub-poblaciones y descubrir patrones ocultos no perceptibles con pocos datos.

Como ejemplo, se considera un modelo de mezclas convexo para una determinada población dado por la Ec. (3.16).

$$\lambda_i p_i(y; \theta_i(x) + ... + \lambda_m p_m(y; \theta_m(x))) \tag{3.16}$$

Los coeficientes $\lambda_j \geq 0$ representan la proporción de la población j y $p_j(y; \theta_j(x))$ es la distribución de probabilidad de la respuesta de la población j.

El vector de los parámetros es representado por $\theta_j(x)$.

Las sub poblaciones pequeñas tienen un λ_j pequeño. Pero si n, el número de datos, es grande, $n\lambda_j$ será grande pudiéndose determinar los parámetros $\theta_j(x)$ que caracterizan a la población j [18].

2. Acumulación de ruido: La estimación de modelos predictivos en *Big Data* implica la estimación simultánea de varios parámetros. El error de estimación acumulado (o ruido) para diferentes parámetros podría dominar las magnitudes de las variables que tienen efectos reales dentro del modelo. En otras palabras, algunas variables con poder explicativo significativo pueden pasarse por alto como resultado de la acumulación de ruido.

   Big Data requiere estimar simultáneamente muchos parámetros: los errores de estimación aumentan cuando la regla de predicción contiene una gran cantidad de tales parámetros. El efecto de acumulación de ruido es importante en altas dimensiones. Una mala performance se puede deber a la existencia de muchas características débiles que no contribuyen a la reducción del error del modelo empleado Por tal motivo es aconsejable la utilización de modelos *sparse* [18]. A tales efectos la regresión *LASSO* (*Sección 4.4*) puede ser considerada para la eliminación de características.

3. Correlación espuria: Para *Big Data*, la correlación espuria se refiere a variables que se encuentran falsamente correlacionadas (en realidad, no lo están) debido al tamaño masivo del conjunto de datos. Se ha demostrado que el coeficiente de correlación entre variables aleatorias independientes aumenta con el tamaño del conjunto de datos [18]. Experimentalmente, algunas variables no relacionadas (a las cuales se les exige que sean independientes), resultan erróneamente correlacionadas (debido a la alta dimensionalidad). Estas situaciones pueden conducir a una selección

incorrecta de las variables y a inferencias estadísticas erróneas [18].
Para evitar este problema se han desarrollado métodos que proponen una selección segura basado en el aprendizaje de correlaciones, la cual filtra las características que tienen correlaciones débiles [19].

4. Endogeneidad incidental. En el análisis de regresión se considera que las variables explicativas son independientes del término residual. La validez de la mayoría de los métodos estadísticos utilizados en el análisis de regresión depende de esta suposición. En otras palabras, la existencia de endogeneidad incidental (es decir, la dependencia del término residual en algunos de los predictores) afecta la validez de los métodos estadísticos utilizados para el análisis de regresión. Aunque la suposición de la exogeneidad generalmente se cumple en muestras pequeñas, la endogeneidad incidental está muy presente en *Big Data* [17].
Considerando un modelo de regresión lineal, el término endogeneidad significa que los predictores están correlacionados con el término correspondiente al ruido. Normalmente se asume que no es así pero en altas dimensiones esto puede ocurrir [17]. No obstante, estas correlaciones son auténticas y pueden ser utilizadas en el análisis de correlaciones.
A diferencia de la correlación espuria, la endogeneidad incidental se refiere a una relación verdadera entre las variables y el término de error.

3.5. Tendencias actuales en Analítica sobre Big Data

En el proceso de *Big Data* la analítica puede ubicarse como la fase siguiente a la gestión de los datos, atendiendo el análisis y modelado y la interpretación de los resultados. Es decir la obtención de inteligencia de *Big Data* [17].

A continuación se mencionan algunas de las analíticas más importantes, las cuales en el contexto de este trabajo se pueden considerar como aplicaciones, de acuerdo a la taxonomía de términos introducida en la *Sección 1.2*.

Text analytics o text mining

Su objetivo es convertir masivas cantidades de textos producidas por humanos en sumarizaciones útiles para la toma de decisiones. Algunas técnicas utilizadas son: extracción de información (obtienen datos estructurados a partir

de inestructurados o estructurados), técnicas de sumarización de textos (efectuar un resumen de varios documentos los cuales pueden ser de extracción, donde no interesa el significado y de abstracción donde interesa la información semántica), técnicas de contestado de preguntas (QA, son programas que contestan preguntas efectuadas en lenguaje natural, por ejemplo el programa *SIRI* de *Apple* o *Watson* de *IBM*); análisis de sentimientos (desarrollado en este trabajo en la *Sección 6.3*) [17].

Analítica sobre audio

La analítica sobre audio comprende los datos que se obtienen de las grabaciones de un *call center* o en el área de la salud (utilizado para efectuar diagnósticos). Actualmente se utilizan sistemas basados en transcripciones y basado en fonética [17].

Analítica sobre video

Se denomina análisis de contenidos de video (*VCA*). El principal desafío consiste en el tamaño de los datos en forma de video. Una aplicación comercial es el reconocimiento de los hábitos de compra de clientes a través de circuitos cerrados de TV [17].

Analítica sobre medios sociales

La analítica sobre medios sociales se basa en datos del tipo estructurado o no, provenientes de redes sociales, *blogs*, *wikis*, o cualquier sitio donde el usuario intercambia información. Se distinguen las analíticas basadas en contenidos (los datos se pueden originar en forma de texto, audio, o video) y en la estructura (análisis de redes sociales). Algunos procesos que permiten obtener información sobre ellas son la detección de comunidades (presentado en la *Sección 6.4, Redes Sociales*), análisis de la influencia social (estudio de la influencia de algunos actores en la red, lo cual es útil a los efectos del marketing). predicción de links (predice futuras conexiones en una red con aplicaciones en biología, seguridad y sistemas de recomendación entre otros) [17].

3.6. Conclusiones

Las características a considerar de un modelo estadístico son la varianza, el sesgo y el balance, que debe existir entre estas tres medidas. Este balance

indica que modelos simples, no son suficientes para describir una relación entre los datos. Por otra parte, los modelos muy complejos presentan la desventaja de generar comportamientos impredecibles.

Al utilizar algoritmos de aprendizaje con múltiples parámetros, se asegura un sesgo bajo; aunque, al utilizar grandes conjuntos de datos de entrenamiento, también se garantiza una varianza pequeña y, en consecuencia, un algoritmo con buena performance en la predicción.

Si bien, en este trabajo, se presenta el uso de las correlaciones especialmente a nivel de los sistemas de recomendación y en la regresión *LASSO*, la búsqueda de estas propiedades en los datos, resulta ser un factor importante en la detección de relaciones no evidentes en las observaciones a estudiar.

Caso de estudio

Viabilidad de definir un modelo de aprendizaje automático si existe una alta variabilidad en los datos.

En situaciones donde los datos se generan masivamente, se plantea el problema de adaptar los conceptos de error, evaluación, conjuntos de entrenamiento y *testing* a modelos de aprendizaje automático. Una posibilidad sería enfocar el problema como si se tratase de un *stream* de datos y diseñar técnicas específicas. Por ejemplo, al analizar la construcción de los conjuntos de entrenamiento y *testing*, si se considera que el flujo de datos es dinámico y generado en forma masiva, una posible solución puede ser empleando ventanas deslizantes, dejando fijos los datos en un intervalo de tiempo. Dado que la variabilidad genera sesgos originados en las diferentes fuentes y también a la aparición de nuevos atributos, el desafío que surge es determinar cómo ésta afecta al modelo.

Capítulo 4

Técnicas generales de Data Mining

En este capítulo se estudian técnicas aplicadas en *Data Mining* que pueden ser utilizadas en contextos de *Big Data*. Las técnicas de *Data Mining* se relacionan con la analítica sobre grandes datos, en el sentido que aquellas pueden escalarse para ser aplicados sobre estos últimos. Asimismo, se describen las siguientes técnicas generales: aprendizaje automático, algoritmos de reducción de la dimensión, de optimización y la regresión.

4.1. Aprendizaje automático

El aprendizaje automático o aprendizaje de máquinas (*Machine-learning, ML*) trata sobre la exploración y desarrollo de modelos matemáticos y algoritmos para aprender de los datos. Su paradigma se centra en los objetivos de clasificación, a través de un modelo que mapea los datos con el conjunto de conocimientos por medio del desarrollo de los algoritmos denominados de aprendizaje [1]. En otras palabras los *ML* son algoritmos de *Data Mining*, utilizados cuando se busca un patrón determinado en los datos. El *ML* puede ser empleado para hacer predicciones, asociaciones, descubrir patrones en los datos y clasificaciones, entre otras tareas.

Estos algoritmos realizan clasificaciones o sumarizaciones de los datos mediante modelos entrenados. Un *ML* se compone de un set de datos de entrenamiento, formado por pares (x, y) donde x es un vector de características, las cuales pueden pertenecer a una categoría o ser numéricas. La y representa una

etiqueta para clasificar a x.

Una aplicación usual de *ML* es considerar x muy grande (por ejemplo, un conjunto de palabras) y, dado un documento que se quiere clasificar en una categoría, retornar si la palabra se encuentra o no en el documento. En este caso la etiqueta y toma un valor booleano.

El objetivo del aprendizaje automático es obtener la función $y = f(x)$ que mejor prediga el valor de y para cada valor de x.

- Si y es real, *ML* es una regresión.
- Si y es un booleano, *ML* se llama clasificación binaria.
- Si y es un miembro de un conjunto finito, a los componentes de ese conjunto se les denomina clases y se tiene clasificación multiclase.

El aprendizaje puede ser por *batch* u *online* (presentado en 2.3, *Stream* de datos). Cuando es por *batch*, se dispone de todos los datos del conjunto de entrenamiento desde un principio, en cambio, cuando es *online*, los datos de entrenamiento se obtienen a medida que avanza el tiempo y, por lo tanto, el modelo puede modificarse.

La ventaja del aprendizaje *online* es que puede ajustarse a grandes conjuntos de datos y, si cambia la población, el modelo se adapta. Otra ventaja, es la posibilidad de implementar el aprendizaje activo, el cual es particularmente útil cuando no se dispone de etiquetas y el clasificador no está seguro de su decisión. En este caso, se puede buscar la clasificación correcta consultando a un experto.

El aprendizaje automático abarca varios métodos estadísticos, regresión lineal y logística, árboles de decisión, perceptrón, redes neuronales, aprendizaje basado en instancias (como ser vecinos más cercanos), máquinas de vectores de soporte (*SVM*), redes bayesianas, métodos de gradiente, modelos ocultos de *Markov* y *deep learning*, entre otros. Estos métodos obtienen buenos resultados al tratar grandes conjuntos de datos, debido a que permiten el procesamiento en paralelo.

Esquema del flujo en un modelo para Big Data

En la Figura 4.1, *Modelo del flujo para un modelo de ML tradicional y para Big Data*, presentamos un esquema de *ML* tradicional (a partir de un conjunto de datos, se extraen características de ellos y se plantea un modelo, el cual es entrenado con algunos de los datos disponibles y los restantes datos lo validan empleando alguna métrica de calidad la cual, habitualmente, evalúa el error). En la misma imagen, también proponemos otro esquema adaptado a *Big Data* justificándolo por las siguientes consideraciones:

- Pueden surgir nuevas características con una gran variabilidad, lo cual es característico del paradigma
- De acuerdo a lo planteado en la *Sección 3.4. Análisis predictivo en Big Data*, la masividad de los datos permitiría identificar pequeñas sub-poblaciones que en otros contextos serían representantes de un sub espacio de *outliers*. Esta posibilidad implicaría que se necesite más de un modelo de *ML* para identificar el comportamiento de dichas sub-poblaciones. En consecuencia, podría ser necesario evaluar a los modelos en forma independiente con diferentes métricas.

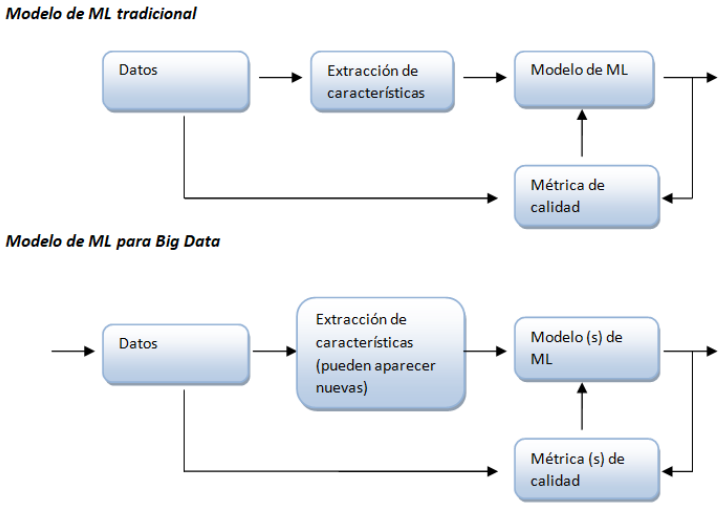

Figura 4.1: Modelo del flujo para un modelo de ML tradicional y para Big Data.

En el *Anexo 6, Métodos empleados en aprendizaje automático*, se presentan algunas técnicas de *ML*.

4.2. Algoritmos de reducción de la dimensión

La reducción de la dimensión se realiza a través de técnicas cuyo objetivo es comprimir las dimensiones de la representación de los datos, reduciendo las columnas (características) pero manteniendo la riqueza de los datos (en este caso son las observaciones). Para poder aplicar las técnicas de reducción de la dimensionalidad, se debe cumplir que los datos representados en un espacio de dimensión D, se puedan representar en un sub espacio de éste, de dimensión d, tal que $d < D$ y que las d dimensiones representen a los datos. Para determinar la dimensión real del conjunto de datos, se calcula el rango de la matriz, que es el número de columnas independientes de ella.

El objetivo del método de reducción es descubrir los ejes, alrededor de los cuales, se distribuyen los datos. El costo de esta modificación es el error en el cual se incurre, ya que la nueva representación no es exacta.

Algunas Aplicaciones de la reducción de la dimensión

- Quitar datos redundantes o con ruidos.
- Visualización (lo que implica reducir la información a 2 o 3 dimensiones).
- Facilita el almacenamiento y procesamiento de los datos.
- Identificación de tópicos en documentos o en matrices de utilidad, en los sistemas de recomendación *(Anexo 9, Técnicas empleadas en las aplicaciones*).

De acuerdo al problema, la representación de los datos genera matrices de dimensión muy alta, que hacen dificultoso aplicar métodos de cálculo numérico.

En el *Anexo* 6, se presentan las técnicas: descomposición *UV*, Análisis de componentes principales *PCA, SVD* y *CUR*. Estas técnicas permiten transformar los datos, utilizando la eliminación de filas o columnas no relevantes, o mediante la introducción de nuevas matrices equivalentes, cuyos cálculos pueden efectuarse computacionalmente.

4.3. Algoritmos de optimización

Los algoritmos de optimización son utilizados por diferentes técnicas y su aplicación implica hallar el mínimo de una función. [20].

Gradiente de una función

Sea $y = g(w)$, derivable, siendo w un vector de R^n. El gradiente es el vector formado con las derivadas parciales de $y = g(w)$, respecto a las componentes de w, (4.1).

$$\begin{pmatrix} \frac{\partial g}{\partial w_0} \\ \frac{\partial g}{\partial w_1} \\ \vdots \\ \frac{\partial g}{\partial w_p} \end{pmatrix} \tag{4.1}$$

Gradiente descendente

El método del gradiente descendente se basa en buscar la dirección en la que una pequeña variación del vector de pesos (vector de los coeficientes del modelo) hace que la función decrezca más rápidamente. En contextos de aprendizaje automático, la función que se pretende minimizar es, generalmente, el error $l(w)$ (diferencia entre el valor predicho y el real).

Se define un parámetro η que representa al «tamaño del paso». Para la elección de η se debe tener en cuenta que si éste es muy chico, la convergencia demorará y, si es muy grande, puede demorar también o llegar a no converger. Para evitar estos problemas se puede variar el valor de η con el tiempo (entre las iteraciones). Esta elección tiene sentido si se asume que al principio se está lejos del valor óptimo, por lo que en ese momento se justifica el uso de pasos grandes. A lo largo de las iteraciones, al aproximarse al valor óptimo, se afina la búsqueda utilizándose pasos pequeños. Para el caso de una dimensión la actualización está dada por la Ec. (4.2).

$$w^{(t+1)} \longleftarrow w^{(t)} + \eta \frac{dl}{dw} \tag{4.2}$$

Para el criterio de convergencia se puede considerar que $|\frac{dl(w)}{dw}| < \varepsilon$, siendo ε un valor de umbral a ser determinado. Cuando el algoritmo del gradiente se acerca al óptimo, es conveniente que se reduzca el tamaño de la condición de convergencia. Una opción para considerar que el algoritmo converge, es medir dicha condición, en todo un ciclo, a lo largo de todas las características (por ejemplo, que sean menores a una tolerancia ε). El algoritmo actualiza la derivada hasta alcanzar w^* que es el objetivo. En la Ec. (4.3) se presenta la actualización en el paso para el gradiente.

$$w^{(t+1)} \longleftarrow w^{(t)} + \eta \nabla l(w^{(t)}) \tag{4.3}$$

Algoritmo en Coordenadas descendentes .

En lugar de resolver un problema de optimización en todas las dimensiones, el algoritmo en coordenadas descendentes lo hace coordenada por coordenada (una por cada paso, siguiendo los ejes). Es decir se actualiza variable a variable, una a la vez.

Aprendizaje automático en grandes datasets

En esta sección se analizan los aprendizajes *online* y el modo en que deben ser entrenados los coeficientes durante el arribo de los datos.

Escalar el algoritmo de gradiente en grandes *datasets*, presenta inconvenientes, dado que requiere un escaneo completo para la actualización de w (lo que implica que todos los puntos tengan que hacer su aporte, en cada pasada). Una solución podría ser que, en lugar de utilizar el total de datos para efectuar las actualizaciones, se lleve a cabo con pequeñas cantidades lográndose, en cada pasada, múltiples actualizaciones. Estas consideraciones conducen a dos variantes del algoritmo del gradiente: gradiente estocástico y gradiente estocástico por lotes.

Algoritmo de Gradiente Estocástico

El gradiente estocástico es una modificación de gradiente descendente, el cual posibilita mayor velocidad en tiempo de ejecución. Mejora el escalado, aunque presenta una menor estabilidad, tornándose complejo para llevarlo a la práctica. El enfoque del gradiente estocástico (GE) considera entrenar a un solo punto dato diferente en cada iteración o a un pequeño conjunto de ellos (entrenamiento por mini lotes), no al conjunto de observaciones en su totalidad [4].

En el *Anexo 6*, se presentan los algoritmos de Gradiente descendente, Gradiente estocástico, Coordenadas descendentes y una aplicación de éste a mínimos cuadrados. En el *Anexo 8*, se estudia el algoritmo del gradiente aplicado a la regresión logística.

4.4. Regresión

Se denomina regresor al modelo de *Data Mining* cuya predicción es un valor determinado. El término regresión se debe a que en estudios estadísticos de la evolución de ciertos grupos biológicos, las características promedio de las nuevas generaciones tendían a moverse hacia las características promedio de la población general, más que a las de las generaciones previas de ese grupo [8].

Las hipótesis de regresión son modelos paramétricos cuya tarea principal es la regresión. Su finalidad es predecir un valor de tipo numérico para una entrada determinada y, para su construcción, se parte de un conjunto de entrenamiento. En esta sección se describen los modelos de regresión lineal, los cuales pueden ser combinaciones lineales de cualquier tipo de función, de acuerdo a los valores de los parámetros de entrada que, en general, son atributos de la realidad.

Dentro de la regresión lineal se encuentra el modelo de regresión simple, la cual considera un único *input* y una sola salida. El modelo de representación es una recta.

La regresión múltiple permite relaciones más complicadas entre las entradas (mayor cantidad de *inputs*) y las salidas (curvas con mayor complejidad que una línea recta).

En la regresión lineal, ya sea simple o múltiple, las variables explicativas participan en forma aditiva y constante para todo el dominio. La suposición de linealidad parece restrictiva y, por consiguiente, puede arrojar malos ajustes. Sin embargo, en algunos casos, es aconsejable su uso. En ocasiones, se puede ampliar la diversidad y obtener modelos lineales más complejos considerando a los predictores como transformaciones de los originales (logaritmos y polinomios, entre otros). También se pueden tomar como predictores a variables binarias, mediante la recodificación en intervalos, lográndose funciones escalera [3].

Modelos de regresión

Los modelos de regresión se clasifican en:

- Modelo de regresión simple el cual tiene una sola variable predictora.
- Modelo de regresión múltiple (o modelo lineal general) con varias variables predictoras.
- Regresión no lineal, donde las entradas no están relacionadas linealmente entre sí, ni por transformaciones.

Métrica para el error: Suma de los residuos cuadráticos

RSS (*Residual Sum of Squares*, suma del cuadrado de los residuos) es la suma de los cuadrados de las diferencias entre el valor que predijo el modelo y el valor real.

Regresión lineal simple

El modelo más simple utiliza sólo una variable x que es el *input* del modelo para predecir algo (*output* o salida), Ec. (4.4). Estos modelos representan la relación entre las variables x e y, mediante una función de la forma $y = f(x)$. El modelo no va a ser 100 % seguro por lo que va a existir una medida de error, ϵ, para cada observación. El error se trata como si fuera aleatorio, por lo tanto, $E[\epsilon] = 0$, siendo $E[\epsilon]$ la esperanza de ϵ (éste puede ser positivo o negativo, dependiendo si la variable de salida cae por debajo o por encima de la curva, lo cual es igualmente probable).

$$y_i = f(x_i) + \epsilon_i \tag{4.4}$$

Generalmente, se trata de elegir el modelo $f(x)$ y luego, utilizando los datos, se estima una función $\hat{f}(x)$.

Estimación por mínimos cuadrados para el modelo de regresión simple

El método de mínimos cuadrados es utilizado para estimar los parámetros desconocidos en un modelo de regresión lineal, minimizando las diferencias entre las respuestas observadas en algún conjunto de datos y las respuestas predichas por su aproximación lineal.

Para un modelo lineal con dos parámetros, este método calcula β_0 y β_1, tales que minimicen la suma de los cuadrados de las desviaciones o errores (4.5)

$$\epsilon_i = Y_i - (\beta_0 + \beta_1 x_i) \tag{4.5}$$

La suma de los cuadrados de los errores se expresa en la Ec. (4.6) [8].

$$\sum_{k=1}^{n} \epsilon_i^2 = \sum_{k=1}^{n} (Y_i - \beta_0 - \beta_1 x_i)^2 \tag{4.6}$$

El estimador b_0 de β_0 y b_1 de β_1 por mínimos cuadrados son, respectivamente, las ecuaciones (4.7) y (4.8) [8].

$$b_0 = \bar{y} - b_1\bar{x} \tag{4.7}$$

$$b_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2} \tag{4.8}$$

Una vez obtenidos los estimadores por mínimos cuadrados, se obtiene la recta de regresión estimada, Ec. (4.9) y Ec. (4.10).

$$\hat{Y}_i = \beta_0 + \beta_1 x_i \tag{4.9}$$

$$\hat{Y}_i = \bar{y} - \beta_1\bar{x} + \beta_1 x_i = \bar{Y} + \beta_1(x_i - \bar{x}) \tag{4.10}$$

Es importante destacar la importancia de lo que representa ϵ_i. Se denomina estimador del residuo i (aunque no es un estimador en sí mismo) y brinda información sobre lo que puede faltar en el modelo de regresión obtenido [3].

Cálculo de los coeficientes empleando RSS

Dado un modelo de regresión lineal simple, se puede calcular sus coeficientes minimizando el *RSS*.

$$y_i = w_0 + w_1 x_i + \epsilon_i \tag{4.11}$$

A partir de la Ec. (4.11), que representa un modelo unidimensional, la «mejor» recta es la que minimiza la distancia entre el valor real y el predicho considerando a todas las observaciones.

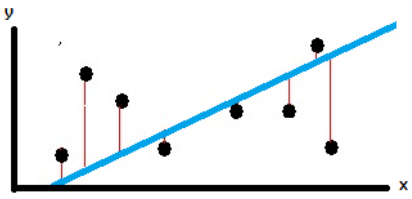

Figura 4.2: Representación gráfica del RSS

La recta en color azul en la Figura 4.2, representa un modelo lineal y se pretende que se encuentre lo más próximo a los puntos-dato. La función

objetivo, (4.12), es hallar el mínimo *RSS*, (4.13).

$$min_{w_0 w_1} \sum_{i=1}^{N} (y_i - [w_0 + w_1 x_i])^2 \tag{4.12}$$

$$min_{w_0 w_1} RSS(w_0, w_1) \tag{4.13}$$

En el *Anexo 7, Regresión,* se presenta el desarrollo de dicha solución en forma directa (obteniéndose los mismos valores, para los coeficientes del modelo, que en el método de los mínimos cuadrados) y en forma iterativa.

4.4.1. Regresión lineal general

En este modelo la salida depende de más de una variable predictora [8].

Generalización del modelo lineal

El modelo lineal se generaliza para D variables predictoras (atributos) en la Ec. (4.14)

$$y_i = w_0 h_0(x_i) + w_1 h_1(x_i) + ... + w_D h_D(x_i) + \epsilon_i = \sum_{j=0}^{D} w_j h_j(x_i) + \epsilon_i \tag{4.14}$$

La función $h_j(x_i)$ puede ser cualquier función. En general, $h_0(x) = 1$.El *input*: $\mathbf{x} = (x[1], x[2], \ldots, x[D])$ es un vector de dimensión D donde $x[k]$ es un escalar correspondiente a la componente k de $\mathbf{x}$. El término $h_k(\mathbf{x})$ es un escalar que corresponde a la característica k; puede ser una función de todos los componentes del vector $\mathbf{x}$. El vector de entrada para el punto t se representa a través de x_t. El término $x_t[k]$ representa al *input k-ésimo* del punto t. El número de observaciones (x_t, y_t) es N.

Notación matricial

A partir de la Ec. (4.14), la salida y_t puede escribirse como la multiplicación escalar de dos vectores. El primer vector es el de los parámetros $\mathbf{w} = w_0, ..., w_D$, el otro vector es el formado por todas las distintas características del *input* $h(x_i) = h_0(x_i), ..., h_D(x_i)$. Al final de la multiplicación se debe sumar el error final, ϵ_i.

Si se consideran las N observaciones con sus D atributos, se puede construir una matriz de dimensión $N \times (D+1)$, pudiéndose reescribir $Y = Hw + \epsilon$, donde:

$$Y = (y_1, y_2 \ldots, y_N)^T$$
$$\epsilon = (\epsilon_1, \epsilon_2 ..., \epsilon_N)^T$$
$$w^T = (w_0, w_1 \ldots, w_D)^T$$

$$H = \begin{pmatrix} h_0(x_1) & h_1(x_1) & \cdots & h_D(x_1) \\ \vdots & \ddots & & \vdots \\ h_0(x_N) & h_1(x_N) & \cdots & h_D(x_N) \end{pmatrix}$$

H es rectangular y posiblemente la cantidad de filas (observaciones) es mucho mayor que la de columnas (que son funciones de las características).

En el *Anexo 7* se obtiene la Ec. (4.15) que representa el cálculo directo de los coeficientes para el modelo de regresión lineal general, minimizando el *RSS*. En dicho anexo se presenta, además, el algoritmo para el mismo cálculo utilizando el gradiente descendente.

$$\hat{\mathbf{w}} = (H^T H)^{-1} H^T \mathbf{y} \tag{4.15}$$

4.4.2. Regresión RIDGE y LASSO

Para hallar los coeficientes en el modelo de regresión, considerando (4.15), es necesario que la matriz $H^T H$ sea invertible. Cuando no lo es, la regresión *RIDGE* es útil, debido a que presenta una modificación simple de la solución de forma cerrada, que conduce a formas siempre invertibles. Esto permite manejar problemas donde se tienen muchas dimensiones, incluso los casos donde éstas superan al número de observaciones. Permite determinar los coeficientes $\mathbf{w}$, definiendo un balance entre la insuficiencia de un modelo simple para describir una relación, y las conductas inesperadas que pueden presentar modelos más complejos (pero más flexibles) al momento de ser utilizados. Esta regresión busca soluciones a los coeficientes $\mathbf{w}$, balanceando automáticamente el sesgo y la varianza (debido a que se ha observado que cuando un modelo está sobreajustado, los parámetros $\hat{w}$ se vuelven grandes en magnitud, tanto

sea un modelo polinomial o cualquier otro) [20].

RIDGE permite medir el ajuste de la curva a los datos. Asimismo presenta un término que codifica la complejidad del modelo. El desafío que se presenta es lograr un balance entre el ajuste a los datos y el término de la complejidad. La regresión *RIDGE* es más genérica que la regresión analizada anteriormente, siendo utilizada, por este motivo, también en otras técnicas de *ML* [20].

La regresión *LASSO* (*Least Absolute Selection Shrinkage Operator*) es un método de regresión de aprendizaje supervisado, también llamado método de regresión penalizado. Es un proceso de contracción de los coeficientes, que permite una mejor interpretación del modelo e identifica las variables más fuertemente asociadas a la variable de respuesta [20].

Regresión RIDGE

El objetivo de la regresión *RIDGE* es balancear automáticamente el sesgo y la varianza en función del tamaño de los parámetros $\hat{w}$. Considerar un modelo representado por la Ec. (4.14), del cual se asume que posee muchas características, implica que el número de dimensiones (D) es grande. En este caso, con pocos datos, el modelo sufre un rápido sobreajuste, pues, al ser el grado del polinomio elevado, es probable que esos puntos verifiquen el polinomio. No obstante, teniendo muchos datos, aún en modelos complejos, puede ocurrir sobreajuste aunque más lentamente. La regresión *RIDGE* busca balancear el ajuste de la función a los datos y la magnitud de los coeficientes. Para ello, se define una función de costo (que es una medida de calidad) como la suma de la medición del ajuste $RSS(w)$, más la medición de la magnitud de los coeficientes,$||w||_2^2$ (en referencia a la norma L_2, considerada como una medida de la complejidad del modelo [20]). La norma L_2 se define en la Ec. (4.16)

$$||w||_2^2 = \sum_{j=0}^{D} ||w||_j^2 \tag{4.16}$$

RIDGE controla la complejidad del modelo mediante la magnitud de los coeficientes (en función de la respuesta al ajuste de los datos) por medio de una función de costo que balancea el ajuste del modelo a los datos y la medida de su complejidad, Ec. (4.17).

$$Costo\ total = RSS(w) + ||w||_2^2 \tag{4.17}$$

La calidad del ajuste se hace mediante la Ec. (4.18), la suma de los residuos al cuadrado (*RSS*).

$$RSS(w) = \sum_{i=1}^{N}(y_i - h(x_i)^T w)^2 = \sum_{i=1}^{N}(y_i - \hat{y}_i(w))^2 \tag{4.18}$$

La Regresión *RIDGE* (o Regularización con la norma L_2), estudia el balance introduciendo un parámetro λ que calibra el control entre el ajuste y las magnitudes de los coeficientes. *RIDGE* permite obtener los coeficientes w, teniendo en cuenta el balance definido en la Ec. (4.19).

Regresión RIDGE

$$RSS(w) + \lambda||w||_2^2 \tag{4.19}$$

Discusión del valor de $||w||_2^2$ *al variar* λ

- Si $\lambda = 0$, la complejidad desaparece y todo se reduce a minimizar la suma de los cuadrados de los residuos. A los coeficientes obtenidos se les denomina $\hat{w}^{RSS}$.
- Si $\lambda \to \infty$, para cualquier solución donde $\hat{w} \neq 0$ el costo total es infinito por lo que el mínimo se obtiene considerando $\hat{w} = 0$. Entonces para $\lambda = \infty$ y $\hat{w} = 0$, el costo total es la suma de *RSS*, que es *0* (el segundo término también es cero ya que $||w||_2^2 = 0$).
- Para $0 \leq \lambda \leq \infty$, se obtiene $0 \leq ||\hat{w}||_2^2 \leq \hat{w}^{RSS}$

El camino de los coeficientes de RIDGE

El método *RIDGE* intenta obtener distintas soluciones de $\hat{w}$ variando λ. Cuando $\lambda = 0$ se tiene la solución de los mínimos cuadrados (*LSS*) y a medida que aumenta λ los coeficientes convergen a cero. Se denomina camino de los coeficientes, a la variación de éstos, según varía λ en $[0, +\infty)$. Un determinado λ^* será la mejor elección para $\hat{w}$ y es el que se utilizará, al obtener el modelo regularizado, para hacer las predicciones.

En la Figura 4.3 se observa el comportamiento de cada una de las características del modelo.

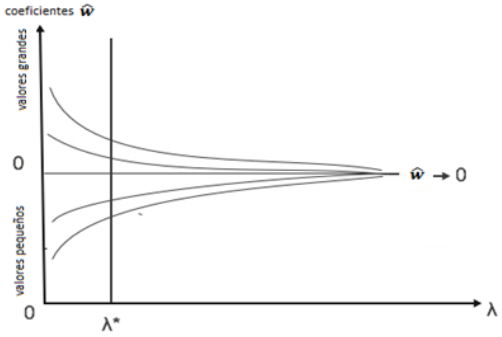

Figura 4.3: Comportamiento de los coeficientes en RIDGE

En el Anexo 7, se describe la optimización del objetivo *RIDGE* despejando directamente y mediante el empleo de una solución *ad hoc*.

Regresión LASSO

En situaciones donde la cantidad de características es muy alta (por ejemplo, en reconocimiento de imágenes) su procesamiento por métodos de regresión no resulta eficiente. Sin embargo, si el peso de algunas de sus características es *sparse*, se puede evitar dicho problema. La regresión *LASSO* impone una restricción sobre la suma de los valores absolutos de los parámetros del modelo, mediante una cota superior constante. Esta provoca la contracción y ocasiona que ciertos coeficientes se contraigan hacia cero siendo descartados y obteniéndose un modelo más simplificado, permaneciendo únicamente los predictores más importantes [20].

La regresión *LASSO* tiene una mayor precisión en la predicción, y mejora en la interpretabilidad del modelo, que el método de los mínimos cuadrados. Esto se debe a que si hay pocas observaciones y se dispone de muchos predictores, la varianza será alta. En este caso *LASSO*, al contraer los coeficientes, puede reducir la varianza, sin aumentar significativamente el sesgo. Para lograr su objetivo, *LASSO* cuenta con un parámetro λ que se ajusta para controlar la penalización. El aumento de λ provoca que los coeficientes se contraigan hasta llegar a cero.

En el contexto de regresión múltiple, se observa que la selección de las características es importante para hacer predicciones en términos de interpretación y eficiencia. En consecuencia, una buena cualidad que posee el método es la interpretabilidad, la cual es alcanzada al permitir sobrevivir las características que son relevantes para las tareas de predicción. De hecho, las dos

motivaciones más importantes en la selección de características son la obtención de interpretabilidad y permitir resolver computacionalmente problemas de dimensión alta.

Como se mencionó, se busca un balance entre el ajuste del modelo en el conjunto de entrenamiento y la medida de la magnitud de los coeficientes. Con el fin de ajustar los modelos, se utiliza el parámetro λ y para medir el ajuste se utiliza *RSS*. Se controla la medida de la magnitud de los coeficientes mediante la norma L_1 (suma sobre los valores absolutos de los coeficientes) lo que conduce a soluciones *sparse*. A esto se le denomina regresión *LASSO* o regresión regular L_1. El parámetro λ calibra cuánto se favorece el *sparsing* de la solución en función del ajuste de los datos de entrenamiento.

Discusión del valor de $||w||_1$ *al variar* λ

La discusión es similar a la efectuada para la regresión *RIDGE*. En este caso, la solución es llamado $\hat{w}^{lasso}$. Considerando la función de costo de *LASSO*, Ec. (4.20), se estudia su comportamiento en base a los valores posibles de λ.

$$Costo\ total = RSS(w) + \lambda||w||_1 \tag{4.20}$$

Si $\lambda = 0$ $\hat{w}^{lasso} = w^{RSS}$ queda solo *RSS*, solución no regularizada
Si $\lambda = \infty$ $\hat{w}^{lasso} = 0$
Si $\lambda \in (0, \infty)$ $0 \leq ||w^{lasso}|| \leq ||w^{RSS}||$

Eliminación de los coeficientes en LASSO

En la regresión *RIDGE*, incluso para valores elevados de λ, los parámetros son pequeños. En cambio, en el método *LASSO*, los patrones tienen distinto comportamiento, ya que los pesos de algunas características toman el valor cero, como se puede apreciar en la Figura 4.4. La imagen muestra el valor de los coeficientes en función de λ: cada uno se representa con una línea de puntos de un determinado color, al aumentar el valor del parámetro, se aproxima a cero, en forma variable. Por lo tanto, se puede tomar un determinado λ, excluir las características indicadas por el modelo, y así obtener una solución *sparse*.

En *LASSO*, para valores grandes de λ, puede ocurrir que casi todas las características desaparezcan del modelo prevaleciendo solamente unas pocas (a diferencia de *RIDGE*, en el cual se conservan todas).

Existen variantes en los algoritmos, combinando las funciones objetivo *LASSO* y *RIDGE* incluyendo además las penalizaciones de las normas L_1 y L_2 [21].

Otros enfoques, aparte de *LASSO*, para abordar el problema de la selección de las características, son a través de la enumeración explícita y empleando un método voraz. En el *Anexo 7*, se efectúa una breve descripción de las dos primeras y se analiza con detalle *LASSO*, incluyendo varios métodos para la optimización de su cálculo (Coordenadas descendentes, con características no normalizadas y normalizadas, entre otros). La visualización de las regresiones *RIDGE* y *LASSO* pueden apreciarse también en el *Anexo 7*.

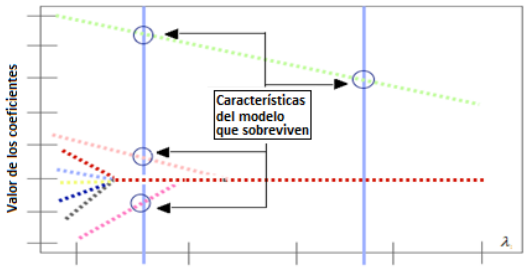

Figura 4.4: Camino de los coeficientes para LASSO

4.4.3. Impacto de Big Data en Estadística

Las características habituales de *Big Data* plantean algunos desafíos: la alta dimensionalidad trae acumulación de ruido y con ello correlaciones falsas; la alta dimensionalidad combinada con un tamaño de muestra grande crea problemas tales como gran costo computacional e inestabilidad algorítmica; la masividad de los datos, provenientes de diferentes puntos de tiempo, usando diferentes tecnologías, crea problemas de heterogeneidad, variaciones experimentales y sesgos estadísticos [18]. Algunos de estos problemas se pueden aliviar con el empleo de modelos estadísticos *sparse* los cuales contienen un pequeño número de parámetros distintos de cero. Estos modelos pueden ser mucho más fácil de estimar e interpretar que un modelo denso. La suposición de *sparse* permite abordar problemas relacionados a *Big Data*, posibilitando la extracción de patrones útiles y reproducibles de grandes conjuntos de datos [22]. Asumir modelos *sparse* tienen una justificación adicional basada en el «*Principio de Apuesta al sparcity*», el cual dice: «*Usar modelos que funcionen en problemas sparse, dado que no hay procedimientos que funcionen correctamente en problemas densos*» [22].

Caso de estudio

Debido a que la masividad de los datos (provenientes de diferentes puntos de tiempo, usando diferentes tecnologías) crea sesgos estadísticos, se propone el análisis del impacto de estos sesgos al considerar modelos de regresión como hipótesis.

4.5. Conclusiones

En el aprendizaje automático los algoritmos son diseñados para la búsqueda de relaciones, clasificaciones o patrones ocultos en los datos. Son especialmente apropiados cuando no se tiene ninguna idea de lo que se busca en ellos.

En el capítulo se presentaron los fundamentos de los algoritmos de reducción de la dimensión, los cuales pueden utilizarse en problemas que contienen muchas dimensiones, sobreviviendo las más importantes.

Los algoritmos de optimización permiten hallar mínimos de funciones utilizadas en el aprendizaje automático (por ejemplo, el mínimo del error generado por la diferencia entre el valor predicho por el modelo y el obtenido como valor real).

Se presentaron los modelos de regresión lineal (simple y general). La regresión lineal simple es representada por una función lineal como función predictora con una sola variable de entrada, que relaciona la entrada con la salida. Asociado a la representación, se analizó cuál es el costo de utilizar una curva como aproximación introduciéndose el *RSS* como una métrica que permite probar cómo se ajustan las distintas soluciones a los datos de entrenamiento. Para ello, se define una función objetivo y se determina el valor de los coeficientes que minimizan el *RSS*.

Luego se estudió la regresión múltiple (con más de una variable de entrada) la cual permite obtener funciones de ajuste más complicadas entre la entrada y la salida.

El sistema de ecuaciones para el modelo de regresión múltiple, es representado por una matriz rectangular H, siendo necesario que $H^T H$ sea invertible, para hallar los coeficientes del modelo, lo cual no siempre se cumple. Adicionalmente, el producto de las matrices intervinientes, para calcular la solución, es computacionalmente muy costoso para altas dimensiones, lo que justifica desarrollar algoritmos que reduzcan la dimensión.

En caso de que $H^T H$ no sea invertible, *RIDGE* presenta una modificación simple que conduce a matrices siempre invertibles. Este método permite el tratamiento de situaciones en las cuales hay muchas dimensiones e, incluso, si el número de éstas, supera al número de observaciones.

Otro objetivo de la regresión *RIDGE* es la obtención de los coeficientes w, balanceando automáticamente el sesgo y la varianza, considerando que cuando un modelo está sobreajustado, los parámetros toman valores grandes. Para ello, *RIDGE*, empleando la norma L_2, lleva a cabo una penalización que genera pesos pequeños.

La regresión *LASSO* tiene un objetivo similar al de *RIDGE*, pero empleando la norma L_1, multiplicada por un parámetro λ, lo que conduce a soluciones *sparse* (con coeficientes iguales a 0). La eliminación de los coeficientes se justifica debido a que, en muchas situaciones, existen atributos que no son necesarios para el modelo, y, asimismo, por razones de eficiencia e interpretabilidad, se desea considerar un conjunto menor de atributos. Este objetivo es alcanzado a través del camino de los coeficientes asociados a *LASSO* (si aumenta el valor de λ, la solución se torna más *sparse*).

LASSO es útil en *Big Data* para reducir las dimensiones, posibilitando la utilización de modelos *sparse*.

Capítulo 5

Algoritmos de Data Mining para Big Data

En el *Capítulo 2* se estableció que las tareas de *Big Data* no son capaces de ser enfrentadas por los métodos habituales, los cuales están sustentados sobre bases de datos relacionales. La analítica sobre *Big Data* es una variación de *Data Mining*, adaptando o diseñando nuevos algoritmos, para poder aplicar dichas técnicas.

En las siguientes secciones se describen las técnicas principales de minería de datos adaptadas a *Big Data* y otras que son exclusivas de este dominio.

5.1. Algoritmos de búsqueda

La búsqueda de ítems similares es efectuada mediante algoritmos cuyo objetivo es encontrar elementos (ítems) con una intersección en común con otros conjuntos (canastos o cestas). Estos algoritmos pueden aplicarse para clasificar artículos por tópico de los documentos en la Web, para detectar plagios, llevar a cabo el filtrado colaborativo (donde clientes con preferencias afines comparten el gusto por productos similares) y para la identificación de un mismo usuario en las redes sociales, entre otros.

5.1.1. Similaridad de documentos

Dado un número N (muy grande) de documentos, para la búsqueda de pares de ellos que sean iguales o parecidos, se puede utilizar las técnicas de *shingling, minhashing y LSH*. Teniendo en cuenta que se busca la similaridad a

nivel de los caracteres, no en el significado, la similaridad de *Jaccard* (*Sección 5.1.2*) es útil para resolver este problema.

Muchos problemas se pueden expresar como la búsqueda de objetos similares en un conjunto. En otras palabras, se puede describir como un método aproximado a la búsqueda de vecinos cercanos en un espacio de alta dimensión (*Sección 5.4*). Para ello se necesita introducir el concepto de distancia.

5.1.2. Similaridad y distancia de Jaccard

La similaridad de *Jaccard* entre dos conjuntos S y T se define en la Ec. (5.1).

$$sim(S,T) = \frac{|S \cap T|}{|S \cup T|} \tag{5.1}$$

La distancia de *Jaccard* se define en (5.2).

$$d(S,T) = 1\text{–}sim(S,T) \tag{5.2}$$

La distancia de *Jaccard* entre bolsas (conjuntos que pueden tener elementos repetidos) considera, para la unión, la suma de todos los elementos de las bolsas, y, para la intersección, la suma del mínimo de veces que aparecen los elementos en común, entre las dos bolsas.

Ejemplo 5.1

Sean dos bolsas A y B.
$A = \{a, a, a, b, b, b, b, c\}$
$B = \{a, a, a, b, b, d\}$

La unión tiene *14* elementos y la intersección tiene *5* elementos (los elementos en común son *a* y *b* y el mínimo de veces que aparece cada uno es *3* y *2*, respectivamente). Es decir $sim(A, B) = \frac{5}{14}$

Observar que la distancia de *Jaccard* entre dos conjuntos iguales es *1* pero la distancia entre dos bolsas idénticas es *1/2*.

En el caso de compras online, se puede considerar que dos clientes son similares si sus conjuntos de ítems comprados tienen una similaridad de *Jaccard* de un 20 %. Esto es suficiente para identificar clientes con gustos similares [4].

5.1.3. Shingles

A los efectos de determinar si dos documentos son iguales o para establecer su grado de similaridad, se define como $k - shingle$ (o $k - grams$) para un documento, a cualquier cadena de caracteres de longitud k, encontrada en él. El hecho de detectar muchos *shingles* en común entre dos documentos, indica que son muy parecidos. Las cadenas que se construyen pueden ser caracteres o palabras o pixeles, entre otros.

Si la elección de k fuese muy pequeña, puede ocurrir que la mayoría de las secuencias de cada letra aparezcan en todos los documentos, obteniéndose una alta similaridad de *Jaccard*, concluyéndose, equivocadamente, que son similares, cuando en realidad podrían no serlo.

En el *Ejemplo 5.2*, presentamos el cálculo de la similaridad de *Jaccard* y en los *Ejemplos 5.3 y 5.4* se muestra como, al detectar plagios, cambiar una palabra afecta solamente a los $k - shingles$ que están a una distancia k, a lo largo de esa palabra. Por otro lado, intercambiar dos párrafos de lugar, afecta $2k$ *shingles* en los bordes de los párrafos (al comienzo y al final).

Ejemplo 5.2

Sea $C = \{a, b, c, a, b\}$, los dos $2-shingles$ en C son: $\{ab, bc, ca\}$ (respetando el orden de la secuencia en C).

	D1	D2	D3	D4
Sh1	1	1	1	1
Sh2	1	1	0	1
Sh3	0	1	0	1
Sh4	1	0	0	1
Sh5	1	0	0	1
Sh6	1	1	1	0
Sh7	0	0	1	0

Tabla 5.1: Ejemplo de *shingles* por documento

En la Tabla 5.1, se representa la ocurrencia de *shingles* en un conjunto de documentos. Si se consideran las columnas 1 y 2, como si fueran los documentos $D1$ y $D2$, la intersección sería $sh1$, $sh2$ y $sh6$ (total 3) y la unión $sh1$, $sh2$, $sh3$, $sh4$, $sh5$, $sh6$ (total 6). La similaridad de *Jaccard* es *1/2* (considerando que son conjuntos).

Ejemplo 5.3

Intercambio en una frase: En la frase «*Un auto choca con otro*», se cambia una palabra: «*Un auto pega con otro*».

Si *k=3*, los *k-grams* que cambian son: $o\_c, \_ch, cho, hoc, oca, ca\_, a\_c$. Estos son los *3-shingles* encontrados a lo largo de la palabra *_choca_*

Ejemplo 5.4

Intercambio de párrafos: Se analiza la cantidad de *shingles* afectados al intercambiar párrafos en un plagio de documentos. Como ejemplo de intercambio de párrafos (Tabla 5.2), se consideran 5 párrafos, el primero empieza con A y termina en a, el segundo comienza con B y termina en b, y así sucesivamente. En dicha figura se visualiza el documento inicial y el que resulta de intercambiar el segundo párrafo ($B...b$) con el cuarto ($D...d$). Considerando $k = 2$ en el $k - shingle$, los cambios son: aB, bC, cD, dE (que totalizan $2k$ cambios en los shingles).

Antes	Después
A...a	A...a
B...b	D...d
C...c	C...c
D...d	B...b
E...e	E...e

Tabla 5.2: Ejemplo de plagio, intercambiando párrafos

El tamaño de k puede depender de la extensión de un documento. No obstante, lo importante es que k sea elegido lo suficientemente grande, como para que la probabilidad de que cualquier *shingle* aparezca en un documento dado, sea baja [4].

5.1.4. Minhashing

Debido a que los conjuntos de *shingles* de muchos documentos son muy grandes, se sustituyen por firmas a las cuales se les exige que, dadas las firmas de dos documentos, sea posible determinar la similaridad de *Jaccard* entre ellos. Estas firmas no dan la exacta similaridad de los documentos que representan pero, en cambio, proveen de una estimación aceptable para el objetivo

planteado [4]. *Minhashing* es una función, representada por h, utilizada para estimar rápidamente la similaridad entre dos conjuntos.

Las firmas que se buscan construir, se encuentran basadas en cientos de operaciones. En este caso serán *minhashings*.

La función $h(C)$ debe entrar en memoria y debe definirse de forma tal que $sim(C_1, C_2)$ sea la misma que $P(h(C_1) = h(C_2))$.

La idea es encontrar una función $h(.)$ la cual verifica:

Si $sim(C_1, C_2)$ es alto, entonces la probabilidad de $h(C_1) = h(C_2)$ es alta.
Si $sim(C_1, C_2)$ es bajo, entonces la probabilidad de $h(C_1) \neq h(C_2)$ es alta.

La función $h(C)$ convierte grandes *sets* de datos en pequeñas firmas, preservando la similaridad [23]. La idea es reemplazar grandes conjuntos por pequeñas representaciones llamadas «firmas». Se intenta determinar la distancia de *Jaccard* entre las firmas, asociando el resultado, a los conjuntos correspondientes. Se espera que la mayor parte de los pares similares caigan en la misma cubeta. La función de *hash* para la similaridad de *Jaccard* se denomina *Minhash* [23].

Sumarización de los sets que preservan la similaridad

Elemento	S_1	S_2	S_3	S_4
a	1	0	0	1
b	1	1	0	0
c	1	0	1	0
d	1	0	1	1

Tabla 5.3: Ejemplo de matriz característica

La Tabla 5.3, presenta un ejemplo de una matriz característica. Ésta se construye colocando una entrada de *1* si el elemento *a* se encuentra en el conjunto S_1, en caso contrario, es un *0*.

Para efectuar un *minhash* de una columna de la matriz característica, se toma una permutación de las filas, siendo el valor del *minhash*, el número de la primera fila en que aparece un 1. Las firmas son el resultado de muchas operaciones (cientos), donde cada una es un minhash.

A continuación presentamos el *Ejemplo 5.5* para ilustrar su cálculo.

Ejemplo 5.5

Sea M la matriz característica inicial de la Tabla 5.3. Se realizó una permutación, quedando las filas, en el orden indicado en la Figura 5.1 (según la primera columna (en color rojo), en la primer imagen y se aprecia una segunda permutación en la imagen de la derecha.

P1	S_1	S_2	S_3	S_4
4	1	0	1	1
1	1	0	0	1
3	1	0	1	0
2	1	1	0	0

Primera permutación aleatoria de M

P2	P1	S_1	S_2	S_3	S_4
3	4	1	0	1	0
2	1	1	1	0	0
4	3	1	0	1	1
1	2	1	0	0	1

Segunda permutación aleatoria de M

Figura 5.1: Permutaciones aleatorias sobre M

Para los valores en la primer imagen de la Figura 5.1, las firmas para cada conjunto S_i son:

$h(S_1) = 1,\ h(S_2) = 4,\ h(S_3) = 1,\ h(S_4) = 1$

Para la imagen de la derecha , las firmas son:

$h(S_1) = 1, h(S_2) = 2, h(S_3) = 1, h(S_4) = 3$

Considerando dos conjuntos S_1 y S_2 y visualizando sus columnas en la matriz característica, existen tres casos posibles:

- X (1 en ambas columnas)
- Y (1 y 0 o 0 y 1)
- Z (ambos 0)

Como la matriz es *sparse*, la mayoría de los casos son de tipo Z. Sin embargo, la $sim(S_1, S_2)$ y la probabilidad de que $h(S_1)$ y $h(S_2)$ sean iguales, es determinada por los conjuntos X, Y. Para ello basta con suponer que existen x filas del tipo X e y filas del tipo Y, entonces se verifica la Ec. (5.3).

$$sim(X, Y) = \frac{x}{x + y} \tag{5.3}$$

La igualdad anterior se debe a que x es el tamaño de $X \cap Y$ y el de $X \cup Y$, es $(x + y)$. A continuación se analiza la probabilidad de $h(S_1) = h(S_2)$.

Si se observan las filas permutadas aleatoriamente, comenzando desde arriba, la probabilidad de encontrar un tipo X antes que un tipo Z en las fila es $\frac{x}{x+y}$. A su vez si se considera que la primera fila es del tipo X antes que la Z, es muy probable que $h(S_1) = h(S_2)$ y por lo tanto $P(h(S_1) = h(S_2)) = \frac{x}{x+y}$. Si son del tipo Y antes que un tipo Z, el valor 1 lo toma de una función de *minhash* sobre un *shingle* y el *set* con los ceros recibirá más adelante la lista permutada. Por lo tanto, $h(S_1) \neq h(S_2)$ y se concluye que $P(h(S_1) = h(S_2)) = \frac{x}{x+y}$.

Cómputo de los Minhash

La idea de permutar las filas en una matriz característica de dimensiones grandes, no es viable computacionalmente. Para ello, se considera una nueva función de *hash* que mapee los números de fila K, a un número similar de cubetas. Algunos pueden caer en la misma cubeta, lo cual se soluciona con un valor de K lo suficientemente grande, evitándose muchas colisiones. En lugar de considerar n permutaciones aleatorias de las filas, se consideran n funciones de *hash* sobre las filas elegidas aleatoriamente.

Algoritmo para la construcción de la matriz de firmas.

Sea $F(i, c)$ el elemento de la matriz firma F para la función h_i y la columna c.

Inicialmente $F(i, c) = \infty$ (para todo i, c)

Para una fila r

1) Computar $h_1(r), ..., h_n(r)$

2) Para cada valor de la columna c

Si $c = 0$, no se hace nada

Si $c = 1$, para cada $i = 1, ..., n$, $F(i, c) = min\{F(i, c), h_i(r)\}$

En el siguiente ejemplo presentamos la construcción de firmas por medio del algoritmo.

Ejemplo 5.6

Se aplica a la Matriz M de la Tabla 5.3 las funciones de *hash* de la Tabla 5.4.

Fila	S_1	S_2	S_3	S_4	$h_1(x) = x \ mod \ 4$	$h_2(x) = 7 \times x \ mod \ 4$
0	1	0	0	1	0	0
1	1	1	0	0	1	3
2	1	0	1	0	2	2
3	1	0	1	1	3	1

Tabla 5.4: Funciones de hash sobre la matriz M

Se inicializa la matriz de firmas con todas sus entradas ∞ (Figura 5.5).

	S_1	S_2	S_3	S_4
h_1	∞	∞	∞	∞
h_2	∞	∞	∞	∞

Tabla 5.5: Inicialización

Luego se aplican las funciones de *hash* a la primer fila en la matriz M, (Tabla 5.6). La notación empleada S_{ij} indica fila i e iteración j.

Para la fila 0 de M	
$F(h_1, S_{10})$	$min\{F(h_1, S_{10}), h_1(0)\} = min\{\infty, 0\} = 0$
$F(h_1, S_{20})$	$\infty (porque \ tienen \ un \ 0)$
$F(h_1, S_{30})$	$\infty (porque \ tienen \ un \ 0)$
$F(h_1, S_{40})$	$min\{F(h_1, s_{10}), h_1(3)\} = min\{F(h_1, s_{10}), h_1(3)\} = min(\infty, 3) = 3$
$F(h_2, S_{10})$	$min\{F(h_2, s_{10}), h_2(0)\} = 0$
$F(h_2, S_{20})$	∞
$F(h_2, S_{30})$	∞
$F(h_2, S_{40})$	$min\{\infty, 1\} = 1$

Tabla 5.6: Resultados de aplicar el *Hash* a M

Luego de la primera iteración del algoritmo se obtienen los resultados de la Tabla 5.7).

	S_1	S_2	S_3	S_4
h_1	0	∞	∞	3
h_2	0	∞	∞	1

Tabla 5.7: Resultados de la primera iteración

Se vuelve a procesar el algoritmo con sus correspondientes resultados (Tablas 5.8 y 5.9).

Para la fila 1 de M	
$F(h_1, S_{11})$	$min\{F(h_1, S_{11}), h_1(1)\} = min\{0, 1\} = 0$
$F(h_1, S_{21})$	$min\{\infty, 1\} = 1$
$F(h_1, S_{31})$	*queda igual* (∞)
$F(h_1, S_{41})$	*queda igual* (3)
$F(h_2, S_{11})$	$min\{F(h_2, s_{11}), h_2(1)\} = min\{0, 3\} = 0$
$F(h_2, S_{21})$	$min\{\infty, 3\} = 3$
$F(h_2, S_{31})$	*queda igual* (∞)
$F(h_2, S_{41})$	*queda igual* (1)

Tabla 5.8: Segunda iteración

	S_1	S_2	S_3	S_4
h_1	0	1	∞	3
h_2	0	3	∞	1

Tabla 5.9: Resultados segunda iteración

En la tercera iteración aún se obtienen cambios (Tablas 5.10 y 5.11).

Para la fila 2 de M	
$F(h_1, S_{12})$	$min\{F(h_1, S_{12}), h_1(1)\} = min\{0, 1\} = 0$
$F(h_1, S_{22})$	*queda igual* (1)
$F(h_1, S_{32})$	$min\{\infty, 2\} = 2$
$F(h_1, S_{42})$	*queda igual* (3)
$F(h_2, S_{12})$	$min\{F(h_2, s_{12}), h_2(1)\} = min\{0, 0\} = 0$
$F(h_2, S_{22})$	*queda igual* (3)
$F(h_2, S_{32})$	$min\{2, \infty\} = 2$
$F(h_2, S_{42})$	*queda igual* (1)

Tabla 5.10: Tercera iteración

	S_1	S_2	S_3	S_4
h_1	0	1	1	3
h_2	0	3	2	1

Tabla 5.11: Resultados tercera iteración

En la cuarta iteración no hay más cambios en la firma (Tablas 5.12 y 5.13).

Para la fila 2 de M	
$F(h_1, S_{13})$	$min\{0,0\} = 0$
$F(h_1, S_{23})$	*queda igual* (1)
$F(h_1, S_{33})$	$min\{1,2\} = 1$
$F(h_1, S_{43})$	$min\{3,3\} = 3$
$F(h_2, S_{13})$	$min\{0,0\} = 0$
$F(h_2, S_{23})$	*queda igual* (3)
$F(h_2, S_{33})$	$min\{2,2\} = 2$
$F(h_2, S_{43})$	$min\{1,1\} = 1$

Tabla 5.12: Cuarta iteración

	S_1	S_2	S_3	S_4
h_1	0	1	1	3
h_2	0	3	2	1

Tabla 5.13: Resultados cuarta iteración (permanece incambiada)

Para la comparación de algunos valores obtenidos se utilizan los últimos resultados (tabla 5.14).

S_1	S_2	S_3	S_4
0	1	1	3
0	3	2	1

Tabla 5.14: Últimos resultados

Cálculo de la similaridad

similaridad firmas		similaridad documentos	
sim (S3, S4)	0	Sim de *Jaccard* (S3, S4)	1/3
sim (S1, S2)	0	Sim de *Jaccard* (S1, S2)	1/4
sim (S2, S4)	0	Sim de *Jaccard* (S2, S4)	0
sim (S2, S3)	1/2	Sim de *Jaccard* (S2, S3)	0

Tabla 5.15: Cálculo de la similaridad

Las diferencias se explican porque se trata de un ejemplo muy chico para que la Ley de los Grandes Números asegure una estimación acertada.

5.1.5. Funciones de hash localmente sensibles

Para las búsquedas de objetos similares, existe el método *LSH* (*Locality Sensitive Hashing*), el cual es utilizado en muchas aplicaciones de *Big Data*,

debido a que gestiona la comparación de, al menos, n pares de objetos linealmente ($O(n)$) [4].

LSH para firmas de minhash

La técnica de *minhash* proporciona un procedimiento que permite la compresión de los documentos, en firmas que preservan la similaridad de cualquier par de ellos. Computacionalmente, resulta muy costoso encontrar los pares con mayor similaridad (incluso, si son pocos los documentos, los pares de ellos pueden ser una gran cantidad). Una posible solución es comparar los pares que son más probables que sean similares, evitando la inspección de todos los pares.

La idea del *LSH* es, a partir de la colección de firmas anteriores, encontrar una lista de pares candidatos a ser similares. Para ello, se define una función de *hash*, y se eligen los candidatos, entre los elementos que caen en la misma cubeta. Es importante considerar minimizar los falsos positivos o falsos negativos, lo cual se logra por medio de la utilización de varias funciones de *hash*. En el *Anexo 8*, *Técnicas de Data Mining y Big Data*, se desarrolla esta técnica.

5.1.6. Obtención de candidatos

Para la obtención de un conjunto de pares de candidatos, para documentos similares, así como también para decidir, cuales lo son [4], se procede del siguiente modo:

1. Definir un valor k y construir los *k-shingles* de cada documento (opcionalmente, aplicar una función de *hash* a los *k-shingles* con un número menor de cubetas).
2. Ordenar los pares de documentos por *shingle*.
3. Elegir un valor n para las firmas de *Minhash* y calcular las firmas de *minhashing* para todos los documentos.
4. Elegir un umbral t, número de bandas b (*Anexo 8*), número de filas r, que cumplan, $br = n$ y $t = (\frac{1}{b})^{\frac{1}{r}}$. Si es importante reducir los falsos negativos, es conveniente reducir t en función de b y r. Si, en cambio, interesa un procesamiento rápido y se quiere limitar la cantidad de falsos positivos, se debe aumentar t.
5. Construir los pares de candidatos mediante la técnica *LSH*, para firmas de *minhash*.

6. Examinar cada par de firmas candidatas y analizar si la fracción de componentes en los cuales coinciden, es al menos t.
7. Opcionalmente, si las firmas son suficientemente similares, confirmar con los propios documentos si son similares o si por razones fortuitas, coincidieron sus firmas.

5.1.7. Resumen de algoritmos de búsqueda

La búsqueda de artículos similares entre conjuntos es un arquetipo que subyace en varios problemas, como el análisis de redes sociales, método de k vecinos más cercanos, *clustering* y sistemas de recomendación, entre otros.

En esta sección se comenzó describiendo el concepto de distancia y similaridad de *Jaccard*. Se definieron los *k-shingles* sobre documentos como cualquier *string* que contiene k caracteres consecutivos. A partir de ahí, se define una técnica de generación de firmas denominada *minhashing* y *LSH*.

El método de *minhashing* realiza la compresión de los documentos, con firmas que mantienen la similaridad de cualquier par de ellos. Están basados en una lista de permutaciones que se hacen sobre la matriz característica. Desde un punto de vista de costo computacional, éste es muy alto, debido a que se debe buscar todos los pares similares.

La técnica de *LSH* considera la colección de firmas de *minhash* para encontrar una lista de pares candidatos a ser similares. En las secciones *5.5, Vecinos más cercanos*, y en *5.6, Arboles de decisión*, *LSH* es empleada como base para solucionar escenarios con grandes cantidades de datos.

5.2. Algoritmos de recuperación de ítems

En el problema de *ítemsets* frecuentes, se busca determinar el total de conjuntos (o canastos) que contienen un determinado conjunto de elementos (o ítems), interesando la cantidad que superan un determinado valor de umbral, denominado cobertura.

Notar que el problema de *ítemsets* frecuentes, es diferente al de la búsqueda de artículos similares visto en la *Sección 5.1* (en ese caso se buscan ítems que tenían una parte del canasto en común, no importando el número total de canastos).

Dados múltiples conjuntos que tienen elementos en común, el descubrimiento de conjuntos de ítems que verifican un determinado patrón, constituye un problema importante que también puede ser abordado como el de la búsqueda de reglas de asociación. Una regla de asociación es una proposición probabilística sobre la ocurrencia de ciertos estados en un conjunto de transacciones. Pueden ser aplicados para análisis de canastos de compras y estudio de textos, entre otros.

Terminología

Canasto de compras: Es una relación $n : n$ entre dos conjuntos; uno de ellos denominado ítems y el otro denominado cesta, canasto o transacción.

Ítemset: Conjunto de ítems.

Soporte o cobertura: Fracción de las transacciones que contienen un ítemset. Es representado por s, Ec. (5.4).

Itemsets frecuentes: Son conjuntos de ítems que aparecen en canastos de compras en una cantidad mayor que el soporte.

K-ítemset frecuente: Son *ítemsets* frecuentes con k elementos.

5.2.1. Reglas de asociación

Las reglas de asociación pueden describirse con la siguiente proposición:
Si $A \Rightarrow i$.
A es un conjunto de ítems e i es un ítem. La interpretación de la regla es la siguiente: Si todos los ítems de A están en un conjunto, entonces el ítem i, «muy probablemente», también lo estará. El concepto «muy probablemente» se representa por una función llamada confiabilidad.

Confiabilidad

Número de instancias que la regla predice correctamente. También se puede definir como $P(i|A)$, es decir la probabilidad de que dada una transacción que contiene a A, también contenga a i, Ec. (5.5)

$$cobertura(X \rightarrow Y) = P(X \cup Y) \tag{5.4}$$

$$confiabilidad(X \rightarrow Y) = P(Y|X) \tag{5.5}$$

Interpretación de la regla de asociación

Suponer que del análisis de un grupo de transacciones se obtiene $cobertura = 0,05$ y $confiabilidad = 0,8$ con respecto a la regla de asociación: $\{A, B\} \rightarrow C$. La interpretación es que el 80 % de los clientes que compran A y B también compra C y, a su vez, el 5 % de los clientes compran los ítems A, B *y* C en una transacción.

El conjunto $A \cup B$ es una cesta que contiene a ambos: A y B. Por lo tanto $P(A \cup B)$ es el porcentaje de transacciones que verifican tener a ambos sobre el total de transacciones (no se debería interpretar como A o B).

El conjunto vacío es un subconjunto de todos los cestos, por lo tanto es el más frecuente, pues se encuentra en todos los canastos.

Ejemplo 5.7

En este ejemplo repasamos los conceptos de itemsets frecuentes, cobertura y confiabilidad. A partir de los valores de la Tabla 5.16, se busca determinar las reglas de asociación existentes en las transacciones. Es necesario analizar todos los casos en las transacciones indicadas y determinar qué condiciones cumplen cada una.

Tid	Ítems comprados
1	Bolígrafo, goma y cuaderno
2	Bolígrafo, compás y cuaderno
3	Bolígrafo, cuaderno y escuadra
4	Goma, escuadra y regla
5	Goma, compás, cuaderno, escuadra y regla
6	Goma, escuadra, regla, bolígrafo y cuaderno
7	Goma, compás, cuaderno, escuadra y bolígrafo

Tabla 5.16: Canastos de compra

El primer conjunto es el vacío, que es parte de todos los conjuntos, su soporte es *7*. El segundo conjunto corresponden a los $1-itemset$, por ejemplo {Bolígrafo} y está presente en *5* conjuntos. Se dice que su cobertura es *5/7*. Si,por ejemplo, se consideran conjuntos *2-ítemsets* frecuentes, cuya cobertura sea mayor o igual a *0,57*, se obtienen los subconjuntos de la Tabla 5.17.

2-ítemsets	Cobertura	
Bolígrafos, Cuadernos	5/7	0,71
Gomas, Cuadernos	4/7	0,57
Cuadernos, Escuadras	4/7	0,57
Gomas, Escuadras	4/7	0,57

Tabla 5.17: Valores de la cobertura

Los *ítemsets* mencionados anteriormente, verifican la condición de cobertura, por lo que se debe comprobar la confiabilidad: $Conf(x \rightarrow y) = P(y|x)$

Para el ejemplo de la Tabla 5.18, si se supone que se acepta una confiabilidad situada en 82 %, se debería estudiar los siguientes casos y ver si la probabilidad condicional es mayor o igual a $0,82$. Considerando esta restricción de confiabilidad, el 2-itemset frecuente $\{cuaderno, bolígrafo\}$ lo cumple en los dos sentidos: $\{Bolígrafos \rightarrow Cuadernos\}$ y $\{Cuadernos \rightarrow Bolígrafos\}$.

$P(\{Bolígrafos \rightarrow Cuadernos\})$	$P(Cuadernos\|Bolígrafos)$	5/5	1
$P(\{Cuadernos \rightarrow Bolígrafos\})$	$P(Bolígrafos\|Cuadernos)$	5/6	0,83
$P(\{Gomas \rightarrow Cuadernos\})$	$P(Cuadernos\|Gomas)$	4/5	0,8
$P(\{Cuadernos \rightarrow Gomas\})$	$P(Gomas\|Cuadernos)$	4/6	0,67
$P(\{Cuadernos \rightarrow Escuadras\})$	$P(Escuadras\|Cuadernos)$	4/6	0,67
$P(\{Escuadras \rightarrow Cuadernos\})$	$P(Cuadernos\|Escuadras)$	4/5	0,8
$P(\{Gomas \rightarrow Escuadras\})$	$P(Escuadras\|Gomas)$	4/5	0,8
$P(\{Escuadras \rightarrow Gomas\})$	$P(Gomas\|Escuadras)$	4/5	0,8

Tabla 5.18: Valores de la confiabilidad para varias transacciones

5.2.2. Patrones frecuentes

Los patrones frecuentes se definen en base a los conceptos anteriores. Estos patrones incluyen: *ítemsets* frecuentes (presentados en la sección anterior), subsecuencias frecuentes o patrones secuenciales (por ejemplo un cliente primero compra un celular, luego compra una tarjeta de memoria y luego un protector) y subestructuras frecuentes (se refiere a formas de estructuras tales como grafos). Si una subestructura ocurre frecuentemente, se le denomina, patrón de estructura frecuente.

Representación de datos del canasto de compras

Patrones muy largos contienen un número combinatorio de sub-patrones, cuyo tratamiento se considera un problema *np–hard*. De hecho, generar todos

los subconjuntos de k elementos de un *ítemset* de n elementos tiene un costo de: $\frac{n^k}{k!}$ [4].

Guardado del conteo de los pares de ítems.

Una forma de representación de los ítems, es numerándolos estableciendo un orden: $i \leq j$. Si están en un arreglo bidimensional ($a[i, j]$), no sería necesario utilizar la mitad de la matriz.

Método de la matriz triangular

Otra solución, consiste en utilizar una matriz triangular. Se guarda $a[k]$, el conteo de los pares $\{i, j\}$ con $1 \leq j \leq k \leq n$ con $k = (i-1)(n-\frac{i}{2}) + j - i$

Resulta entonces un conteo en orden lexicográfico: $\{1, 2\}, \{1, 3\}, ...$ Luego $\{2, 3\}, \{2, 4\}, ...$, terminando en $\{n-1, n\}$.

Método de las triplas

El $\{i, j, c\}$ se interpreta como que el par $\{i, j\}$ ocurre c veces. Para ello, se utiliza un *hash* con i, j como claves. Con este método, si $c = 0$ no se necesita guardar información (a diferencia de la solución de la matriz triangular, que sí lo exige). Pero, a su vez, exige mantener en memoria tres números para cada par. Por regla general, si $\frac{1}{3}$ como mínimo de $\binom{n}{2}$ aparecen en algún canasto, se debería utilizar la matriz triangular. Si es menor, es conveniente emplear el método de las triplas [4].

