UNIVERSIDAD
DE LA REPUBLICA
URUGUAY

Aplicación de aprendizaje automático a la detección de fraude en tarjetas de crédito

Lucas Langwagen Fripp

Programa de Posgrado en Ingeniería Matemática
Facultad de Ingeniería
Universidad de la República

Montevideo – Uruguay
Octubre de 2019

UNIVERSIDAD
DE LA REPUBLICA
URUGUAY

Aplicación de aprendizaje automático a la detección de fraude en tarjetas de crédito

Lucas Langwagen Fripp

Tesis de Maestría presentada al Programa de Posgrado en Ingeniería Matemática, Facultad de Ingeniería de la Universidad de la República, como parte de los requisitos necesarios para la obtención del título de Magister en Ingeniería Matemática.

Director de tesis:
Ing. Mag. PhD. Prof. Ignacio Ramirez Paulino

Codirector:
Ing. Klaus Rotzinger

Director académico:
Dr. Prof. Paola Bermolen

Montevideo – Uruguay
Octubre de 2019

Langwagen Fripp, Lucas

Aplicación de aprendizaje automático a la detección de fraude en tarjetas de crédito / Lucas Langwagen Fripp. - Montevideo: Universidad de la República, Facultad de Ingeniería, 2019.

xi, 125 p. 29, 7cm.

Director de tesis:

Ignacio Ramirez Paulino

Codirector:

Klaus Rotzinger

Director académico:

Paola Bermolen

Tesis de Maestría – Universidad de la República, Programa de Ingeniería Matemática, 2019.

Referencias bibliográficas: p. 101 – 103.

1. Fraude en tarjetas de crédito, 2. Aprendizaje automático, 3. Extracción de características, 4. Aprendizaje no supervisado. I. Ramirez Paulino, Ignacio *et al.* II. Universidad de la República, Programa de Posgrado en Ingeniería Matemática. III. Título.

INTEGRANTES DEL TRIBUNAL DE DEFENSA DE TESIS

Dr. Prof. Álvaro Pardo

Dr. Prof. Matías Bourel

Ing. Prof. Alicia Fernández

Montevideo – Uruguay
Octubre de 2019

A aquellos y aquellas cuya
paciencia impidió que se sofoque
la mecha.

Agradecimentos

En primer lugar, quisiera expresar mi enorme gratitud por mi tutor, el Dr Ignacio Ramirez. Su dirección, su acompañamiento y su paciencia fueron fundamentales para la realización de este trabajo.

Luego, quisiera agradecer a PayGroup/Evertec, empresa donde pude desarrollar la investigación que culmina con esta tesis y que proporcionó los datos utilizados para la misma, apostando a la innovación. De entre las grandes personas que conocí allí, quisiera destacar a mi amigo Ignacio Gomez, compañero permanente e inmejorable durante este proyecto.

Adicionalmente, estoy sumamente agradecido a ICT4V, institución que gracias a la gestión de su director Daniel Kofman y a la colaboración de sus partners financió el proyecto en el cual se enmarcó esta tesis y facilitó numerosas contactos y herramientas que fueron claves para el desarrollo del mismo (workshops, seminarios y visitas). Entre los actores con los que ICT4V me puso en contacto quisiera destacar y agradecer enormemente al Dr. Álvaro Pardo, por el impulso fundamental que dio a esta investigación con su disposición a compartir y difundir el conocimiento. Tampoco puedo dejar de mencionar al profesor Josef Kittler y sus alumnos Cemre Zor y Francisco Aparicio, quienes aportaron las ideas que fueron el germen de esta tesis durante el workshop en el cual participaron en la Universidad Católica del Uruguay.

Además, agradezco enormemente todos los investigadores y profesionales con quienes tuve contacto durante mi visita académica al laboratorio LINCS y el Institut Mines-Télécom Paristech en Francia; particularmente, al director del Instituto de Computación Gerard Memmi y al Dr. Albert Bifet quién me recibió y acompañó durante esta visita, compartiendo sus amplios conocimientos en el área.

Y finalmente, a mi familia y amigos: gracias por darme la seguridad y la certeza de saber que siempre vamos a estar juntos, en todo cambio y en toda transformación.

"Begin at the beginning," the King said, very gravely, "and go on till you come to the end: then stop."

Lewis Carroll, Alice in Wonderland

RESUMEN

En esta tesis se aborda el problema de la detección de fraude en tarjetas de crédito mediante el uso de modelos construidos con técnicas de Aprendizaje Automático. Después de un análisis del estado del arte y de la evaluación de un procedimiento de creación de modelos anteriormente utilizado por Evertec (empresa de medios de pagos que impulsa esta investigación), se propone un método novedoso de extracción de características. El mismo busca obtener variables que exploren el comportamiento habitual del cliente y permitan detectar desviaciones. Luego, estas nuevas variables pueden usarse como entrada para el modelo y aumentar su poder predictivo. Además de explicar la forma de cálculo, en este trabajo se describen posibles optimizaciones de las variables calculadas mediante la exploración de metaparámetros y se presentan los resultados obtenidos sobre bases de datos reales.

Palabras claves:
Fraude en tarjetas de crédito, Aprendizaje automático, Extracción de características, Aprendizaje no supervisado.

ABSTRACT

In this thesis, Machine Learning techniques are used to tackle the problem of fraud detection in credit card transactions. In the beginning, state of the art papers are reviewed along with Evertec's (payment methods company which leads this project) previously used methodologies. After that, a new feature extraction method is proposed, aimed to create variables that help detect deviations form a client's typical behaviour. Then, these new characteristics can be used as an input for a model, heightening its predictive power. In addition to explaining the calculations involved, we discuss possible optimizations of the new variables by using metaparameters and present the results obtained in real life databases.

Keywords:
Credit card fraud, Machine Learning, Feature extraction, Unsupervised learning.

Tabla de contenidos

Capítulo 1

Introducción

El negocio de pagos con tarjetas genera réditos a través de un volumen muy grande de transacciones con poco margen de ganancia por transacción. Si bien los fraudes son raros (del orden de 1 fraude cada miles de transacciones, dependiendo del contexto), estos pueden reducir significativamente el margen de ganancia de la institución financiera, ya que el reintegro del monto de una única transacción puede ser igual a la suma de la ganancia marginal de una gran cantidad de transacciones legítimas. Es por eso que existen en el mercado distintas herramientas informáticas que son capaces de estudiar las transacciones de cada cliente y, mediante algún mecanismo, producir una alarma cuando alguna de ellas parece sospechosa. Luego, un equipo de analistas de riesgo estudia las mismas y eventualmente toma medidas en función del riesgo de la alerta; éstas podrían ser por ejemplo: alertar mediante una aplicación al tarjeta habiente sobre la alarma generada, llamarlo para verificar la veracidad de la compra, o eventualmente bloquear su plástico.

Sin embargo, el manejo de las alertas tiene asociado un costo directo (sueldo de los analistas, costo de las llamadas, etc.) e indirecto, pero también relevante (molestias a los clientes, pérdida de confianza en la institución), que pueden ser una potencial fuente de pérdidas para la entidad. Esto genera una tensión entre los casos extremos de no generar demasiadas alarmas (entonces no hay detección de casi ningún fraude) y evitar la mayoría de los fraudes (a un costo de análisis posiblemente insostenible). La búsqueda del equilibrio entre estos dos extremos es un problema no trivial y de suma importancia, al que se destinan cada vez más tiempo y recursos, y que plantea un desafío interesante a nivel científico para desarrollar nuevos y mejores modelos de detección.

1.1. Sobre Evertec y *RiskCenter*

Evertec es una empresa de software que ofrece soluciones tecnológicas para el control y manejo del riesgo. Actualmente, varias entidades financiaras (tanto adquirentes como emisoras de tarjetas de crédito) utilizan un sistema de Evertec llamado *RiskCenter* para el monitoreo de fraude transaccional. El mismo recibe una copia de las interacciones entre el cliente y la entidad, y cuando una interacción parece sospechosa genera una alerta utilizando alguno de los siguientes enfoques:

Reglas

Consiste en una evaluación en cadena de condiciones expresadas sobre los datos de las transacciones. Para cada transacción, según cuáles de estas condiciones cumpla, se le asigna un "score de riesgo" (esto es: un número que puede interpretarse como la probabilidad de que la transacción sea un fraude). Cada institución puede elegir que nivel de riesgo está dispuesta a tolerar, y en base a este umbral, se alertan todas las transacciones con un score mayor al mismo.

Desde un punto de vista algorítmico, esta aproximación es equivalente a un árbol de decisión. La estructura del mismo puede ser la recomendada por Evertec (en cuyo caso se entrena automáticamente para maximizar el desempeño) o de lo contrario puede ser construido en base a conocimiento de negocio de los analistas de riesgo de la institución. Las reglas tienen la ventaja de ser fácilmente comprensibles, pero generalmente tienen un desempeño sensiblemente peor que otros algoritmos más sofisticados.

Modelos predictivos

RiskCenter ofrece también la posibilidad de incorporar un modelo más complejo basándose en clasificadores más complejos de Aprendizaje Automático (redes neuronales, Random Forest, etc). Estos modelos resultan más oscuros para la institución financiera, pero alcanzan los mejores resultados.

Como manera de enriquecer las variables propias de una transacción, *RiskCenter* tiene la capacidad de calcular "acumuladores". Estos permiten considerar el comportamiento previo del cliente en función de un período de tiempo determinado, un criterio de filtro y una función de agregación

(por ejemplo: sumando los montos de transacciones de la tarjeta en el mes anterior o promediando la cantidad de transacciones por día en un comercio determinado en el último mes, etc). Si bien existe mucha flexibilidad para definir acumuladores, la capacidad de cálculo de la herramienta es limitada. Además, la velocidad de cálculo en línea se ve afectada si se agregan demasiadas variables o si las mismas exploran un período de tiempo demasiado extenso.

Para la construcción de un modelo de detección de fraude actualmente se realiza un trabajo caso a caso para cada cliente que desee incluir esta capacidad en su instalación de RiskCenter. El proceso se hace en forma manual y requiere aproximadamente de unas ocho semanas. Los desafíos más importantes a los que se enfrentan los analistas estadísticos de la empresa a la hora del desarrollo son los siguientes:

- **Dilución del fraude:** como ya se mencionó, el fraude es un evento muy infrecuente (entre 1 en 30000 y 1 en 1000, dependiendo del contexto). Ésto afecta mucho la capacidad de los algoritmos de aprender.
- **Demoras:** el volumen de los datos suele ser grande (millones de transacciones), por lo que se requiere un período considerable de tiempo para procesarlos (incluyendo su carga en la base de datos, limpieza, conversión a un formato adecuado, etc). Esto hace que las iteraciones entre evaluar un modelo y mejorarlo lleven un largo tiempo, limitando la exploración de opciones para cada cliente.
- **Obsolescencia:** los patrones de fraude están en constante evolución, pues los criminales se adaptan a los sistemas de seguridad y al ver frustrados sus intentos, desarrollan nuevas maneras de evadir los controles. Por lo tanto, los modelos estáticos tienden a degradar su desempeño a lo largo del tiempo.

1.2. Proyecto Evertec - ICT4V

En la búsqueda de mejorar las capacidades y desempeño de los modelos predictivos, Evertec se alía con ICT4V, un centro tecnológico donde participan universidades y empresas del sector público y privado para generar valor e innovación a través de las Tecnologías de la Información. Es así que en conjunto, ambas instituciones lanzaron un proyecto titulado "*Aplicación de*

aprendizaje automático a la prevención de fraude en transacciones de crédito". Como parte del mismo, se desarrollaron las siguientes actividades:

- Contratación de dos analistas estadísticos para participación en el proyecto (entre ellos, el autor de este documento). Los mismos fueron al mismo tiempo colaboradores en Evertec y becarios de maestría de ICT4V, desarrollando sus planes de maestría y sus tesis en torno a los objetivos del proyecto.
- Participación en actividades relacionadas en ICT4V: dictado y asistencia a charlas en el seminario del centro y en el Grupo de Lectura sobre Aprendizaje Automático, participación en congresos como el CIARP (*Congreso Iberoamericano de Reconocimiento de Patrones*) y workshops como "*Reconocimiento de Patrones & Detección de Anomalías*" y "*Big and Complex Data Theory, Applications and Value Creation*".
- Participación en un intercambio académico con el Institut Mines-Télécom Paristech de dos meses de duración con el profesor Albert Bifet.

Como resultado final del proyecto se obtuvo una nueva metodología de extracción de características que puede ser usada para enriquecer los modelos de detección de fraude, una serie de scripts que implementan y automatizan la misma (y todas las etapas adicionales de construcción de un modelo), y este texto que detalla el proceso completo de la investigación.

1.3. Organización del documento

Esta tesis está dividida en los siguientes capítulos:

1. **Introducción:** El presente capítulo, donde comentamos los aspectos generales del problema y como se desarrolló la investigación que dio origen a esta tesis.
2. **Marco teórico y diagnóstico inicial del problema:** Este capítulo comienza repasando los principales conceptos del área de Aprendizaje Automático y estableciendo las notaciones que luego serán utilizadas durante el resto del documento. Luego se presentan las bases de datos con las que trabajamos, repasando sus principales características. Finalmente, se hace un análisis crítico de la metodología previa que utilizaba la empresa para construir modelos y del estado del arte en

detección de fraudes. En base a este análisis, obtendremos conclusiones que motivarán la construcción de una nueva metodología descrita en capítulos posteriores.

3. **Definición de un Score de Outlierness:** Aquí es donde presentamos nuestro principal aporte al problema de la detección de fraudes: una forma de extraer características que llamaremos Score de Outlierness. Haremos una definición del mismo paso a paso, explicando los razonamientos que lo motivan.
4. **Optimización de parámetros del Score de Outlierness:** Una vez presentada la metodología, nos abocaremos en este capítulo a mostrar los metaparámetros que se pueden definir y discutiremos la forma en que pueden optimizarse para obtener el mejor rendimiento posible en cada modelo que se quiera construir. El capítulo finaliza con un resumen del nuevo procedimiento de construcción de modelos propuesto.
5. **Procedimiento final e implementación:** En esta sección se resumen el procedimiento completo desarrollado a partir de lo presentado en los capítulos anteriores . Así mismo, se presenta el paquete que implementan el procedimiento y se presenta su performance.
6. **Resultados:** En esta sección se presenta y analiza el desempeño de los modelos construidos con la metodología desarrollada en las bases disponibles. Adicionalmente, se obtienen conclusiones sobre el Score de Outlierness desarrollado en base al estudio de la importancia de las variables utilizadas por los modelos.
7. **Conclusiones finales:** El documento se cierra con una reflexión sobre la totalidad del proceso de investigación y una discusión sobre las posibles líneas de investigación futura.

Capítulo 2

Antecedentes y estado del arte

2.1. Conceptos generales de Aprendizaje Automático

En términos generales, el Aprendizaje Automático es una disciplina híbrida entre Estadística y Ciencias de la Computación en la que se usan diferentes técnicas y algoritmos con el objetivo de generar sistemas que sean capaces de realizar predicciones y/o tomar decisiones por sí mismos en base a datos (decimos que los algoritmos, se *entrenan* sobre los datos para aprender la tarea que queremos).

Las técnicas utilizadas pueden clasificarse de muchas maneras. Por ejemplo: en problemas de regresión la salida del sistema es un valor predicho para una variable continua. En cambio, en problemas de clasificación, se dispone de dos o más clases y la salida es una clase predicha para cada dato sobre el que se evalúa. Los problemas de clasificación en dos clases (como el nuestro) pueden pensarse naturalmente como un problema de regresión donde la salida es la probabilidad de pertenencia a la clase positiva (o un score de afinidad si admitimos valores fuera del intervalo $[0; 1]$). En problemas de clasificación, es usual llamar *clasificadores* a los algoritmos que se entrenan sobre los datos.

Otra distinción en base a los datos es según si se tiene o no acceso a la salida esperada para las instancias de entrenamiento (por ejemplo, la clase a la que pertenece cada dato en problemas de clasificación). Dependiendo de a que contexto apliquen, diferenciamos entre técnicas *supervisadas* y *no supervisadas* (respectivamente)[1].

[1]Cuando se tiene la salida esperada solamente para algunos casos se habla de técnicas

Validación de modelos

Un problema que aparece continuamente en el área es el de la complejidad de los modelos: los algoritmos más sencillos no siempre logran capturar las relaciones más difíciles entre atributos y aprender los patrones satisfactoriamente, pero si la complejidad de los algoritmos es excesiva, puede ocurrir que los patrones se aprendan demasiado bien: esto llevaría a que observemos un desempeño extremadamente preciso en los datos de entrenamiento pero muy pobre cuando es puesto en funcionamiento con datos que nunca observó. Este fenómeno se conoce como *sobreajuste* a los datos (overfitting en inglés) y es un peligro que se encuentra siempre presente al entrenar modelos[2].

Para poder diagnosticar y evitar caer en este problema, una buena práctica es particionar el conjunto de datos totales en datos de entrenamiento (que serán observados por el algoritmo) y datos de validación que se usarán exclusivamente para estimar el desempeño del algoritmo. Cuando se empieza con modelos más sencillos, se espera que el desempeño sea parecido en ambos conjuntos (y no demasiado bueno). A medida que la complejidad aumenta, ambos errores deberían disminuir hasta un punto óptimo (que queremos determinar) en el que el error de validación comienza a subir dado que el modelo comienza a sobreajustarse a los datos de entrenamiento.

Una técnica muy extendida para la evaluación de modelos es la conocida como *k-fold Cross Validation*. Consiste en partir el conjunto de datos en k partes (*folds* en inglés) del mismo tamaño y entrenar k algoritmos iguales en $k-1$ particiones, dejando la restante para validación (usando una vez cada partición para validación). La ventaja de este método es que permite mantener la evaluación en conjuntos de datos no observados al mismo tiempo que se tienen k evaluaciones distintas, lo que permite observar la variabilidad en el desempeño.

Extracción de características

Es la etapa en la construcción de modelos en la que se calculan características nuevas a partir de las ya existentes. Una técnica en particular

semi-supervisadas.

[2]El fenómeno contrario por el que el modelo es demasiado sencillo para los datos se conoce en inglés como *underfitting* y no tiene una traducción estándar al español.

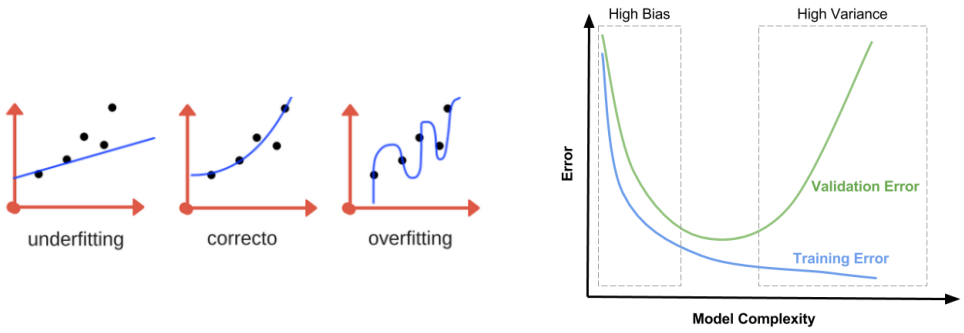


Figura 2.1: Ilustración de los conceptos mencionados con datos artificiales. En la gráfica de la izquierda se muestran tres modelos que intentan explicar los datos. Un modelo lineal es demasiado sencillo para capturar la relación, mientras que el modelo polinomial de la derecha, si bien ajusta perfectamente a todos los puntos, no generalizará bien para datos nuevos por su excesiva variabilidad. La ilustración de la derecha muestra dos curvas representando los errores en los conjuntos de entrenamiento (azul) y validación (verde) en función de la complejidad del modelo. Antes del punto óptimo, decimos que el modelo tiene un sesgo (*bias*) alto, mientras que luego del mismo decimos que el modelo tiene alta varianza (*variance*).

a la que haremos referencia es la conocida como *One Hot Encoding*. En la misma, a partir de una variable nominal X con recorrido $\{x_1, \ldots, x_k\}$ se construyen k variables continuas $Y_i \in [0, 1]$ que se calculan como $Y_i = \mathbb{1}(X = x_i),\ i = 1, \ldots, k$. Este método es ampliamente usado en la literatura para entrenar clasificadores que solo aceptan variables continuas sobre datos nominales. En un caso como el nuestro en que casi todas nuestras variables son nominales debemos tener precaución al aplicar este método, ya que aumenta considerablemente la cantidad de variables con las que se trabaja (sobre todo teniendo en cuenta el tamaño del recorrido de algunas características como el código MCC o el identificador de comercio) lo que acarrea problemas a nivel computacional.

Selección de características

Para la gran mayoría de los problemas del área, más características no implica necesariamente una mejoría en el desempeño. Esto es principalmente por dos razones: la primera es una cuestión de implementación, ya que al aumentar la cantidad de variables aumenta notablemente la complejidad de los modelos, el tamaño de las bases de datos y el tiempo de entrenamiento de los algoritmos. La segunda razón es que tener demasiadas características

aumenta la tendencia al sobreajuste y puede llevar a un peor desempeño. El proceso en el cual se determina un subconjunto óptimo de características a partir de todas las disponibles se denomina *selección de características*. donde $\mathrm{Sop}(X_i)$

Según las técnicas utilizadas, los métodos de selección de características pueden clasificarse de la siguiente manera:

1. **Métodos de filtrado:** son aquellos métodos en los que se utiliza una medida (llamada *función de filtrado*) que se calcula para cada variable del conjunto total como indicador de su utilidad. Por ejemplo: la información mutua (Zhao et al., 2018) o la correlación de cada característica con la variable objetivo (en el caso supervisado).
2. **Métodos wrapper:** consisten en entrenar un clasificador para cada subconjunto de características que se quiera evaluar y utilizar el desempeño en un conjunto de validación como medida para elegir el subconjunto óptimo.
3. **Métodos mixtos:** son aquellos que dan como salida al mismo tiempo una función de filtrado en los datos y el desempeño en un conjunto de validación. Por ejemplo: un *Random Forest* es un clasificador basado en árboles de decisión que además da un score para cada característica utilizada llamado la *importancia* de cada variable (ver definición en (Louppe et al., 2013)).

La ventaja de los métodos que utilizan filtrado es su facilidad y velocidad. Sin embargo, dado que muchas veces las funciones no logran capturar el efecto del desempeño conjunto de dos o más variables entre sí, los mejores resultados se obtienen utilizando métodos wrapper. Estos son por lo general más lentos de ejecutar, ya que en teoría, si se parte de N características, tendríamos que explorar 2^N subconjuntos del conjunto total hasta encontrar el óptimo global (lo cual se vuelve impracticable rápidamente incluso para valores bajos de N). Por estos motivos es que se utilizan distintas aproximaciones subóptimas para encontrar óptimos locales en tiempos razonables. Dos técnicas *greedy* de uso extendido (si bien no son las únicas) son las que presentamos a continuación:

- **Forward stepwise selection:** partir del subconjunto vacío e ir agregando secuencialmente la característica X_i que en cada etapa da la mayor mejoría en el desempeño de entre las X_j que aún no fueron agregadas.

- **Backward stepwise selection:** partir del subconjunto completo de características e ir quitando secuencialmente la característica X_i que en cada etapa da la mayor mejoría en el desempeño de entre las X_j que aún no fueron descartadas.

Clasificadores para aprendizaje supervisado

Una vez que se han ejecutado las etapas anteriores de extracción y selección de características y se ha llegado a un conjunto de variables, es momento de entrenar un clasificador sobre los datos para que aprenda a predecir la variable objetivo (suponiendo que contamos con ella, es decir, en el caso de aprendizaje supervisado). Existe una vasta cantidad de algoritmos que se han diseñado con este fin, por lo que la elección del más apropiado no es una cuestión sencilla y en la práctica suele estar ligado a cuestiones relativas a los datos: la disponibilidad, el tipo de datos (numéricos, strings, etc) y la cantidad son factores determinantes. En (Jain et al., 2000) se dividen los algoritmos de clasificación en tres grandes familias:

1. **Métodos basados en similaridad:**
   La idea general de estos métodos es comparar cada instancia nueva contra un prototipo de cada clase usando alguna métrica, para luego asignarle la clase cuyo prototipo sea más parecido a la instancia. Algunos de los algoritmos más usados en esta familia:
   - k-NN (k *Nearest Neighbours*): cada instancia se compara contra sus k vecinos más cercanos (usando alguna distancia, por ejemplo la euclídea) y se le asigna la clase mayoritaria dentro de sus vecinos.
2. **Métodos probabilísticos:**
   Estos métodos utilizan el principio de máxima verosimilitud u otras aproximaciones probabilísticas para tomar decisiones. Algunos de los algoritmos más usados en esta familia:
   - Clasificadores bayesianos ingenuos: haciendo ciertas suposiciones de independencia entre las variables condicionadas a la clase (de ahí la denominación de *ingenuos*), se utiliza la formula de Bayes para derivar las probabilidades de pertenecer a cada clase para una nueva instancia. Luego, se utiliza alguna regla para asignar una clase (por ejemplo, la clase con mayor probabilidad a posterior).

- Clasificadores lineales o logísticos: utilizan el principio de máxima verosimilitud para determinar la combinación lineal de las variables que mejor se ajuste a los datos observados.

3. **Métodos basados en fronteras de decisión:**
   Se busca separar el espacio de características de la base en regiones, de manera que en cada una queden separadas lo más posible las clases. Algunos de los algoritmos más usados en esta familia:

   - SVM (*Support Vector Machines*): buscan aprender el hiperplano que mejor separe las dos clases en el conjunto de entrenamiento (suponiendo clasificación binaria). Utilizando funciones de kernel, se pueden aprender fronteras no lineales.
   - Perceptrón - Redes neuronales: el perceptrón en sus inicios fue pensado como un algoritmo que aprende un plano separador entre las clases que se va actualizando a medida que llegan nuevas instancias. Ante su imposibilidad de resolver problemas no linealmente separables, se inventaron las Redes Neuronales que son una combinación de perceptrones de manera que en conjunto pueden aprender fronteras no lineales.
   - Árboles de decisión: se construye un conjunto de reglas de decisión binarias que aplicadas secuencialmente forman un flujo con forma de árbol. Cada instancia, según sus características, llega a un nodo terminal, donde se clasifica según las instancias de entrenamiento que cayeron en el mismo nodo. Este procedimiento equivale a construir fronteras de decisión como partes de hiperplanos paralelos a algún eje.

Dada la gran variedad y constante aparición de nuevas metodologías, esta división es a grandes rasgos, por lo que puede haber algoritmos que estén en más de una familia o en ninguna. Sin embargo, los más frecuentemente usados quedan bien representados en las mismas.

Combinación de clasificadores

Dado que distintos algoritmos pueden estar basados en heurísticas muy diferentes, los modelos resultantes pueden no clasificar bien las mismas instancias, aún cuando su desempeño global sea similar. Incluso podría darse

este fenómeno entrenando dos modelos con un mismo algoritmo que dependa de elementos estocásticos (por ejemplo: un árbol de decisión que en cada paso elige un subconjunto aleatorio de las variables para generar un nodo). Por este motivo, se han desarrollado numerosos algoritmos de combinación de clasificadores y así combinar de manera óptima la salida de dos modelos que pueden haberse especializado en dos tipos distintos de instancias. Uno de los más conocidos es el desarrollado por (Breiman & Schapire, 2001) para combinar árboles de decisión en un solo clasificador llamado *Random Forest*.

Otra de las técnicas que mencionamos dada su importancia es conocida como *Boosting* (Freund & Schapire, 1999) y fue pensada inicialmente para combinar modelos simples (o débiles) pero diversos en un gran modelo robusto. En la implementación original se utilizaban árboles de decisión de baja profundidad como modelos débiles (aunque puede generalizarse a cualquier clasificador). El algoritmo entrenaba progresivamente cada uno de estos árboles aplicando pesos a la instancias, los cuales se iban actualizando en la iteración i según el desempeño del árbol $i-1$: las instancias que hubieran sido mal clasificadas por éste tenían mayor peso (y lo contrario para las bien clasificadas). De esta manera, se conseguía que cada árbol se enfocara en clasificar correctamente lo que el árbol anterior no había podido. Luego, se asigna un peso a cada árbol y se considera que la decisión final del conjunto es el resultado de la votación ponderada de todos los árboles.

Sobre esta idea se ha seguido trabajando hasta llegar a algoritmos sumamente potentes como es el caso de *XGBoost* (Chen & Guestrin, 2016). Este se basa en la misma idea que el *Boosting* original, pero modificando (entre otras cosas) la manera en que se calculan los pesos de las instancias y los árboles. En este nuevo algoritmo se incluyen términos de regularización (para controlar el tamaño de los árboles, y en consecuencia, el overfitting) y se permite incluir una función de costo personalizada. El resultado suele ser un conjunto de árboles más pequeños, más robustos y de mejor desempeño que la implementación clásica de Random Forest.