Aunque este tiempo es mayor que el tiempo de transferencia de disco a memoria, usualmente k varía entre 2 y 3 y, si k es mayor, se pueden descartar elementos del *ítemset* (n se reduce). Es decir, en el proceso de construcción, si dos elementos no están juntos, se pueden eliminar todos los súper conjuntos que podrían contener a esos dos elementos.

Monotonicidad

Si un conjunto I es frecuente también lo es cualquiera de sus subconjuntos. Si existe una tripla frecuente, entonces existen tres pares frecuentes contenidos dentro de ella. Existen pares frecuentes contenidas en triplas no frecuentes. Por lo tanto, se espera encontrar más pares frecuentes que triplas frecuentes, etc. En la búsqueda de las reglas de asociación, el principal problema de estos algoritmos es la memoria, que resulta ser el punto crítico al momento de buscar

pares frecuentes (triplas frecuentes no son un problema ya que son raras) [4]. Esto se resuelve mediante el método de la matriz triangular o el método de las triplas que es más eficiente [4]. En el *Anexo 8* se describen brevemente los algoritmos *A Priori* y para grandes conjuntos de datos los algoritmos *PCY*, *Multihash* y *Multi Stage*, con algunos ejemplos. En la Figura 5.2 presentamos un esquema donde se visualiza un ordenamiento de los algoritmos señalados con algunas de sus características.

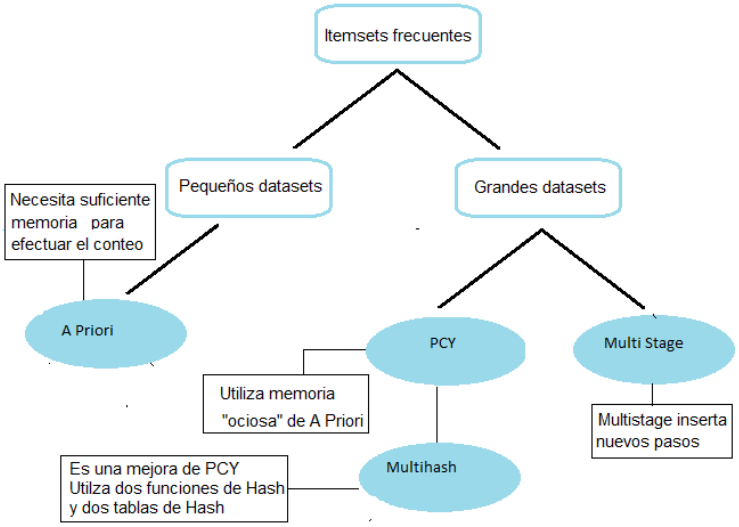

Figura 5.2: Diferentes algoritmos de ítemsets frecuentes

Algoritmos en pocos pasos

Los algoritmos en pocos pasos resuelven problemas distintos a la búsqueda *itemsets* más frecuentes. Por ejemplo, búsqueda de dos ítems comprados juntos, sin necesidad de buscar toda la colección de *ítemsets* frecuentes. La idea es comprimir el trabajo a uno o dos pasos [4]. Los algoritmos considerados en el *Anexo 8* son: algoritmo simple aleatorio, algoritmo *SON* y algoritmo *Toivonen*.

Reglas de asociaciones secuenciales

Las reglas de asociaciones secuenciales expresan patrones de comportamiento secuencial (separados en el tiempo). Los registros deben tener una marca de tiempo o alguna forma de saber el orden cronológico en el cual ocurrieron. Mediante un identificador, se sabe en qué orden sucedieron los eventos y sobre qué individuo. Estos registros forman una secuencia. Una sub-secuencia se define como cualquier conjunto ordenado de los elementos de la secuencia [10].

Algoritmo A Priori All

Un algoritmo para el aprendizaje de esta asociación secuencial, es el *A Priori All*, que tiene las siguientes fases [3]:

1. Ordenación por identificador, siguiendo la ordenación temporal.
2. Selección del conjunto de ítems: se refiere a que alcanzan a un mínimo de cobertura, respecto a cada individuo.
3. Transformación y renombramiento: se asigna a cada conjunto de ítems frecuentes un identificador. Cada secuencia se transforma de forma tal, que contenga los ítems frecuentes. Finalmente, se renombra cada conjunto por su identificador, en cada una de las secuencias.
4. Construcción de secuencias frecuentes: a partir del conjunto anterior, se construyen incrementalmente el conjunto de secuencias, que cumplen el criterio de cobertura.
5. Selección de secuencias máximas: filtra el conjunto de secuencias frecuentes, de manera que no haya subsecuencias, a partir de las de mayor tamaño

5.2.3. Resumen de algoritmos de recuperación de ítems

El modelo de *ítemsets* frecuentes consiste en dos conjuntos (ítems y canastos) con relaciones $n : n$ entre sus elementos. Los *ítemsets* frecuentes son los ítems que están en un conjunto y su cantidad es mayor que un umbral determinado.

Las reglas de asociación pueden ser vistas como la siguiente proposición: «Si $A \Rightarrow i$», donde A es un conjunto de ítems e i es un ítem.

Existen dos formas de medir la calidad de la regla: la precisión (o confiabilidad) y la cobertura (o soporte).

La forma de explorar los *ítemsets* frecuentes es mediante el algoritmo *A Priori*, el cual elimina los subconjuntos grandes partiendo de pequeños, utilizando el principio que indica: «Para que un conjunto sea frecuente, los subconjuntos que lo forman deben serlo también». Al evitar contarse todos los pares se optimiza el uso de la memoria.

El conteo de pares de ítems en muchos conjuntos es un problema *np-hard*. Por tal motivo los algoritmos que se diseñan pretenden optimizar el uso de la memoria.

El algoritmo *PCY* es una mejora de *A Priori* en cuanto a un mejor aprovechamiento de la memoria que este último no utiliza. Otros algoritmos utilizados son *multihash*, *SON* y *Toivonen*, entre otros.

Casos de estudio identificados

En la *Sección 5.2.2, Patrones Frecuentes,* se mencionó a las subsecuencias frecuentes que motivan los siguientes casos de estudio:

Subsecuencias frecuentes y sistemas de recomendación.

Identificación de secuencias de compra por parte de un usuario a nivel de sistemas de recomendación y las métricas necesarias para evaluarlas y las predicciones que se puedan realizar.

Confiabilidad y cobertura en subsecuencias frecuentes

Se plantea redefinir los términos de confiabilidad y cobertura en las subsecuencias frecuentes considerando el tiempo que transcurre al ocurrir los items.

Las reglas de asociación se definieron del tipo «Si $A \Rightarrow i$». Se busca definir la regla para las subsecuencias. Por ejemplo, suponer que el usuario compra el artículo A, luego el B y bastante tiempo después el C. En este caso la regla de asociación habría que adaptarla a estas compras secuenciales y, en base a ella, definir la cobertura y la confiabilidad condicionado a que las compras se realicen dentro de un determinado intervalo de tiempo.

Una idea a desarrollar es considerar una regla de asociación más simple: el usuario compra el artículo x. Luego, pasado un tiempo t, compra al ítem y. De esta forma la confiabilidad se define en la Ec. (5.6)

$$Conf(x \rightarrow y) = P(y|x,t) \tag{5.6}$$

La probabilidad $P(y|x,t)$ se interpreta como la probabilidad de que un usuario compre al ítem y , dado que el mismo usuario compró x antes de que transcurra un tiempo t.

5.3. Algoritmos de clasificación

La clasificación es una tarea utilizada frecuentemente en *ML*. Es empleada en una amplia gama de aplicaciones, incluyendo el análisis de opiniones, la

orientación de anuncios, clasificación de imágenes, la detección de spam, evaluación de riesgos y diagnósticos médicos, entre otros. Una clasificación asigna una tupla a una clase o a varias clases. En este caso se denomina categorización. Algunas técnicas utilizadas en la clasificación son: regresión lineal, logística y vecinos más cercanos.

El clasificador lineal más común es la regresión logística, el cual, además de predecir una clase, proporciona una probabilidad asociada con la predicción (denominada clasificación suave). En caso de que una observación pertenezca a varias categorías, el clasificador debería seleccionar la de mayor probabilidad.

5.3.1. Clasificador lineal

Un clasificador es un modelo representado por una función, que permite separar a un espacio en regiones. Tomando una entrada definida por las características presentes en los datos, el modelo realiza una predicción. En los Ejemplos 5.8, 5.9 y 5.10 presentamos la clasificación lineal aplicada al análisis de sentimientos.

Ejemplo 5.8

El análisis de opiniones o de sentimientos es un procedimiento que intenta identificar los sentimientos de un usuario que emite una opinión en la Web, a través de sitios o blogs. Una forma de implementarlo es mediante un clasificador del tipo lineal que asocia un coeficiente con los pesos de cada palabra en la opinión (habitualmente una frase).

Se considera el caso particular una frase que expresa la opinión de un estudiante sobre la calidad de un curso. En la frase se identifican dos entidades: la entrada (que es la frase en sí) y las características (las palabras que la componen). El modelo debe realizar una predicción sobre la opinión indicando si es positiva o negativa (se puede agregar otros estados: neutro, muy negativo o muy positivo, etc.).

El algoritmo, en su forma más simple, podría indicar si el comentario es positivo o negativo, contando las palabras positivas y negativas y, la que tenga mayor cantidad, es la que identifica el tipo de comentario. Una función que devuelve la suma ponderada de las palabras, puede ser el clasificador lineal. Este clasificador puede mejorarse a través de un adecuado entrenamiento, agregando un sistema de *score* (representado por s), determinándose si está por debajo

o por encima de un umbral a determinar. Entrenar un clasificador de tales características significa obtener coeficientes (ponderaciones) de cada palabra, para que las predicciones de las siguientes entradas, sean adecuadamente clasificadas. La performance de un clasificador depende de la interrelación entre el tamaño de las muestras, número de características y complejidad del modelo.

En el *Ejemplo 5.9* presentamos un ejemplo de su funcionamiento.

Ejemplo 5.9

Se dispone de una tabla con valores (Figura 5.19), la cual contiene un conjunto de palabras con sus respectivos pesos y se busca clasificar la siguiente frase (x) como positiva o negativa.

x: «El curso al cual asistí es excelente y el seguimiento también es excelente, pero el sistema de evaluación es malo»

El *score* correspondiente es: $s(x) = 2.2 - 1.1 = 3$ (El resto de las palabras al tener peso *0* no aportan a la suma).

Palabra	Peso
Bueno	1
Malo	-1
Desastroso	-2
Excelente	2
Lamentable	-2.5
Brillante	2.5
curso, cual, al, asistí, cine, es, que, pero, el, donde, del, y, evaluación, sistema	0

Tabla 5.19: conjunto de palabras con sus pesos

La regla que se dispone para el clasificador es:
Si $s(x) > 0$ $\hat{Y} = +1$, (indica que la opinión es positiva)
Si $s(x) < 0$, $\hat{Y} = -1$ (indica que la opinión es negativa)
En el caso de la frase x el clasificador indicaría que se trata de una opinión positiva.

Fronteras de decisión

Se considera, a continuación, un conjunto de frases con dos palabras x_1 y x_2, con sus respectivos pesos w_1 y w_2 y sea una función *score*, $s(f) = w_1.\#x_1 + w_2.\#x_2$, siendo f una frase. La función $s(f)$ determina una gráfica que define una frontera de decisión. Todas las frases se clasifican de acuerdo

a la cantidad de ocurrencias de las dos palabras x_1 y x_2, es decir los ejes coordenados serán $\#x_1$ y $\#x_2$ para las distintas frases $f, g, ..., h$ (Figura 5.3). De acuerdo a la cantidad de palabras x_1 y x_2, cada frase $f, g, ..., h$ se ubica en alguno de los semiplanos definidos por la recta $w_1.\#x_1 + w_2.\#x_2 = 0$, que se considera como la frontera de decisión. De acuerdo al signo del *score*, las frases se ubican en el semiplano correspondiente. Los puntos que pertenecen a la frontera de decisión tienen igual probabilidad de pertenecer a uno u otro semiplano. Generalmente, los problemas de este tipo, tienen miles de palabras que determinan hiperplanos.

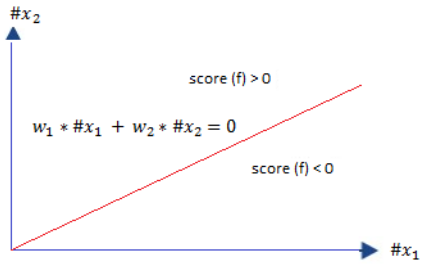

Figura 5.3: Clasificador lineal, fronteras de decisión

Modelo para un clasificador

Si el modelo es un hiperplano de dimensión D, cada *input* x tiene D dimensiones: $x[2], x[3], .., x[D]$. El x_j denota el *input* $j - ésimo$. La expresión $h_j(x_i)$ denota la característica $j - ésima$ del $i - ésimo$ punto dato x_i. El modelo de hiperplano simple se representa por $\hat{y}_i = sign(Score(x_i))$ siendo $Score(x_i)$ expresado en la Ec. (5.7).

$$Score(x_i) = w_0 + w_1 x_i[1] + ... + w_d x_i[d] = \mathbf{w}^T \mathbf{x_i} \tag{5.7}$$

Generalizando la ecuación anterior, la Ec. (5.8) representa el modelo de hiperplano.

$$Score(x_i) = w_0 h_0(x_i) + w_1 h_1(x_i) + \ldots + w_D h_D(x_i) = \sum_{j=0}^{D} w_j h_j(x_i) = \mathbf{w}^T h(\mathbf{x_i}) \tag{5.8}$$

Las funciones $h_0(x_i), ..., h_D(x_i)$ representan características, y h_0 es una constante. La predicción se representa por $\hat{y}$, que es el signo del *score* de una entrada en particular. Si es mayor que cero, predice +1, si es menor que cero

predice -1. Cuando es cero, puede tomar cualquiera de los dos valores $+1$ o -1. El objetivo es optimizar la función *Score*, para que el error en la predicción sea el menor, ajustando los coeficientes w_i.

Clasificación con una probabilidad asociada

En ciertos casos, interesa estimar la probabilidad de que un comentario sea positivo, cuando éste contiene términos positivos y negativos. Es decir, aportar la predicción con una probabilidad sindicada a ésta.

Ejemplo 5.10

Suponer que se tienen dos opiniones sobre un curso:

f_1: El curso fue excelente
f_2: El curso fue bueno pero la evaluación mala

La frase f_1: es un comentario claramente positivo pero f_2 es un poco dudoso. Es posible asociar probabilidades a la predicción y para diferenciar ambos casos debería cumplirse que $P(+1|f_1) > P(+1|f_2)$. El menor valor de la segunda probabilidad refleja la mayor duda de que sea positivo el comentario de f_2. Para el caso de las opiniones, cada una de ellas es un par (x_i, y_i) donde x_i representa ciertas características, tales como la cantidad de vocablos «bueno» o la cantidad de «excelente». El valor de y_i representa la etiqueta que le corresponde a las opiniones.

El aprendizaje del clasificador es optimizado utilizando métricas de calidad sobre el conjunto de entrenamiento, recibiendo como *output* las probabilidades condicionales: $\hat{P}$ en función de los parámetros $\hat{w}$ para el modelo. Por lo tanto, $\hat{P}$ es de utilidad para predecir $\hat{y}$ la clase predictiva que, en el ejemplo, es una clasificación del sentimiento de quien emitió la opinión.

El objetivo es entrenar las probabilidades condicionales $\hat{P}$, a partir de los datos, y luego, utilizarlo para predecir la clase más probable a la que pertenece cada frase. Si dicha probabilidad es mayor que un cierto valor, la opinión es positiva, y, si es menor, negativa. Si x es la frase para clasificar, el objetivo es encontrar la clase más probable a la que debe pertenecer $\hat{P}(y|x)$. Estimando la probabilidad $\hat{P}(y|x)$, se mejora la interpretabilidad del modelo, al tomar en cuenta la probabilidad con que ocurre la predicción.

5.3.2. Regresión logística

La regresión logística realiza predicciones cuyas variables de salida son categóricas, ya sea con dos clases o más. Es empleado para determinar la probabilidad de la respuesta utilizando una función logística.

Modelos lineales generalizados

Los modelos lineales generalizados permiten modelar situaciones expandiendo las suposiciones utilizadas para los modelos lineales. Estos modelos resuelven eficientemente algunos problemas para los cuales los métodos de regresión lineal presentan limitaciones. Asimismo, permiten formar regresiones cuando se tienen ciertas restricciones en la salida, es decir, cuando la salida pertenece a un intervalo.

Se busca relacionar $Score(x_i)$ con $\hat{P}(y = +1|\mathbf{x}; \hat{w})$ donde $\mathbf{x}; \hat{w}$ indica el vector $\mathbf{x}$ parametrizado por $\hat{w}$.

La función $Score(x_i) = w^T h(x_i)$ varía de $-\infty$ a $+\infty$, e interesa que si el *Score* es muy grande, sea casi una certeza que $\hat{y}_i = +1$. En este caso se cumple que $\hat{P}(y = +1|x_i) = 1$. Si el *Score* es muy negativo significa que $\hat{P}(y = +1|x_i) = 0$ y que $\hat{P}(y = -1|x_i) = 1$. Si el *Score* es 0, la frase se encuentra sobre la frontera de decisión y la opinión se considera neutra. En este caso es deseable que: $\hat{P}(y = +1|x_i) = 0.5$ y $\hat{P}(y = -1|x_i) = 0.5$.

La función que relaciona *Score* con $\hat{P}$ representa un caso de modelo lineal generalizado y consiste en comprimir el recorrido del modelo lineal (en este caso el *Score*) al intervalo $[0, 1]$.

Interpretación del $Score(x_i)$

Las etiquetas son: $\hat{y}_i = -1$ e $\hat{y}_i = +1$.

$Score(x_i) = w^T h(x_i)$ tiene un recorrido de $-\infty$ a $+\infty$ y el recorrido de $\hat{P}(y = +1|x_i)$ es $[0, 1]$.

La interpretación de la función de probabilidad es:

$\hat{P}(y = +1|x_i) = 0$ corresponde a un $Score(X_i) \to -\infty$.
$\hat{P}(y = +1|x_i) = 0.5$ corresponde a un $Score(X_i) = 0$.
$\hat{P}(y = +1|x_i) = 1$ corresponde a un $Score(X_i) \to +\infty$.

En forma análoga es la interpretación para $y = -1$.

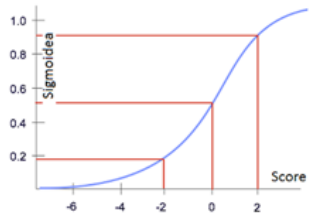

Figura 5.4: Función sigmoidea

Modelo de regresión logística

La regresión logística es un ejemplo de este tipo de funciones, dado que verifica la caracterización anterior. La Figura 5.4 grafica la función sigmoidea, que es utilizada por la regresión logística actuando como una función g que representa al modelo lineal generalizado. (Ec. (5.9)).

$$sigmoidea(Score) = \frac{1}{(1 + e^{-Score})} = g(Score) \tag{5.9}$$

A partir de $Score(x_i) = w^T h(x_i)$, se pueden definir las Ecuaciones 5.10 y 5.11.

$$\hat{P}(y = +1|x_i) = g(w^T h(x_i)) \tag{5.10}$$

$$\hat{P}(y = +1|x_i) = sigmoid(Score) = \frac{1}{1 + e^{-Score}} = \frac{1}{1 + e^{-w^T h(x_i)}} = \frac{1}{1 + e^{(w_0 h_0(x_i) + \ldots + w_D h_D(x_i))}} \tag{5.11}$$

Para que $g(w^T, x_i) \geq 0,5$, debe cumplirse que $w^T h(x_i) \geq 0$. Esto es debido a que $\frac{1}{1+e^{-w^T h(x_i)}} = 0,5$, si $w^T h(x_i) = 0$. Por lo tanto, si se predice $+1$, cuando $g(w^T, x_i) \geq 0,5$ entonces $w^T h(x_i) \geq 0$ y si se predice 0, el valor obtenido debe cumplir que $w^T h(x_i) < 0$.

Fronteras de decisión no lineales

En la Figura 5.5 se observa que, además de las fronteras de decisión lineales, existen fronteras no lineales. Por ejemplo, si la frontera de decisión fuera una circunferencia: $w^T h(x_i) = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1^2 + w_3 x_2^2$

Lo anterior podría predecir 1 si $w^T h(x_i) = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1^2 + w_3 x_2^2 \geq 0$ y predecir 0 si $w^T h(x_i) = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1^2 + w_3 x_2^2 < 0$. La frontera de decisión no depende de los datos de entrenamiento, sino de la hipótesis en función de los parámetros.

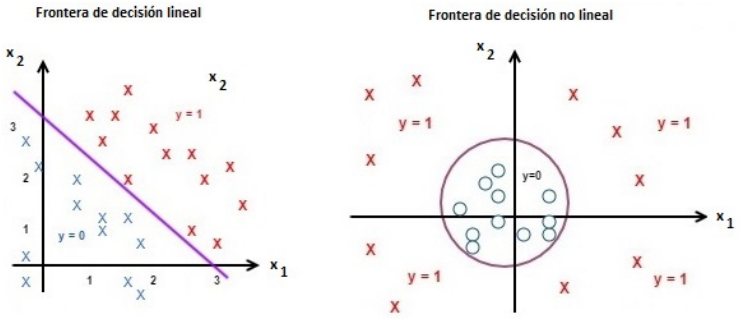

Figura 5.5: Fronteras de decisión

Métrica de calidad ajustando los coeficientes

En la Figura 5.6, la métrica de calidad para hallar el mejor clasificador, es aquella que maximiza la probabilidad de que los datos queden correctamente clasificados (en el ejemplo considerado se trata de una función de w_0, w_1 y w_2). Para ello, se debe maximizar el estimador, o la métrica de calidad, ajustando los coeficientes w_i, mediante un algoritmo adecuado.

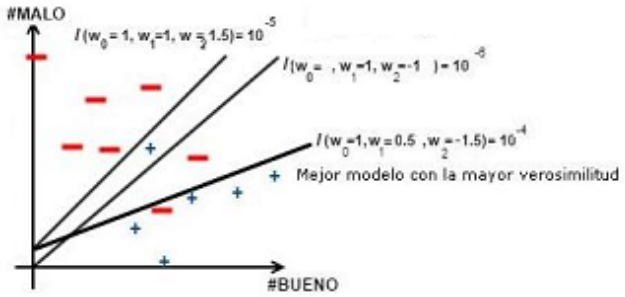

Figura 5.6: Calidad de los modelos

5.3.3. Clasificación multiclase

En lugar de realizar la clasificación en dos clases, se tienen $C_1, C_2, \ldots, C_k$ clases. Se presentan algunos métodos, a continuación.

Uno versus todos

Para el entrenamiento se utiliza el enfoque *1 vs. todos*. En lugar de dos categorías, se emplean k categorías. El entrenamiento del clasificador consiste en asignar el valor «+1», si la observación corresponde a una categoría, y su valor es «−1», si corresponde a cualquier otra. Para cada clase, se determina una línea que la separa del resto de los elementos de las otras clases. Si x_i pertenece a una clase determinada, el clasificador le asigna un *score* mayor que cero. Si está del otro lado de la línea, le otorga un valor menor que cero.

Por lo tanto, del lado de la clase C, se cumple: $Score(x_i) > 0$ y la $P(y = c|x_i, \mathbf{w}) > 0,5$.

En caso de estar en cualquier otra región diferente, el $Score(x_i) < 0$ y la $P(y = c|x_i, \mathbf{w}) < 0,5$

El método *1 vs. todos*, se aplica para cada una de las clases. Para la clasificación multi-clase, dada una determinada entrada, se debe comparar el resultado en todas las clases, y la que cuente con la mayor probabilidad será la seleccionada.

Todos los pares (All pairs)

Se construye un clasificador binario, únicamente, a partir de las muestras de dos clases, y se ignoran las restantes. El proceso se repite para todos los pares posibles de clases, obteniéndose $\frac{n(n-1)}{2}$ clasificadores, que luego son combinados (En la *Sección 5.6*, se desarrolla con más profundidad la combinación de los clasificadores).

5.3.4. Resumen

La clasificación es una de las tareas nombradas en el *Capítulo 1*. Existen numerosas técnicas que son utilizadas para realizar clasificaciones. Un clasificador es un algoritmo que distribuye a las observaciones en distintas clases. Se asume que a una observación le corresponde una sola clase. Se puede medir la calidad de un clasificador, a través de medidas de probabilidad asociadas a la clasificación.

La correspondencia de una función general, denominada *Score* (con dominio en el conjunto de los reales), con el modelo, se puede realizar a través de la función sigmoidea (utilizada en la regresión logística). Esta función comprime

el dominio de los reales al intervalo $[0, 1]$, que coincide con el recorrido de cualquier función de probabilidad.

Una métrica de calidad habitualmente utilizada para determinar el mejor clasificador, es el estimador de máxima verosimilitud.

5.4. Algoritmos de vecinos más cercanos

Luego de obtenido el modelo por regresión paramétrica, la flexibilidad para cambiarlo es limitada, debido a que está basado en un conjunto de características. Si se dispone de abundantes datos, se pueden considerar métodos más flexibles que los paramétricos.

El método de los vecinos más cercanos (*Nearest Neighbors)* es utilizado como un modelo de clasificación no paramétrico y también para tareas de regresión. Este tipo de hipótesis permite que la complejidad del modelo aumente a medida que crece la cantidad de los datos observados dando, como resultado, que los ajustes se adapten localmente a las observaciones. La técnica considera un conjunto de entrenamiento pre-procesado y almacenado cuyos elementos se denominan vecinos, los cuales definen varias clases (etiquetas). La decisión se toma cada vez que se debe clasificar un nuevo punto, al cual se le asigna la etiqueta mayoritaria, en base a los k vecinos más próximos. Para ello, se tiene en cuenta [4]:

- Distancia a utilizar.
- Cuántos vecinos se deben considerar.
- Peso de los vecinos más cercanos (función *kernel*).
- Cómo se define la etiqueta para el nuevo punto.

Técnica 1-NN

El más simple del método de vecinos más cercanos, es el $1 - NN$. Es la predicción para un punto x, fundamentada en los demás puntos del conjunto de entrenamiento (determinándose, mediante una distancia entre ellos, cuál es el más cercano, asociándole la misma etiqueta al punto x).

El algoritmo sigue los siguientes pasos.

Inicializar todas las distancias a infinito para los N puntos

Se tiene un dato x que se quiere clasificar
Para $i = 1$, $i \leq N$
    Calcular las distancias del punto x a x_i
Devolver el punto x_i cuya distancia a x es la menor

El método $1 - NN$ en múltiples dimensiones se puede visualizar mediante el empleo del diagrama o mosaico de *Voronoi*, el cual divide al espacio de entradas en regiones. Ante una nueva observación, se determina a qué celda corresponde. Aunque el orden para determinar los bisectores es n^2, el orden para construir el diagrama de *Voronoi*, puede alcanzar $O(nlog(n))$, empleando un algoritmo denominado *Fortune* (por su creador *Steven Fortune*), el cual se basa en un barrido del plano. Si las fronteras que determinan las regiones de *Voronoi* son pre-procesadas adecuadamente, se puede determinar la región a la que corresponde el punto dato en $O(\log n)$ comparaciones, en lugar de $O(n \log n)$ [4].

Algoritmo k-NN para la regresión

Para la regresión, el algoritmo *K-NN* compara a los k vecinos más parecidos. Se tiene N pares de puntos $(x_1, y_1), \ldots, (x_N, y_N)$ y un punto x_q para estimar, el objetivo es buscar los k puntos más cercanos a x_q, y se promedian sus etiquetas, Ec. (5.12).

$$\hat{Y_q} = \frac{1}{k}(y_{NN1} + y_{NN2} + \ldots + y_{NNk}) = \frac{1}{k}\sum_{j=1}^{k} y_{NNj} \tag{5.12}$$

Se ordenan los k vecinos de acuerdo a la distancia creciente al punto x_q. El nuevo reordenamiento queda: $x_{NN1}, x_{NN2}, \ldots, x_{NNk}$ (donde x_{NN1} es el vecino más cercano a x_q). Para predecir el valor que le corresponde a x_q se toma el promedio de las etiquetas.

Algoritmo para la regresión usando k-NN

Entrada: dato x_q al cual se quiere predecir su valor (obtener $\hat{y}_q$)
Dado un conjunto de pares de puntos $T = (x_1, y_1), (x_2, y_2), ..., (x_N, y_N)$, definir:
$S = (x_{NN1}, x_{NN2}, \ldots, x_{NNk})$ conjunto de los k puntos más cercanos a x_q
Para todo x_i tal que pertenece a T y no pertenece a S se cumple:
$distancia(x_i, x_q) \geq distancia(x_{NNk}, x_q)$

Predicción: $\hat{y}_q = (1/k)(y_{NN1}, y_{NN2}, \ldots, y_{NNk}) = (1/k)\sum_{j=1}^{k} y_{NNj}$

Una limitación que presenta el método es la discontinuidad, debido a que entre valores puede haber saltos y un vecino cercano puede estar o no en una de las regiones. Esta situación puede provocar el mencionado salto en la predicción. Las discontinuidades pueden ser o no significativas. Este problema se puede resolver modificando el algoritmo $k - NN$ agregándole pesos a los puntos.

Algoritmo k-NN ponderado

Esta técnica considera que las observaciones del conjunto de entrenamiento tienen pesos asociados. Sea w_i el peso para (x_i, y_i), entonces la etiqueta para x_q se obtiene mediante la fórmula de la Ec. (5.13).

$$\hat{y_q} = \frac{\sum_{i=1}^{k} w_i y_i}{\sum_{i=1}^{k} w_i} \tag{5.13}$$

En este caso, el ajuste local es definido alrededor de cada una de las observaciones, siendo la asignación del valor de salida, dependiente de la menor distancia de la entrada a los puntos disponibles (tomando por ejemplo la inversa de la distancia como peso). De esta forma los vecinos más alejados contribuyen menos en la clasificación.

Elección del parámetro K en el método.

Si no se dispone de información previa sobre el valor de k, su elección es difícil de ser realizada [10]. Se puede aplicar el método de validación cruzada para determinarlo.

Regresión al núcleo

La regresión al núcleo proporciona una forma de construir una función continua y genérica que decrezca conforme aumenta la distancia. En lugar de utilizar la inversa de la distancia como peso, se puede construir la función mediante una función núcleo o *kernel*. A tales efectos se define $C_{qNNj} = kernel_{\lambda}(distancia(x_{NNj},\ x_q))$

La función *kernel* puede construirse, por ejemplo, con una distribución normal donde $\lambda = \sigma^2$ (Ec. (5.14))[4].

$$Peso(x) = e^{\frac{-(x-q)^2}{\sigma^2}} \tag{5.14}$$

El peso de un punto x, está dado por la Ec. (5.14), siendo σ la desviación estándar de la distribución y q la media. Los puntos que están a una distancia σ de q se consideran fuertemente pesados, siendo los más alejados, de menor peso [4]. La función de predicción $\hat{y_q}$ para un punto x_q , basándose en la Ec. (5.12) está dada en la Ec. (5.15)

$$\hat{y_q} = \frac{\sum_{i=1}^{N} C_{qi} Y_i}{\sum_{i=1}^{N} C_{qi}} = \frac{\sum_{i=1}^{N} Kernel_\lambda(distancia(x_i, x_q)).Y_i}{\sum_{i=1}^{N} Kernel_\lambda(distancia(x_i, x_q))} \tag{5.15}$$

La ventaja de utilizar una función *kernel*, que es continua, es que está definida para todos los puntos del conjunto de entrenamiento y podría esperarse que la función entrenada también sea continua [4]. Para obtener una buena elección del parámetro λ, se puede emplear, el método de validación cruzada.

5.4.1. K-NN en Big Data

La performance de la regresión por $k - NN$ es variable, dependiendo de que haya disponibles muchos datos, y también de que las dimensiones no sean muchas, ya que es sensible a que los datos cubran el espacio de entradas.

Si se aborda un problema con un número de dimensiones grande, se requerirá un número exponencial en la dimensión de datos, para poder cubrir el espacio y tener una buena performance en la regresión $k - NN$ $(O(exp(d)))$. En estos casos, es conveniente utilizar métodos paramétricos. Tampoco es adecuado el uso del método de $k - NN$, en contextos de grandes volúmenes de datos y altas dimensiones. Además si N es muy grande, el problema es computacionalmente costoso. Algunas técnicas para tratar los problemas de grandes conjuntos de datos, incluyen *KD-trees, R-trees y Quad Trees* [4].

En el caso de tratarse de un número considerable de dimensiones los resultados no son efectivos, debido a la maldición de la dimensionalidad. Generalmente, se utilizan técnicas para reducir la dimensión como ser las tratadas en la *Sección 4.2* aplicadas sobre atributos de los datos y, más específicamente, los archivos VA [4].

Archivos VA (aproximación de vectores)

En espacios de altas dimensiones, para acelerar la búsqueda lineal se guar-

dan dos archivos: en uno de ellos se guardan los vectores originales y en el otro una representación resumida (geométrica) de ellos. Al buscarse los vecinos, inicialmente se escanea dentro de sus representaciones geométricas y si hay coincidencia se busca en el otro archivo. De esta forma se ahorra tiempo en la ejecución [4].

LSH

Es posible aplicar *LSH* para el método de vecinos más cercanos. Suponer un espacio compuesto por documentos donde se pretende clasificar uno nuevo con aquel que sea igual o más parecido. A tal fin se considera que cada palabra es una dimensión en el espacio. La idea es tomar planos aleatoriamente que pasen por el origen y a cada observación (documento) se le asigna un valor 1 o 0 si se encuentra en un semiplano o en otro. Si se tratase de un espacio de dos dimensiones con n puntos dato y k rectas que pasan por el origen, a cada uno de los puntos, se le asigna el valor 1 o 0 de acuerdo al semiplano donde se encuentre (Figura 5.7, imagen A). Cada recta asigna un valor a cada punto, creándose una codificación en *bits* para cada observación. Estos códigos son indexados en una tabla de *hash*, interesando que los documentos similares tengan los mismos códigos para efectuar búsquedas rápidas. Los puntos que están muy juntos en el espacio (por ejemplo, que comparten palabras en común en el caso de los documentos) tienen una menor probabilidad de estar separados por una línea determinada que los que son menos similares entre sí (ya que están más separados en el espacio de características). Este procedimiento se puede repetir para todos los puntos obteniéndose n firmas. Los puntos que están muy juntos en el espacio, con alta probabilidad, tendrán los mismos códigos de corte. Puede ocurrir que puntos muy distintos tengan igual codificación (falsos positivos), En la imagen A de la Figura 5.7, los puntos A y B tienen igual codificación pero son bastante distantes). Este tipo de problemas se puede solucionar utilizando más rectas, obteniéndose codificaciones diferentes para los puntos A y B.

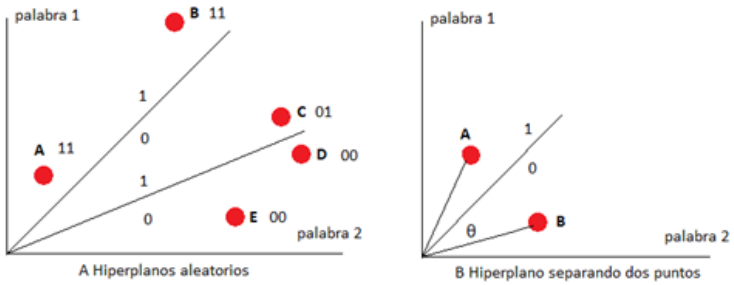

Figura 5.7: LSH

Las observaciones similares tendrán el mismo código de *hash*, colisionando en la misma cubeta. Al emprender una búsqueda, no es necesario hacerlo exhaustivamente a través de todo el conjunto de datos, sino solamente en los documentos que se encuentren en la misma cubeta.

Luego de realizado el proceso de división y codificación, éste se repite varias veces (L), aleatoriamente y, por lo tanto, se obtienen L particiones diferentes del espacio. Se espera que mediante el uso de diferentes particiones aleatorias, los puntos que están muy juntos y que están accidentalmente separados por una línea en una tabla *hash* (falsos negativos, caso de los puntos C y D en la imagen A de la Figura 5.7), no están separados en otra tabla *hash*. Posteriormente, para asegurar la calidad de los vecinos devueltos, típicamente todos los puntos que colisionan se comparan con el documento de consulta utilizando una métrica de similaridad (por ejemplo, similitud de coseno) y se descartan aquellos debajo de un umbral.

Además de L, la cantidad K de líneas (que también es la longitud del código *hash*) es el otro parámetro *LSH* importante. El aumento de k reduce la tasa de falsos positivos (intuitivamente, cuantas más líneas, es menos probable que puntos distantes, caigan en el mismo lado de todas las K líneas). La configuración adecuada de los parámetros K y L es específica de la aplicación y su configuración puede afectar la precisión y la velocidad de una implementación práctica de *LSH*.

Las ventajas de *LSH* son la velocidad de indexación (sólo particiona al azar su espacio de características y toma el producto de cada punto con los vectores normales de hiperplano para obtener los códigos del *hash*). Las desventajas incluyen la memoria requerida para contener las tablas de *hash*, y la precisión potencialmente menor frente a esquemas de partición más controlados

Costo computacional de LSH (determinación de la cantidad de hiperplanos)

Sea n el número de documentos, d el promedio de la longitud de ellos (cantidad de palabras) y k el número de hiperplanos que definen 2^k regiones (*hashcodes*). El costo promedio para ubicar la cubeta correspondiente a un documento es $d.k$ (para cada observación x e hiperplano H_i se debe efectuar el producto: $x^T H_i$). En promedio, el número de colisiones en el *hash* será: $n/2^k$. El costo para comparar el documento x en el *hash* es $d.n/2^k$. Por lo tanto, encontrar a la cubeta y efectuar las comparaciones dentro de ella tiene un costo de $d.k + (d.n)/2^k$. Si n es grande conviene que k sea una función del $\log n$. De esta forma el costo de la comparación es constante y finalmente el costo total será $d\log(n) + cte$, es decir $O(log(n))$.

Errores en LSH

Como se había señalado, puede ocurrir que puntos próximos no caigan en la misma cubeta (falsos negativos) o que puntos alejados coincidan con el mismo *hashcode* (falsos positivos).

El caso de falsos negativos se interpreta como que, dados dos documentos similares a y b, éstos nunca coinciden en la misma cubeta en ninguno de los L procesos. Por lo tanto, interesa calcular la probabilidad de que ningún *hashcode* para los documentos a y b, obtenidos en los L procesos, coincidan. Los procesos son independientes (cada conjunto de hiperplanos fue elegido aleatoriamente) e idénticamente distribuidos (es el mismo proceso aleatorio) por lo tanto la probabilidad buscada es el producto de la probabilidad en cada proceso y éstas son iguales (Ec. (5.16)).

$$P(\textit{ningún hashcode de a y b coincidan en los L procesos}) = \\ = P(hashcode(a) \neq hashcode(b))^L \quad (5.16)$$

Los *hashcode* son diferentes si al menos difieren en un *bit* o lo que es equivalente a 1 menos la probabilidad de que tengan todos los *bits* iguales. Como estos bits se construyeron en base a la aleatoriedad de los hiperplanos, los k bits de cada *hashcode* son independientes e idénticamente distribuidos (Ec. (5.17)).

$$P(hashcode(a) \neq hashcode(b))^L = \\ (1 - P(todos\ los\ bits\ son\ iguales))^L = (1 - p^k)^L \quad (5.17)$$

La probabilidad p representa que los documentos a y b tengan un *bit* igual.

Para los falsos positivos se realiza un razonamiento similar considerando la probabilidad de que las observaciones a y b tienen al menos un *hashcode* igual en los L procesos.

Para calcular la probabilidad de que un hiperplano separe a dos documentos a y b, se considera que ellos se encuentran en una híper esfera unitaria formando un ángulo θ con el origen (imagen B de la Figura 5.7) y que los planos pueden elegirse aleatoriamente en el primer cuadrante ($\frac{\pi}{2}$), obteniéndose la probabilidad $\frac{\theta}{\pi/2}$. El ángulo θ verifica $\theta = \arccos(A^T B)$ (ya que son vectores unitarios). La probabilidad para que coincidan es: $1 - (2/\pi)\arccos(A^T B)$.

5.4.2. Resumen

El método $k - NN$ es usado para la clasificación y la regresión. El método presenta limitaciones de performance cuando son muchas las dimensiones de las entradas o muchos los datos.

Para la regresión el método consiste en buscar a los k vecinos más próximos (parecidos) a la nueva observación y se hace un promedio para estimar el valor que le corresponde. Para la clasificación, se devuelve la clase mayoritaria entre los k vecinos más próximos.

Para altas dimensiones o grandes cantidades de datos existen otros algoritmos (*VA, KD-Tree, LSH*) que resuelven las limitaciones de $K - NN$.

5.5. Árboles de Decisión

En algunos casos, construir un modelo lineal de predicción con un gran número de variables de entrada, no es una buena elección, por lo que es preferible optar por un árbol de decisión (AD), el cual pueda resolver las interacciones de una gran cantidad de variables. Los AD son modelos estadísticos que son considerados métodos de predicción supervisados y son utilizados en la clasificación, regresión, agrupamiento, estimación de probabilidades, etc. A

estos métodos se les denominan *Tree-Based Methods* [10].

5.5.1. Construcción del árbol

A partir de un conjunto de observaciones se considera un umbral o categorización en las características de los datos. De acuerdo a la respuesta al recorrer cada característica se toma una decisión y de esta forma se construye el árbol. Habitualmente, se utilizan los procedimientos de validación cruzada, a los efectos de evitar sobreajustes.

A continuación presentamos el *Ejemplo 5.11* que se va procesando conforme avanza el desarrollo de esta sección.

Ejemplo 5.11

Se considera un grupo de escolares, de los cuales se tiene la calificación del año anterior, en tres categorías (medio, malo y excelente), la cantidad de horas promedio que estudian diariamente en dos categorías (poco y mucho), el estrato social al que pertenecen (alto y bajo) y etiquetas asociadas a cada uno que indican si el escolar está con riesgo de perder el año lectivo actual (Si/No). Se intenta predecir la etiqueta que le correspondería a una nueva observación.

En este caso el AD es un modelo de clasificación que indica en base a los datos anteriores a qué clase pertenece el estudiante (con riesgo o sin riesgo). En la Tabla 5.20 se muestran algunos puntos dato como *set* de entrenamiento.

Calificación (Nota)	Horas	Estrato	Riesgo (Si/No)
Excelente	Mucho	Alto	No
Bueno	Poco	Alto	Si
Excelente	Mucho	Bajo	No
Malo	Poco	Medio	Si
Bueno	Poco	Medio	No
Malo	Mucho	Medio	No
Bueno	Mucho	Bajo	No

Tabla 5.20: Datos para construir un árbol de decisión

Dada una entrada x, el AD permite decidir cuál es la clase que le corresponde; para ello, se atraviesa el árbol y se toma una determinada predicción $\hat{y}$.

Para los datos de la Tabla 5.20, la técnica pretende obtener la representación de un AD, como el de la Figura 5.8.

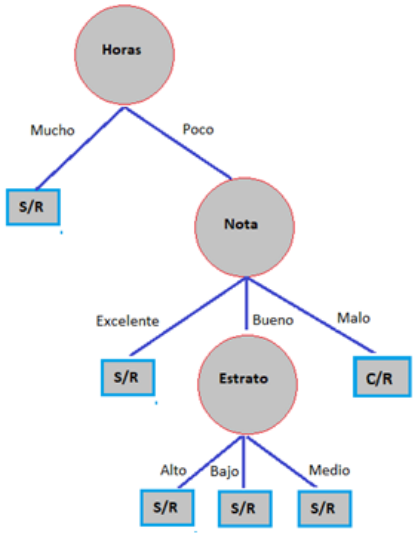

Figura 5.8: Construcción del AD

Dados los datos, se trata de entrenar al *AD*. La calidad del entrenamiento se basa en la métrica utilizada. Para el ejemplo, se utiliza la razón de los errores sobre el total de predicciones (Ec. (5.18)), que es el error de clasificación (existen otras métricas).

$$Error = \frac{(\#predicciones\ incorrectas)}{(\#muestras)} \tag{5.18}$$

Si no hay errores, el valor es cero y, en el caso extremo en que todas las predicciones son erróneas, se obtiene un valor 1. Se selecciona la que posee menor error de clasificación.

Entropía y Ganancia de información

En lugar de utilizar una métrica basada en la razón de los errores sobre el total de observaciones, se utilizan conceptos basados en la entropía de la teoría de la información. El algoritmo clásico de este tipo es denominado *ID3*. Es usado para ayudar a decidir qué atributo debe ser el siguiente en seleccionarse para dividir el árbol. Para tales efectos se selecciona el atributo que tiene mayor ganancia de información. En general, un atributo que discrimina más objetos, tiende a reducir más la entropía y debería ser seleccionado en el nodo para efectuar la siguiente división.

Sea un conjunto de entrenamiento, D con m diferentes etiquetas (que definen m clases C_i para la clasificación) y sea $C_{i,D}$ el conjunto de observaciones

de la clase C_i en D. Para el cálculo de la entropía necesaria para clasificar una observación en D se utiliza la Ec. (5.19).

$$s(D) = -\sum_{i=1}^{m} p_i \log_2(p_i) \tag{5.19}$$

p_i es la probabilidad que una observación en D pertenezca a la clase C_i y se calcula por medio de $\frac{|c_{i,D}|}{|D|}$. El valor $s(D)$ expresa la cantidad de información promediada necesaria para identificar la etiqueta de la clase para una observación en D. Si en el cálculo de la entropía se presentara una probabilidad 0 , entonces se considera la expresión $0 \log_2 0$ igual a 0. A continuación se considera la partición de los datos de D sobre un atributo A que tiene v valores distintos (para cada uno de ellos hay una determinada cantidad de observaciones). Cada atributo A, permite dividir a D en v subconjuntos. Para la clasificación se necesita calcular la Ec. (5.20).

$$S_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} . s(D_j) \tag{5.20}$$

El término $\frac{|D_j|}{|D|}$ es el peso de la clase j sobre todos las observaciones en D.

$s(D_j)$ es la entropía requerida para clasificar una observación de D en base al atributo A. Finalmente se utiliza el concepto de ganancia de información, por medio de la diferencia entre la entropía original y la de cada atributo. Se debe evaluar la ganancia para cada uno de ellos y el que obtenga el mayor valor es seleccionado para realizar la división del árbol [10]. La ganancia de información para un atributo A se calcula utilizando la fórmula $G(A) = s(D)–s_A(D)$

Otros índices similares basados en la probabilidad son el *Gain Ratio* y el *Gini Index* [10]. En el *Anexo 8*, se presenta un ejemplo de construcción de un *AD* usando la entropía.

Orden para la selección de las características

Para seleccionar la característica más adecuada para hacer la primer división, se puede optar por la que presenta menor error de clasificación. Continuando con el ejemplo de la Tabla 5.20, se desarrolla la construcción del *AD* correspondiente. Para ello, sería conveniente estimar el error en cada una de las características. Si se dispone de información estadística de que el criterio para al atributo «Calificación», es que los que tuvieron nota «Bueno» o «Excelente» no tienen riesgo de perder el año y los que tuvieron nota «Malo» sí lo tienen, se

pueden proyectar las columnas Calificación y Riesgo para determinar el error de acuerdo con la Ec. (5.18).

Se observa que para la franja «Excelente» existen dos casos clasificados correctamente, es decir, no hay error. Para la franja «Bueno» existe un caso riesgoso, que constituye un error. Para la franja «Malo» se encuentra un error. En total se tiene dos errores en siete observaciones para la característica «Calificación».

Para la característica «Horas», se observa que los estudiantes de la franja «Mucho» no son riesgosos y los de la franja «Poco», si. En cuanto al estrato social se tiene conocimiento a priori que las clases «Alto» y «Medio» no son riesgosos, pero «Bajo» sí lo es. Si se toma en consideración la característica «Horas» (considerando que dedicar pocas horas de estudio es riesgoso de perder el curso y muchas no) se observa que no hay errores para la franja «Mucho» y para la franja «Poco» se encuentra 1 error. Finalmente la característica «Estrato» (considerando que los de «Alto» y «Medio» no son riesgosos) se tiene; «Alto» un error, «Medio» un error y «Bajo» dos errores, totalizando cuatro. Se procede de igual manera para el resto de los atributos observándose que Horas es el atributo que posee menor error (Figura 5.9).

Atributo	Valor	#erróneos	*Error*
Calificación (Nota)	Excelente	o	2/7
	Bueno	1	
	Malo	1	
Horas	Mucho	0	1/7
	Poco	1	
Estrato	Bajo	2	4/7
	Medio	1	
	Alto	1	

Figura 5.9: Cálculo del error en los atributos de un árbol de decisión

Algoritmo voraz de predicción para un AD

Paso 1: Árbol vacío

Paso 2: Dado un subconjunto S en un nodo del árbol, Seleccionar una característica para dividir los datos

Para cada característica $h_i(x)$:

Dividir S de acuerdo a $h_i(x)$

Calcular el error de clasificación de la división

Paso 3: Elegir la característica $h^*(x)$ con el menor error de clasificación para hacer la división.

Paso 4: Para cada división del árbol:

Si no se encuentran más características (el nodo es una hoja), hacer la predicción con la clase mayoritaria

Si no, dirigirse al Paso 2 y continuar recursivamente la división

Criterios de parada

- Si todos los puntos están bien clasificados, el nodo al cual llegan, se convierte en hoja. Luego el algoritmo continúa con los demás nodos, con una nueva característica.
- El segundo criterio para detenerse es cuando no quedan más características para decidir.

El algoritmo voraz indica que la división por *Horas* es la que presenta el menor error y se comenzaría la división del árbol por esa característica.

Si la característica *Horas* tiene el valor *Mucho*, es clase mayoritaria y todos los casos no son riesgosos, por lo tanto, hay que discriminar para el caso Horas = *Poco* (Tabla 5.21).

Calificación (Nota)	Horas	Estrato	Riesgo (Si/No)
Bueno	Poco	Alto	Si
Malo	Poco	Medio	Si
Bueno	Poco	Medio	No

Tabla 5.21: Datos para Horas = *Poco*

En la Tabla 5.22 se presentan los errores para este caso.

Atributo	Valor	#erróneos	Error
Calificación (Nota)	Excelente	0	1/3
	Bueno	1	
	Malo	0	
Estrato	Bajo	0	2/3
	Medio	1	
	Alto	1	

Tabla 5.22: Errores en los datos para Horas = *Poco*

El atributo *Calificación* es el que posee menor error de clasificación por lo tanto será la siguiente característica para hacer la separación del *AD*. Para los valores *Excelente* y *Malo*, se observa que son clase mayoritaria, debiéndose realizar la división para la siguiente característica con el valor *Bueno*. Procediendo de igual forma, se obtiene, para la característica *Estrato* (discriminando para Calificación = *Bueno* y Horas = *Poco*), los valores en la Tabla 5.23.

Calificación (Nota)	Horas	Estrato	Riesgo (Si/No)
Bueno	Poco	Alto	Si
Bueno	Poco	Medio	No

Tabla 5.23: Datos para Horas = *Poco* y Calificación = *Bueno*

En este caso, no hay clase mayoritaria pudiendo ser cualquiera de las dos etiquetas.

Existen árboles de decisión cuya salida es multiclase. Por ejemplo, en la Tabla 5.20, se puede agregar en la columna «Riesgo (si/no)», una tercera posibilidad como ser «descartado de continuar estudiando».

En el *Anexo 8* se presenta la construcción de un *AD* con salida multiclase utilizando el concepto de ganancia de información.

Entradas numéricas

En este caso es conveniente separar en regiones al espacio de las entradas que no son categóricas. Si una característica tiene valores numéricos, por ejemplo la edad, se puede considerar un umbral de corte en un valor determinado, de la Figura 5.10.

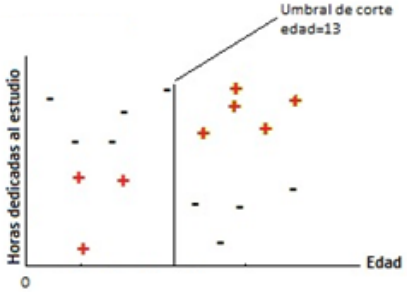

Figura 5.10: AD para entradas continuas (edad)

Al momento de determinar el umbral t^*, para separar *inputs* numéricos, basta observar que dadas dos observaciones consecutivas, si se divide de acuerdo al punto medio de ellas, el error de clasificación no se modifica, ya que

los puntos a la derecha del corte y a la izquierda son los mismos que antes del corte, por lo que sólo se necesita considerar los puntos medios entre las observaciones disponibles para realizar las divisiones.

Los *AD*, al igual que la regresión logística, tienen diferentes posibilidades, debiéndose optar por la técnica más conveniente. Por ejemplo, en la regresión logística, es posible considerar líneas oblicuas como fronteras, mientras que en los *AD* no es posible, ya que solo se pueden realizar cortes paralelos a los ejes.

Algoritmos voraces para la construcción del AD

Ambos métodos para la construcción del *AD* (minimización del error o ganancia de la información) son inviables si se procesa una gran cantidad de datos o atributos. Analizar los múltiples escenarios es un problema *np-hard*, por lo que una solución puede ser el empleo de un algoritmo voraz recursivo. Se debe definir el criterio para particionar el árbol (en función de las características) y definir la condición de parada.

Escalabilidad

Si el *dataset* es tan grande que no entra en memoria existen variaciones como *Rain Forest* y *BOAT* [10].

5.5.2. Tratamiento de los datos faltantes

El impacto de los datos faltantes en un *AD* puede ocurrir durante el entrenamiento o al momento de la predicción (pueden faltar datos en el conjunto de entrenamiento, así como datos del *input* al momento de la predicción). En este caso se pueden utilizar las estrategias nombradas en el *Capítulo 2*.

Los algoritmos empleados por los *AD* deben ser lo suficientemente robustos, como para adaptarse a los datos faltantes. Para ello, en cada nodo, se considera la opción de que en caso de no existir el dato, el algoritmo indica cuál rama se debe seguir. Un criterio podría ser elegir la rama que presenta menor error en la clasificación. En este caso, sería necesario modificar el paso 2 del algoritmo voraz. Este enfoque puede utilizarse en la etapa de entrenamiento y en la predicción, dado que mejora la calidad del modelo.

En el *Anexo 8* se desarrollan los conceptos de entrenamiento, podas y evaluación de los árboles de decisión.

5.5.3. Resumen

Los árboles de decisión permiten resolver las interacciones de una gran cantidad de variables (características), las cuales pueden ser numéricas o categóricas. Las predicciones pueden realizarse por clase mayoritaria, predicciones con probabilidad y clasificación multiclase.

Los *AD* son proclives al sobreajuste, lo cual se ve reflejado al momento de calcularse el error de entrenamiento, que tiende a cero, cuando los modelos se tornan más complejos. El sobreajuste puede solucionarse por medio de podas.

El empleo de los *AD* para la regresión es por medio de constantes asociadas a las diferentes regiones que las condiciones definen. Recorriendo el árbol, se busca a qué región pertenecen las variables de entrada y se asigna el valor de la constante correspondiente como valor de la salida [20].

Caso de estudio

Se propone, para modelos con muchas características, construir muchos árboles de pocas características y poca profundidad, y luego ensamblarlos, utilizando *MapReduce*.

Una posible solución, a profundizar, consistiría en que la función *Map* agrupara de acuerdo a un atributo (característica) y la función *Reduce* debería indicar cómo se construye el sub árbol considerando la salida de los diversos *Maps*. Luego se vuelve a repetir el ciclo *MapReduce* hasta alcanzar una determinada condición de parada.

5.6. Combinación de clasificadores

A los efectos de mejorar la precisión de los clasificadores se pueden combinar varios de ellos. En general, el resultado obtenido por la combinación es mejor que los devueltos por los clasificadores individualmente. La forma de combinar las clasificaciones puede hacerse por voto mayoritario. La calidad del ensamble de los clasificadores depende de la precisión de ellos y de su diversidad [3]. Si son parecidos, posiblemente cometan los mismos errores y, por votación mayoritaria, la clasificación sería incorrecta. En cambio, si son diferentes, si uno de ellos incurre en un error, es de esperar que los otros no lo cometan y, por voto de la mayoría, se efectiviza una clasificación correcta.

Este problema de aprendizaje comienza con un conjunto de datos y una etiqueta asociada $+1$ y -1 a cada uno (aunque también es aplicable a multi clases). Luego se entrenan los clasificadores: $f_1(x), f_2(x), \ldots, f_T(x)$. Para todos estos algoritmos es necesario determinar cómo realizar la combinación del conjunto de sus salidas. Esta es la idea del aprendizaje basado en conjuntos de modelos (ensamble de modelos). Dado un conjunto de pesos para las salidas $w_1, \ldots, w_n$ y los *outputs* $f_1, \ldots, f_n$, se multiplican los *outputs* por los pesos, se suman y se obtiene el signo. Si es positivo la salida es $+1$, si no es positivo, -1 (se trata de un esquema ponderado de votos).

A continuación se describen los métodos de *Bagging* y *Boosting* y, como un caso particular de éste, *AdaBoost*.

Bagging (Bootstrap aggregating)

El clasificador *Bagging* es un ejemplo de ensamblador. Consiste en elegir N observaciones de entrenamiento uniformemente, con reemplazo, para el aprendizaje de un modelo. Dicho proceso se repite T veces, obteniéndose T modelos. Al momento de clasificar una nueva observación, ésta es pasada por los modelos obtenidos y la clase mayoritaria indica cual es la clasificación que le corresponde [10].

Boosting

El algoritmo de *Boosting* se originó a partir de una conjetura matemática de *Kearns and Valiant* en 1988, la cual presenta la siguiente interrogante: «Se pueden combinar clasificadores débiles para obtener mejores algoritmos entrenados?». En 1990 *Schapire* demuestra que sí es posible. A partir de esa fecha se desarrolló el algoritmo de *Boosting* el cual utiliza una combinación de clasificadores débiles mediante un esquema de votos, con el fin de que el error de clasificación sea menor [24].

Un clasificador fuerte es aquel cuyo error en la clasificación es aceptable, mientras que uno débil presenta una performance que es apenas mejor que una clasificación aleatoria.

La motivación de la técnica es que si una observación es mal clasificada por un método de aprendizaje, debe obtener la misma clasificación, por parte de, al menos, la mitad de los clasificadores. Generalmente, es de esperar que la situación de mala clasificación sea la menos probable, es decir, que pocos

modelos clasifiquen mal una observación. Si cada modelo tiene un error p, se considera a dicho valor como la probabilidad de que una observación x, elegida aleatoriamente, sea mal clasificada. Además, se supone que los errores cometidos por cada modelo son independientes, por lo que, si p es pequeño, la probabilidad de que se produzca un gran número de errores de clasificación es aún menor. Dicha suposición de independencia es discutible, ya que algunos modelos pueden ser engañosos. No obstante, haciendo los modelos independientes (disminuyendo la correlación entre sus errores) la técnica mencionada funciona correctamente.

El mecanismo para construir los componentes del conjunto utilizado por *Boosting*, se basa en la asignación de un peso a cada muestra del conjunto de entrenamiento. Es decir, cada observación lleva asociada un peso $w \geq 0$. Cuanto mayor es el peso, mayor es su importancia en el aprendizaje del modelo. En cada iteración, se entrena un modelo que minimiza la suma de los pesos de las muestras clasificadas erróneamente [25]. Los errores que aparecen en cada iteración actualizan los pesos de los elementos de la muestra, incrementando el peso de las que dieron error y reduciendo el peso de los acertados. De esta manera, se consigue que el modelo que se entrena en la siguiente iteración, otorgue mayor relevancia a los ejemplos clasificados en forma errónea por los modelos anteriores. El algoritmo construye los nuevos modelos, tratando de corregir los errores cometidos anteriormente, asignando para la predicción, mayor relevancia a los modelos que tienen mejor comportamiento.

AdaBoost

AdaBoost (*Adaptive Boosting*) es un algoritmo de *Boosting*, elaborado por *Freund and Schapire* en1999 cuyo objetivo es obtener un modelo optimizado de predicción, que permita encontrar clasificadores que mejoren el modo de obtener una solución óptima. Posee la propiedad de que si el algoritmo de aprendizaje de entrada es débil, el *Adaboost* devuelve un modelo que clasifica con una gran precisión a los datos de entrenamiento, para un modelo M_n obtenido con un n suficientemente grande.

Se trata de un algoritmo que inicializa los N datos con un peso uniforme $\frac{1}{N}$.

Estos pesos se actualizan en cada iteración, según las observaciones mal clasificadas. Las iteraciones consisten en considerar un nuevo modelo que extrae una cantidad de datos con reemplazo. Cada observación tiene un peso que se

actualiza de acuerdo al error y la probabilidad de ser seleccionada depende de su peso. Si la muestra seleccionada es clasificada mal su peso aumenta y viceversa. De este modo los nuevos modelos muy probablemente vuelven a clasificar a los datos que fueron mal clasificados en los modelos previos

En el *Anexo 8*, se describe el proceso de entrenamiento de *AdaBoost* y las condiciones en que puede ocurrir sobreajuste.

5.6.1. Resumen

El método de aprendizaje por ensamble considera una selección de modelos y combina sus predicciones. Se ha comprobado experimentalmente que la técnica de *Boosting* presenta una mejora respecto a los clasificadores que actúan en forma aislada.

La idea principal del método es que si se tienen m modelos, cada uno de ellos con una probabilidad p de error y, asumiendo que éstos son independientes, el error se vuelve más pequeño (considerando el producto de ellos).

Un ejemplo de estos algoritmos es *AdaBoost*, cuyo objetivo es obtener un modelo optimizado de predicción, que permita encontrar clasificadores que mejoran el modo de obtener una solución óptima.

Caso de estudio

El ensamblaje de modelos se realiza sobre aquellos que son paramétricos con aprendizaje. Se pretende trasladar la idea para los tareas descriptivas. Por ejemplo, las técnicas de *clustering* son realizadas directamente por un algoritmo, el cual indica los clústeres que se forman y dónde se agrupan los datos. Para este caso particular, el objetivo sería adaptar la idea a una búsqueda optimizada de clústeres. La utilidad de tal técnica podría reflejarse en mejoras en las etapas iniciales de identificación de los agrupamientos, proporcionando como salida los centroides iniciales o la cantidad de clústeres que podrían considerarse.

5.7. Redes Neuronales Artificiales

Las Redes Neuronales Artificiales (*RNA*) son algoritmos de aprendizaje automático formado por unidades de entrada/salida conectadas donde cada conexión tiene asociado un peso. En esta sección se describe brevemente la

teoría de las redes neuronales artificiales (*RNA*). Se incluyen dos ejemplos que motivan su empleo, su estructura, forma de funcionamiento y aplicaciones para la tarea de clasificación y para el área de *deep learning*.

Ejemplo 5.12

Sea un problema de clasificación lineal, con muchas variables no lineales (por ejemplo $y = w_0 + w_1 x_1^3 + w_2 x_1 x_2 + ...$ Este problema puede ser resuelto con la función sigmoidea, vista en la *Sección 5.3.2*. No obstante, incluso con pocas características, al considerar los términos cuadráticos, la cantidad de éstos son $\frac{N^2}{2}$ (por ejemplo, para el caso en que hay 100 características se alcanzan 5000 variables de grado 2 y si son términos cúbicos serían 170000). Esta cantidad tan grande de términos puede producir modelos propensos a sobreajuste. Este tipo de situaciones son comunes en clasificadores para el reconocimiento de imágenes (visión computarizada) debido a que, cuando un detector mira a un objeto, genera una matriz de valores de intensidades que indican el brillo de cada pixel de la imagen. Al entrenar a una *ML* para este reconocimiento, sería conveniente disponer de un conjunto de entrenamiento de muestras etiquetadas como positivas (concuerdan con el objeto a reconocer) y negativos (que no lo hacen).

Para tener una idea de la dimensión del espacio de las características, si se tienen imágenes pequeñas de 50×50 pixeles, utilizando el código de colores *RGB*, se necesitarían 7500 dimensiones. En el caso de las escalas de grises, cada una de las cuáles tiene una intensidad entre *0 y 255*, para cada muestra de entrenamiento, debe considerarse, aproximadamente, tres millones de dimensiones. Por este motivo, los métodos vistos hasta el momento, no son aconsejables para tratar los problemas indicados. En cambio, las *RNA* son más apropiadas para el entrenamiento de modelos que contienen muchas características y grandes cantidades de datos.

Ejemplo 5.13

La Figura 5.11. representa un clasificador lineal en la forma de una red neuronal de una capa, donde la salida representa el *score* obtenido a partir de los coeficientes de las entradas.

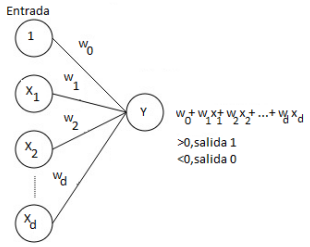

Figura 5.11: Representación de un clasificador lineal con una red neuronal

No obstante, no todas las funciones pueden representarse con un clasificador lineal. Por ejemplo un clasificador lineal no puede representar la función *XoR* (Figura 5.12). Resulta imposible obtener una función lineal que separe en regiones los unos de los ceros, por lo que se necesitan características no lineales para hallar un separador (es un problema similar al ejemplo anterior, en el cual no se puede establecer fronteras lineales para separar imágenes).

1	**0**
0	**1**

Figura 5.12: XoR

Las *RNA* son buenas en el reconocimiento de patrones y construyendo reglas simples para problemas complejos. A su vez tienen una gran capacidad de entrenamiento por lo que son utilizadas en investigaciones, en el campo de la inteligencia artificial. A su vez, presentan propiedades importantes tales como el desarrollo de computación distribuida, tolerancia al ruido en la entrada y aprendizaje automático [26].

Representación

Una *RNA* es un grafo dirigido donde los nodos se llaman unidades y las aristas conexiones. Cada conexión tiene un peso asociado w_{ij} que determina la intensidad y el sentido de la conexión. Cada unidad i calcula la suma ponderada de sus entradas (Ec. (5.21))

$$in(i) = \sum_{j=0}^{N} w_{j,i} a_j \tag{5.21}$$

Luego, se aplica una función de activación a la suma para calcular la salida (Ec. (5.22)).

$$a_i = g(in(i)) \tag{5.22}$$

La Ec. (5.23) define matemáticamente a una neurona artificial.

$$y(x) = g(\sum_{i=0}^{n} w_i x_i) \tag{5.23}$$

Una neurona se caracteriza por sus valores de entradas, n *inputs* $(x_0, x_1, \ldots, x_n)$, un conjunto de pesos asociados, $(w_0, w_1, \ldots, w_n)$ y una salida $y(x)$. La función g se denomina función de activación e indica el valor del *output* de la neurona. Para las *RNA* es preferible tener una función suave y diferenciable de salida la cual, generalmente, retorna 0 o 1 o sino -1 y +1 (sin embargo, la función identidad, que a veces es utilizada como activadora, no cumple esas restricciones). Los *inputs* y los pesos pueden tomar cualquier valor real, aunque habitualmente varían en torno a valores alrededor de 0.

Existen muchas funciones de activación: por umbral (Ec. (5.24)), sigmoidea (5.25)) y tangente hiperbólica (5.26)), entre otras.

$$g(x) = \begin{cases} 1 & \text{si } x + t > 0 \\ 0 & \text{si } x + t \leq 0 \end{cases} \tag{5.24}$$

$$g(x) = \frac{1}{1 + e^{-2s(x+t)}} \tag{5.25}$$

$$\begin{aligned} g(x) = \tanh(s(x+t)) = \frac{\sinh(s(x+t))}{\cosh(s(x+t))} = \\ = \frac{e^{s(x+t)} - e^{-s(x+t)}}{e^{s(x+t)} + e^{-s(x+t)}} = \frac{e^{2(s(x+t))} - 1}{e^{2(s(x+t))} + 1} \end{aligned} \tag{5.26}$$

Las dos últimas funciones son suaves y diferenciables y difieren, además, en que la primera varía su recorrido de *0 a 1* y la segunda de -1 *a* $+1$. El término w_0 es opcional y representa al sesgo que es agregado para aumentar la flexibilidad del modelo. En una red neuronal la capa uno corresponde a las neuronas de entrada y la última capa corresponde a las de salida. Las capas intermedias se denominan capas ocultas.

Estructura

Existen dos tipos de estructuras de redes neuronales: acíclicas o de pro-

pagación para adelante y cíclicas o redes recurrentes [26]. En estas últimas, sus salidas se alimentan de sus entradas, por lo que se trata de un sistema dinámico, que alcanzará un determinado estado al final de su entrenamiento (estable, oscilante o caótico) [27]. En el *Anexo 8* se presentan los métodos de propagación para adelante y de *back propagation*.

5.7.1. RNA en clasificación

La clasificación en las redes neuronales artificiales, se basa en el enfoque *1 vs todos*, analizado en la Sección 5.3, Clasificación. En la de la Figura 5.13 se ve una red neuronal cuya salida son cuatro posibles valores que representan las categorías de clasificación. La salida pertenece a R^4

Figura 5.13: Clasificación múltiple en RNA

Para una clasificación binaria basta con una neurona de salida. Para una clasificación multiclase con k clases se necesitan k unidades de salida.

Redes neuronales profundas y superficiales

Las redes neuronales profundas utilizan múltiples capas ocultas y las superficiales utilizan una sola capa oculta con muchas neuronas. Empíricamente se ha constatado, que, para problemas no lineales, es preferible utilizar las profundas ya que una superficial necesita de una gran cantidad de neuronas en su capa oculta, lo que haría más costoso calcular la salida (aparentemente, debido a que en cada una de esas neuronas se debería realizar la función de decisión, que incluye realizar una operación de multiplicación o suma en punto flotante). En cambio para las profundas, las operaciones realizadas en las capas bajas pueden considerarse como subrutinas, las cuales pueden ser reutilizadas en las capas siguientes. A su vez, desde un punto de vista de costo computacional, las profundas son menos costosas ya que las superficiales no reutilizan apropiadamente las subrutinas [28].

La aplicación de *Kernel Methods* en el área de *deep learning* es empleado cuando las redes superficiales utilizan una gran cantidad de neuronas, lo cual constituye un campo de investigación activo [28].

5.7.2. Aprendizaje profundo (deep learning)

El *deep learning* (aprendizaje profundo) utiliza la arquitectura de redes neuronales. Entre sus aplicaciones se destacan la visión computarizada, reconocimiento del habla y modelado del lenguaje.

Modelo para el aprendizaje profundo

Los desafíos principales se centran en que se necesitan muchos datos, los cuales deben estar etiquetados. Por lo tanto es computacionalmente costoso, y los parámetros difíciles de tunear (elección de la arquitectura, tipos de los parámetros, algoritmos de aprendizaje).

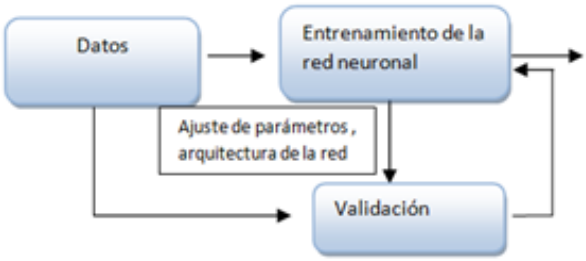

Figura 5.14: Diagrama de Flujo para aprendizaje profundo

En la de la Figura 5.14. el flujo comienza a partir de cierta cantidad limitada de datos etiquetados, de los cuáles se extraen características mediante una *RNA*. El conjunto se divide entre conjunto de entrenamiento y de validación.

En los últimos años se han hecho avances en algoritmos que han mejorado la performance de las *RNA*. Dos de ellos son: *Rectified linear Units* (*ReLUs*, unidades lineales rectificadas) y *dropout* (abandono). *ReLUs* mejora la velocidad de entrenamiento, optimizando el algoritmo y *dropout* mejora la calidad de la predicción resultando en una mayor generalización. *ReLUs* utiliza la función de activación representada por $g(z) = max(0, z)$, en lugar de la función sigmoidea.