2.2. Notaciones y terminología

En esta sección definiremos las notaciones y los términos de Aprendizaje Automático o del área de detección de fraudes que usaremos en el resto del documento:

- Para referirnos a una transacción genérica de una base de datos usaremos la notación $\vec{t} = (x_1, \ldots, x_k)$ donde las columnas son las características $X_1, \ldots, X_k$. Cada coordenada i-ésima $(\vec{t})_i = x_i$ corresponde al valor de la transacción en la columna X_i.
- Dada una transacción genérica $\vec{t}_{n+1}$ que es la $n+1$-ésima hecha por esa tarjeta, diremos que el *índice* de esta transacción es $n+1$.
- Dada una transacción genérica $\vec{t}_{n+1}$ cuyo índice es $n+1$, denotaremos como $T(\vec{t}_{n+1}) = \{\vec{t}_1, \ldots, \vec{t}_n\}$ al conjunto de histórico previo (es decir: el conjunto de las n transacciones anteriores hechas por la misma tarjeta). Asumimos que las transacciones $\vec{t}_i$ están indizadas según el orden ascendente por la fecha. Las coordenadas de una transacción $\vec{t}_j$ las representaremos como $x_1^j, \ldots, x_k^j$. El conjunto de histórico en la coordenada i-ésima es $T_i(\vec{t}_{n+1}) = \{x_i^1, \ldots, x_i^n\}$
- $|A|$ es el cardinal del conjunto A, es decir, la cantidad de elementos de A.
- Dada una variable aleatoria discreta X, $\mathrm{Rec}(X)$ representa el conjunto de valores que puede tomar.
- Dada una variable aleatoria discreta X y un conjunto de instancias A de X, $\mathrm{Sop}(X, A)$ representa el conjunto de valores en $\mathrm{Rec}(X)$ que aparecen al menos una vez en el conjunto A. Por simplicidad, en contextos donde el conjunto de datos A está dado, omitiremos esta coordenada y escribiremos $\mathrm{Sop}(X)$.
- $H(X) = -\sum_{x \in \mathrm{Rec}(X)} \log_2(p_x) p_x$ es la entropía teórica de X. Por simplicidad, también usaremos la notación $H(X)$ para la entropía empírica calculada usando las probabilidades estimadas $\hat{p}_x$ en un conjunto de datos dado.
- Similarmente al caso anterior, usaremos la notación $I(X, Y) = H(X) + H(Y) - H(X, Y)$ para referirnos a la información mutua empírica calculada usando probabilidades estimadas en un conjunto de datos.
- Llamaremos **dilución** de un conjunto de transacciones a la relación entre cantidad de legítimas y cantidad de fraudes.
- Un modelo de detección de fraudes es un algoritmo que asigna a cada transacción genérica $\vec{t}$ un número $s(\vec{t})$ que llamaremos **score de riesgo**[3]

[3]Podemos asumir que $s(\vec{t}) \in [0, 1]$, pues en caso contrario siempre se puede aplicar alguna transformación monótona adecuada.

(o simplemente **score**).

- El **umbral de detección** (o **umbral** en forma abreviada) es un número $u \in [0, 1]$ que se usa para poner en funcionamiento el modelo de detección de fraudes, ya que *RiskCenter* generará una alerta por cada transacción $\vec{t}$ tal que su score asignado $s(\vec{t})$ cumpla que $s(\vec{t}) \geq u$.
- Dado un modelo, un umbral u y un conjunto de transacciones $A = P \cup N$ a evaluar (donde las transacciones de P son fraudulentas y las de N son legítimas), utilizaremos las notaciones usuales para referirnos a los conjuntos que describen la clasificación del modelo:

  Falsos positivos: $FP = \{\vec{t} \in N : s(\vec{t}) \geq u\}$
  Verdaderos positivos: $TP = \{\vec{t} \in P : s(\vec{t}) \geq u\}$
  Falsos negativos: $FN = \{\vec{t} \in P : s(\vec{t}) < u\}$
  Verdaderos negativos: $TN = \{\vec{t} \in N : s(\vec{t}) < u\}$

- Cuando se quieren optimizar varios parámetros $\theta_1, \ldots, \theta_n$ de un modelo en simultáneo, llamaremos *búsqueda de grilla* (o *grid search*) a la técnica que consiste en elegir conjuntos Θ_i de valores que se quieren explorar para cada θ_i y armar una grilla con todos las combinaciones posibles[4] de valores de Θ_i. El óptimo se elegirá probando con un subconjunto de valores de la grilla. En caso que se exploren todos los valores, diremos que la búsqueda es *exhaustiva*[5].
- En el contexto del punto anterior y análogamente a lo que ocurre con la selección de variables (como vimos bajo **Selección de Características** en la sección 2.1), diremos que usamos *wrapping* para evaluar los puntos de una grilla si la medida usada para elegir la mejor combinación $(\theta_1, \ldots, \theta_n)$ es el desempeño de un modelo entrenado con estos parámetros.
- Durante este documento, mediremos siempre el desempeño de un modelo usando la técnica de *k-fold Cross Validation* o *validación cruzada* (definida bajo el título **Validación de Modelos** en la sección 2.1) para poder trazar la variabilidad de los resultados.

[4]Es decir: la grilla es el producto cartesiano $\Theta_1 \times \ldots \times \Theta_n$.

[5]En contextos donde el tamaño de la grilla vuelve implausible una búsqueda exhaustiva, se pueden explorar otras técnicas. Sin embargo, en los casos particulares abordados en esta tesis, las grillas construidas

2.3. Presentación de los conjuntos de datos

Para desarrollar la investigación se tuvo a disposición tres bases de datos de instituciones financieras distintas, donde cada una corresponde al total de transacciones recibidas por el banco en un período de un año. Cada base tiene una cantidad de columnas distintas, pero en todos los casos se nos proporcionó una etiqueta de fraude para cada transacción. Además, se decidió particionar las bases 1 y 2 en dos subconjuntos cada una pues en ambos casos la misma institución financiera trabaja con dos emisores distintos, por lo que el significado de algunos campos depende del subconjunto sobre el que se trabaje. A continuación se presentan algunos indicadores de las bases:

Base	Columnas	Legítimas	Fraudes	Tarjetas	Tarjetas con fraude	Dilución
1.1	15	35.000.000	9.881	740.349	2.975	3.580 : 1
1.2	15	4.700.000	4.965	144.925	1.712	959 : 1
2.1	10	9.000.000	12.035	125.105	3.068	777 : 1
2.2	10	49.000.000	34.028	755.841	8.265	1.434 : 1
3	12	128.500.000	112.510	3.126.965	12.567	1.150 : 1

Tabla 2.1: Indicadores para cada una de las bases de datos con las que se trabajó. *Base* es el identificador de cada uno de los conjuntos de datos ($i.j$ indica el subgrupo j de la base i). *Columnas* representa la cantidad total de columnas que se mantuvieron después de ejecutar una limpieza de datos. *Legítimas* y *Fraudes* son las cantidades de transacciones legítimas (aproximadas) y fraudulentas respectivamente. *Tarjetas* y *Tarj. Fraude* son las cantidades de tarjetas totales y que algúna vez tuvieron fraude respectivamente. *Dilución* representa la cantidad promedio de legítimas por cada fraude en la base.

Adicionalmente, listamos algunos de los atributos principales con los que se cuenta en la mayoría de las bases:

Monto Monto de cada transacción en una moneda de referencia.

Fecha y hora Fecha y hora local de la transacción en formato `YYYYMMDD HH:MI:SS`. De esta columna podemos extraer cuatro columnas más que son: el día del mes, el día de la semana, la hora del día y el tiempo desde la última transacción de la misma tarjeta. Sin embargo, para trabajar con las tres primeras, debemos tener la precaución de considerar que tienen una métrica circular (ver Apéndice para más detalles).

Moneda Código de la moneda local de la transacción.

País Código del país donde se realiza la transacción.

Código de comercio y/o terminal Identificador cada comercio o terminal donde se realiza la transacción. Este código es asignado

internamente por cada institución financiera y por lo general no tenemos una manera de saber a que comercio refiere un código particular.

MCC Abreviación de *Merchant Category Code*. Es una de las columnas que aparece en formato estándar en todas las bases de dato. Es un valor numérico que se asigna a cada comercio, agrupando aquellos que pertenecen a un ramo similar. Por ejemplo: en el estándar de VISA, el valor 6011 representa retiros en cajeros automáticos, mientras que los valores entre 3000 y 3350 representan distintas aerolíneas.

Modo de entrada Identificador del método por el cual se ingresó el número de tarjeta y la fecha de vencimiento en el sistema del emisor. Por ejemplo: banda magnética, chip, *contactless*, manual (para transacciones hechas en internet), etc. La codificación de este campo sigue un estándar.

PIN Entry Capability Indicador de la capacidad y el estado de la terminal para aceptar el código PIN (la terminal puede o no aceptar PINs o puede estar temporalmente inhabilitada).

Indicador de PIN Indica si, efectivamente, el cliente ingresó su PIN en la terminal.

Código de respuesta Código que indica si la transacción fue aprobada por el emisor de la tarjeta de crédito, y en caso de no serlo, el motivo de rechazo (por ejemplo: saldo insuficiente, errores de procesamiento, PIN inválido, etc). En algunas bases, este campo no aporta información pues las únicas transacciones con las que disponemos son las aprobadas.

2.3.1. Consideraciones

Finalmente, para terminar de caracterizar el contexto del problema mediante los datos, discutiremos algunas de las particularidades de los mismos y sus consecuencias a la hora de realizar el modelado:

1. Como puede apreciarse en la sección anterior, disponemos de una cantidad marginal de fraudes en comparación con el total de transacciones de la base. Dentro del área de Aprendizaje Automático este fenómeno se conoce como *desbalance de clases* y genera problemas conocidos (aunque no se tenga una única y clara solución para los mismos, como veremos en 2.5). La mayor dificultad que ocasiona es la degradación de las medidas de desempeño, ya que la cantidad de falsas alarmas

se dispara rápidamente. Esto tiene consecuencias a la hora de evaluar los resultados pero también en el entrenamiento, pues las métricas que usan los algoritmos de clasificación durante el entrenamiento no siempre funcionan adecuadamente en un contexto de desbalance.

2. La gran mayoría de las variables que tenemos a disposición son nominales. De hecho, las únicas variables continuas que están presentes en todas las bases son el monto y las derivadas de la fecha y hora (ver Apéndice para detalles de cómo se calculan). Este hecho es de suma importancia a la hora de buscar algoritmos o metodologías en la literatura, pues no todos son aplicables (al menos de forma directa) a datos de esta naturaleza (profundizaremos al respecto en la sección 2.5).
3. Una característica nominal X que presenta valores $x \in \text{Rec}(X)$ con demasiados pocos casos en la base puede generar una tendencia al sobreajuste, ya que no hay suficientes instancias para que las decisiones que un clasificador tome en base al caso $\{X = x\}$ sean robustas. En el caso de algunas variables (como el número de tarjeta) esto ocurre con todos los valores, así que no podemos usarla para la clasificación[6]. Otras características como el identificador de comercio o el código MCC presentan algunos valores marginales pero otros que sí aparecen una cantidad significativa de veces, por lo que muchas veces optamos por transformarlas antes de usarlas en un clasificador. Por ejemplo, dado un parámetro k que representa la cantidad mínima de casos aceptables y suponiendo que $\text{Rec}(X) = \{x_1, \ldots, x_n\}$, podemos usar la variable $\tilde{X}$ definida como:

$$\tilde{X}(\omega) = \begin{cases} i & \text{si } |\{X = X(\omega)\}| > k \\ n+1 & \text{en otro caso} \end{cases}$$

   De esta manera se agrupan todos los casos que ocurren menos de k veces en la base.
4. Cuando un modelo debe dar una respuesta en cuestión de segundos, decimos que la evaluación debe ser en *real-time*. Para algunas aplicaciones

[6]También contribuye al sobreajuste el hecho de que los números de tarjeta no son una característica global, sino que van cambiando con cada tarjeta nueva, de manera que un modelo que use el número de tarjeta para decidir se vuelve inutil sobre tarjetas nuevas. Esto ocurre también (aunque en menor medida) con el número de terminal o el número de comercio si las reglas aprendidas refieren a comercios o terminales que no se mantienen en el tiempo.

puede ocurrir que una demora de minutos sea aceptable, por lo que decimos que la evaluación del modelo es *near real-time*. Esta distinción es importante pues en ambos contextos, cambian las variables que están disponibles. Por ejemplo: el código de respuesta sólo está disponible para modelos en *near real-time*. Cuando un cliente requiere un modelo que funcione en *real-time*, de acuerdo con el flujo de la transacción, el modelo debe dar una respuesta antes que el emisor de la tarjeta de crédito, por lo que no conocemos el valor del código de respuesta para la transacción que se está queriendo evaluar (aunque sí para su histórico). En consecuencia, esta característica no puede ser utilizada para entrenar clasificadores que deban funcionar en *real-time* (aunque sí podemos usarla para crear acumuladores).

5. Si bien para la mayoría de las variables listadas anteriormente la codificación sigue un estándar determinado por el emisor, si para la clasificación se utilizan variables que no cumplan con esto (por ejemplo, un código de producto de uso interno en la institución financiera) es posible que el modelo aprendido se degrade si en un futuro aparecen valores nuevos o algunos de los observados cambia su significado.
6. Los fraudes de la base se marcan en base a los reclamos de los clientes. Esto implica que los fraudes cometidos en fechas cercanas a la fecha de extracción de los datos tienen una menor tendencia a estar marcados pues los clientes tuvieron menos tiempo para reconocerlos y realizar sus descargos. En algunos casos, en la etapa de limpieza de datos es necesario descartar las instancias más recientes para evitar usar información incorrecta.

2.4. Críticas al procedimiento inicial

En el momento de realizar el estudio del estado del arte, la metodología de creación de modelos que se utilizaba era puramente supervisada, es decir, tratando el problema como uno de clasificación en dos clases. A continuación hacemos un esquema mostrando las etapas del procedimiento, para luego pasar a discutir críticamente cuáles eran las decisiones de diseño más usuales en cada una de ellas:

1. Limpieza de datos.

- Selección de las columnas con información útil.
- Agrupación de valores del recorrido de variables nominales (como vimos en la sección 2.3.1).

2. Exportación.
3. Partición de la base de datos en conjuntos de entrenamiento, validación y testing.
4. Determinación de "cortes".
5. Generación de características mediante acumuladores (usualmente entre 100 y 200 variables nuevas).
6. Selección de características:
   - Filtrado usando criterio de ganancia de información (para descartar características pobres rápidamente dentro de la gran cantidad creadas en el paso anterior).
   - *Backward stepwise selection* con *wrapping*.
7. Optimización de parámetros del clasificador.
8. Evaluación en conjunto de testing.