El motivo por el cual *ReLUs* es más efectiva que la función sigmoidea es un tema abierto de investigación [28].

Red neuronal convolucional

Una red neuronal convolucional es una *RNA* que supone, explícitamente, que las entradas son imágenes, lo que permite codificar ciertas propiedades en la arquitectura, alcanzándose una mayor eficiencia con menos parámetros. Modelan de forma consecutiva pequeñas piezas de información que luego son combinadas en las capas más profundas. Por ejemplo, la primera capa intenta detectar los bordes y las capas posteriores tratan de combinarlos en formas más simples y luego se buscan patrones de las diferentes posiciones de los objetos, su iluminación y escalas de grises, entre otras cosas. Las capas finales, utilizando los patrones obtenidos, realizan una predicción efectuando una suma ponderada de ellos. Son muy efectivas para tareas de visión artificial en tareas de clasificación y segmentación de imágenes, entre otras aplicaciones. Una aplicación de este tipo de *RNA* son las *Fast R-CNN* (*Fast Region -based Convolutional Network*), que constituyen un clasificador de objetos basado en redes convolucionales y son utilizadas no solo para clasificación de imágenes, sino también para la clasificación de objetos. Está probada su mayor eficiencia y velocidad respecto a otros métodos [29].

5.7.3. Resumen

Una *RNA* es un grafo dirigido donde los nodos se llaman unidades y las aristas conexiones.

Cuando se trata de problemas con millones de características en los datos, los métodos empleados en la clasificación y regresión resultan insuficientes. Una alternativa para su solución es abordarlos bajo la óptica de redes neuronales artificiales. Su empleo, por lo tanto, es apropiado para el entrenamiento de modelos que contienen muchas características y grandes cantidades de datos.

Adicionalmente, las *RNA* presentan propiedades importantes tales como el desarrollo de la computación distribuida, tolerancia al ruido en la entrada y aprendizaje automático.

Para una explicación más detallada de las redes neuronales se puede consultar *(Hassoun, M. H., 1995). Fundamentals of Artificial Neural Networks*, de *MIT Press.* que presenta una amplia cobertura de los conceptos de las *RNA*) y *Hertz et al., 1991 Hertz, J., Krogh, A., and Palmer, R. G. (1991). Introduction to The Theory of Neural Computing*, de Addison-Wesley Publishing Company, que hace una descripción matemática profunda. En [28] se

encuentran consideraciones prácticas relacionadas con la implementación de una *RNA*.

En relación al diseño del aprendizaje profundo en el área de la visión se debería tener en consideración el *hardware*, la masividad de datos y las estrategias para su entrenamiento y arquitectura.

Caso de estudio

Aplicar *MapReduce* para implementaciones en paralelo de las *RNA* posibilitaría significativos ahorros de tiempo durante su entrenamiento.

Temas abiertos de investigación

ReLUS

Investigación de los motivos por los cuales *ReLUs* es más efectiva que la función sigmoidea.

Kernel Methods

Aplicaciones de *Kernel Methods* en el área de *deep learning* sobre redes superficiales que utilizan una gran cantidad de neuronas.

Deep Learning

Varios aspectos aún están abiertos para su estudio: fundamentar la teoría de fondo, su entrenamiento, evitar mínimos locales y diseño de arquitecturas generales

5.8. Clustering

El *clustering* o agrupamiento es un método de aprendizaje no supervisado descriptivo, para la identificación de grupos de datos similares, en base a una función de distancia. El objetivo es que la similaridad entre los puntos del clúster sea alta, pero baja entre los agrupamientos. Los métodos de *clustering* analizan los datos generando una etiqueta para cada grupo, a diferencia de la clasificación que analizan datos ya etiquetados. Los términos *clustering* y agrupamiento son utilizados indistintamente a lo largo del capítulo. Estos métodos tratan de determinar grupos de individuos similares y, a partir del

agrupamiento, se toman diferentes decisiones para cada grupo. Una forma de definir un clúster es mediante un centro y una forma.

Los puntos del clúster son vectores $\mathbf{x_i}$ tales que $\mathbf{x_i} = (x_{i1}, ..., x_{id})$, donde $i = 1..N$ son propiedades de un espacio, en el cual cada componente $x_{il} \in A_l$ (con $l = 1...d$) representa atributos numéricos, categorías o texto, entre otros. El conjunto de observaciones y atributos se pueden disponer en una matriz $N \times d$. Normalmente estos clústeres no tienen intersección común entre ninguno, y de la unión de todos ellos más los *outliers* (datos anómalos) se obtiene el conjunto original X.

$$X = C_1 \cup ... \cup C_k \cup C_{outliers}, C_i \cap C_j = \phi,\ i \neq j$$

Teniendo en consideración el tamaño del *dataset*, éstos pueden residir en memoria o en almacenamiento secundario (lo cual requiere algoritmos especiales para su manejo).

Los métodos de *clustering*, necesariamente, deben asociarse a una función denominada distancia que, dados tres puntos cualesquiera, *x,y,z* del espacio, cumple las siguientes propiedades:

- No negativa: $d(x, y) \geq 0$ y $d(x, y) = 0 \longleftrightarrow x = y$
- Propiedad simétrica: $d(x, y) = d(y, x)$
- Desigualdad triangular: $d(x, z) \leq d(x, y) + d(y, z)$

Los espacios se pueden clasificar en euclídeos y no euclídeos. Los puntos en un espacio $n - dimensional$ euclídeo son vectores de dimensión n y sus componentes pertenecen al dominio de los números reales. La norma L_2 es la medida de distancia habitual en estos espacios y se presenta en la Ec. (5.27).

$$d([x_1, x_2, \ldots, x_n], [y_1, y_2, \ldots, y_n]) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{5.27}$$

Análogamente, en la Ec. (5.28), se define la distancia en función de la norma L_r.

$$d([x_1, x_2, \ldots, x_n], [y_1, y_2, \ldots, y_n]) = (\sum_{i=1}^{n} |x_i - y_i|^r)^{\frac{1}{r}} \tag{5.28}$$

El caso $r = 1$ se denomina distancia de *Manhattan*.

Una propiedad importante que debe cumplirse para los espacios euclídeos, es que el promedio de un conjunto de puntos debe existir y pertenecer al espacio. Por este motivo un espacio con distancia de *Jaccard* no es euclídeo ya que no tiene sentido hablar de promedio de conjuntos. En el caso de distancia de *Hamming*, el promedio conduce a valores que pueden no ser un vector con $0's$ y $1's$, por lo que tampoco define un espacio euclídeo. Igual consideración se tendría con vectores cuyas componentes son enteros (espacios discretos).

5.8.1. Clasificación de los métodos de clustering

Existen diversas formas de clasificar a los métodos de agrupamiento. Una forma puede ser dividiéndolos en clústeres jerárquicos y por asignación de puntos. Otra forma de clasificación, que considera el tipo de espacio que caracterizan a las observaciones, identifica métodos para espacios euclídeos y no euclídeos (dentro de este tipo se encuentra, por ejemplo, el algoritmo *GRGPF,* el cual es de tipo jerárquico y también por asignación de puntos). De acuerdo al tamaño del *dataset*, se pueden ordenar en técnicas que gestionan a datos que pueden residir en memoria durante la ejecución del algoritmo u otras en las cuales debe utilizarse el disco haciendo pases de los datos. Otra forma de categorización cubre a los algoritmos que se adaptan a realidades donde los datos se generan como un *stream* (algoritmo *BDMO*, por ejemplo) y otro ordenamiento considera si se utilizan métodos de aprendizaje automático o no.

Siguiendo el primer abordaje sugerido se menciona la siguiente clasificación:

1. Métodos Jerárquicos

- Algoritmos aglomerativos.
- Algoritmos divisivos.

2. Métodos por asignación de puntos

- *clustering* Probabilístico
- Medoides (Objetos del *dataset* cuya disimilitud promedio es la menor a los demás puntos del clúster. Son insensibles a los *outliers* [30]).
- *K-medias*
- *BFR*
- *CURE*

5.8.2. Maldición de la dimensión

En espacios con un gran número de dimensiones, la distancia entre pares de puntos es parecida. La maldición de la dimensionalidad se refiere a que las distancias entre un par de puntos generados en forma aleatoria son, aproximadamente, similares.

Una forma de describir el efecto es considerar una muestra de una variable aleatoria tomada con distribución uniforme en un híper cubo unitario de dimensión p [3]. Su representación es: $[-1, 1]^p$. Para $p = 1$ el 100 % de los puntos de una bola de centro en el origen y radio 1 se encuentran en el híper cubo. Si la dimensión es $p = 2$, la bola abarca el círculo de radio la unidad, es decir π. Notar que los puntos de la bola cubren $\frac{\pi}{4}$, que es, aproximadamente, el *79 %* del cubo. Si $p = 10$, solo el 0,25 % de las observaciones estarán a una distancia menor que 1 del origen. En la práctica se ha observado que el hecho de agregar características puede degradar el desempeño, si el número de muestras es pequeño en relación al de atributos. Si n representa a la cantidad de puntos y d a la cantidad de dimensiones, una buena práctica es tener en consideración que se verifique: $\frac{n}{d} \geq 10$ [31]. En un número de dimensiones mayores a 15 ya se pueden observar los efectos severos de dicho fenómeno [30].

5.8.3. Características del agrupamiento

En la Figura 5.15 se puede apreciar el diagrama para el método de *clustering* .

El coeficiente $\hat{w}$ representa a los parámetros y el modelo de *ML* es la clusterización. No se dispone de etiquetas en el sentido de que no hay supervisión. En la figura la palabra etiquetas se refiere a los centros (de los clústeres) al cual se pretende incorporar una observación. La métrica de calidad considera la distancia a los centros.

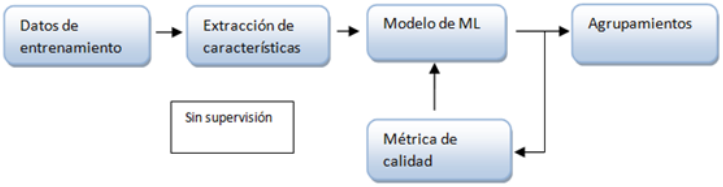

Figura 5.15: Bloque de flujo para el método de clustering

Para la construcción de estos algoritmos se debe considerar:

- Asignación de las observaciones.
- Medida de distancia.
- Condición de parada.
- Orden del algoritmo y cantidad de datos

Asignación de las observaciones

Se determina un conjunto inicial de clústeres y se clasifican los puntos de acuerdo al clúster más cercano. A veces, es necesario efectuar una estimación de los clústeres iniciales. Se pueden combinar o dividir los clústeres y los *outliers* pueden permanecer sin clúster.

Criterios para definir la distancia

- Si los datos son vectores, la medida de similaridad puede ser la distancia del coseno, *Hamming*, etc. (la similaridad y la distancia están directamente relacionadas, Sección 5.1.2)
- Si los datos son conjuntos se puede utilizar la distancia de *Jaccard* (por ejemplo comparación de documentos).
- Si los datos son puntos se puede usar la distancia euclídea.

Condición de parada para el algoritmo

Algunas condiciones de parada pueden ser:

- Se sabe de antemano cuántos clústeres existen (llegando a esta cantidad, se detiene el algoritmo).
- Cuando una nueva combinación produce clústeres de mala calidad.
- El diámetro del clúster obtenido excede un umbral.
- Análogamente se puede establecer condiciones respecto al radio.
- Si la densidad de un clúster es menor que un umbral. La densidad se puede definir como cantidad de puntos sobre el volumen (razón del número de observaciones y el radio del clúster).

Orden del algoritmo y cantidad de datos

Sin descartar la importancia que tiene el orden del algoritmo, al manejarse una gran cantidad de datos, incluso los problemas que presentan un orden

lineal, se vuelven inmanejables. En estos casos se puede utilizar algoritmos que gestionan conjuntamente la memoria y el disco. Otra opción es utilizar la técnica de *MapReduce* para dividir las tareas, como, por ejemplo, el método de *k-medias* (*Sección 5.8.5*).

5.8.4. Clustering jerárquico

El clustering jerárquico normalmente lleva a cabo una partición del espacio por medio de la construcción de un árbol, en el que las hojas son los datos y los nodos son subconjuntos. Son algoritmos que pueden ser utilizados en pequeños conjuntos de datos. Cuando el espacio no es euclídeo, se utilizan clustroides (en ausencia de centroides) o punto medio en los clústeres [4]. Los clustroides no necesariamente son puntos pertenecientes al clúster.

De acuerdo a la construcción de dicho árbol, se dividen en:

- Divisivo (*top-down*): se inicia con un solo clúster y, sucesivamente, se van dividiendo en nuevos clústeres, hasta cumplirse la condición de parada.
- Aglomerativo (*bottom-up*): comienza con cada punto perteneciendo a un clúster y luego, de acuerdo a la «cercanía», se fusionan las observaciones. Se continúa hasta que se satisface una condición de parada.

Motivación del clustering jerárquico

- Evita el problema de determinar el número de clústeres de antemano.
- Permite el uso de dendogramas en el cual se visualizan los clústeres en distintas granularidades, posibilitando detener el algoritmo tempranamente.
- Permite definir cualquier métrica de distancia entre puntos.
- Permite capturar formas más complejas de los clústeres.

Dendograma

Un dendograma captura las siguientes características de un agrupamiento:

- Cuáles son los posibles clústeres.
- Clústeres fusionados.
- Distancia de fusión entre ellos.

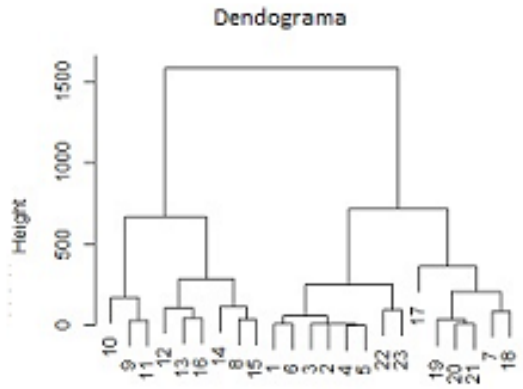

Figura 5.16: Dendograma de cantidad de conexiones agrupadas por hora en el Plan Ceibal

En la Figura 5.16, los números en los extremos indica las horas y en el eje vertical se indica la cantidad de conexiones. Si se efectúa el corte en *1000* conexiones, se puede visualizar dos clústeres y si el corte se realizara en *500* conexiones se observan cuatro agrupaciones. Este dendograma se extrajo del estudio conducido en el *Capítulo 7 Caso de Estudio: Plan Ceibal*.

Clústeres jerárquicos en espacios no euclídeos

En estos espacios es posible emplear la distancia de *Jaccard, Edit* o Coseno, por ejemplo, pero no las basadas en la ubicación de los puntos. Por lo tanto no es viable usar el concepto de centroide, siendo sustituido por el clustroide. Éste, a diferencia del centroide, es un punto existente en el conjunto, debiendo ser el más «cercano» a todos los demás en el clúster según alguno de los siguientes criterios:

- Menor máxima distancia a otros puntos.
- Menor distancia promedio a otros puntos.
- Menor suma de los cuadrados de las distancias a otros puntos. Es decir, siendo d la distancia, se verifica $min \sum_{x \in C} d(x, c)^2$ siendo c el clustroide del clúster C.

Las nociones de distancia se mantienen para espacios no euclídeos [4] cambiando centroide por clustroide.

Orden del algoritmo de agrupamiento jerárquico

Los agrupamientos jerárquicos son de orden $O(n^3)$ y, por lo tanto, no es conveniente su utilización cuando se trata de muchos datos. El orden cúbico es debido a que inicialmente se comparan todas las distancias, obteniéndose n^2 comparaciones. En el siguiente paso es $(n-1)^2$; y en el siguiente es $(n-2)^2$ y así sucesivamente. La suma de todas estas operaciones es de $O(n^3)$. Es posible realizar un ordenamiento de los clústeres y obtener un $O(n^2 \log n)$ pero aún sigue siendo alto cuando los datos no entran en memoria [4].

5.8.5. Clústeres por asignación de puntos: *k-medias*

El método de *k-medias* es una alternativa al alto costo computacional de los métodos jerárquicos al tratar grandes conjuntos de datos [4]. En esta sección se analiza el algoritmo, la elección del número k, elección de los k puntos iniciales y la complejidad del algoritmo. Finalmente se presenta el método de *MapReduce* aplicado a *k-medias*. Es importante considerar que para grandes *datasets* existen algoritmos más eficientes que *k-medias* (*BFR* y *CURE*, entre otros).

Método de k-medias

Se dispone de un número k determinado, el cual representa al número de clústeres. A continuación, se inicializan los k clústeres, asignando un punto por clúster que son los centroides provisorios. Luego, todos los puntos del *dataset* son asignados al clúster, en el cual, la distancia será menor al centroide. Posteriormente, el centroide de cada clúster es actualizado y luego se reasignan las observaciones, pudiendo, incluso, cambiar de clúster. Iterativamente se repite la actualización del centroide y de los puntos hasta alcanzar una convergencia.

K-medias puede ser calculado con el algoritmo en coordenadas descendentes. En este caso, se itera en dos pasos: en uno de ellos se asignan observaciones al clúster más cercano, y en el otro se actualizan los centros de los clústeres, definiéndose una función objetivo a minimizar.

El algoritmo asigna la observación x_i, al clúster cuyo centro es el más próximo, μ_j, como se expresa en la Ec. (5.29). Luego se corrigen los centros de los clústeres, ajustándolos al promedio de las observaciones asignadas, Ec. (5.30).

$$z_i \leftarrow argmin_j \| \mu_j - x_i \|_2^2 \tag{5.29}$$

$$\mu_j = \frac{1}{n_j} \sum_{i:z_i=j} x_i \tag{5.30}$$

Las dos ecuaciones anteriores se pueden combinar y, en este caso, se puede iterar sobre una función objetivo, lo que implica poder utilizar el método de coordenadas descendentes.

El método de *k-medias* es muy sensible a la inicialización ya que puede conducir a distintas convergencias locales.

Inicialización inteligente de los k (k-means++)

La inicialización inteligente busca determinar los centros de los clústeres lo más alejados posible entre sí, no aleatoriamente. En primer lugar se elige aleatoriamente un centro de un clúster. Se busca el nuevo centro entre los puntos restantes, de tal forma que la probabilidad de que un punto x sea seleccionado como nuevo centro (proporcional a la distancia al cuadrado), sea la mayor. Con los dos nuevos centros se buscan los puntos más próximos a cada centro, seleccionando aquel punto cuya distancia al cuadrado es la mayor y se repite hasta lograr k centros. Inicializar los centros de esta manera es más costoso que hacerlo aleatoriamente, aunque permite obtener una convergencia más rápida. El método *k-means++* mejora la calidad de los óptimos locales obtenidos.

Otra elección de los k puntos iniciales

Otra posibilidad consiste en tomar una muestra que entre en la memoria y hacer un *clustering* jerárquico en k aglomeraciones. Luego, para definir los k puntos, se consideran los puntos más cercanos al centroide de cada clúster encontrado.

Si k es muy grande los clústeres se ajustan más a los datos, pudiéndose considerar como un efecto similar al sobreajuste (aunque el sobreajuste es exclusivo del aprendizaje supervisado).

Definición del número k

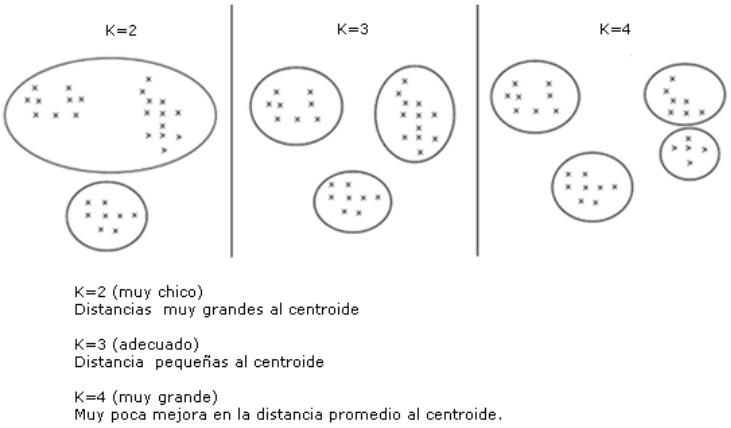

Figura 5.17: Representación de los clústeres al variar k

La determinación de k puede ser realizada por ensayo y error, en tanto no se disponga de información previa. Se puede comprobar que al aumentar k el promedio de las distancias a los centroides decrece (Figura 5.17). Otro método que ayuda a determinar k es la gráfica del promedio de los diámetros de los clústeres en función del número de clústeres (Figura 5.18), cuando no es tan pronunciada la caída, se encuentra el mejor valor para k [4].

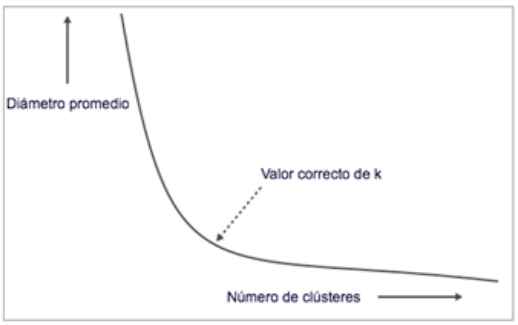

Figura 5.18: Valor de k

Si no existe ninguna idea del valor correcto de k se puede encontrar una relación en un número de operaciones de agrupamiento haciendo crecer a k exponencialmente. Se comienza ejecutando el algoritmo de *k-medias* para $k = 1, 2, 4, 8$ *etc.* Finalmente, se encuentran dos valores v y $2v$ entre los cuáles disminuye poco el promedio. El valor de k se encuentra en el paso anterior (entre $\frac{v}{2}$ y v)

Complejidad de k-medias y el volumen de datos: algoritmos BFR y CURE

Para cada punto se computa la distancia a cada centroide por lo tanto es de orden $O(kn)$. En lo que se refiere a alcanzar la convergencia, no puede calcularse un límite teórico [4].

Cuando se trata de un volumen importante de datos, interesa encontrar un algoritmo como el estudiado, pero que sólo deba realizar una pasada sobre los datos. Algunos algoritmos adecuados para resolver este problema son BFR y CURE [4]. La descripción de *BFR* se presenta en el *Anexo 8*.

La principal limitación de *BFR* es la restricción de que los grupos se deben encontrar distribuidos normalmente en cada dimensión. Si las elipses forman cierto ángulo con los ejes, no funciona el algoritmo BFR, siendo apropiado *CURE* (*clustering* usando representantes). Este método es del tipo de asignación de puntos a los clústeres y asume una distancia euclídea permitiendo a los grupos adoptar cualquier forma. Su funcionamiento se puede leer detalladamente en [4].

MapReduce en K-medias

Se había mencionado que los métodos *BFR* y *CURE* son más eficientes que *K-medias* como métodos de *clustering* para grandes *datasets*, pero es posible que *K-medias* pueda adaptarse a grandes cantidades de datos aplicando *MapReduce* en sus iteraciones.

La fase de «Clasificar» del método consiste en: asignar observaciones al clúster más cercano. Un punto x_i es asignado al cluster z_i, luego de evaluarse cuál es el centro de cluster μ_j, más cercano.

Implementación de la fase clasificar con la función Map

Para el caso del *Map* el programa recibe el *set* de centros de los clústeres, además de un punto a clasificar, comparando así, cuál es el que se encuentra a menor distancia. Se obtiene un par (clúster, punto).

$map([\mu_1, \mu_2, ..., \mu_k], x_i)$, siendo μ_i el centro del clúster i, y x_i el punto a clasificar.

Implementación del recentrar con la función Reduce

La fase de «recentrar» del método corrige los centros de los clústeres como media de las observaciones asignadas.

La función *Reduce* se encarga de recentrar calculando el promedio sobre el total de puntos del clúster. Recentrar se corresponde con la función *Reduce* de *MapReduce* y consiste en corregir los centros de los clústeres como media de las observaciones asignadas: $\mu_j = \frac{1}{n_j} \sum_{i:z_i=k} x_i$.

Es decir, para el total de los puntos de un clúster j se determina el promedio, siendo equivalente a recentrar, obteniéndose una nueva media para dicho clúster.

La función *Reduce* ejecuta: $emit(z_j, x_i)$, siendo z_i el identificador del clúster.

5.8.6. Clústeres en modelos de stream de datos

El objetivo es obtener los clústeres de un *stream* de datos. Se considera el modelo de ventanas deslizantes en el cual se observan N puntos en un determinado momento.

Se trata de ubicar el o los centroides (espacio euclídeo) o clustroides (espacio no euclídeo) en m puntos ($m \leq N$).

De acuerdo a la presunción que se haya hecho, los m puntos del *stream* se procesan de acuerdo a la técnica que se utilice. En este enfoque se asume que la estadística de los datos permanece fija [4]. Sin embargo, en muchos casos los datos estadísticos cambian, modificándose los centros, produciéndose divisiones o fusiones.

Es decir, dados m puntos se definen clústeres que, al procesarse una nueva cantidad de puntos, podrán variar. El algoritmo *BMDO* (*Babcock, Datar, Motwani, O'Callaghan*) es un método para procesar un *stream* de datos con estadísticas variables. Se puede ver un resumen de su funcionamiento en [4].

5.8.7. Clústeres probabilísticos

El método de *k-medias* únicamente considera los centros, pero no las formas de los clústeres. Tampoco resuelve adecuadamente si los clústeres no están muy separados o si no están bien definidos los límites entre ellos (solapamiento); ni si los tamaños de los clústeres son muy diferentes, o los ejes no están alineados. Una solución es mediante el empleo de modelos probabilísticos, los cuales proporcionan asignaciones suaves acompañadas de una probabilidad. A su vez, consideran la forma, además del centro y permite el entrenamiento en base a parámetros (es decir se trata de modelos paramétricos, a diferencia de los considerados previamente en este capítulo).

Modelo de Mezcla de Gausianas.

En contextos estadísticos, el modelo de mezcla de gausianas (GMM), es empleado para representar la presencia de sub-poblaciones en un conjunto, sin requerir que en un *dataset* observado éstas sean identificadas.

Se considera un grupo de clases formadas por puntos, donde cada clase posee una distribución normal en cada una de sus características. Estas gausianas pueden tener alguna estructura correlacionada entre las diferentes características. Se busca identificar cada uno de las clases. Los puntos están en una distribución gausiana y la mezcla de éstas, es una combinación de esas funciones de densidad de probabilidad. El modelo de mezcla de gausianas es formado con una combinación convexa de las gausianas individuales, por lo que, por definición de combinatoria convexa, todos sus coeficientes suman uno y cada uno de dichos coeficientes es positivo. Por lo tanto, se trata, a su vez, de una función de distribución de probabilidad [32].

El *GMM* corresponde a una distribución de mezcla, que representa la distribución de probabilidad de alguna observación en la población en general [33].

Es necesario asumir que las poblaciones tienen diferente tamaño por lo que resulta adecuado asignarle un valor a su importancia. Para ello, se puede realizar un promedio con pesos. Formalmente se introduce un *set* de pesos de clústeres, π_k, uno para cada clúster: $\pi_1, \pi_2, \ldots, \pi_n$. Dichos pesos representa la proporción de datos de cada una de las categorías. Cada uno de ellos varían entre *0 y 1*.

$$0 \leq \pi_k \leq 1 \tag{5.31}$$

$$\sum_{k=1}^{K} \pi_k = 1 \tag{5.32}$$

Las ecuaciones (5.31) y (5.32) establecen las condiciones para la combinación convexa.

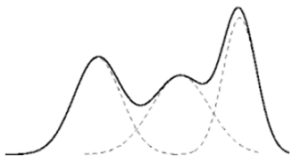

Figura 5.19: Mezcla de Gausianas

En la Figura 5.19 se aprecia en línea llena la mezcla de gausianas de las gausianas (representadas en trazo interrumpido).

Interpretación de los términos de la mezcla

De acuerdo al modelo, interesa averiguar cuál es la probabilidad de que un dato pertenezca a un determinado clúster k. Dada la observación x_i, sea z la función de asignación a un clúster, que verifica: $p(z(x_i) = k) = \pi_k$. Dada una observación x_i de un clúster k, también interesa calcular cuál es la probabilidad de que sea seleccionada. Esta probabilidad está dada por al Ec. (5.33), donde μ_k es le media y Σ_k la varianza de la distribución normal del clúster k.

$$p(x_i|z(x_i) = k, \mu_k, \Sigma_k) = N(x_i|\mu_k, \Sigma_k) \tag{5.33}$$

Las distribuciones normales pueden tener zonas en común, por lo tanto, en algunos casos, no resulta claro en qué lugar asignar el dato. Por ejemplo, si se trata de clasificación de imágenes, donde interesa separar dos tipos de imágenes diferentes, puede ocurrir que alguna imagen contenga a ambos tipos.

El método es más flexible que el método de *k-medias*, donde se asumen clústeres de forma esférica.

Un algoritmo para inferir los parámetros (estimadores de máxima verosimilitud) de los clústeres es el de esperanza-maximización, presentado en el *Anexo 8*

5.8.8. Resumen

Las técnicas de *clustering* son métodos de aprendizaje no supervisado descriptivo, destinados a la identificación de agrupaciones de datos similares, en base a una función de distancia.

Es posible la adaptación de algunos métodos para el tratamiento de grandes *datasets* utilizando *MapReduce* (por ejemplo: *k-medias*)

5.9. Conclusiones

Desde el más alto nivel de descripción, este capítulo abordó los fundamentos de las técnicas de *Data Mining* y su aplicación a grandes conjuntos de datos que no entran en la memoria. El enfoque que se tomó fue bajo una óptica algorítmica que se complementa con los abordajes desarrollados en los capítulos anteriores donde se dio un mayor énfasis al entrenamiento y evaluación de modelos de *ML*.

Se presentaron algoritmos de búsqueda, de recuperación de ítems, de clasificación, vecinos más cercanos, árboles de Decisión, ensamble de clasificadores, redes neuronales artificiales y *clustering*.

Las condiciones en que se generan los datos en *Big Data*, requieren que las técnicas de *Data Mining* deban ser analizadas y adaptadas a los problemas que deben gestionar un gran *dataset* que no puede cargarse en memoria, o una cantidad muy grande de dimensiones o realidades donde los datos se generan constantemente como son los *streams* de datos. En general, una forma de resolver el problema es utilizando la tecnología *MapReduce*, que permite un procesamiento en paralelo. En la búsqueda de ítems similares, se utilizaron las técnicas de *minhashing* y *LSH*, las cuales tienen una muy alta aplicabilidad en problemas que envuelven a grandes *datasets* posibilitando soluciones a muchos problemas del estilo.

Algunas tipos de técnicas no fueron desarrolladas, aunque sería importante su análisis y aplicabilidad para *Big Data* (algoritmos genéticos, series temporales, lógica difusa y razonamiento basado en casos, entre otros).

Capítulo 6

Aplicaciones

En este capítulo se presentan cuatro aplicaciones bajo el contexto en que se definieron en la *Sección 1.2, Taxonomía de términos en Big Data*.

6.1. Análisis de Links

En la exploración de información en Internet a través de un motor de búsqueda, es importante el orden en que se presentan los resultados y su coincidencia con el contenido de lo solicitado. En general, un motor de búsqueda está formado por una interfaz (para que el usuario consulte), una base de datos (que almacena los resultados) y un indexador (encargado de recorrer la Web y agregar información de nuevos sitios a la base de datos). En sus orígenes, las búsquedas en Internet funcionaban deficientemente debido a la baja calidad de los algoritmos empleados y por la acción de los *spammers* que intentaban engañar al usuario conduciéndolo a sitios Web no deseados para beneficio propio [4]. *Google* fue el primer buscador en vencer a los *spammers* mediante una tecnología denominada *Page Rank* que presentaba una lista de respuestas a una búsqueda ordenada por relevancia decreciente. A partir de los *spammers* se originaron técnicas para manipular el *Page Rank* de los sitios, denominadas *link spam* (o *search engine optimization*, para engañar a los usuarios). A modo de contramedida, se creó el *Trust Rank* [4].

En esta sección se cubren los siguientes aspectos para calcular la importancia de nodos en un grafo: *Page Rank*, topic-specific, *HITS* (o *hubs and authorities*) y algoritmos de detección de *spam*. Debido a que el *Page Rank* puede ser considerado como una sumarización, el análisis de links se ha consi-

derado, para el presente trabajo, como una aplicación de las técnicas y tareas nombradas en el *Capítulo 1*.

Técnicas y tareas en el análisis de links

De acuerdo a la clasificación de las tareas y técnicas para *Big Data* mencionadas, se presenta en la Tabla 6.1, las técnicas asociadas al *Page Rank*, que, como se mencionó, llevan a cabo la tarea de sumarización de los *rankings* de importancia de los sitios Web. Se incluye el algoritmo de *Page Rank* como una técnica específica para este problema. A su vez se indican otras tareas y métodos que podrían aplicarse para su cálculo.

Tarea / Técnica	Predictivo		Descriptivo				
	Clasificación	Regresión	Agrupamiento (Clústeres)	Reglas de Asociación	Correlaciones/ Factorizaciones	Detección de Anomalías	Sumarización
Algoritmo de Page Rank							V
Redes Neuronales		V					V
Máquinas de vectores soporte							V
MapReduce		V					V
Vecinos más próximos		V	V				V
Algoritmos genéticos							V
Análisis de grandes grafos							V

Tabla 6.1: Utilización de técnicas y tareas en el análisis de links

6.1.1. Page Rank

Se denomina *Term spam* a las técnicas para engañar a los motores de búsqueda, intentando mostrar que una página es algo que no es en realidad, cuando es enviada una *query* a un motor de búsqueda, basada en palabras. Para evitar el *Term Spam*, *Google* introdujo dos innovaciones [4].

- El algoritmo de *Page Rank*: el cual determina a qué lugar, los usuarios de internet, ubicados en un sitio cualquiera, podrían converger si eligieran links aleatoriamente (es decir, determinar en qué sitio de la Web se congregarían dichos usuarios).
- El cálculo del contenido de un sitio Web se encuentra dado por términos que aparecen en las páginas, que apuntan a dicho sitio (aún si un *spam* creara millones de sitios apuntando a una página, para darle importancia,

no afectaría su *ranking*, dado que esos sitios no tienen una importancia real).

Importancia de una página (Page Rank)

La Web puede visualizarse como un grafo dirigido, donde los nodos son los sitios y las aristas son los hipervínculos de un sitio a otro. La importancia de una página es dada por una función (recursiva), que depende de cuántas páginas «importantes» apuntan a ella.

El *Page Rank* de una página en la Web es una función que asigna un número real a cada página que se encuentra indexada, midiendo en términos generales, cuán «importante» ésta es [4]. La información de cada vértice debe ser actualizada frecuentemente. La forma de llevarlo a cabo, es a través de un proceso que reasigna valores a cada nodo. En determinado momento, el algoritmo converge (significando que el valor del *Page Rank*, para un nodo determinado, queda estable). Esto se debe cumplir para todos los nodos.

El *Page Rank* de un sitio A tiene una dependencia recursiva al tratarse de una función del valor del *Page Rank* de los sitios que tienen hipervínculo apuntando hacia A.

De esta definición, se genera un sistema de ecuaciones que, con el agregado de restricciones, permiten determinar el *ranking* de cada sitio.

Debido a la gran cantidad de sitios, los métodos de eliminación gausiana no son aplicables, utilizándose el método iterativo de la potencia. Al estudiar la convergencia de las soluciones, se estudian dos casos, en los cuales no se obtiene convergencia (*dead ends* y *Web traps*). Para que el modelo pueda resolver estas situaciones al seguimiento de los vínculos se le agrega un indexado probabilístico dado por un factor de amortiguamiento β [34].

El problema que se intenta resolver es, dado un visitante al azar, determinar la probabilidad de visitar una página, o, en otras palabras, cuáles son los sitios con mayor probabilidad de ser visitados en forma aleatoria.

Formulación de Page Rank en base al flujo

En (6.1) Se define el *ranking* de una página.

$$r_j = \sum_{i \to j} \frac{r_i}{d_i} \tag{6.1}$$

En la Ec. (6.1) d_i indica el grado de los links de salida (hipervínculos) de la página j. La notación $i \to j$, representa que la página i tiene un hipervínculo a la página j. Si la página j con importancia r_j tiene n links de salida, los votos que aporta j, a cada uno de sus links son r_j/n . La importancia de la propia página j es la suma de los votos de los links que apuntan al nodo j. De esta forma, la página j tiene más votos si muchas páginas la apuntan. A los efectos de penalizar la importancia que una página, por sí misma, pudiera obtener apuntando a otras, es que se divide por la cantidad de links salientes.

La recursividad provoca que también influya la importancia de las páginas que apuntan al sitio. Debido a ello, se le denominan ecuaciones de flujo. La solución del sistema de ecuaciones que se obtiene, permite obtener el *ranking* de cada sitio Web.

Ejemplo 6.1

En la Figura 6.1 el *Page Rank* del sitio j es $r_j = \frac{r_i}{6} + \frac{r_k}{4}$.

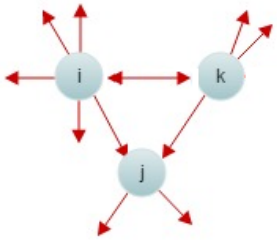

Figura 6.1: Cálculo del Page Rank

El sistema de ecuaciones que se obtiene es

$r_i = \frac{r_k}{4}$

$r_j = \frac{r_i}{6} + \frac{r_k}{4}$

$r_k = \frac{r_i}{6}$

El sistema es indeterminado por lo que se considera, adicionalmente, la restricción: $r_i + r_j + r_k = 1$.

Las ecuaciones de flujo se pueden resolver con el método de *Gauss,* sin embargo, para este tipo de sistema de ecuaciones, correspondientes a grafos grandes, no es posible aplicar eliminación Gausiana, ya que es de orden $O(n^3)$.

Incluso el método iterativo es de orden cuadrado. No obstante, se puede aprovechar el hecho de que la matriz es *sparse* (en promedio, cada página tiene 10 hipervínculos en ella).

Formulación Matricial (Método iterativo de las potencias)

Se considera una matriz M de adyacencia estocástica (la suma de las columnas es la unidad). La dimensión de M es $n \times n$, siendo n la cantidad de sitios Web. En el *Anexo 9*, Método de la Potencia y *Page Rank*, se puede leer el método.

Spider Trap

Un *Spider Trap* es un grafo cuyas salidas en el grafo están dentro de un grupo, el cual, sobre el final de la iteración del método iterativo de las potencias, absorbe toda la importancia.

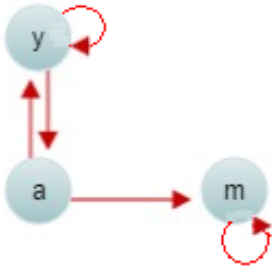

Figura 6.2: Representación de un Spider Trap

El ejemplo de *Spider Trap* de la Figura 6.2 conduce a las siguientes ecuaciones:

$$\begin{array}{cccccc} r_y & \frac{1}{3} & \frac{2}{6} & \dots & 0 \\ r_a & \frac{1}{3} & \frac{1}{6} & \dots & 0 \\ r_m & \frac{1}{3} & \frac{3}{6} & \dots & 1 \end{array}$$

Problema del dead end

Los *dead ends* son nodos de la red que no tienen link de salida. En la Figura 6.3, se pueden apreciar dos ejemplos donde el nodo b y el m son *dead ends*, en cada imagen, respectivamente.

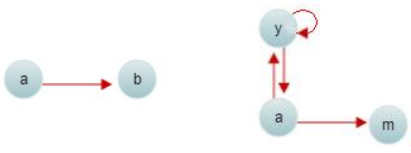

Figura 6.3: Dead Ends

Se diluye la importancia al aproximarse a cero. Aplicando el método para la primera red de la Figura 6.3, se obtiene:

$$\begin{array}{cccc} r_a & 1 & 0 & 0 & 0 \\ r_b & 0 & 1 & 0 & 0 \end{array}$$

Aplicando el método para la segunda red de la Figura 6.3, se obtiene:

$$\begin{array}{cccc} r_y & \frac{1}{3} & \frac{2}{6} & \dots & 0 \\ r_a & \frac{1}{3} & \frac{1}{6} & \dots & 0 \\ r_m & \frac{1}{3} & \frac{1}{6} & \dots & 0 \end{array}$$

Sobre el final todos los nodos tienen *ranking* 0.

Solución para los spider traps y dead ends

Si una matriz representa un *dead end*, deja de ser estocástica ya que los componentes de alguna columna suman cero (significa que no existe probabilidad de salida). Una matriz donde la suma de las columnas es a lo sumo 1 se llama sub estocástica. Estos problemas se resuelven agregando un factor de amortiguación modificando la forma de calcular el *Page Rank*, lo que posibilita a un internauta aleatorio, con probabilidad β, seguir un link al azar. En la llamada iterativa del algoritmo, la actualización se hace en base a la Ec. (6.2).

$$r^{'} = \beta M r + (1 - \beta)\frac{e}{n} \tag{6.2}$$

Resulta razonable suponer $\beta > 1/2$. Es decir, estando en una página, se tiende a usar más los vínculos que allí están, que hacer una nueva elección al azar. Normalmente β se encuentra entre *0.8 y 0.9* [35].

En dicha ecuación, r es el vector actual de *rankings*, n la cantidad de nodos, y M la matriz de transición. El término $\beta M v$ representa con probabilidad β, que el visitante «siga» un link de salida desde donde se encuentra.

$(1 - \beta)e/n$ es un vector con todas sus componentes $(1 - \beta)/n$. Representa la introducción con probabilidad $1 - \beta$ de un link aleatorio. Si no existen *dead ends*, tanto la probabilidad de no seguir un link de salida por el navegante o de introducir uno nuevo, es la misma: $1 - \beta$, lo cual se puede visualizar como que el internauta actual decidiera entre seguir un link de salida, o «saltar» a otro sitio aleatorio (interpretado como introducir una dirección manualmente en el explorador).

Entrada: $Grafo\ G$ y parámetro β (G puede contener *dead ends* y *spider traps*)

$Iniciar\ r_j^{(0)} = \frac{1}{n}, t = 1$

Hacer

$\forall j: r'^{(t)}_j = \sum_{i \to j} \beta \frac{r_i^{(t-1)}}{d_i}$

$r'^{(t)}_j = 0$ *si el grado de entrada de* j *es* 0

$\forall j: r_j^{(t)} = r'^{(t)}_j + \frac{1-S}{n}$ *donde* $S = \sum_j r'^{(t)}_j$

$t = t + 1$

Mientras $\sum_j \left| r_j^{(t)} - r_j^{(t-1)} \right| > \varepsilon$

Figura 6.4: Algoritmo de Page Rank

En el cuadro de la Figura 6.4 se presenta el algoritmo de *Page Rank* el cual inicializa todos los *rankings* de igual manera: $\frac{1}{n}$. Las iteraciones operan hasta que se alcanza la convergencia. En cada iteración se actualiza el *score* de cada sitio. Cuando un nodo no tiene aristas de entrada su *ranking* es cero.

En el caso de que se encuentra un *dead end*, hay que calcular qué cantidad del total del *Page Rank*, se ha perdido y recuperarlo. Para ello se suma las componentes del vector actualizado del *Page Rank* (llamado S). Se supone que S es menor que 1 y eso significa que $1 - S$ es lo que se ha perdido. Entonces en cada entrada correspondiente a cada nodo, se suma $\frac{(1-S)}{n}$.

6.1.2. Page Rank sensible a tópicos

El *Page Rank* sensible a tópicos es una mejora del algoritmo anterior, donde, a algunas páginas de interés para el usuario, se les asigna un peso mayor. El algoritmo crea un vector para cada tópico, sesgando el *Page Rank*, a favor de páginas que pertenecen a determinada categoría. En este caso, la función de iteración se modifica obteniéndose la siguiente igualdad.

$$r' = \beta M r + (1 - \beta)\frac{e_S}{|S|}$$

El valor $|S|$ es el número de páginas del tópico considerado y e_S es un vector cuyos componentes correspondientes a los sitios de S son uno. El resto de las componentes valen cero [4].

6.1.3. Otras Técnicas

Link Spam

Se denomina de esta forma a las técnicas cuyo objetivo es aumentar artificialmente el *Page Rank* de un sitio. Las técnicas *Trust Rank* y *Spam Mass* se utilizan para disminuir el *ranking* de páginas gestionadas por *Link Spam*. [4].

Trust Rank

Se consideran un conjunto de páginas para las cuales se sabe, de antemano, que no son *spam*. Luego de obtenido el conjunto, se trata como si fuera un caso de *Page Rank* sensible a tópicos [4].

Spam Mass

Se trata de determinar qué fracción del *ranking* viene del *spam*. Para ello se calcula el *Page Rank* r y el *Trust Rank* t. El *spam mass* del sitio es: $\frac{(r-t)}{r}$. Si el valor es cercano a cero o negativo, la página es confiable. Si es cercana a uno, es considerada *spam* [4].

Hubs and authorities, HITS

HITS (Hyperlink-Induced Topic Search) considera dos aspectos para definir la importancia de una página. A las páginas seleccionadas que aportan información útil sobre un tópico, se les denomina *Authorities* y a las páginas que no aportan información pero que facilitan la búsqueda sobre el tópico, se les denomina *hubs*. A través de esas dos características, se puede construir un *ranking* para los sitios Web [4].

6.1.4. Conclusiones

El *Page Rank* es una sumarización que indica la importancia que tienen todos los sitios en la Web. Para ello, el sitio debe estar indexado, de lo contrario, no tiene posibilidades de ser calificado.

El *Page Rank* modela el comportamiento de un usuario que se encuentra en un sitio de internet y puede elegir al azar entre los hipervínculos contenidos en el sitio Web o saltar aleatoriamente a un sitio ingresando la dirección en el *browser*.

Esta representación no tiene en consideración el contenido de los sitios, lo que ha originado modelos alternativos, como, por ejemplo, el *Page Rank* sensible a tópicos.

6.2. Sistemas de Recomendación

Los sistemas de recomendación (*SR*) han sido diseñados en base a algoritmos que intentan predecir la respuesta de un usuario, sobre un determinado ítem que pertenece a un conjunto de opciones existentes (por ejemplo, dada una película, decidir si «me gusta» o «no me gusta»). Las entidades usuarios e ítems se relacionan en los *SR* a través de una matriz, llamada matriz de utilidad, la cual tiene como filas a los usuarios y como columnas a los ítems. Sus elementos son un valor real, que indica el grado de preferencia que tiene el usuario sobre cada ítem.

Los dos tipos de *SR* más frecuentes son:

- Sistemas basados en contenido
- Sistemas de filtrado colaborativo

Los sistemas de recomendación basados en contenidos construyen un perfil de los ítems en base a atributos. Por ejemplo, si el ítem es un libro, se pueden considerar atributos como: autor, año y género, entre otros. Este perfil es constituido por pares atributo-valor (para el caso de documentos podría ser el *TF-IDF* de las palabras y para las imágenes, los *tags* sobre éstas, donde los usuarios expresan sus comentarios). También se necesita un perfil del usuario, el cual se extrae de la matriz de utilidad. En el *Anexo 9* se presenta el cálculo del *TF-IDF*.

En el caso de filtrado colaborativo uno de los algoritmos más usuales, es del tipo «usuario-usuario», el cual calcula la distancia entre pares de usuarios, basándose en alguna medida de cuánto ellos coinciden en la calificación de ciertos artículos. Cuanto mayor sea la coincidencia, menor será la distancia. Si la distancia entre ellos es menor que un umbral preestablecido, se dice que forman parte de una vecindad. Sin embargo, este enfoque no funciona correctamente, debido a que muchos pares de usuarios tienen pocos ratings en común. A esto se le suma el hecho de que son muy dinámicos, ya que las distancias cambian rápidamente. Esta dinámica implica realizar los cálculos de las distancias

al momento, debiendo el usuario esperar mucho tiempo por esta información (tiempo que, en ocasiones, no resulta aceptable).

A raíz de esto, es preferible utilizar algoritmos «ítem-ítem», los cuales calculan la distancia entre pares de artículos que usuarios han votado aproximadamente igual. Por ejemplo, personas a las que les gusta la música de *Mozart*, están próximos a que les guste la música de *Beethoven*. Si se aplica el concepto de vecindad a dicho contexto, se puede interpretar como que *Mozart* y *Beethoven* se encuentran en el mismo vecindario. Las distancias entre ítems que se basan en los ratings de los usuarios son relativamente estables en el tiempo (considerando que éstos son millones).

En cierta forma, se puede decir que los sistemas de recomendación, realizan sus recomendaciones en base al cálculo de las distancias, las cuales son empleadas para realizar recomendaciones [36]. Se utiliza un enfoque parecido a los algoritmos de vecinos más cercanos. Una medida de distancia utilizada es la correlación, la cual se desarrolla en el *Anexo 9*.

6.2.1. Técnicas y tareas en los Sistemas de recomendación

De acuerdo a la clasificación de tareas y técnicas, en la Tabla 6.2 se presentan las tareas asociadas a los Sistemas de Recomendación.

Tarea / Técnica	Predictivo		Descriptivo				
	Clasificación	Regresión	Agrupamiento (Clústeres)	Reglas de Asociación	Correlaciones/ Factorizaciones	Detección de Anomalías	Sumarización
Ítemsets similares	V		V	V			
Redes Neuronales	V	V	V				V
Ítemsets frecuentes	V		V	V			
LSH	V		V				V
Reducción de la dimensión	V	V			V		
Vecinos más próximos	V	V	V		V		
Máquinas de vectores soporte	V	V	V		V		
Algoritmos genéticos	V	V	V	V	V		V
MapReduce	V	V	V	V	V		V
Árboles de decisión							V
Ensamble de clasificadores	V		V				

Tabla 6.2: Utilización de técnicas y tareas en los sistemas de recomendación

6.2.2. Modelado de los Sistemas de recomendación

Modelo

Dados el conjunto C de clientes y el conjunto S de elementos, se puede definir una función llamada de utilidad (por ejemplo, la matriz de utilidad), la cual asigna una calificación a un par (cliente, artículo).

Función de utilidad $f : C \times S \rightarrow Ranking$

El *ranking* es un conjunto de calificaciones el cual puede ser una categoría de uno a cinco, o un número entre cero y diez o cualquier otro rango. Generalmente, es un conjunto ordenado de manera que un valor bajo indica que al usuario le gusta el producto en menor medida y un valor más alto, que le gusta más. Otros sistemas de calificación se basan en cantidad de estrellas. Generalmente, poseen una escala de *1* a *5*, pero en algunos casos, se permite calificar en otras escalas.

Considerando el problema de usuarios que ven películas, se puede construir una matriz de utilidad, donde en las filas estarían los usuarios, y en las columnas, las películas. Notar que, habitualmente, esta matriz es *sparse*.

En los sistemas recomendación, una vez que un usuario calificó ciertas películas con una calificación alta, éstas pueden ser recomendadas.

Se identifican tres problemas importante en los *SR*:

- Forma de recopilar las votaciones al inicio del proceso, para ingresar valores a la matriz.
- Extrapolar calificaciones desconocidas a partir de otras conocidas. Interesa principalmente, para calificaciones ya asignadas a algunos ítems, conocer cuál sería la recomendación para un usuario del cual no se conoce su respuesta.
- Forma de evaluar un *SR*, una vez que éste extrapole calificaciones desconocidas, a partir de calificaciones conocidas.

En el *Anexo 9* se presentan los desafíos planteados junto a consideraciones para el análisis de los *SR*, ratings, preferencias, *ranking*, evaluación de los *SR* (incluyendo las métricas) y el problema del arranque en frío. Asimismo se presentan los sistemas de recomendación basados en contenidos y de filtrado colaborativo (usuario–usuario e ítem – ítem).

6.2.3. Conclusiones

Los *SR* pueden considerarse como una aplicación de las técnicas y tareas señaladas antes. Debido a la complejidad y posibilidades de estudio los *SR* presentan desafíos en múltiples áreas.

La evaluación de los *SR* es esencial para asegurar su calidad. Por otro lado, la gran mayoría de los *SR* necesita asumir ciertas hipótesis para simplificar su diseño. En el *Anexo 9*, se estudia el concepto de evaluación y métricas de los *SR*, dándose énfasis a las diferentes medidas que deben emplearse (de acuerdo al objetivo específico) y sus limitaciones.

Caso de estudio

Estudiar la viabilidad del diseño de una red neuronal para establecer una separación en regiones para la predicción del *score* de un ítem para un determinado usuario, construyéndose un *ranking* de preferencias.

6.3. Análisis de Sentimientos

Con el crecimiento del número de usuarios de Internet, quienes, entre ellos, tienen interés en expresar sus opiniones, sobre noticias, o efectuar calificaciones en la Web, generan una gran cantidad de datos, los cuales poseen importante información intrínseca sobre los sentimientos de dichos usuarios. Las fuentes de datos son muchas y variadas y, en ocasiones, las opiniones están embebidas en un gran documento de texto, o en foros o en blogs. La búsqueda e identificación de estos sentimientos son improbables de ser realizadas por un ser humano, en su totalidad, por lo que se necesitan sistemas de descubrimiento y sumarización de la información. El análisis de los datos, necesariamente debe incluir el procesamiento e interpretación de esa gran cantidad de opiniones, lo que origina varios desafíos. En esta instancia se requiere la separación en categorías de la carga de los sentimientos implícitos, provenientes de muchos usuarios, de una manera automática, lo que da lugar a una nueva disciplina denominada Análisis de Sentimientos o Minería de Opiniones.

6.3.1. Técnicas y tareas en análisis de sentimientos

Recordando el ejemplo de la *Sección 5.3.1, Clasificador lineal*, se observa que se trata de una clasificación de una frase en base a la opinión expresada

en ella. En la tabla 6.3 se muestran algunas de las técnicas que pueden ser empleadas por los sistemas de análisis de sentimientos para llevar a cabo las tareas. Al igual que en otras aplicaciones, existen técnicas de análisis de sentimientos *ad hoc* adaptadas a sus propósitos. Como ejemplo, la tabla indica que se pueden usar las técnicas de redes neuronales, *LSH*, algoritmos genéticos, *MapReduce*, árboles de decisión y técnicas de procesamiento de lenguaje natural, entre otras, para realizar una tarea de sumarización para el análisis de sentimientos.

Tarea / Técnica	Predictivo		Descriptivo				
	Clasificación	Regresión	Agrupamiento (Clústeres)	Reglas de Asociación	Correlaciones/ Factorizaciones	Detección de Anomalías	Sumarización
Ítemsets similares	V						
Redes neuronales	V						V
Ítemsets frecuentes	V						
TF-IDF	V						
LSH	V						V
Reducción de la dimensión	V				V		
Vecinos más próximos	V				V		
Máquinas vectores soporte	V				V		
Algoritmos genéticos	V				V		V
MapReduce	V				V		V
Árboles de Decisión	V						V
Regresión logística	V						
Naive Bayes	V						
Maximización Expectación	V						
Perceptrón	V						
Ensamblaje clasificadores	V						
Procesamiento de lenguaje natural	V						

Tabla 6.3: Técnicas y tareas en análisis de sentimientos

6.3.2. Algoritmo para análisis de sentimientos

Como se mencionó anteriormente, los algoritmos utilizados son de *ML* (por ejemplo *SVM* o *Naive Bayes* [37]), por lo que necesitan de un conjunto de entrenamiento para su construcción, con el objetivo de darles un valor a las nuevas frases ingresadas, basándose en la distribución que las palabras presenten y, si es necesario, con un peso asociado a ellas. Al ser un *ML* del tipo clasificación, se requiere disponer de atributos de los datos para poder efectuar correctamente su tarea. Algunas de ellos pueden ser:

- Términos y sus frecuencias.

- Términos especiales en un texto, como pueden ser adjetivos, que, en algunos casos, se tratan de un sentimiento implícito.
- Palabras de opinión y frases: son palabras que indican algo bueno, malo o frases que muestran una opinión pero en forma encubierta. Por ejemplo: «a caballo regalado, no se le mira el pelo».
- Dependencia sintáctica (dependencia entre palabras).
- Negaciones: palabras con negaciones tienen un impacto en la opinión.

En el *Anexo 9* se presenta la aplicación de técnicas de *ML* para el análisis de sentimientos.

6.3.3. Análisis de sentimientos colectivo

El análisis de sentimientos colectivo es una idea que consistiría en utilizar las herramientas disponibles para capturar sentimientos de un colectivo (por ejemplo, seguidores de un cuadro de fútbol) y luego desarrollar herramientas de aprendizaje automático que serían entrenadas para obtener sentimientos de dicho colectivo.

Una aplicación podría ser el combate a la violencia en el fútbol previniendo, en base a los sentimientos que se puedan identificar, actos delictivos de los seguidores conocidos como «barras bravas».

El diagrama de tal herramienta en el contexto de análisis de sentimientos se puede visualizar en la Figura 6.5.

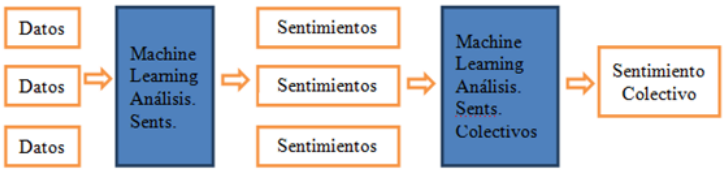

Figura 6.5: Diagrama para Análisis de Sentimientos Colectivos

6.3.4. Conclusiones

Es usual decir que, con el volumen enorme de tráfico de información que fluye a través de las diferentes redes sociales (*Twitter, Facebook* y *Blogs* temáticos, entre otros), es posible escuchar la voz del consumidor. El mecanismo utilizado para lograrlo se denomina análisis de sentimientos.

Los métodos usados en la clasificación se pueden usar en el análisis de sentimientos.

Caso de estudio

El análisis de sentimientos colectivo consistiría en utilizar las herramientas disponibles para capturar sentimientos de un grupo de personas con un interés común, para luego desarrollar herramientas de *ML* entrenadas para obtener una etiqueta sobre el sentir del colectivo. Sobre este tema no se ha encontrado información específica en la Web. El documento más similar recuperado fue un artículo denominado «*Sentiment Analysis by Collective Inspection on Social Media Content*». (Análisis de sentimientos por inspección colectiva de los medios sociales) [38] pero que trata un tema diferente.

6.4. Redes Sociales

El Análisis de las Redes Sociales (*ARS*) se focaliza en la estructura reticular de las relaciones sociales, buscando dar cuenta del efecto que provocan en el comportamiento de los individuos, en los distintos grupos sociales y en la sociedad en su conjunto. Abordar los procesos sociales desde una dimensión relacional implica dar cuenta de los patrones y estructuras de las relaciones sociales. El *ARS* busca estudiar dichos procesos desde su particular configuración en redes [39].

Del análisis de las redes sociales, con datos a gran escala, se puede obtener considerable cantidad de información. Una red social se puede modelar con un grafo, donde existen subconjuntos de nodos, que forman una comunidad, los cuales se encuentran altamente interconectados entre sí. Para identificar comunidades se pueden utilizar algunos de los algoritmos de clusterización. Sin embargo, las comunidades pueden compartir nodos, superponiéndose y, por lo tanto, en estos casos, las técnicas de *clustering* no son aplicables [4].

En la Figura 6.6 puede considerarse que existen dos sub-comunidades, D, E, F y D, G, F las cuales comparten los nodos D *y* F. Notar, como se mencionó, que los algoritmos de *clustering* no pueden resolver este problema.

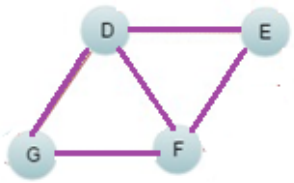

Figura 6.6: Sub-comunidades en un grafo

Las redes sociales tienen las siguientes características:

- Existe una colección de entidades que participan en ella (no exclusivamente personas, pueden ser correos electrónicos o documentos científicos, como una red colaborativa, red de teléfonos celulares, etc.).
- Existe al menos una relación entre los miembros (por ejemplo, «ser amigo de», la cual puede ser valorizada).
- Condición de localidad: si A está relacionado con B y con C, existe una alta probabilidad de que B y C estén relacionados entre sí.

Técnicas y tareas en redes sociales

Las técnicas que pueden ser utilizadas para las tareas involucradas en el análisis de las redes sociales se observan en la Tabla 6.4.

Tarea / Técnica	Predictivo		Descriptivo				
	Clasificación	Regresión	Agrupamiento (Clústeres)	Reglas de Asociación	Correlaciones/ Factorizaciones	Detección de Anomalías	Sumarización
Ítemsets similares	V						
Redes neuronales	V						V
Ítemsets frecuentes	V						
Análisis de grandes grafos	V						
LSH	V						V
Reducción de la dimensión	V				V		
Vecinos más próximos	V				V		
Máquinas vectores soporte	V				V		
Algoritmos genéticos	V				V		V
MapReduce	V				V		V
Árboles de Decisión	V						V
Regresión logística	V						
Naive Bayes	V						
Maximización Expectación	V						
Perceptrón	V						
Ensamblaje clasificadores	V						
Redes Bayesianas	V			V	V		V

Tabla 6.4: Técnicas y tareas para las redes sociales

6.4.1. Betweenness

Algunos algoritmos de búsqueda de comunidades en una red utilizan el concepto de Betweenness (intermediación).

Dada una arista (m, n) en un grafo, se define la intermediación como la cantidad de pares de vértices cuyo camino más corto contiene a dicha arista.

Ejemplo 6.2

Este ejemplo se basa en uno similar presentado en [4]. En la Figura 6.7, la intermediación de BC es 9. Los caminos más cortos que pasan por la arista BC son:

$< A, B, C >, < A, B, C, E >, < A, B, C, E, F >, < D, B, C >, < D, B, C, E >, < D, B, C, E, F >, < B, C >, < B, C, E > y < B, C, E, F >$

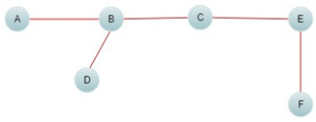

Figura 6.7: Betweennes

Notar que cualquiera de los nodos que forman la comunidad A, B, C, para comunicarse con la comunidad formada por los nodos C, D, F, necesariamente deben recorrer la arista BC. En total hay 3×3 caminos mínimos que contienen a BC. Con el mismo razonamiento, se puede calcular la intermediación de cualquier otra arista (en este ejemplo se puede comprobar que ninguna llega a 9). Es intuitivo pensar, que la arista BC define o separa dos comunidades del grafo.

6.4.2. Búsqueda de comunidades

En toda red social interesa la búsqueda de comunidades. Los siguientes métodos descubren comunidades.

- Algoritmo de *Girvan-Newman*, el cual se puede aplicar para descubrir comunidades. pero sin miembros en común (no consideran a nodos del grafo cuando pertenecen a más de una comunidad).
- Descubrimiento directo de comunidades (Ítemsets Frecuentes).
- Búsqueda de comunidades por partición de grafos.

Los siguientes algoritmos analizan el problema del solapamiento de comunidades.

- Búsqueda de comunidades que se superponen.
- Conteo de triángulos en redes sociales.

Estos algoritmos se describen detalladamente en el *Anexo 9*.

6.4.3. Análisis de las Redes Sociales

En términos analíticos, una red social es una estructura social compuesta por un conjunto finito de actores y configurada en torno a una serie de relaciones entre ellos, que se puede representar en forma de uno o varios grafos [39].

La topología de estas «redes reales» muestra una concentración de relaciones en unos pocos nodos, a la vez que la mayoría de los nodos de la red poseen una cantidad muy baja de relaciones. Estos nodos híper-conectados se denominan *hubs* [40].

Los *hubs* permiten comprender mejor la distribución en leyes de potencia de las relaciones en las redes reales. La regla de *Pareto* sirve para ilustrar la distribución de las relaciones en una red que presenta *hubs*. Por lo tanto, en las redes que presentan esta estructura, hay una tendencia a que los *hubs* concentren alrededor del 80 % de las relaciones dentro de una red, y que el restante 20 % de las relaciones se distribuya entre todos los demás nodos [39]. Es el mismo concepto que las colas pesadas, utilizado en los sistemas de recomendación (*Anexo 9*). La mayor parte de las redes presentan esta distribución donde los *hubs* tienen menor presencia que los otros nodos. En estadística, se dice que esta distribución de los grados es *right-skewed* [41]. Para los grafos dirigidos (como la Web, si se consideran los hipervínculos entre ellas como aristas), interesa el estudio de la distribución de los grados de salida y de entrada. Dados p_i que representa la fracción de los vértices que tienen grado de entrada i y p_j los de salida j, interesa efectuar el análisis de la distribución de probabilidad conjunta de los grados de entrada y de salida p_{ij}. Esta distribución es bidimensional y no puede representarse en un histograma como los mencionados de colas pesadas. Debería ser realizado con un grafo tridimensional y no debería descartarse la existencia de correlaciones entre las dos variables. Esta área de investigación no ha sido suficientemente explorada [41].

Una propiedad interesante en estas redes es que si la misma gráfica es construida usando escalas logarítmicas, la distribución se aproxima a una representación lineal [41], obteniéndose la Ec. (6.3).

$$p_k = Ck^{-a} \tag{6.3}$$

Por este motivo se dice que la Web sigue, aproximadamente, una distribución de Ley de Potencias.

6.4.4. Conclusiones

Las redes sociales a gran escala ofrece desafíos en todos los niveles: tecnológicos, sociales y científicos, entre otros. En el estudio de las redes sociales no es viable utilizar los métodos de agrupamientos planteados en el *Capítulo 5* si se trata de comunidades que se pueden superponer, debiéndose emplear otros algoritmos.

Línea de Investigación

Consiste en el análisis de la distribución de probabilidad conjunta de los grados de entrada y de salida de los nodos de una red representada por un grafo dirigido.

Capítulo 7

Caso de Estudio: Plan Ceibal

La masiva generación de datos puede ser percibida en nuestras vidas cotidianas. Su recolección, procesamiento, gestión y análisis son utilizados con el objetivo de generar conocimiento o información destinada a contribuir en el proceso de la toma de decisiones. El área de la educación no es ajena a esta realidad. Un ejemplo lo constituyen los datos originados a través de cursos en línea de alcance masivo.

El caso de estudio presentado busca identificar tendencias y agrupaciones en el uso del Plan Ceibal, el cual tiene una amplia cobertura en el territorio nacional y ofrece acceso a Internet a todos los estudiantes (niños y adolescentes) desde todos los centros educacionales (Primaria, Secundaria y Universidad del Trabajo). El desafío radica en que los datos no se restringen exclusivamente a registros de estudio en particular, sino que también abarcan accesos a cualquier otro tipo de sitios. Se proponen técnicas de agrupamiento para realizar el análisis, el cual se basa en un espacio definido por el número de conexiones en un intervalo de tiempo. El conocimiento obtenido puede ser útil como apoyo para la toma de decisiones (relativo al uso, previsiones y tendencias de dicha red). El Plan Ceibal diariamente genera un tráfico de varios *terabytes*, además de un registro de navegación, el cual es almacenado. Existen estudios relativos a usos de técnicas de agrupamiento aplicados a ambientes de educación. Algunos de ellos consolidan los diferentes tipos de algoritmos de agrupación aplicados en el contexto de la minería de datos educativos, para abordar diferentes problemas que se presentan en *EDM* (*Educational Data Mining*) [42] mientras que otros aplican técnicas clásicas o adaptan algoritmos a situaciones concretas [43]. La mayoría de dichas publicaciones son efectuadas consideran-

do *datasets* generados sobre aplicaciones específicas de estudio. El desafío que presenta este trabajo, es que los datos abarcan registros de acceso a cualquier tipo de sitio Web (entre ellos sitios asociados a la educación).

7.1. Alcance del Trabajo

Cada local de estudio tiene asignado un conjunto de direcciones *IP*, éstas son fijas, conocidas y están asociadas a su ubicación geográfica y al tipo de local de estudio. No se dispone de ningún medio de reconocimiento de los usuarios asegurándose, de esta forma, la protección de su identidad, pero dificultando las tareas de minería de datos. Los registros de navegación disponibles son almacenados en formato de texto plano y contienen ciertos atributos de las visitas a los sitios Web. De cada observación generada, se utilizan los atributos: *fecha, hora, IP y URL solicitada*. Los restantes campos (que no son de interés para el estudio) los crean los propios sistemas: *firewalls* e Internet, entre otros. A pesar de que la cantidad de dimensiones del problema es reducida y que no es posible obtener ninguna característica personal de los usuarios, se espera poder relacionarlos o buscar patrones en la utilización de los recursos. En general se pretende encontrar patrones o características que permitan describir las siguientes situaciones:

- Mediante el análisis de la red, probar la existencia de horarios en el que se observan aproximadamente la misma cantidad de conexiones en centros de estudio en determinados días.
- Variación de la cantidad de conexiones en el correr del año para los distintos centros de estudio (observados en su totalidad o relacionados con visitas a sitios de estudio).
- Existencia de aglomeraciones de conexiones en horario de clases o de usuarios que se conectan a un sitio Web de determinado tipo, en relación a sus lugares de residencia (esta característica habitualmente está asociada a la situación económica y social).

7.2. Objetivo

El objetivo principal es aplicar técnicas de analítica sobre los datos obtenidos de los archivos que se generan diariamente por la navegación de los

usuarios de la red, proporcionando indicadores destinados al soporte para la toma de decisiones. A continuación se enumeran los objetivos generales.

- Relación de visitas a sitios con material educativo comparados con: visitas a sitios de diverso contenido; entre turnos a lo largo del día y en distintas épocas del año. Por ejemplo, detectar la utilización de sitios Web de esparcimiento comparado con sitios de estudio, por hora, día o períodos de tiempo.
- Detectar los horarios de uso de la red que son de utilización máxima o intermedia. Asimismo determinar los horarios que son de escasa o nula utilización a nivel nacional.

Para el presente estudio y siguiendo los lineamientos de los objetivos generales, se definen los siguientes objetivos específicos:

- Búsqueda de patrones en el uso de la red según los horarios.
- Exploración de algunos clústeres que respondan a los puntos señalados en los objetivos generales.
- Detectar *outliers*(centros de estudio que difieren en la utilización habitual de la red, en relación a los horarios).

7.3. Análisis y metodología empleada

El trabajo se organiza llevando a cabo las siguientes tareas:

Etapa 1: Colección de los datos, sanitación, estudio de los datos faltantes y medidas para su solución.

Etapa 2: Análisis exploratorio de los datos

Etapa 3: Extracción de información por medio de herramientas estadísticas.

Etapa 4: Obtención de conocimiento que apoye a la toma de decisiones.

7.3.1. Etapa 1: Colección y limpieza de los datos

Los datos son registros que se almacenan diariamente en archivos de texto plano (formato *.log*). Por día se generan entre $200MB$ y $1GB$ de registros (aproximadamente un total de *1,5* millones de entradas en promedio). Un ejemplo de registro es:

$$2016.2.24\ 3:23:12\ -\ 10.28.239.9$$

$$http://superfighterweb.com/movil/explorar.php\ GET\ 871\ 0\ 1200\ text/html\ -$$

Por medio de una rutina desarrollada en *Python* y aplicando expresiones regulares al atributo «*URL solicitada*», se eliminan registros que son irrelevantes. Por ejemplo, actualizaciones de sistemas operativos, auditorías, notificaciones de antivirus, entre otros. Un ejemplo de las expresiones regulares empleadas para este caso es:

$$'\backslash S+clients3\backslash S+','\backslash S+windowsupdate\backslash S+','\backslash S+mcafee\backslash S+','\backslash S+ 216.239.32.20/generate_204\S+','\backslash S+clients1\backslash S+','\backslash S+connectivitycheck\backslash S+','\backslash S+URLMOD\backslash S+','\backslash S+msftncsi\backslash S+'$$

La información contenida en las *URLs* correspondientes a los strings utilizados, se refiere a procesos automatizados y fueron proporcionadas por los técnicos del Plan. Los datos resultantes, se almacenan en una base de datos *MongoDB*. Luego de finalizado el proceso, se ejecutan rutinas (programadas en *Javascript*) que, empleando nuevas expresiones regulares (las cuales definen las categorías para las *URLs* accedidas), asocian los registros de navegación con los locales a los que pertenecen. Los locales son identificados por un código y, adicionalmente, se dispone de una tabla que indica sus características: nombre y ubicación entre otros atributos. Este proceso genera archivos en formato *json* (uno para cada día y para cada expresión regular detectada en la búsqueda en la base de datos). Debido a que los archivos *json* obtenidos tienen anidaciones, los datos son «aplanados» pasándolos a otra base de datos.

Se dispone de datos correspondientes a *214* días desde *febrero 2016 a mayo 2017*, irregularmente distribuidos, presentando considerables períodos de tiempo sin datos. Los locales de estudio son aproximadamente *1100*. Luego del procesamiento, se dispone de *3:300.006* registros, que suman *122:205.324* conexiones. Una primera sumarización de los datos permite clasificarlos en categorías, obteniéndose cierta cantidad de registros para cada una de ellas (Tabla 7.1, Categorías de los sitios y cantidades).

Categoría	Cantidad
Redes sociales	69:526.210
Sitios de Búsqueda	32:636.343
Utilitarios	7:752.560
Juegos	5:044.463
Estudio	4:716.237
Apoyo educación	1:445.627
Periódicos	1:083.627

Tabla 7.1: Categorías de los sitios y cantidades

En la Figura 7.1, se visualiza, en la primer imagen, las proporciones de las categorías exploradas. En la misma figura, la segunda imagen indica las proporciones de las expresiones regulares para la educación y la tercer imagen para las redes sociales. En esta última es interesante observar que *Instagram* tiene un mayor uso que *Facebook* (para la muestra considerada y considerando las características de los usuarios del sistema).

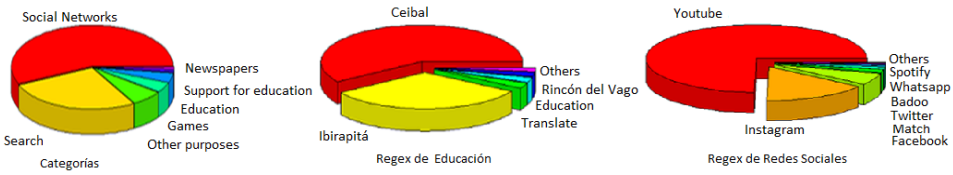

Figura 7.1: Sumarizaciones iniciales

7.3.2. Etapa 2 Análisis exploratorio de los datos

Para la etapa de análisis exploratorio de los datos, es empleado el lenguaje *R* (versión *3.6.1*) conectado a la base de datos *MongoDB* (versión *3.2.10*). A los efectos de evaluar si existe una tendencia al agrupamiento de los datos se utiliza el *estadístico de Hopkins* [10] el cual indica qué tan alejados se encuentran datos de la muestra, seleccionados aleatoriamente, de presentar una distribución uniforme (cuanto más alejado de tal distribución mayor es la probabilidad de que los datos se agrupen). Para ello se utiliza la función *hopkins* del paquete *clustertend* disponible en *R* obteniéndose valores cercanos a *0,003*, lo que es indicativo de presencia de agrupamientos (el estadístico varía de *0 a 1 y 0,5* indica distribución uniforme).

Con el propósito de obtener cualitativamente alguna información de los datos, se considera una muestra de ellos. La determinación del tamaño de una muestra en estudios cualitativos generalmente es un juicio subjetivo, tomado a

medida que la investigación avanza, recomendándose el concepto de saturación para lograr un tamaño de muestra apropiado (saturación se le denomina a la situación en la cual, agregar nuevas observaciones, no mejora las perspectivas de obtener nueva información) [44]. A tales efectos, se consideran los registros generados en el acceso al Plan, en un determinado día y se contabilizan las conexiones a Internet en cada hora y en cada centro de estudio, analizando su distribución empírica. Observando que los datos disponibles cubren alrededor de un año, se consideró conveniente, a los efectos de obtener muestras representativas, repetir los estudios eligiendo días aleatoriamente, generando nuevas muestras, hasta lograr la saturación. En esta etapa, también se pretende identificar candidatos a ser posibles *outliers* durante el proceso.

La Tabla 7.2, Conexiones por local y por hora en una fecha determinada, es un ejemplo de cómo se dispondrán los datos: las filas representan los centros de estudio, las columnas los rangos de horas y en las celdas se encuentran la cantidad de conexiones de cada centro de estudio por rango de horas.

Local / Horas	0000 - 0059	0100 - 0159	0200 - 0259	.....	2200 - 2259	2300 - 2359
23456	2	1	5		500	475
32345	3	5	3		651	597
13246	1	3	0		321	384

Tabla 7.2: Conexiones por local y por hora en una fecha determinada

7.3.3. Etapa 3: Extracción de información

Con el objetivo de proporcionar claridad al plantear los casos de estudio, se define un procedimiento que proporciona una notación adaptada al problema denominado clústeres, el cual tiene $n + 1$ argumentos. El cabezal del procedimiento $clusteres(x; x_1, x_2, ..., x_n)$ indica la aplicación de cualquier método de agrupamiento para determinar los clústeres x, a partir de los atributos $x_1, x_2, ..., x_n$ de los datos. Dichos atributos no son necesariamente independientes entre sí (puede ocurrir que algún campo se genere a partir de otros). En caso de que los datos sean dinámicos (variables con el tiempo), se agrega al tiempo como un nuevo argumento, utilizándose la notación $clusteres(x; x_1, x_2, ..., x_n, t)$.

Casos de estudio

clusteres(horas de uso; cantidad de conexiones): Indica la búsqueda de horarios en que la cantidad de conexiones es similar (corresponde a uno de los objetivos específicos planteados, determinación de patrones en el uso de la red según los horarios).

clusteres(horas de acceso a la red; locales de estudio): Con datos similares a los de la Tabla 7.2, se busca identificar candidatos a ser agrupamientos indicando los clústeres formados por las observaciones en las cuales se contabiliza, aproximadamente, la misma cantidad de conexiones en un determinado día, en varios locales de estudio. Se busca identificar locales que presentan un comportamiento parecido en lo que se refiere a la conectividad en determinados horarios durante el día, basándose en la cantidad de conexiones por hora. En este sentido dos locales se consideran próximos si la cantidad de conexiones en cada intervalo de tiempo son parecidas con un umbral de tolerancia. Se debe considerar que el espacio definido no es euclídeo, al momento de definir el algoritmo de *clustering* a ser utilizado. Como se mencionó en la *Sección 5.8*, un espacio es euclidiano si el promedio de cualquier conjunto de sus puntos pertenece al espacio [4].

La discretización del tiempo en intervalos definidos en horas es justificada considerando que las horas del día indican las actividades de las personas. Por ejemplo, el turno matutino de los alumnos de primaria comienza *8 a.m.* y finaliza a las *12* del mediodía y el turno vespertino comienza *13 p.m.* y finaliza *17 p.m.*, condicionando el resto de las actividades. Aunque no debe descartarse realizar el estudio con otros extremos de intervalos o considerar una mayor granularidad.

Se considera que un punto x pertenece a un clúster cuyo clustroide (un punto, existente o no, que oficia como centroide en clústeres en espacios no euclídeos) es y, si $d(x, y) \leq \delta$, es decir si la diferencia entre la cantidad de sus conexiones es menor que un umbral a ajustar.

Método de clustering a ser utilizado

Es necesario utilizar un método que permita la adaptación de los nuevos datos generados continuamente. Por lo tanto, se descartan los métodos jerárquicos que son de orden n^3 [4]. En los métodos por asignación de puntos, el algoritmo *k medias* se descarta porque es recomendado para espacios euclídeos [4]. Cuan-

do los datos aumenten en gran cantidad, para los espacios que no son euclídeos, tratándose de grandes *datasets*, se sugiere utilizar el algoritmo *GRGPF* (que se adapta a este tipo de problemas), o métodos de procesamiento en paralelo utilizando *MapReduce* [4]. En base a las consideraciones previas, en esta etapa se procedió a utilizar paquetes de *Data Mining* disponibles en *R*. Dicho lenguaje tiene implementados los algoritmos *PAM* (*Partitioning Around Medoids*) y *CLARA (Clustering Large Applications)*. Entre ambos se decide la utilización de *CLARA* debido a que se adecúa al procesamiento de grandes conjuntos de datos. *CLARA* efectúa la búsqueda de clústeres en base a medoides (puntos ya existentes del clúster), prescindiendo de la búsqueda de centroides, que no tendría sentido en el tipo de espacio considerado.

7.3.4. Etapa 4: Apoyo a la toma de decisiones

De los indicadores planteados en la *Sección 7.2, Objetivo*, se mencionan algunos elementos que pueden brindar información de utilidad:

- Tendencias en el acceso a la red: búsqueda de patrones en su utilización según los horarios y variación de la cantidad de conexiones o tráfico de sitios relacionados con estudio a lo largo del año.
- *Outliers*: centros de estudio que difieren en la utilización de la red en relación a los horarios, lo que permitiría identificar centros que utilizan mínimamente el servicio o centros que lo sobre utilizan, lo cual podría estar asociado a la presencia de intrusos en la red.
- Horarios comunes a todos los centros de estudio donde la utilización es mínima, lo cual serviría para programar tareas de mantenimiento en el servicio sin afectar a los usuarios, entre otras posibilidades.

7.4. Resultados Obtenidos

Resultados de la Etapa 2 (Análisis Exploratorio de los datos)

Los resultados comprenden el análisis cualitativo de los datos y la búsqueda de *outliers*.

Análisis cualitativo de los datos

Como ejemplo del estudio de una sola muestra de registros, se considera el día *02/03/2016* disponiéndose de los datos de *89* locales. Por medio de métodos estadísticos descriptivos se obtiene una distribución empírica en el uso de la red, reflejada en la gráfica de la Figura 7.2. En la primer imagen, *Histograma de cantidad conexiones/hora*, se puede comprobar, como era de esperarse, que en los horarios de *8 a 17* (horario de clases) el uso es mayor y se reduce en el resto de las horas.

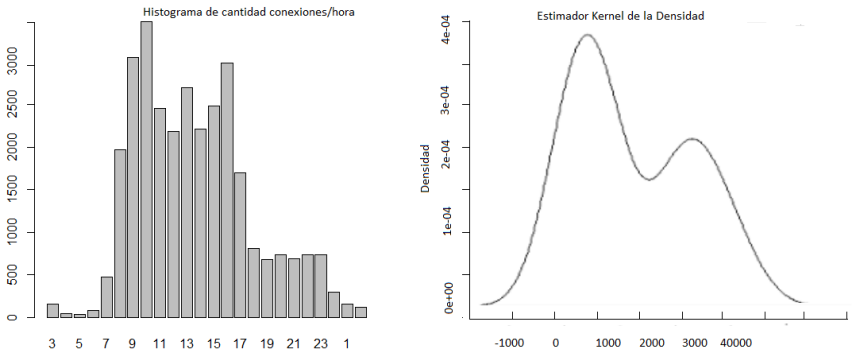

Figura 7.2: Utilización de la Red para un día específico

Debido a que el histograma es un método pobre para determinar la forma de una distribución (dado que puede ser fuertemente afectado por las cubetas elegidas) y, para suavizar la gráfica, se utiliza la función densidad de núcleo *(Kernel Density Plot)*, disponible en *R*. A los efectos prácticos se consideraron 20 muestras generadas aleatoriamente sin reposición con mil observaciones cada una. En la segunda imagen de la Figura 7.2, Estimador *Kernel* de la Densidad, se puede observar qué tan alejado estarían las observaciones de una distribución normal, tendiendo a asemejarse a una mezcla de gausianas.

Búsqueda de outliers

Previo a cada procesamiento se eliminaron las observaciones que pudieran considerarse anómalas. Para la identificación de estos datos anómalos se consideró, de cada muestra, los locales, agrupados por fecha y hora. Esto es debido a que días de la semana diferentes tienen comportamientos diferentes. Por ejemplo, la actividad de un domingo difiere en sus características de otros días, pero posiblemente sean similares para otros centros de estudio. Este método se centra en una medida de distancia apropiada (diferencia del promedio de

la cantidad de conexiones). Por ejemplo, si se considera la hora *4 a.m.*, generalmente se observa que la actividad es reducida en casi todos los centros de estudio (el máximo detectado en varias observaciones es de *26*). La presencia de un local con una actividad muy alta indicaría la presencia de un *outlier*. A partir de los datos se calculó su media y desviación típica y se considera anómalo todo dato que está fuera del intervalo $[\mu(x) - 3\sigma(x), \mu(x) + 3\sigma(x)]$. A tal fin, se utiliza las funciones disponibles en *R*, $mean(x)$ y $sd(x)$ en cada uno de los atributos (la hora del día en que se considera la cantidad de conexiones). Los *outliers* son retirados de la muestra y luego se normalizan los datos por medio de la función *scale* con una media de *0* y desviación estándar *1*.

Resultados de la Etapa 3 (Extracción de información utilizando herramientas estadísticas)

Los resultados obtenidos en el análisis exploratorio sugieren que las observaciones se ajustan a un patrón al considerarse la cantidad de conexiones por hora en los centros de estudio, procediéndose a la búsqueda de agrupamientos en el dataset, cuyos resultados se presentan a continuación.

clusteres(horas de uso; cantidad de conexiones)

Se utiliza la funcionalidad *clara* del paquete *CLARA* de *R*. La invocación del algoritmo se efectúa por medio de: $clara(dataset, k, samples = n)$, la cual aplica el algoritmo al *dataset* con n muestras de los datos buscando k clústeres y considerando a los medoides como sus centros. A los efectos de tener una idea del valor de k, se grafica un dendograma que proporciona una visualización de los posibles clústeres en la Figura 7.3, Clústeres de conexiones por hora. Se aprecia que la actividad de conexiones se puede agrupar en dos clústeres. Se ejecuta el algoritmo *CLARA* sobre el *dataset* pudiéndose visualizar, en la segunda imagen de la misma figura, los dos clústeres claramente diferenciados: el de mayor densidad corresponde a los horarios centrales del día (8 a 16 horas), que se corresponden con los horarios de actividad educativa (se utilizó la funcionalidad $fvizcluster$ disponible en el paquete *factoextra* de *R*).

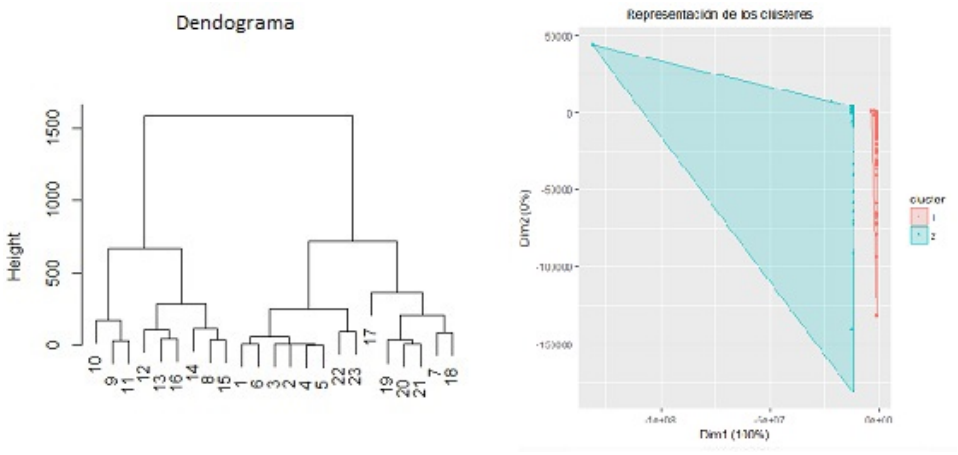

Figura 7.3: Clústeres de conexiones por hora

clusteres(horas de acceso a la red; locales de estudio)

Los resultados obtenidos utilizando *CLARA* sobre un conjunto correspondiente a *374* locales de estudio permiten establecer la presencia de tres clústeres y un *outlier* (Figura 7.4). Los clústeres están compuestos por *97, 137 y 139* locales.

Figura 7.4: Clústeres de centros de estudio con tráfico de actividad educativa similar

Resultados de la Etapa 4

Las tendencias en el acceso a la red, como podría ser la búsqueda de patrones en su utilización según los horarios, fue analizado en la Etapa 4. La búsqueda de centros de estudio que difieren en la utilización de la red en relación a los horarios (*outliers*) fue tratado en la Etapa 2.

Con respecto a la variación de la cantidad de conexiones o tráfico de sitios relacionados con estudio a lo largo del año, la Figura 7.5, representa el comportamiento de la cantidad de conexiones para cada día de la muestra (sean o no sitios relacionados con estudio). Aproximadamente refleja el incremento del tráfico en el primer semestre del año lectivo *2016*, el receso a partir de noviembre y en verano, y la vuelta a la actividad en marzo del año *2017*. La línea vertical roja indica el salto en los días disponibles (de mayo a noviembre no existen registros).

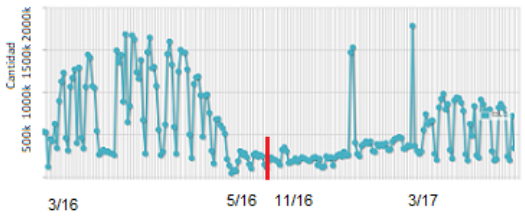

Figura 7.5: Tráfico general

La Figura 7.6, representa el tráfico en los días de la muestra para actividades de estudio. La línea vertical roja, en la figura, indica que de mayo a noviembre no se dispone de registros. Se destaca el hecho de que de noviembre de *2016* a marzo de *2017*, se visualiza escasa actividad coincidente con las vacaciones lectivas.

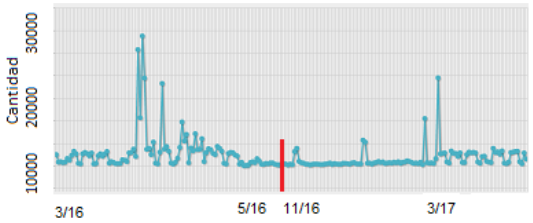

Figura 7.6: Tráfico para Sitios de Estudio

Finalmente, en la la Figura 7.7, se grafica el comportamiento de las expresiones regulares utilizadas, mediante la representación de la cantidad de veces que la expresión regular empleada no fue detectada en el resumen de cada día. En ella se puede visualizar que, a lo largo del tiempo, existe una tendencia a la baja de dicha cantidad (posiblemente debido al aumento del uso del sistema).

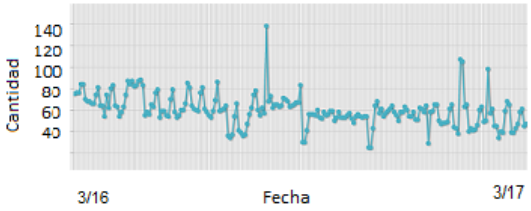

Figura 7.7: Comportamiento de las expresiones regulares

7.5. Aplicabilidad de Big Data en el Plan Ceibal

El abordaje inicial del caso de estudio del Plan Ceibal se enfocó a una generación masiva de datos. A tales efectos, para su almacenamiento, se utilizó una base de datos no relacional (*MongoDB*). A pesar del enfoque original, los datos disponibles para el estudio fueron colectados manualmente, entregados en mano en soportes digitales, debido a limitaciones de infraestructura. Por tal motivo, el estudio se orientó bajo una óptica de *Data Mining* clásica, pero fijando las bases para que fuese escalable a *Big Data*. Al respecto, se señalan las siguientes consideraciones:

- El tráfico diario generado son de varios *terabytes* (datos de octubre del año 2017) previéndose un aumento considerable y, en consecuencia, se espera un incremento en los registros de navegación con los cuales se realizó el trabajo. El empleo de una base de datos no relacional deja abierta la posibilidad de un almacenamiento masivo una vez aumente la generación de los datos.
- La analítica utilizada deberá adaptarse a los datos masivos, empleando algoritmos apropiados que se enfoquen a la obtención de sumarizaciones de los datos, las cuales proporcionen una idea de las tendencias en el uso de la red.
- En caso de tareas que requieran un aprendizaje no supervisado, se podría optar por técnicas de *clustering* tales como *CURE*, algoritmos *multistage*, entre otros.
- Si se tratase de modelos de aprendizaje supervisado se debería tener en cuenta el Análisis predictivo en *Big Data* presentado en el *Capítulo 3*.
- Adicionalmente, se puede considerar el escenario en que los registros van

llegando en forma de *stream* con el procesamiento respectivo como se estudió en el *Capítulo 2*.

- El análisis de la presencia de *outliers* (por ejemplo, centros de estudio puntuales que difieren en la utilización de la red en relación a los horarios), podría dirigirse hacia la identificación de pequeñas poblaciones de centros de estudio que difieran en el uso de la red respecto a la gran mayoría. La detección de poblaciones de este tipo, permitiría obtener patrones ocultos en el aprovechamiento de los recursos en lugar de considerarlos como datos anómalos.

7.6. Conclusiones

De los resultados presentados en la *Sección 7.4*, se puede considerar que los objetivos específicos fueron alcanzados. Las búsquedas de clústeres propuestas en los casos de estudio son, prácticamente, resultados directos de la aplicación de algoritmo *CLARA*.

En el *Capítulo 8, Conclusiones y Trabajos Futuros*, se presenta la evaluación de las herramientas utilizadas y propuestas de casos de estudio detectados.

Capítulo 8

Conclusiones y Trabajo Futuro

Este capítulo resume las conclusiones del trabajo de Maestría y se presentan los casos de estudio que podrían resultar de interés, obtenidos durante su confección.

8.1. Conclusiones

Se presentan las conclusiones generales del trabajo y las conclusiones arribadas en el caso de estudio.

8.1.1. Conclusiones generales

Dentro de los objetivos planteados, una apreciación general permite afirmar que fueron cumplidos.

El primer objetivo consistió en un exhaustivo análisis que posibilitara contar con una descripción de las técnicas para llevar a cabo las tareas de analítica sobre *Big Data*. El análisis de los modelos, su evaluación, las herramientas de aprendizaje automático y las tareas y técnicas analizadas, hacen posible abordar un amplio espectro de las tareas, contando con un detallado fundamento de las metodologías empleadas por los algoritmos. Se realizó un profundo relevamiento de las técnicas y de las tareas. En relación a las tareas, se estudiaron la clasificación, regresión, agrupamiento, reglas de asociación, búsqueda de *outliers*, sumarizaciones y correlaciones. En cuanto a las técnicas se desarrollaron algoritmos de aprendizaje automático, búsqueda de artículos similares, itemsets frecuentes, vecinos más cercanos, árboles de decisión, *boosting*, *clustering*, redes neuronales artificiales y métodos estadísticos de aprendizaje.

El segundo de ellos abarcaba la elaboración de una clasificación de las técnicas de analítica que se encuentran en el *Capítulo 1*. Junto con ella, se introdujo el concepto de técnica y de tarea. Las tareas pueden ser interpretadas como el trabajo que hay que realizar y las técnicas se pueden entender como las herramientas para llevarlo a cabo.

El tercer objetivo, consistente en el caso de estudio aplicado al Plan Ceibal es explicado en la próxima sección.

A lo largo del trabajo, se presentó la técnica de *stream*, la cual permite realizar cálculos de datos a gran escala y es la base de soluciones para el procesamiento de *Big Data*.

Posteriormente se analizaron los principales conceptos del modelado estadístico, los tipos de errores y su evaluación.

Finalmente, se presentó en la segunda parte del trabajo el aporte donde se identifica el concepto de aplicaciones de las tareas y técnicas mencionadas. En tal sentido se estudió el *Page Rank*, sistemas de recomendación, análisis de sentimientos y redes sociales (*Capítulo 6*).

8.1.2. Conclusiones del caso de estudio: Plan Ceibal

En relación a la evaluación de las herramientas utilizadas y considerando que fueron empleadas luego de haberse descartado otras, se considera que para las dimensiones que presentaban los datos, resultaron adecuadas. Al momento de la selección de la base de datos para el procesamiento de los datos, se descartó el uso de las relacionales y se optó por la utilización de *NoSQL* (*MongoDB*) debido a sus posibilidades de escalabilidad (pensando en futuros trabajos) y por el hecho de ser capaces de crecer, en número de máquinas, con pocos recursos. Otro factor que se tuvo en cuenta es la posibilidad de optimizar consultas para la lectura de grandes cantidades de datos. Finalmente, se consideró, al momento de la selección de las herramientas, de que tanto *R* como *MongoDB* son de uso libre. Para el procesamiento inicial se descartó el uso de bases de datos relacionales debido a que sus posibilidades de escalabilidad se limitan a residir en grandes máquinas con costosas licencias o que directamente no se pueden adaptar a los casos en que los datos se generen a escala de *Big Data*.

Los resultados obtenidos fueron presentados a las responsables técnicos del plan Ceibal.

8.2. Trabajos Futuros

Se presentan las posibilidades de tareas futuras bajo dos ópticas: para el caso de estudio (Plan Ceibal) y para las líneas de investigación y posibles casos de estudio identificados durante el desarrollo de la tesis.

8.2.1. Caso de estudio del Plan Ceibal

Haciendo uso de la notación introducida en el *Capítulo 7*, se presentan los siguientes tareas que podrían ser de interés analizar.

clusteres(horas de acceso a la red, local de estudio, tiempo)

El último argumento, tiempo, indica que se consideran los registros de varios días. La información obtenida puede indicar para un local determinado, si existen horarios fijos durante el día en que la cantidad de usuarios es máxima u horarios en que se mantiene constante la cantidad de conexiones, o que, por el contrario, se detectan fluctuaciones en el uso del servicio (cualquiera sea el caso, la información permitiría obtener conclusiones relativas al uso de la red). La identificación de tales clústeres determinará las horas de acceso a la red más utilizadas (horas pico) en el correr de los días a nivel nacional o en una región geográfica determinada. El estudio se podrá conducir cuando se disponga de más días de registros pudiéndose emplear series temporales.

clusteres(locales de estudio, horas de acceso a la red, tiempo)

Identifica centros de estudio que se agrupan a determinada hora para acceder a la red en el correr de varios días. En este caso la función distancia entre centros de estudio a y b se define como $d(a, b) = 0$ si $a = b$ y $d(a, b) = 1$ si $a \neq b$.

clusteres(tipo sitio web, horas de acceso a la red, ubicación (status social), tiempo)

Se obtienen los tipos de sitio Web accedidos en función de su situación socio económica. Al disponerse de otros atributos en los datos tales como sitio Web visitado y ubicación geográfica de los centros de estudio, es posible realizar otros análisis. Estos nuevos campos proporcionan mayor riqueza a la información obtenida considerando que, en general, el lugar de residencia se relaciona con su situación socio económica. Por ejemplo se pueden obtener los

tipos de sitio Web accedidos en función de su ubicación. Asimismo, se puede determinar si existe una clusterización de conexiones en horario de clases o de usuarios que se conectan a determinados sitios en la Web, de acuerdo al barrio. En este sentido, se recopiló la información necesaria para identificar ciertos patrones. Los resultados fueron presentados en el *workshop LALA 2018, Learning Analytics Latin America, "Dibujando el mapa de tráfico de redes escolares en línea"*, como se mencionó en el Capítulo 1.

Tareas para soporte a la toma de decisiones

A continuación se enumeran otras tareas desprendidas de los objetivos generales del caso de estudio y que puedan ser de utilidad para las autoridades del Plan Ceibal al momento de la toma de decisiones. Es de destacar que son resultados inmediatos y que es posible abordar la búsqueda de conocimiento más elaborado. Por ejemplo, la cantidad de conexiones en un centro de estudio no es suficiente en caso de relacionar el uso de la red con el rendimiento escolar. Para ello debería utilizarse la razón cantidad de conexiones/cantidad de alumnos.

- Investigación de patrones que relacionen empleo de la red o determinados sitios de la Web con locales considerados exitosos a nivel educacional. En este sentido, un indicador que sería interesante obtener es la identificación de alguna utilización especifica del Plan que se pueda relacionar con locales de estudio cuyos niveles de aprobación o calificaciones obtenidas por los alumnos son aceptables. A tales efectos se debería disponer de datos adicionales, como ser la información de cantidad de alumnos de los centros de estudio, entre otras características, para evitar conclusiones que no contemplen la realidad completa de los estudiantes.
- Análisis del tráfico por medio de series temporales. Se trata de conducir una investigación, empleando esta técnica, que permita realizar pronósticos de la utilización de la red y determinar características estacionales, a los efectos de extraer conclusiones relativas a las necesidades del servicio.
- Proporción de visitas a sitios de estudio comparados con visitas a otros sitios en general; cotejando entre los diferentes barrios de Montevideo; o el interior del País con las capitales departamentales.
- Determinar sitios Web preferidos por centro de estudio, por zonas geográficas y estrato social

- Diferencias que se pueden identificar entre departamentos del interior y Montevideo en cuanto al uso del servicio (acceso a la red).
- Tendencias del uso distinguiendo entre turnos a lo largo del día y entre distintas épocas del año.
- Utilización de sitios Web de esparcimiento comparado con sitios de estudio, por hora, día o períodos.
- Barrios o lugares donde se estudia más (contabilizando cantidad de conexiones a sitios de estudio).
- Detectar otros horarios de uso de la red que no son pico o son de escasa utilización a nivel nacional (y emplear esos momentos en caso de tener que hacer mantenimiento de *hardware*).
- La búsqueda de sitios relacionados con temas educacionales a través de expresiones regulares, permite agruparlos según su frecuencia de visitas y de acuerdo al tipo de centro de estudio (primaria o secundaria). Esta información será de utilidad al momento de evaluar los materiales de soporte de los cursos incluyendo sitios Web educativos. Adicionalmente, contextualizando dicha información con otros datos (ubicación geográfica u horario de acceso), sería posible generar conocimiento más real del uso de los recursos en los diferentes departamentos del país. Lo anterior podría enriquecerse estudiando los comportamientos en los distintos años. Para ello, se debería disponer de más registros que se recopilarán en los años venideros.
- Relacionado con el punto anterior, se puede entrenar a un clasificador por medio de algoritmos de aprendizaje automático, para evaluar la performance de las expresiones regulares utilizadas (lo cual redundará en la calidad de los clústeres obtenidos).

8.2.2. Casos de estudio propuestos y líneas de investigación

Durante el proceso de desarrollo de la documentación, se confeccionaron algunos casos de estudio y se extrajeron, de la literatura consultada, ciertas líneas de investigación.

Casos de estudio identificados

Introducción

- La clasificación de las tareas y métodos del *Capítulo 1*, no comprende a los métodos bayesianos. Se propone incluirlos para disponer de un ordenamiento más completo.

Imputación

- Estudio de técnicas de *clustering* aptas para grandes *datasets*, adaptando *ML* para situaciones de datos incompletos (Ampliado en el *Capítulo 2*).
- En el procedimiento de imputación estandarizada, utilizar *k-shingles* para las comparaciones, con un k adecuado de tal forma que si todos los *k-shingles* coinciden con una palabra de un diccionario con los términos estándar, se imputa o se descarta. Además establecer el valor del k, utilizando la distancia edit (Ampliado en el *Capítulo 2*).
- Analizar en *Big Data* la forma de establecer que los atributos que faltan pueden estar correlacionados (en la población o en sub-poblaciones de los datos). También se propone el estudio estadístico de alguna particular distribución de probabilidad de ocurrencia (Ampliado en el *Capítulo 2*).

Outliers en Big Data

- Desarrollar técnicas para que los *outliers* de *Data Mining* tradicional puedan ser pequeñas poblaciones en *Big Data* y desarrollar la teoría y técnicas para detectar datos anómalos en contextos de *Big Data* (Ampliado en el *Capítulo 2*)

Modelos

- Definición de un modelo de aprendizaje automático si existe una alta variabilidad en los datos (desde el punto de vista del entrenamiento y testing del modelo, investigándose bajo el punto de vista de los sesgos originados en las diferentes fuentes y también a la aparición de nuevos atributos) (Ampliado en el *Capítulo 3*).

Regresión

- Debido a que la masividad de los datos, provenientes de diferentes puntos de tiempo, usando diferentes tecnologías crea sesgos estadísticos se propone el análisis del impacto de estos sesgos al considerar modelos de regresión como hipótesis (Ampliado en el *Capítulo 4*).

Algoritmos de recuperación de ítems (Ampliado en Sección 5.2)

- Subsecuencias frecuentes y sistemas de recomendación.
- Confiabilidad y cobertura en subsecuencias frecuentes

Árboles de decisión (Ampliado en Sección 5.5)

- Se plantea utilizar *MapReduce* para el ensamble de los árboles con pocas características y poca profundidad.

Combinación de técnicas para tareas descriptivas (Ampliado en Sección 5.6)

- Estudiar la factibilidad de extender el concepto de combinación de clasificadores a tareas descriptivas. Se trata de evaluar si, dado un modelo de algoritmo usado en esas tareas, se mejoran los resultados obtenidos al combinarse con los resultados de otros algoritmos del mismo tipo. El objetivo es adaptar la idea a una búsqueda optimizada de clústeres.

Redes neuronales (Ampliado en Sección 5.7)

- Aplicar *MapReduce* en implementaciones en paralelo para *RNA*.

Sistemas de recomendación basados en redes neuronales (Ampliado en Sección 6.2)

- Se presentó la viabilidad del diseño de una red neuronal para establecer una separación en regiones para la predicción del *score* de un ítem para un determinado usuario, construyéndose un *ranking* de preferencias.

Análisis de sentimientos (Ampliado en Sección 6.3)

- El análisis de sentimientos colectivo consistiría en utilizar las herramientas de *ML* para capturar sentimientos de un grupo de personas con un interés común. Se propone investigar la viabilidad de esta aplicación.

Métodos bayesianos (Ampliado en Anexo 4)

- En el *Capítulo 3* se nombraron los métodos estadísticos de aprendizaje, los cuales son expuestos en el *Anexo 4*. Estos métodos bayesianos permiten trabajar con incertidumbres. El caso de estudio que se describe es el de aplicar métodos bayesianos a las tareas que se presentaron en el *Capítulo 1*, a través del estudio de las posibilidades de adaptar los algoritmos, también aludidos en dicho anexo, a un enfoque bayesiano. A lo largo del trabajo, en algunos algoritmos, se tuvo en cuenta la asignación de probabilidades a las observaciones (árboles de decisión, por ejemplo) pero se propone ampliarlo más metódicamente.

Se entiende que proveer de la información de incertidumbre si está disponible, a los métodos, no debería empeorar su performance. Por ejemplo, si se considera el método de vecinos más cercanos, y se dispone de la información de la distancia y de una probabilidad asociada de que el punto a clasificar se encuentre próximo a determinados vecinos, el método tendría que ser más eficiente al momento de la búsqueda (es un caso de k vecinos ponderados, pero con la restricción de que los coeficientes, correspondientes a los pesos, corresponden a una distribución de probabilidad).

Datos Unarios (Ampliado en el Anexo 9)

- En los *SR* basados en datos unarios, se encuentra el problema de la imposibilidad de interpretación y del arranque en frío.

Líneas de investigación estudiadas durante el trabajo

Línea de investigación 1: Redes neuronales (ReLUs, Ampliado en la Sección 5.7)

ReLUs es una función activadora más efectiva que la función sigmoidea para las redes neuronales. El estudio de este tipo de función es un tema abierto de investigación.

Línea de investigación 2: Redes neuronales (Kernel Methods, Ampliado en la Sección 5.7)

Aplicaciones de estos métodos en el área de *deep learning* sobre *RNA* superficiales que utilizan una gran cantidad de neuronas.

Línea de Investigación 3: Deep learning (Ampliado en la Sección 5.7)

Línea de Investigación 4: Datos Unarios (Ampliado en el Anexo 9) Algunos puntos identificados son: estudio de la fundamentación de su teoría, el entrenamiento de los modelos, evitar mínimos locales y construcción de arquitecturas generales

Línea de Investigación 5: Análisis de las redes sociales (Ampliado en la Sección 6.4)

Consiste en el análisis de la distribución de probabilidad conjunta de los grados de entrada y de salida de los nodos de un grafo dirigido.

8.3. Trabajos Futuros

Los trabajos futuros están relacionados con los casos de estudio y líneas de investigación sugeridas en las conclusiones.

El trabajo realizado genera la interrogante sobre las posibilidades de contar con una metodología integral de analítica sobre *Big Data*.

Hacia una metodología para analítica de Big Data

Considerando que una metodología es un conjunto de técnicas que se siguen en una investigación científica, se pueden utilizar las técnicas descriptas para realizar un esbozo del diseño de ella. Pero debido a los desafíos de *Big Data*, la elaboración de tal metodología implica, en primera instancia, definir bajo que óptica/s realizarla.

Algunas formas de abordar esta tarea:

- Contemplando las *3 v's* de *Big Data*
- Considerando los conceptos de Tareas, métodos o aplicaciones (Como primer paso se puede complementar la clasificación de las tareas y métodos introducidos en el *Capítulo 1*, contemplando a los métodos bayesianos).
- En base a la tecnología a utilizar

- En base al objetivo del estudio
- Combinación de puntos anteriores

Referencias bibliográficas

[1] Suthaharan S. *Machine Learning Models and Algorithms for Big Data Classification*. Editorial Springer, New York, USA, 1st. edition, 2016.

[2] Stonebraker M. International conference on data engineering. URL: http://bigdata.snu.ac.kr/icde2015, April 2015. Accedido 24-03-2018.

[3] Hernández J., Ramírez M., and Ferri C. *Introducción a la minería de datos*. Editorial Pearson, España, 1st. edition, 2004.

[4] Leskovec J., Rajaraman A., and Ullman J. *Mining of Massive Dataset*. Cambridge University Press, USA, 2nd. edition, 2014.

[5] Flajolet P. and Martin G. Probabilistic counting algorithms for data base applications. *Journal of Computer and System Sciences*, pages 182–209, Oct 1985.

[6] Medina F. and Galván M. *Imputación de datos: teoría y práctica*. Naciones Unidas, CEPAL, División de Estadística y Proyecciones Económicas, Santiago de Chile, 3rd. edition, 2007.

[7] Kamakshi L., Harp S., Goldman R., and Samad T. Imputation of missing data using machine learning techniques. *AAAI*, 1996.

[8] Cannavos G. *Probabilidad y estadística Aplicaciones y métodos*. Editorial McGraw Hill, México, 1st. edition, 1988.

[9] Chandola V., Banerjee A., and Kumar V. Anomaly detection: A survey. *ACM Computing (CSUR)*, 41(3):58, 2009.

[10] Han J., Kamber M., and Pei J. *Data Mining Concepts and Techniques*. Morgan Kaufmann - Elsevier, USA, 3rd. edition, 2012.

[11] Reeves L. *A Manager's Guide to Data Warehousing*. Wiley Publishing Inc., Indianapolis, Indiana,USA, 2009.

[12] Dean J. and Ghemawat S. Mapreduce: Simplified data processing on large clusters. *ACM Computing (CACM)*, 51(1), 2009.

[13] Anil J., Robert P., Duin W., and Jianchang M. Statistical pattern recognition: A review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(1), Jan 2000.

[14] Hastie T. James G., Witten D. and Tibshirani R. *An Introduction to Statistical Learning with Applications in R*. Editorial Springer, New York, USA, 2nd. edition, 2009.

[15] Banko M. and Brill E. Scaling to very very large corpora for natural language disambiguation. In *ACL '01 Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*, pages 26–33, Toulouse, France, July 2001.

[16] Mayer-Schonberger V. and Cukier K. *Big Data la revolución de los datos masivos*. Turner Publications, Madrid, España, 1st. edition, 2013.

[17] Gandomi A. and Haider M. Beyond the hype: Big data concepts, methods, and analytics. *International Journal of Information Management*, Apr 2015.

[18] Fan J., Han F., and Liu H. Challenges of big data analysis. *National Science Review*, 42(2), Feb 2014.

[19] Fan J. and Lv J. Sure independence screening for ultrahigh dimensional feature space (with discussion). *Journal of the Royal Statistical Society*, 70:849–911, Apr 2008.

[20] Hastie T., Tibshirani R., and Friedman J. *The Elements of Statistical Learning Data Mining, Inference, and Prediction*. Editorial Springer, New York, USA, 2nd. edition, jan 2017.

[21] Zou H. and Hastie T. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society. Series B (Statistical Methodology)*, 67(2):301–320, Dec 2005.

[22] Tibshirani R. Hastie T. and Wainwright M. *Statistical Learning with Sparsity. The Lasso and Generalizations*. CRC Press A Chapman and Hall book, USA, 1st. edition, 2015.

[23] Indyk P. Datar M., Immorlica N. and Mirrokni V. S. Locality-sensitive hashing scheme based on p-stable distributions. In *SCG '04 Proceedings of the twentieth annual symposium on Computational geometry*, pages 253–262, Brooklyn, New York, USA, June 2004.

[24] Freund Y. and Schapire R. A short introduction to boosting. *Journal of Japanese Society for Artificial Intelligence*, 14(5):771–780, Sept 1999.

[25] Ratsch G. and Warmuth M. Efficient margin maximizing with boosting. *Journal of Machine Learning*, 6:2131–2152, Jan. 2005.

[26] Gaster B. and Jones M. A polymorphic type system for extensible records and variants. In: Report NOTTCS-TR-96-3, Department of Computer Science, University of Nottingham, Nottingham, England, Nov 1996.

[27] Russell S. and Norvig P. *Inteligencia Artificial - Un enfoque moderno*. Editorial Pearson Prentice Hall, Madrid, España, 2nd. edition, 2004.

[28] Andrew M. Dai and Quoc V. Le. Semi-supervised sequence learning. In *NIPS*, 2015.

[29] Girshick R. Fast r-cnn. *IEEE International Conference on Computer Vision (ICCV)*, 22(1), Sept 2015.

[30] Berkhin P. *A Survey of Clustering Data Mining Techniques*. Editorial Springer, Berlin, Germany, 1st. edition, 2006.

[31] Jain A. Statistical pattern recognition: A review. *Transactions on Pattern Analysis and Machine Intelligence*, 22(1):4–37, Jan 2000.

[32] Reynolds D. *Gaussian Mixture Models*, pages 659–663. Editorial Springer US, Boston, MA, 2009.

[33] Shalizi C. *Advanced Data Analysis from an Elementary Point of View*. Cambridge University Press, Cambridge, United Kingdom, 1st. edition, 2015.

[34] Markarian R. and Moller N. *Como cuantificar la importancia individual en una estructura de enlaces: Google-Page Rank*. Prepublicaciones de matemática de la Universidad de la República del Uruguay, 2004.

[35] Brin S. and Page L. The anatomy of a large-scale hypertextual web search engine. *Comput. Netw. ISDN Syst.*, 30(1-7):107–117, April 1998.

[36] Konstan J. and Riedl J. Recommended for you. *Spectrum, IEEE*, 49:54–61, Oct. 2012.

[37] Liu B. Sentiment analysis and subjectivity. In *Handbook of Natural Language Processing, Second Edition. Taylor and Francis Group, Boca*, 2010.

[38] Rocha L. and Silveira T. Saci: Sentiment analysis by collective inspection on social media content. *Web Semantics: Science, Services and Agents on the World Wide Web*, 34(3), 2015.

[39] Aguirre J. L. *Introducción al Análisis de Redes Sociales*. Centro Interdisciplinario para el Estudio de Políticas Públicas, Buenos Aires, Argentina, 1st. edition, Dec 2011.

[40] Barabási A. L. *Linked*. Editorial Penguin Group, Westminster, Maryland, USA, 6th. edition, 2003.

[41] Newman M.E.J. *Networks, An Introduction*. Oxford University Press, USA, 1st. edition, 2010.

[42] Binti I. Dutt A., Aghabozrgi S. and Mahroeian H. Clustering algorithms applied in educational data mining. *International Journal of information and Electronics Engineering*, 5(2), Mar 2015.

[43] Qi X. Flann N. Xu B., Recker M. and Ye L. Clustering educational digital library usage data: A comparison of latent class analysis and k-means algorithms. *JEDM — Journal of Educational Data Mining*, 5(2):38–68, Mar 2013.

[44] Glaser B. G. and Strauss A. L. *The discovery of grounded theory: Strategies for qualitative research*. Aldine Transaction, New Jersey, USA, 1st. edition, 2006.

[45] Tufte E. *Beautiful Evidence*. Graphics Press, L.L.C., Cheshire, USA, 1st. edition, 2006.

[46] Duda R., Hart P., and Strok D. *Pattern Classification*. JOHN WILEY and SONS, INC, New York, USA, 2nd. edition, 2000.

[47] Yoonsuh J. Multiple predicting k-fold cross-validation for model selection. *Journal of Nonparametric Statistics*, 30(1):197–215, 2018.

[48] Davis J. and Goadric M. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd International Conference on Machine Learning (ICML)*, USA, 2006.

[49] Modis T. The singularity myth. *January 2006 Technological Forecasting and Social Change*, 73(2):197–215, Jan 2006.

[50] Han J., Pei J., and Yin Y. Mining frequent patterns without candidate generation. *SIGMOD Rec.*, 29(2):1–12, May 2000.

[51] Quoc V. A tutorial on deep learning part 1: Nonlinear classifiers and the backpropagation algorithm, Dec 2015.

[52] Ricci F., Rokach L., Shapira B., and Kantor P. *Recommender Systems Handbook*. Springer, USA, 2nd. edition, 2015.

[53] Konstan J., Terveen L., and Riedl J. Evaluating collaborative filtering. *ACM Transactions on Information Systems*, 22(1):5–53, Jan 2006.

[54] Pennock D, Horvitz E., and Giles C. Social choice theory and recommender systems: Analysis of the axiomatic foundations of collaborative filtering. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, pages 729–734. AAAI Press, 2000.

[55] Lathia N., Hailes S., and Capra L. Trust-based collaborative filtering. *IFIP International Federation for Information Processing*, 263(2):119–134, 2008.

ANEXOS

Anexo 1

Visualización

El objetivo de la visualización es mostrar gráficamente los resultados obtenidos del modelo de analítica utilizado. Los siguientes principios a considerar en la etapa de visualización son propuestos por Edward Tufte [45].

- Comparaciones: Mostrar comparaciones (si existe otra hipótesis contra la cual comparar)
- Causalidad: Mostrar algún mecanismo que explique lo que ocurre, como, por ejemplo, una estructura sistemática que se obtenga de los datos y se pueda visualizar.
- Datos multivariados: Mostrar datos multivariados significa mostrar la mayor cantidad de datos posible ya que los datos de por si son multivariados, y la información se modifica a cada instante.
- Integración: Implica integrar toda la evidencia, no solo quedarse con un gráfico o una tabla sino de combinar diferentes modalidades de los resultados en una sola presentación, ya que la información se hace más rica.
- Descripción: Describir y documentar la evidencia que se tiene. Por ejemplo, si el resultado final es una gráfica realizada por un programa, es necesario preservar el código que lo originó a los efectos de dar credibilidad a la presentación. También implica el empleo de etiquetas adecuadas y escalas, entre otras consideraciones.
- Contenido: Este principio es el más importante, indica que la presentación de la información depende de la calidad, integridad y relevancia de su contenido.

Anexo 2

Stream de Datos

En este anexo se presentan dos ejemplos de aplicaciones de *streams* de datos. En el primero se analizan las dificultades, al emplear un método que preserva la respuesta a una consulta determinada, introduciendo únicamente el ruido que proviene del hecho de que se está muestreando al azar (extraído de *Mining of Massive Datasets Leskovec J, Rajaraman A, Ullman J*) y el segundo describe un caso del algoritmo de *Flajolet- Martin.*

En la *Sección 2.2, Stream de datos*, se mencionó que uno de los principales desafíos en esta área es la obtención de una muestra de los datos que sea representativa.

Muestreo al azar

Suponer que, a partir de la llegada de los datos en un *stream*, se quiere analizar su flujo, guardando una fracción de él, por ejemplo, $\frac{1}{10}$.

A los efectos de ilustrar el cuidado al tratar este tipo de problemas, se considera un sistema que recibe *queries* de distintos usuarios y se busca estudiar la conducta de ellos. Se pretende determinar qué porcentaje de las *queries* fueron realizadas dos veces. A tales efectos se revisan $\frac{1}{10}$ del total de registros en el mes, obtenidos aleatoriamente. Aunque las fluctuaciones estadísticas introducen ruido, si las *queries* son muchas, por ley de los grandes números, se puede asegurar que cada usuario tiene $\frac{1}{10}$ de sus *queries* guardadas, por lo que, analizando sobre esa muestra, el resultado es estadísticamente aceptable [4].

Se considera ahora una nueva situación, en la cual se desea encontrar el promedio de *queries* efectuadas dos veces por usuario, tomando $\frac{1}{10}$ de los datos (en el caso anterior el objetivo era determinar el porcentaje sobre todas las *queries* realizadas).

Suponer que el usuario realiza s búsquedas una sola vez, d dos veces y no hizo más búsquedas. Al tomar una muestra de un décimo del total, la fracción de los que aparecen dos veces en la muestra es $\frac{d}{(10s+19d)}$. Al final de la sección se explica cómo se obtiene esa cantidad.

Sin embargo, la respuesta correcta de las consultas duplicadas es $\frac{d}{(s+d)}$

Esta situación se origina por el hecho de que, en realidad, se busca elegir una muestra al azar en un determinado instante, de forma independiente. Sin embargo, el valor $\frac{d}{(10s+19d)}$, se obtiene tomando una muestra de la décima parte de las *queries*, no la décima parte de las instancias de la búsqueda.

La obtención de la probabilidad $\frac{d}{(10s+19d)}$ de la muestra (que finalmente no es representativa) se presenta a continuación y posteriormente un procedimiento para obtener una muestra representativa.

En la muestra de $\frac{1}{10}$ del *stream*, de las *queries* enviadas una vez, se puede esperar encontrar $\frac{s}{10}$.

De las enviadas dos veces, se puede esperar ver $\frac{d}{100}$. Esto es d veces la probabilidad de dos ocurrencias en la muestra de $\frac{1}{10}$, o sea $d\frac{1}{10}\frac{1}{10} = \frac{d}{100}$

De las enviadas dos veces, que aparecen dos veces en el *stream* completo, las que aparecen una sola vez en la muestra de $\frac{1}{10}$ son: $\frac{18d}{100}$. Esto es debido a que la probabilidad de que una de las dos ocurrencias esté en la muestra es $\frac{1}{10}$ y que la otra está en la parte de $\frac{9}{10}$ como no seleccionada. Por lo tanto, $\frac{1}{10}.\frac{9}{10} = \frac{9}{100}$. Esto es para cada una de las ocurrencias por lo tanto es d veces $\frac{18}{100}$.

Entonces se tiene que las ocurrencias dobles en la muestra son: $\frac{d}{100}$ y las que aparecen una vez son: $\frac{s}{10} + \frac{18d}{100}$

La fracción de las que aparecen dos veces es:

$$\frac{\frac{d}{100}}{\frac{s}{10} + \frac{18d}{100} + \frac{d}{100}} = \frac{d}{10s + 19d}$$

Ejemplo de algoritmo de Flajolet- Martin

El algoritmo debe devolver un número estimativo del número de elementos diferentes en el *stream*.

Sea $s =< 2, 4, 10, 8, 4, 10, 2, 4, 10 >$ un *stream* de datos recibido en ese orden. En s hay *4* objetos diferentes. Sea una función de *hash*: $h(x) = (2 * x + 3)$ *mod* 5 (de la cual se sabe que puede almacenar los *4* objetos diferentes

del *stream*). En la Tabla 2.1 se presentan los datos de entrada, los obtenidos al pasarlos por $h(x)$, su representación en binario y la cantidad de ceros que tiene, al final, cada *string*. Finalmente el valor máximo en la última columna es 2. Por lo tanto se considera $R = 2$ y un valor estimado de la cantidad de objetos diferentes del *stream* es $2^R = 2^2 = 4$.

x	h(x)	Binario	Cantidad de ceros
2	2	010	1
4	1	001	0
10	3	011	0
8	4	100	2
4	1	001	0
10	3	011	0
2	2	010	1
4	1	001	0
10	3	011	0

Tabla 2.1: Valores del Algoritmo de Flajolet- Martin

Anexo 3

MapReduce

En este anexo se describe la ejecución de *MapReduce* y algunas de sus aplicaciones.

Ejecución de *MapReduce*

La ejecución del proceso de *MapReduce*, se realiza distribuyéndolo entre varios equipos (miles en algunos casos). A continuación se describe el funcionamiento completo [4]

Paso 1: La librería de *MapReduce* en el programa del usuario divide los datos de entrada, en un conjunto de M particiones. Luego se generan múltiples copias del programa en las distintas máquinas del clúster.

Paso 2: Una de estas copias del programa es especial y toma el papel de «maestro». Las demás son copias que el máster asigna a los *workers*, existen M funciones *Map* y R funciones *Reduce* para asignar.

Paso 3: Un *worker* que tiene una tarea de *Map* lee el contenido de su parte desde el input, «parseando» los pares (clave, valor), para crear una nueva pareja de salida, tal y como se especifica en su programación. Los pares clave y valor producidos por la función *Map* se almacenan como *buffer* en la memoria.

Paso 4: Periódicamente, los pares (clave, valor) almacenados en el *buffer* se escriben en el disco local, repartidos en R regiones. Las regiones de estos pares (clave, valor) son pasados al máster, el cual es responsable de redirigir a los *workers* que tienen asignadas tareas de *Reduce*.

Paso 5: Cuando un *worker* de tipo *Reduce* es notificado por el máster con la localización de una partición, éste emplea llamadas remotas para generar

lecturas de la información almacenada en los discos duros de los diversos *workers* de tipo *Map*. Luego de ser leídos los datos intermedios, los agrupa por claves. El ordenamiento es necesario debido a que, por regla general, muchas claves de diversas funciones *Map* pueden ir a una misma función *Reduce*.

Paso 6: El *worker* de tipo *Reduce* itera sobre el conjunto de valores ordenados intermedios, y lo hace por cada una de las claves únicas encontradas, tomando la clave y el conjunto de valores asociados a ella. Esta función genera una salida que es añadida al archivo de salida del *Reduce* de su partición.

Paso 7: Cuando todas las tareas *Map* y *Reduce* se han completado, el máster despierta al programa del usuario, retornándole el control.

Las salidas se distribuyen en un archivo completo o, en su defecto, se reparten en R archivos, que pueden ser la entrada de otro *MapReduce* o puede ser procesado por cualquier otro programa que necesite de ese *output*.

Tolerancia a Fallos

El mecanismo de *MapReduce*, es tolerante a fallos cuando uno de los *workers* presenta una falla. Aunque la probabilidad de fallo sea baja, es altamente probable que algún *worker* quede desactivado (debido a que una gran cantidad de máquinas trabajan simultáneamente), precisamente por un fallo de la máquina que le daba soporte. Para determinar la cantidad de *mappers* y *reducers* necesarios, la regla general es hacer M bastante mayor que el número de nodos. De esta forma, si un nodo que está haciendo una función de *Map* o *Reduce*, cae, el administrador, que está constantemente enviando *pings*, al enterarse de la falla, puede distribuir las tareas entre los demás [4].

Granularidad de las tareas

Como se mencionara anteriormente, hay M tareas de *Map* y R tareas de *Reduce*, pero idealmente es conveniente que M y R sean considerablemente mayores a la cantidad de *workers*. Esto se justifica debido a que, al tener múltiples tareas para hacer, se mejora el balance de carga y la recuperación, cuando un nodo cae. Existen límites experimentales para el tamaño de M y R. El máster debe hacer $O(M + R)$ decisiones en la agenda de tareas y mantener $O(M * R)$ estados en memoria [4].

Multiplicación de matrices

Se considera la multiplicación de dos matrices M y N las cuales cumplen con las restricciones necesarias en sus dimensiones para poder multiplicarse. La matriz M se considera como una relación $M(I, J, A)$ cuyas tuplas son (i, j, m_{ij}) y análogamente la matriz N, que simboliza a $N(J, K, B)$. Una ventaja que presenta esta representación para matrices grandes y que son, a su vez, *sparse*, es que facilita las operaciones.

El producto MN se puede hacer utilizando un *join* natural de $M(I, J, A)$ y $N(J, K, B)$ teniendo el atributo j en común que produce tuplas del tipo (i, j, k, v, w). En realidad se necesita $(i, j, k, v.w)$, ya que $v.w$ representa el producto $m_{ij}.n_{jk}$. Una vez obtenido, se aplica agrupamiento en función de los atributos i y k y agregación $v.w$. Para ello, es necesario operar dos procesos *MapReduce* [4]. En el primer *Map*: por cada elemento m_{ij} de M se produce el par: $(j, (M, i, m_{ij}))$ y lo mismo para cada elemento n_{jk} de N : $(j, (N, k, n_{jk}))$. A continuación, la función *Reduce* examina las listas asociadas para cada clave j. Para cada valor proveniente de M y cada valor de N, produce una clave: (i, k) y un valor igual al producto de los dos valores: $m_{ij}.n_{jk}$.

Luego se corre el segundo proceso de *MapReduce*.

En este caso la función *Map* es la identidad: para cada par (clave, valor) obtenido, $((i, k), v)$, produce el mismo par y la función Reduce, para cada clave (i, k), devuelve la suma de los valores asociados que será el elemento ik correspondiente a la matriz del producto.

Anexo 4

Métodos Estadísticos de Aprendizaje

En los entornos reales, la incertidumbre es el factor más importante a considerar. La potencia de la visión bayesiana del aprendizaje es debida a que proporciona soluciones generales a los problemas del ruido, sobreajuste y predicción óptima y permite su utilización tanto para tareas descriptivas y predictivas (*Capítulo 1*) [3].

En este anexo, se describe un método para el aprendizaje de modelos probabilísticos denominado redes bayesianas.

En la *Sección 3.1, Modelos*, se mencionó que, disponiendo del conocimiento de las densidades de probabilidades condicionales $p(\mathbf{x}|C)$, siendo $\mathbf{x}$ un vector de características y C un estado de la naturaleza, clase, o variable discreta, se daba paso al desarrollo de la teoría de la decisión bayesiana. Esta teoría da por supuesto que el problema de la decisión se basa en términos probabilísticos [46].

Los métodos bayesianos permiten trabajar con incertidumbres (llamadas evidencias), cuantificándolas a través de la teoría de la probabilidad. Posibilitan realizar un uso descriptivo (búsqueda de relaciones de independencia entre las variables, u otro tipo de relación) y predictivo (a través de clasificadores, encontrando reglas de asignación de individuos a una clase).

4.1. Teoría de la decisión de Bayes

Se dispone de una muestra de n individuos, de los cuales se conoce a qué clase pertenecen y se dispone de medidas de p variables cuantitativas, que definen las clases. Se supone que cada clase tiene asignada una distribución de probabilidad entre sí.

Sea q la cantidad de clases: $\{C_1, C_2, .., C_q\}$. Para cada clase C_k existe una función de densidad de probabilidad dada por $f_k(x) = f(x|C_k)$. A su vez, cada una de las clases tiene una probabilidad $P(C_k)$ *a priori* (la cual se basa en el conocimiento que se tiene de antemano).

Con estas dos probabilidades se determina la regla de asignación a los nuevos elementos. Las variables x_j no siempre permiten separar las clases entre sí, debido al solapamiento que pudiera existir entre ellas.

Para evitar errores, la asignación se hace en función de la minimización de la probabilidad de mala clasificación, o maximización del acierto. Se demuestra [3] que esta asignación implica asignar la maximización de la probabilidad a posteriori ($P(C_k|\mathbf{x})$).

Ejemplo 4.1

Suponer que se dispone de becas para otorgar entre alumnos, efectuando una asignación al azar, utilizando una moneda.

En ausencia de información, el porcentaje de error es de 50 %. Pero, si se poseyese la información adicional, de que a quienes se les entregaron beca en otra oportunidad, el 60 % resultaron buenos estudiantes (probabilidades *a priori*), se podría volver a asignar el 100 % de las becas al mismo grupo de estudiantes (que contiene la clase de mayor probabilidad) llegándose a que la equivocación al asignar becas, sea menor (40 %).

Decisión Bayesiana

La decisión bayesiana es un enfoque estadístico para la clasificación. Consiste en estudiar las probabilidades de tomar decisiones incorrectas para cuantificar los costos y diseñar estrategias para minimizarlos.

Suponer que se tiene que decidir entre dos clases C_1 y C_2 y que el costo de tomar una decisión incorrecta es el mismo en cualquiera de las dos. Si, a su vez, se dispone sólo de la probabilidad *a priori*, la regla de decisión más razonable, es optar por la probabilidad de ocurrencia mayor entre las dos. Asimismo, es razonable pensar que la probabilidad del error, al tomar tal decisión, es la menor de las probabilidades.

4.2. Teorema de Bayes de hipótesis MAP

El enfoque bayesiano permite obtener resultados óptimos, si bien es altamente costoso, ya que contemplará todos los casos de las hipótesis (los cuales pueden ser infinitos). Por lo tanto, es conveniente acudir a métodos aproximados. En este sentido, el enfoque más común es realizar las predicciones basadas en una única hipótesis, la cual será la más probable. En otras palabras, se pretende efectuar la predicción a través de una hipótesis que maximice $P(C_k|x)$. El teorema de *Bayes* es utilizado para calcular la probabilidad a posteriori ($P(C_k|x)$), dada por la Ec. (4.1).

$$P(C|x) = \frac{(P(x|C).P(C))}{P(x)} \tag{4.1}$$

La probabilidad $P(C|x)$ es denominada verosimilitud de que la hipótesis x haya producido al conjunto de observaciones C.

Si se tratara de un problema de clasificación con una variable de clase C y un conjunto de variables predictoras $r = (A_1...A_n)$ el teorema de *Bayes* es formulado en la Ec. (4.2).

$$P(c|r) = \frac{(P(r|c).P(c))}{P(r)} \tag{4.2}$$

Probabilidad y verosimilitud

En estadística, se hace una distinción entre probabilidad y verosimilitud. La probabilidad se utiliza para describir posibles resultados futuros, antes de que los datos estén disponibles. La verosimilitud se utiliza después de que los datos se encuentran disponibles, para describir una función de los parámetros, dada una determinada salida.

Si C tiene k valores $\Omega_c = \{c_1, c_2, \ldots, c_k\}$, el resultado más plausible y que define la clasificación, es el que maximiza la probabilidad *a posteriori* de los atributos y se le denomina, probabilidad máxima *a posteriori* (MAP).

$$\begin{aligned} C_{MAP} = \; & arg\ max\ P(C|A_1, ..., A_n) \\ & arg\ max \frac{p(A_1|c, ..., A_n|c).p(c)}{P(A_1, ..., A_n)} \\ & arg\ max\ p(A_1|c, ..., A_n|c).p(c) \\ & argmax p(A_1|c, ..., A_n|c).p(c) \\ (Con\ c \in \Omega_c) & \end{aligned} \tag{4.3}$$

En la búsqueda del máximo, no es necesario considerar al divisor que es constante. No obstante la Ec. (4.3) es computacionalmente costosa, ya que se deben calcular las distribuciones de probabilidad de muchas variables. Esto se puede mejorar considerando una suposición de independencia [3].

Naive Bayes (NB)

Se asume, aunque no siempre es el caso, que todas las características (atributos) utilizadas para el aprendizaje, son independientes para la elección particular de la conducta que se quiere aprender (dada una clase se asume que los atributos son condicionalmente independientes entre sí, es decir, si la ocurrencia o no ocurrencia de uno de ellos junto con la de otro, se da en forma independiente dado un tercer atributo).

Por ejemplo, en el análisis de texto, existen palabras que podrían ir juntas y por lo tanto son dependientes (ejemplo: «*red bayesiana*»).

Estimación de los parámetros en Naive Bayes

Dada la hipótesis de independencia y a partir de (4.3), la expresión para obtener las hipótesis MAP se presentan en la Ec. (4.4).

$$C_M AP = argmaxp(A_1, ..., A_n|c).p(c) = argmaxp(c) \prod_{i=1}^{n} p(A_i|c) \tag{4.4}$$

$Con\ c \in \Omega_c$

Los parámetros a estimar son $p(A_i|c)$ para cada característica A_i y los valores de la probabilidad *a priori* de $p(c)$.

Generalmente, para lograr que los cálculos sean estables es conveniente trabajar con el logaritmo, debido a que las multiplicaciones de tantas probabilidades menores a 1, pueden ocasionar que se pierda la precisión en la máquina, dejando todos los puntos en cero.

4.3. Redes Bayesianas

Las redes bayesianas son modelos de representación del conocimiento con incertidumbre. Utilizan grafos dirigidos acíclicos que representan el conocimiento, indicando la dependencia o independencia entre las variables del modelo.

La parte cuantitativa del modelo de una red bayesiana puede representarse como una distribución de probabilidad P que tiene algunas dependencias por medio de la ecuación (4.5).

$$I(X, Y|Z) \longleftrightarrow P(X|YZ) = P(X|Z) \tag{4.5}$$

Se considera la distribución de probabilidad P, como un modelo de dependencias. Los X, Y y Z son subconjuntos de variables del modelo y el operador I indica la relación de independencia condicional (X es condicionalmente independiente de Y, conocido Z) [3]. La codificación de las relaciones de independencia hacen posible almacenar la distribución de probabilidad conjunta más eficientemente. Dadas n variables aleatorias que pueden tomar dos valores, se necesitan 2^n parámetros para especificar la distribución conjunta, pero en una

red bayesiana solo existe dependencia de un nodo respecto a sus padres en el grafo (Figura 4.1).

Ejemplo 4.2

Un estudiante hace un curso representado por C. Su aprobación depende de las horas de estudio *(H)* del estudiante y de su inteligencia *(I)*. Esto define una red bayesiana con tres variables aleatorias. Cada una de ellas es representada por un nodo, al cual se le asocia una distribución de probabilidad. En el caso del nodo C, la distribución de probabilidad es condicional, ya que depende de H *e* I.

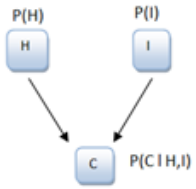

Figura 4.1: Redes bayesianas

En la Figura 4.1, se observa el grafo dirigido acíclico con sus respectivas probabilidades. La red bayesiana produce una distribución de densidad conjunta. Se define una regla de la cadena para redes bayesianas que considera las distintas densidades de probabilidad conjuntas y las multiplica. Si la información disponible no es una distribución de probabilidad conjunta, se considera las probabilidades simples (en la Figura 4.1 serían los nodos H *e* I, Ec. (4.6)).

$$P(H, I, C) = P(H).P(I).P(C|H, I) \tag{4.6}$$

De acuerdo a los estados de H,I y C, se forman distintos escenarios.

Por ejemplo, en la Tabla 4.1 se considera el estado «horas de estudio» el cual posee varias franjas horarias *(clases* h_1 , h_2 *y* h_3 *)*. La clasificación del estado «inteligencia» se considera como un *booleano* (si / no).

Los porcentajes de cada clase se indican en la Tabla 4.1. Cada una de estas clases posee una probabilidad asociada, al igual que las clases de inteligentes y no inteligentes. Con esta información, se pueden calcular todas las entradas para la distribución de probabilidad conjunta.

$0 \leq horas \leq 2(h_1)$	$2 < horas < 10(h_2)$	$10 \leq horas(h_3)$
40 %	55 %	5 %
$Inteligente(i_1)$	$No\ inteligente(i_2)$	
40 %	60 %	
	C (aprobado)	C (no aprobado)
h_1, i_1	0.4	0.6
h_1, i_2	0.7	0.3
h_2, i_1	0.1	0.9
h_2, i_2	0.5	0.5
h_3, i_1	0.6	0.4
h_3, i_2	0.9	0.1

Tabla 4.1: Tablas de valores para la RB de la Figura 12.1

Generalizando a partir del *Ejemplo 4.2,* una red bayesiana se define como un grafo dirigido, acíclico, cuyos nodos son variables aleatorias $X_1, \ldots, X_n$. Para cada nodo X_i, existe una distribución de probabilidad $P(X_i|Par_G(X_i))$. La notación $Par_G(X_i)$ indica los padres de X_i en el grafo (nodos que apuntan a X_i). Las redes bayesianas representan una distribución de probabilidades conjunta mediante la regla de la cadena para redes bayesianas (Ec. (4.7))

$$P(X_1, \ldots, X_n) = \prod_{i=1} P(X_i|Par_G(X_i) \tag{4.7}$$

Se demuestra que las redes bayesianas son una distribución de probabilidades, pues al ser el producto de probabilidades, es mayor o igual a cero. Para demostrar que suman la unidad, se observa en la (Ec. (4.8)) que la suma interior $\sum_C P(C|H, I) = 1$, (es la suma de todos los eventos de C).

$$\begin{aligned} \sum_{H,I,C} P(H, I, C) = & \sum_{H,I,C} P(H).P(I).P(C|H, I) \\ & \sum_{H,I} P(H).P(I). \sum_{C} P(C|H, I) \end{aligned} \tag{4.8}$$

Análogamente:

$$\sum_{H,I} P(H).P(I) = \sum_{H} P(H). \sum_{I} P(I) = 1 \tag{4.9}$$

Las redes bayesianas son aptas para llevar a cabo razonamientos causales e inter-causales. El primero de ellos se refiere a observar las probabilidades en un nodo de la red, dejando fijo otro.

El razonamiento inter-causal considera más de un nodo fijo y se observa como varían otros nodos. Puede ocurrir incluso, que el nodo afectado no tenga dependencia con algunos de los otros nodos involucrados. A partir de los razonamientos mencionados, se estudia en las redes bayesianas, cuál es la estructura de nodos en que éstos se afectan entre sí [27].

El entrenamiento de estas redes puede realizarse con el algoritmo del gradiente inicializando los pesos aleatoriamente, siendo actualizados en cada iteración aceptándose alguna solución local del óptimo [10].

4.4. Conclusiones

Los métodos bayesianos permiten trabajar con incertidumbres, posibilitando la búsqueda de relaciones de independencia entre las variables y el diseño de clasificadores para la predicción y resolución de tareas descriptivas.

Las redes bayesianas son modelos que representan conocimiento con incertidumbre, a través de grafos dirigidos acíclicos que modelan la información acerca de las relaciones entre las variables. Esta representación es expresada en términos de probabilidades condicionadas.

Caso de estudio

Las tareas que pueden realizar las redes bayesianas pueden ser predictivas y descriptivas. Se plantea el problema de estudiar la viabilidad de utilizar las redes bayesianas para diseñar los algoritmos respectivos bajo esta óptica. Por ejemplo, si se trata de reglas de asociación $A \rightarrow i$, se busca contestar a la pregunta si se puede re-diseñar empleando una red bayesiana.

Para ello se puede considerar que dicha regla debe ser un patrón frecuente en los datos y que sería necesario de que las variables presenten una distribución de probabilidades, sobre las relaciones entre ellas.

Anexo 5

Errores y Evaluación de los Modelos

En este anexo se desarrolla en detalles los conceptos de errores en los modelos y su evaluación.

5.1. Errores en los modelos

Al aumentar la complejidad del modelo, el error de entrenamiento disminuye, aunque no es indicativo de que las predicciones, en términos generales, sean mejores.

Balance entre el sesgo y la varianza

Aunque lo ideal es mantener valores bajos de varianza y sesgo, éstos se encuentran negativamente asociados; el crecimiento de uno de los valores implica el decrecimiento del otro. En términos generales, si el modelo aumenta en complejidad, la varianza tiende a crecer y el sesgo a disminuir [20].

Sea el *MSE (Mean Squared Error)* definido como el promedio de los errores entre el valor predicho y el real, elevados al cuadrado.

Al probar un modelo sobre el conjunto de datos de *testing*, es posible que el modelo no se adapte, obteniéndose un alto grado de error. Cuando un modelo tiene un pequeño *MSE* en la fase de entrenamiento, pero un *MSE* alto en la etapa de *testing*, se dice que está sobre ajustado (*overfitting*).