Si el desempeño en validación no era satisfactorio en el punto 7, se volvía al punto 5, realizando las iteraciones necesarias hasta que el desempeño se volviera aceptable o no se modificara con el agregado sucesivo de variables. Además, el clasificador de preferencia era Random Forest por su capacidad de manejar atributos nominales sin necesidad de pasar por el problema de transformarlos en variables continuas.

En las etapas iniciales de la investigación se realizó un diagnóstico de este procedimiento. A continuación presentamos las principales críticas que se realizaron y cuando corresponda, las alternativas propuestas:

2.4.1. Medidas de desempeño

Para medir el desempeño de un modelo de detección de fraude nos interesa controlar tres cosas: la cantidad de fraudes evitados, la pérdida monetaria evitada y la cantidad de falsas alarmas generadas. Estas magnitudes, sin embargo, no son fijas para cada modelo de detección: recordemos que, como vimos en la sección 2.2, la salida de un modelo es un score de fraude para cada transacción (que podemos suponer se encuentra en el intervalo $[0, 1]$). Como

las transacciones alertadas son aquellas cuyo score s cumpla $s > u$, la cantidad de fraudes alertados puede ser controlada por la institución usuaria del modelo eligiendo un valor adecuado para u.

Dado un umbral u, suponiendo que la clase positiva son los fraudes y utilizando la notación definida en la sección 2.2, definimos a continuación algunas medidas de desempeño de modelos, basandonos en las magnitudes anteriormente mencionadas:

- La **efectividad** es la proporción de fraudes que son correctamente detectados por el modelo. Se define como:

$$Ef(u) = \frac{|TP|}{|TP| + |FN|}$$

- La **efectividad en monto** es la proporción de pérdidas monetarias evitadas por el modelo con respecto al total si el mismo no estuviera funcionado. Esta medida complementa a la anterior pues es fundamental que los modelos logren evitar los fraudes que representan mayores pérdidas para la compañía. Se calcula mediante la siguiente formula:

$$EfM(u) = \frac{\sum_{\vec{t} \in TP} (\vec{t})_i}{\sum_{\vec{t} \in P} (\vec{t})_i}$$

  suponiendo que la coordenada i-ésima $(\vec{t})_i$ representa el monto de la transacción $\vec{t}$.
- La **precisión** de un modelo es la proporción de transacciones alertadas cuyas alarmas fueron correctas. Es decir:

$$Prec(u) = \frac{|TP|}{|TP| + |FP|}$$

- El **False Positive Rate** o **Review** de un modelo es la proporción de transacciones legítimas que el modelo alertó innecesariamente. Esto es:

$$FPR(u) = \frac{|FP|}{|FP| + |TN|} = \frac{|FP|}{|N|} \tag{2.1}$$

- Los **falsos positivos promedio por fraude** son la cantidad esperada de alarmas que deben ser procesadas por un analista para encontrar un

fraude. Se calcula como el inverso de la precisión, es decir:

$$FPPF(u) = \frac{1}{Pr} = \frac{|TP| + |FP|}{|TP|} = 1 + \frac{|FP|}{|TP|} \tag{2.2}$$

El usuario del modelo, al hacer variar el valor de u entre 1 y 0, obtiene valores de Efectividad y Efectividad en Monto que varían entre 0 y 1 respectivamente [7]. Sin embargo, al controlar la Efectividad, se pierde control sobre la cantidad de falsas alarmas: los mejores modelos serán entonces aquellos que minimicen alguna de las medidas de cantidad de falsas alarmas de nuestra elección como función de la Efectividad.

Las curvas resultantes al graficar la $Prec$ y FPR en función de la Efectividad son llamadas PRC y ROC[8] respectivamente. Para ilustrar que forma aproximada se espera que presenten las curvas, en los gráficos 2.2, 2.3 y 2.4 presentamos curvas de desempeño para cuatro modelos artificiales usando las tres medidas introducidas de cantidad de falsas alarmas:

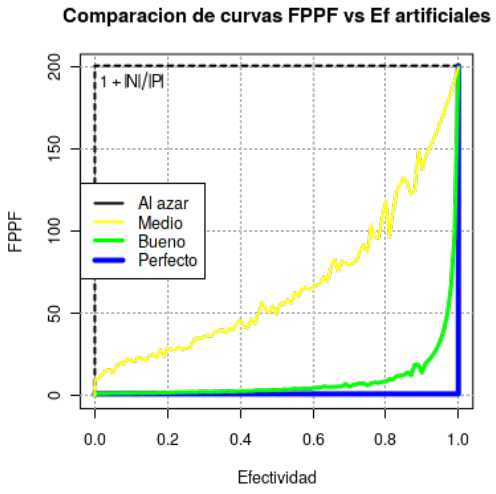


Figura 2.2: Comparación entre curvas de desempeño construidas para cuatro modelos artificiales. La curva punteada representa el azar (esto es el peor modelo posible: el score de cada transacción es un valor uniforme en [0,1]). La curva amarilla corresponde a un modelo de peor desempeño que el asignado a la curva verde. La curva azul representa el modelo perfecto (todos los fraudes tienen score 1 y las legítimas score 0).

[7] $u = 1$ implica no generar ninguna alarma, por lo que no hay detección. $u = 0$ implica alertar todas las transacciones, por lo que se evitan todos los fraudes

[8] La curva que en la literatura se suele llamar ROC es en realidad la gráfica de Efectividad en función de *Review*, pero haremos un abuso de notación y llamaremos por el mismo nombre a su inversa.

Para la curva de $FPPF$ en función de la Efectividad (gráfico 2.2):

- $u = 1$ implica $Ef = 0$ y entonces $FPPF = 0$ (no se generan falsas alarmas).
- A medida que u disminuye y la Efectividad crece, $FPPF$ varía de forma no necesariamente monótona [9]. Cuanto mejor sea el modelo, más bajos valores de $FPPF$ y por lo tanto, menor el área de la curva.
- $u = 0$ implica $Ef = 1$. Como se alertan todas las transacciones, $FP = N$ (toda transacción legítima genera una falsa alarma), $TP = P$ (todo fraude es detectado correctamente) y entonces $FPPF = 1 + \frac{|N|}{|P|}$, que es la dilución de la base de datos utilizada (según notación introducida en la sección 2.2 y usando la ecuación (2.2)).

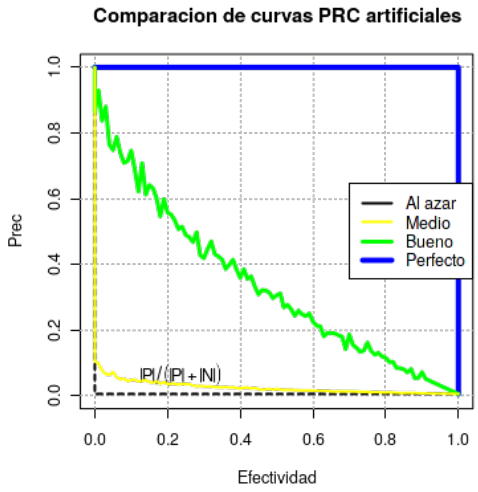


Figura 2.3: Comparación entre curvas PRC construidas para cuatro modelos artificiales. Los colores de las curvas obedecen al mismo código que el gráfico anterior.

Para la curva PRC (gráfico 2.3):

- $u = 1$ implica $Ef = 0$ y entonces $Prec = 0$ (no se generan falsas alarmas).
- A medida que u disminuye y la Efectividad crece, la $Prec$ varía de forma no necesariamente monótona [10]. Cuanto mejor sea el modelo, más altos valores de $Prec$ y por lo tanto, mayor el área de la curva.

[9] En este ejemplo se muestran curvas crecientes para lograr una comparación más simple entre las mismas. Más adelante veremos ejemplos con datos reales donde esto no sucede.

[10] Idem que en el caso anterior.

- $u = 0$ implica $Ef = 1$. Como se alertan todas las transacciones, $FP = N$ (toda transacción legítima genera una falsa alarma), $TP = P$ (todo fraude es detectado correctamente) y entonces $Prec = \frac{|P|}{|P|+|N|}$, que es el porcentaje de fraudes que hay en la base.

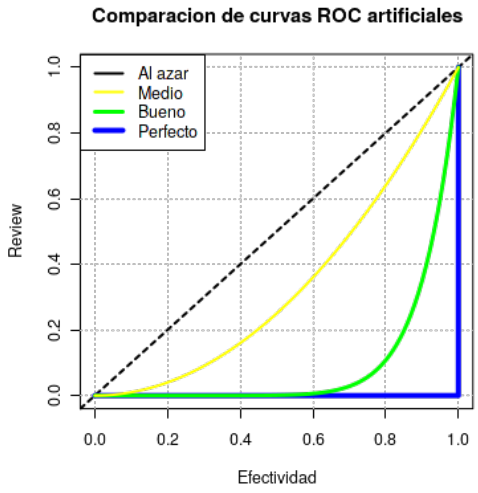


Figura 2.4: Comparación entre curvas ROC construidas para cuatro modelos artificiales. Los colores de las curvas obedecen al mismo código que el gráfico anterior.

Para la curva PRC (gráfico 2.4):

- $u = 1$ implica $Ef = 0$ y entonces $Rev = 0$ (no se generan falsas alarmas).
- A medida que u disminuye y la Efectividad crece, el *Review* no puede disminuir (en la ecuación (2.1), el numerador aumenta y el denominador se mantiene constante).
- $u = 0$ implica $Ef = 1$. Como se alertan todas las transacciones, $Rev = 1$.

Ahora que hemos ilustrado la forma teórica que presentan, discutiremos algunos aspectos de su utilización en la práctica y los motivos por los cuáles , siguiendo la recomendación de la literatura, proponemos dejar de utilizar los Falsos Positivos Por Fraude (como se hacía en el procedimiento anterior). Para comenzar la discusión, en las gráficas 2.5, 2.6 y 2.7 comparamos dos modelos construidos con datos reales usando las tres curvas introducidas anteriormente:

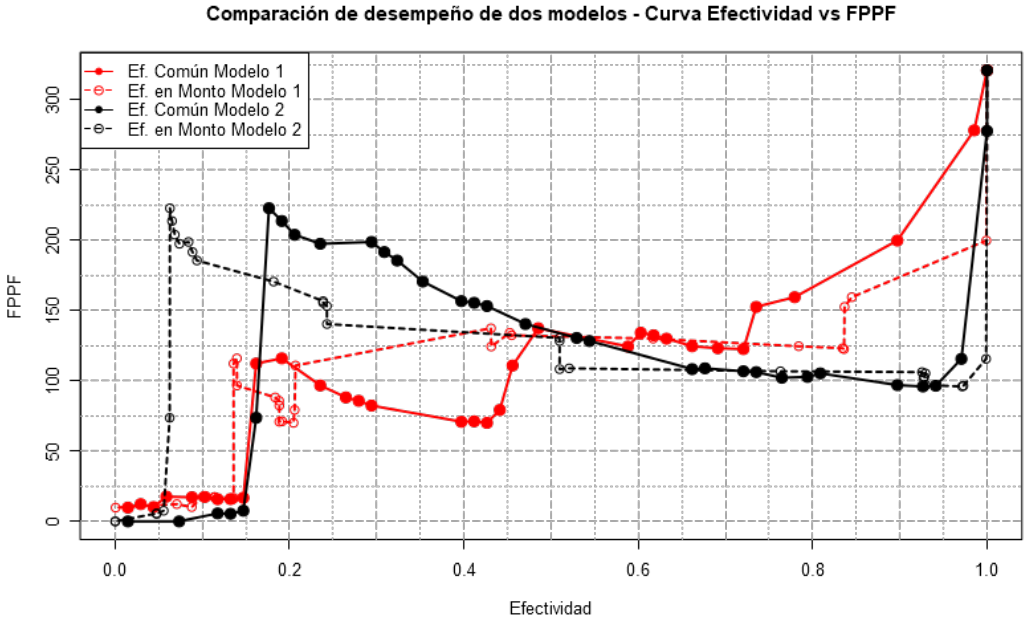


Figura 2.5: Desempeños de dos modelos medidos como la relación entre la efectividad común y la efectividad en monto contra los Falsos Positivos promedio Por Fraude.

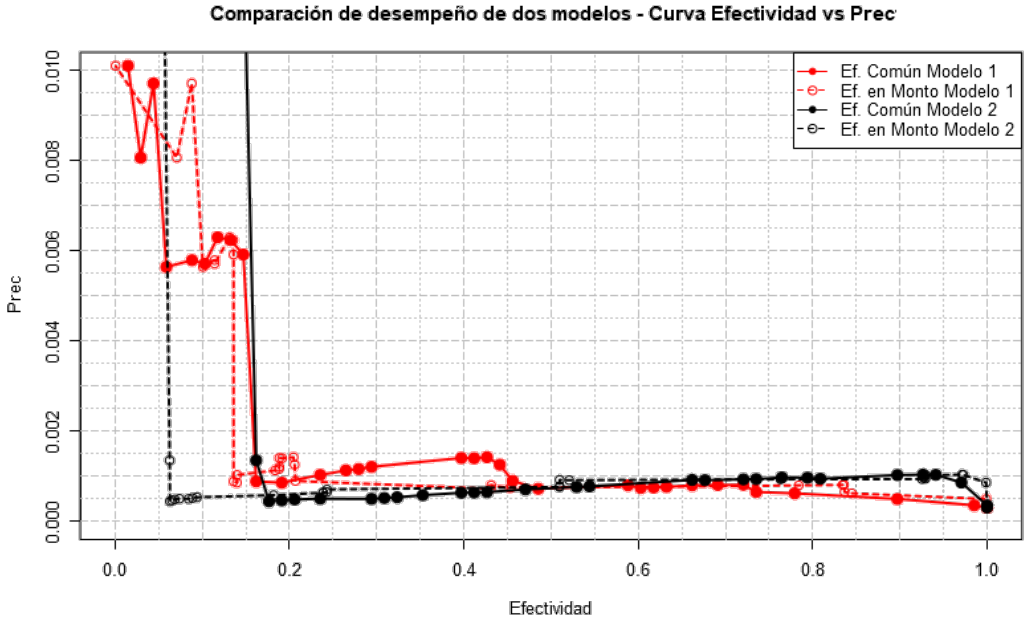


Figura 2.6: Desempeños de los mismos dos modelos de la gráfica 2.5 medidos como la relación entre la efectividad común y la efectividad en monto contra la Precisión.