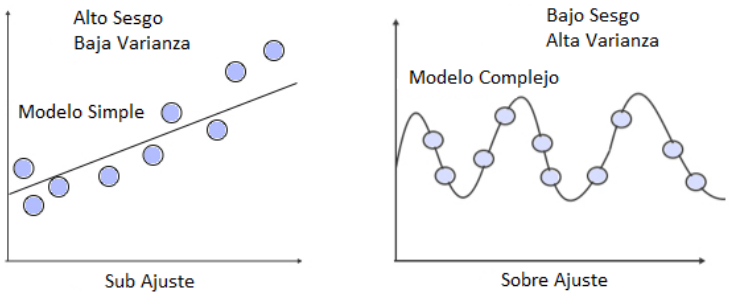

Figura 5.1: Modelos sub ajustados y sobre ajustados

En la Figura 5.1 se ve un modelo simple y otro más complejo. En el de la derecha, el modelo se adapta más a los datos por lo que el error es menor, el sesgo es menor y la varianza alta. Se dice que el modelo está sobre ajustado: posiblemente nuevas observaciones no se ajusten al modelo. Por otro lado, el modelo de la izquierda tampoco predice de manera correcta, ya que tiene un *MSE* alto en el entrenamiento. Se considera que el modelo está sub ajustado, por lo que se desconocen muchos patrones en el entrenamiento, lo que hace crecer su sesgo.

Figura 5.2: Balance entre sesgo y varianza

En la Figura 5.2 se muestra la variación del error en función de la complejidad. En un modelo simple existen errores de predicción, el sesgo es alto pero la varianza baja (modelo sub ajustado). Al aumentar la complejidad del modelo, el sesgo disminuye y la varianza aumenta. A su vez, para modelos más complejos, el error de predicción o sesgo decrece en el conjunto de entrenamiento. Se observa que, a partir de cierta complejidad del modelo, el *MSE* comienza a aumentar en el conjunto de validación, debido a que posiblemente haya sobreajuste. En la misma figura se grafica como varían el sesgo y la varianza al aumentar la complejidad del modelo. Hay un punto donde se da el balance perfecto. En dicho punto es donde el error de *testing* es menor, el

modelo tendrá bajo sesgo y baja varianza y donde ambos valores hacen que el error de *test* sea el menor (se encuentra un mínimo del *MSE*). Es el punto ideal al momento de considerar el balance.

Variación del error en función de la complejidad del modelo

El análisis previo sugiere el estudio del error del modelo, en función de su complejidad. Para ello, se observa cuál es el error generalizado o error verdadero. El objetivo de dicho error es medir la precisión del algoritmo. Usualmente, la performance de un algoritmo de *ML*, se mide a través de las gráficas del error generalizado, a lo largo del proceso de entrenamiento. Cuando la complejidad es alta (varianza alta), al considerarse diferentes conjuntos de entrenamiento en los datos, se obtendrán cambios muy pronunciados en los modelos. Desde el punto de vista del sesgo, se utiliza el promedio de las curvas, el cual se encuentra próximo a la curva real, por lo que el sesgo será pequeño (al disminuir el error generalizado hasta llegar a su mínimo, se observa en la misma figura, que el error de *testing* también disminuye).

En la Figura 5.3 se observa el comportamiento de los errores al aumentar la complejidad del modelo. El error de entrenamiento disminuye, ya que el modelo se ajusta mejor a los puntos dato y el error de *testing* va oscilando alrededor de la gráfica que representa el error generalizado. Este comienza a disminuir pero, al agregarse más complejidad, aumenta.

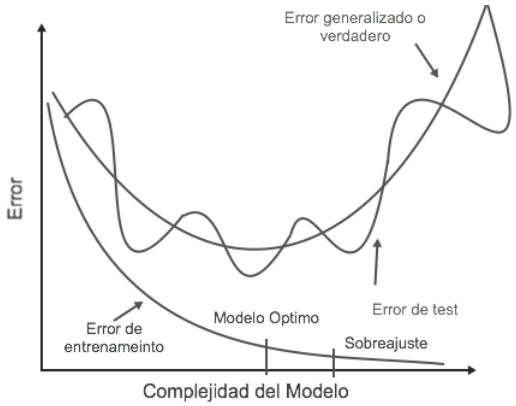

Figura 5.3: Error de entrenamiento, generalizado y de *testing* vs. complejidad del modelo (Basado en: *An Introduction to Statistical Learning with Applications in R Gareth James, Trevor Hastie, Daniela Witten, Robert Tibshirani - 2017*)

Performance de un modelo en función de la cantidad de datos y de los atributos

Sea un modelo que predice un valor en función de una sola característica. Con este modelo, no se podrán efectuar predicciones aunque haya disponibles muchos datos. Se puede decir que este modelo tiene alto sesgo (el sesgo es el error que se introduce al utilizar un modelo que quiere aproximarse a una situación real, e indica qué tan alejado se encuentra el modelo, de los verdaderos valores). Por lo tanto, es conveniente utilizar múltiples parámetros empleando algoritmos que aseguren un sesgo pequeño. El hecho de que con diversos parámetros se puedan ajustar funciones más complejas, asegura un sesgo menor.

Con un modelo complejo, como se expresó anteriormente, el conjunto de entrenamiento posiblemente se ajuste adecuadamente, pudiendo esperarse que el error de entrenamiento sea bajo. Por otro lado, si el conjunto de entrenamiento es masivo (la cantidad de observaciones es considerablemente mayor a la de parámetros), se puede asegurar que el modelo no se sobreajusta. En consecuencia, el error de entrenamiento estará próximo al error de *test* [20]. Si el error de *test* es próximo al error de entrenamiento y éste es pequeño, se puede inferir que el error de *test*, también es pequeño. En conclusión, al utilizar algoritmos de aprendizaje con múltiples parámetros, se puede asegurar un sesgo bajo y utilizando grandes conjuntos de datos de entrenamiento, se puede garantizar una varianza pequeña. En consecuencia se dispondría de un algoritmo con buena performance en la predicción.

Comportamiento del error verdadero y de entrenamiento en función de la cantidad de datos en el conjunto de entrenamiento

En esta sección se considera un modelo fijo, es decir no aumenta su complejidad (cambian los parámetros ajustando el modelo). Con escasos datos el error será alto, pues la función de estimación posiblemente devuelva pobres resultados sobre la verdadera relación entre x e y. Al agregar más datos los resultados de la estimación mejoran, por lo que el error disminuye hasta llegar a un límite inferior donde el error se compone por la suma del sesgo más el ruido de los datos.

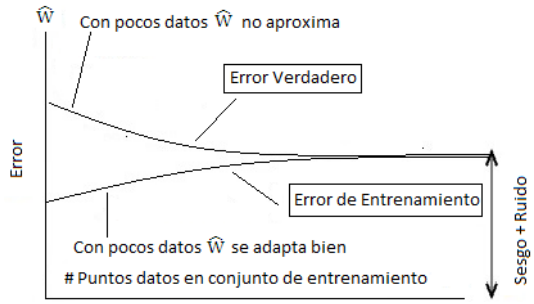

Figura 5.4: Error en función de la cantidad de datos con complejidad fija (Origen: *An Introduction to Statistical Learning with Applications in R, Gareth James, Trevor Hastie, Daniela Witten, Robert Tibshirani - 2017*)

Si se considera el error de entrenamiento, con escasos datos, un modelo puede ajustarse adecuadamente, pero si se aumentan los puntos de entrenamiento, en un modelo fijo, posiblemente no se adecúe a esa mayor cantidad de observaciones. El caso extremo es cuando se utilizan todos los datos (utópicamente) del universo, cuyo límite es el dado por el sesgo y el ruido, siendo exactamente en esas circunstancias, que ocurriría el error verdadero, coincidiendo en el mismo extremo. Esta situación se representa en la Figura 5.4.

5.2. Estimación del error generalizado

Debido a que es improbable calcular el error generalizado de un modelo, dado que se necesita calcularlo sobre todos los posibles puntos, se intentará obtener una estimación. Para ello, se consideran varios conjuntos de entrenamiento, cada uno de los cuales produce un modelo. La idea es estimar el error generalizado evaluándolo sobre todos ellos. Por lo tanto, a partir de diferentes conjuntos de entrenamiento interesa determinar cuál es la performance promedio. Una forma de medir el error generalizado a lo largo de todos los conjuntos de entrenamiento, es mediante la esperanza del error en la predicción, calculada sobre todos los conjuntos de entrenamiento. Esta función es llamada error de predicción esperado, y es representada en la Ec. (5.1). Es el valor de error esperado sobre los *datasets* de entrenamiento formados por las tendencias del error generalizado.

$$Error\ prediccion\ esperado\ =\ E_{entren}[error\ generalizado\ \ \hat{w}\ (set\ entren)] \tag{5.1}$$

Si x_t es una característica de los datos, se demostrará que el error de predicción esperado en x_t es (5.2):

$$\sigma^2 + [sesgo(f_{\hat{w}}(x_t))]^2 + var(f_{\hat{w}}(x_t)) \tag{5.2}$$

Sean n observaciones $(x_1, y_1), \ldots, (x_n, y_n)$, y sea $p(x_i, y_i) = P(X = (x_i, y_i))$, la probabilidad de que la observación (x_i, y_i) sea considerada para el cálculo del error generalizado (no todos los puntos intervienen en el cálculo del error verdadero). El error generalizado se puede considerar como la esperanza sobre todos los posibles pares (x_i, y_i) de las pérdidas.

Los $\hat{w}$ son los parámetros del ajuste para un determinado conjunto de entrenamiento y $f_{\hat{w}}(x_t)$ es el modelo para dichos parámetros. El error de predicción esperado es calculado sobre todos los conjuntos de entrenamiento.

Las tres fuentes de error son el ruido, el sesgo y la varianza [20].

A partir del modelo $y = f_w(x) + \varepsilon$, se presentan las siguientes definiciones:

$$\begin{aligned} Funcion\ estimada\ promedio &= f_{\bar{w}}(x) && (5.3)\\ Funcion\ real &= f_w(x) && (5.4)\\ sesgo(f_{\hat{w}}(x_t)) &= f_w(x_t) - f_{\bar{w}}(x_t) && (5.5) \end{aligned}$$

La varianza del modelo, σ^2, coincide con el ε y no tiene relación con los datos. Es decir : $\sigma^2 = \epsilon$.

El término $[sesgof_{\hat{w}}(x_t))]^2$ brinda una noción de cuánto el modelo puede, en promedio, ajustarse a la relación real entre x e y.

El objetivo es calcular la esperanza de que una curva se ajuste sobre un conjunto de entrenamiento específico, promediado sobre todos los posibles conjuntos de entrenamiento de tamaño N. Este ajuste promedio es llamado $f_{\bar{w}}$ (Ec. (5.3)). El sesgo es la diferencia entre la relación real y el ajuste promedio (Ec. (5.5)) y es elevado al cuadrado para que tenga igual magnitud que la varianza.

Con respecto a la varianza, considerando el ajuste promedio, interesa saber para todos los posibles ajustes, cuánto se desvían del ajuste esperado.

Formalmente, la varianza de una variable aleatoria es la esperanza de la variable aleatoria menos la media al cuadrado [8]. Para el caso del cálculo de la varianza de la función de estimación se tiene:

$$var(f_{\hat{w}}(x_t)) = E_{entren}[(f_{\hat{w}(entren)}(x_t) - f_{\bar{w}}(x_t))^2] \tag{5.6}$$

La definición en la Ec. (5.6) indica que la esperanza es calculada sobre todos los conjuntos de aprendizaje. Esta varianza muestra, para cada función obtenida con un conjunto de N datos de entrenamiento, cuánto difiere con un ajuste que se considera el promedio.

E_{entren} indica que se efectúa el cálculo sobre todos los conjuntos de entrenamiento de tamaño N.

$f_{\hat{w}(entren)}(x_t)$ es el ajuste sobre un conjunto específico de entrenamiento.

$f_{\bar{w}}(x_t)$ es la desviación de un ajuste específico de la función estimada promedio en x_t sobre todos los conjuntos de entrenamiento.

Para hallar la esperanza, el valor aleatorio sería la función estimada sobre un conjunto de entrenamiento (calculada sobre cada uno de los atributos).

Error de predicción esperado y derivación formal de las tres fuentes de error

Dada una observación x_t se considera la función de pérdida dada en la Ec. (5.7).

$$L(y, f_{\hat{w}}(x_t)) = (y - f_{\hat{w}}(x_t))^2 \tag{5.7}$$

Bajo la hipótesis de que la función de pérdida es el error cuadrático, la esperanza del error de predicción para x_t, (5.1), es expresada en la Ec. (5.8).

$$E_{train,y_t}[(y_t - f_{\hat{w}(train)}(x_t))^2] \tag{5.8}$$

A partir de (5.8) se puede deducir una expresión para el error de predicción esperado para la característica x_t:

$$E_{train,y_t}[(y_t - f_{\hat{w}(train)}(x_t))^2] = \\ = E_{train,y_t}[(y_t - f_{w(true)}(x_t) + f_{w(true)}(x_t) - f_{\hat{w}(train)}(x_t)))^2]$$

Utilizando la notación $f_{w(\hat{train})}(x_t) = \hat{f}$ y $f_w true(x_t) = f$ y la propiedad: $E(a+b)^2 = E(a^2) + 2.E(a.b) + E(b^2))$ se obtiene:

$$E_{train,y}[(y-f)^2] + 2E_{train,y}[(y-f)(f-\hat{f})] + E_{train,y}[(f-\hat{f})^2]$$

En el término $(y-f)$, el sumando y no es una función de los datos de entrenamiento. Tampoco lo es f, ya que es la verdadera función, siendo que se trata de la diferencia entre la función real y la verdadera función (ϵ). Por lo tanto, en el primer término $E_{entren,y}[(y-f)^2] = E(\varepsilon^2)$ y en el segundo término, considerando que $(y-f)$ y $(f-\hat{f})$ son variables aleatorias independientes, se verifica:

$$E_{train,y}[(y-f)(f-\hat{f})] = E_{train,y}[(y-f)]E_{train,y}[(f-\hat{f})]$$

Teniendo en cuenta que $E(\varepsilon) = 0$, puesto que es la esperanza sobre el ruido, se desprende que la $E(\varepsilon^2)$ es la varianza en todo el universo. Entonces se tiene:

$$E_{train,y}[(y-f)^2] = E(\epsilon^2) = \sigma^2$$

$$2E_{train,y}[(y-f)(f-\hat{f})] = 2E_{train,y}(\epsilon)E_{train,y}(f-\hat{f}) = 0$$

(porque $E_{train,y}(\epsilon) = 0$)

$E_{train,y}[(f-\hat{f})^2] = MSE(\hat{f})$ (*mean squared error* de $\hat{f}$)

Se obtiene:

$$E_{train,y_t}[(y_t - f_{\hat{w}(train)}(x_t))^2] = \sigma^2 + MSE(\hat{f}) \tag{5.9}$$

El término de ruido es independiente y la esperanza de variables independientes es la multiplicación de esas esperanzas.

A continuación se demuestra que: $MSE[\hat{f}] = sesgo^2(\hat{f}) + var(\hat{f})$

$MSE[f_{\hat{w}(train)}(x_t)] = E_{train}[(f_{w(true)}(x_t) - f_{\hat{w}(train)}(x_t))^2] =$
$= E_{train}[((f_{w(true)}(x_t) - f_{\bar{w}}(x_t)) + (f_{\bar{w}}(x_t) - f_{\hat{w}(train)}(x_t)))^2]$

Notación:

$\bar{f} = f_{\bar{w}}(x_t)$ función estimada promedio en todos conjuntos de entrenamiento

$MSE[\hat{f}] = E_{train}[(f_{w(true)}(x_t) - f_{\hat{w}(train)}x_t))^2] =$
$E_{train}[(f-\bar{f})^2] + 2E_{train}[(f-\bar{f})(\bar{f}-\hat{f})] + E_{train}[(\bar{f}-\hat{f})^2]$

En el primer sumando, la función f es la verdadera relación y $\bar{f}$ es el promedio de los ajustes obtenidos sobre todos los conjuntos de entrenamiento. Considerando a $\bar{f}$ la función ajustada sobre un conjunto de entrenamiento, ésta puede ser vista como la media de una variable aleatoria, por lo que es independiente de los datos, promediada sobre todos los conjuntos de entrenamiento, entonces no hay esperanza (nada es aleatorio a nivel de las tendencias de los conjuntos de datos). Por lo tanto, $f - \bar{f}$ es independiente de los datos de entrenamiento:

$$E_{train}[(f - \bar{f})^2] = (f - \bar{f})^2 = sesgo^2(f)$$

En el segundo término, aplicando el mismo razonamiento, $f - \bar{f}$ sale como factor de la esperanza:

$$2E_{train}[(f - \bar{f})(\bar{f} - \hat{f})] = 2(f - \hat{f})E_{train}[\bar{f} - f]$$

En forma análoga, del factor $(\bar{f} - \hat{f})$, la función estimada promedio $\bar{f}$, sale fuera de la esperanza y sólo queda la esperanza de $\hat{f}$ la cual es $\bar{f}$. Esto es debido a que tomando un ajuste sobre un conjunto de entrenamiento específico, si se considera la esperanza sobre todos los conjuntos de entrenamiento resulta que es la definición de $\bar{f}$ (el ajuste promedio). Por lo tanto se obtiene:

$$2(f - \hat{f})E_{train}[f - \hat{f}] = 2(f - \hat{f})(\bar{f} - \bar{f}) = 0$$

El tercer término es la esperanza de una función menos su media, todo ello elevado al cuadrado, es decir, es una variable aleatoria menos su media al cuadrado, lo cual coincide con la definición de varianza: $E[(\hat{f} - \bar{f})^2] = var(f)$

De esta forma se obtiene (5.10).

$$MSE[\hat{f}] = sesgo^2(\hat{f}) + var(\hat{f}) \tag{5.10}$$

Finalmente sustituyendo (5.9) en ((5.10)) se obtiene:

$$\textit{Error de predicción esperado para } x_t = \sigma^2 + sesgo^2(\hat{f}) + var(f) \tag{5.11}$$

En conclusión, si x_t es una característica de los datos, el promedio del error de predicción en x_t es lo indicado en la Ec. (5.11).

5.3. Evaluación de los modelos

Luego de construido el modelo, es necesario evaluarlo de la manera más exacta posible. La separación de las observaciones en conjuntos de entrenamiento y de *testing* permite realizar tal evaluación. Es conveniente que ambos conjuntos sean disjuntos, aunque no siempre es posible.

Una opción más realista sería realizar la evaluación en función de los costos asociados a los errores del modelo. En este caso, se selecciona el modelo que tiene menor costo asociado a los errores, en lugar de elegir una hipótesis que tiene el menor número de errores. En contextos de regresión, no tiene sentido comparar los valores predichos si son iguales a la clase real, pues son valores numéricos, por lo que se utiliza la distancia entre el valor predicho y el real (o el cuadrado de ella, entre otros). El mejor modelo es aquel que minimiza la distancia media entre ambos valores de los puntos utilizados para la evaluación [20]. Para los métodos descriptivos se utilizan otras métricas: complejidad del modelo y de los datos a partir del modelo o, en el caso de clústeres, se utilizan mediciones de la aglomeración.

En ocasiones, no se dispone de una cantidad suficiente de datos, por lo que se puede considerar un conjunto de observaciones que forman parte de la evidencia y dividirlo en dos nuevos subconjuntos: de entrenamiento y de *testing*.

5.3.1. Técnicas de evaluación

A continuación se describen algunas técnicas de evaluación.

Validación simple

La validación simple es el método más básico de evaluación, donde se separan los datos, en un conjunto de entrenamiento y en otro de *testing*. Este es utilizado para evaluar el modelo que fue obtenido, a partir del conjunto de entrenamiento. La validación simple reserva un porcentaje de los datos como conjunto de validación y no son utilizados para construir el modelo (puede variar del 5 *al* 50 %) [3]. La división de los datos en los dos grupos, es aleatoria. La crítica a este enfoque, es que el modelo de validación es desarrollado con un sólo conjunto de datos, lo que no conduce a modelos realmente útiles. Para solucionar este problema se utiliza la validación cruzada.

Validación Cruzada

El objetivo de la validación cruzada es definir un conjunto de *test*, para validar un modelo durante la etapa de entrenamiento. Es una alternativa a la validación simple, cuando no se poseen muchos datos. Los datos se dividen en dos conjuntos equitativos para estimar la precisión predictiva del modelo. Inicialmente, se construye un modelo con el primer conjunto y se predicen los resultados en el segundo. Se calcula el ratio de error. A continuación se procede de igual forma intercambiando los conjuntos, calculándose un segundo ratio de error. Finalmente se construye un modelo con todos los datos y se calcula el promedio de los ratios, lo que puede resultar de utilidad para mejorar la precisión [3].

Existen varios métodos de validación cruzada. Se mencionan *LOOCV*, *K-fold Cross* y *LPO*.

LOOCV (Leave One Out Cross Validation, validación cruzada dejando uno afuera)

El método *LOOCV* aparta una observación del conjunto de entrenamiento para ser utilizada para validar. Luego, el proceso se repite cambiando al punto que fue apartado, como conjunto de validación. El error de *test* se basa en una observación, por lo que dicho error es demasiado variable. Por lo tanto, se trata de un estimador pobre del error verdadero. Sin embargo, si el proceso es repetido para todas las observaciones, apartando una diferente para la validación, se obtienen tantos errores de *test* como cantidad de observaciones haya. Estos errores de *test* se pueden promediar obteniéndose el error de *test* general. La ventaja de este método, es que en el proceso de determinación de los coeficientes de regresión, éstos tendrán menor sesgo, debido a que el modelo es ajustado en todas las observaciones, excepto en una. La otra ventaja, radica en que los valores de los coeficientes no varían al cambiar los conjuntos de entrenamiento (ya que casi no existe diferencia entre ellos) [3].

Presenta la desventaja que para grandes conjuntos de datos, es computacionalmente costoso.

Validación Cruzada k-fold (k-fold Cross)

La validación cruzada *k-fold* es un método intermedio entre un conjunto de validación y LOOCV. Los datos se dividen aleatoriamente en K grupos. Un

grupo se reserva como un conjunto de *testing*. Los $(K-1)$ grupos restantes se utilizan para entrenamiento del modelo. Esto genera un error parcial. El procedimiento es repetido K veces, cambiando el conjunto de *testing* en cada iteración. El error final, se calcula como el promedio de los k errores parciales [3].

Una ventaja de este método sobre $LOOCV$ es que requiere menos esfuerzo computacional. También proporciona estimaciones más precisas del promedio del error de *test*.

$LOOCV$ tiene menor sesgo en el error de *test* (pues en cada modelo los conjuntos de entrenamiento son casi iguales). Pero las estimaciones calculadas en cada muestra de la validación cruzada están altamente correlacionadas y, por lo tanto, la media de esos estimados tendrá una varianza alta. En cambio, en la validación cruzada *K-fold*, la varianza es menor en el error de *test* pues hay menos superposición en los conjuntos considerados. Se ha determinado que tomando los valores $K = 5$ o $K = 10$ en la validación cruzada *K-fold*, el error de *test* estimado tiene menor sesgo y varianza [47].

Validación LPO (Leave-p-out)

La validación cruzada con *p-out* deja p observaciones como conjunto de validación. Es computacionalmente costosa pero los métodos distribuidos utilizados para *Big Data* pueden ser aplicados en esta técnica [1].

5.3.2. Evaluación de los modelos de clasificación

Para el caso de la clasificación se espera que un buen clasificador tenga por lo menos una performance mejor que uno que trabaja aleatoriamente. Por ejemplo un clasificador binario aleatorio tiene un error de $0,5$ y un clasificador en K clases tiene un error de $(1 - 1/K)$.

El error de la muestra se define como el porcentaje de errores sobre el total de las observaciones disponibles. Si $h(x)$ es el modelo y $f(x)$ la función real, el error de la muestra s es el expresado en la Ec. (5.12).

$$error_{s(h)} = \frac{1}{n} \sum_{x \in S} \delta(f(x) \neq h(x)) \tag{5.12}$$

El valor n representa el total de casos.

En la Ec. (5.13) se expresa el error verdadero.

$$error_v(h) = Probabilidad_{x \in F}[(f(x) \neq h(x))] \tag{5.13}$$

El error verdadero de una hipótesis h con respecto a la función objetivo f y una distribución F, es la probabilidad de que h prediga un resultado diferente a f en una muestra tomada de acuerdo con dicha distribución F.

Evaluación de los modelos de Precisión y recuperación

Los modelos de recuperación de la información son aquellos que para una determinada consulta devuelven al usuario un conjunto de posibles soluciones. Los ejemplos más notorios son los motores de búsqueda en la *Web* y los sistemas de recomendación. Estos modelos también pueden ser considerados como un tipo de clasificador. La precisión (o relevancia) y la exhaustividad (también llamada recuperación o alcance) tienen diferentes objetivos. La precisión es el porcentaje de ítems seleccionados que son relevantes. Si la cantidad de objetos devueltos por el sistema, son representados mediante $\#s$ y la cantidad de los que son relevantes por $\#rs$, la precisión P se define en la Ec. (5.14).

$$P = \frac{\#rs}{\#s} \tag{5.14}$$

Si la precisión es alta, es más probable que el usuario que recibe las sugerencias del sistema no pierden demasiado de su tiempo mirando las cosas que son irrelevantes, no obstante, se asume que ha recibido más información útil de la que necesita.

La precisión por sí sola no es una buena métrica en algunos casos. Por ejemplo, si un clasificador binario con salida en dos clases: A y B, clasifica todo como A y si se conoce de antemano que el 95 % de los datos son de A, se obtiene una precisión muy alta. Sin embargo, el clasificador no es muy acertado, ya que no cataloga nada como B.

Para complementar a la precisión se utiliza la exhaustividad, definida como el porcentaje de ítems relevantes seleccionados. En la Ec. (5.15) se define el alcance R, siendo $\#r$ la cantidad de objetos relevantes y $\#rs$ el total de ítems relevantes seleccionados.

$$R = \frac{\#rs}{\#r} \tag{5.15}$$

Tanto para la exhaustividad, como para la precisión, para evitar la división

entre cero, se puede sumar 1 al numerador y al denominador.

Matriz de confusión

En base a los objetos recuperados y a la relevancia de ellos, es posible realizar otro tipo de evaluaciones, a través de un cuadro donde se enumeran la cantidad de casos, para los escenarios posibles entre recuperación y relevancia, teniendo en cuenta los errores y aciertos del sistema. Por ejemplo, si los elementos recuperados son relevantes, se tiene un caso de verdadero positivo (TP), y, si no lo son, se trata de un escenario de falso positivo (FP). Si no se recuperan elementos irrelevantes, se trata de verdadero negativo (TN), y, si son relevantes y no recuperados se denominan falso negativo (FN). Las posibilidades se ven en la Tabla 5.1. Las cuatro posibilidades definen un cuadro denominado matriz de confusión.

	Relevante	Irrelevante
Recuperados	TP	FP
No Recuperados	FN	TN

Tabla 5.1: Matriz de confusión

Tomando en cuenta esta matriz, la precisión y el alcance se reescriben de acuerdo a las ecuaciones (5.16) y (5.17), respectivamente.

$$P = \frac{TP}{(TP + FP)} \tag{5.16}$$

$$R = \frac{TP}{(TP + FN)} \tag{5.17}$$

5.3.3. Balance entre precisión y recuperación

Al estudiar el balance entre precisión y exhaustividad, es importante observar qué sucede en los extremos. Un clasificador optimista, es aquel, que a la mayoría de los *inputs* los clasifica positivamente. La mayor parte de los positivos son calificados como tales. Esto significa que tiene una exhaustividad perfecta, no implicando lo mismo en términos de precisión. El clasificador pesimista califica a muy pocos positivos, solamente los clasifica si está seguro de que son positivos. Tienen alta precisión, pues los calificados positivos, es muy probable que lo sean, sin embargo, poseen muy baja exhaustividad.

Se puede establecer un balance entre modelos de baja precisión y los de alta precisión, o de un modelo de alto alcance a uno de bajo alcance. Para ello, se utilizan probabilidades para el *trade off* entre precisión y alcance. Si se considera un valor de la probabilidad de que la clasificación sea $+1$, se establece un umbral a partir del cual, si la probabilidad es mayor que aquel, entonces es $+1$, y, en caso contrario, es -1. De acuerdo al umbral, se puede definir si el clasificador es optimista o pesimista.

Si el umbral es $0,999$, predice positivo cuando está completamente seguro y predice al resto de los casos negativamente, por lo que se le califica como un clasificador pesimista. Si el umbral es muy bajo, por ejemplo, 0.001, significa que la mayoría de los *inputs* son clasificados positivamente, por lo que se le califica como un clasificador optimista.

Por lo tanto, variando al umbral entre *0* y *1*, se puede alternar de un clasificador optimista a uno pesimista.

Curvas de precisión vs alcance para interpretar la performance de un clasificador

Generalmente, pueden existir regiones sobre las cuales uno de los clasificadores es más efectivo que el otro. En la Figura 5.5 se comparan dos clasificadores A y B. El punto de corte de las dos curvas, definen dos regiones, donde en cada una de ellas, uno de los clasificadores tiene una performance superior al del otro, alternándose.

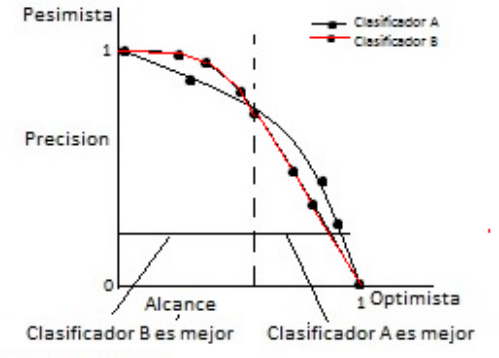

Figura 5.5: Comparación de clasificadores

Curvas ROC

Las curvas *ROC (Receiver Operating Characteristic)*, representan gráficamente la sensibilidad de un problema de decisión binaria, de acuerdo a la

variación del umbral de discriminación [48]. La curva ROC describe la relación entre precisión y recuperación, a lo largo de un balance que muestra cómo la recuperación del sistema crece conforme la precisión decrece. Otra interpretación de este gráfico, es la representación de la razón de verdaderos positivos frente a la razón de falsos positivos, considerados de acuerdo a la variación del umbral de discriminación (valor a partir del cual se decide que un caso es positivo).

El análisis de la curva ROC, o simplemente análisis ROC, proporciona herramientas para seleccionar los modelos posiblemente óptimos y descartar otros modelos independientemente del costo de la distribución de las dos clases sobre las que se decide. La curva ROC es independiente de la distribución de las clases en la población. Este tipo del análisis se relaciona de forma directa y natural con el análisis de costo/beneficio en toma de decisiones diagnósticas. El eje de las ordenadas representa a los verdaderos positivos (TP) (todos los objetos que deberían ser aceptados por el filtro). El eje de las abscisas representa los falsos positivos (FP). En la primer imagen de la Figura 5.6 se representa el balance $TP - FP$, donde se debe aceptar un 20 % de FP para obtener un 80 % de TP (a los efectos de un buscador Web esto implicaría una baja performance).

El punto $(0, 0)$ indica que no se devuelve ningún TP aunque no hubo ningún FP. El punto $(1, 1)$ indica que se devuelve todos TP si bien son todos FP, es decir, permite pasar todo el contenido, no filtrando nada.

En la segunda imagen de la Figura 5.6 se representa la calidad de los modelos donde el que tiene mayor área (línea azul), es el mejor. Cada curva, dependiendo del punto donde comienza a crecer aceleradamente, denominado «punto codo», indica qué tipo de clasificador es [49]. Se puede apreciar dicho punto en la tercera imagen de la Figura 5.6.

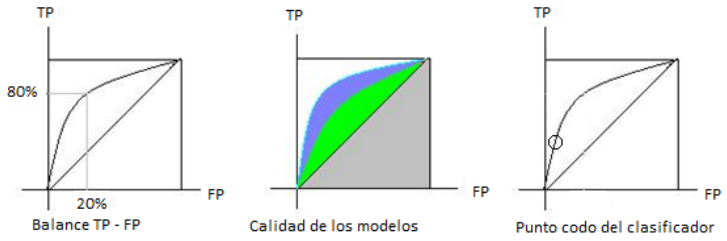

Figura 5.6: Curvas ROC

Cuanto más pequeño sea el valor de los FP para alcanzar el punto codo,

mejor será el clasificador. Si bien la curva que se obtiene es de utilidad, se acostumbra considerar el área bajo la curva. Para una curva que se comporta adecuadamente, su área suele tender a 1 (que es el área total de un cuadrado de lado igual a la unidad). Cuanto más próximo al valor 1 se encuentre, mejor filtro se obtendrá y, cuanto más próximo a $\frac{1}{2}$, se trata de un modelo casi aleatorio.

5.3.4. Conjuntos de entrenamiento, evaluación y costos

En lugar de separar al total de observaciones en dos conjuntos, se divide en 3 subconjuntos: entrenamiento, validación y *testing*. A partir de ellos, se estudia un procedimiento para seleccionar el modelo con mejor desempeño y probarlo. En el caso de que la cantidad de datos sea suficiente, se prepara el modelo con el conjunto de entrenamiento, se seleccionan entre varios modelos a través de un parámetro sobre el *set* de validación y, finalmente, se prueba sobre el conjunto de *testing* (el conjunto de *testing* nunca debería ser utilizado para las instancias del aprendizaje).

El conjunto de entrenamiento proporciona una curva de ajuste $\hat{w}_\lambda$. La idea es aplicar un *test* de performance sobre el conjunto de validación a los efectos de seleccionar λ^* para que $\hat{w}^*_\lambda$ presente el menor error de *test*. Luego se evalúa el error generalizado de $\hat{w}^*_\lambda$ sobre el conjunto de *testing*. Los errores pueden determinarse del siguiente modo:

Error de entrenamiento

$$Error_{train}(w) = \frac{1}{2m}\sum_{i=1}^{m}(h_w(x^{(i)}) - y^{(i)})^2 \tag{5.18}$$

Error de validación

$$Error_{valid}(w) = \frac{1}{2m_{valid}}\sum_{i=1}^{m_{valid}}(h_w(x^{(i)}_{valid}) - y^{(i)}_{valid})^2 \tag{5.19}$$

Error de test

$$Error_{test}(w) = \frac{1}{2m_{test}}\sum_{i=1}^{m_{test}}(h_w(x^{(i)}_{test}) - y^{(i)}_{test})^2 \tag{5.20}$$

Las dimensiones para la partición en los tres conjuntos pueden ser por ejemplo: conjunto de entrenamiento 80 %, validación 10 % y *testing* 10 %. También

puede ser 50 %, 25 %, 25 % respectivamente [3]. En caso de no disponerse de suficientes puntos para el conjunto de *testing*, se puede usar la validación cruzada como técnica para controlar el balance entre el ajuste y la capacidad predictiva (en este caso no se está evaluando el modelo sino que se está controlando dicho balance).

En la validación cruzada *K-fold*, las N observaciones son divididas en k conjuntos formados aleatoriamente de tamaño $\frac{N}{k}$, quedando fijos para todo el proceso. Cada uno de dichos subconjuntos será un *set* de validación. Las observaciones restantes se usan para entrenar el modelo para cada valor de λ.

Dado un valor λ, se considera un bloque de entre los K subconjuntos y se ajusta la curva con el resto de los datos obteniéndose un $\hat{f}$ en función de λ y N. Del bloque de validación se obtiene un $error_k(\lambda)$. Esto se repite para cada uno de los otros bloques. Luego se calcula el error de validación cruzada de λ, que es el promedio del error obtenido en cada uno de los bloques.

Algoritmo para validación cruzada K-fold

Dados N puntos y un número K, dividir aleatoriamente el *dataset* en N/k conjuntos. Para $k = 1 \dots K$

Elegir el subconjunto de validación k y considerar los restantes como conjuntos de entrenamiento.

Estimar $\hat{w}_{\lambda^{(k)}}$ en los conjuntos de entrenamiento.

Calcular el error de validación en el subconjunto de validación $k : error_k(\lambda)$.

Calcular el error promedio de la VC (validación cruzada)

$VC(\lambda) = (1/K)(\sum_{k=1}^{K} error_k(\lambda))$

Luego, repetir el proceso para cada uno de los λ a ser considerados, eligiéndose el λ que minimiza el error de *VC*.

Evaluación de hipótesis basada en costo

Como ejemplo introductorio, suponer que existe un sistema al cual se desea entrenar para detectar y eliminar correos electrónicos entrantes que son *spam*. En este caso se puede presentar situaciones en las cuales es preferible que no detecte un correo *spam* y lo deje pasar a que identifique un correo bueno como *spam* y lo elimine. Esta evaluación es común en diagnóstico de enfermedades y detección de intrusos. En este tipo de evaluación, se calcula el costo de un

error de un clasificador. Como se había expresado, la precisión no es la mejor medida para evaluar un modelo. La manera más habitual de expresar estos costos, en términos de clasificación, es a través de una matriz de costos. Si el clasificador tiene n categorías, la matriz es cuadrada de dimensión n. Las filas interpretan lo que predice el clasificador, las columnas el comportamiento real y en la celda correspondiente, irá un valor que indica el costo del error o del acierto (en el caso de acierto es cero). A su vez se construye otra matriz llamada «de confusión», con la misma estructura donde se enumeran los casos que ocurren. El costo viene dado por la siguiente función (Ec. (5.21)) [3]:

$$costo = \sum_{1 \leq i \leq n, 1 \leq j \leq n} C_{i,j}.M_{i,j} \tag{5.21}$$

La notación anterior implica la multiplicación elemento a elemento de las matrices de confusión y de costos.

Utilizando la ecuación (5.21) se determina el costo del clasificador, para el cual es deseable que sea lo menor posible.

Ejemplo 5.1

Matriz de costos C (se refiere a acciones tomadas ante un paciente y el costo que un error implica, en este caso el paciente puede ser alta, observación o CTI).

		Real	
Matriz de Confusión	Alta médica	Observación	CTI
Alta médica	30	3	2
Observación	12	32	1
CTI	3	5	12

Estimado

		Real	
Matriz de Costos	Alta médica	Observación	CTI
Alta médica	0	15000	100000
Observación	1500	0	50000
CTI	3000	2000	0

Estimado

Figura 5.7: Matriz de confusión y de costos asociados

En el ejemplo de la Figura 5.7 el costo es: 332000

Anexo 6

Métodos empleados en aprendizaje automático

En este anexo se analizan algoritmos que emplean aprendizaje automático para su funcionamiento, describiéndose detalladamente algunos de los conceptos introducidos en el *Capítulo 4, Técnicas generales de Data Mining*.

6.1. Clasificadores

En esta sección se analiza con mayor detalle el perceptrón y la *SVM* en la clasificación.

6.1.1. El perceptrón

El perceptrón es una función utilizada para efectuar clasificaciones binarias. La decisión se realiza a través de un umbral apropiado, que define dos regiones, donde son asignadas las observaciones [4]. Se define por medio de una función f, un vector $\mathbf{x}$ y otro vector de pesos $\mathbf{w}$ tales que, cada componente x_i de $\mathbf{x}$ tiene asociado un peso w_i de w y un valor de umbral θ, que verifican la Ec. (6.1).

$$f(\mathbf{x},\mathbf{w}) = +1 \; si \; \mathbf{x}.\mathbf{w} \geq \theta \; y \; si \; no \; f(\mathbf{x},\mathbf{w}) = -1 \tag{6.1}$$

La ecuación $\mathbf{x}.\mathbf{w} = \theta$ es un hiperplano que divide al espacio en dos partes, el cual permite clasificar a los puntos. Puede ocurrir que los datos se distribuyan con cierta geometría en el espacio, de manera tal que no sea posible, en primera

instancia, una solución basada en un hiperplano. En algunos casos, en los cuales no es evidente tal separación, existen soluciones en las que se pueda aplicar el perceptrón. Por ejemplo, si los datos se distribuyen en dos anillos concéntricos y disjuntos, se pueden usar coordenadas polares y la separación del hiperplano puede realizarse en los puntos cuyo radio es mayor o menor que un determinado umbral [4].

Algoritmo para el perceptrón

En el algoritmo se inicializa $\mathbf{w}$ con todas sus componentes cero. Se asume que el umbral es cero ($\theta = 0$). Se define un parámetro de razón de aprendizaje η real positivo el cual no puede ser pequeño pues la convergencia sería lenta, ni grande, pues el modelo puede saltar de una solución a otra, oscilando alrededor del óptimo, provocando una convergencia lenta. Por otro lado, puede ocurrir que no haya convergencia.

A continuación, se considera cada dato de entrenamiento $(\mathbf{x}, y)$ donde el punto $\mathbf{x}$ tiene asociada una etiqueta y.

Sea $y^1 = \mathbf{w}.\mathbf{x}$. Si y^1 e y tienen el mismo signo, $\mathbf{w}$ permanece incambiado. Si y^1 e y tienen distinto signo o $y^1 = 0$, se reemplaza $\mathbf{w}$ por $\mathbf{w} + \eta y \mathbf{x}$. Esta transformación mueve al hiperplano en el sentido de que $\mathbf{x}$ permanece en el lado correcto del semi hiperplano (Figura 6.1).

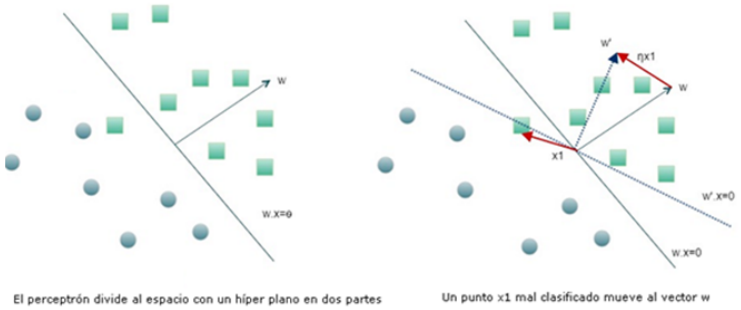

Figura 6.1: El perceptrón (Origen: *Mining of Massive Datasets Leskovec J, Rajaraman A, Ullman J - 2014*)

Ejemplo 6.1

El siguiente ejemplo es una adaptación de un caso proveniente de *Mining of Massive Datasets Leskovec J, Rajaraman A, Ullman J)* Se considera los datos de entrenamiento de la Tabla 6.1. Se entrena un perceptrón para decidir si un correo electrónico es *spam* o no. El conjunto de entrenamiento consiste

X	Y	Recompensa	El	De	Rusia	Y
a	1	1	0	1	1	+1
b	0	0	1	1	0	-1
c	0	1	1	0	0	+1
d	1	0	0	1	0	-1
e	1	0	1	0	1	+1
f	1	0	1	1	0	-1

Tabla 6.1: Tabla de entrenamiento para email *spam*

en pares $(\mathbf{x}, y)$ donde $\mathbf{x}$ es un vector formado por ceros y unos $x_i = 1$, si la palabra x_i está presente en el email y $x_i = 0$ cuando no lo está). La etiqueta $y = +1$, significa que el correo es reconocido como *spam* e $y = -1$, cuando no lo es. Se toma como ratio de aprendizaje $\eta = \frac{1}{2}$.

Se inicializa el vector $\mathbf{w}$, $\mathbf{w} = [0, 0, 0, 0, 0]$, y se considera la primera observación, $\mathbf{x} = [1, 1, 0, 1, 1]$, la cual corresponde al caso $\mathbf{x} = \mathbf{a}$. Por lo tanto, $\mathbf{w.a} = 0$, es decir, $y^1 = 0$. De acuerdo con el algoritmo, se cambia $\mathbf{w}$ por $\mathbf{w} + \eta y \mathbf{x}$. Geométricamente w se mueve en la dirección del vector $\mathbf{x}$=$\mathbf{a}$.

$$\mathbf{w} \longleftarrow \mathbf{w} + (\frac{1}{2})(+1)\mathbf{a}$$

$$\mathbf{w} = [0, 0, 0, 0, 0] + [\frac{1}{2}, \frac{1}{2}, 0, \frac{1}{2}, \frac{1}{2}] = [\frac{1}{2}, \frac{1}{2}, 0, \frac{1}{2}, \frac{1}{2}]$$

A continuación se considera el vector $\mathbf{x} = \mathbf{b}$, obteniéndose:

$$\mathbf{w.b} = [\frac{1}{2}, \frac{1}{2}, 0, \frac{1}{2}, \frac{1}{2}].[0, 0, 1, 1, 0] = \frac{1}{2}$$

Al ser la etiqueta de $\mathbf{b}$ igual a -1, se considera que está mal clasificado, por lo tanto:

$$\begin{aligned} \mathbf{w} = \mathbf{w} + (1/2)(-1)\ \mathbf{b} &= \\ [1/2, 1/2, 0, 1/2, 1/2] - &\quad [0, 0, 1/2, 1/2, 0] \\ &= \quad [1/2, 1/2, -1/2, 0, 1/2] \end{aligned}$$

Sucesivamente, se aplica el mismo procedimiento con los demás vectores. De esta manera se construye un vector $\mathbf{w}$ que clasifica correctamente todos los vectores del conjunto de entrenamiento.

Convergencia del perceptrón

La convergencia del perceptrón no está garantizada, ya que depende de que los datos sean linealmente separables. Aún existiendo un hiperplano que separe los dos tipos de muestras, el vector **w** puede repetirse y así, entrar en un *loop*. Sin embargo, si se trata de datos masivos, la repetición de un vector puede ocurrir luego de un gran intervalo de tiempo y si el algoritmo termina bajo alguna condición de parada, no se podría detectar esta repetición. Para estos casos particulares, el algoritmo *Winnow* asegura la convergencia [4].

Algoritmo Winnow

El algoritmo *Winnow* funciona considerando un vector **x** formado por $0's$ y $1's$ y las etiquetas de salida son *+1* y *-1*. Sólo produce valores positivos para los componentes de los pesos. Se computa **w.x** y se compara con θ .

- Si $\mathbf{w.x} > \theta$ e $y = +1$ o $\mathbf{w.x} < \theta$ e $y = -1$, el ejemplo está bien clasificado y no se modifica **w**.
- Si $\mathbf{w.x} \leq \theta$ e $y = +1$, el **w** es muy bajo donde las componentes de **x** son 1 y se duplican para dichos componentes los valores del peso. Es decir, si la componente $x_i = 1$, w_i pasa a valer $2 * w_i$.
- Si $\mathbf{w.x} \geq \theta$ e $y = -1$ el **w** es muy alto donde las componentes de x son 1, por lo que se dividen entre dos las correspondientes componentes del peso. Es decir, si $x_i = 1$, w_i, pasa a valer $w_i/2$

Esto se aplica a todos los elementos del conjunto de entrenamiento. La elección del factor 2 es para aumentar el peso en caso de que el producto resultara un número bajo, y la clase es +1. Es $\frac{1}{2}$ cuando se quiere disminuir el peso en el caso de que el producto sea mayor que el umbral y la etiqueta es -1.

Variación del umbral θ

En ocasiones, no se conoce de antemano cuál es el mejor valor para θ, por lo que es incorporada al vector **w** como una componente más [4]. En tal caso se tiene $\mathbf{w} = [w_1, \ldots, w_d, \theta]$ y $\mathbf{x} = [x_1, \ldots, x_d, -1]$. De este modo se evalúa $\mathbf{w.x} - \theta \geq 0$. También puede adaptarse *Winnow* a esta modalidad, considerando los vectores modificados de **x** y **w**, teniendo en cuenta que si la

etiqueta es positiva y se necesita aumentar los w_i, se asigna $w_i = 2 * w_i$ y θ pasa a valer $\frac{\theta}{2}$.

Perceptrón multi clase

Sea k, el número de posibles etiquetas para un conjunto de entrenamiento que está formado por k clases ($k \geq 3$). Se comienza con la clase i, y sea θ el umbral común a todas ellas. Se entrena el perceptrón para cada clase, obteniéndose k vectores de pesos w donde cada uno de ellos define una clase (cada una identificada por w_i). Dados una clase w_i y un punto $\mathbf{x}$ que se pretende clasificar, se calcula $\mathbf{w_i}.\mathbf{x}$ para todo $i = 1, 2, ..., k$. y se considera el mayor valor obtenido. Si éste supera al umbral, $\mathbf{x}$ pertenece a la clase a la que correspondió dicho máximo, si no, el punto no pertenece a ninguna [4].

Limitaciones del perceptrón

En la Figura 6.2 se ilustran tres limitaciones del algoritmo.

Caso A: asumiendo el costo de sobre ajustar el modelo, se puede resolver los puntos que están mal clasificados.

Caso B: el punto rojo puede pertenecer a la clase cuadrados o a la clase círculos dependiendo del hiperplano que se considere.

Caso C: ilustra el caso en que generalmente más de un hiperplano pueden separar las clases (si existen) y algunos, al estar desviados hacia una de las dos clases pueden no ser los óptimos, ya que el perceptrón converge conforme el hiperplano separador alcanza la región entre las clases.

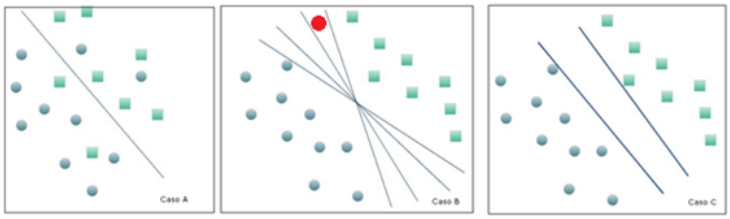

Figura 6.2: Limitaciones del perceptrón (Origen: *Mining of Massive Datasets Leskovec J, Rajaraman A, Ullman J - 2014)*

Perceptrones con MapReduce

Consiste en ejecutar productos escalares **w.x** en paralelo.

Función *Map* : Cada tarea de *Map* conoce el vector **w** en todo instante y calcula para cada vector **x** el producto $\boldsymbol{w.x}$ comparándolo con la etiqueta *(+1 o -1)* . Si coincide, no es necesario formar ningún par (llave, valor), en caso contrario, se origina un par $(i, \eta y x_i)$ que va a ser la coordenada que va a modificarse. Si $x_i = 0$, tampoco es necesario crear el par. El producto $\eta y x_i$ es el incremento que se debe agregar a w_i.

Función *Reduce*: suma todos los incrementos asociados a cada componente de cada una de las observaciones. Si el vector **w** cambia se debe ejecutar nuevamente el algoritmo con el mismo conjunto de entrenamiento.

6.1.2. Máquinas de vectores de soporte (SVM)

Las limitaciones mencionadas del perceptrón a la hora de separar regiones en el espacio de los datos, pueden ser solucionadas por medio de las máquinas de vectores de soporte *(Suport Vector Machine, SVM)*.

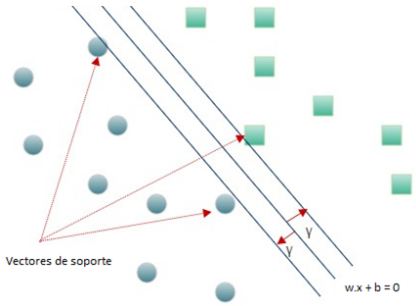

Figura 6.3: Máquinas de vectores de soporte (Origen: *Mining of Massive Datasets Leskovec J, Rajaraman A, Ullman J - 2014)*

Una *SVM* selecciona el hiperplano con el mayor margen entre éste y los conjuntos de entrenamiento. En la Figura 6.3, el coeficiente b de la ecuación $\mathbf{wx} + b = 0$ cumple la misma función que θ en el perceptrón. Los vectores de soporte son los puntos que definen los dos hiperplanos paralelos. En el caso del plano se necesitan dos puntos para definir la recta en una clase y otro para obtener el paralelo en la otra clase (si la dimensión del espacio es d se requieren, al menos, $d+1$ vectores de soporte [4]). La función objetivo se plantea a continuación.

Función objetivo

Dado un conjunto de entrenamiento $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$, se busca maximizar γ (variando w y b) con la restricción de que $y_i(w.x_i+b) \geq \gamma$ para todo $i = 1, 2, ..., n$.

El valor de y_i puede ser *1* o *-1* y determina de qué lado del hiperplano está el punto x_i. Si $y = +1$ entonces $\mathbf{w.x} \geq \gamma$ y si $y = -1$ entonces $\mathbf{w.x} < -\gamma$. El planteo anterior no funciona bien ya que aumentando w y b también aumenta γ. Esto se resuelve normalizando $\mathbf{w}$ (dividiendo al vector por su norma $\frac{\mathbf{w}}{\|\mathbf{w}\|}$). Normalizar al vector implica transformarlo en un vector unitario. Se busca maximizar γ, que es el factor que multiplica $\frac{\mathbf{w}}{\|\mathbf{w}\|}$, el cual separa a los hiperplanos paralelos.

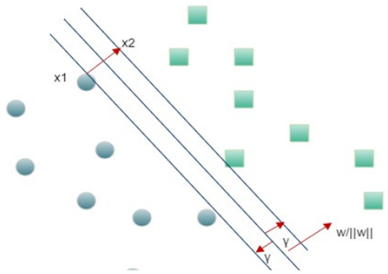

Figura 6.4: Replanteo de la función objetivo de la SVM (Origen: *Mining of Massive Datasets Leskovec J, Rajaraman A, Ullman J - 2014)*

En la Figura 6.4 se puede apreciar a $\mathbf{x_1}$, como uno de los vectores de soporte y $\mathbf{x_2}$ su proyección sobre el hiperplano más alejado. La distancia entre $\mathbf{x_1}$ y $\mathbf{x_2}$ en unidades $\frac{\mathbf{w}}{\|\mathbf{w}\|}$ es 2γ, obteniéndose la Ec. (6.2).

$$x_1 = x_2 + 2\gamma \frac{\mathbf{w}}{\| \mathbf{w} \|} \tag{6.2}$$

Al estar $\mathbf{x_1}$ en el hiperplano definido por $\mathbf{w.x} + b = +1$, sustituyendo $\mathbf{x_1}$ por $\mathbf{x}$ se obtiene la Ec. (6.3).

$$\mathbf{w}.(x_2 + 2\gamma \frac{\mathbf{w}}{\| \mathbf{w} \|}) + b = 1 \tag{6.3}$$

Reagrupando términos, se obtiene la Ec. (6.4).

$$w.x_2 + b + 2\gamma \frac{\mathbf{w}}{\| \mathbf{w} \|} = 1 \tag{6.4}$$

Considerando que $\mathbf{w}.\mathbf{x}_2+b=-1$ (porque pertenece al hiperplano $w.x+b=-1$), se obtiene $\gamma\frac{\mathbf{ww}}{\|\mathbf{w}\|}=1$. Debido a que $\mathbf{w}.\mathbf{w}=\parallel \mathbf{w} \parallel^2$, se llega a la Ec. (6.5).

$$\gamma = \frac{1}{\parallel \mathbf{w} \parallel} \tag{6.5}$$

Las consideraciones anteriores permiten reformular el problema de optimización, pasando de un problema de maximización a otro de minimización (de $\parallel \mathbf{w} \parallel$), puesto que al disminuir éste, el parámetro γ aumenta [4].

Reformulación de la Función Objetivo

Dado un conjunto de entrenamiento $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$, minimizar $\parallel \mathbf{w} \parallel$ (variando $\mathbf{w}$ y b) con la restricción de que $y_i(\mathbf{w}.x_i + b) \geq 1$ para todo $i = 1, 2, ..., n$.

Separadores óptimos

Luego de establecidos los vectores de soporte, puede ocurrir que haya puntos mal clasificados o demasiado cercanos al hiperplano separador (Figura 6.5).

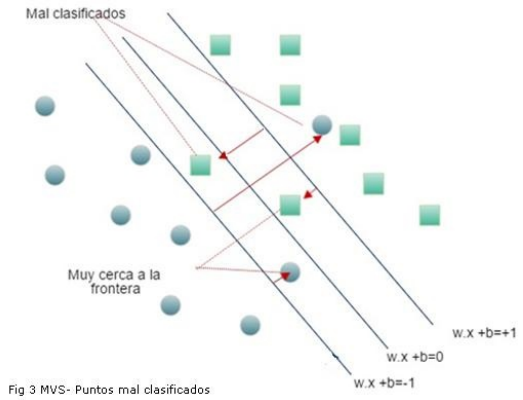

Figura 6.5: *SVM*, puntos mal clasificados o cercanos al borde (Basado en: *Mining of Massive Datasets Leskovec J, Rajaraman A, Ullman J - 2014*)

Es de interés penalizar los puntos mal clasificados proporcionalmente a la longitud del vector que apunta al punto. Los puntos que están cerca de la frontera pero bien clasificados, son penalizados en forma proporcional a la distancia del plano separador correspondiente a cada uno. Considerando los aspectos nombrados, se minimiza la función (6.6) [4].

$$f(w,b) = \frac{1}{2}\sum_{i=1}^{n} {w_i}^2 + C\sum_{i=1}^{n} max\{0, 1 - y_i(\sum_{j=1}^{d} w_j x_{ij} + b)\} \tag{6.6}$$

El primer término fomenta a obtener un $\| w \|$ pequeño como se formula en la función objetivo (se utiliza $\frac{\|w\|^2}{2}$ porque es lo mismo que minimizar $\| w \|$ y porque la derivada parcial $\frac{\partial}{\partial w_i}$ de cada sumando es w_i, quedando con igual dimensión a los efectos de posibilitar la suma con los valores del segundo término. Este segundo término corresponde a la penalización que se mencionó anteriormente y se expresa en la Ec. (6.7).

$$L(x_i, y_i) = C \sum_{i=1}^{n} max\{0, 1 - y_i(\sum_{j=1}^{d} w_j x_{ij} + b)\} \tag{6.7}$$

La constante C se denomina parámetro de regularización e intenta reflejar la importancia de una mala clasificación. Si no hay problemas con dichos puntos se toma un C pequeño, de lo contrario se considera un valor mayor [4]. El segundo término se denomina función bisagra [4]. Sea $z_i = y_i(\sum_{j=1}^{d} w_j x_{ij} + b)$. Si $z_i \geq 1$, $L(x_i, y_i)$ toma el valor 0, pero aumenta a medida que z_i decrece. Por lo tanto, dicha función no es derivable para $z_i = 0$, debiéndose operar de la siguiente manera [4]:

Si $y_i = +1$ (la muestra es positiva):

$$\frac{\partial L}{\partial w_i} = 0 \; si \; \sum_{j=1}^{d} w_j x_{ij} + b \geq 1 \; y \; si \; no \; es \; - x_{ij} \tag{6.8}$$

Si $y_i = -1$ (la muestra es negativa):

$$\frac{\partial L}{\partial w_i} = 0 \; si \; \sum_{j=1}^{d} w_j x_{ij} + b \leq -1 \; y \; si \; no \; es \; x_{ij} \tag{6.9}$$

Finalmente (6.8) y (6.9) se pueden consolidar en una sola ecuación (6.10), incluyendo el valor de y_i.

$$\frac{\partial L}{\partial w_i} = 0 \; si \; y_i(\sum_{j=1}^{d} w_j x_{ij} + b) \geq 1 \; y \; si \; no \; es \; - y_i x_{ij} \tag{6.10}$$

Procesamiento de grandes datasets

Para grandes *datasets* es preferible usar gradiente descendente estocástico que hace un procesamiento por pequeñas cantidades de puntos de la muestra.

También es de interés utilizar implementaciones en paralelo de la *SVM* mediante el empleo de *MapReduce*. Para ello se distribuye entre los *mappers*

las muestras y los valores de w y b que son usados por cada uno de ellos. Se itera todas las veces que sea necesario y luego el reduce promedia los w y b actualizados que pueden ser transferidos nuevamente a los *maps* volviéndose a repetir el ciclo.

6.1.3. Conclusiones

El conjunto de entrenamiento del perceptrón presenta únicamente dos etiquetas: positivo y negativo. De acuerdo a la etiqueta, los puntos correspondientes son separados por un hiperplano. La convergencia a este plano, se logra ajustando la razón de aprendizaje sobre la dirección donde se calcula el promedio de los puntos mal clasificados. Debido a las limitaciones que presenta este modelo para lograr la convergencia, se introdujo el algoritmo *Winnow*, el cual asegura la convergencia en caso de existir un hiperplano separador.

Determinar la mejor recta que separa las regiones es una limitación del perceptrón, lo cual es subsanado por las máquinas de vectores de soporte.

Las máquinas de vectores de soporte pueden manipular millones de características. El modelo es caracterizado por un vector normal al plano separador y expresado por una función objetivo. Una forma de optimizar la función objetivo es por medio del algoritmo de gradiente descendente si los *datasets* son pequeños. Si son de gran volumen, se puede utilizar gradiente estocástico o técnicas de procesamiento paralelo.

6.2. Reducción de la dimensión

A continuación, se describen algunas técnicas para la reducción de la dimensión en matrices: *PCA* (análisis de componentes principales), descomposición *UV*, *SVD* (descomposición por valores singulares) y *CUR*.

6.2.1. Análisis de componentes principales, PCA

La técnica *PCA* permite, dado un conjunto de datos identificados por sus coordenadas en un espacio de alta dimensión, realizar la búsqueda de las direcciones donde los datos se alinean mejor. Los datos se disponen en una matriz M y el método consiste en calcular los valores propios de MM^T o de M^TM [3]. El objetivo del algoritmo es que los datos en altas dimensiones, pueden

ser sustituidos por su proyección sobre los ejes más importantes. Estos ejes están asociados a los vectores con los mayores valores propios. De esa forma, los datos originales se representan en un espacio de menor dimensión [4].

6.2.2. Descomposición UV

Es una instancia del método de descomposición *SVD*. Dada la matriz M, se trata de obtener otras dos matrices U y V, tales que: $M = U \times V$ (con $dim(M) = m \times n$, $dim(U) = m \times d$ y $dim(V) = d \times n$)

Se trata de encontrar U y V, tales que su producto se aproxima a M en los valores que no se encuentran en blanco (es frecuente que estas matrices sean sparse, por ejemplo, la matriz de utilidad de los *SR*). En este caso, se puede considerar que los usuarios y los ítems pueden ser caracterizados en d dimensiones, pudiéndose emplear los valores obtenidos en $U \times V$, para predecir los valores sin entrada en M. La métrica más adecuada para medir la proximidad entre M y $U \times V$, es *RMSE* [4], tomando el cuadrado de la diferencia entre los elementos de M, que no están en blanco y el correspondiente en $U \times V$, para luego calcular el promedio, dividiendo entre la cantidad de entradas que no están en blanco en M y efectuando su raíz cuadrada. A los efectos prácticos, se puede considerar minimizar la suma de los cuadrados en lugar del *RMSE* [4].

6.2.3. Descomposición en valores singulares (SVD)

La descomposición *SVD* permite la representación exacta de cualquier matriz y también permite eliminar las partes menos importantes de ésta.

Sea A una matriz de dimensiones $m \times n$, que se quiere descomponer en el producto de 3 matrices dado por la Ec. (6.11).

$$A_{[m \times n]} = (U_{[m \times r]})(\sum_{[r \times r]})(V_{[n \times r]})^T \tag{6.11}$$

La matriz A representa a la matriz de entrada. La matriz U es singular izquierda de vectores (en el caso de los *SR*, m puede representar documentos o películas y r pueden representar conceptos). La matriz $\sum$ está formada con los valores singulares, es una matriz diagonal (tiene todos sus valores positivos, además de estar ordenados en forma decreciente) y representa la «fuerza» de cada concepto [4]. La matriz V es matriz singular derecha. El valor de la

dimensión r es pequeño. Se puede pensar que el valor r es el rango de la matriz A. La matriz U se puede considerar muy fina, es decir, pocas columnas y m filas, y a la matriz V^T como un pequeño número de filas, pero con n columnas (representando a los vectores singulares).

$$A_{[m\times n]} \approx U \sum V^T \tag{6.12}$$

El teorema de descomposición *SVD* se expresa en la Ec. (6.12) y establece que se puede descomponer una matriz real en el producto de las tres matrices, $U, \sum, V$, siendo dicha descomposición única y todas las entradas, valores reales. Las matrices U y V son ortogonales en sus columnas, las cuales son vectores unidad. A su vez, las columnas de U y de V son ortogonales entre sí; en otras palabras, esos vectores forman una base del espacio. Se verifica que $V^TV = I$ y $U^TU = I$. La importancia de SVD radica en la interpretación de las r columnas que forman la matriz U.

Un ejemplo de uso es considerar a la matriz A, la cual contiene filas que representan a los usuarios y columnas a las películas, la persona indica si le gustó o no la película, calificando con 1 estrella si no le gustó y con 5 si le gustó mucho. La idea del método es descomponer la matriz utilizando *SVD*, con la intención de descubrir conceptos. Algunos de esos conceptos no son interesantes y son los que se tratan de reducir a través del método. Una forma de hacerlo es considerando al conjunto de los p menores valores singulares como cero, lo que permite eliminar las correspondientes p filas de U y V (una detallada justificación se puede leer en [4]).

Costo computacional del SVD

SVD permite identificar correlaciones lineales en los datos (es capaz de identificar dimensiones a lo largo de las cuales, se distribuyen los datos)

Relación entre SVD y descomposición en valores propios

Dada una matriz, se realiza su *SVD* y, a partir de ésta, se hace su descomposición en valores propios. La relación entre valores propios y valores singulares, es que el cuadrado de éstos, son los valores propios de la matriz [4]. La descomposición en valores singulares, da la mejor aproximación del rango en términos de la norma de *Frobenius*. Esto otorga la mejor proyección de los datos, en un espacio de dimensiones menor (es decir, si se vuelve al espacio

original, la suma de los cuadrados de los errores, al reconstruir, es la menor). Sin embargo, *SVD* tiene el problema de ser difícil de interpretar (determinar a qué concepto corresponde cada vector singular). Otro problema, es que muchas veces la matriz A, tiene muchos ceros, pero al hacer el *SVD* las matrices U y V transpuestas se vuelven densas. Por este motivo, éste método no es muy utilizado [4].

Existen otras técnicas, que mantienen la condición de *sparse*, como por ejemplo, la descomposición *CUR*.

6.2.4. Descomposición CUR

El método *CUR* es otra técnica de reducción de dimensión. Dada la matriz A, se pretende descomponerla en el producto de 3 matrices: C, U y R. En este método, el error de reconstrucción es un poco más grande, pero el costo de computación, es menor. La matriz C contiene ciertas columnas de la matriz A. La matriz R contiene filas de A. El método presenta como ventaja principal que si A es *sparse*, las dos matrices C y R (que son las que se corresponderían con U y V en *SVD*) son *sparse* también. Sólo la matriz denominada U es densa pero no afecta porque es pequeña. Se demuestra que la convergencia de A se asegura a medida que aumenta el valor de la dimensión r. No obstante, en la práctica, se utilizan valores más chicos de r obteniéndose buenas descomposiciones [4].

Funcionamiento de CUR

Antes de ver el funcionamiento se efectúa el cálculo de la seudo inversa de una matriz diagonal. El siguiente resultado se necesita para justificar la obtención de la matriz U en la descomposición.

Sea Z una matriz diagonal, si sus valores son distintos de cero, se obtiene su inversa como la matriz diagonal con la inversa de los elementos de Z

Suponer que $W = XZY$, siendo XZY un *SVD*, (la matriz X es orto-normal en las columnas e Y lo es en las filas [4]) y, debido a la orto-normalidad, $X^{-1} = X^T$ e $Y^{-1} = Y^T$. Si X, Z *e* Y tienen inversas, se verifica:

$$W^{-1} = Y^{-1}Z^{-1}X^{-1} = Y^T Z^{-1} X^T$$

Al construir la inversa de Z, que es matriz diagonal, puede ocurrir que algún elemento de su diagonal sea 0. En este caso se define Z^+, también matriz diagonal, formada por el valor 0 si la entrada correspondiente en Z es 0, y el

valor de su inverso si no lo es ($Z.Z^+$ es una matriz diagonal con valores 0 y 1 en ella). De ahí que a Z^+ se le denomina seudo-inversa.

Sea A una matriz $m \times n$, se elige un número r de conceptos que será utilizado en la descomposición. La descomposición CUR de A comienza con una elección aleatoria de r columnas de dicha matriz, formándose una matriz C, de dimensiones $m \times r$. De la misma manera se eligen r filas de A, formándose otra matriz R, de dimensiones $r \times n$.

Para seleccionar una columna se calcula el cuadrado de la norma de *Frobenius* de dicha columna y se divide por el cuadrado de la misma norma de la matriz A. Este procedimiento establece una probabilidad que es utilizada para la selección para cada fila o columna. Es importante notar que la misma columna, puede ser seleccionada muchas veces. Con las filas, se procede de la misma forma que con las columnas. Luego de elegidas las componentes de C y R, es conveniente escalar los valores dividiendo por la raíz cuadrada del número esperado de duplicados de esa fila o columna de A. El algoritmo es similar para las filas.

Algoritmo para muestrear columnas

Entrada: $A \in R^{m \times n}$,tamaño de la muestra c
Salida: $C_d \in R^{m \times c}$
Para $x = 1 : n$ (columnas)
$P(x) = \sum_i A(i,x)^2 / \sum_{i,j} A(i,j)^2$
Para $i = 1 : c$ (muestreo de columnas)
Tomar $j \in 1 : n$ basado en la distribución de $P(x)$
Calcular $C_d(:,i) = \frac{A(:,j)}{\sqrt{cP(j)}}$

En caso de que haya duplicados se pueden descartar o, si no, multiplicar la columna o fila original, por la raíz cuadrada del número de duplicados. No obstante también se puede trabajar con los elementos duplicados, teniendo en cuenta que el rango de las matrices de la descomposición será menor.

Se debe construir una matriz U, cuya dimensión es $r \times r$.

Para ello se construye previamente, una matriz W cuya entrada w_{ij}, es la que se encuentra en A, donde la fila i es la que corresponde a la fila i de R, y la columna j es la correspondiente a la columna j en C.

Se calcula el SVD de W obteniéndose XZY^T. Sea Z^+ la seudo inversa de Z.

Entonces: $U = YZ^{+2}X^T$

CUR representa una aproximación a la matriz A.

6.3. Optimización

En esta sección, se describen los algoritmos de gradiente, gradiente estocástico y el método de coordenadas descendentes.

6.3.1. Gradiente descendente

El algoritmo de gradiente descendente, tal como se describió en la *Sección 4.3*, permite hallar el mínimo de una función derivable. El siguiente es uno de sus algoritmos, donde l es la función (de error o de pérdida) que se pretende minimizar en función de sus coeficientes w

Algoritmo de Gradiente descendente

Iniciar $w^{(1)} = 0, t = 1$
Hasta lograr la convergencia
Para $j = 0...D$
$\quad parcial[j] = \sum_{i=1}^{N} \frac{\partial l(w)}{\partial w_i}$ (la suma sobre todos los puntos)
$w_j^{t+1} \longleftarrow w_j^t + \eta partial[j]$
$t \longleftarrow t + 1$

6.3.2. Gradiente estocástico

El enfoque del gradiente estocástico (GE) considera entrenar a un solo punto dato diferente en cada iteración, no al conjunto de observaciones en su totalidad. Como base de la actualización en cada paso el gradiente estocástico utiliza la Ec. (6.13).

$$\frac{\partial l(w)}{\partial w_j} \approx \frac{\partial l_i(w)}{\partial w_j} \tag{6.13}$$

Algoritmo de Gradiente Estocástico

Iniciar $w^{(1)} = 0, t = 1$
Hasta lograr la convergencia
Para $i = 1...N$
    Para $j = 0...D$
        $parcial[j] = \frac{\partial l_i(w)}{\partial w_j}$ (cada t se toma un diferente punto i)
    $w_j^{t+1} \longleftarrow w_j^t + \eta partial[j]$
    $t \longleftarrow t + 1$

En teoría, el gradiente descendente es más lento en converger que el gradiente estocástico pero éste es más sensible a la elección de los parámetros tales como el *step size* y, por tal motivo, presenta problemas prácticos al momento de su implementación [20]. El gradiente estocástico converge rápidamente, alrededor del óptimo aunque no lo hace suavemente, sino que oscila alrededor de él. Esto es debido a que se evalúa, en cada iteración, en un solo punto (o en unos pocos) y la actualización obtenida no es tan acertada como la que se obtiene en el gradiente descendente (que evalúa todos los puntos). Por lo tanto el camino del gradiente estocástico es oscilante y cuando llega al óptimo oscila alrededor de él. Si los registros se encontraran ordenados, se podría introducir sesgo en el modelo. Para evitar eso, se sugiere una mezcla aleatoria de los datos antes de correr el algoritmo [20].