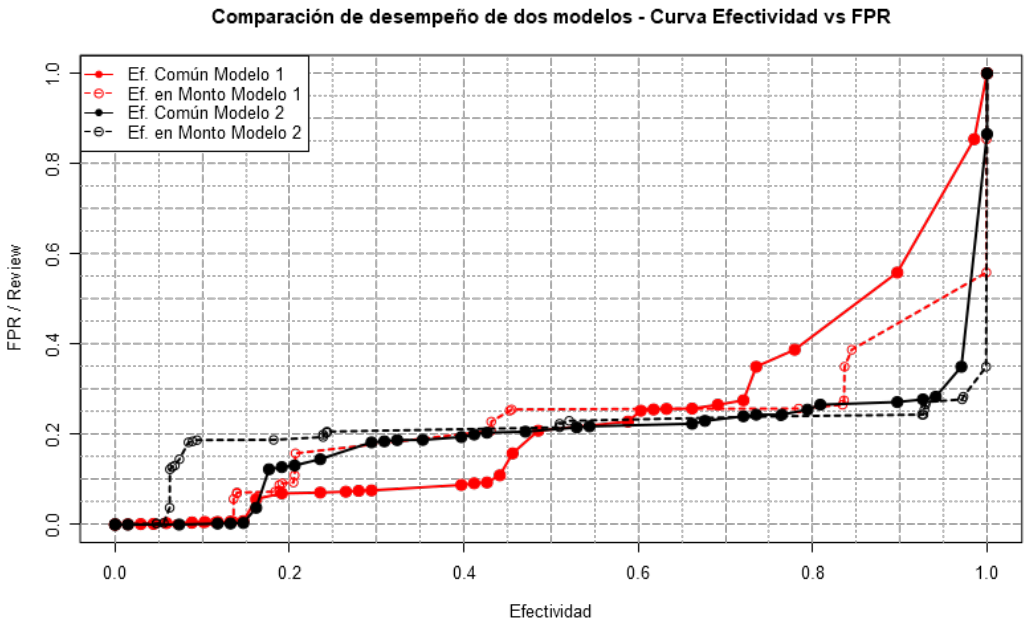


Figura 2.7: Desempeños de los mismos dos modelos de la gráfica 2.5 medidos como la relación entre la efectividad común y la efectividad en monto contra el *False Positive Rate* o *Review*.

Lo primero que debemos mencionar es que, a diferencia del comportamiento ilustrado en las gráficas 2.2 y 2.3, la variación de los $FPPF$ y la Precisión en función de la Efectividad no es necesariamente monótona. Por lo tanto, a menos que una de las curvas domine completamente a la otra, la comparación de modelos debe hacerse manualmente. Por ejemplo: en la gráfica 2.5, el modelo 1 es preferible en efectividades entre 0.15 y 0.5 aproximadamente mientas que el modelo 2 es el mejor en los restantes valores. Sin embargo, cuando tenemos una gran cantidad de modelos a comparar, la exploración manual puede volverse inviable[11]. En estos casos, es usual comparar el área encerrada por cada curva o AUC (*Area Under the Curve*) para evitar la inspección manual. En nuestro contexto en particular, proponemos restringir el área a un intervalo de efectividades de interés, ya que valores muy bajos no son relevantes y valores muy altos casi seguramente provocan que el número de falsas alarmas sea inadmisible.

[11]Esto ocurre con muchísima frecuencia: por ejemplo, cuando se quiere encontrar el valor óptimo de un parámetro de un clasificador, y se prueba con una gran cantidad de valores.

Consideraciones sobre los *FPPF* y la curva PRC vs la curva ROC

Teniendo en cuenta entonces que la mayor parte del tiempo estaremos haciendo comparaciones que no pueden hacerse de forma manual, introduciremos ahora algunas cuestiones que surgen al considerar la AUC para las tres curvas presentadas anteriormente:

1. La escala de la curva Efectividad vs $FPPF$ depende de la dilución del conjunto considerado. Cuando estamos haciendo validación cruzada, cada sorteo puede dar conjuntos con diluciones muy distintas (sobre todo cuando hay pocos fraudes o pocas tarjetas en el conjunto original), por lo que los valores de AUC para esta curva pueden diferir enormemente para el mismo modelo evaluado en conjuntos distintos. [12] Este problema puede solucionarse fácilmente considerando la Precisión o el *Review*, ya que ambas medidas están en el intervalo $[0, 1]$.
2. Los $FPPF$ pueden ser una medida engañosa en contextos con poco fraude. Observemos la gráfica 2.5: puede verse que para el modelo 2, en las efectividades cercanas a 0.15, hay tres puntos consecutivos graficados donde los $FPPF$ valen aproximadamente $10, 75$ y 225. Esto podría inducirnos a pensar que de los tres umbrales u_1, u_2 y u_3 tomados para generar estos puntos, el primero es ampliamente mejor que los otros dos. Sin embargo, tenemos que tener en cuenta que estos umbrales alcanzan efectividades muy bajas, por lo que los $FPPF$ fueron calculados usando una pequeña cantidad de fraudes. Si los tres umbrales son consecutivos, es probable que u_3 capture solo un fraude más que u_2 y dos más que u_1 (en el conjunto de evaluación usado para construir la gráfica). Dado que los tres valores de $FPPF$ se calculan como cocientes y los tres denominadores usados son valores bajos y parecidos, cualquier efecto aleatorio en los numeradores (que es esperable dado que las legítimas son muchas más que los fraudes) puede perturbar los valores de los cocientes.
   Este fenómeno no ocurre con tanta fuerza para las curvas PRC y ROC.
3. En el contexto de detección con clases desbalanceadas, se recomienda utilizar la curva PRC en lugar de ROC. Por ejemplo, en (Saito & Rehmsmeier, 2015) se mencionan casos donde dos modelos (de buen

[12] Esto es especialmente relevante en nuestro caso donde trabajamos con pocos fraudes, ya que la dilución de los conjuntos puede cambiar notoriamente con una pequeña variación en la cantidad de fraudes (por ejemplo, en las distintas *folds* al hacer validación cruzada).

y mal desempeño respectivamente) no son distinguibles según la curva de ROC pero sí por la curva PRC. Sin embargo, la construcción de estos ejemplos se basa siempre en evaluar en dos conjuntos con diluciones distintas. En nuestro caso, si tenemos el cuidado de diseñar nuestros experimentos para siempre comparar desempeños sobre los mismos conjuntos, evitaríamos el problema. Por ejemplo: si estamos buscando el valor óptimo de un parámetro θ de un clasificador y para eso exploramos los valores $\{\theta_1, \ldots, \theta_n\}$, alcanza con que hagamos el sorteo de las k particiones de validación cruzada previamente, y luego para cada valor θ_i utilicemos siempre las mismas k particiones del conjunto de entrenamiento. Con este simple procedimiento, nos aseguramos que nunca vamos a comparar desempeños de dos valores de θ distintos en conjuntos que puedan tener diferente dilución.[13]

4. Adicionalmente, la curva de ROC es monótona creciente, lo que la vuelve más estable que las curva PRC y Efectividad vs *FPPF* en contextos con pocos fraudes (como se ilustra en las gráficas 2.5, 2.6 y 2.7).

Teniendo en cuenta los motivos expuestos, proponemos utilizar el AUC de la curva ROC como medida de desempeño de un modelo cuando realicemos pruebas y optimización de parámetros. Las otras medidas las utilizaremos solamente para reportar resultados o en casos donde sea factible la comparación manual.

Consideraciones sobre la efectividad en monto

La efectividad en monto es una medida extremadamente sensible a valores extremos. Cuando el número de ejemplos de transacciones fraudulentas es reducido, esto puede generar que la estimación de curvas de desempeño sea muy imprecisa, ya que fácilmente puede ocurrir que un sólo fraude del conjunto de evaluación represente un porcentaje alto de las pérdidas. Tomemos el ejemplo de la figura 2.8.

Para el mismo, se formo un conjunto V sorteando 50 tarjetas con fraudes del conjunto de entrenamiento de la base 1.1. Luego, se hicieron dos conjuntos T_1 y T_2 mediante dos sorteos de 10.000 tarjetas del conjunto de entrenamiento.

[13] Al hacer validación cruzada, puede ser que la dilución de cada *fold* sea distinta, pero si cada θ se evalúa en exactamente las mismas *folds*, la comparación es equitativa entre los desempeños de cada elección de θ en cada *fold*.

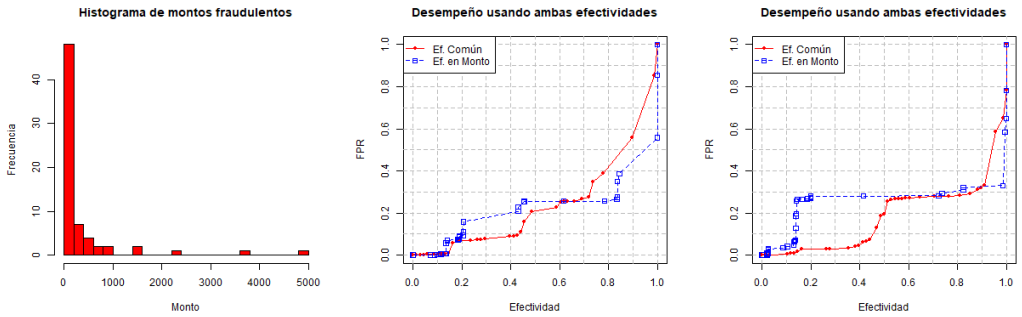


Figura 2.8: Comparación de desempeños de dos modelos en un conjunto con pocos fraudes usando efectividad común y en monto.

En la gráfica de la izquierda, se muestra el histograma de los montos de los fraudes en el primer conjunto. En las dos gráficas siguientes, se muestran las curvas de desempeño de un Random Forest de 2 árboles entrenado en T_1 y T_2 respectivamente y evaluado en V en ambos casos. Se grafican dos curvas por cada modelo: una de FPR contra Efectividad y otra de FPR contra Efectividad en monto.

El conjunto V de transacciones sobre el que se evalúan los dos modelos de las gráficas tiene solo 68 fraudes. Los tres valores atípicos más grandes en el histograma de la izquierda representan el 21 %, 16 % y 10 % del monto total respectivamente. Llamemos ahora s_1, s_2 y s_3 a los scores de las transacciones que toman estos valores extremos. Cuando se quiere generar la curva de FPR vs Efectividad en monto usando el umbral $u_1 = s_1$, ocurre que necesariamente, el punto correspondiente a este umbral tiene que dar un "salto" de al menos 0.21 hacia la derecha con respecto al punto anterior de la curva, pues al pasar de un umbral u (inmediatamente anterior a u_1) a u_1, la transacción con score s_1 pasa a ser detectada y las perdidas evitadas aumentan súbitamente en un 21 % (o más, si hay más fraudes con score exactamente igual a s_1). Si se hubiera utilizado la efectividad común, los saltos son menos abruptos porque cada fraude detectado aumenta en la misma cantidad ($\frac{1}{68} \simeq 1,4\,\%$ en este caso) la efectividad asociada a un umbral. Esto se ve claramente en las dos gráficas de desempeño anteriores.

Nuestra propuesta frente a este problema es usar la efectividad común durante la construcción de los modelos y dejar la evaluación de la efectividad en monto para la presentación de los resultados finales.

2.4.2. Partición en conjuntos

Anteriormente, el particionado para determinar los conjuntos de entrenamiento, validación y testing no se hacía aleatoriamente, sino que era de acuerdo a la fecha. Esto es: el conjunto de entrenamiento se armaba con el primer 60 % de las transacciones más antiguas, el conjunto de validación con el 20 % posterior y finalmente el 20 % más nuevo se utilizaba para testing.

El mayor problema que trae realizar un particionado de esta manera (o de cualquier otra manera que no contemple el número de tarjeta) es que se están incluyendo instancias de una misma tarjeta en conjuntos diferentes. Esto genera que los conjuntos no sean independientes, ya que transacciones de la misma tarjeta están altamente correlacionadas[14]. Utilizar acumuladores solo contribuye a agravar esta situación, dado que los mismos usan información de transacciones pasadas, que podrían estar en un conjunto diferente al de la transacción sobre la que se está calculando.

Como consecuencia, los resultados obtenidos por los modelos entrenados bajo estas particiones fueron excesivamente optimistas, pues las transacciones que tenían que clasificar en testing estaban correlacionadas con información ya vista por el clasificador en el conjunto de entrenamiento. Por este motivo es que no compararemos los resultados finales obtenidos por nuestra metodología con los obtenidos utilizando el procedimiento anterior.

Nuestra propuesta es entonces utilizar un sorteo aleatorio *por tarjeta* para determinar los conjuntos. Esto es: cada tarjeta es asignada aleatoriamente a uno de los tres conjuntos (manteniendo las proporciones 60 %, 20 % y 20 %) y luego las transacciones van a parar al conjunto que les corresponda según su número de tarjeta.

2.4.3. Uso de acumuladores

Un acumulador es una característica nueva Y cuyo valor y en una transacción genérica $\vec{t}_{n+1}$ se calcula aplicando una función sobre los valores de una característica X en un subconjunto $A(\vec{t}_{n+1})$ de transacciones del histórico $T(\vec{t}_{n+1})$ de $\vec{t}_{n+1}$.

[14]Por ejemplo, la suma de montos de ambas transacciones no puede superar el límite de crédito si fueron hechas en el mismo mes.

- **Variable de acumulación:** característica X_i continua cuyos valores en el subconjunto $A(\vec{t}_{n+1})$ se usan para armar el vector de acumulación v.
- **Función de acumulación:** la función f que se debe aplicar sobre el vector v de valores a acumular. Las funciones que soporta *RiskCenter* son `count` (longitud del vector de acumulación), `sum` (suma), `mean` (promedio), `min` (mínimo), `max` (máximo) y `sd` (desviación estándar muestral).
- **Ventana de tiempo:** si la ventana de tiempo es de V unidades de tiempo, las transacciones que pertenezcan a $A(\vec{t}_{n+1})$ deben tener una diferencia de V unidades de tiempo o menos con $\vec{t}_{n+1}$.
- **Concepto:** si se usa una característica X_j como concepto, esto quiere decir que las transacciones $\vec{t}_i$ que pertenezcan a $A(\vec{t}_{n+1})$ deben tener el mismo valor que $\vec{t}_{n+1}$ en la coordenada j-ésima. Esta opción puede usarse por ejemplo para acumular solamente en transacciones que hayan sido hechas en el mismo país.
- **Filtro:** si se usa una característica X_h como filtro con valor H, esto quiere decir que las transacciones $\vec{t}_i$ que pertenezcan a $A(\vec{t}_{n+1})$ deben cumplir $(\vec{t}_i)_h = H$. Esta opción puede usarse por ejemplo para acumular solamente transacciones que pertenezcan a un MCC riesgoso.

Como veremos en la sección 2.5, los acumuladores son una de las pocas técnicas cuya utilización aparece repetidamente en los artículos del área ((Whitrow et al., 2009), (Jha et al., 2012), (Bhattacharyya et al., 2011) and (Van Vlasselaer et al., 2015)). Por este motivo, y dado que *RiskCenter* es capaz de realizar estos cálculos de forma eficiente y en línea es que tradicionalmente se utilizó esta técnica para enriquecer el conjunto de características para entrenar un modelo.

Sin embargo, como observamos en la sección 2.3.1, el monto de una transacción es la única variable continua que podríamos usar para acumular. Por lo tanto, aunque las características distintas que podemos crear son numerosísimas (variando los otros cuatro parámetros fuera de la variable de acumulación), todas ellas están altamente correlacionadas, pues dada una transacción $\vec{t}_{n+1}$, todo acumulador será siempre una de las 6 funciones posibles aplicada sobre un vector de acumulación cuyos valores son siempre los mismos (los montos de las transacciones anteriores a $\vec{t}_{n+1}$), aunque descartemos algunos de ellos según las ventanas, filtros o conceptos. Este fenómeno explica por qué

la utilización de muchos acumuladores aumenta la tendencia al sobreajuste, ya que se aumenta la redundancia del conjunto de características que se usa para entrenar. A continuación mostramos los resultados de un experimento que ilustran esta afirmación:

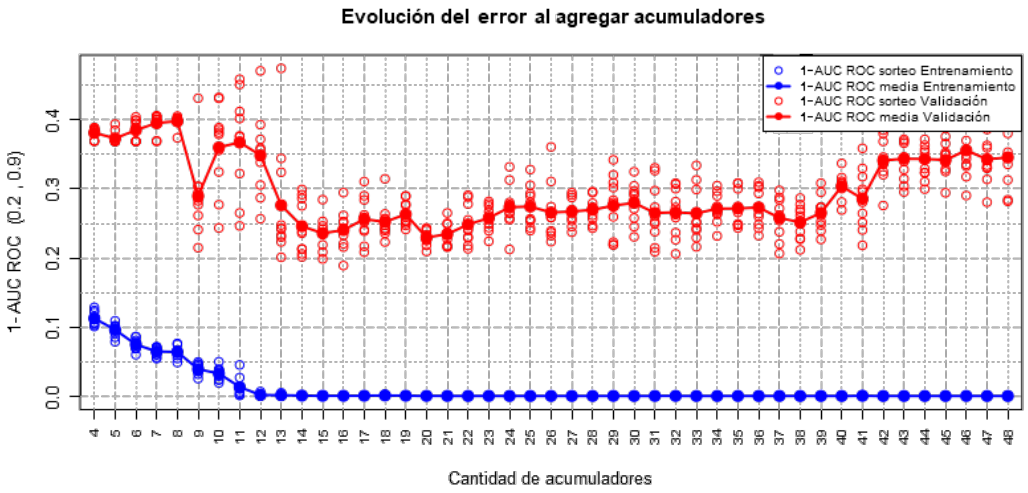


Figura 2.9: Evolución del error promedio en entrenamiento y en validación al aumentar la cantidad de acumuladores. Para este experimento, se calcularon 48 acumuladores en los conjuntos de entrenamiento y validación de la base 1.1 probando todas las combinaciones posibles de un conjunto de funciones de acumulación, ventanas de tiempo, conceptos y filtros. Luego, se particionó el conjunto de entrenamiento en 10 folds y en cada una se entrenó un Random Forest de 2 árboles. El error se midió evaluando cada uno de estos modelos en el conjunto de validación usando el área encerrada por la curva inversa a la ROC restringida a efectividades en el intervalo $(0.2; 0.9)$.

Como se puede ver en la figura 2.9, el desempeño óptimo en validación se obtiene con una pequeña cantidad de acumuladores (unos 12 aproximadamente)[15]. En algoritmos de clasificación basados en árboles (como los que usamos en nuestro problema), si en la construcción de un nodo se elige la mejor variable de un subconjunto aleatorio (lo que es la práctica usual), podríamos confundir al algoritmo al agregar demasiadas características redundantes ya que aumenta la probabilidad de que el subconjunto aleatorio sorteado no contenga la variable óptima para generar el nodo.

En conclusión, dado que se pretende que el número de acumuladores sea bajo, proponemos no realizar un *backwards selection* como método de selección de acumuladores sino que un *forward selection* donde se busque no solo incluir variables con alta relación con la etiqueta de fraude sino que también con baja redundancia con las ya incluidas. En definitiva, un pseudo-código del algoritmo propuesto es el siguiente:

Algoritmo 1 Inclusión secuencial de acumuladores

function InclusiónAcumuladores($\rho_{\text{máx}}, \rho_{\text{mín}}, \tilde{X}_1, \ldots, \tilde{X}_h, Z_1, \ldots, Z_k$)
  $A = \{\tilde{X}_1, \ldots, \tilde{X}_h\}$ ▷ Acumuladores discretizados.
  $B = \{Z_1, \ldots, Z_k\}$ ▷ Variables óptimas al momento.
  while $(\rho_1 < \rho_{\text{máx}})$ & $(\rho_2 > \rho_{\text{mín}})$ **do**
    $\rho_{1j} = \text{máx}\left\{ \dfrac{I(X_j, Z)}{\text{mín}\{H(X_j), H(Z)\}} : Z \in B \right\} \ \forall j : X_j \in A$
    $\rho_{2j} = \dfrac{I(X_j, Y)}{H(Y)}, \ \forall j : X_j \in A$
    $i = \text{argmax}_{\{j : X_j \in A\}}\{(1 - \rho_{1j}) \times \rho_{2j}\}$
    $\rho_1 = \rho_{1i}$; $\rho_2 = \rho_{2i}$;

    if $(\rho_1 < \rho_{\text{máx}})$ & $(\rho_2 > \rho_{\text{mín}})$ **then** ▷ Si el óptimo cumple las restricciones...
      $B = A \setminus \{X_i\}$ ▷ ... saco la variable del conjunto a evaluar...
      $B = B \cup \{X_i\}$ ▷ ...y la agrego al conjunto de las que usaremos.
    else ▷ Si la variable óptima no cumple las restricciones...
      $\rho_1 = 1$; $\rho_2 = 0$ ▷ ...parar la ejecución.
    end if
  end while
end function

Observando que:

[15]Tengamos en cuenta que para este experimento no se utilizaron variables que no fueran acumuladores. En la práctica, al incluir otras características que también podrían tener alta redundancia con los mismos, el número óptimo a incluir puede ser todavía menor.

- Y es la etiqueta de fraude.
- X_j son los acumuladores.
- Z_h son las variables óptimas al momento de la ejecución.
- $H(X)$ es la entropía de la variable aleatoria X.
- $I(X, Y)$ es la información mutua entre las variables X e Y.
- $\tilde{X}_j$ representa a la discretización del acumulador X_j en n intervalos generados con puntos de corte iguales a los cuantiles de nivel $\frac{i}{n}, i = 0, \ldots, n$. La ventaja de usar variables discretas es que permite usar la información mutua como medida de correlación, la cual es fácil de calcular y no está sesgada por la escala en que estén las variables. Además, al discretizar por cuantiles, todas las variables resultado tienden a uniformizarse, por lo que es más equitativa su comparación con la etiqueta de fraude.
- ρ_{1j} es el máximo de la información mutua normalizada entre el acumulador X_j y todas las variables Z ya incluidas en el modelo. Por propiedades de la información mutua, cada término evaluado en la maximización está entre $[0, 1]$ y puede interpretarse de manera que valores cercanos a 1 indican alta correlación entre las variables.
- ρ_{2j} es la información mutua entre el acumulador X_j y la etiqueta de fraude, normalizada por la entropía de esta última para que se cumpla que $\rho_{2j} \in [0, 1]$ (no se normaliza por $H(X)$ para evitar que una variable con entropía muy cercana a 0 obtenga un valor alto de ρ_{2j}).

2.4.4. Determinación de "cortes"

Llamamos *cortes* a cada uno de los subconjuntos obtenidos al dividir la base de datos con algún criterio dependiente de los valores de las columnas, de manera que se entrena un modelo distinto en cada subconjunto. Generalmente la utilización de corte obedece a decisiones comerciales o peticiones del cliente: por ejemplo, es usual que las instituciones financieras soliciten modelos distintos para cada uno de sus productos o emisores. En la práctica, usar cortes puede mejorar el poder de clasificación. Esto es intuitivo si tenemos en cuenta que las realidades pueden ser muy distintas dentro de una misma base de datos, por lo que al permitir que los algoritmos de clasificación aprendan sobre un conjunto más restringido de datos, es más fácil que los mismos puedan distinguir patrones más específicos y de hecho, es posible que las decisiones

de diseño óptimas en cada subconjunto varíen significativamente. Un ejemplo sencillo es el siguiente:

Corte 1: modo de entrada manual

- La variable "indicador de PIN" siempre vale 0 por lo que deja de ser significativa.
- Hay mucha mayor proporción de fraudes, por lo que suelen ser necesarios modelos menos complejos (por ejemplo: en un Random Forest, menos árboles de menor profundidad).
- Por lo anterior, suele convenir elegir umbrales que alcancen alta efectividad, aunque eso conlleve no llegar a una tan alta precisión.

Corte 2: modo de entrada banda

- La variable "indicador de PIN" es sumamente útil para descartar legítimas (casi no ocurren fraudes cuando toma el valor 1)
- Los fraudes son mucho menos frecuentes, por lo que suelen ser necesarios modelos más complejos (por ejemplo: en un Random Forest, más árboles de mayor profundidad).
- Por lo anterior, suele convenir elegir umbrales que tengan alta precisión para no disparar el número de falsas alarmas total.

Desde un punto de vista algorítmico, definir cortes es equivalente a construir un árbol de decisión con múltiples nodos (uno por cada corte) y al final de cada uno de ellos agregar un modelo diferente (entrenado para reconocer fraudes en esa realidad particular). En este sentido, determinar variables que satisfagan condiciones como la de los primeros dos puntos en el ejemplo anterior es un problema que podría resolverse automáticamente (alcanza con estudiar la distribución de la etiqueta de fraudes condicionada a cada variable). El punto tres sin embargo está ligado al desempeño de los modelos, por lo que debe hacerse de forma manual (pues como ya vimos, debemos reportar una curva PRC de un modelo y no solamente un umbral óptimo).

Inicialmente, los cortes se determinaban manualmente u obedeciendo pedidos de cada entidad. Como mejora, proponemos un algoritmo básico que (teniendo en cuenta las observaciones anteriores) explora las distribuciones de cada variable de una base y sugiere cortes según un valor de la variable que genere un cambio significativo en la distribución condicionada de la etiqueta

de fraude: si se encuentra que en cierto subconjunto ocurren muchos fraudes y pocas legítimas, se sugiere crear un corte en ese subconjunto. Por otro lado, si se encuentra que en el mismo ocurren muchas transacciones legítimas y pocos fraudes, se sugiere descartar ese subconjunto. En detalle, el algoritmo de sugerencias es el siguiente:

Dados los parámetros $N_1, N_2, \alpha_3, \alpha_4, \alpha_5, N_6$ y N_7.
Sean F y L la cantidad total de fraudes y legítimas en la base definidas como:

$$F = \left|\{\vec{t} \in D : y = 1\}\right|, \quad L = \left|\{\vec{t} \in D : y = 0\}\right|$$

Para cada variable X_i nominal y para cada $x_j^i \in \text{Rec}(X_i) = \{x_1^i, \ldots, x_{n_i}^i\}$, sean:

$$F_j^i = \left|\{\vec{t} \in D : y = 1, (\vec{t})_i = x_j^i\}\right|, \quad L_j^i = \left|\{\vec{t} \in D : y = 0, (\vec{t})_i = x_j^i\}\right|$$

Se sugiere hacer un corte de entrenamiento según la condición $X_i = x_j^i$ si se cumple que:

- $F_j^i > N_1$ (cantidad absoluta mínima de fraudes en el corte).
- $L_j^i > N_2$ (cantidad absoluta mínima de transacciones en el corte).
- $F_j^i/F > \alpha_3$ (porcentaje mínimo de fraudes en el corte).
- $L_j^i/F_j^i \times \alpha_4 < L/F$ (el corte tiene una dilución al menos α_4 veces menor a la dilución original).

Se sugiere hacer un corte de descarte según la condición $X_i = x_j^i$ si se cumple que:

- $F_j^i/F > \alpha_5$ (porcentaje máximo de fraudes en el corte).
- $L_j^i > N_6$ (cantidad absoluta mínima de transacciones en el corte).
- $L_j^i/F_j^i > N_7$ (el corte tiene una dilución de al menos α_7 legítimas por fraude).

Cortes de entrenamiento

Olvidémonos por un momento de los cortes de descarte. Como esta implementado actualmente, el algoritmo devuelve un conjunto de sugerencias de particiones de la base. Luego, el usuario puede implementar secuencialmente las condiciones que desee, de manera que el último corte es el que tiene menor prioridad. Esto es:

- Si la primera condición implementada es $X_{i_1} = x_{j_1}^{i_1}$, se generan dos subconjuntos aplicándola en la totalidad de la base (salvo en los cortes de descarte).
- Si la k-ésima condición implementada es $X_{i_k} = x_{j_k}^{i_k}$, se generan dos subconjuntos aplicándola en el conjunto de las transacciones que cumplen $X_{i_{k-1}} \neq x_{j_{k-1}}^{i_{k-1}}, \ldots, X_{i_1} \neq x_{j_1}^{i_1}$

Esto le da una estructura natural de ramificación a los cortes de entrenamiento. Sin embargo, es necesario tener en cuenta que si las sugerencias son dadas sobre el mismo conjunto de datos (por ejemplo, sobre el conjunto entero) entonces podría no ser deseable implementar los cortes sugeridos secuencialmente, pues habría que considerar la interacción entre los mismos. Por ejemplo: que dos cortes tengan suficiente cantidad de fraudes en el conjunto total, no garantiza que esto se siga cumpliendo al aplicar el segundo corte sobre las transacciones que no pertenecen al primero. Más aún, casos como este son sumamente probables teniendo en cuenta la alta correlación que existe entre las variables (como mencionamos en la sección 2.3.1). Por este motivo, proponemos usar el algoritmo de sugerencia de cortes con la siguiente estrategia *greedy*:

1) Ejecutar el algoritmo con la base completa.
2) Elegir el corte sugerido óptimo en este paso.
3) Repetir el paso 1 en cada uno de los cortes determinados hasta que no se sugieran más cortes.