Para elegir el paso para *GE*, se procede de forma similar al método del gradiente. Sin embargo, la elección de η presenta dificultades debido a que el gradiente estocástico es más inestable. Si es demasiado pequeño, convergerá lentamente, y si es muy grande oscilará de manera impredecible. Una posible solución puede ser tratar con varios valores exponencialmente separados, con el objetivo de graficar el comportamiento para un η pequeño y otro grande. Otra opción podría ser, como en el algoritmo del gradiente, empezar con pasos grandes y posteriormente, decrecerlos.

6.3.3. Método de Coordenadas descendentes

El objetivo es minimizar la función $l(w)$ sobre todos los posibles w siendo l una función de muchas variables $l(w_0, \ldots, w_D)$. Generalmente, hallar un mínimo para todas las coordenadas, resulta complicado, entonces se puede hacer por coordenadas descendentes.

Algoritmo por coordenadas descendentes

Inicializar $\hat{w} = 0$ (o de alguna forma más conveniente)

Mientras no haya convergencia:

Tomar una coordenada j

$\hat{w_0} \longleftarrow min_w\ l(\hat{w_0}, \hat{w_1}, ..., \hat{w_{j-1}}, w, \hat{w_{j+1}}, ..., \hat{w_d})$

Siendo w el mínimo sobre la coordenada j y $\hat{w_0}, \hat{w_1}, ..., \hat{w_d}$ los valores obtenidos de los coeficientes en iteraciones previas.

Al ejecutarse el algoritmo, se comprueba que el camino de las coordenadas descendentes está alineado con los ejes (su interpretación gráfica se presenta en la Figura 6.6). Un criterio para la selección de las coordenadas para optimizar, es realizarlo aleatoriamente (en este caso se denomina coordenada descendente estocástica). Otra opción puede ser estableciendo un *round robin*, donde se establece un ciclo $0, 1, 2, ..., D$, repitiéndose la iteración. En este algoritmo, no existe *step-size*.

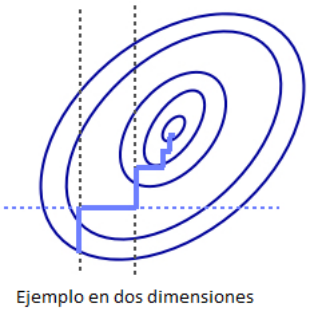

Figura 6.6: Representación de coordenadas descendentes (Origen: *Statistical Learning with Sparsity. The Lasso and Generalizations Hastie T., Tibshirani R., Wainwright M. - 2016*)

6.3.4. Normalización de las características

Consiste en transformar un atributo de los datos de entrenamiento en cierto rango de valores, para evitar medidas muy distintas entre las características. Dado un conjunto de puntos con sus correspondientes características, dispuestos en una matriz, se realiza la normalización por característica (columnas de la matriz). Sea $h_j(x_k)$ un elemento genérico de dicha matriz.

Sea $\bar{h_j}(x_k) = \frac{h_j(x_k)}{\sqrt{\sum_{i=1}^{N} h_j(x_i)^2}}$, se verifica que: $\sum_{i=1}^{N} \bar{h_j}(x_i)^2 = 1$

Se le denomina normalizador a la Ec. (6.14).

$$z_j = \sum_{i=1}^{N} h_j(x_i)^2 \tag{6.14}$$

Algoritmo de Coordenada descendente para mínimos cuadrados (RSS)

A partir de la Ec. (6.15) se efectúan las derivadas parciales para minimizar el *RSS*

$$RSS(w) = \sum_{i=1}^{N}(y_i - \sum_{j=0}^{D} w_j \bar{h_j}(x_k))^2 \tag{6.15}$$

($\bar{h}$ son las características normalizadas)

Sea w_j^* el vector w con todas sus componentes fijas excepto j, calculándose la derivada parcial respecto a w_j.

$$\begin{aligned}\frac{\partial RSS(w)}{\partial w_j} &= \\ &= \sum_{i=1}^{N} 2(y_i - \sum_{j=0}^{D} w_j \bar{h_j}(x_i))(-\bar{h_j}(x_i)) \\ &= -2\sum_{i=1}^{N} \bar{h_j}(x_i)(y_i - \sum_{j=0}^{D} w_j \bar{h_j}(x_i)) = \\ = -2\sum_{i=1}^{N} \bar{h_j}(x_i)(y_i - \sum_{k\neq j} w_k \bar{h_k}(x_i) - w_j \bar{h_j}(x_i))&(sacando\ w_j\ fuera\ de\ la\ suma)\end{aligned}$$

Finalmente se obtiene (6.16).

$$\frac{\partial RSS(w)}{\partial w_j} = -2\sum_{i=1}^{N} \bar{h_j}(x_i)(y_i - \sum_{k\neq j} w_k \bar{h_k}(x_i)) + 2w_j \sum_{i=1}^{N} \bar{h_j}(x_i)^2 \tag{6.16}$$

Si z_j representa a la suma de los cuadrados de las características normalizadas, se verifica: $z_j = \sum_{i=1}^{N} \bar{h_j}(x_i)^2 = 1$

Se considera $p_j = \sum_{i=1}^{N} \bar{h_j}(x_i)(y_i - \sum_{k\neq j} w_k \bar{h_k}(x_i))$, que es la suma sobre todas las características no iguales a j de w_k multiplicado por la característica

h_k, esto es, el valor predicho sin incluir la característica j.

El óptimo de RSS se calcula igualando la Ec. (6.16) a 0.

$\frac{\partial RSS(w)}{\partial w_j} = -2p_j + 2w_j = 0$ obteniéndose $\hat{w}_j = p_j$.

Es importante el hecho de que a $\hat{w}_j$ se le asigna el valor p_j. Se observa la correlación entre las características y los residuos de la predicción, dejando fuera a j del modelo. Si la correlación es alta, tienden a coincidir, lo cual significa que la característica j es importante para el modelo, por lo que resulta conveniente agregarla a éste, para mejorar la predicción. Si no son correlacionados, tienden a no coincidir en las observaciones y no se incluiría en el modelo. Este análisis conduce al algoritmo por coordenadas descendentes aplicado al *RSS*

Algoritmo por coordenadas descendentes para RSS

Inicializar $\hat{w} = 0$ (o de alguna forma más conveniente)
Mientras no haya convergencia:
    Para $j = 0, 1, 2, ..., D$
        Calcular $p_j = \sum_{i=1}^{N} \bar{h_j}(x_i)(y_i - \hat{y}_i(\hat{w}_j^*))$
    Setear $\hat{w}_j = p_j$
(siendo $\hat{y}_i(\hat{w}_j^*)$, la predicción sin la característica j y la expresión $(y_i - \hat{y}_i(\hat{w}_j^*))$ el residuo sin la característica j).

Resumen

El algoritmo del gradiente permite resolver problemas de optimización. Si la cantidad de características es grande y el *dataset* también lo es, el algoritmo no escala adecuadamente. Una pequeña modificación del algoritmo de gradiente, la cual conduce al gradiente estocástico, puede mejorar la performance, aunque la convergencia es menos estable.

Anexo 7

Regresión

En este anexo se desarrolla el cálculo del mínimo del *RSS* para el caso unidimensional y para el modelo de regresión lineal general. Se analizan algunas características de las regresiones *RIDGE* y *LASSO*. Por último se describen técnicas para la selección de atributos para los modelos de regresión.

Mínimo del RSS en un modelo lineal

A continuación se resuelve la Ec. (7.1) de la *Sección 4.4, Regresión.*

$$min_{w_0,w_1} \sum_{i=1}^{N} (y_i - [w_0 + w_1 x_i])^2 \tag{7.1}$$

Por las características de dicha función, la solución al problema es única y al aplicar el algoritmo del gradiente descendente, converge al mínimo.

Para el caso $g_i(w) = (y_i - [w_0 + w_1 x_i])^2$, al ser diferenciable, se obtiene:

$$\frac{\partial RSS(w)}{\partial w_0} = \sum_{i=1}^{N} \frac{\partial (y_i - [w_0 + w_1 x_i])^2}{\partial w_0} = -2 \sum_{i=1}^{N} (y_i - [w_0 + w_1 x_i])$$

$$\frac{\partial RSS(w)}{\partial w_1} = \sum_{i=1}^{N} \frac{\partial (y_i - [w_0 + w_1 x_i])^2}{\partial w_1} = -2 \sum_{i=1}^{N} (y_i - [w_0 + w_1 x_i]) x_i$$

Las dos ecuaciones anteriores definen un sistema de ecuaciones (7.2), el cual puede resolverse directa o iterativamente.

$$\nabla RSS(w_0, w_1) = \begin{cases} -2\sum_{i=1}^{N}[y_i - (w_0 + w_1 x_i)] \\ -2\sum_{i=1}^{N}(y_i - [w_0 + w_1 x_i])x_i \end{cases} \tag{7.2}$$

Resolución directa

$$\nabla RSS(w_0, w_1) = 0$$

La solución de 7.2 es:

$$w_0 = \frac{\sum_{i=1}^{N} y_i}{N} - w_1 \frac{\sum_{i=1}^{N} x_i}{N}$$

$$w_1 = \frac{\sum_{i=1}^{N} y_i x_i - \frac{\sum_{i=1}^{N} y_i \sum_{i=1}^{N} x_i}{N}}{\sum_{i=1}^{N} {x_i}^2 - \frac{\sum_{i=1}^{N} x_i \sum_{i=1}^{N} x_i}{N}}$$

Para hallar los coeficientes, alcanza con calcular las expresiones:

$\sum_{i=1}^{N} y_i$, $\sum_{i=1}^{N} x_i$, $\sum_{i=1}^{N} y_i x_i$ y $\sum_{i=1}^{N} x_i^2$

Resolución en forma iterativa (método del gradiente descendente)

A partir de la Ec. (7.2) se obtiene (7.3)

$$\nabla RSS(w_0, w_1) = \begin{cases} -2\sum_{i=1}^{N}[y_i - (w_0 + w_1 x_i)] = -2\sum_{i=1}^{N}[y_i - \hat{y}_i(w_0, w_1)] \\ -2\sum_{i=1}^{N}(y_i - [w_0 + w_1 x_i])x_i = -2\sum_{i=1}^{N}([y_i - \hat{y}_i(w_0, w_1)])x_i \end{cases} \tag{7.3}$$

Para efectuar las actualizaciones del método de gradiente descendente, se emplea la Ec. (7.4).

$$w^{t+1} = w^t - \eta \nabla RSS(w) \tag{7.4}$$

De (7.3) y (7.4) se obtiene la actualización para cada paso:

$$\begin{array}{lcll} {w_0}^{(t+1)} & \leftarrow & {w_0}^{(t)} & +2\eta \quad \sum_{i=1}^{N}[y_i - \hat{y}_i(w_0, w_1)] \\ {w_1}^{(t+1)} & \leftarrow & {w_1}^{(t)} & +2\eta \quad \sum_{i=1}^{N}([y_i - \hat{y}_i(w_0, w_1)])x_i \end{array}$$

Estas actualizaciones se iteran hasta llegar a una convergencia aceptable.

Mínimo del RSS para un modelo de regresión lineal general

Se trata ahora de encontrar una función de regresión pero con múltiples variables que representan características que van a encontrarse en el modelo.

Un ejemplo es la regresión polinomial cuyo modelo es:

$$y_i = w_0 + w_1 x_i + w_2 x_i^2 + \cdots + w_p x_i^p + \epsilon_i.$$

Las partes literales: $1, x_i, x_i^2, ..., x_i^p$ son las diferentes características de los datos y $w_1, w_2, ..., w_p$ son los parámetros que se deben calibrar. Si se disponen de n observaciones, se originan las siguientes ecuaciones:

$$\begin{aligned}
y_1 &= w_0 + w_1 x_{11} + w_2 x_{12} + ... + w_k x_{1k} + \epsilon_1 \\
y_2 &= w_0 + w_1 x_{21} + w_2 x_{22} + ... + w_k x_{2k} + \epsilon_2 \\
&\vdots \\
y_n &= w_0 + w_1 x_{n1} + w_2 x_{n2} + \cdots + w_k x_{nk} + \epsilon_n
\end{aligned}$$

El modelo lineal se puede expresar matricialmente como:

$$Y = Xw + \epsilon \tag{7.5}$$

Siendo: $Y = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix}$ $X = \begin{bmatrix} 1 & x_{11} & \cdots & x_{1k} \\ 1 & x_{21} & \cdots & x_{2k} \\ \vdots & \ddots & \vdots & \\ 1 & x_{n1} & \cdots & x_{nk} \end{bmatrix}$ $W = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_n \end{bmatrix}$ $\epsilon = \begin{bmatrix} \epsilon_0 \\ \epsilon_1 \\ \vdots \\ \epsilon_n \end{bmatrix}$

Cálculo del error en una dimensión

Analizando el *RSS* para el caso de regresión general, el residuo sigue siendo la diferencia entre la observación y el valor predicho en la curva $k-$dimensional. El análisis del residuo es útil para ajustar los valores de los parámetros a los efectos de que se aproxime a la función real. Convencionalmente la matriz X se designa H. Llamando $\hat{\mathbf{Y}}$ a los valores predichos por el modelo para un determinado vector de coeficientes $\mathbf{w}$, se verifica $\hat{\mathbf{Y}} = H\mathbf{w}$. Sumando el vector $\mathbf{Y}$ a ambos miembros se obtiene el vector de los residuos: $\mathbf{Y} - H\mathbf{w} = \mathbf{Y} - \hat{\mathbf{Y}}$.

A continuación se deriva la definición de *RSS* considerando el producto $(\mathbf{Y} - H\mathbf{w})^T(\mathbf{Y} - H\mathbf{w})$, para luego estudiar su gradiente. Este producto corresponde al producto interno de los vectores de los residuos que es la definición del *RSS*. Considerando que $\mathbf{w}^T h(x_i) = h^T(x_i)\mathbf{w}$, se obtiene la Ec. (7.6).

$$RSS(w) = \sum_{i=1}^{N}(y_i - \hat{y}_i(w))^2 = \\ = \sum_{i=1}^{N}(y_i - h^T(x_i)w)^2 \\ = (\mathbf{Y} - H\mathbf{w})^T(\mathbf{Y} - H\mathbf{w}) \tag{7.6}$$

En el caso unidimensional se verifica la Ec. (7.7) que se puede extender para más dimensiones a la Ec. (7.8).

$$\frac{d}{dw}(y - hw)(y - hw) = -2h(y - hw) \tag{7.7}$$

$$\nabla RSS(w) = \nabla[(y - Hw)^T(y - Hw)] = -2H^T(y - Hw) \tag{7.8}$$

Al igual que en el caso de regresión lineal simple (con una sola característica), se calcula el mínimo del RSS de las dos formas: directamente (obteniéndose (7.9)) y aplicando gradiente descendente sobre las coordenadas.

Cálculo directo para $\nabla RSS(w) = 0$

$\nabla RSS(w) = -2H^T(\mathbf{y} – H\hat{\mathbf{w}}) = 0$
$-2H^T y + 2H^T H\hat{\mathbf{w}}$
$H^T H\hat{\mathbf{w}} = H^T\mathbf{y}$
$(H^T)^{-1}H^T H\hat{\mathbf{w}} = (H^T H)^{-1}H^T\mathbf{y}$

$$\hat{\mathbf{w}} = (H^T H)^{-1}H^T\mathbf{y} \tag{7.9}$$

Este resultado es válido si $H^T H$ tiene inversa. La matriz H tiene dimensiones $N \times D$, por lo que $H^T H$ es $D \times D$. El valor N es el número de observaciones y D el número de características. Si el número N es muy grande, este procedimiento se torna computacionalmente costoso, debido a que el orden del cálculo de la inversa es $O(D^3)$, por lo que es necesario buscar un método más eficiente (por ejemplo el gradiente descendente). Además presenta el problema de que, para que sea invertible, debe cumplirse que $N > D$, aunque, en la mayoría de los casos se verifica que $N >> D$ (lo que facilita la obtención de filas linealmente independientes, pudiéndose calcular su inversa).

Cálculo utilizando el gradiente descendente para $\nabla RSS(w) = 0$

$w^{(t+1)} \leftarrow w^{(t)} - \eta \nabla RSS(w^{(t)})$
$w^{(t+1)} \leftarrow w^{(t)} + 2\eta H^T(y – Hw^{(t)})$ $\hat{y}(w^{(t)}) = Hw^{(t)}$

Actualización de cada característica

$RSS(w) = \sum_{i=1}^{N}(y_i – h(x_i)^T w)^2 =$
$= \sum_{i=1}^{N}(y_i – w_0 h_0(x_i) – w_1 h_1(x_i) - \ldots - w_D h_D(x_i))^2$

$$
\begin{aligned}
RSS(w) = & \sum_{i=1}^{N}(y_i – h(x_i)^T w)^2 \\
& \sum_{i=1}^{N}(y_i – w_0 h_0(x_i) – w_1 h_1(x_i) - ... - w_D h_D(x_i))^2 \\
& \frac{\partial RSS(w)}{\partial w_j} = \\
= & \sum_{i=1}^{N}(y_i – w_0 h_0(x_i) – w_1 h_1(x_i) - ... - w_D h_D(x_i))(-h_j(x_i)) = \\
= & -2\sum_{i=1}^{N} h_j(x_i)(y_i – h(x_i)^T w)
\end{aligned}
$$

La actualización en la característica *j-ésima* es:

$w_j^{(t+1)} \leftarrow w_j^{(t)} - \eta(-2\sum_{i=1}^{N} h_j(x_i)(y_i – h(x_i)^T w^{(t)}))$

$h(x_i)^T w^{(t)} = \hat{y}(w^{(t)})$

Algoritmo de gradiente descendente para minimizar RSS

Inicializar $w^{(1)} = 0$ (o de cualquier mejor forma) para $t = 1$
Mientras $\| \nabla RSS(w^{(t)}) \| > \epsilon$
    $for\ (j = 0...D)$
        $partial[j] = -2\sum_{i=1}^{N} h_j(x_i)(y_i – \hat{y}_i w^{(t)})$
        $\hat{w}_j^{(t+1)} \leftarrow \hat{w}_j^{(t)} - \eta.partial[j]$
$t \leftarrow t + 1$

Regresión no lineal

A menudo existen relaciones entre las variables X e Y que no son lineales, incluso luego de aplicarse transformaciones. Por ejemplo, la ecuación de la evolución del desarrollo del tamaño de los organismos es expresada en (7.10).

$$W = A(1–e^{Bt})^C \tag{7.10}$$

En la ecuación anterior W es una medida del tamaño del organismo, t la edad y A ,B y C son parámetros desconocidos a ser estimados a partir de los datos. No existe ninguna transformación para tal ecuación que conduzca a una ecuación lineal que permita el uso de RSS para estimar A, B y C. Este tipo de problemas se pueden resolver minimizando la función objetivo:
$\sum [W–A(1–e^{Bt})^C]^2$, en forma iterativa, a través de un algoritmo adecuado.

7.1. Regresión Ridge

En la *Sección 4.4.2, Regresión RIDGE y LASSO*, la ecuación ($Costo\ total = RSS(w) + \parallel w_2 \parallel^2$) proporciona una función de costo para *RIDGE*, utilizada para balancear el ajuste de la función a los datos y la medida de la complejidad del modelo.

Entrenamiento del modelo

El entrenamiento del modelo se realiza minimizando el *RSS* (*Sección 4.4.1*). La magnitud de los coeficientes con notación matricial se expresa en (7.11).

$$\parallel w \parallel_2^2 = w_0^2 + w_1^2 + w_2^2 + \ldots + w_D^2 = \mathbf{w}^T\mathbf{w} \tag{7.11}$$

En forma matricial el costo de la regresión RIDGE es el expresado en la Ec. (7.12).

$$RSS(\mathbf{w}) + \lambda \parallel \mathbf{w} \parallel_2^2 = (\mathbf{y} - H\mathbf{w})^T(\mathbf{y} - H\mathbf{w}) + \lambda\mathbf{w}^T\mathbf{w} \tag{7.12}$$

Cálculo del gradiente del costo de la regresión RIDGE

$$\begin{aligned}\nabla[RSS(w) + \lambda \parallel w \parallel_2^2] = \nabla[(y - Hw)^T(y - Hw) + \lambda w^T w] \\ \nabla[(y - Hw)^T(y - Hw)] + \lambda\nabla[w^T w]\end{aligned} \tag{7.13}$$

Se obtiene:

$\nabla\ cost(w) = -2H^T(y - Hw) + 2\lambda w$

Solución de forma directa

$\nabla\ cost(w) = -2H^T(y - Hw) + 2\lambda I w = 0$ (introduciendo la matriz identidad). Su solución, despejando directamente, es la Ec. (7.14).

$$\hat{w}^{ridge} = (H^T H + \lambda I)^{-1} H^T y \tag{7.14}$$

Solución ad hoc de $H^T H + \lambda I$

Si a la matriz $H^T H$ se le suma un múltiplo λ de la matriz identidad, se verifica que $H^T H + \lambda I$ es invertible si $\lambda > 0$, incluso si $N < D$ (*Hoerl and Kennard* propusieron esta solución *ad hoc*, con el parámetro λ en $[0, +\infty)$ que debe ser ajustado).

A la regresión RIDGE también se le denomina técnica de regularización. Esto es debido a que λI , hace que $H^T H + \lambda I$ sea más regular, ya que permite hallar la inversa, incluso en la situación en que N es menor que D. No obstante, resolver el sistema aún sigue siendo de orden cúbico.

Enfoque con gradiente descendente

La actualización de la característica j del costo se expresa en la Ec. (7.15).

$$w_j^{(t+1)} \leftarrow w_j^{(t)} - \eta[-2\sum_{i=1}^{N} h_j(x_i)(y_i - \hat{y}_i(w^{(t)})) + 2\lambda w_j^{(t)}] \tag{7.15}$$

El último término $2\lambda w_j^{(t)}$ corresponde a la *j-ésima* coordenada de $\mathbf{w}$. La Ec. (7.15) se puede escribir como la Ec. (7.16).

$$w_j^{(t+1)} \leftarrow w_j^{(t)} - \eta[-2\sum_{i=1}^{N} h_j(x_i)(y_i - \hat{y}_i(w^{(t)})) + 2\lambda w_j^{(t)}] \tag{7.16}$$

Considerando η y λ positivos, menores que uno, que verifiquen: $0 < 2\eta\lambda < 1$, cada coeficiente $w_j^{(t)}$ es multiplicado por un número positivo menor que uno, pudiendo reducirse su valor. Con el nuevo término, puede llegar a reducirse en alguna iteración el $w_j^{(t+1)}$ (ya que se toma el valor del coeficiente, se contrae y luego se le suma el RSS). En otras palabras, se hace la contracción y se verifica

cómo se ajusta el modelo. De hecho, es deseable la presencia de pequeños coeficientes para evitar el sobreajuste. En el caso del modelo de regresión lineal, no se hace la contracción, sólo se verificaba que se adaptara mejor el modelo en cada iteración, lo cual conducía a la aparición del sobreajuste.

Algoritmo para la Regresión *RIDGE*

Inicializar $w^{(1)} = 0$ (o de cualquier otro valor) para $t = 1$
Mientras $\| \nabla RSS(w^{(t)}) \| > \epsilon$
    for (j=0 ... D)
        $partial[j] = -2 \sum_{i=1}^{N} h_j(x_i)(y_i - \hat{y}_i w^{(t)})$
        $w_j^{(t+1)} \leftarrow (1 - 2\eta\lambda) w_j^{(t)} - \eta partial[j]$
$t \leftarrow t + 1$

Validación de la solución de RIDGE

La solución que se quiera obtener de *RIDGE*, depende de λ, el parámetro de penalización. El siguiente es un algoritmo para la validación de *RIDGE* utilizando validación cruzada *K-fold* y considerando K subconjuntos de datos: $k = 1, ..., K$.

Algoritmo para la validación de RIDGE

For $k = 1, ..., K$
    Estimar $\hat{w}_\lambda^{(k)}$ (en función de los otros subconjuntos restantes de entrenamiento)
    Calcular el error $error_k(\lambda)$ en el subconjunto de validación (k)
    Calcular el error promedio: $CV(\lambda) = 1/k \sum_{i=1}^{N} error_k(\lambda)$

Visualización de la regresión RIDGE

La visualización de *RIDGE* permite explicar el motivo de la obtención de soluciones *sparse*. Para el caso de dos dimensiones la Ec. (7.17) muestra el balance.

$$RSS(w) + \lambda \| w \|_2^2 = \sum_{i=1}^{N} (y_i - w_0 h_0(x_i) - w_1 h_1(x_i))^2 + \lambda(w_0^2 + w_1^2) \quad (7.17)$$

La Ec. (7.17) se puede escribir como lo indica la Ec. (7.18).

$$\sum_i y^2 + w_0^2 \sum_i h_0^2 + w_1^2 \sum_i h_1^2 + terminos\ cruzados = constante \tag{7.18}$$

Para el caso considerado, con dos características, h_0 y h_1, la Ec. (7.17) representa una elipse, obteniéndose, para cada residuo de las sumas de los cuadrados, una diferente. En todos los puntos del contorno de la elipse w_0 y w_1, tienen el mismo valor del residuo, el cual es el *RSS*. Para minimizar el valor del residuo, basta con considerar una elipse más interior, logrando, de esta forma, disminuirlo.

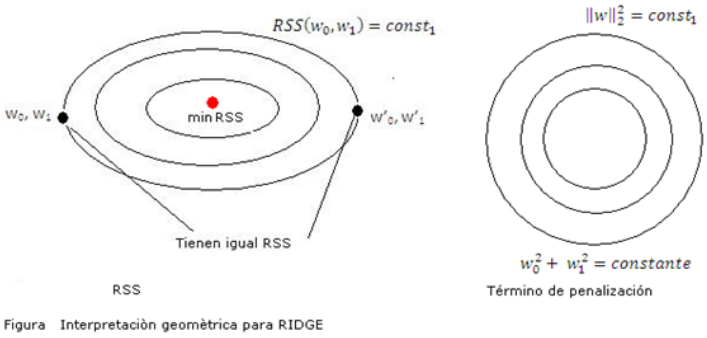

Figura 7.1: Visualización geométrica de RIDGE

En la primer imagen de la Figura 7.1, el punto rojo indicaría el mínimo entre todos los posibles valores de w_0 y w_1, que corresponde al $\hat{\mathbf{w}}$ para *LSS* (método de mínimos cuadrados). A su vez en la segunda imagen de la misma figura se puede visualizar los contornos generados por *RSS* y por la norma L_2 . Hasta ahora sólo se ha considerado la minimización de los *RSS*, a continuación se analiza la penalización L_2.

Considerando la penalización L_2, si la suma $w_0^2 + w_1^2$, segundo sumando de la Ec. (7.17), es igual a una constante ($w_0^2 + w_1^2 = constante$), la curva es una circunferencia. El centro de dicha circunferencia es el $(0, 0)$. Si se quiere minimizar la L_2, la constante será igual a cero (la cual sería el mínimo).

Se suman los dos contornos juntos, las elipses son centradas en la solución de mínimos cuadrados y la norma L_2 es centrada en cero. A su vez, λ es un valor que pondera la influencia de la penalización en la norma L_2.

Visualización del costo de RIDGE en 2 dimensiones

Al considerar la solución para un valor dado del parámetro λ, se obtiene un balance entre la suma de los residuos de los cuadrados y la magnitud de los coeficientes. El parámetro λ realiza dicho balance entre el *RSS* y la L_2.

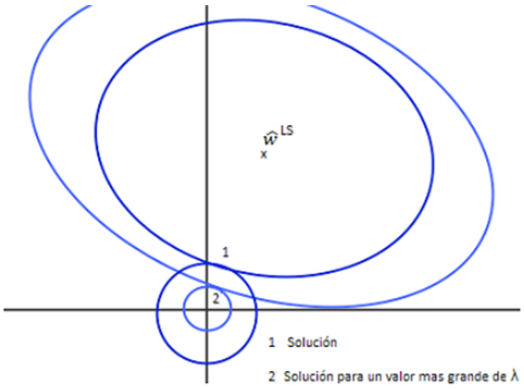

Figura 7.2: Representación gráfica del balance de la solución completa de RIDGE (Origen: *An Introduction to Statistical Learning with Applications in R, James G., Witten D., Hastie T., Tibshirani R. - 2017*)

El punto de tangencia de la circunferencia y la elipse es la solución para *RIDGE*. En la Figura 7.2, la solución de *RSS* está en el punto marcado con w^{LS}. Si $\lambda = 0$, w^{LS} coincide con el origen. Cada valor de λ conduce a soluciones diferentes. Para valores altos de la norma L_2 se obtienen mayores valores del *RSS*. Por otro lado, si se aumenta el valor del parámetro λ, se obtiene un valor menor de L_2. Esta situación es la que caracteriza al balance.

7.2. Elección de las características

En la *Sección 4.4.2* se nombraron diversos métodos para la elección de las características de un modelo.

Enumeración explícita

Es un método que efectúa una elección del subconjunto de características. Se comienza con un modelo de tamaño cero, sin características, es decir, presenta únicamente ruido. Luego, se busca entre los modelos que poseen una única característica y se escoge el modelo con menor *RSS*. A continuación se estudia cuál es el error de entrenamiento de ese modelo. Luego, operando

de igual forma, con dos características, se obtiene un nuevo modelo y así sucesivamente. Finalmente, se selecciona el modelo que tiene menor error de entrenamiento (Figura 7.3). No necesariamente las características que tienen el mejor modelo, tienen que incluir la característica del primer modelo. Es decir, las características no están necesariamente anidadas de modelo a modelo. Esto se repite hasta completar las D características. En este último caso (contemplando las D características) se obtiene un único modelo.

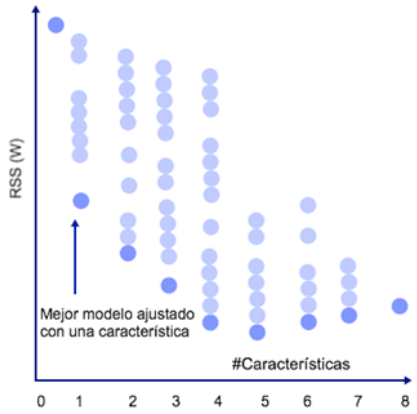

Figura 7.3: RSS en función del número de características

La selección no depende del modelo con el menor error de entrenamiento, ya que, a medida que aumenta la complejidad, el error disminuye. Para la validación se puede utilizar la validación cruzada *K-fold*.

Considerando la cantidad de características del modelo e indexando los distintos modelos, de acuerdo a las características presentes en él, se puede determinar la cantidad de modelos.

En el siguiente ejemplo se muestra que para un valor elevado de características, el algoritmo no es adecuado.

$y_i = \epsilon_i$ [000...000] Modelo con cero características (solo ruido).

$y_i = w_0 h_0(x_i) + \epsilon_i$ [100..000] Modelo con solo w_0 distinto de cero.

$y_i = w_1 h_1(x_i) + \epsilon_i$ [010...000] Modelo con w_1 distinto de cero.

$y_i = w_0 h_0(x_i) + w_1 h_1(x_i) + \epsilon_i$ [110...000] Modelo con w_0 y w_1 distintos de cero.

$y_i = w_0 h_0(x_i) + w_1 h_1(x_i) + ... + w_D h_D(x_i) + \epsilon_i$ [111...111] Modelo con todas las características presentes.

Para D características en total existen $2^{(D+1)}$ modelos. Por lo tanto, cuando

D toma un valor elevado, el método se torna computacionalmente intratable, por lo cual se propone un algoritmo voraz como alternativa.

Algoritmo voraz

El algoritmo voraz selecciona características, iniciando con un conjunto de ellas, las cuales son recorridas. Luego se selecciona la mejor característica y se continúa iterando. Se puede inicializar sin características o con un conjunto que incluye características previamente conocidas y que se sabe de antemano que están presentes. Si el modelo se inicia sin características, el siguiente modelo, que presenta una única característica, coincide con el modelo construido con el método anterior, pues es el que tiene el menor *RSS*. En los siguientes pasos, y tratándose de un algoritmo voraz, se mantienen las características seleccionadas en el paso anterior. Este punto es el que marca la diferencia del voraz con el método de enumeración explícita.

Ambos métodos inician y finalizan en iguales condiciones, pues cuando se tienen las D características, el modelo es único, cualquiera sea la técnica empleada para obtenerlo. El error en el método voraz, es decreciente. Aún si aumentara, se podría agregar la nueva característica con peso cero, evitando que siga creciendo. A los efectos de detener el procedimiento voraz, se utiliza un conjunto de validación, empleando el método de validación cruzada.

Complejidad del algoritmo voraz con paso hacia adelante

En el primer paso, se evalúan D modelos en el segundo $D-1$, en el tercero $D-2$ y así sucesivamente. El orden es $O(D^2)$ (menor que 2^D, del método de enumeración explícita).

7.3. Regresión LASSO

En las técnicas antes nombradas (enumeración explícita y voraz), se efectúan búsquedas sobre un conjunto discreto de posibles soluciones. Como alternativa, se puede comenzar con el modelo completo, con todos sus coeficientes y luego comprimir algunos de ellos hasta que sean exactamente cero, eliminándose las características correspondientes a ellos y manteniendo las restantes. A tales efectos, se puede considerar un umbral, a partir del cual, las características con valores menores, son consideradas nulas. Bajo estas perspectivas, se desarrolló el método de regresión *LASSO*.

Solución de la función de costo de LASSO

De la *Sección 4.4.2*, se considera la siguiente función de costo para *LASSO*.

$$Costo\ total = RSS(w) + \lambda \| \ w \ \|_1 \tag{7.19}$$

Se busca calcular el gradiente de la Ec. (7.19). Si se intenta resolver este problema, tomando el gradiente e igualándolo a cero, no existe la derivada de la función valor absoluto para $x = 0$, correspondiente a la norma L_1 . Por lo tanto, se modifica el algoritmo del gradiente descendente, derivándose coordenada a coordenada (*Anexo 6*).

Derivación de la fórmula para LASSO no normalizado en coordenadas descendentes

Se quiere calcular $\frac{\partial(RSS(w)+\lambda\|w\|_1)}{\partial w_j}$ sin las características normalizadas. Para ello se considera la ecuación la Ec. (7.20).

$$RSS(w) + \lambda \| \ w \ \|_1 = \sum_{i=1}^{N}(y_i - \sum_{j=0}^{D} w_j h_j(x_i))^2 + \lambda \sum_{j=0}^{D} |w_j| \tag{7.20}$$

Del *Anexo 6* se tiene:

$$\frac{\partial RSS(w)}{\partial w_j} = -2p_j + 2w_j Z_j \tag{7.21}$$

Queda pendiente determinar la derivada parcial del segundo término L_1 para lo cual se calcula $\lambda\frac{\partial|w_j|}{\partial w_j}$ que presenta el inconveniente que no es diferenciable. Para resolverlo se utiliza el concepto de subgradiente. Los gradientes pueden ser considerados como cotas inferiores de funciones convexas. En la Figura 7.4 se considera que si el punto *(a,g(a))* pertenece a la función g que es convexa, considerando a la recta tangente t a ella, representada por: $y–g(a) = \nabla(g(a))(x - a)$, se cumple que cualquier punto *(b,g(b))* verifica que $g(b) > g(a) + \nabla(g(a))(b - a)$

.

Si una función es derivable en un punto x, el gradiente es único. Los subgradientes son una cota inferior de una función y generalizan la idea para funciones no diferenciables.

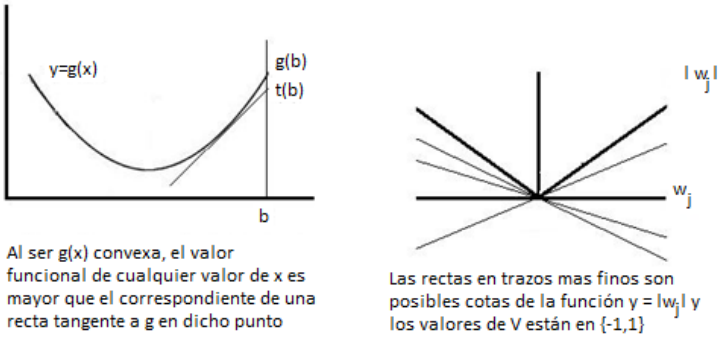

Figura 7.4: Subgradiente para $y = |w_j|$

Definición: V es subgradiente si verifica $g(b) > g(a) + V(b - a)$.

Notación: $V \in \sigma(g(x))$ significa que V es un subgradiente de $g(x)$.

El cálculo del subgradiente del término $\lambda|w_j|$ se expresa en la Ec. (7.22).

$$\lambda \frac{\partial |w_j|}{\partial w_j} = \begin{cases} -\lambda & \text{si } w_j < 0 \\ [-\lambda, \lambda] & \text{si } w_j = 0 \\ \lambda & \text{si } w_j > 0 \end{cases} \tag{7.22}$$

Para una función con valor absoluto todos los planos de la Figura 7.4 son los que serían cotas inferiores (las rectas cuyo coeficiente angular varía en el rango de *-1* a *1*). De las ecuaciones (7.20) y (7.21) se obtiene la Ec. (7.23) y la Ec. (7.24).

$$\partial_{w_j}[Costo\ LASSO] = -2p_j + 2w_j z_j + \begin{cases} -\lambda & \text{si } w_j < 0 \\ [-\lambda, \lambda] & \text{si } w_j = 0 \\ \lambda & \text{si } w_j > 0 \end{cases} \tag{7.23}$$

$$\partial_{w_j}[Costo\ LASSO] = \begin{cases} 2z_j w_j - 2p_j - \lambda & \text{si } w_j < 0 \\ [-2p_j - \lambda, -2p_j + \lambda] & \text{si } w_j = 0 \\ 2z_j w_j - 2p_j + \lambda & \text{si } w_j > 0 \end{cases} \tag{7.24}$$

Para hallar el mínimo de la Ec. (7.24), se calcula $\partial_{w_j} CostoLASSO = 0$.

Para $w_j < 0$, $2z_j \hat{w}_j - 2p_j - \lambda = 0$ se necesita que $p_j < -\lambda/2$ porque $\frac{p_j + \lambda/2}{z_j}$

Para $w_j = 0$, se necesita que $0 \in [-2p_j - \lambda, -2p_j + \lambda]$ con $-2p_j + \lambda \geq 0$ y $-2p_j - \lambda \leq 0$

Para $w_j > 0$, $2z_j w_j - 2p_j + \lambda = 0$ se necesita que $p_j > \lambda/2$ porque $\frac{p_j - \lambda/2}{z_j}$

Entonces, para el valor absoluto, V está en el intervalo de $[-1, +1]$ ($V \in \partial_{w_j}[Costo\ LASSO]$)

Hasta ahora, se tiene que el subgradiente de la función absoluta varía entre -1 y $+1$, por lo tanto, cuando w_j es cero, el valor varía entre $-\lambda$ y $+\lambda$. En la segunda imagen de la Figura 7.4, se aprecian algunas soluciones.

La solución óptima al problema planteado en (7.24) es lo expresado en la Ec. (7.25).

$$\hat{w}_j = \begin{cases} \frac{p_j + \lambda/2}{z_j} & \text{si } p_j < -\lambda/2 \\ 0 & \text{si } p_j \in [-\lambda/2, \lambda/2] \\ \frac{p_j - \lambda/2}{z_j} & \text{si } p_j > \lambda/2) \end{cases} \tag{7.25}$$

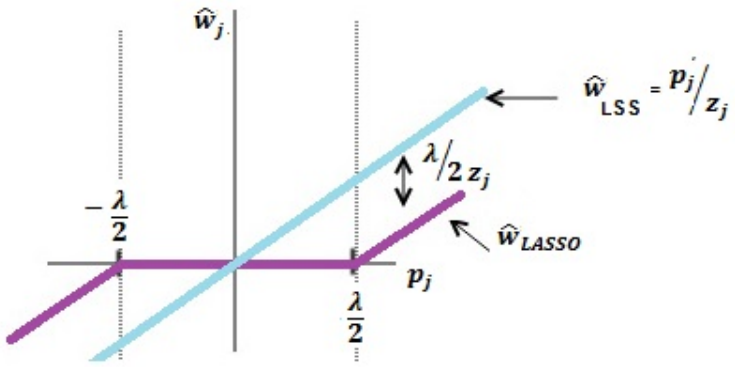

Figura Variación de los coeficientes para LASSO y para LSS

Figura 7.5: Variación de los coeficientes para LASSO y LSS (Origen: *An Introduction to Statistical Learning with Applications in R, James G., Witten D., Hastie T., Tibshirani R. - 2017*)

Notar que p_j es la correlación y que se puede calcular. Para el caso en que $w_j = 0$, existe una sola solución. Sin embargo, para que el subgradiente sea el óptimo, tiene que contener al cero, que también es un óptimo. De acuerdo al umbral utilizado para *LASSO*, se puede observar en la Figura 7.5 que, para el caso de características no normalizadas, se empieza con $\lambda = 0$ y se obtiene

la recta correspondiente a $\hat{w}_{LSS}$ (color celeste), cuyo coeficiente angular es igual a $\frac{p_j}{z_j}$. (El divisor z_j es el normalizador sobre todas las características al cuadrado).

En la misma figura, al observar las soluciones *LASSO*, de $-\frac{\lambda}{2}$ a $\frac{\lambda}{2}$, se tiene el umbral de los coeficientes exactamente igual a cero relativos a *LSS*. Fuera del rango, la diferencia entre *LSS* y *LASSO* son aquellos coeficientes que están contraídos por el valor $\frac{\lambda}{2z_j}$ (recordar que el valor de p_j es para características no normalizadas).

Algoritmo para LASSO con Características no Normalizadas

En el correspondiente algoritmo, como se mencionó, se observa la correlación entre las características y los residuos de la predicción, dejando fuera a j del modelo. Para *LASSO*, el valor de $\hat{w}_j$, depende del valor de λ y del valor del término de correlación p_j. Para determinados valores de p_j, se verifica que $\hat{w}_j = 0$, y, bajo estas condiciones, se originan las soluciones *sparse*.

Algoritmo de Coordenadas descendentes para LASSO con Características no Normalizadas

Inicializar $\hat{w} = 0$ (o de alguna forma más conveniente)
Mientras no haya convergencia:
Para $j = 0, 1, 2, ..., D$
Calcular $p_j = \sum_{i=1}^{N} \bar{h}_j(x_i)(y_i - \hat{y}_i \hat{w}_j^*)$
Setear $\hat{w}_j$ de acuerdo a la Ec. (7.26)

$$\hat{w}_j = \begin{cases} p_j + \frac{\lambda}{2} & \text{si } p_j < -\frac{\lambda}{2} \\ 0 & \text{si } p_j \text{ en } [-\frac{\lambda}{2}, \frac{\lambda}{2}] \\ p_j - \frac{\lambda}{2} & \text{si } p_j > \frac{\lambda}{2} \end{cases} \tag{7.26}$$

Esta técnica con la cual se actualiza *LASSO*, se denomina «umbral suave» (*soft thresholding*). La Figura 7.6 muestra la variación de $\hat{w}_j$ en función de la correlación p_j.

En *LSS* se asigna $\hat{w}_j = p_j$ en los casos en que λ es cero. En dicha figura, en color rojo, se visualiza cómo se elimina la característica, si p_j pertenece a $[-\frac{\lambda}{2}, \frac{\lambda}{2}]$. Fuera de ese rango se incluye la característica en el modelo pero se contrae el peso relativo a la solución de mínimos cuadrados, por un factor igual a $\frac{\lambda}{2}$. Por este motivo, se le denomina umbral suave, ya que se contrae uniformemente, llevando el valor cero en algunas regiones.

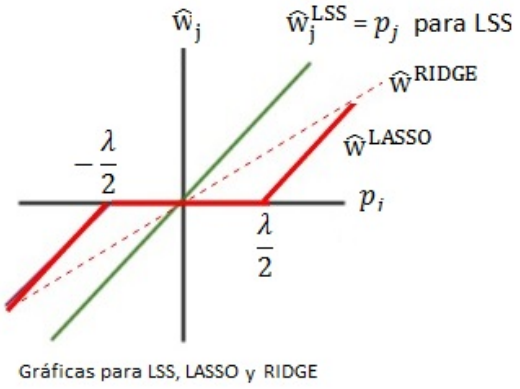

Figura 7.6: Umbral suave para LASSO y LSS (Origen: *An Introduction to Statistical Learning with Applications in R, James G., Witten D., Hastie T., Tibshirani R. - 2017*)

Algoritmo para LASSO con características normalizadas en coordenadas descendentes

Pre calcular: $Z_j = \sum_{i=1}^{N} h_j(x_i)^2$ (normalizador)
Inicializar $\hat{w} = 0$ (o de alguna forma más conveniente)
Mientras no haya convergencia:
Para $j = 0, 1, 2, ..., D$
Calcular $p_j = \sum_{i=1}^{N} h_j(x_i)(y_i - \hat{y}_i(\hat{w}_j^*))$
Setear w_j de acuerdo a la Ec. (7.27).

$$\hat{w}_j = \begin{cases} \frac{(p_j + \lambda/2)}{z_j} & \text{si } p_j < -\frac{\lambda}{2} \\ 0 & \text{si } p_j \text{en } [-\frac{\lambda}{2}, \frac{\lambda}{2}] \\ \frac{(p_j - \lambda/2)}{z_j} & \text{si } p_j > \frac{\lambda}{2} \end{cases} \tag{7.27}$$

Evaluación de la convergencia

Al aplicar el algoritmo del gradiente a funciones convexas, al acercarse al óptimo, es conveniente que se reduzca el tamaño de la condición de convergencia. Una opción para considerar que el algoritmo convergió, es medir dicha condición, en todo un ciclo, a lo largo de todas las características (por ejemplo, que sean menores a una tolerancia ϵ).

Elección de λ

El procedimiento para la elección de λ es similar al empleado en *RIDGE*. Con suficientes datos, se toma un conjunto de validación, utilizándolo para

formar distintos modelos, variando la complejidad de acuerdo a λ. Si no se tienen datos suficientes, se puede realizar validación cruzada. Normalmente, se elige λ más pequeño que el valor óptimo para seleccionar el modelo, debido a que para obtener precisión en la predicción, se obtienen mejores valores en un conjunto finito. En cambio, si se tomaran valores de λ mayores, una mayor cantidad de características serían anuladas, con la consecuencia de que se obtenga una menor precisión.

Visualización de la regresión LASSO

Para *LASSO* el contorno asociado al *Residual Sum of Squares* es el mismo que para *RIDGE*.

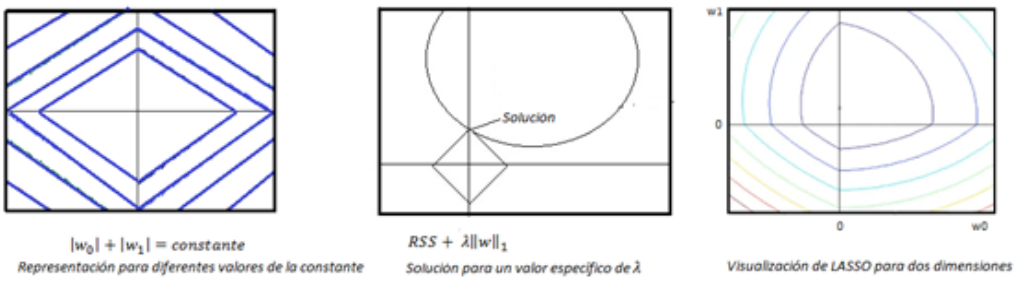

Figura 7.7: a) penalización b) una solución c) el costo de LASSO (Origen: *An Introduction to Statistical Learning with Applications in R, James G., Witten D., Hastie T., Tibshirani R. - 2017*)

$$RSS(w) + \lambda\| \ w \ \|_1 = \sum_{i=1}^{N}(y_i - w_0 h_0(x_i) - w_1 h_1(x_i))^2 + \lambda(|w_0| + |w_1|) \quad (7.28)$$

A través de la Ec. (7.28) se visualizará el balance. Considerando el segundo término de penalización, L_1 , se considera: $|w_0|+|w_1| = constante$. El contorno se puede visualizar en la primera imagen de la Figura 7.7, con forma de diamante. Cada uno de los puntos en la superficie tiene la misma norma L_1 y se considera que es igual a una constante. Los diamantes «más externos» tienen una norma L_1 igual a una constante mayor. Es conveniente que la magnitud sea lo más pequeña posible, como en *RIDGE*. Un valor posible es 0, que es el mínimo sobre w de la norma L_1.

En *LASSO* se debe sumar $RSS + \lambda \parallel w \parallel_1$, es decir elipses y un conjunto de «figuras de diamante» multiplicados por un λ y optimizarlas para obtener el mínimo.

Observando la solución para un determinado λ, debido a la forma de diamante que genera L_1, el encuentro entre la elipse y el diamante ocurre en $y = 0$ o $w_0 = 0$ y ahí se encuentra la razón por la cual aparecen las soluciones *sparse* en *LASSO*. Cuando λ es pequeño, la penalización es pequeña y ninguno de los coeficientes es 0. Esto tiene sentido, ya que se trata, exclusivamente, de la solución de mínimos cuadrados y además ningún coeficiente es 0. Al aumentar λ, las soluciones se tornan más *sparse* y aparecen coeficientes iguales a 0.

En la curva de ajuste, tanto en *LASSO* como en *RIDGE*, los coeficientes están contraídos, relativos a la solución de regresión. En estas condiciones, el algoritmo da suficiente regularización, lo cual conduce a un ajuste suave. Al aumentar λ, lleva a una suavidad aún más pronunciada del ajuste, sin embargo, si λ es grande, puede provocarse «exceso de suavidad».

Anexo 8

Técnicas de Data Mining y Big Data

En este anexo se estudian con mayor profundidad características de las técnicas de Data Mining mencionadas en el *Capítulo 5*.

8.1. LSH

LSH consiste en aplicar las funciones de *hash* múltiples veces a los ítems, de tal forma que los que son similares tengan mayor probabilidad de caer en la misma cubeta.

Luego de realizar un *minhash* de firmas se construye la matriz de firmas, tal cual se vio en la *Sección 5.1.5*. Se puede utilizar una técnica de división por bandas de r filas para seleccionar los elementos a los cuales aplicar cada función de dispersión.

Técnica de bandas

Esta técnica consiste en dividir a la matriz en un número entero b de bandas que contengan r filas cada una (Figura 8.1). Si M es la matriz de firmas, la notación $M(i, c)$ indica las entradas de i filas que corresponden a la columna c de M.

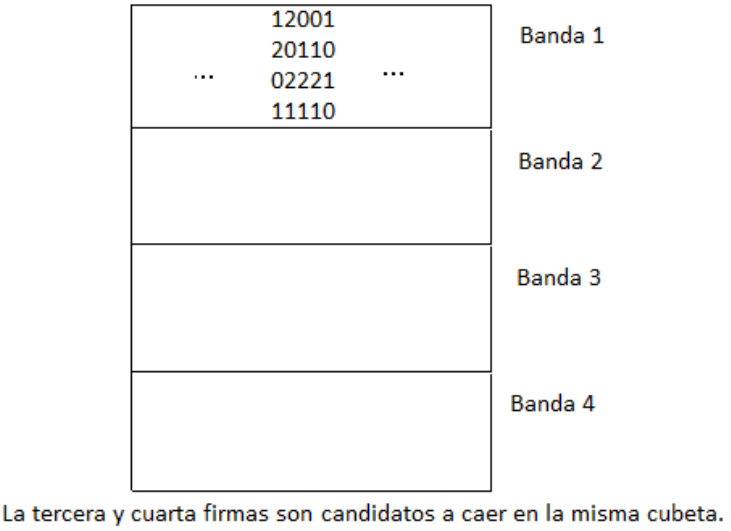

Figura 8.1: División de la matriz de firmas en 4 bandas de 4 filas cada una

Se define un umbral de similaridad t, $0 < t < 1$ y se buscan columnas en la matriz, para las cuales se considera que las firmas coinciden en una fracción $\geq$t, del total de filas. Es decir $M(i, c) = M(i, d)$ en, al menos, una fracción t, de los valores de i. Este proceso se repite varias veces para las columnas, considerando pares candidatos a ser iguales a aquellos que, al menos una vez, coinciden en la misma cubeta. Para cada banda, se aplica una función de *hash* a cada porción de columnas, en k cubetas (considerando al valor de k lo más grande posible). Se ajustan los parámetros b y r para considerar la mayor cantidad de «pares similares» y pocos «no similares». Notar que cuanto más similares sean dos columnas de la matriz de firmas, más posibilidades habrá, de que sean iguales en alguna banda. Sea s la similaridad de *Jaccard* entre dos documentos. Aplicando el resultado obtenido (la probabilidad de que el *minhash*, para cualquier permutación de las filas, produzca el mismo valor para dos conjuntos, es igual a la similaridad de *Jaccard*, para dichos sets), se deduce que la probabilidad que dos firmas de *minhash* coincidan, es también s.

La potencia s^r es la probabilidad de que todas las firmas de una banda coincidan. La resta $1 - s^r$, es la probabilidad de que las firmas no coincidan en, al menos, una fila de una banda. La expresión $(1 - s^r)^b$, es la probabilidad de que las firmas no coincidan en todas las filas, de ninguna de las bandas. La expresión $1 - (1 - s^r)^b$, es la probabilidad de que las firmas coincidan en todas las filas, de al menos una banda. Este caso convierte a los documentos en candidatos a ser similares.

Ejemplo 8.1

Suponer que se exige a C_1 y C_2 un 90 % de similaridad, y se debe emplear *LSH* con $b = 20$ y $r = 5$. La probabilidad de que sean iguales en una banda, es $(0.9)^5 = 0,59$. Esto se interpreta como el 60 % de probabilidad de que las firmas sean iguales en una banda (pasando a ser candidatos). La probabilidad de que C_1 *y* C_2 no sean similares en ninguna de las 20 bandas, es $(1-0,59)^{20} = 0,000000018$. Se interpreta como que 1 en 50 millones del 80 % de los posibles sets similares serán falsos negativos.

La gráfica de $1-(1-s^r)^b$, es denominada *step courve* o *s-courve*. Considerando el resultado de que la probabilidad de que coincidan dos firmas, es igual a la distancia de *Jaccard* entre documentos, el análisis de una firma de la matriz de firmas en función de s se puede apreciar en la Figura 8.2. Si $s < t$ la probabilidad de compartir una cubeta es cero y, si $s > t$, es uno.

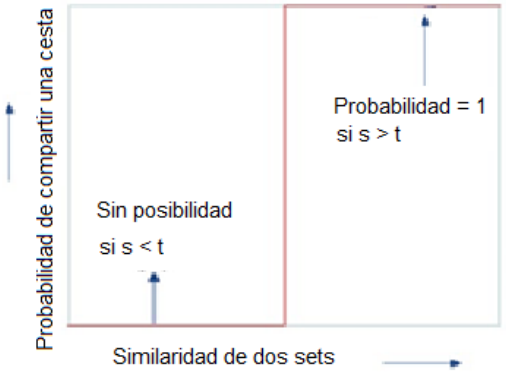

Figura 8.2: Probabilidad de coincidencia en una banda

Un valor posible para t es: $(1/b)^{1/r}$ [4].

Familias de funciones localmente sensibles

La técnica de *LSH* aplicada a funciones de *minhashing*, es un ejemplo de familia de funciones localmente sensibles. Se estudian otras familias de funciones que producen pares de candidatos, las cuales se aplican a cierto espacio, con una distancia definida en él. En el caso de *LSH*, el espacio son los conjuntos y la distancia es la de *Jaccard*.

Se consideran funciones que toman dos ítems y éstas deciden si ellos son candidatos a formar un par similar.

Definición: $(d_1.d_2, p_1, p_2) - sensible$

Sean $d_1 < d_2$ distancias sobre un determinado espacio. Una familia F de funciones es $(d_1.d_2, p_1, p_2) - sensible$, si, para todo f de F, se verifican las siguientes condiciones [4]:

- Debe tender a construir pares cercanos, que sean pares candidatos
- Deben ser estadísticamente independientes: es posible estimar la probabilidad de que dos o más funciones brinden una respuesta dada por el producto de eventos independientes.
- Deben ser eficientes en dos formas:

    1. Identificar pares candidatos en mucho menor tiempo, que el tiempo que lleva revisarlos a todos.
    2. Deben ser combinables para construir funciones que mejoren la detección de falsos positivos o falsos negativos, en menor tiempo.

Observaciones

Para dos pares de puntos que interesa que sean similares, la probabilidad p_1 se pretende que sea alta y su distancia (d_1) pequeña. En cambio para distancias grandes entre los dos puntos (d_2), se pretende que la probabilidad de ocurrencia, p_2, sea pequeña).

Dados dos ítems x e y, $f(x) = f(y)$ indica que x e y son candidatos, probablemente, a ser similares, luego de ser pasados por la función f. El caso de que $f(x) \neq f(y)$ es una indicación de que no son candidatos, hasta que otra función «indique» que sí lo son.

La familia de funciones *minhash* es un ejemplo en el cual cada firma es obtenida en alguna permutación posible, de las filas de la matriz característica.

Dados las distancias $d_1 < d_2$, una colección de funciones, se dice $(d_1.d_2, p_1, p_2) - sensible.$ si para cada f de F se verifica

Si $d(x, y) \leq d_1$ entonces $p(f(x) = f(y)) >= p_1$

Si $d(x, y) \geq d_2$ entonces $p(f(x) = f(y)) <= p_2$

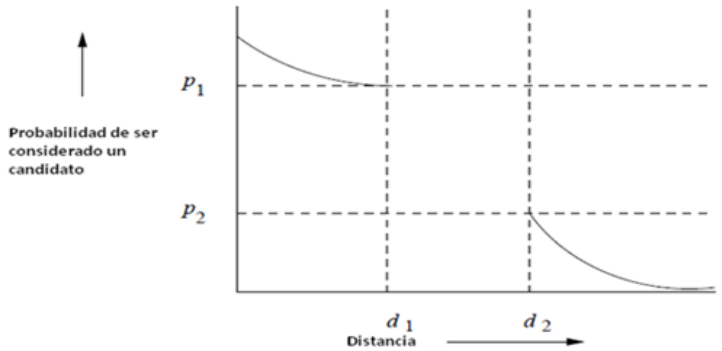

Figura 8.3: Comportamiento de una familia $(d_1.d_2, p_1, p_2)$*sensible* (Origen: *Mining of Massive Datasets Leskovec J, Rajaraman A, Ullman J - 2014*)

No expresa lo que sucede si la distancia d varía entre d_1 y d_2. No obstante, se pueden acercar d_1 y d_2 tanto como se quiera, con la penalización de que las probabilidades p_1 y p_2 también se acercarán. A su vez hay una cota superior para el valor de p_2 y una inferior para p_1 (Figura 8.3).

Ejemplo 8.2

La familia de funciones *minhash* es $(d_1, d_2, 1 - d_1, 1 - d_2)$–*sensible*, con $0 \leq d_1 < d_2 \leq 1$.

Demostración:

Si $d(x, y) \leq d_1$, siendo d la distancia de *Jaccard*, se verifica que:
$Sim(x, y) = 1 - d(xy) \geq 1 - d_1$.

Pero $Sim(x, y)$ es igual a la probabilidad de que la función de *minhash*, envíe x e y, a la misma cubeta, por lo tanto se verifica que $p(f(x) = f(y)) >= 1 - d_1$, que es la condición exigida para ser una familia $(d_1, d_2, 1 - d_1, 1 - d_2)$–*sensible*.

Un razonamiento similar es aplicado con d_2.

Ejemplo 8.3

Si $d_1 = 0.2$ y $d_2 = 0.6$, se puede aseverar que el valor del *minhash* es $(0.2, 0.6, 0.8, 0.4)$–*sensible*, cuya interpretación es la siguiente: si la distancia de *Jaccard* de x e y es como máximo 0.2 y se verifica que $sim(x, y) \geq 0.8$, pudiéndose asegurar que la probabilidad que la función de *minhash* envíe x e y a la misma cubeta, es $0, 8$.

Si $d(x, y) \geq 0.6$, $sim(x, y) \leq 0.4$, entonces la probabilidad de que x e y sean enviados a la misma cubeta no puede ser mayor que 0.4.

8.2. Algoritmos de recuperación de ítems

A continuación se presentan el algoritmo *A Priori*, algunos algoritmos para grandes *datasets* y los denominados algoritmos en pocos pasos.

Algoritmo A Priori

El algoritmo *A Priori* permite explorar los *ítemsets* frecuentes. Elimina los subconjuntos grandes partiendo de pequeños, utilizando la siguiente idea: si un conjunto es frecuente, los subconjuntos que lo forman también deberán serlo. Al evitar contarse todos los pares se optimiza el uso de la memoria. El objetivo es identificar ítems que comparten una regla (por ejemplo son comprados juntos por varios clientes). El algoritmo *A Priori* reduce el número de pares a ser contados y se basa en la regla: «Si el *ítemset* $\{a, b, c\}$ es frecuente, entonces $\{a, b\}$,$\{a, c\}$ y $\{c, b\}$ son frecuentes». El método establece que cualquier subconjunto de un *ítemset* frecuente es también frecuente, derivando así que si cualquier subconjunto de un *ítemset* S es no frecuente, entonces no es posible que S sea frecuente, lo que permite realizar podas de los subconjuntos.

Como se había mencionado anteriormente, este algoritmo consta de varios pasos.

Funcionamiento del Algoritmo A Priori

Primera pasada: Dado que los elementos de un *ítemset* son normalmente palabras, se realiza un mapeo en un arreglo de enteros (Arreglo 1). Luego, se considera otro vector (Arreglo 2) donde a cada celda le hace corresponder la cantidad de ocurrencias de dicho índice, de acuerdo al Arreglo 1.

Entre pasada y pasada: Entre los *singletons* (conjuntos con un elemento), se examina cuál es frecuente (dependiendo del umbral s, por ejemplo $s = 1\,\%$). Se considera ahora el otro vector, Arreglo 2, de longitud n. Si el ítem no es frecuente, se coloca un 0, si lo es, se coloca un número del 1 al m (siendo m la cantidad de ocurrencias del ítem). A dicho arreglo se le denomina tabla de ítems frecuentes.

Segunda pasada:

1. Para cada cesto determinar cuáles ítems son frecuentes buscando en la tabla de ítems frecuentes.
2. En un bucle doble, generar todos los pares de ítems frecuentes en el cesto.
3. Para cada par, agregar 1 en la cuenta de dichos pares en la estructura para el conteo utilizada.

Al final de la segunda pasada determinar en la estructura de conteos, cuáles pares son frecuentes.

A priori para todos los conjuntos de ítemsets frecuentes

Se repite la misma idea pero se evita contar la totalidad de los subconjuntos.

Sea s el umbral, para pasar del tamaño k al $k+1$, se debe considerar dos conjuntos: el de los candidatos, que son de tamaño $k+1$ (formados a partir de conjuntos $k-frecuentes$) y otro conjunto, con los que realmente son frecuentes, también de tamaño $k+1$.

Si se buscan subconjuntos de k+1 elementos, el procedimiento es el siguiente: Se construyen los candidatos mediante el conteo en cada cesto a partir de los $k-itemsets$ frecuentes y luego, a partir de estos se filtran los subconjuntos T con $k+1$ elementos. Si la cantidad de conjuntos contabilizados de alguno de los T supera al umbral s, entonces T es $k+1$ frecuente.

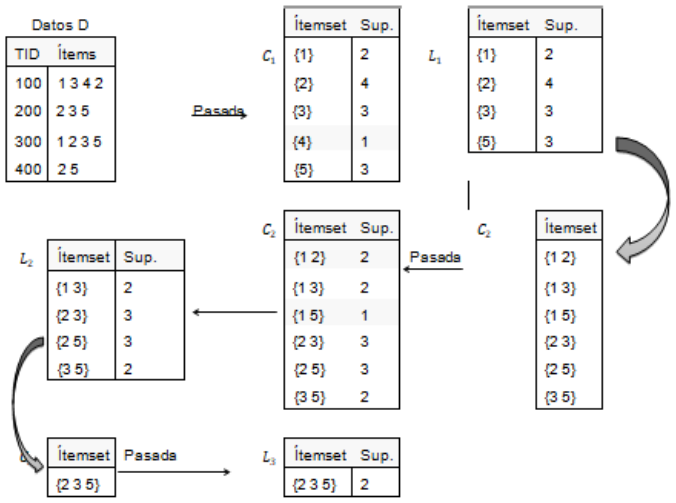

Figura 8.4: Ejemplo del algoritmo A Priori

En la Figura 8.4 se puede apreciar el funcionamiento del algoritmo *A Priori*.

Ejemplo 8.4

Considerar que $A = \{a, b, c, d, e, f, g, h, i, j\}$ es un conjunto de ítems y suponer que los conjuntos $2 - frecuentes$ son $\{a, b\}, \{b, c\}$, $\{c, d\}, \{d, e\}, \{e, f\} y \{f, g\}$. En principio, se pueden eliminar los ítems h, i y j, ya que no son frecuentes, pues no se encuentran en los conjuntos binarios anteriores. Los ítems a y g aparecen solamente una vez, por lo tanto no contribuyen en ninguna tripla frecuente. Entonces, las posibles triplas están formadas, si existen, por los ítems del conjunto $\{b, c, d, e, f\}$.

Transacciones con más de un ítem a la derecha

Cuando se desea utilizar campos con más de un ítem a la derecha, se utiliza la siguiente propiedad:

Dado $\{a, b\} \longrightarrow \{c, d\}$, se puede dividir en $\{a, b\} \longrightarrow \{c\}$ y $\{a, b\} \longrightarrow \{d\}$

Estas dos reglas tienen igual cobertura que la primera, pero la confianza es mayor que $\{c, d\}$ juntos. Esto brinda una regla para aumentar el miembro de la derecha, si bien es necesario controlar que la confianza esté dentro del valor adecuado (pues la confianza por separado es mayor, que si estuvieran juntos).

Algoritmos para grandes datasets

El algoritmo *A Priori* funciona correctamente, siempre que el paso con mayor demanda de memoria (habitualmente la fase de contabilizar los pares), sea suficiente. En dichas situaciones, el cuello de botella es la memoria disponible. Por ellos, algunos algoritmos tratan de reducir el número de pares candidatos.

A Priori funciona si dispone de suficiente memoria para realizar dicho conteo. En el caso de grandes *datasets*, existen otros algoritmos que aprovechan el hecho de que *A Priori* deja una gran cantidad de memoria no usada durante el conteo en el primer paso. El algoritmo *PCY* (desarrollado por Park, Chen y Yu [4]) aprovecha dicha memoria.

El algoritmo *Multihash* es una mejora de *PCY*: en lugar de dos diferentes tablas de *hash* en dos pasos sucesivos, se utilizan dos funciones de *hash* y dos tablas de *hash* en memoria principal en el primer paso.

Algoritmo PCY

El algoritmo *PCY* tiene dos pasos y entre ellos se ejecutan otras acciones.
Pase 1
Para cada cesto:{
    Para cada ítem
        Agregar 1 al conteo en la tabla de ítems frecuentes
        Para cada par de ítems:{
            Aplicar una función de *hash* al par que lo asigna a una cubeta
            Agregar 1 al conteo de esa cubeta
        }
}
Entre pases
Se mapean a un vector de *bits*, siendo *1*, si el número de la cubeta es mayor que el umbral s (correspondiente a un sub conjunto frecuente) y siendo cero, en caso contrario.

Los enteros de 4 *bytes* son reemplazados por *bits*, lo que ahorra espacio en el segundo pase. Luego, se deciden cuales ítems son frecuentes, listándose para el segundo pase.

Pase 2 Se cuentan los pares i, j que cumplen: a) i, j son ítems frecuentes, b) el par i, j se mapea con 1 en el vector de bits.

Consideraciones sobre la utilización de la memoria

Cada cubeta de una tabla de *hash* necesita de 2 a 4 *bytes*, por lo que las cubetas, utilizan de 1/4 a $\frac{1}{2}$, del número de *bytes* en memoria [4].

En el segundo pase, se debe llevar un control de las triplas (ítem, ítem, conteo) por lo que se necesita eliminar 2/3 de los pares candidatos, para que sea más eficiente que el algoritmo *A Priori*.

Ejemplo 8.5

El siguiente ejemplo explica claramente el funcionamiento del algoritmo (extractado de [50]).

Sean los siguientes conjunto de ítems:
$A = \{1, 2, 3, 4, 5\}$
$B_1 = \{1, 2, 3\}$

$B_2 = \{1, 4, 5\}$
$B_3 = \{1, 3\}$
$B_4 = \{2, 5\}$
$B_5 = \{1, 3, 4\}$
$B_6 = \{1, 2, 3, 5\}$
$B_7 = \{2, 3, 5\}$
$B_8 = \{2, 3\}$

Sea $s = 3$ y $L_1 = \{1, 2, 3, 5\}$, se obtiene el bitmap $\{1, 0, 1, 1, 1\}$.

El caso en que el número de cubetas es mayor o igual que s, se denomina cubeta frecuente. Si es menor que s, ninguno de sus pares puede ser frecuente y pueden ser eliminados como candidatos. Para el pase *2* sólo se cuentan pares que están en cubetas frecuentes.

Considerar la función $h(i, j) = (i + j)\ mod\ 5$
Los posibles pares son sus respectivos valores $h(i, j)$.
Los ítems frecuentes son: $1, 2, 3, 5$.

Pares posibles (i, j)	h(i, j)	Observaciones
~~(1,4)~~,(2,3)	0	(1,4) Eliminado porque 4 no es frecuente
(1,5),~~(2,4)~~	1	(2,4) Eliminado porque 4 no es frecuente
(2,5),~~(3,4)~~	2	(3,4) Eliminado porque 4 no es frecuente
(1,2),(3,5)	3	
(1,3),~~(4,5)~~	4	(4,5) Eliminado porque 4 no es frecuente

Figura 8.5: Ejemplo algoritmo PCY

Conteo de pares candidatos (Figura 8.6)

Conteo de pares candidatos	Par	Cantidad
	(2,3)	4
	(1,5)	2
	(2,5)	3
	(1,2)	2
	(3,5)	2
	(1,3)	4

Figura 8.6: Pares candidatos

Finalmente, los *ítemsets* frecuentes son:
$\{1\}, \{2\}, \{3\}, \{5\}, \{1, 3\}, \{2, 3\}, \{2, 5\}$

Notar que si se hubiera utilizado el algoritmo *A Priori*, hubiera tenido que recorrerse todos los pares candidatos.

Algoritmo multistage

El algoritmo *multistage* agrega un nuevo *hash* para reducir el número de pares candidatos. En el segundo paso, no se cuentan los pares candidatos como lo hace *PCY*, sino que crea un *hash* con una nueva función. Se recorren los cestos, sin contarlos, pues ya se dispone del conteo previamente realizado, aunque se debe mantener la información de qué ítems son frecuentes, puesto que se necesita en el segundo y tercer paso.

En el paso 2, se aplica una función de *hash* a los pares $\{i, j\}$ si y solo si i, j y el conjunto $\{i, j\}$ son frecuentes (información obtenida por PCY). Por lo tanto se espera que haya menos cubetas en el segundo *hash*. Luego se mapea a un nuevo *bitmap*.

Al final un par $\{i, j\}$ es candidato si y solo si:

- i, j son frecuentes.
- $\{i, j\}$ caen como cubeta frecuente en la tabla de *hash* 1.
- $\{i, j\}$ caen como cubeta frecuente en la tabla de *hash* 2.

Algoritmo multihash

Se crean las dos tablas de *hash* y dos funciones de *hash* en el primer paso, compartiendo la memoria. No habrá problemas mientras el promedio del conteo de las cubetas sea menor que el umbral de cobertura. En este caso, se opera con la mayor parte de las cubetas de la tabla de *hash* considerándose no frecuentes.

Algoritmos en pocos pasos

Son utilizados para resolver problemas del tipo de búsqueda de dos ítems comprados juntos (comparten el mismo cesto), sin necesidad de buscar toda la colección de *ítemsets* frecuentes.

Algoritmo simple aleatorio

Se utiliza una muestra de los cestos del *dataset*, debiéndose adaptar la cobertura s. Por ejemplo, si se toma el $0,1\,\%$, se considera $s/1000$. La elección

puede ser llevada a cabo, tomando los cestos que tienen una probabilidad p fija de ocurrencia (lo que implica recorrer todo el *dataset*). Si se tiene la certeza de que los cestos aparecen en forma aleatoria, no es necesario recorrer todo el *dataset*. Estos cuidados se deben a que, generalmente, los datos no se encuentran uniformemente distribuidos, ya que ítems que no son usuales en un determinado momento, pueden serlo después (si son canastos de compras de una casa de electrodomésticos, puede ocurrir que discos duros externos de 200 *Teras* no sean populares en estos días, pero probablemente lo sean en el futuro).

Luego de tener la muestra que debería ocupar una parte de la memoria, el resto de ella es empleada para correr los algoritmos, tales como *A Priori*, *PCY*, entre otros.

Puede ocurrir que se encuentren *ítemsets* frecuentes en el *dataset* y no en la muestra (falsos negativos) o frecuentes en la muestra y no en el *dataset* (falsos positivos). Con una muestra grande se soluciona este tipo de problemas, ya que un *ítemset* cuya cobertura es mucho mayor que el umbral, seguramente será considerado en una muestra aleatoria y lo mismo para los infrecuentes, cuya cobertura es mucho menor que el umbral en el *dataset*. Sin embargo, no ocurre lo mismo, si las coberturas están próximas al umbral.

Llevando a cabo un segundo paso, se pueden contar los frecuentes obtenidos en la muestra del *dataset* y si no lo son eliminarlos (serían los falsos positivos). Para eliminar parte de los falsos negativos, si existe disponibilidad de recursos de memoria, es posible bajar el umbral del soporte, de tal forma que aparezcan nuevos candidatos a ser frecuentes y se contabilizan en un nuevo paso en el *dataset*, eliminándose, en el caso de que no lo sean.

Algoritmo SON

Considerando la probabilidad p nombrada en el algoritmo simple aleatorio, se pasa todo el *dataset* dividido en p trozos de tamaño proporcional a $1/p$. Cada trozo es tratado como una muestra, ejecutándose el algoritmo simple aleatorio, utilizándose un umbral de ps. Se almacenan en disco todos los *ítemsets* frecuentes. Luego se considera la unión de estos conjuntos, los cuales constituyen la lista de candidatos a ser frecuentes. Es de resaltar que no existen falsos negativos, porque todos los itemsets frecuentes están entre los candidatos.

Para que un *ítemset* sea candidato a no ser frecuente en cualquier trozo,

su cobertura debe ser menor que ps y cada trozo es p del total, por lo tanto, el total de trozos, es $1/p$, obteniéndose que para ese *ítemset*, su cobertura es menor que $p(1/p)s$. Es decir, menor que s, por lo tanto, no es frecuente en el *ítemset*.

En la segunda pasada se cuentan los candidatos que tienen por lo menos un soporte s.

SON en MapReduce

El algoritmo *SON* puede implementarse con MapReduce [4]. La función *Map*, a partir del cesto asignado, busca los *ítemsets* frecuentes utilizando el algoritmo simple aleatorio, bajándose el umbral de cobertura de s a ps. Cada tarea de *Map* toma una fracción p del total. La salida es una clave valor del tipo: $(f, 1)$ donde$f$ es un *ítemset* frecuente de la muestra.

La función *Reduce* considera un conjunto de las salidas del *Map* y produce claves que son los *ítemsets*. Dicha salida constituirán a los candidatos.

La segunda función *Map* recibe toda la salida de la primera función *Reduce*, y un trozo del *dataset*, para posteriormente contabilizar cuántos de los *ítemsets* frecuentes candidatos, se encuentran en cada cesto. La salida es un conjunto de claves valor (C, v), siendo C, un candidato y v, la cobertura para dicho *ítemset*.

La segunda función Reduce toma los *ítemsets* que vienen en *(C,v)* y suma los valores v para cada uno, lo que produce, como resultado, la cobertura total de C. Si esa suma es mayor o igual a s, indica que C es frecuente y forman parte de la salida.

Algoritmo Toivonen

Asumiendo que se dispone de suficiente memoria, este algoritmo utiliza un paso sobre una pequeña muestra y un paso completo sobre los datos. No se crean falsos negativos ni positivos, pero puede haber alguna probabilidad de que algunos casos no brinden respuesta alguna (en este caso se debe repetir).

Se toma una pequeña muestra, se emplea el algoritmo simple aleatorio y se define el umbral menor a su valor proporcional. Por ejemplo: *0,9 ps*. Cuanto más bajo sea el umbral, más memoria se requiere para contener los *ítemsets* frecuentes, pero el algoritmo es más eficiente.

Luego de construido el conjunto desde la muestra, se construye la frontera negativa, definida por los *ítemsets* que no son frecuentes en la muestra, pero cuyos subconjuntos inmediatos (con un ítem menos), sí lo son.

Ejemplo 8.6

Sea:$\{A, B, C, D, E\}$ cuyos *ítemsets* frecuentes son:
$\{A\}$, $\{B\}$, $\{C\}$, $\{D\}$, $\{B, C\}$, $\{C, D\}$.

Tener en cuenta que ϕ (conjunto vacío) también es frecuente, mientras haya suficientes canastos como el umbral s. Dicho hecho hace que E se encuentre en la frontera negativa.

También $\{A, B\}$, $\{A, C\}$, $\{A, D\}$ y $\{B, D\}$ lo están (debido a que sus subconjuntos inmediatos son frecuentes).

De las triplas, ninguna está en la frontera negativa, ya que si a cualquiera de las posibles triplas se le quita un ítem en particular, no quedan todos los subconjuntos frecuentes. Por ejemplo: si al conjunto $\{A,\ B,\ C\}$ se le retira A, queda$\{B, C\}$, el cual no es frecuente.

Por lo tanto, la frontera negativa está compuesta por:
E, $\{A, B\}$, $\{A, C\}$, $\{A, D\}$ y $\{B, D\}$.

Para finalizar, se realiza una pasada por el *dataset* completo y se contabilizan los frecuentes de la muestra y los de la frontera negativa. Puede ocurrir que ningún elemento de la frontera negativa sea frecuente en el *dataset* y en ese caso, todos los frecuentes de la muestra, son los frecuentes del *dataset* (recordar que la frontera negativa se forma agregando un ítem a la vez, asumiendo que el conjunto previo es frecuente). También puede ocurrir que algún elemento de la frontera negativa sea frecuente en el *dataset*. En este caso, no se puede estar seguro de que no haya tuplas más grandes, que sean frecuentes en la muestra, o en la frontera negativa, que también sean frecuentes en el *dataset*. Estos son los casos en los que no existe respuesta y se debe repetir con otra muestra.

Toivonen nunca produce falsos positivos, ya que solo extrae los *ítemsets* que, luego del conteo, son frecuentes. A su vez, tampoco produce falsos negativos lo cual se demuestra a continuación.

Sea S frecuente en el *dataset*, no encontrándose en la muestra, ni en la frontera negativa. Todos los subconjuntos de S por monotonía son frecuentes. Sea T el subconjunto de menor tamaño, entre todos los subconjuntos que no son frecuentes en la muestra. Se demuestra que T pertenece a la frontera negativa porque:

1. No es frecuente en la muestra.
2. Cualquier subconjunto de T, es frecuente en la muestra (dado que si no lo fuera, habría un subconjunto en S que sería menor en tamaño que T, que, a su vez, no sería frecuente en la muestra, contradiciendo la elección de T, que es el menor subconjunto de S no frecuente en la muestra).

Entonces T es frecuente en el *dataset* y está en la frontera negativa.

8.3. Arboles de Decisión

En esta sección se desarrollan algunos conceptos sobre el entrenamiento, podas y evaluación de los AD y la construcción de un AD multiclase utilizando el concepto de ganancia de información.

Entrenamiento de los Arboles de Decisión

El objetivo es buscar árboles simples, para lo cual se pueden utilizar dos criterios: detenerse antes de que el árbol se torne complejo o llevar a cabo una poda (*prunning*). En este caso, el algoritmo avanza hasta terminar y, luego, efectúa la eliminación de las hojas que no son importantes.

Modalidades de parada anticipada

- Límite en la profundidad: Se determinará una profundidad máxima en la cual detenerse. El AD debería ser simple y el error verdadero bajo, sin embargo, a partir de cierta profundidad el error verdadero comienza a aumentar. Para ello, se utiliza el set de validación o validación cruzada. La crítica a este procedimiento es que las ramas son cortadas, en su totalidad, de igual manera. En ocasiones, no es suficiente utilizar validación cruzada u otro tipo de validación, debido a que se presentan diferentes soluciones dependiendo de la forma de organizar los datos, debiéndose correr el algoritmo varias veces y, aún así, posiblemente, sin obtener la respuesta que se desea. Además, es de esperar que algunas ramas crezcan más que otras.
- Error de clasificación estacionario: El algoritmo se detiene cuando, a partir de nuevas divisiones, los errores de clasificación no mejoran. Es decir,

los errores se mantienen constantes, para cualquier división que se pruebe. Para ello, se puede definir un umbral, a partir del cual, si el error no disminuye, no se vuelve a dividir. Dicho método es útil en la práctica, aunque presenta dificultades en la elección del umbral. Si el error de entrenamiento no decrece en forma suave, no sería fácil determinar la condición. Es necesario no detener la recursión si los errores de entrenamiento no decrecen lo suficiente. Esta condición tiene a favor que genera una heurística que no realiza divisiones innecesarias, aunque una de las desventajas que posee es que se pueden perder buenas divisiones que pudieran venir luego de una que no es útil.

- Tamaño mínimo de un nodo: Este criterio es útil si en el nodo quedan escasos puntos, ya que, al realizarse nuevas divisiones, se corre el riesgo de sobreajuste. Se previene el empleo de patrones en los datos, que no son suficientes para que sean representativos, en proporción al resto de las observaciones. Para ello, se define un mínimo que indica al algoritmo no continuar dividiendo el mismo nodo.

Podas para evitar el sobreajuste

Los AD son propensos al sobreajuste (debido a las anomalías y al ruido) [10] reflejándose en el error de entrenamiento que tiende a cero cuando los modelos se tornan más complejos. El error verdadero se reduce hasta cierto punto, y luego empieza a aumentar. A medida que se recorre el árbol en profundidad, el modelo se ajusta más a los datos de entrenamiento, tornándose propenso a que haya sobreajuste, ya que las fronteras de decisión se vuelven más complejas. Por esta razón es necesario realizar podas en el árbol [10].

A partir de un AD complejo, se busca simplificarlo en otro árbol con un menor error verdadero, aunque sobrevenga un mayor error de entrenamiento.

Sea la función que cuenta la cantidad de hojas de un árbol $L(T)$. Al considerar el número de hojas de un árbol, si $L(T)$ es muy grande posiblemente haya sobreajuste y, si es muy chico, posiblemente se trate de un modelo muy simple. Se desea hallar un balance entre los dos: que el árbol tenga el mejor ajuste y que la complejidad sea la menor posible (en este caso la complejidad aumenta con el número de hojas el árbol). El costo total del modelo es una medida del ajuste y de la complejidad.

Evaluación de los árboles de decisión

Se quiere balancear un buen ajuste a los datos y la complejidad del modelo.

El costo total de un modelo se puede definir como:

Costo total $=$ *medida del ajuste* $+$ *medida de la complejidad.*

Si el error es grande, existe un mal ajuste, y si la complejidad es alta el modelo tiende al sobreajuste. Para el caso de los AD, la ecuación del balance es:

$$C(T) = Error(T) + \lambda L(T) \tag{8.1}$$

$C(T)$: representa la función costo total.

$Error(T)$: significa al error en el modelo.

$L(T)$: indica el número de hojas.

Para balancear entre esas dos cantidades se introduce el parámetro λ, el cual verifica el ajuste en el set de validación, utilizando validación cruzada.

Balance entre el ajuste y la complejidad

Considerando la Ec. (8.1) se discutirá según el parámetro λ el tipo de entrenamiento para el AD.

Si $\lambda = 0$: caso de entrenamiento estándar del AD.

Si $\lambda = \infty$: penalización infinita, $\hat{y}$ es la clase mayoritaria. Se trataría de un árbol que no tiene ninguna decisión, es decir, es sólo la raíz.

Si λ está entre esos valores: se debería analizar el balance entre el ajuste y la complejidad y se puede establecer un criterio para la poda del árbol.

Algoritmo de poda de un árbol de decisión

Se comienza desde abajo del árbol y, utilizando la función costo, se compara el costo total, antes y después de hacer la poda.

Poda (T, N)

Calcular el costo de T usando $C(T) = Error(T) + \lambda L(T)$

Sea T_{chico} el árbol obtenido luego de podar el sub árbol por debajo de N

Calcular $C(T_{chico}) = Error(T_{chico}) + \lambda L(T_{chico})$

Si $C(T_{chico})$ es menor que $C(T)$, podar T_{chico}

Para prevenir el sobreajuste en un *AD*, es conveniente utilizar altos tempranos, considerando el balance entre error de clasificación y la complejidad del árbol.

Ejemplo de construcción de un AD considerando la ganancia de información

A los efectos de la construcción de un *AD* consideramos la Tabla 8.1 que representa un conjunto de entrenamiento con datos de estudiantes cuyas etiquetas de salida son: *C/R* (con riesgo), *S/R* (sin riesgo) y *D* (descartado de continuar estudiando).

Nota	Horas	Estrato	Riesgo
Excelente	Mucho	Alto	S/R
Excelente	Poco	Alto	S/R
Excelente	Nada	Medio	C/R
Bueno	Medio	Medio	C/R
Bueno	Nada	Medio	C/R
Bueno	Poco	Bajo	D
Malo	Mucho	Alto	S/R
Malo	Medio	Medio	S/R
Malo	Mucho	Bajo	S/R
Malo	Poco	Bajo	D
Excelente	Poco	Medio	D
Malo	Poco	Medio	D

Tabla 8.1: Datos para construir un árbol de decisión

Cálculo de la información de las clases

Con Riesgo: 3/12
Sin Riesgo: 5/12
Descartado: 4/12

$$I(C/R, S/R, D) =$$
$$= \quad -(3/12)log_2(3/12) - (5/12)log_2(5/12) - (4/12)log_2(4/12) = 1,552$$

El cálculo de la información de cada atributo se observa en la Tabla 8.2.

Atributo	Porcentaje de cada valor		Información de cada valor de los atributos
Nota	Excelente (E)	S/R 2/4	$I(E) = -2/4log_2 2/4 - 1/4log_2 1/4 - 1/4log_2 1/4 = 1,5$
		C/R 1/4	
		D 1/4	
	Bueno (B)	S/R 0/3	$I(B) = -0/3log_2 0/3 - 2/3log_2 2/3 - 1/3log_2 1/3 = 0,87$
		C/R 2/3	
		D 1/3	
	Malo (M)	S/R 3/5	$I(M) = -3/5log_2 3/5 - 2/5log_2 2/5 - 0/5log_2 0/5 = 0,97$
		C/R 2/5	
		D 0/5	
Horas	Mucho (Mu)	S/R 3/3	$I(E) = -2/4log_2 2/4 - 1/4log_2 1/4 - 1/4log_2 1/4 = 1,5$
		C/R 0/3	
		D 0/3	
	Medio (Me)	S/R 1/2	$I(B) = -0/3log_2 0/3 - 2/3log_2 2/3 - 1/3log_2 1/3 = 0,87$
		C/R 1/2	
		D 0/2	
	Poco (P)	S/R 1/5	$I(P) = -1/5log_2 1/5 - 4/5log_2 4/5 = 0,716$
		C/R 0/5	
		D 4/5	
	Nada (N) (M)	S/R 3/5	$I(N) = -2/2log_2 2/2 = 0$
		C/R 2/5	
		D 0/5	
Estrato Social	Alto (A)	S/R 3/3	$I(A) = 0$
		C/R 0/3	
		D 0/3	
	Medio (M)	S/R 1/6	$I(M) = -1/6log_2 1/6 - 3/6log_2 3/6 - 2/6log_2 2/6 = 1,45$
		C/R 3/6	
		D 2/6	
	Bajo (B)	S/R 1/3	$I(B) = -1/3log_2 1/3 - 0/3log_2 0/3 - 2/3log_2 2/3 = 0,87$
		C/R 0/3	
		D 2/3	

Tabla 8.2: Información de los atributos

A continuación, se determina cuál es el atributo con mayor ganancia de información (diferencia entre la entropía original y la de cada atributo) (Tabla 8.3).

Atributo	Ganancia de información $G(A) = s(D) - s_A(D)$
Nota	$1,552 + 1,5 * log_2 1,5 - 0,87 * log_2 0.87 - 0,97 * log_2 0.97 = 2.209$
Horas	$1,552 - 0,716 * log_2 0,716 = 1,552 - 0,34 = 1,212$
Estrato	$1,552 + 1,45 * log_2 1,45 - 0,87 * log_2 0.87 = 2,161$

Tabla 8.3: Ganancia de información de cada atributo

El atributo *Nota* es el que tiene mayor ganancia de información y por medio de éste se realiza la primer discriminación por cada uno de los valores

que puede tomar este atributo.

Los registros para Nota = *Excelente* se observan en la Tabla 8.4.

Nota	Horas	Estrato	Riesgo
Excelente	Mucho	Alto	S/R
	Poco	Alto	S/R
	Nada	Medio	C/R
	Poco	Medio	D

Tabla 8.4: Datos correspondientes a nota = *Excelente*

Se debe investigar el atributo, entre *horas* y *estrato*, que tiene mayor ganancia de información. Para ello es necesario calcular la ganancia de las clases para la Nota = *Excelente*, en base a las etiquetas de salida:

$$I(C/R, S/R, D) = -1/4 * log_2 1/4 - 2/4 * log_2 2/4 - 1/4 log_2 1/4 = 1.5$$

A continuación se presentan los cálculos para *horas* y *estrato* dejando fijo al atributo *hora*.

Atributo (Nota)	Porcentaje de cada valor (horas)		Información de cada valor del atributo
Excelente	Mucho (M)	S/R 1/1	$I(M) = -1log_2 1 = 0$ (es debido a que de acuerdo a la tabla anterior siempre es S/R)
		C/R 0/1	
		D 0/1	
	Poco (P)	S/R 1/2	$I(P) = -1/2log_2 1/2 - 1/2log_2 1/2 = 1$
		C/R 0/2	
		D 1/2	
	Nada (N)	S/R 0/1	$I(N) = 0$ (es debido a que de acuerdo a la tabla anterior siempre es C/R)
		C/R 1/1	
		D 0/1	
Atributo (Nota)	Porcentaje de cada valor (estrato)		Información de cada valor del atributo
Excelente	Alto (A)	S/R 2/2	$I(A) = -1log_2 1 = 0$ (es debido a que de acuerdo a la tabla anterior siempre es S/R)
		C/R 0/2	
		D 0/2	
	Medio (M)	S/R 0/2	$I(M) = -1/2log_2 1/2 - 1/2log_2 1/2 = 1$
		C/R 1/2	
		D 1/2	
	Bajo (B)	S/R No hay info	Puede utilizarse alguna técnica de datos faltantes para completarlo
		C/R No hay info	
		D No hay info	

Tabla 8.5: Información en *horas* y *estrato* para Nota = *Excelente*

Cálculo de la ganancia de información para Horas y Estrato

Atributo	Ganancia de información$G(A) = s(D) - s_A(D)$
Horas	$1,5 - 1 * log_2 1 = 1,5$
Estrato	$1,5 - 1 * log_2 1 = 1,5$

Tabla 8.6: *GI* para los atributos horas y estrato

El hecho de que son idénticas la ganancia de información para los dos atributos, se interpreta que se puede hacer la discriminación con cualquiera de ellos, indistintamente. En la Figura 8.7 se considera el atributo Horas como nodo para hacer el corte y se puede visualizar la construcción parcial del *AD* (las aristas sin etiquetas indican que aún no se ha completado la construcción).

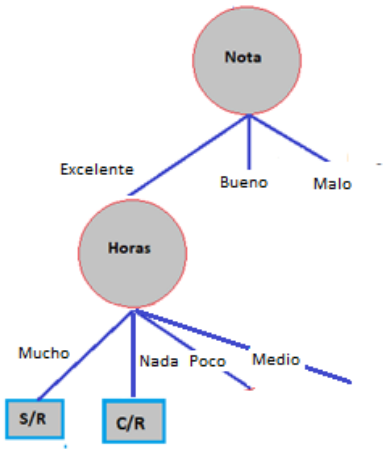

Figura 8.7: Proceso de construcción del AD

El procedimiento se repite para los valores restantes del atributo *Nota*, obteniéndose la Tabla 8.7

Nota	Horas	Estrato	Riesgo
Bueno	Medio	Medio	C/R
	Nada	Medio	C/R
	Poco	Bajo	D

Tabla 8.7: Datos correspondientes a nota = *Bueno*

Se debe investigar cual atributo (entre horas y estrato) tiene mayor ganancia de información. Previamente se calcula la ganancia de las clases, obteniéndose:

$$I(C/R, S/R, D) = -1/3log_2(1/3) - 2/3log_2(2/3) = 0.96$$

La información de cada valor de los atributos se ve en la Tabla 8.8.

Atributo (Nota)	Porcentaje de cada valor (horas)		Información de cada valor del atributo
Bueno	Medio (M)	S/R 1/1	$I(M) = -1log_2 1 = 0$
		C/R 0/1	
		D 0/1	
	Poco (P)	S/R 0	$I(P = -1log_2 1 = 0$
		C/R 0/1	
		D 1/1	
	Nada (N)	S/R 0	$I(N) = 0$ (es debido a que de acuerdo a la tabla anterior siempre es C/R)
		C/R 0/1	
		D 1/1	
Atributo (Nota)	Porcentaje de cada valor (estrato)		Información de cada valor del atributo
Bueno	Alto (A)	S/R 0	$I(A) = -1log_2 1 = 0$ (es debido a que de acuerdo a la tabla anterior siempre es S/R)
		C/R 0	
		D 0	
	Medio (M)	S/R 0	$I(M) = 1$
		C/R 2/2	
		D 0/2	
	Bajo (B)	S/R 0	$I(B) = 0$
		C/R 0	
		D 1/1	

Tabla 8.8: *GI* para los atributos horas y estrato para Nota = *Bueno*

El cálculo de la ganancia de información para *Horas* y *Estrato* aportan igual resultado. Por lo tanto, el corte se puede hacer indistintamente por cualquiera de ellos (Tabla 8.8). Luego se continúa con los demás atributos.

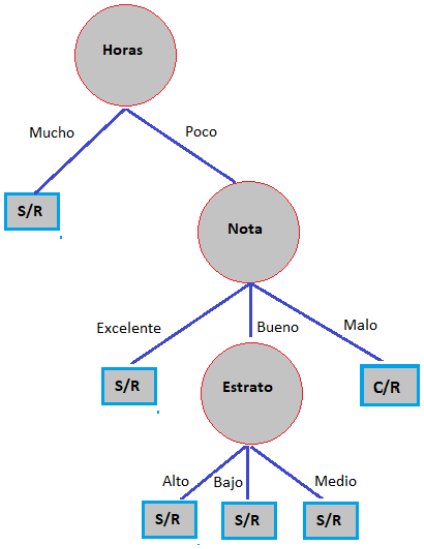

Figura 8.8: Proceso de construcción del AD (continuación)

8.4. Boosting

Se presenta el teorema de *AdaBoost* y una implementación del algoritmo correspondiente.

Teorema de AdaBoost

Bajo ciertas condiciones técnicas el error de entrenamiento tiende a cero, si el número de iteraciones o de modelos tiende a infinito. Sin embargo, las condiciones técnicas exigidas en algunos casos no es posible obtenerlas.

Algoritmo de AdaBoost

De acuerdo a la descripción del algoritmo dada en la *Sección 5.6, Combinación de clasificadores*, se debe definir: el cálculo de cada coeficiente $\hat{w}_t$, la forma de recalcular cada peso α_i y el tratamiento de los errores.

Cada dato de entrenamiento tiene un peso asociado α_i.

$$Peso\ total\ de\ los\ errores = \sum_{i=1}^{N} \alpha_i \mathbf{1}(\hat{y}_i \neq y_i) \tag{8.2}$$

$$Peso\ total\ de\ todos\ los\ puntos = \sum_{i=1}^{N} \alpha_i \tag{8.3}$$

$$Error\ ponderado = \frac{Peso\ total\ de\ los\ errores}{Peso\ total\ de\ todos\ los\ puntos} \tag{8.4}$$

Una forma de actualización de cada peso se puede llevar a cabo, empleando la siguiente fórmula.

$$\hat{w}_t = \frac{1}{2} \ln \frac{(1 - error\ ponderado(f_t))}{(error\ ponderado(f_t))} \tag{8.5}$$

El $\hat{w}_t$ es el nuevo cálculo de los coeficientes para un clasificador $f_t(x)$.

La fórmula que brinda $\hat{w}_t$, es derivada del teorema de *AdaBoost*.

Los α_i se obtienen en base a los errores de los clasificadores $f_t(x)$. La actualización de los α_i se realiza en función de los errores del clasificador. Si clasifica bien, el α_i decrece. En caso contrario, aumenta. Es decir, si $f_t(x_i)$ para una observación x_i, obtiene una etiqueta igual a la real, el algoritmo disminuye el α_i correspondiente y viceversa. Una posible forma de actualizar los pesos α_i se presenta a continuación

$$\alpha_i \longleftarrow \begin{cases} \alpha_i e^{-\hat{w}_t} & \text{si } f_t(x_i) = y_i (\text{correcto}) \\ \alpha_i e^{\hat{w}_t} & \text{si } f_t(x_i) \neq y_i (\text{incorrecto}) \end{cases} \tag{8.6}$$

De esta forma decrecen los α_i de los puntos cuya predicción fue correcta, aumentando al fallar la predicción. Notar que, en caso de ser $\hat{w}_t = 0$, la importancia no cambia, sea la predicción correcta o no (en este caso se puede considerar que el clasificador es aleatorio).

Por lo tanto, se incrementa la importancia del error, de manera tal que el siguiente modelo del clasificador prestará especial atención a ese punto determinado.

Pesos normalizados

Si se comete gran cantidad de errores durante la clasificación, y de forma continuada, el α_i puede crecer en demasía, en cambio, si no se cometen errores, puede haber un gran decrecimiento. Esto puede ocasionar inestabilidad

computacional, por lo que es necesario normalizar los valores al final de cada iteración hasta llegar a un máximo de uno. Para ello se puede utilizar la siguiente actualización.

$$\alpha_i \longleftarrow \frac{\alpha_i}{\sum_{i=1}^{N} \alpha_j}$$

Con las anteriores consideraciones se obtiene un algoritmo para entrenamiento de un ensamblador

Algoritmo AdaBoost

Asignar el mismo peso a todos los puntos: $\alpha_i = 1/N$

Para $t = 1...T$

Entrenar $f_i(x)$ con los datos con pesos α_i

Calcular coeficientes $\hat{w}_t$

$$\hat{w}_t = \frac{1}{2} \ln \frac{(1 - errorPonderado(f_t))}{(errorPonderado(f_t))}$$

Recalcular los pesos α_i

$$\alpha_i \longleftarrow \begin{cases} \alpha_i e^{-\hat{w}_t} & \text{si } f_t(x_i) = y_i (\text{correcto}) \\ \alpha_i e^{\hat{w}_t} & \text{si } f_t(x_i) \neq y_i (\text{incorrecto}) \end{cases}$$

Normalizar los pesos α_i

$$\alpha_i \longleftarrow \frac{\alpha_i}{\sum_{i=1}^{N} \alpha_j}$$

Predicción: $\hat{y} = sign(\sum_{t=1}^{T} \hat{w}_t f_t(x))$

Sobreajuste en Boosting

El error de *test* tiene relación con el error verdadero.

Al aumentar el número de iteraciones, el error de *test* logra estabilizarse y el error de entrenamiento debe continuar disminuyendo, de acuerdo al teorema. Eventualmente, *Boosting* puede sobre ajustarse, por lo que es necesario determinar cuidadosamente el máximo número de componentes de T. Es posible mantener esta situación bajo control, observando el error de *testing* (si luego de disminuir, comienza a subir, se está en presencia de sobreajuste).

Criterios para detener el algoritmo

Es posible emplear un parámetro, el cual ajuste determinado balance, tal como el que se lleva a cabo para regresión logística. Para *Boosting* puede ser el número de modelos en función de la calidad del ajuste (como el parámetro λ utilizado en la regularización de la Regresión de *RIDGE).*

8.5. Redes neuronales

A continuación se describe el proceso de aprendizaje de las *RNA* con propagación para adelante y *Back Propagation.*

Propagación para adelante

$a_i^{(j)}$: Es la notación que representa la activación de la unidad i en la capa j. (Valor calculado por la función g).

$\theta^{(j)}$: Es la matriz de pesos controladoras del mapeo de la capa j a la capa $j+1$

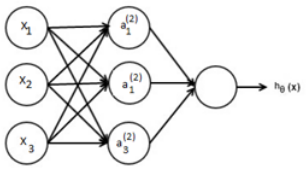

Figura 8.9: Red neuronal con una capa oculta de tres neuronas

Las ecuaciones de propagación, para la red neuronal de la Figura 8.9, son:

$$a_1^{(2)} = g(\theta^{(1)}x_0 + \theta_{11}^{(1)}x_1 + \theta_{12}^{(1)}x_2 + \theta_{13}^{(1)}x_3)$$
$$a_2^{(2)} = g(\theta_{20}^{(1)}x_0 + \theta_{21}^{(1)}x_1 + \theta_{22}^{(1)}x_2 + \theta_{23}^{(1)}x_3)$$
$$a_3^{(2)} = g(\theta_{30}^{(1)}x_0 + \theta_{31}^{(1)}x_1 + \theta_{32}^{(1)}x_2 + \theta_{33}^{(1)}x_3)$$
$$h_\theta(x) = a_1^{(3)} = g(\theta_{10}^{(2)}a_0^{(2)} + \theta_{11}^{(2)}a_1^{(2)} + \theta_{12}^{(2)}a_2^{(2)} + \theta_{13}^{(2)}a_3^{(2)})$$

Sea x el vector inicial de inputs ($a^{(1)} = x$) y $\theta^{(1)}$ los coeficientes antes de aplicar la función g de activación en la capa 1.

$a^{(2)}$: Son los valores que se obtienen en la capa 2 luego de aplicar g:

$$z = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad z^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix} \text{ y los valores de } z^{(2)} = \theta^{(1)}x \text{ y de } a^{(2)} = g(z^{(2)})$$

A partir de $a^{(2)}$, se obtiene: $z^{(3)} = \theta^{(2)}a^{(2)}$

Finalmente se obtiene la salida de la red: $h_\theta(x) = a^{(3)} = g(z^{(3)})$

Backpropagation (BP)

Consiste en aplicar dos fases al proceso de aprendizaje constituyéndose un ciclo de propagación-adaptación. Partiendo de una muestra de entrada, mediante un estímulo, éste avanza hacia la siguiente capa de las neuronas de la red y luego continúa la propagación a través de todas las capas superiores hasta generar una salida. Luego se compara el resultado obtenido en las neuronas de salida, con la salida que se desea obtener, y se calcula un valor del error para cada neurona de la última capa.

Estos errores se transmiten hacia atrás, partiendo de la capa de salida, hacia todas las neuronas de la capa intermedia que contribuyen directamente a la salida, recibiendo el porcentaje de error aproximado a la participación de la neurona intermedia en la salida original. Este proceso se repite, capa por capa, hasta que todas las neuronas de la red hayan recibido una señal de error que describe su contribución relativa al error total. Basándose en la señal de error percibida, se actualizan los pesos de conexión de cada neurona, para hacer que la red converja hacia un estado que permita clasificar correctamente todos los patrones de entrenamiento.

La importancia de este proceso consiste en que, a medida que se entrena la red, las neuronas de las capas intermedias se organizan a sí mismas, de tal modo que ellas aprenden a reconocer distintas características del espacio total de entrada. Cuando se les presente un patrón arbitrario de entrada que contenga ruido, o incompleto, las neuronas de la capa oculta de la red responden con una salida (si la nueva entrada contiene un patrón que las neuronas hayan aprendido a reconocer durante su entrenamiento).

Si se considera una función $costo(i)$ para medir la calidad en la predicción a la salida de la RNA, se puede obtener el error del valor de la activación que se tiene en la unidad j en una capa determinada y que pueden considerarse como las derivadas parciales. Por ello, uno de los principales objetivos del

algoritmo de *back propagation* es calcular el gradiente (derivadas parciales) de una determinada función objetivo. Se trata, de hecho, de una implementación de la regla de la cadena adaptada a redes neuronales artificiales [51] debido a que la *RNA* es una función de funciones y *back propagation* realiza la derivada sobre ella.

Una propiedad importante del *back propagation* es que algunos de los valores intermedios calculados pueden ser almacenados y guardados para calcular el gradiente en el pase para atrás.

8.6. Algoritmo BFR

De la *Sección 5.8, Clustering*, un algoritmo para gestionar grandes *datasets* es el *BFR*. Consiste en una variante de *k-medias*. Es apropiado para espacios euclídeos de alta dimensión y grandes cantidades de datos en disco. Se asume que los clústeres se ubican normalmente distribuidos alrededor de un centroide en un espacio euclídeo. Las desviaciones estándar en las diferentes dimensiones pueden variar. El método es eficiente para tratar los clústeres ya que el orden del algoritmo no es del orden de los datos sino en función de la cantidad de clústeres ($O(\#cluster)$). El hecho de que las agrupaciones estén normalmente distribuidas, indica que alrededor de la media se encuentran la mayor parte de las observaciones (campana de *Gauss*). Cada eje es independiente de los demás, siendo posible calcular la probabilidad de que un punto se ubique a cierta distancia del centroide. Por lo tanto, los clústeres son elipsoides cuyos ejes son paralelos a los ejes de coordenadas. Si la desviación estándar es mayor a lo largo de un eje, el clúster tiende a alargarse más en esa dirección.

Procedimiento

El algoritmo de *BFR* comienza seleccionando k puntos.

Desde la carga inicial se selecciona una cantidad k inicial de centroides mediante alguno de los métodos utilizados en *k-medias*.

El procedimiento considera una muestra de datos, leyendo del disco, que entre en memoria, en la cual, asimismo, se encuentran datos de los k clústeres, además de otra información. Los datos son leídos, procesados y luego usualmente descartados de la memoria. A continuación, se leen porciones de datos del disco, considerándose tres conjuntos de datos a tener en cuenta:

- Conjunto de descarte (DS): son los puntos que ya están sumarizados y no se necesitan.
- Conjunto comprimido (CS): consiste en información de los puntos que se han encontrado próximos entre sí pero no pertenecen a ningún clúster, se incluyen en mini clústeres que no tienen relación aún con los k clústeres.
- Puntos aislados (RS): son guardados en memoria a la espera de ser asignados a un clúster ya que no están en ninguno de los conjuntos anteriores. En la medida que surgen nuevos puntos, podrán ser integrantes de los mini clústeres de CS.

Para cada grupo, el conjunto de descarte (DS) y el comprimido (CS) son caracterizados, respectivamente, por:

- El número de puntos N
- El vector SUM, cuya componente i-ésima es la suma de las coordenadas de los puntos en su dimensión i-ésima.
- El vector $SUMSQ$ cuya i-ésima componente es la suma de los cuadrados de las coordenadas en la i-ésima dimensión.

El tamaño de cada clúster es representado por $2d + 1$, en el cual d es la dimensión del espacio (esto es debido a que SUM y $SUMSQ$ tienen, cada uno, dimensión d y hay que agregarle el escalar N que ocuparía una dimensión más.

El promedio en cada dimensión (centroide) se calcula mediante: SUM_i/N, siendo SUM_i la i-ésima componente de SUM

La varianza en la dimensión i del clúster de DS es:

$$Var_i = (SUMSQ_i/N) - (SUM_i/N)^2 \tag{8.7}$$

La desviación estándar es la raíz cuadrada de lo expresado en la Ec. (8.7).

Notar la importancia de la restricción de que los clústeres se encuentren alineados con los ejes. Si así no fuese, se requeriría una matriz de covarianza para tener la misma información del clúster, representada por una matriz $d \times d$ (la cual sería demasiado grande para la memoria).

Funcionamiento del algoritmo

Carga de puntos en memoria:

- Encontrar los puntos que son «suficientemente cercanos» a un centroide y añadir dichos puntos al clúster y al *DS*.
- Utilizar cualquier algoritmo de agrupamiento que corra en memoria principal y así poder agrupar los puntos restantes. Los clústeres de más de un punto que se obtienen se agregan al *CS* y los que son un solo punto van al *RS*.
- Conjunto *DS*: Ajusta las estadísticas de los clústeres con los nuevos puntos $(N, SUM, SUMSQ)$. Se debe considerar la fusión de conjuntos comprimidos en el *CS*.
- Si se trata de la última vuelta, se fusiona el total de los conjuntos de *CS* y todos los puntos de *RS* en el clúster más cercano o son descartados.

Determinar cuándo un punto es agregado a un clúster

Para agregar un punto a un clúster se sugiere el empleo de la distancia *Mahalanobis* (menor que un umbral).

La distancia de *Mahalanobis* cuantifica la probabilidad de que un punto esté cerca del centroide. Sea $C = (c_1, ..., c_d)$ un centroide con desviaciones estándar $\sigma_1, \sigma_2, ..., \sigma_d$.

Sea $P = (x_1, ..., x_d)$ un punto del cual se busca calcular la distancia a C.

La desviación estándar σ_i corresponde a los puntos en el clúster en la *i-ésima* dimensión.

Normalizando cada dimensión se obtiene: $y_i = \frac{x_i - c_i}{\sigma_i}$

La Ec. (8.8) indica la distancia de Mahalanobis de x a C [4].

$$d(x, C) = \sqrt{\sum_{i=1}^{d} (\frac{x_i - c_i}{\sigma_i})^2} \tag{8.8}$$

Considerando la dimensión i de un punto x, el cual se encuentra a una desviación estándar del centroide C, se verifica que $x_i - c_i = \sigma_i$. Por lo tanto la distancia normalizada es 1. Por último, $y_i = \frac{x_i - c_i}{\sigma_i}$ mide el número de desviaciones estándar alejada de la media en la dimensión i.

Si los grupos se distribuyen normalmente en d dimensiones, entonces después de la transformación, una desviación estándar es $\sqrt{d}$ (porque $\sqrt{\sum_{i=1}^{d} 1^2} =$

$\sqrt{d}$). Es decir, el 68 % de los puntos de la agrupación tendrá una distancia de *Mahalanobis* menor que $\sqrt{d}$.

Se acepta un punto en un clúster si su distancia de *Mahalanobis* es menor que algún umbral, por ejemplo, *2* desviaciones estándar.

Determinar cuándo combinar dos CS

Se combinan los dos mini clústeres si la varianza combinada se encuentra ubicada por debajo de cierto umbral. Luego es pasado al *DS*. El cálculo de la varianza de sub clústeres combinados, es fácilmente realizado empleando N, SUM y $SUMSQ$.

8.7. Clústeres Probabilísticos y GMM

Una distribución gausiana es apropiada para ciertas distribuciones de datos como, por ejemplo, para categorizar imágenes en función de sus dimensiones de color. En este caso las dimensiones son: intensidad de un color determinado, caracterizada por la media (la cual centra la distribución) y la varianza, que representa a la dispersión de la distribución.

Para una distribución gausiana en una dimensión, se considera una distribución normal, $N(x|\mu, \sigma)$ y, para más dimensiones, $N(x|\mu, \mathbf{\Sigma})$.

En la distribución anterior, μ, $\mathbf{\Sigma}$ son parámetros vectores y x es un vector con las coordenadas de la observación. En el caso bidimensional las curvas de nivel son elipses y la distribución gausiana se encuentra totalmente especificada con un vector de medias (que centran la distribución en cada dimensión), Ec. (8.9), y una matriz de covarianza (la cual especifica la dispersión y orientación de la distribución), Ec. (8.10). En la diagonal de dicha matriz, se tienen los términos de varianza de cada dimensión.

$$\mu = [\mu_A, \mu_B] \tag{8.9}$$

$$\mathbf{\Sigma} = \begin{bmatrix} \sigma_{A^2} & \sigma_{A,B} \\ \sigma_{B,A} & \sigma_{B^2} \end{bmatrix} \tag{8.10}$$

La Figura 8.10, considera dos observaciones A y B, La matriz $\mathbf{\Sigma}$ es simétrica y muestra en la diagonal principal a las varianzas de ambas observaciones. La segunda diagonal, denominada diagonal de covarianza, indica la estructura de

correlación entre A y B (orientación de las elipses de las curvas de nivel). Si la diagonal de covarianza es cero, significa que no hay correlación entre A y B.

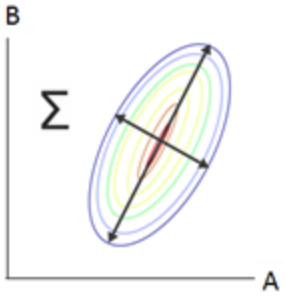

Figura 8.10: Matriz de covarianzas

Tomando iguales valores de las varianzas en la diagonal principal, la distribución tiene forma circular, ya que la dispersión a lo largo de cada dimensión es la misma. Si la matriz está completa, significa que hay correlación, lo cual implica que si el valor de A es alto, el de B también lo es. Es decir, indica si las variables aleatorias están correlacionadas positivamente. La misma idea se extiende para más dimensiones.

Asumiendo que se dispone de los parámetros fijos de los clústeres μ_k, σ_k, p_{ik}, se busca calcular la asignación suave con ellos. Dicha asignación se cuantifica empleando el vector de responsabilidad, Ec. (8.11). Para cada observación, representada por x_i, se asigna una responsabilidad r_i para cada clúster k. La responsabilidad r_{ik} es un valor dado por la Ec. (8.12) y se considera tomado por el cluster K para la observación i. Indica la probabilidad de la asignación de la observación x_i al cluster k dados los modelos del parámetro y la observación x_i

$$r_i = [r_{i1}, r_{i2}, ..., r_{ik}] \tag{8.11}$$

$$r_{ik} = p(z_i = k|\{\pi_j, \mu_j, \Sigma_j\}_{j=1}^{K}, x_i) \tag{8.12}$$

El conjunto $\{\pi_j, \mu_j, \Sigma_j\}_{j=1}^{K}$ representa a los parámetros del modelo para cada clúster. Dados dos clústeres, si una observación se encuentra cerca del centro de uno de ellos, éste toma mayor responsabilidad para la asignación del punto.

A continuación se considera la Ec.*(5.33)* de la *Sección 5.8.7, Clústeres*

probabilísticos, la cual indica cuál es la probabilidad de que una observación x_i sea seleccionada para un clúster $p(x_i|z(x_i) = k, \mu_k, \Sigma_k) = N(x_i|\mu_k, \Sigma_k)$.

La responsabilidad de la asignación se hace considerando la probabilidad a priori de que una observación provenga de un clúster, por la probabilidad de que x_i se encuentre en dicho clúster y, usando (6.22) se obtiene la Ec. (8.13).

$$r_{ik} = \pi_k N(x_i|\mu_k, \Sigma_k) \tag{8.13}$$

Para que sea una probabilidad válida, se necesita normalizar sobre todos las asignaciones a los clústeres obteniéndose la Ec. (8.14) que define la responsabilidad del cluster k al adjuntar la observación i.

$$r_i k = \frac{\pi_k N(x_i|\mu_k, \Sigma_k)}{\sum_{j=1}^{k} \pi_j N(x_i|\mu_j, \Sigma_j)} \tag{8.14}$$

Estimación de los parámetros de los clústeres

No se dispone, hasta este momento, de los parámetros de los clústeres. Partiendo de puntos ya asignados a clústeres, éstos no afectan la estimación de los parámetros de otros clústeres (es similar al método de *k-medias*, en el cual los puntos asignados actualizaban la media de cada clúster).

A continuación, se actualizan los centros y, además, la forma de los clústeres. Para la estimación de los parámetros se utiliza el estimador de máxima verosimilitud. Es decir, se busca, entre todos los parámetros, el que maximice la probabilidad de que los datos observados caigan en el modelo.

Cada clúster está especificado por la distribución gausiana con dos parámetros: la media y la covarianza. Se busca encontrar el estimador de máxima verosimilitud (EMV) de ellos [33], ecuaciones (8.15) y (8.16),

$$\hat{\mu}_k = \frac{1}{N_k} \sum_{ienk} x_i \tag{8.15}$$

$$\hat{\Sigma}_k = \frac{1}{N_k} \sum_{ienk} (x_i - \hat{\mu}_k)(x_i - \hat{\mu}_k)^T \tag{8.16}$$

La proporción del clúster $\hat{\pi}_k$, se determina mediante la Ec. (8.17).

$$\hat{\pi}_k = \frac{N_k}{N} \tag{8.17}$$

El valor N_k es el número de observaciones en el cluster K y N el número total de observaciones. Planteado de esta forma, el valor $\hat{\pi}_k$ no es únicamente específico para modelos de mezclas gausianas sino para todos, en general. Sin embargo, no se conocen las asignaciones aún, por lo que es posible considerar el caso en que se tienen las asignaciones suaves. Esto es más genérico ya que las observaciones no son de un solo clúster, sino de todos con su respectiva responsabilidad. Para todas las observaciones son asignados pesos, que definen al vector de responsabilidad. Para ello, a cada observación x_i, se le multiplica por r_{ik}. En este contexto N_k es la suma de las responsabilidades para el clúster k.

$$\hat{\mu}_k = \frac{1}{N_k^{soft}} \sum_{i=1}^{N} r_{ik} x_i \tag{8.18}$$

$$\hat{\Sigma}_k = \frac{1}{N_k^{soft}} \sum_{i=1}^{N} (x_i - \hat{\mu}_k)(x_i - \hat{\mu}_k)^T \tag{8.19}$$

$$N_k^{soft} = \sum_{i=1}^{N} r_{ik} \tag{8.20}$$

Finalmente $\hat{\pi}_k$ es el peso total del clúster k respecto al número de observaciones, Ec. (8.21).

$$\hat{\pi}_k = \frac{N_k^{soft}}{N} \tag{8.21}$$

Ejemplo 8.7

En la Tabla 8.9 se presenta un ejemplo de cuatro observaciones y su probabilidad de pertenecer a tres clústeres predefinidos, obteniéndose una matriz de responsabilidad, considerando los vectores de responsabilidad para cada observación.

Matriz de responsabilidad	Clúster A	Clúster B	Clúster C
Observación 1	0,02	0,8	0,18
Observación 2	0,05	0,9	0,05
Observación 3	0,12	0	0,88
Observación 4	0,1	0,85	0,05
N_k^{soft}	0,29	2,55	1,16
$\hat{\pi}_k = \frac{N_k^{soft}}{N}$	0,0725	0,6375	0,29

Tabla 8.9: Matriz de responsabilidad

Se consideran cuatro puntos y tres clústeres, con sus respectivos vectores de responsabilidad. La suma de cada fila suma uno, pues son las probabilidades de una observación repartida entre todos los clústeres. La suma de cada columna denominada conteo suave, indica la fracción de responsabilidad sobre todos los puntos de cada clúster. Finalmente, aplicando la Ec. (8.21), se observa que el clúster B tiene un mayor peso sobre las observaciones (proporción de datos en cada uno de los clústeres).

Algoritmo de Expectación-Maximización para mezcla de gausianas

Se pretende estimar directamente los parámetros de los clústeres.

Expectación-Maximización para mezcla de gausianas

Repetir hasta alcanzar la convergencia:

Se maximiza la verosimilitud dado $r_{ik}^{(1)}$

Se calcula $\{\hat{\pi}_k^{(1)}, \hat{\mu}_k^{(1)}, \hat{\Sigma}_k^{(1)}\}$

Se recalcula la responsabilidad $r_{ik}^{(2)}$

La estimación (*E-step*) de las responsabilidades del clúster dadas las estimaciones de los parámetros se calculan a partir de la (8.22).

$$r_i k = \frac{\hat{\pi}_k N(x_i|\hat{\mu}_k, \hat{\Sigma}_k)}{\sum_{j=1}^{k} \hat{\pi}_j N(x_i|\hat{\mu}_j, \hat{\Sigma}_j)} \tag{8.22}$$

La maximización (*M-step*) de la verosimilitud sobre los parámetros, dadas las responsabilidades actuales, se evalúa sobre: $\hat{\pi}_k, \hat{\mu}_k, \hat{\Sigma}_k|\hat{r}_{ik}, x_i$.

Esto se hace para cada punto dato. A continuación, se vuelve a estimar los parámetros de los clústeres, iniciándose una nueva iteración: dado $r_{ik}^{(1)}$ se re-estiman

$\hat{\pi}_k^{(1)}, \hat{\mu}_k^{(1)}, \hat{\Sigma}_k^{(1)}$ y se vuelve a recalcular las responsabilidades $\hat{r}_{ik}^{(2)}$ (ya que las elipses cambiaron).

Anexo 9

Técnicas empleadas en las aplicaciones

En este anexo se estudian en mayor detalle las técnicas empleadas en las aplicaciones mencionadas en el *Capítulo 6*.

9.1. Método de la Potencia y Page Rank

En el *Sección 6.1, Análisis de links*, se mencionó el método iterativo de las potencias para la resolución de un sistema de ecuaciones como alternativa a otros métodos. Se considera una matriz M de adyacencia, estocástica, de dimensiones $n \times n$, siendo n la cantidad de sitios *Web* existentes. La distribución de probabilidad para un usuario de Internet que visita a los sitios en forma aleatoria es un vector, donde en la componente u, se describe la probabilidad de que se encuentre en el sitio u. Sea la página i que tiene d_i links de salida. Los elementos de M se definen de la siguiente manera:

Si i apunta a j $m_{ji} = \frac{1}{d_i}$
sino
$m_{ji} = 0$

El vector de rankings r es un vector con una entrada por cada página. siendo r_i el valor de la página i. Se verifica que $\sum_i r_i = 1$. Las ecuaciones de flujo se pueden escribir $r = Mr$. Se puede considerar que el vector de rankings r, es un vector propio de M. De hecho, al ser la matriz M estocástica, el valor propio al cual se asocia, es el principal, y, a su vez, el máximo [41].

Método iterativo de las potencias

El método de las potencias es un procedimiento iterativo que calcula sucesivas aproximaciones a los vectores y valores propios de una matriz. El método se usa principalmente para calcular el vector propio con el mayor valor propio en matrices grandes.

Considerando una red con n nodos, el método se escribe:

Entrada: Matriz M
Inicialización: $r^{(0)} = [1/n, ..., 1/n]^T$
Iterar: $r^{(t+1)} = M.r^{(t)}$
Condición de parada: $|r^{(t+1)} - r^{(t)}|_1 < \epsilon$
(siendo $|x|_1 = \sum_{i=1}^{n} |x_i|$ la norma L_1 con $r_j^{(t+1)} = \sum_{i \to 1}(r_i^{(t)})/d_i$)

Ejemplo 9.1

Aplicación del método iterativo de la potencia para calcular el *Page Rank* de cada nodo en la siguiente matriz M.

$$M = \begin{bmatrix} 1/2 & 0 & 0 \\ 1/2 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

La evolución de r al aplicar el método, para una determinada condición de parada, es la siguiente:

$$\begin{array}{cccccccc} r_x & 1/3 & & 1/6 & & 1/12 & & 1/24 \\ r_y & 1/3 & \to & 1/2 & \to & 5/12 & \to & 13/24 \\ r_z & 1/3 & & 1/3 & & 1/2 & & 5/12 \end{array}$$

Las componentes del vector inicial son 1/3, corresponde a la inicialización ($n = 3$). Con una condición de parada adecuada, el último vector sería la última actualización del vector de rankings r.

Para que exista la convergencia anterior, debe cumplirse que el grafo sea fuertemente conexo (existe un camino para cualquier par de nodos) y no deben existir *dead ends*.

9.2. Características de los sistemas de recomendación

En esta sección se considera un marco de trabajo para analizar los distintos *SR* en base a varias dimensiones de análisis [36], *ratings*, *rankings* y su evaluación.

9.2.1. Enfoques para el análisis de los SR

Para el análisis de los *SR* se pueden utilizar diferentes abordajes [52]: de acuerdo al dominio, a su propósito, contexto de la recomendación, origen de las opiniones, nivel de personalización, privacidad e integridad, e interfaces, entre otros.

Dominio

El dominio responde a la interrogante de qué es lo que se está recomendando. Éste puede ser:

- Contenido orientado comercialmente (productos, pareo de preferencias, *bundles* que son paquetes de compra (si se compra a, se «cierra el trato» si, en realidad, se compra a, b y c juntos, favoreciéndose tanto al usuario como a la empresa).
- Secuencias (lista de ítems que se recomiendan donde importa el orden).
- Con un interés particular en una característica (libros, por ejemplo).

Propósito

Algunos de los propósitos de un *SR* pueden ser:

- Recomendaciones en sí mismas.
- Para consumo (compras, información, etc.)
- Educación
- Construcción de una comunidad sobre productos o contenidos

Un ejemplo de educación y construcción de una comunidad lo constituye *TripAdvisor* el cual no tiene como objetivo que se compre un viaje, sino educar al usuario a tomar la mejor decisión, sin guiarlo. Uno de los objetivos es

introducir a los usuarios a dicha comunidad, donde podrá brindar su opinión personal, o *feedback*, dentro de la comunidad de *TripAdvisor.*

Contexto de la recomendación

Interesa determinar qué hace el usuario al momento de la recomendación dado que el contexto puede afectarla en algunas situaciones.

Origen de las opiniones (de quien son)

Los sistemas de recomendación, desde un punto de vista amplio, se basan en la opinión de los usuarios. También se tiene en cuenta de dónde proviene la opinión: si proviene de un experto, «de todo el mundo» o de gente no idónea en la materia, entre otras posibilidades.

Nivel de personalización

El nivel de personalización establece cuatro categorías:

- Genérico o no personalizado: todos los usuarios reciben la misma recomendación.
- Demográfico: donde el objetivo es un grupo de personas que pertenecen a cierto rango, por ejemplo, el grupo etario.
- Efímeros: atendiendo la actividad actual del usuario (por ejemplo: un *SR* de música tiene en cuenta lo que el usuario escucha en este momento y recomienda en base a este hecho).
- Persistentes: se tiene en cuenta los intereses a largo plazo del usuario.

Privacidad e integridad

- Privacidad: Desde esta dimensión, un *SR* contempla los siguientes aspectos:

  1. Necesidad o no, por parte del *SR*, de saber quién es el usuario.
  2. Posibilidad de ser utilizado en forma anónima.
  3. Posibilidades de que el *SR* pueda negar algunas de las preferencias.
  4. Posibilidad de que el *SR* asegure que la información no se filtrará, de alguna manera, a través del sistema, hacia el exterior.

- Integridad: La integridad está relacionada con la honestidad del *SR* y contempla:

1. Sesgos en el *SR*. Estos sesgos se denominan reglas de negocio y, en algunos casos, son positivos y amistosos con el consumidor (por ejemplo, un *SR* nunca debiera recomendar algo que no tiene en stock). En otros casos, las reglas de negocio indican no vender cualquier ítem que no tenga una calificación alta de recomendación, o no recomendar algo si se cree que la persona comprará de todos modos, pues se perdería la recomendación. Las reglas de negocio son explicadas con mayor detalle en la próxima sección (Principales desafíos en los *SR*).
2. Vulnerabilidad a la manipulación externa: los usuarios, en general, están preocupados por la vulnerabilidad de un *SR* a la manipulación externa (por ejemplo inundar un nuevo producto con calificaciones positivas al lanzarse al mercado).
3. Transparencia de los *SR*: asociado a la reputación que pudieran tener.

Interfaces

- De entrada

1. Implícita: es decir, la frecuencia con que se visita una página o cada cuanto se cambian los *ratings*, información del tipo «el usuario compró algo».
2. Explícita: se le solicita al usuario indicar los ítems que le gusta o cuánto le gusta de acuerdo a una determinada escala.

- De salida

1. Predicciones: Efectuadas por el *SR* para el usuario
2. Recomendaciones: Considera el tipo de salida que el usuario está recibiendo (por ejemplo si se obtienen recomendaciones para un conjunto de artículos).
3. Filtrados: Contesta a la pregunta de cómo se efectúa el filtrado (por ejemplo, desde una lista de búsqueda, o de una selección en particular).

Estos aspectos conducen a una discusión sobre los diferentes niveles de la interfaz, que pueden tener en consideración aspectos como los siguientes:

- El sistema se presenta como un agente que ayuda al usuario (por ejemplo: presenta un cuadro de diálogo donde se puede volver, efectuar críticas y decir que lo recomendado no es lo que se quería).
- El *SR* está actuando más como un motor de búsqueda o como una herramienta.

9.2.2. Principales desafíos en los SR

El análisis de los *SR* presenta desafíos en varias direcciones.

Inconsistencia de los ratings

Un problema común en los algoritmos de filtrado colaborativo y en los basados en contenido es la inconsistencia de los *ratings*, porque los usuarios no siempre califican el mismo ítem de la misma forma, ya sea porque los gustos pueden variar de un usuario a otro, o porque el estado de humor es diferente, etc. En este sentido existen recomendadores que solicitan al usuario volver a calificar un ítem si entienden que una calificación recibida está fuera de lo que considera el *SR* como forma habitual de votación [36].

Altas dimensiones

Cuando las dimensiones son altas , sería conveniente tener en cuenta el costo computacional de estas técnicas. Una forma de tratar estos algoritmos, es mediante la reducción de la dimensión. Generalmente puede tratarse de matrices con millones de usuarios que votan cuánto les gusta un artículo, de los cuales hay miles. La reducción de la dimensión busca, de alguna manera, factorizar esa gran matriz, utilizando la técnica *SVD* (por ejemplo, transformándose en una matriz de intereses de categorías de cada usuario). Si los artículos fueran películas, esta información podría presentarse en la categoría que le pudiera llegar a interesar a los usuarios: policiales, románticas o algún otro tipo.

Reglas del negocio

Los *SR* tienen que adaptarse a las reglas del negocio (adaptarse al usuario y a la compañía que ofrece el producto) [36].

Algunas empresas consideran aspectos tales como el tiempo en que el usuario mira un producto, o la cantidad de veces que lo visita. Estas características

son utilizadas por los *SR* para mejorar sus predicciones. Incluso, para los usuarios anónimos, se utilizan *cookies* almacenadas en sus computadoras. Las reglas de negocio previenen que el *SR* realice recomendaciones obvias y, a su vez, maximizan las ventas, sin perder la credibilidad. Por ejemplo, recomendar ítems que ya todo el mundo tiene o que siempre compran, bajo una óptica estadística es un buen resultado, pero ofertarlos no hará que sea comprado y, por otro lado, imposibilita la visibilidad de otros ítems en el *ranking*.

Para generar confianza en los *SR*, se proporciona al usuario cierto grado de transparencia, al indicarles cómo se eligió cierto ítem, permitiéndoles corregir el perfil de ellos mismos, en el caso de que no les parezca adecuado lo que se les ofrece [36].

Evaluación de los SR

Las formas de evaluar este tipo de algoritmos, es midiendo las diferencias entre los *ratings* que el usuario aporta y el que devuelve el *SR*. Otra medida común es el grado en que las recomendaciones se ajustan a las compras reales, aunque pueden aparecen deficiencias que deben subsanarse por otras mediciones [36].

9.2.3. Fenómeno de Colas pesadas

La diversidad en los *SR* trata, por ejemplo, el caso de que si a alguien le gustan los libros referentes a pescar, en lugar de presentar todo tipo de libros de pesca, recomendar algo más genérico, como podría ser diferentes libros de actividades al aire libre.

Los comercios físicos (*brick and mortar*) únicamente pueden ofrecer un número limitado de ítems, los cuales, generalmente, son los más populares. En cambio, las empresas online pueden hacer que todos los artículos estén disponibles para el usuario, lo cual se representa por el fenómeno de largas colas o colas pesadas. Estas se refieren a algunas distribuciones estadísticas (Ley de potencias y distribuciones de *Pareto*, entre otras). Este modelo es distinto al modelo tradicional en el cual los productos que se deben vender son los que tienen mayor rotación.

La gráfica de esta distribución se puede apreciar en la Figura 9.1. Se ordenan las transacciones en orden decreciente de su cantidad de ocurrencias (la población disminuye gradualmente). En muchos casos, los acontecimientos de

baja ocurrencia, representados por la región celeste, pueden llegar a abarcar una parte más importante en el gráfico que la porción verde (más frecuentes).

Cuando una larga cola está presente, se acostumbra decir que se tienen en consideración los gustos de las minorías, y los usuarios disponen de una serie de opciones adicionales para elegir. También para las empresas, se trata de una oportunidad para introducir productos en la categoría de «nicho», lo cual fomenta la diversificación de productos. Las oportunidades que surgen a causa de esta situación afectan a la cultura de la sociedad: los proveedores tienen capacidades ilimitadas de almacenamiento físico y pueden satisfacer las demandas de los usuarios, que antes no eran conformadas.

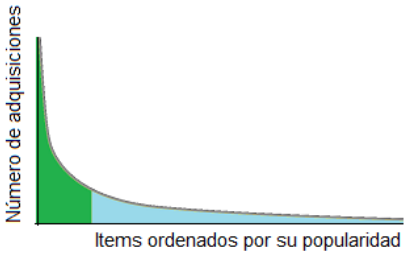

Figura 9.1: Largas colas (o colas pesadas)

Se espera que un *SR* del tipo filtrado colaborativo, aumente la diversidad ayudando a descubrir nuevos productos. Debido a que los sistemas basados en filtrado colaborativo pueden recomendar productos fundamentado en las ventas pasadas o *ratings*, si algunos productos no tienen mucha información histórica, no pueden ser recomendados. Este efecto se describe como: «Los ricos se hacen más ricos *(rich-get-richer)*», para productos con mucha demanda [4]. Por lo tanto, los *SR* del tipo filtrado colaborativo, se han desarrollado para promover la diversidad y las colas pesadas, llegando a la recomendación de nuevos e inesperadas ítems (fenómeno de *serendepity*, desarrollado más adelante).

9.2.4. Filtrado de Información (Information Filtering)

Al inicio de las búsquedas en Internet, la recuperación de información de grandes bases de datos ejecutadas mediante índices, o utilizando un atributo, era, relativamente, estática. Las técnicas de *IR* («*information retrieval*») basadas en *TF-IDF* (probabilidad de que en un determinado documento, sea una función de los términos de búsqueda que aparecen en dicho documento) son

reemplazadas por el concepto «*Information Filtering*», utilizado en los sistemas de recomendación. Dicho concepto trata de un flujo de nuevos contenidos, basados en la información obtenida de las necesidades, gustos o preferencias de cada individuo que son mayormente estáticas, si bien, el contenido que acarrean estos estados de ánimo, es dinámico. Por lo tanto, en lugar de indexar cada ítem, el cual será relevante por un corto período de tiempo, se trata de modelar las necesidades o preferencias del usuario, en función de información obtenida a partir de dichos ítems. Estos perfiles se pueden crear «a mano», o empleando el aprendizaje automático y, a través de *feedbacks*, mantenerse actualizados. El flujo comienza al llegar un nuevo contenido, éste pasa por los filtros mencionados y se construye un modelo adecuado a las preferencias del usuario [52].

9.2.5. Ratings

Para considerar un modelo de *SR* deben estar presentes tres elementos: usuarios, ítems y *ratings*.

Se identifican tres categorías de *ratings*: no personalizado (basados en filtro de contenidos), de colaboración (personalizados) y otros.

Ratings no personalizados

Se basan en resúmenes estadísticos no personalizados y son los obtenidos de las siguientes formas

- Datos de la comunidad, de origen externo.
- Resúmenes de *ratings* de la comunidad (por ejemplo el best seller o el mejor hotel en una ciudad). Estos resúmenes se construyen sobre una tabla de calificaciones que es una matriz donde se tienen usuarios e ítems.
- *ratings* basados en filtro de contenidos.

El usuario califica los ítems y a partir de allí, se construye el modelo sobre los atributos del ítem. El modelo se aplica a nuevos ítems y se califica tomando como base a los valores de los atributos o características. Por ejemplo, determinar que a un usuario en particular le gustan las películas románticas y que le gustan las películas donde actúa un determinado actor. Con estos sistemas se puede representar las preferencias del usuario en términos de los atributos, se puede predecir cuánto un usuario gustará de un ítem y generarle

recomendaciones. No sirve para determinar cuál ítem es el más popular entre los usuarios [52].

Un modelo puede no estar basado en las películas que le gustan al usuario, sino de las propiedades (atributos) de las películas que le gustan. Se origina una matriz llamada vector clave o un vector gusto, que tendrá elementos como: acción, ciencia-ficción, nombre de algún actor, etc. Cada vez que se tenga la opción de valorar una película, se actualizará el modelo, es decir, se modifican los valores. Cuando llegue el momento de hacer una recomendación, dada una película, se calcula su valor en cada uno de esos atributos. Se calcula el producto escalar de los dos vectores (el de preferencias del usuario y el de la película) y, de esta forma, se estima cuánto le puede gustar la película a un usuario que no la haya visto antes.

Ratings de colaboración (personalizados)

Para predecir se utiliza la opinión de otras personas en lugar de los atributos de los ítems. Incluyen el filtrado colaborativo ítem–ítem y filtrado colaborativo usuario-usuario. Es un modelo en el que intervienen dos entidades: el usuario (que construye un grupo de puntajes) y los artículos, basado en el grupo de puntajes obtenidos. Con estos valores se construye una matriz de *ratings*. El objetivo es, o bien llenar esos valores que faltan predecir, o seleccionar celdas que puedan ser elementos a los que se busca recomendar. En la matriz, las filas son los usuarios y las columnas los artículos. La matriz es, normalmente, *sparse*. Si no lo fuera significaría que se tendrían las preferencias de los usuarios casi completas.

En el filtrado colaborativo usuario-usuario, se selecciona un vecindario de individuos con gustos afines y se utilizan sus opiniones.

En el filtrado colaborativo ítem-ítem, la forma de averiguar sobre un determinado artículo, es seleccionar otros artículos, para los cuales se tiene una opinión sobre ellos, y ver cómo otras personas calificaron al ítem considerado. Se establecen relaciones entre los artículos, a través de las votaciones, usando esos artículos y sus puntuaciones, para triangular recomendaciones.

Otros ratings

Se incluyen los recomendadores interactivos, basados en críticas y diálogos y los sistemas de recomendación híbridos [52].

Tipo de preferencias

Se identifican dos tipos de preferencias: explícitas e implícitas.

- Explícitas.

  Incluyen: calificar, realizar una revisión y votar (el usuario brinda información a otros). Es necesario decidir cuál es la interfaz: sistemas de estrellas, pulgar arriba y abajo, «me gusta» o calificación unitaria (+1 si le gusta al usuario y no hay acción si no le gusta), entre otros.

- Implícitas

  Son las detectadas por el *SR* y pueden ser realizadas durante el consumo del ítem (obtención de datos basados en consumo), luego del consumo (basado en la memoria del usuario) y de acuerdo a la expectativa (cuando aún no se ha consumido).

9.2.6. Ranking

En esta sección se trata de observar las distintas maneras de calcular y desplegar las predicciones, teniendo en cuenta consideraciones tales como calificar ítems cuyos datos son escasos e incluso cambiantes con el tiempo.

Controversia

Se denomina controversia en los *SR* a la situación en la que se tienen muchos votos a favor y muchos en contra, sobre un ítem determinado. Es diferente un ítem que sólo tiene 20 calificaciones «me gusta», a otro que tiene *7000* «*me gusta*» y *6980* «*no me gusta*» (en el primero todos aunque sean solo *20* han votado «me gusta» y en el segundo, aunque la diferencia entre las dos opciones es *20*, las votaciones están repartidas aproximadamente a la mitad, entre las dos).

Mostrar las preferencias

El propósito de mostrar las preferencias, es de ayudar al usuario a comprar, leer o mirar un ítem. Hay diversas formas de calcular las preferencias:

- Promedio de las preferencias.

- Proporción de votos positivos.
- Mostrar las diferencias entre votos a favor y en contra.

Las formas de mostrar los resultados de los *SR* son:

- *Rating* promedio: Presenta la desventaja de depender de la forma de calificar de los individuos. Algunos utilizan la parte alta de la escala y otras, la parte baja.
- *Net upvotes*: Son el número de «me gusta» de un *SR*. Muestran la popularidad de un ítem. No son de utilidad en el caso de una controversia.
- Porcentaje mayor o igual a un valor: este tipo de rating es interesante desde una perspectiva de soporte, pues se interpreta directamente que la persona votó positivamente a un ítem. También mejora el problema de que las personas, en ciertos sistemas de recomendación, votan de por sí «alto» a ítems que otros votan «bajo».
- Distribución completa de votos: La distribución total de los *ratings* consiste en mostrar dicha distribución al usuario. Sirve para que los usuarios vean lo que le gusta a la gente normalmente.

Confianza

Se trata de determinar cuál es la confianza, dado un ítem, que éste sea tan bueno como dice su score. La confianza es afectada cuando se dispone de pocos datos. A través de un promedio amortiguado, se puede solucionar la escasez de datos [52]. Para ello, se asume que, sin evidencia, todos los ítems están en el promedio. Se inicializan todos los ítems con el promedio global (suma de los *ratings* de los usuarios sobre la cantidad de ellos). Luego, al tener k *ratings* sobre u ítems, se utiliza la Ec. (9.1)

$$\frac{\sum_u r_{ui} + k\mu}{n + k} \tag{9.1}$$

Se asume que cada ítem tiene un número arbitrario k de *ratings* adicionales multiplicado por el promedio global de los *ratings* (μ).

El valor de k controla al valor promedio ya que lo multiplica y varía en base a la evidencia recibida del ítem. A medida que aumentan los datos reales, los sumandos adicionales pierden relevancia. El parámetro k modera el efecto de tener, inicialmente pocos *ratings* en algún extremo.

Tolerancia a riesgos

Se trata del balance entre el beneficio de que una aplicación obtenga una buena recompensa, al tomar un alto riesgo y la realización de recomendaciones conservadoras. Para solucionar la tolerancia a riesgos se utilizan intervalos de confianza estadísticos. Se puede estimar cuál es la confianza en el promedio o la probabilidad de que alguien califique algo positivamente. A medida que se disponen de más evidencias (datos) dicho intervalo se va reduciendo. Utilizando el extremo inferior, se obtendrá una recomendación conservadora, por otro lado, si se toma el extremo superior implica que se corren ciertos riesgos al momento de tomar decisiones sobre las cuales, no se tiene suficiente información como para saber qué implicancias o riesgos presentan.

Desactualización (Dominio del tiempo)

Se refiere a que puede ocurrir que el ítem recomendado sea viejo y ya no sea utilizado [52]. Se nombran dos soluciones *ad hoc*.

El sitio hacker news (*https://thehackernews.com/*) utiliza una fórmula que puntúa ítems basado en el tiempo (Ec.(9.2))

$$\frac{(U - D - 1)^{\alpha}}{(t_{now} - t_{post})^{\gamma}} \times P \tag{9.2}$$

El numerador tiene el total de *upvotes* menos los *downvotes* menos uno y el denominador indica la edad del ítem en horas, que decae polinómicamente mediante un exponente γ. De esta forma los nuevos ítems tienen una mayor influencia (γ puede ser $1,8$ y es llamado gravedad).

El factor P indica una penalización del tipo real de negocio para influenciar o no una determinada comunidad, por parte del dueño del sitio Web. Por ejemplo si un artículo no es interesante y el dueño del sitio no quiere que se le recomiende demasiado a la comunidad que lo sigue, P es el factor a utilizar para ajustarlo.

El sitio de noticias *Reddit* (*https://www.reddit.com*) es una aplicación que trata a los upvotes y al tiempo en forma independiente. Por un lado toma el logaritmo en base 10 del máximo entre 1 y $|U - D|$, moderando las votaciones iniciales (de *11 a 100* tienen casi el mismo impacto que los 10 primeros votos) luego agrega el término de decaída, considerando el tiempo en que fue agregado el ítem. Si tiene más votos negativos la puntuación es mala colocándolo al final de la lista (9.3).

$$\log_{10} max(1, |U - D|) + \frac{sign(U - D)t_{post}}{45000} \tag{9.3}$$

Dominio y consideraciones de negocios

Entre otras muchas consideraciones, se refiere a las edades o comunidades a las que va dirigido el *SR*: contesta a la pregunta si es comprensible el ranking para los usuarios a los que va dirigido.