Queda para investigaciones futuras la determinación de una función de costo en el paso 2 para que la determinación del corte óptimo no sea manual (como es hoy en día). Una vez que se pueda automatizar la estrategia *greedy*, como la misma es sub-óptima, sería interesante estudiar la viabilidad de una optimización global que tenga en cuenta todas las posibles secuencias de cortes.

Cortes de descarte

En el caso de los cortes de descarte ya no importa la interacción entre los mismos, pues todas las transacciones que cumplan *alguna* de las condiciones que los definen recibirán el mismo tratamiento a nivel del modelo: no serán usadas para entrenar.

Sin embargo, aunque estas transacciones se asuman prácticamente sin fraudes, pertenecen al histórico de las tarjetas y pueden darnos pistas acerca

de los comportamientos de los clientes. Por lo tanto, debemos distinguir entre dos tipos de transacciones en un corte de descarte (que ejemplificaremos a continuación):

- Ejemplo 1: transacciones con chip. Si bien presentan una concentración de fraudes tan baja que usarlas en el modelo puede generar demasiadas falsas alarmas, las queremos introducir en el cálculo del Score de Outlierness.
- Ejemplo 2: pagos del saldo. Estas transacciones no deberían pertenecer al perfil del cliente, ya que no nos dan información acerca de la manera en que el mismo opera e incluso pueden sesgarlo incorrectamente (por ejemplo: los pagos suelen ser una vez por mes y por montos mucho mayores que las compras individuales).

Esta diferenciación se hace utilizando conocimiento de negocio, por lo que debe ser una decisión manual del usuario. Para que la mayor prioridad la tengan las transacciones del segundo tipo, se les agrupa en el corte con índice -1, mientras que a las transacciones del primer tipo se les asigna el corte 0. De esta manera se mantiene la relación entre la numeración de los cortes y su orden de ejecución en cadena.

2.5. Estado del arte en detección de fraude en tarjetas de crédito

En las etapas iniciales de la investigación se realizó una búsqueda bibliográfica para entender cual es el estado del arte en el área. En este proceso, nos encontramos con que las publicaciones de acceso libre son pocas y casi siempre realizadas por investigadores de universidades, ya que las empresas del ramo no quieren divulgar sus procedimientos. Esta reserva también se ve en el manejo de los datos: son varios los artículos que citan las dificultades que presentan para hallar bases y que en consecuencia recurren a datos artificiales o de acceso público ((Lei & Ghorbani, 2012)). Incluso en los casos que se usan datos reales, las bases tienen un desbalance considerablemente menos pronunciado que el nuestro ((Dal Pozzolo et al., 2014), (von Matt & Dacunha-Castelle, 2014), (Jha et al., 2012)) o una cantidad de transacciones mucho

menor ((Mahmoudi & Duman, 2015)) por lo que tenemos menor confianza en que los resultados publicados sean extrapolables a nuestro contexto.

Las aproximaciones supervisadas tratan al problema como uno de clasificación en dos clases, y como tal, en la mayoría de los casos es abordado de la manera tradicional: procesando los datos, construyendo características y entrenando un clasificador. No observamos sin embargo que exista un algoritmo de clasificación que se prefiera sobre el resto: lo más usual es que se pruebe con regresiones lineales, SVM, árboles de decisión, redes neuronales y/o Random Forest para quedarse con el que presente mejor desempeño (generalmente es alguno de los últimos dos (Dal Pozzolo et al., 2014), (West & Bhattacharya, 2016), (von Matt & Dacunha-Castelle, 2014)).

Una técnica que sí se repite en una gran cantidad de artículos ((Whitrow et al., 2009), (Jha et al., 2012), (Bhattacharyya et al., 2011), (Van Vlasselaer et al., 2015)) es la utilización de acumuladores (o *agregación de transacciones*) como método de extracción de características. En (Bahnsen et al., 2015) y (Bahnsen et al., 2016) se propone una técnica similar que permitiría agregar información (acumular) con variables temporales, que tienen la particularidad de presentar una métrica circular.

En (Van Vlasselaer et al., 2015) se propone un método de extracción de características basado en análisis de redes: mediante la construcción de un grafo que relacione las tarjetas con los comercios en que se opera, se propone ejecutar un algoritmo de propagación para finalmente obtener *scores de exposición* del comercio, tarjeta-habiente y transacción que puedan ser usados como características.

Otra familia de métodos serían aquellos que piensan al problema como uno de detección de anomalías, teniendo en cuenta que los fraudes son casos raros y se espera que se desvíen del comportamiento normal. En este sentido, las aproximaciones se enfocan en modelar un comportamiento típico y determinar la distancia de una transacción al mismo. Se usan aproximaciones muy diversas, como por ejemplo sistemas inmunitarios artificiales (Halvaiee & Akbari, 2014), clustering (Daneshpazhouh & Sami, 2014), (Lei & Ghorbani, 2012), Análisis de Componentes Principales (Chandola et al., 2007) (o PCA por sus siglas en inglés) o detectores de outliers basados en la entropía (Daneshpazhouh & Sami, 2014). En algunos casos, se combinan aproximaciones supervisadas

y no supervisadas para obtener el modelo final (Duman & Ozcelik, 2011), Van Vlasselaer et al. (2015).

El problema generado por el desbalance de clases es tratado de varias maneras distintas. En algunos casos se realiza un sub-muestreo de las transacciones legítimas Duman & Ozcelik (2011) mientras que en otros se utilizan técnicas como SMOTE (Chawla et al., 2002) para generar fraudes artificiales Dal Pozzolo et al. (2014). En (Bahnsen et al., 2015) y (Bahnsen et al., 2016) los resultados son presentados en un subconjunto de menor tamaño pero con mayor proporción de fraude, lo cual podría pensarse como otra manera de reducir el desbalance (si bien no se explicita como se trata al conjunto restante).

Otra aproximación que mitiga el desbalance sin modificar los datos es incorporar matrices de costo en los algoritmos de clasificación (Mahmoudi & Duman, 2015). Esto además tiene la utilidad adicional de inclinar los clasificadores a que enfoquen su atención en los fraudes más caros. En estos casos, el desempeño de los modelos es también evaluado con métricas adaptadas para que se tengan en cuenta las pérdidas monetarias evitadas y no solo la cantidad de fraudes detectados (von Matt & Dacunha-Castelle, 2014) (Sahin et al., 2013).

2.5.1. Conclusiones del estado del arte

Aquí presentamos un análisis crítico de la revisión bibliográfica y las conclusiones a las que llegamos sobre la aplicabilidad de las técnicas vistas a nuestro problema en particular:

- Teniendo en cuenta que ya existen implementaciones potentes y estables de algoritmos de clasificación en `R` (y en otros lenguajes), si quisiéramos optimizar el clasificador, podríamos rápidamente probar con varios algoritmos hasta encontrar el que de los mejores resultados (nuestra elección por defecto será *XGBoost*, definido en la sección 2.1). Por lo tanto, decidimos no poner el foco de la investigación en los clasificadores sino que en el proceso de extracción de características, que es una etapa crítica del desarrollo de modelos de detección donde existen menos estándares que guíen las decisiones de diseño y depende mucho del contexto y tipo de datos.

- El uso de acumuladores como método de extracción de características, si bien encontramos que es extendido en el área, presenta las limitantes que analizamos en detalle en la sección 2.4.3. Utilizar PCA sería otra alternativa (teniendo en cuenta además su relación con detección de outliers) pero decidimos no hacerlo en una primera instancia dado que las variables con las que contamos son en su gran mayoría nominales y el algoritmo está diseñado para ser usado con variables continuas[16].
- Las medidas de desempeño que propusimos en 2.4.1 están en acuerdo con lo propuesto en la literatura, pues tienen en cuenta las pérdidas monetarias. Una alternativa interesante que en un futuro podríamos explorar es la utilización de funciones de costo para el entrenamiento.
- En las etapas previas al comienzo de este proyecto, ya habíamos explorado el uso de sub-muestreo de legítimas o sobre-muestreo de fraudes sin resultados positivos en nuestras bases. Afortunadamente, podemos usar cortes (tal como explicamos en 2.4.4) como una alternativa a las técnicas de muestreo, ya que al trabajar en conjuntos más reducidos podemos disminuir el desbalance o la cantidad total de transacciones (para que el conjunto se vuelva más manejable).

Teniendo en cuenta las observaciones anteriores, decidimos comenzar la investigación explorando técnicas de detección de anomalías para modelar la normalidad de los clientes. En el próximo capítulo estudiaremos en detalle la metodología que desarrollamos basándonos en estas ideas.

2.6. Resumen del análisis inicial

En este capítulo hemos presentado el contexto inicial del problema así como los datos con los que trabajamos. Analizando el proceso anterior de creación de modelos para determinar que prácticas podían ser mejoradas, se decidió:

- Crear los conjuntos de entrenamiento, validación y testing mediante un sorteo aleatorio a nivel de tarjetas con las proporciones 60 %, 20 % y 20 %.
- Mantener el uso de cortes y determinarlos mediante el algoritmo propuesto.

[16]Probamos de aplicar PCA sobre el espacio formado por los acumuladores (que son variables continuas), pero el aporte del algoritmo no fue significativo, probablemente por la alta correlación entre las columnas.

- Mantener el uso de acumuladores pero seleccionarlos utilizando un algoritmo de *forward selection*, vigilando que se minimice la redundancia y se maximice el poder descriptivo.
- Utilizar ROC/AUC como medida de desempeño de modelos al momento de realizar exploración de parámetros y utilizar la curva PRC solamente para presentar los resultados en testing.

Del estudio del estado del arte surgió la posibilidad de reemplazar Random Forest por XGBoost como clasificador preferido. No se llegó a encontrar un método extendido que siempre de mejores resultados en el contexto de prevención de fraudes, aunque surgió la iniciativa de aplicar técnicas de detección de anomalías a nuestro problema.

Esto concluiría el trabajo inicial de investigación. Nos dedicaremos en los capítulos siguientes a explicar detalladamente como se llevó a cabo la segunda etapa de la investigación, en la que realizamos una propuesta nueva que no se basa en el procedimiento anterior con el objetivo de mejorar la detección usando la idea de generar perfiles de clientes.

Capítulo 3

Definición de un Score de Outlierness

El objetivo de este capítulo es presentar detalladamente la metodología de cálculo de variables que llamamos Score de Outlierness y que constituye nuestro principal aporte a la detección de fraudes.

3.1. Hacia la creación de un perfil de cliente

Tal como expusimos en el capítulo 2, las críticas al procedimiento anterior de creación de modelos fueron una de las primeras pistas para saber hacia que dirección ir al principio de la investigación. Uno de los problemas detectados más conocido por los analistas era la imposibilidad de crear perfiles de clientes, que es uno de los requerimientos que más se escucha de parte de las propias instituciones financieras pero que los modelos de antes solo lograban satisfacer parcialmente. Efectivamente, si bien el uso de acumuladores y la creación de cortes son aproximaciones en esta dirección, ambas son limitadas.

La determinación de perfiles de clientes es un problema que puede ser visto desde diversas ópticas. En un principio, decidimos concentrarnos en utilizarlos como ayuda para la detección: la idea motivadora es que, si tenemos bien caracterizado el comportamiento de un cliente, deberíamos poder detectar fácilmente cuando una transacción no ha sido efectuada por la propia persona, que en definitiva, es como se dan la gran mayoría de los fraudes en tarjeta de crédito. Tratando de llevar estas ideas a una proposición lo más general posible, podríamos formular la siguiente hipótesis:

Las transacciones fraudulentas deberían tener mayor tendencia a ser distintas al comportamiento habitual del cliente

Intuitivamente, esto parece razonable. En términos matemáticos, lo que estamos diciendo es que si logramos caracterizar una distribución del comportamiento transaccional del cliente, los fraudes deberían ser *outliers* de la misma. Esta manera de pensar el problema implica un cambio de óptica: ya no pensamos en clasificación supervisada, sino más bien en detección de anomalías no supervisada: podemos crear los perfiles solo mirando los datos de cada cliente y luego usar la etiqueta de fraude para contrastar la hipótesis. El hecho de concentrarnos en el histórico de cada cliente y no en la totalidad nos dice que estamos tratando el problema como uno de detección de *outliers contextuales* (en lugar de *globales*), lo cual tiene sentido: lo que para un cliente puede ser un comportamiento extraño (y por lo tanto sospecha de fraude) puede ser completamente normal para otro.

3.2. Una primera aproximación

Consideremos en primer lugar el caso más sencillo: sean $X_1, \ldots, X_k$ los atributos de la base de datos y $T(\vec{t}_{n+1}) = \left\{\vec{t}_i = (x_1^i, \ldots, x_k^i) : i = 1, \ldots, k\right\}$ el histórico previo de una transacción genérica $\vec{t}_{n+1}$. Supondremos el caso en que n (la cantidad de transacciones en el histórico) es lo suficientemente grande como para realizar estimaciones estadísticamente significativas de las distribuciones de las X_i (en el próximo capítulo discutiremos el caso contrario). Dos de las maneras más sencillas para determinar en este contexto si $\vec{t}_{n+1}$ es anómala respecto a su histórico $T(\vec{t}_{n+1})$ podrían ser:

1. Calcular el valor en $\vec{t}_{n+1}$ de una distribución estimada con los valores de $T(\vec{t}_{n+1})$.
2. Hallar la distancia al vecino más cercano de $\vec{t}_{n+1}$ en el conjunto $T(\vec{t}_{n+1})$.

Ambos métodos son fáciles de modelar y además se complementan, como puede verse en el ejemplo ilustrado en la gráfica 3.1.