9.2.7. Evaluación de los SR

Se considera la evaluación para *SR* colaborativos. Las métricas en los *SR* son utilizadas para evaluar un algoritmo de recomendación, o las bondades de un *SR* al hacer una recomendación. La evaluación de los *SR* es dificultosa por las siguientes razones:

- Son dependientes en algunos casos del *dataset* (considerar el caso en que los datos tengan más usuarios que ítems y viceversa). Consideraciones similares se deben tener en cuenta como ser: densidad y escalas de los *ratings*, entre otros.
- El objetivo del *SR* puede influenciar su evaluación (puede ser que el objetivo es la precisión, u orientado a la toma de decisiones o a lograr la satisfacción del usuario).

Las medidas de evaluación se pueden clasificar en tres categorías [53]:

- Precisión en la predicción (se mide que tan alejado está la predicción de la realidad).
- Precisión en la clasificación (frecuencia en que el *SR* toma decisiones correctas o incorrectas de los ítems).
- Precisión en el ranking (performance de un *SR* al ordenar los ítems en la salida del algoritmo).

Se describen las siguientes métricas: precisión en el ranking, soporte para la toma de decisiones, centradas en el usuario, precisión al predecir y para objetivos empresariales.

La evaluación de los *SR* también puede hacerse a nivel de sus salidas: predicciones y recomendaciones incluyendo los *top n*.

Métricas para la precisión al predecir

Las mediciones de precisión y medidas de error calculan hasta qué punto una predicción está alejada de los datos que se disponen y forman un conjunto de métricas de soporte para la toma de decisiones de un usuario.

Métricas de errores

Desde sus inicios, las medidas de precisión y error empleadas con los *SR* son: *MAE*, *RMSE* y *MSE*. Éstas miden qué tan alejada se encuentra la predicción del sistema, de la real del rating de un usuario, en relación a un ítem [53].

Los factores a considerar, entre otros, pueden ser:

- Cálculo del promedio.
- Determinar si cada predicción tiene el mismo peso.
- Determinar si es lo mismo considerar a un usuario que calificó muchas veces frente a otro que no lo hizo (a nivel de algoritmos se debe analizar si es lo mismo hallar el *MAE* de un usuario que hace, por ejemplo, *2000* predicciones frente a otro que hace *30000*).

A continuación se presentan *MAE, MSE, RMSE* y *NMAE* aplicados a *SR*.

MAE (mean absolute error)

Se presenta en la Ec. (9.4). Es el promedio de la diferencia entre la predicción y el rating (opinión real), simbolizados por P y R, respectivamente. A los efectos de que no se pierdan posibles cancelaciones entre valores, se toma el valor absoluto $|P - R|$.

$$MAE = Average(|P - R|) = \frac{\sum_{ratings} |P - R|}{\#ratings} \tag{9.4}$$

MSE (Mean squared error)

En forma similar se define el *MSE* (Ec. (9.5)).

$$\frac{\sum_{ratings} (P - R)^2}{\#ratings} \tag{9.5}$$

Elimina el valor absoluto ya que se considera el cuadrado de la diferencia y penaliza más los errores grandes que los pequeños. La desventaja es que no

es intuitivo (la escala en que mide no permite una interpretación de lo que significa, por ejemplo, obtener $0,7$ estrellas al cuadrado de error).

RMSE (Root Mean Squared Error)

Para solucionar el problema anterior se calcula la raíz cuadrada del *MSE* en la Ec. (9.6).

$$RMSE = \sqrt{\frac{\sum_{ratings}(P-R)^2}{\#ratings}} \tag{9.6}$$

MAE Normalizado (NMAE)

Un *MAE* de 0.3 indica que un *SR* está 0.3 alejado la predicción del valor real aunque este valor está asociado a la escala que se utilice (se debe considerar que es distinto si 0.3 es sobre una escala de *1 a 10* a que lo sea sobre una escala de 1 a 5).

A los efectos de evitar el problema que surgiría de comparar distintos *SR*, los cuales pueden variar sus respectivas escalas de *ratings*, se puede normalizar el *MAE*. El *NMAE* normaliza errores dividiendo el *MAE* sobre el rango de los *ratings*, Ec. (9.7).

$$NMAE = \frac{1}{n(r_{alto} - r_{bajo})} \sum_{u,i} |p_{u,i} - r_{u,i}| \tag{9.7}$$

Donde r_{bajo} y r_{alto} son los máximos y mínimos en las escalas de *ratings* consideradas. Los valores del *NMAE* varían en *[0,1]*. Son difíciles de interpretar para un *SR* pero útiles para evaluar la performance de éstos.

Métricas centradas en el usuario

Se enumeran las siguientes medidas para los *SR* tomando en consideración al usuario:

- Cobertura (cuán a menudo un usuario acepta la predicción de un ítem).
- Retención del usuario (determinar si el usuario aún sigue utilizando el *SR*).
- Captación de la recomendación (comprobar si el usuario acepta la recomendación).

- Satisfacción (verificar si la recomendación satisfizo al usuario, lo que habitualmente es medido por encuestas).

Métricas de soporte para la toma de decisiones

Miden cuánto un *SR* ayuda al usuario a tomar una buena decisión. Se analizarán:

- Tipos de errores al evaluar
- Precisión y exhaustividad (P y R)
- Métrica Sensitividad y especificidad
- Mean-average precission (MAP)

Tipos de errores al evaluar

El objetivo del *SR* es ayudar al usuario a tomar una buena decisión. Por ejemplo si se predice un score alto y al usuario no le gusta la película, el *SR* falló. Otro error puede ocurrir si, al considerar una lista *top-n*, apareciese una mala película en ella.

Precisión y exhaustividad (P y R)

Actualmente lo más utilizado es la estadística construida sobre métricas de precisión y recuperación de la información. Estas métricas se emplean en dominios unarios (como es el caso de un sitio de ventas donde el usuario compra o no hace nada) por lo que, en otros dominios, el *SR* debe convertir la salida a una escala binaria (podría ser, por ejemplo, considerar un umbral que separa los dos valores).

Al momento de la utilización de esta métrica, se debe tener presente:

- Se necesita disponer de datos reales de los usuarios. El problema de P y R es que fueron desarrollados con la idea de que el usuario conoce si el ítem es bueno o no, asumiendo que es la misma opinión para todos los usuarios. Sin embargo, determinar si algo es bueno o no, es personal a cada usuario. Una solución es simular una P y R, a través de la limitación del análisis a ítems que ya han sido ranqueados. En esencia, cubren el rating de un ítem del *dataset*, observando si el *SR* lo recomienda y luego lo compara contra el rating recibido. Esto permite identificar sesgos. Un sesgo, por ejemplo, es el introducido por muchos usuarios que califican sólo positivamente (para ellos no existen ítems que son malos).

- P y R cubre todo el *dataset* , no una lista de artículos más recomendados (*top n*). Para resolver esto se puede utilizar «*precision@n*» definida en la Ec. (9.8).

$$P@n = \frac{N_{r@n}}{n} \tag{9.8}$$

Con «*precision@n*» se resuelve el problema de enfocarse en todo el *dataset*, haciéndolo sobre los más relevantes (*top n*). Pero persiste la limitación de que para un nuevo ítem, si no se tiene un *rating*, el *SR* asume que no puede ser recomendado.

Métrica Sensitividad y especificidad

Precisión y recuperación no hacen uso del cuadrante inferior derecho, correspondiente a los TN (verdaderos negativos) de la matriz de confusión que corresponden a los correctamente clasificados que son irrelevantes o no gustan. Para ello se introducen nuevas métricas: sensitividad y especificidad [53].

La sensitividad mide la cantidad de positivos calificados como tales sobre el total que son relevantes (es igual a la recuperación).

$$\frac{TP}{(TP + FN)} \tag{9.9}$$

La especificidad mide la cantidad de negativos calificados como tales sobre el total que son irrelevantes. Es la fracción de ítems irrelevantes correctamente descartados.

$$\frac{TN}{(FP + TN)} \tag{9.10}$$

Mean-average precission (MAP)

Se trata de un híbrido entre búsquedas en un ranking y soporte a la toma de decisiones. En *IR* (recuperación de la información), el MAP toma múltiples *queries* a lo largo de las diferentes posiciones de los primeros n resultados [53].

$$MAP = \frac{\sum_{q=1}^{Q} AveP(q)}{Q} \tag{9.11}$$

donde:

$AveP = \frac{\sum_{k=1}^{n} P(k)*rel(k)}{\#docs\ relevantes}$

$P(k)$ es la precisión que corresponde al *SR* para una recuperación k.

$rel(k)$ tiene el valor 1 si el ítem en la posición k del *ranking* es relevante, sino 0.

Q es el número total de *queries*, n es el de *top* n.

Lo que en realidad se calcula, es el área bajo la curva de $P - R$, promediándose sobre las *queries*. En los *SR*, el *MAP* se ha adoptado en muchas formas especialmente sobre usuarios lo que hace difícil la estandarización para su cálculo [53].

Análisis ROC, ventajas y desventajas

La curva refleja la variación de *TP* en función de los *FP* al variar el punto de corte.

Ventajas

- Es apropiado para evaluaciones de sistemas de recomendación frente a conjuntos de usuarios reales.
- Las métricas consideradas son bien conocidas en sí mismas, así como también en sus propiedades.
- *ROC* es una medida robusta basada en un número.

Desventajas

- Se debe conocer de antemano si una recomendación específica ha sido de utilidad por un usuario final, lo que es difícil de obtener.
- Para ser debidamente probada se necesita disponer de grandes conjuntos de datos.

Métricas de precisión en el ranking

Evalúan la performance de un *SR* al ordenar los ítems obtenidos en la salida del algoritmo.

Hay cuatro diferentes algoritmos: *Mean Reciprocal Rank*, *Spearman Rank Correlation*, *Discounted Cumulative Gain* (también se define *nDCG*) y *Fraction of Concordant Pairs*. Los más utilizados son *MRR* y *nDCG* [53].

Mean Reciprocal Rank (MRR)

Es una medida estadística para evaluar un proceso que produce una lista de resultados ordenados por la probabilidad de su correctitud. El ranking recíproco mide qué tan lejos se debería avanzar para encontrar un resultado útil. Suponiendo que la recuperación de ítems es una secuencia, e i el *ranking* (lugar en la lista) del primer ítem aceptable, su recíproco se calcula como *1/i*.

El *MRR* toma la lista de testing de recomendaciones utilizada para medir la calidad del *SR* y se computa la media del recíproco de los *rankings* de cada lista, siendo ése el *score* del algoritmo [53].

$$MRR = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{rank_i} \tag{9.12}$$

En divisor $rank_i$ significa el lugar del *ranking* del primer elemento relevante en la *query* i.

Ejemplo 9.2

En la tabla 9.1, se presentan datos de diferentes *queries* sobre distintas palabras

Query	Resultado	1er Respuesta correcta	Ranking	Ranking Recíproco
Comidas	Perro, gato, sopa	Sopa	3	1/3
Cursos	Análisis 1, Lógica	Análisis 1	1	1
Números	Agua, uno, tres	Uno	2	1/2
SO	Switch, Linux, dos	Linux	2	1/2

Tabla 9.1: Tabla de valores para *Mean Reciprocal Rank*

$$MRR = (1/3 + 1 + 1/2 + 1/2)/4 = 0.58$$

El *MRR* es de utilidad para determinar las bondades del *SR* al ordenar los ítems. Cuanto más próximo se encuentre el *MRR* al valor 1, mayor será la calidad. Esto mide qué tan bien el *SR*, coloca los ítems buenos al principio o cuántos ítems no deseados por el usuario están antes que el primero bueno.

Spearman Rank Correlation (SRC)

La correlación de *Spearman Rank* es la correlación de *Pearson* de los rankings de los ítems (*Pearson* fue utilizado en filtrado colaborativo usuario-usuario). El método requiere tener algo con lo cual comparar. Para ello, se utiliza el *output* rankeado para el *SR*. También se necesita un *ranking* obtenido de mediciones reales. Si se dispone de *rankings*, sólo se deben ordenar los ítems de acuerdo al rating del usuario y si no se dispone se necesita algo como *ground truth* (conocimiento obtenido del terreno) para computar la correlación. El método *SRC* penaliza errores de clasificación, Ec. (9.13).

Considerar el *dataset* $r_1 = \{r_1(1), r_1(2), ..., r_1(n)\}$ y otro conjunto de datos $r_2 = \{r_2(1), r_2(2), \ldots, r_2(n)\}$, donde r_1 es el *ranking* correcto y r_2 es el que se obtiene del *SR*.

$$src = \frac{\sum_i (r_1(i) - \mu_1)(r_2(i) - \mu_2)}{\sqrt{\sum_i (r_1(i) - \mu_1)^2}\sqrt{\sum_i (r_2(i) - \mu_2)^2}} \tag{9.13}$$

μ_1 representa el promedio de las observaciones de r_1: $\mu_1 = \frac{1}{n}\sum_{i=1}^{n} r_{1i}$

Si los dos órdenes de presentación de los ítems son el mismo, la correlación del rango de Spearman es 1. Pero si el *SR* coloca los ítems en orden equivocado, se penaliza este mal posicionamiento bajando el valor de la correlación. El *SRC* penaliza cualquier mal posicionamiento de igual forma. Esta penalización uniforme no es conveniente, pues trata de igual manera tanto si el *SR* falla en el lugar 11 al 13, a que si falla en el puesto 1 al 3 (a los efectos prácticos, interesa observar lo que sucede en los primeros lugares de la lista).

Discounted Cumulative Gain (DCG) y nDCG

Intenta evitar el problema que se mencionó del *SRC*. Busca un algoritmo que tenga mayor peso en los primeros ítems que en los últimos. Para ello utiliza una función de descuento. La ganancia acumulada (*Cumulative Gain, CG)*, es la suma de la relevancia (calificada) de los resultados de una búsqueda. Para una posición p, del *ranking,* el *CG* se define en la Ec. (9.14).

$$CG_p = \sum_{i=1}^{p} rel_i \tag{9.14}$$

rel_i es la relevancia en el lugar i (habitualmente, tiene el valor cero o uno).

La ganancia acumulada con descuento (*Discounted Cumulative Gain, DCG*) considera que los ítems relevantes que aparecen en una posición más baja en un resultado de una búsqueda, deben ser penalizados en su relevancia calificada, reduciéndola logarítmicamente respecto a la posición del resultado. El *DCG* para una posición p se define en la Ec. (9.15) [52].

$$DCG_p = rel_1 + \sum_{i=2}^{p} \frac{rel_i}{\log_2 i} \tag{9.15}$$

Los resultados de búsqueda varían su longitud de acuerdo a la consulta. Por lo tanto, el empleo de *DCG* no es aconsejado para comparar diferentes búsquedas. Para subsanar este problema, es conveniente normalizar. Se define la ganancia normalizada acumulada con descuento en la Ec. (9.16).

$$nDCG(R) = \frac{DCG(R)}{DCG(R_{perfect})} \tag{9.16}$$

$R_{perfect}$ es un *ranking* perfecto que significa que la lista de *ratings* se encuentra ordenada perfectamente (por el *ranking* real).

Caso de estudio: Datos unarios

Son aquellos que no poseen *ranking*, sólo consisten de un «me gusta» o «clickear sobre un link», por ejemplo. Los datos unarios o unitarios siempre se consideran positivos (la calificación consiste en un «click» del usuario frente a no hacerlo). Su uso es habitual en aplicaciones comerciales donde el usuario compra o no un ítem. No se tiene conocimiento si la ausencia de compra es debido a que no le agradó el producto, o por algún otro hecho. En los *SR* basados en datos unarios, se encuentra el problema de la imposibilidad de interpretación cuando el usuario elige una opción o cuando no opta por ninguna. Al no disponerse de casos negativos aparece el problema de cómo evitar evaluar al *SR* penalizándolo en forma injusta. El desafío de los datos unarios radica en que, sin información, no permiten determinar si las recomendaciones para el usuario son acertadas.

Métricas para objetivos empresariales

Desde un punto de vista de los objetivos empresariales normalmente no interesa la precisión, sino aumentar las ventas. Las diferentes medidas no sólo están relacionadas a la experiencia de los usuarios o a verificar si el *SR* es

correcto, sino a los objetivos para los cuales éste se implementó (generalmente, el *SR* tiene un fin comercial) y en ocasiones se debe medir ese propósito. En el sentido de la búsqueda de propósitos específicos aparecen nuevos métodos más sofisticados: cobertura, diversidad y *serendepity*.

Cobertura

Es el porcentaje de productos para los cuales el *SR* puede hacer una predicción o predicciones personalizadas o predicciones basadas en un umbral de confianza. Es utilizada para los casos donde existen predicciones directas, como métricas de fondo para comparar recomendadores $top - n$.

Diversidad y diversificación

Evalúa qué tan diferentes son los ítems recomendados. Se puede construir una lista de similaridad de un ítem con el resto y calcular su promedio. Los que obtengan un valor menor son los más disímiles del resto. Diversificar una lista de $top - n$ es retirar al ítem más similar a los anteriores de dicha lista, y cambiarlo por el que sigue en el lugar $n + 1$ [52].

Serendepity

El *serendipity* puede ser considerado como una métrica que mide una recomendación que resulta agradable, como si fuese una sorpresa o un descubrimiento inesperado. En los *SR* significa obtener un resultado inesperado que condujo a una agradable experiencia. Este estado de ánimo se puede representar por medio de la fórmula dada en la Ec. (9.17).

$$serendepity = \frac{1}{N}\sum_{i=1}^{N} max\{Pr(s_i)Prim(s_i), 0\} * rel(s_i) \tag{9.17}$$

$rel(s_i)$ es la relevancia del ítem s_i.

Pr es la predicción o la probabilidad de una predicción positiva dada por el *SR*.

$Prim$ representa a una predicción primitiva (que debe calcularse) y que mide la obviedad del ítem.

Se calcula cuánto difieren entre sí, *Pr y Prim*. Si la diferencia es positiva significa que la predicción del ítem es un valor mayor al de ser obvio. Si el valor de la primitiva es mayor que la predicción (es más obvio), se obtiene un

resultado negativo. En este caso, el valor del *serendepity* es cero. El N que se considerará es el de la lista *top-n*. La primitiva de obviedad puede ser, por ejemplo, medida por la popularidad. En la práctica no sería necesario efectuar los cálculos, sino que bastaría con eliminar los ítems más populares u obvios lo cual implica experimentación y ajustes.

Se puede utilizar un cálculo de diversificación como enfoque para conservar los ítems que provocarían más *serendepity*. El objetivo es que los usuarios consuman artículos menos populares y que se encuentran en lugares más profundos del catálogo (colas pesadas).

Objetivos empresariales para un SR

En este caso no se orientan al usuario que compra, sino que las métricas son medidas para el comercio que tiene su *SR*. Sobre los siguientes aspectos se puede evaluar un *SR* desde el punto de vista de una empresa.

- Venta inmediata: obtener un cliente que compra más en un determinado instante.
- Venta inmediata sin devolución: que el cliente no devuelva sus compras (El *SR* debería haber recomendado algo útil para evitar esta situación).
- Valor a largo plazo del cliente: es el tiempo de espera hasta la próxima compra, la empresa espera que el cliente retorne para efectuar nuevas compras.
- Referenciación: el cliente puede recomendar este sitio de compras a otros usuarios.

9.2.8. Los problemas del arranque en frío

En general, cuando entra en producción un *SR*, o cuando se ofrece un nuevo producto, se presenta el problema de arranque en frío. De esta manera se denomina a situaciones derivadas de la presencia de nuevos usuarios, nuevo sistema y nuevos ítems, que se reflejan en la falta de datos para poder operar. El problema de arranque en frío se presenta de dos formas principalmente.

- Recomendar nuevos elementos.
- Recomendaciones a los nuevos usuarios.

Posibles soluciones

- Preguntar/forzar a los usuarios a valorar un conjunto de elementos
- Utilizar otros métodos (por ejemplo, basada en el contenido, demográficas o, simplemente, no personalizado) en la fase inicial
- Valoración por defecto: asignar valores por defecto a los elementos que sólo uno, o pocos, usuarios hayan valorado.

9.3. Funcionamiento de los sistemas de Recomendación

Los sistemsa de recomendación pueden ser orientados a contenidos o colaborativos.

9.3.1. Sistemas de recomendación basados en contenidos

La idea principal de este tipo de *SR* es efectuar recomendaciones a un cliente, basado en la confección de un perfil de preferencias de atributos asociados a ítems, los cuales fueron previamente ranqueados altos por el propio cliente. Por ejemplo, películas con los mismos actores que actuaban en películas calificadas altas por el cliente, artículos en la Web con el mismo contenido, etc. Dado un usuario se busca un set de ítems que éste ha calificado alto ya sea utilizando datos implícitos o explícitos. Con estos datos, se construye un perfil del ítem (ítem - profile). Por ejemplo si se considera el ítem «libro», siendo que al usuario le gustan «novela» y «enciclopedia», se puede construir el perfil del ítem (éste sería: libro, novela y enciclopedia). A partir de este perfil del ítem, se puede construir un perfil del usuario y, luego de tener dicho perfil, se puede comparar con el catálogo para presentarle otras ofertas que podrían gustarle (de esta forma se obtiene un *SR* basado en contenidos).

El universo de las posibles palabras claves define el contenido del espacio donde cada palabra es una dimensión; cada ítem tiene una posición en el espacio definida por un vector y cada usuario tiene un perfil de gustos o preferencias, que también constituye un vector en el espacio. La comparación de qué tan parecidos son el perfil de gustos y el vector de palabras claves del ítem es una medida de qué tan cercanos están el producto y el gusto del usuario.

Ejemplo 9.3

En la Figura 9.2 se ve la representación de un ítem o producto (por ejemplo una determinada película de nombre *P1*) cuyas dimensiones en el espacio de palabras claves son romance y entretenimiento. También se observa el vector de gustos *G1* correspondiente a un usuario a quien le gusta poco el romance y mucho el entretenimiento. La distancia del coseno entre *P1* y *G1*, indicaría cuánto puede llegar a gustarle el ítem *P1*, al usuario cuyo perfil de gustos es *G1*.

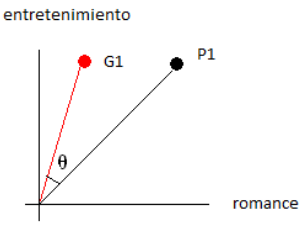

Figura 9.2: Ángulo entre el perfil de gustos y de ítems

El cálculo del coseno del ángulo es más fácil de realizar, que determinar el ángulo entre esos vectores. Si se obtiene el valor 1, es una indicación de coincidencia total.

$$\cos\theta = \frac{P1.G1}{\parallel P1 \parallel \parallel G1 \parallel}$$

Interesa, además, reducir dimensiones innecesarias. Por ejemplo, las palabras «romance» y «romántico» deben definir la misma dimensión.

Perfiles

Los perfiles se definen tanto con los ítems como con los usuarios.

Perfil de los ítems

Una forma de representar un ítem es a través de un vector con todo el universo de ítems y colocar un 0 o 1 de acuerdo a si el ítem se encuentra o no, en el conjunto, basado en datos que se disponen del usuario. También puede ser el conteo de las ocurrencias o $TF - IDF$, entre otros.

Perfil del usuario

El perfil del usuario es necesario crearlo y debería tener los mismos componentes que los del ítem. La base para obtener esta información se puede extraer de la matriz de utilidad, la cual combina los gustos con la importancia. Un usuario puede tener más de un perfil.

Similaridad y predicción

Como se planteó anteriormente, la idea es tomar un par formado con el perfil usuario y el perfil ítem, y calcular cuál sería el rating de ese usuario para ese ítem. Ambos son vectores en un espacio de alta dimensión. Esta técnica es usada para la búsqueda en documentos.

La finalidad es, dado un espacio de palabras claves aplicado sobre un conjunto de documentos d y una consulta q de búsqueda sobre ellos (representada por un vector con las palabras seleccionadas), encontrar los documentos que más coinciden con la consulta q.

Para ello, la medida de similaridad a utilizar, será la de similaridad entre documentos (conteo de ocurrencias). Frecuentemente estos vectores son normalizados. Generalizando, se obtiene la distancia del coseno:

$$\cos\theta = \frac{d.q}{\parallel d \parallel \parallel q \parallel} \tag{9.18}$$

Donde:

$$d_j = (w_{1,j}, w_{2,j}, \ldots, w_{t,j})$$

$$q = (w_{1,q}, w_{2,q}, \ldots, w_{n,q})$$

$$\parallel q \parallel = \sqrt{\sum_{i=1}^{n} {q_i}^2}$$

Sea $V_d = [w_{1,d}, w_{2,d}, \ldots, w_{N,d}]^T$ el vector de pesos para un documento d, donde $w_{t,d}$ es el $TF - IDF$ de t en d.

$$w_{t,d} = tf_{t,d}.log\frac{|D|}{|d^1 \in D|t \in d^1|} \tag{9.19}$$

$tf_{t,d}$ es la frecuencia del término t en el documento d.

$\log \frac{|D|}{|d^1 \in D | t \in d^1|}$ es la frecuencia inversa del documento y $|D|$ es el número total de documentos.

$|d^1 \in D | t \in d^1|$ es el número de documentos que contienen al término t.

La similaridad se calcula utilizando la distancia del coseno (Ec. (9.20)).

$$sim(d_j, q) = \frac{d_j.q}{\parallel d_j \parallel \parallel q \parallel} = \frac{\sum_{i=j}^{N} w_{i,j} w_{i,q}}{\sqrt{(\sum_{i=j}^{N} w_{i,j}^2)} \sqrt{\sum_{i=j}^{N} w_{i,q}^2}} \tag{9.20}$$

Una ventaja que presenta este modelo sobre una representación en un vector de booleanos es que los *rankings* tienen diferente relevancia. Sin embargo, tiene la limitación de que documentos grandes no quedan bien representados, por lo que pueden aparecer falsos positivos y falsos negativos.

LSH

Si las dimensiones con las que se trabajan son grandes y se consideran *SR* para documentos (deportes, moda, etc.), se puede considerar la búsqueda de artículos similares, utilizando un algoritmo *LSH*. En tal caso, se tendrá en consideración qué tipo de similaridad se utilizará. Se puede emplear a la distancia de *Jaccard*, si se trata de la distancia entre conjuntos de palabras, o distancia del coseno, si los conjuntos de palabras son tratados como vectores.

Las técnicas presentadas de *shingling*, *minhashing* y *LSH* buscan una similaridad lexicográfica, sin embargo, en los *SR* se busca la similaridad entre algunas palabras comunes entre los documentos. La metodología para hallar documentos similares sería, luego de tener la medida de distancia de *Jaccard* o del coseno, hacer un *minhash* para *Jaccard* o hiperplanos aleatorios para distancia del coseno y aplicar *LSH* a los efectos de hallar pares de documentos similares [4].

Ventajas

- Totalmente basado en contenidos.
- El perfil es entendible.
- Los cálculos son computacionalmente posibles.

- Flexibilidad (puede integrarse con sistemas basados en consultas o en casos).
- No se necesitan datos de otros usuarios como sucede en el filtrado colaborativo para hacer recomendaciones.
- Existe la posibilidad de recomendar ítems nuevos y no conocidos (ya que no dependen del *rating*).

Desventajas

- Al ser el modelo tan simplificado no puede manejar el concepto de interdependencia. Por ejemplo, si a un usuario le gusta el ítem a en la categoría A y adicionalmente le gusta el ítem b en la B, esta realidad no puede ser expresada en un espacio de palabras claves.
- Encontrar las características adecuadas es complejo. Por ejemplo, imágenes, películas, música.
- Es propenso a sobreajuste.
- Nunca recomienda artículos a usuarios cuyo perfil no es el seleccionado, no considerando que las personas pueden tener múltiples intereses.
- No es capaz de utilizar las consideraciones de calidad hechas por otros usuarios.
- Problema de inicio en frio para nuevos usuarios. Se trata de construir un perfil de usuario para una persona interesada que no ha calificado aún a ningún ítem. Generalmente, se empieza con un perfil promedio y luego evoluciona con los datos aportados por el usuario en forma más personal.

9.3.2. Filtrado colaborativo

Se trata de determinar un conjunto de N usuarios, cuyos gustos sean similares a los de un usuario u. A este conjunto de N usuarios se les denomina vecindario de usuarios similares a u. El objetivo del sistema es encontrar otros ítems que son calificados altos por los N a los efectos de recomendarlos a u.

	P1	P2	P3	P4	P5	P6
Usr1	5		5	5		1
Usr2	4	2				
Usr3			1	1	4	
Usr4		2				4

Figura 9.3: Ejemplo Usuario-ítem

La similaridad entre usuarios debería medirse de manera tal que, si dos usuarios califican dos ítems en forma muy diferente, su distancia deberá ser mayor a que si calificaran de forma parecida.

La tabla de la Figura 9.3 contiene varios ítems que no han sido ranqueados, por lo que la similaridad entre vectores debería definirse considerando que muchas coordenadas de los vectores son desconocidas. Se busca determinar la similaridad que existe, entre aquellos usuarios con gustos similares, en comparación con otros, que no comparten los mismos gustos (la distancia de *Jaccard* no sería útil pues considera cantidades de ítems y los *SR* toman en consideración a las calificaciones) [52].

Utilizar la distancia del coseno, ingresando el valor cero en las entradas donde no hay valor, sería una manera errónea de medir la similaridad, ya que se considera cero a los ítems que podrían tener algún otro valor. A los efectos de subsanar esta carencia, se utiliza el concepto de coseno centrado. Éste considera cada valor, restándole la media de su correspondiente fila en la matriz, obteniéndose una matriz normalizada [52].

En la tabla de la Figura 9.4 se aprecian los valores del coseno centrado correspondientes a la tabla de 9.3. Por ejemplo a los valores iniciales de la primera fila se le restó el valor de la media (en este caso es 4).

	P1	P2	P3	P4	P5	P6
Usr1	1		1	1		-3
Usr2	1	-1				
Usr3			-1	-1	2	
Usr4		-1				1

Figura 9.4: Valores utilizando el coseno centrado

Notar que si se suma cualquier fila se obtiene el cero, eso es debido a que se centró el *rating* de cada usuario respecto al cero. Un *rating* positivo, se interpreta como que al usuario le gustó un determinado ítem por arriba del promedio y viceversa. En este caso, la distancia del coseno se vuelve la distancia entre los vectores de los *ratings* centrados. Los valores obtenidos son más representativos ya que no se consideraron los valores desconocidos como si fueran cero (*rating* negativo), sino como el promedio.

La dualidad de la Similaridad

Dado un usuario u, sus vecinos y un ítem i que se quiere ofrecer a u,

interesa predecir el *rating* que éste daría a i, para determinar si es conveniente proponérselo. La forma de proceder es, comenzando por normalizar la matriz de utilidad, tomar los k vecinos más similares a u, promediar sus *ratings* (a los cuales se les restó el *rating* promedio al normalizar la matriz) sobre el ítem i, y, luego, sumar dicho valor al *rating* promedio de u, sobre todos los ítems. Con la normalización se corrige el problema de que a cualquiera de los k usuarios, o el propio u, califiquen a la mayor parte de los ítems, muy alto o muy bajo.

En forma dual [4] se puede utilizar la similaridad entre ítems, para estimar la entrada de (u, i) en la matriz. Para ello, se debe encontrar los m ítems (m es un número arbitrario) más similares a i y tomar el promedio de los *ratings* otorgados por u sobre los esos ítems. Se considerarán sólo los ítems que han sido calificados por u, siendo conveniente normalizar la matriz.

Balance entre utilizar similaridad entre ítems o entre usuarios

Se puede establecer un *trade off* respecto a la dualidad de la similaridad al momento de ser utilizada.

Para usuarios similares, se ejecuta el proceso una sola vez para el usuario u y se estiman todos los lugares en blanco de su fila, en la matriz de utilidad (en caso de que no haya valores de los k vecinos, el blanco se llena con el promedio de u). Si se trabaja con ítems similares, se deberá calcular la similaridad entre todos los ítems antes de calcular la fila para u.

Por otro lado, la similaridad entre ítems proporciona información más confiable, por el hecho de que es más fácil encontrar ítems relacionados con el mismo género para usuarios que gustan sólo de ítems de una sola variedad.

Cualquiera sea el enfoque a adoptar antes de tomar una decisión, se puede calcular los ítems preferidos para cada usuario ya que la matriz de utilidad cambia lentamente.

Filtrado colaborativo usuario–usuario

Como se había planteado en el filtrado colaborativo, dado un usuario $u \in U$, sus vecinos N y un ítem i que se quiere ofrecer a u, interesa predecir el *rating* que daría u a i. Para generar esta predicción, se define una función $s : U\text{x}U \rightarrow R$ la cual, inicialmente, es utilizada para calcular un vecindario N de vecinos de u, con $N \subseteq U$. Una vez determinado N, el sistema combina los *ratings* de sus usuarios y genera una predicción de las preferencias de u sobre

un ítem i, generalmente calculando el promedio de los pesos de los *ratings* de los usuarios de N sobre el ítem i, Ec. (9.21).

$$p_{u,i} = \bar{r}_u + \frac{\sum_{u^1 \in N} s(u, u^1)(r_{u^1,i} - \bar{r}_{u^1})}{\sum_{u^1 \in N} |s(u, u^1)|} \tag{9.21}$$

Operando de la misma forma que en los *SR* basados en contenidos, se puede restar la media de los *ratings* $\bar{r}_u$ compensando las diferencias del uso de las escalas de *ratings* por parte de los usuarios y con algunos cambios adicionales se puede normalizar a $z - score$ [52], obteniéndose la Ec. (9.22).

$$p_{u,i} = \bar{r_u} + \sigma_u \frac{(\sum_{u^1 \in N} s(u, u^1)(r_{u^1,i} - \bar{r}_{u^1}))/\sigma_{u^1}}{\sum_{u^1 \in N} |s(u, u^1)|} \tag{9.22}$$

σ_u: Es la desviación estándar de los *ratings* del usuario u.

El promedio de los pesos es el más utilizado y, por regla general, funciona correctamente. Además es consistente con la *Teoría de la Elección Social* al ser construida sobre la base de modelos de conducta humana. Esta teoría trata de las preferencias del individuo y de la sociedad como un todo. En este marco *Pennock et al.* [54] demuestran que el promedio de los pesos es el único método de agregación que produce resultados consistentes.

A los efectos de determinar el número de vecinos para hacer los cálculos, algunos investigadores sostienen que se pueden utilizar todos los vecinos, lo que tiene como inconveniente el costo computacional y la entrada de ruido. Por lo tanto, se puede limitar el número de vecinos a ser utilizados, considerando aquellos «más parecidos» y eliminando los que tienen baja correlación y producen ruido. No obstante, el número de vecinos debe ser determinado, de acuerdo al contexto del problema. Algunos autores *(Harlocket et al.)*, sugieren de 20 a 50, como un número de partida consistente. *Lathia et al.* [55] proponen una adaptación dinámica, al número de vecinos, de tal forma que minimicen el error [55].

Función de similaridad

Un elemento crítico en el filtrado colaborativo usuario–usuario es la determinación de la función de similaridad. A continuación se presentan algunas de ellas.

Correlación de Pearson

La correlación de Pearson se expresa en la Ec. (9.23) y es calculada entre

dos *ratings* de usuarios.

$$s(u,v) = \frac{\sum_{i \in I_u \cap I_v}(r_{u,i} - \bar{r_u})(r_{v,i} - \bar{r_v})}{\sqrt{\sum_{i \in I_u \cap I_v}(r_{u,i} - \bar{r_u})^2}\sqrt{\sum_{i \in I_u \cap I_v}(r_{v,i} - \bar{r_v})^2}} \tag{9.23}$$

I_u representa la cantidad de ítems calificados por el usuario u.

Al evaluarse esta correlación entre dos usuarios con pocos *ratings* en común, el *SR* opera limitadamente, subsanándose al considerar un límite inferior del número de *ratings* en común, para lograr la correlación. Por debajo de dicho umbral, la similaridad se escala hacia abajo [52].

Costo computacional

Este sistema tiene problemas de escalabilidad conforme la cantidad de usuarios crece. Es de orden $O(|U|)$ o mayor. Si n es la cantidad de usuarios y m la de ítems, el cálculo de la correlación entre dos usuarios es $O(n)$. El orden para calcular todas las correlaciones para un usuario es $O(mn)$. Finalmente, si se quisieran hallar todas las correlaciones para todos los posibles pares, el orden es $O(m^2.n)$.

Filtrado colaborativo ítem – ítem

A los efectos comerciales, se utiliza el enfoque basado en ítems (filtrado colaborativo ítem-ítem), que es menos costoso computacionalmente. Utiliza la similaridad entre los patrones de *ratings* de los ítems. Si dos ítems tienen iguales calificaciones por parte de los usuarios, entonces se pueden considerar similares. Notar que es parecido al enfoque de *SR* por filtrado de contenido, pero ahora la similaridad de los ítems es deducida de los patrones de preferencia en lugar de los atributos de los ítems.

Este abordaje es interesante cuando la cantidad de usuarios es mucho mayor que los ítems (por ejemplo *Netflix*). El hecho de que un usuario agregue o actualice información, es muy difícil que afecte significativamente la similaridad entre ítems, especialmente si el ítem tiene muchos *ratings*. Por lo tanto, este hecho puede ser utilizado para pre-computar las similaridades entre ítems.

9.4. Análisis de sentimientos

El análisis de sentimientos está adquiriendo cada vez mayor importancia para ayudar a las empresas y a los individuos, a reaccionar de manera oportuna a los incidentes que afecten a su negocio y reputación. Se trata de una rama de *ML* y, técnicamente, se basa en el uso de técnicas de clasificación de texto. Por ejemplo, en la *Sección 5.3.1, Clasificador lineal*, se presentaron las técnicas de clasificación, donde el ejemplo base utilizado era un clasificador para analizar los sentimientos originados por un curso recibido por un estudiante.

9.4.1. Utilización de ML para análisis de sentimientos

Una clasificación de sentimientos basada en aprendizaje supervisado, consiste en la aplicación del método de clasificación aplicada al análisis de sentimientos.

Se considera un conjunto de datos (comentarios) manualmente etiquetados en positivos y negativos (etapa de entrenamiento). Se utiliza el test denominado ratio de la verosimilitud (*Likelihood Ratio*). Por medio de este *test*, se pretende determinar si un nuevo comentario es positivo o negativo.

Como ejemplo base, se supone que se reciben los *tweets*, referentes a un curso en línea, representados en la Tabla 9.2.

Total	**Opinión**	**Sentimiento**
200	Me **gustó** el curso **mucho**	Positivo
100	El curso resultó **aburrido**	Negativo
200	Resultó **interesante**	Positivo
100	El curso es altamente **recomendable**	Positivo
200	El moderador es **ineficiente**	Negativo
100	Resultó tan **interesante** como para no ir **nunca** más	Negativo
300	El curso no fue para nada **aburrido**	Positivo

Tabla 9.2: Ejemplo de opiniones recibidas

Se puede calcular la razón de comentarios positivos respecto al total y la misma razón de los comentarios negativos.

$$P(+) = \frac{comentarios\ positivos}{total\ comentarios\ (calculados\ a\ priori)} = 800/1200 = 0,667$$

$$P(-) = \frac{comentarios\ negativos}{total\ comentarios\ (calculados\ a\ priori)} = 400/1200 = 0,333$$

Posteriormente, se computa la verosimilitud para cada una de las palabras resaltadas de la Tabla 9.2, que se presenta en la Tabla 9.3.

P(gustó |+)=200/800	0,25
P(mucho |+)=200/800	0,25
P(aburrido |+)=300/800	0,375
P(interesante |+)=200/800	0,25
P(recomendable |+)=100/800	0,125
P(ineficiente |+)=1/800	0,00125
P(nunca |+)=1/800	0,00125
P(gustó |-)=1/400	0,0025
P(mucho |-)=1/400	0,0025
P(aburrido |-)=100/400	0,25
P(interesante |-)=100/400	0,25
P(recomendable |-)=1/400	0,0025
P(ineficiente |-)=200/400	0,50
P(nunca |-)=100/400	0,25

Tabla 9.3: Cálculo de las verosimilitudes

Se puede observar que no existen comentarios positivos que incluyan la palabra «ineficiente», sin embargo, no se puede ingresar cero, pues la fórmula perdería sentido. Por lo tanto, se reemplaza por un pequeño número como uno, que es el más próximo a cero. A esto se le denomina *smoothing* y es importante, pues se incluye la probabilidad de la verosimilitud.

Se busca clasificar una nueva opinión sobre el curso y determinar si es positiva o negativa. Si la frase que se quiere clasificar es: «El curso estuvo recomendable e interesante». Las palabras «recomendable» e «interesante» ya existen en el «corpus» del ejemplo.

El método consiste en calcular la probabilidad de que con esas palabras sea un comentario positivo y dividirlo entre la probabilidad de que con dichas palabras sea un comentario negativo. Si el valor es mayor que uno se decide que es un comentario positivo y viceversa. Para el ejemplo, no se dispone de algunas palabras, como por ejemplo, «nunca». En este caso, en lugar de utilizar $P(nunca|+)$, se utiliza $1 - P(nunca|-)$. De igual manera se procede con las palabras ausentes en la frase tanto para comentarios positivos o negativos.

$$
\begin{aligned}
A \quad &= P(recomendable|+)P(interesante|+)(1-P(gustó|+)) \\
&(1-P(mucho|+))(1-P(aburrido|+)) \\
&(1-P(ineficiente|+))(1-P(nunca|+)) \\
&= 0,125.0,25.0,75.0,75.0,625.0,99875.0,99875 = 0,011
\end{aligned}
$$

$$
\begin{aligned}
B \quad &= P(recomendable|-)P(interesante|-)(1-P(gustó|-)) \\
&(1-P(mucho|-))(1-P(aburrido|-)) \\
&(1-P(ineficiente|-))(1-P(nunca|-)) \\
&= 0,0025.0,25.0,9975.0,9975.0,75.0,5.0,75 = 0,00017
\end{aligned}
$$

$Ratio = A/B > 0$, por lo tanto es un comentario positivo.

9.5. Redes Sociales

En esta sección se analizan técnicas para descubrir comunidades.

- Algoritmo de *Girvan-Newman*, el cual se puede aplicar para descubrir comunidades. pero sin miembros en común.
- Descubrimiento directo de comunidades (*Ítemsets* Frecuentes).
- Búsqueda de comunidades por partición de grafos.

Asimismo, se presentan métodos utilizados para analizar el solapamiento de comunidades.

- Búsqueda de comunidades que se superponen.
- Conteo de triángulos en redes sociales.

9.5.1. Búsqueda de Comunidades

A continuación se analizan los algoritmos de *Girvan-Newman*, de descubrimiento directo de comunidades y búsqueda de comunidades por partición de grafos.

Algoritmo de *Girvan-Newman*

Es un algoritmo que permite calcular el número de caminos más cortos que pasan por cada arista en un grafo. Este método visita cada uno de los nodos y cuenta el número de caminos más cortos a los otros nodos. Los grafos no son dirigidos ni ponderados. El algoritmo busca determinar los caminos más cortos entre todos los nodos visitándolos una vez. Se basa en efectuar una recorrida en amplitud del grafo (*BFS*), visitando cada nodo X una sola vez y calcula el número de caminos más cortos desde X a todos los nodos que pasan por sus aristas. Considerando que cada nivel de la recorrida es la longitud del camino más corto, se deduce que las aristas definidas por los nodos que están en dicho nivel, no pueden ser parte del camino más corto del nodo X.

Las aristas *DAG* (grafo dirigido acíclico), se definen a partir de una recorrida *BFS* en un grafo, como una arista entre niveles.

Si el vértice Y está un nivel más arriba que X (más próximo al vértice donde se inicia el *BFS*), al vértice Y se le llama padre de X y al nodo X, hijo de Y. Un hijo puede tener más de un padre.

El algoritmo comienza con un recorrido *BFS*. El nivel de cada nodo, desde un vértice X, en un recorrido *BFS*, es el camino más corto entre ellos. Por lo tanto, las aristas entre nodos en el mismo nivel no pueden ser parte del camino más corto.

Pasos

1. A partir de un nodo X se hace un *BFS* (desafectándose aristas que no están en el recorrido).
2. Se etiqueta cada nodo por el número de caminos más cortos que le llegan desde la raíz.
3. Para cada arista a, calcular la suma de todos los caminos más cortos desde el nodo X hasta el nodo Y que contienen a la arista a.

Este se hace para todos los nodos del grafo.

Criterio para la suma

- Cada hoja de un *DAG* (hoja es un nodo sin *DAG* en el siguiente nivel) lleva un crédito.

- Si el nodo no es hoja, lleva uno, más la suma de los créditos de las aristas DAG desde el nodo del nivel inferior.
- Sea una arista que es DAG que comienza en el vértice Z, sean sus padres $(Y_1, Y_2, ..., Y_k)$. Sean P_i el número de caminos más cortos desde la raíz hasta Y_i (este número fue el calculado en el paso 2). Si P_i es la cantidad de caminos más cortos desde la raíz a Y_i, el crédito para la arista (Y_i, Z) es:

$$\frac{Z.P_i}{\sum_{i=j}^{k} P_j} \tag{9.24}$$

Luego de calcular los créditos de cada nodo como si fueran la raíz, se suman los créditos de cada arista.

Ejemplo 9.4

Presentamos un ejemplo del algoritmo de *Girvan-Newman (GN)* aplicado al grafo de la Figura 9.5

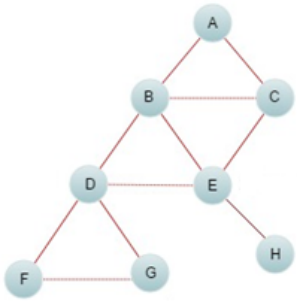

Figura 9.5: Algoritmo de *Girvan-Newman*

Iniciando el BFS desde A, no se consideran para el cálculo las aristas: BC, DE *y* FG.

En la Figura 9.6 se desafectan las aristas y se aplica el primer y segundo paso de algoritmo.

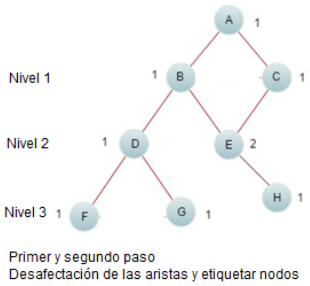

Figura 9.6: Algoritmo de *Girvan-Newman* (primera parte)

Primer paso del algoritmo GN: recorrido en BFS y determinación DAG.

Segundo paso de GN: etiquetar los nodos de acuerdo al número de caminos desde los padres.

Tercer paso de GN: cálculo sobre las aristas y nodos. Se puede visualizar en la Figura 9.7.

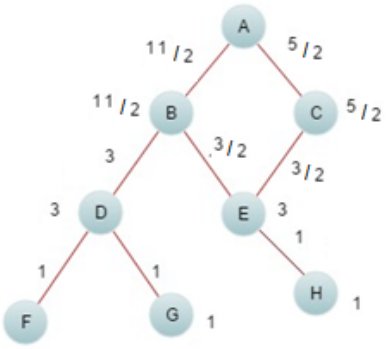

Figura 9.7: Resultado del Algoritmo de *Girvan-Newman*

En el nivel 3, los vértices F, G, H tienen crédito 1 por ser hojas y por tal motivo transfieren dicho crédito a sus aristas. Las aristas $(D, F), (D, G)$ y (E, H) tienen crédito 1.

En el nivel 2, el nodo D, que no es hoja, lleva crédito 1 más el crédito de las aristas de los DAG entrando por abajo. Notar que ese valor encierra la cantidad de caminos más cortos desde A pasando por D, F y G (que son 3).

$$Crédito\ para\ D\ =\ 1+1+1$$

De la misma forma se calcula para el nodo E.

$$Crédito\ para\ E\ =\ 2+1$$

En el nivel 1, comenzando por el nodo B, el nodo D tiene un padre, por lo que le transfiere su crédito a la arista (B, D), es decir 3. Pero el nodo E tiene dos padres, por lo que debe dividir su crédito entre las aristas (E, B) y (E, C) proporcionalmente a los valores en los nodos B y C. Debido a que cada uno de los vértices tiene uno, la división es en partes iguales (si los valores de B y C fuesen respectivamente 2 y 3, se dividiría en 2/5 del crédito para B y 3/5 para C). Para el ejemplo, E tiene crédito 3 por lo tanto las aristas (E, B) y (E, C) llevan 3/2. El crédito para B es 1 más el que entra por abajo a través de las aristas: 3 y 3/2.

Crédito para $B = 1 + 3 + 3/2 = 11/2$

Análogamente se calcula para el nodo C.

Crédito para $C \ = \ 1 + 3/2 = 5/2$

Las aristas (A, B) y (A, C) llevan el crédito de los nodos B y C respectivamente.

Para completar el cálculo de la intermediación se repite el cálculo para cada nodo considerado como la raíz.

Al repetirse cada camino dos veces, se divide entre 2.

El valor de cada arista es su contribución a la intermediación a los caminos más cortos desde A. Se comprueba que la arista (A, B) es la de mayor intermediación.

Búsqueda de comunidades usando la intermediación

Se pueden formar clústeres eliminando las aristas con mayor intermediación hasta que el grafo tenga un número adecuado de componentes.

La Figura 9.8 pertenece a un ejemplo del libro *Mining of Massive Datasets, Jure Leskovec, Anand Rajaraman, Jeffrey D. Ullman* [4] y analiza un grafo que tiene calculadas las intermediaciones de cada arista.

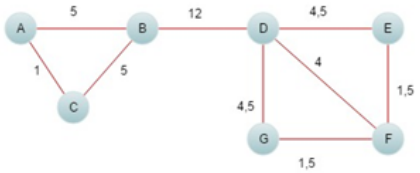

Figura 9.8: Búsqueda de comunidades

La arista (B, D) tiene la intermediación más alta. Luego lo siguen $(A, B), (B, C)$ y finalmente (D, G) y (D, F).

De la eliminación de dichas aristas, se forman cuatro clústeres:

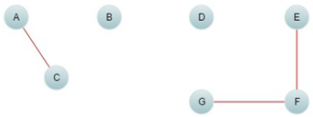

Figura 9.9: Comunidades detectadas

A *y* C están más próximas y B puede ser considerado como un «traidor» a la comunidad $\{A, B, C\}$. En la misma situación se encuentra el nodo D respecto a $\{D, E, F, G\}$ (Figura 9.9).

El orden del algoritmo GN para un grupo de n nodos y a aristas es: $O(na)$ [4].

El orden para n muy grande es computacionalmente costoso. Por tal motivo, se acostumbra tomar un subconjunto de nodos (aleatoriamente), los cuales son utilizados como raíces para conducir el BFS.

Descubrimiento directo de comunidades (Ítemsets Frecuentes)

Se estudiará una técnica para descubrir comunidades, observando los subconjuntos de nodos que tienen un número grande de aristas en común (este requerimiento se encuentra relacionado con la búsqueda de ítemsets frecuentes, que fue estudiada en la *Sección 6.3, Algoritmos de recuperación de ítems*).

No se puede garantizar que un grafo con muchas aristas, tenga un clique (grafo completo) grande, pero sí se podría afirmar que, si existe un grafo bipartito con muchas aristas, se puede obtener un grafo bipartito completo grande [4]. La demostración se encuentra más adelante en esta sección.

La importancia de este tipo de grafos es que tanto un grafo bipartito grande, como un clique grande, pueden ser considerados como los núcleos de las comunidades y, a partir de ellos, agregar nuevos nodos.

Si los nodos son iguales (del mismo tipo, por ejemplo, usuarios), se pueden agrupar los nodos aleatoriamente en dos conjuntos y considerar que si la comunidad existe, la mitad de esa comunidad cae en un conjunto y la otra mitad en otro. En este caso, se puede esperar que la mitad de las aristas vayan de un conjunto a otro.

Búsqueda de sub-grafos bipartitos completos

Sea G un grafo bipartito, se pretende hallar instancias de $K_{s,t}$ en G. Esto se puede ver como una búsqueda de ítemsets frecuentes. Sean los ítems nodos del lado izquierdo de G. La instancia $K_{s,t}$ que se busca, tiene t nodos a la izquierda (ítems) y s nodos a la derecha (cestas).

Los miembros de un canasto son los nodos del lado izquierdo conectados a él. Sea s el umbral de cobertura (cantidad de nodos que la instancia Ks,t tiene del lado derecho).

El problema de encontrar instancias de $K_{s,t}$ desde la óptica de búsqueda de ítemset frecuentes F, de tamaño t, se puede formular: «Si un conjunto de t nodos del lado izquierdo es frecuente, entonces deben estar todos ellos en, por lo menos, s canastos».

Notar que los canastos son todos los nodos del lado derecho.

Ejemplo 9.5

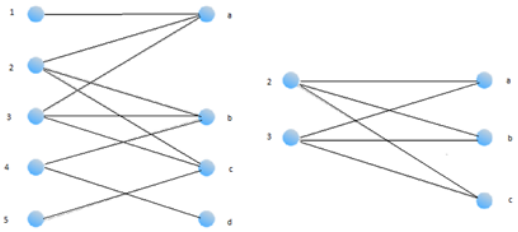

Figura 9.10: Instancia de grafo bipartito $K_{3,2}$

Sea el grafo de la Figura 9.10 donde se considera $s = 3$ y $t = 2$. Significa que existen instancias de G tal que $k_{3,2}$, concretamente, dos ítems del lado izquierdo, estén al menos en tres canastos.

El canasto a tiene $3, 2$.
El canasto b tiene $2, 3$.
El canasto c tiene $2, 3$.

El grafo es bipartito completo.

Demostración de que un grafo bipartito G con suficientes aristas, tiene al menos una instancia de Ks, t

Sea n, el número de nodos del lado izquierdo de G y del lado derecho (se considera la misma cantidad, n, en ambos conjuntos, lo cual no limita

la demostración). Sea d el grado promedio de todos los nodos y se intenta contar el número de ítemsets frecuentes de tamaño t que tiene un cesto con d ítems. Dicho número obtenido del lado derecho, da el total de subconjuntos del tamaño t. Si este número se divide entre $\binom{n}{t}$, se tiene la frecuencia promedio de todos los ítemsets de tamaño t. Por lo menos uno debe tener dicha frecuencia. Si ese promedio es s, la instancia Ks,t existe.

Cálculos

Suponer que $d(v_i) = d_i$ siendo v_i un nodo de la derecha (canasto i). Este canasto contribuye con $\binom{d_i}{t}$ ítemsets de tamaño t. La contribución total de los n nodos de la derecha está dado en (9.25):

$$\sum_i \binom{d_i}{t} \tag{9.25}$$

Esta suma tiene un mínimo si $d_i = d$ (sin demostración) [4].

Se sustituye d_i por d en (9.25) (si la cantidad de ítemsets frecuentes es menor, la demostración no es afectada).

La contribución total de los n nodos de la derecha para contabilizar los ítemsets de tamaño t es: $n\binom{d}{t}$. El número de ítemsets de tamaño t es $n\binom{d}{t}$. Entonces el promedio del conteo de ítemsets de tamaño t es $n\frac{\binom{d}{t}}{\binom{n}{t}}$.

Esta expresión tiene que ser al menos s si existiera una instancia $K_{s,t}$.

Haciendo operaciones se obtiene:

$n(d)(d-1)...(d-1+t)/(n(n-1)...(n-t+1))$

Suponiendo que $n >> d >> t$, se obtiene:

$d(d-1)...(d-t+1) \approx d^t$
$n(n-1)...(n-t+1) \approx n^t$

Por lo tanto, es necesario que $n(d/n)^t \geq s$

Si hay una comunidad con n nodos de cada lado, con un grado promedio d y $n(d/n)^t \geq s$, esta comunidad tiene garantizada la existencia de una instancia $K_{s,t}$. Por lo tanto, se pueden utilizar los métodos de ítemsets frecuentes para hallarlos en grafos que incluyen a otros (*A Priori*, etc.). Es decir si hay una instancia $K_{s,t}$ en un grafo, debe haber por lo menos otra en otro grafo que lo incluya.

Búsqueda de comunidades por partición de grafos

Se trata de un nuevo enfoque donde se intenta obtener comunidades a través de cortes adecuados del grafo (debe existir un balance entre el tamaño del corte y la diferencia en los tamaños de los conjuntos, creados con el corte).

Identificación de buenos clústeres

La arista de corte de un grafo es un corte imaginario que define dos conjuntos A y B en el conjunto de nodos (tener en cuenta que, usualmente, en teoría de grafos, una arista de corte de un grafo conexo es la que define componentes conexas, si es eliminada).

El objetivo del corte es dividir los nodos en dos conjuntos, de manera tal que las aristas que unen nodos de diferentes conjuntos, sea mínima. Otra condición necesaria, es que los dos conjuntos sean aproximadamente de igual tamaño.

Se considera que un «buen» corte debe balancear el tamaño del corte en sí mismo y el tamaño de los conjuntos que se definen luego del corte. El denominado corte normalizado genera «buenos» cortes. Para definirlo se necesita disponer, previamente, del concepto de volumen.

Matriz laplaciana

Sea A la matriz de adyacencia de un grafo $K_{s,t}$ y D la matriz de los grados de G (esta matriz es diagonal con el grado de cada uno de los vértices).

La matriz laplaciana es una matriz $n \times n$, que se obtiene de la diferencia de la matriz diagonal menos la matriz de adyacencia ($L = D - A$).

Los valores y vectores propios de la matriz L proporcionan una idea para efectuar la partición del grafo. Se demuestra que esta información es proporcionada por los menores valores propios y sus auto-vectores.

El menor valor propio de L es O y su correspondiente vector propio es $[1, 1, ..., 1] = \mathbf{1}$

Se demuestra que $L.\mathbf{1} = 0$ porque si se considera la fila i de L, tiene d en su diagonal y d ocurrencias de –1 en el resto (los restantes elementos son cero). Entonces, para todos los elementos del vector resultante del producto $L1$ se verifica $d + (-1)d = 0$.

Finalmente se concluye que $L.\mathbf{1} = 0.\mathbf{1}$, por lo que el cero es un valor propio.

Al ser la matriz L simétrica, verifica que el segundo valor propio mínimo de L es el mismo que se obtiene de hallar el valor mínimo de $\mathbf{x}^T L\mathbf{x}$ [4], siendo

x un vector columna de n componentes, tomándose dicho mínimo bajo las siguientes restricciones:

- La longitud de x es 1, $\sum_{i=1}^{n} x_i^2 = 1$
- El vector x es ortogonal al vector propio asociado al menor valor propio.

Aplicando el resultado anterior a una matriz laplaciana L, y recordando que 1 es el vector propio asociado al menor valor propio, se obtiene 9.26.

$$x^T.\mathbf{1} = \sum_{i=1}^{n} x_i = 0 \tag{9.26}$$

Además, se verifica que:

$$x^T Lx = x^T(D - A)x = x^T Dx - x^T Ax \tag{9.27}$$

Considerando que $Dx = [d_1x_1, d_2x_2, ..., d_nx_n]$, se obtiene (9.28).

$$x^T Dx = \sum_{i=1}^{n} d_i x_i^2 \tag{9.28}$$

En el término $x^T Ax$, la componente i del vector columna Ax es la suma sobre j de los x_j $(j = 1 \ldots .n)$, tales que hay una arista (i, j) en el grafo. Por lo tanto $-x^T Ax$ es la suma $-2x_ix_j$ de todos los pares de puntos i, j tales que hay una arista entre ellos (el factor 2 es debido a que cada par i, j corresponden a dos términos: $-x_ix_j - x_jx_i$).

Juntando el resultado anterior con la Ec. (9.28), se puede agrupar los términos de la Ec. (9.27). Para cada par i, j de $-x^T Ax$ se tiene el término $-2x_ix_j$.

De la Ec. (9.28) se distribuyen los términos $d_ix_i^2$ en todos los pares que tienen al nodo i. En resumen, se puede asociar cada par i, j que forma una arista con la expresión $x_i^2 - 2x_ix_j + x_j^2 = (x_i - x_j)^2$.

Como resultado, $x^T Lx$ es igual a la suma sobre todas las aristas del grafo i, j de $(x_i - x_j)^2$.

Entonces, el segundo VAP menor es el que minimiza $(x_i - x_j)^2$ con las siguientes restricciones:

- $\sum_{i=1}^{n} x_i^2 = 1$ (lo que implica que no todos los componentes del vector pueden ser cero).
- El vector propio es ortogonal a **1** (de la Ec. (9.26)).

Debido a estas dos restricciones, dicho vector propio tendrá componentes positivas y negativas.

Teniendo en consideración el problema de la partición del grafo, una opción es dividir al conjunto de nodos en dos conjuntos: uno cuyos vectores correspondientes tienen componentes positivas y otros con las componentes negativas, cuyos tamaños son probablemente parecidos.

A su vez, $(x_i - x_j)^2$ es menor si x_i y x_j son de igual signo (pues de distinto signo se suman los valores y el cuadrado puede ser grande). Por lo tanto minimizar $x^T Lx$ bajo las restricciones nombradas, tiende a que x_i y x_j tengan el mismo signo, si existe la arista (i, j).

Este hecho permite identificar un corte en el grafo, separando los nodos en dos grupos, en base al signo: los que tienen la componente negativa pertenecen a un grupo y los positivos al otro.

El método anterior es llamado método espectral para particionar grafos.

9.5.2. Solapamiento de comunidades

Se analizarán los fundamentos para la búsqueda de comunidades que se superponen y la técnica para el conteo de triángulos.

Hasta ahora se había considerado que una persona pertenecía sólo a una comunidad. Sin embargo, normalmente son miembros de más de una, por lo que se presentan nuevos enfoques para este problema.

Al tratarse de dos comunidades, se espera que dentro de una comunidad, haya una densidad alta de aristas, pero en la intersección deberá ser más alta aún, ya que son individuos que se conocen entre sí de las dos comunidades y es más probable que estén relacionados en el grafo.

En forma análoga, si se consideraran tres comunidades, en la intersección común a las tres, es de esperar que haya mayor densidad.

Modelo de grafo de afiliación

Se puede aplicar el *EMV* para determinar alguna característica del grafo. Suponer que se dispone de un modelo y se busca en un grafo, un subgrafo de nodos relacionados entre sí (ser amigos, por ejemplo).

El modelo de grafo de afiliación puede emplearse para detectar grafos sociales sobre comunidades. Para ello, se consideran parámetros que determinan

la probabilidad de que un subgrafo esté formado por amigos. Se puede asumir que el valor de los parámetros que maximizan dicha probabilidad (EMV) aplicada a una instancia particular, proporciona el modelo correcto.

Se asume:

- Hay un número de comunidades y de individuos, ambos dados.
- Cada comunidad puede tener cualquier membrecía (cantidad de miembros). La membrecía es un parámetro del modelo.
- Cada comunidad C tiene una probabilidad p_c de que un par de miembros de la comunidad estén relacionados entre sí, a través de una arista dado que pertenece a C. Esta probabilidad es otro parámetro del modelo.
- Si un par de nodos están en dos o más comunidades distintas y no tienen arista común de acuerdo con la regla número 3, existe una arista entre ellos.

Se busca calcular la probabilidad de que un grafo con un determinado número de nodos sea generado por el mecanismo precedente.

Para ello, se considera a dos individuos u y v que pertenecen a las comunidades C y D. La probabilidad de que no haya arista entre ellos es el producto de las probabilidades de que no hay arista en ninguna de las dos comunidades.

La probabilidad de que no haya arista entre u y v en la comunidad C es: $(1 - p_c)$

Generalizando para cualquier cantidad M de comunidades, la probabilidad de que no exista tal arista es: $\prod_{C\ en\ M}(1 - p_C)$ Por lo tanto la probabilidad de que haya arista es:

$$p_{uv} = 1 - \prod_{C\ en\ M} (1 - p_C) \tag{9.29}$$

En el caso en que u y v no estén en ninguna comunidad, la probabilidad que se le asigna es un valor ϵ muy pequeño, si no, no se podría nunca asignar una probabilidad distinta de cero a ningún conjunto de comunidades que no tengan cada par de individuos compartiendo una comunidad.

Si se tiene el conocimiento de los nodos que están en determinada comunidad, se puede calcular la probabilidad de un grafo dado, por intermedio de la probabilidad de las aristas [4].

Sea M_{uv}, el conjunto de comunidades a los cuales u y v pertenecen.

Si E es la probabilidad de ser las aristas observadas en el grafo, se obtiene la Ec. (9.30).

$$\prod_{(u,v)\ en\ E} p_{uv} \prod_{(u,v)\ no\ en\ E} (1 - p_{uv}) \tag{9.30}$$

El procedimiento se debe continuar a través de la asignación de miembros a las comunidades contemplando que el EMV para tal distribución es la mayor para todas las asignaciones posibles.

Una posible técnica para llevarlo a cabo puede ser el algoritmo de gradiente descendente. Su utilización no es inmediata ya que sería necesario estudiar cómo se realiza la incorporación de nodos a la composición de comunidades, la cual se realizaría en pasos discretos (a diferencia de utilizar una función continua, como, habitualmente, el método emplea). Una forma de llevarlo a cabo, es comenzando con una configuración inicial y hacer pequeños cambios, como una inserción o borrado de un individuo en una comunidad. Para cada configuración, se estima la mejor p_c para todas las comunidades por gradiente descendente. No se puede asegurar la convergencia para este tipo de solución [4].

Conteo de triángulos

La búsqueda de triángulos en una red social se justifica por el hecho de que si el nodo u es amigo de v, y v del w, la probabilidad de que u y w sean amigos es más alta que el promedio. Por lo tanto, en una red social con individuos relacionados entre sí, es más probable encontrar triángulos, que en un grafo aleatorio y, de esta forma, tener una medida para determinar si un grafo es una red social.

Se ha demostrado que es posible medir la edad de una comunidad en base a los triángulos que presenta, basándose en un argumento similar al del párrafo anterior de los vértices *u, v* y *w*.

Algoritmo para encontrar triángulos

Se considera un grafo con n nodos y m aristas, con $m \geq n$. Se asume que los nodos son enteros: 1,2, ...

Se define un nodo peso pesado como un nodo cuyo grado es, al menos, $\sqrt{m}$.

Un triángulo de nodos de pesos pesados, es aquel cuyos vértices son nodos de pesos pesados.

Notar que no puede haber más de $2\sqrt{m}$ nodos pesos pesados (si los hubiera, se obtendría, como suma de los grados, más de $2\sqrt{m}\sqrt{m} = 2m$, y al contabilizarse doble cada arista, quedaría un número mayor que m de aristas).

Asumiendo que el grafo se encuentra representado por sus aristas, se realiza un pre procesamiento del grafo.

- Calcular el grado de cada nodo. Para ello se examina cada arista y esto tiene un orden $O(m)$.
- Crear un índice con la clave formada con un par de nodos, que forman cada arista, lo cual permite saber, dados dos nodos, si existe una arista entre ellos. Su construcción es de orden $O(m)$. Con una tabla de *hash* es suficiente para almacenar esta información. El tiempo esperado de ejecución para contestar una consulta sobre la existencia de una arista es constante. Desde una óptica de cómo se ejecuta, este algoritmo es óptimo. No se dice nada con respecto al peor caso. No obstante, en un número grande de ítems, existe una alta probabilidad de comportarse como se esperaría [4].
- Crear otro índice de aristas, cuya clave es igual a uno de los nodos, de manera tal que dado un nodo, se puedan recuperar sus nodos vecinos, en un tiempo proporcional al número de esos nodos. Se puede utilizar una tabla de *hash*.

Se ordenan los nodos por su grado de tal forma que dados los nodos u y v:
v es anterior a u si y solo si:
$deg(v) < deg(u)$.
ó, si $deg(v)$ es igual a $deg(u)$ y $v < u$.

Operación para los triángulos de pesos pesados

Teniendo en cuenta que el número de nodos pesos pesados es de $O(\sqrt{m})$, la cantidad de ellos debe ser de orden $O(m^{3/2})$. Esto es debido a que el total de ellos son $\binom{\sqrt{m}}{3}$. Finalmente, buscando en el índice de aristas, se puede averiguar si existen las tres aristas en un tiempo $O(1)$, por lo que, el tiempo computacional para encontrar todos estos tipos de triángulos es de $O(m^{3/2})$.

Operación para los otros triángulos

Si dos vértices v_1 y v_2 son pesos pesados, se ignora esta arista.

Si v_1 no es peso pesado, v_2 es cualquiera, siendo v_1 anterior a v_2 y se consideran $u_1, ..., u_k$ los nodos adyacentes a v_1, se verifica que $k < \sqrt{m}$. Se pueden encontrar esos nodos, a través del tercer índice en un tiempo $O(k)$, cuyo peor caso es cuando dicho vecino es un peso pesado quedando $O(\sqrt{m})$.

Para cada u_i se puede verificar mediante el primer índice si existe arista (u_i, v_2), en un tiempo $O(1)$. Asimismo, se puede determinar el grado de u_i en un tiempo $O(1)$ porque se contaron todos los grados de los nodos en el paso 1. Se contabiliza el triángulo $\{v_1, v_2, u_i\}$ si y solo si, existe la arista $(ui, v2)$ y v_1 es anterior a u_i. De esta manera, se previene de contar más de una vez un triángulo. Por lo tanto, el tiempo para procesar los nodos adyacentes a v_1 es $O(\sqrt{m})$. Como hay m aristas, el tiempo de procesamiento es $O(m^{3/2})$.

Por lo tanto, se tiene un costo de $O(m)$ en preprocesar, un costo de $O(m^{3/2})$ para procesar los pesos pesados y el mismo orden para los otros triángulos, resultando en que el orden del algoritmo es $O(m^{3/2})$.

Este algoritmo es óptimo ya que si se considera un grafo completo, con n nodos, la cantidad total de triángulos es $\binom{n}{3}$. Al no poderse enumerar esos triángulos en menos tiempo que el número de triángulos, se tiene que en dicho grafo cualquier algoritmo verifica un tiempo $\Omega(n^3)$. Sin embargo, la cantidad de aristas m es de orden $O(n^2)$, por lo que se verifica que el tiempo en el grafo es $\Omega(m^{3/2})$.

Si en lugar de considerar un grafo completo, se compara un grafo *sparse*, se podría adicionar al grafo anterior una cadena de nodos (incluyendo sus aristas) de cualquier longitud, hasta n^2 nodos. Esta cadena no agrega más triángulos y duplica el número de aristas $(n^2 - n)$. Con este procedimiento se pueden agregar todos los nodos que se quieran y la razón nodos sobre aristas se aproxima a uno. Pero como sigue existiendo el mismo $\Omega(m^{3/2})$ triángulos, que en el caso anterior, este límite se mantiene, cualquiera sea el ratio m/n.

9.5.3. Empleo de *MapReduce*

Para grandes grafos, es posible dividir las tareas con *MapReduce*. Para ello, se puede hacer un *join* triple (en el *Capítulo 2* se presentaron implementaciones de las operaciones del álgebra relacional con *MapReduce*), donde cada arista es enviada a un número de *reducers* proporcional a la raíz cúbica del número total de *reducers* [4].

Considerar que los nodos están numerados: $1, ..., n$ y siendo E el conjunto

de aristas, si la arista $E(a, b)$ es una tupla de dicha relación, se consideran como enteros (a es menor que b). De esta forma se eliminan los lazos y se evita contar una arista dos veces.

En álgebra relacional, lo anterior se puede obtener mediante los *joins* naturales:

$$E(X, Y) \bowtie E(X, Z) \bowtie E(Y, Z)$$

o, en lenguaje *SQL*:

SELECT $e1.A,\ e1.B,\ e2.B$
FROM $E\ e1,\ E\ e2,\ E\ e3$
WHERE $e1.A = e2.A\ AND\ e1.B = e3.A$
$AND\ e2.B = e3.B$

Notar que en el *join*, cada triángulo aparece una sola vez. El triángulo formado por $v1,\ v2\ y\ v3$ aparece cuando X, Y, Z son tres nodos en orden numérico, es decir, $X < Y < Z$. Si $v1,\ v2\ y\ v3$ están ordenados, X sólo puede ser $v1$, Y sólo puede ser $v2$ y Z sólo puede ser $v3$.