En este caso artificial que construimos, podemos ver fácilmente que la transacción $\vec{t}_1$ (azul) sería detectada como un outlier por cualquiera de los dos métodos. Por otro lado, si la transacción $\vec{t}_2$ (roja) corresponde a un comportamiento legítimo, podría ocurrir que su bajo valor según la distribución

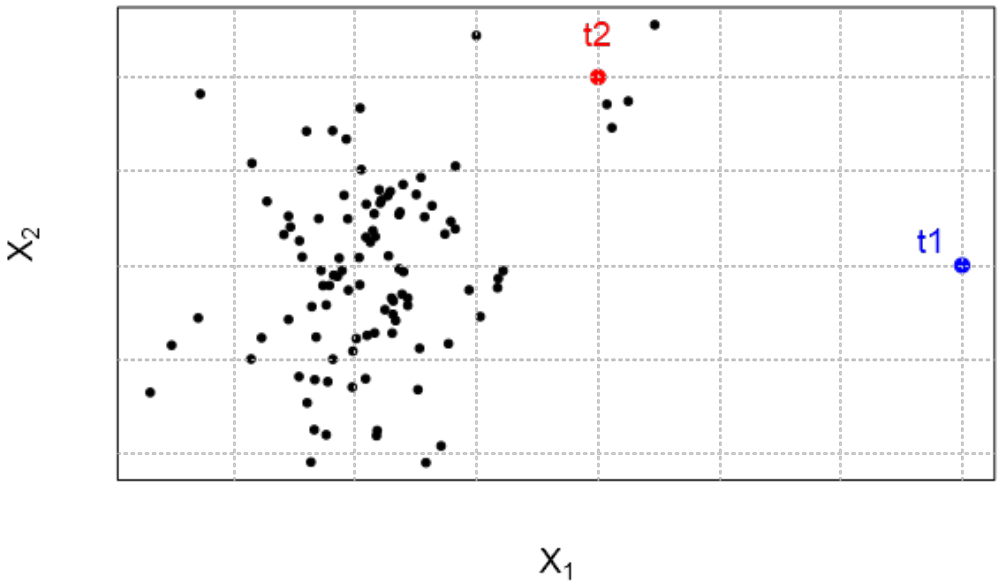


Figura 3.1: Distribución de transacciones de $T(\vec{t}_{n+1})$ respecto a dos variables X_1 y X_2 continuas. En azul se marca la transacción $\vec{t}_1$ y en rojo la transacción $\vec{t}_2$

provoque que sea alertada como una anomalía. En ese caso, la distancia al vecino más cercano podría ayudarnos a detectar que en realidad es parecida a comportamientos legítimos previos. De esta manera, entendemos que la distancia al vecino más cercano es una buena evidencia para complementar la distribución.

Teniendo en cuenta lo anterior, y como un primer modelo sencillo que iremos refinando, nos propondremos definir una función w (que llamaremos ***función de rareza***) y una distancia d tales que la cantidad

$$SO(\vec{t}_{n+1}) = w(\vec{t}_{n+1}) \times \min_{i=1,\dots,n} \left\{ d(\vec{t}_{n+1}, \vec{t}_i) \right\} \tag{3.1}$$

sea una buena medida de "*outlierness*" de la transacción $\vec{t}_{n+1}$ con respecto a $T(\vec{t}_{n+1})$ (donde w es una función que depende de la distribución aprendida de los datos del histórico $T(\vec{t}_{n+1})$).

3.2.1. Definición de Rareza

En el caso particular de las bases que nos han sido presentadas, ocurre que la mayoría de las características X_i son nominales, lo cual simplifica enormemente el problema de estimar su distribución pues podemos sencillamente usar histogramas. Además, toda variable continua Y puede ser discretizada en una variable nominal X usando intervalos I_j y definiendo $X = \sum_j j \times \mathbb{1}(Y \in I_j)$ Por lo tanto, nos abocaremos al caso en que todas las X_i son variables nominales. El problema de hallar el criterio óptimo para la discretización de las variables continuas será tratado en el siguiente capítulo.

Tenemos entonces variables $X_i,\ i = 1, \dots k$ discretas, cada una con su histograma calculado en el histórico T de una tarjeta. Si para cada histograma definimos una función w^i entonces podemos determinar w como $w = \sum_{i=1}^{k} w^i$. Esta definición nos permite trabajar con un solo histograma por vez. En el caso en que queramos considerar el histograma conjunto para las variables X_j y X_h se reduce al anterior considerando $\tilde{X} = (X_j, X_h)$.

A continuación mostraremos paso a paso la deducción de cómo llegamos a una expresión para las funciones de rareza por coordenada w^i que cumpla propiedades deseables. Consideremos tres variables X_1, X_2 y X_3 nominales cuyos histogramas son los que se presentan a continuación:

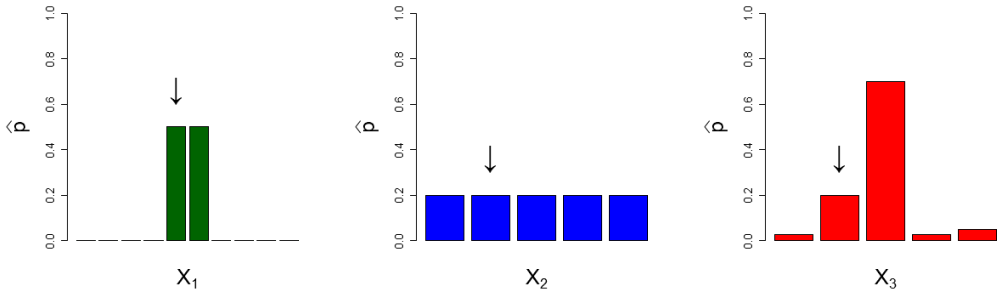


Figura 3.2: Histogramas para tres variables nominales X_1, X_2 y X_3 (de izquierda a derecha). Las flechas indican tres valores x_1, x_2 y x_3 dentro del recorrido de cada una.

Consideremos a su vez los tres valores $x_i \in \mathrm{Rec}(X_i),\ i = 1, 2, 3$ señalados. Buscamos definir funciones de rareza w^i a partir de la distribución de cada X_i:

- $w_1^i(x_i) = \dfrac{1}{\hat{p}(x_i)}$

  Esta es una de las maneras más sencillas y naturales de definir un grado de outlierness a partir del histograma de X_i. Sin embargo, con

esta definición, tendríamos que $w_1^i(x_1) = 2 < 5 = w_2^i(x_2)$ (la rareza de x_1 según X_1 sería menor a la de x_2 según X_2). Esto no parece ser una propiedad deseable, pues en ambos casos se trata de distribuciones uniformes. Esto podría volverse más problemático para variables X_i con una gran cantidad de categorías, ya que en ese caso se dispararían los valores de w_1^i. Por lo tanto, consideramos oportuno incluir un factor de normalización para el caso uniforme:

- $w_2^i(x_i) = \dfrac{1}{\hat{p}(x_i)\,|\mathrm{Sop}(X_i)|}$[1]
  Con esta nueva definición de la función de rareza tenemos que el caso anterior queda cubierto ya que $w_2^1(x_1) = \frac{1}{1/2\times 2} = 1 = \frac{1}{1/5\times 5} = w_2^2(x_2)$. No obstante, también se cumple que $w_2^3(x_3) = \frac{1}{1/5\times 5} = w_2^2(x_2)$ (la rareza de x_2 según X_2 sería igual a la de x_3 según X_3) el cual también parece un caso patológico. Después de todo X_2 es una variable uniforme, por lo que ningún valor es inusual, pero X_3 tiene un valor claramente muy frecuente. Tendría sentido entonces pedir que la función de rareza considere más inusual un valor como x_3, que aunque tiene la misma frecuencia que x_2, ocurre en una variable más concentrada (con menor entropía). Teniendo en cuenta lo anterior, ajustamos w_2^i según la entropía normalizada:
- $w_3^i(x_i) = \dfrac{\log_2\left(|\mathrm{Sop}(X_i)|\right)}{H(X_i)\hat{p}(x_i)\,|\mathrm{Sop}(X_i)|}$
  Con esta última definición logramos que los valores de rareza finalmente cumplan $w_3^3(x_3) = w_3^2(x_2) = w_3^1(x_1)$ que es lo que queríamos en los ejemplos anteriores.

En resumen, la manera en que hemos definido la rareza de una transacción es la siguiente:

$$w(\vec{t}_{n+1}) = \sum_{i=1}^{k} w_j^i(x_i) \text{ donde } \vec{t}_{n+1} = (x_1, \ldots, x_k) \tag{3.2}$$

donde $j \in \{1, 2, 3\}$ se determinará en cada caso particular.

3.2.2. Definición de una distancia nominal

Cuando queremos definir una distancia entre transacciones, ocurre exactamente lo contrario que con la función de rareza: el caso en que las

[1] Recordemos que $\mathrm{Sop}(X_i)$ es el soporte de la variable aleatoria X_i, tal como fue definido en 2.2

variables X_i son nominales hace más difícil el problema, ya que no hay una definición natural de distancia que sea utilizada con frecuencia en la literatura, salvo la que se conoce como *distancia de Hamming*:

$$d_H(\vec{t}_{n+1}, \vec{t}_i) = \sum_{j=1}^{k} \mathbb{1}\left(x_j^{n+1} \neq x_j^i\right) \text{ donde } \vec{t}_i = (x_1^i, \ldots, x_k^i) \tag{3.3}$$

En definitiva, esta distancia cuenta la cantidad de variables X_i en las que difieren las transacciones $\vec{t}_{n+1}$ y $\vec{t}_i$. Alternativamente, podríamos transformar adecuadamente las X_i para llevar el espacio de características a uno continuo. Sin embargo, los métodos más utilizados para este fin tienen sus limitaciones:

- En un caso como el nuestro en que necesitamos calcular distancias sobre los datos transformados, usar *One Hot Encoding*[2] puede no ser lo más indicado, no solo por los problemas expuestos en 2.1, sino que también porque las variables Y_i construidas, si bien son continuas, solo pueden tomar los valores 0 o 1. Esto provoca, por ejemplo, que si d_E es la distancia euclídea en el espacio transformado, d_H es la distancia de Hamming definida según (3.3) y f es la transformación que lleva el espacio nominal al continuo, entonces se cumple que $d_E(f(\vec{t}_1), f(\vec{t}_2)) = \sqrt{2d_H(\vec{t}_1, \vec{t}_2)}$, por lo que no hay gran diferencia entre trabajar con un espacio u otro.
- Codificación Ordinal es la transformación de una variable nominal X con recorrido $\{x_1, \ldots, x_k\}$ en Y continua calculada como $Y = \sum_{h=1}^{k} h\mathbb{1}(X = x_h)$. El mayor problema de realizar este cambio es que la métrica en el espacio así transformado depende enormemente del orden en que se consideren los valores x_h del recorrido de X. En casos de variables donde exista una relación de orden natural esto no presenta un problema, pero esto no ocurre para la mayoría de las características en nuestro caso. Existen técnicas extendidas para codificar los valores de acuerdo a un orden establecido por los niveles de fraude que presente cada uno (por ejemplo, el cálculo de los llamados *WOEs* o *Weights of Evidence*), pero esto puede no ser lo más adecuado en nuestro contexto porque queremos caracterizar la normalidad de cada cliente independientemente del comportamiento global.

Teniendo en cuenta lo anterior, decidimos restringirnos al caso nominal.

[2]Método de extracción de características definido en 2.1.

Como hicimos anteriormente para deducir una fórmula para la función de rareza, nos basaremos en una definición sencilla (la distancia de Hamming (3.3)) y la complejizaremos para que cumpla propiedades deseables. Para entender las limitaciones de esta definición inicial, observemos el siguiente ejemplo:

	X_1 = Indic. PIN	X_2 = Moneda	X_3 = Id Comercio
$\vec{t}_1$	1	\$	Id_1
$\vec{t}_2$	0	€	Id_2
⋮	⋮	⋮	⋮
$\vec{t}_{n+1}$	1	\$	Id_2

Tabla 3.1: Valores de tres variables X_1, X_2 y X_3 en las transacciones $\vec{t}_1$ y $\vec{t}_2$ del histórico $T(\vec{t}_{n+1})$ y en la transacción entrante $\vec{t}_{n+1}$ de una tarjeta.

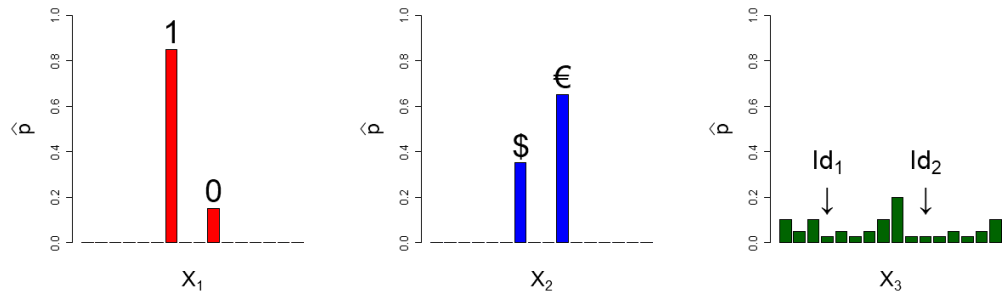


Figura 3.3: Histogramas de las variables X_1, X_2 y X_3. Se marcan los valores que aparecen en el cuadro anterior.

Nuevamente estamos considerando un histórico de transacciones $T((\vec{t}_{n+1})) = \{\vec{t}_i,\ i = 1, \ldots, k\}$ y tres variables nominales X_1, X_2 y X_3 cuyo significado es el descrito en el cuadro superior. Como podemos ver, si usáramos la distancia de Hamming tendríamos que $d(\vec{t}_{n+1}, \vec{t}_1) = 1 < 2 = d(\vec{t}_{n+1}, \vec{t}_2)$. Si el resto de las transacciones están a distancia mayor, tendríamos que $\vec{t}_1$ es el vecino más cercano a $\vec{t}_{n+1}$. Sin embargo, $\vec{t}_1$ solo se parece a $\vec{t}_{n+1}$ en dos variables muy poco informativas como son X_1 y X_2: en este caso, ambas tienen un recorrido de tamaño 2. En cambio, $\vec{t}_2$ y $\vec{t}_{n+1}$ son iguales en X_3, la cual está notoriamente más distribuida. Tendría sentido, por lo tanto, suponer que $\vec{t}_{n+1}$ debería estar más cerca de $\vec{t}_2$ que de $\vec{t}_1$, porque se parece a $\vec{t}_2$ en variables que tienen más información que las similares con $\vec{t}_1$. Para traducirlo

en términos de una fórmula, podríamos definir:

$$d(\vec{t}_{n+1}, \vec{t}_i) = \sum_{j=1}^{k} \left[H(X_i) \times \mathbb{1}\left(x_j^{n+1} \neq x_j^i \right) \right] \text{ donde } \vec{t}_i = (x_1^i, \ldots, x_k^i) \qquad (3.4)$$

Con esta distancia estamos ponderando las similitudes según las entropías de las variables.

3.3. Resultados experimentales

En esta sección presentaremos algunos experimentos iniciales que se hicieron para evaluar cuáles fórmulas de cálculo de la distancia y de la rareza deberían utilizarse y cómo afecta la cantidad de histórico al desempeño de las mismas. Los cálculos se hicieron tomando aleatoriamente 10.000 tarjetas del conjunto de entrenamiento en todas las bases de datos diferentes. En cada una de ellas, se descartaron transacciones pertenecientes a cortes con muy pocos fraudes y se particionó el conjunto de entrenamiento en 5 subconjuntos de transacciones según la cantidad de histórico. La manera en detalle de determinar las particiones fue la siguiente:

- Se construyó la variable $X =$*índice de la transacción* (usando la notación vista en la sección 2.2 del capítulo 2).
- Se definieron $q_0, q_1, \ldots, q_6$ como los cuantiles q_α de niveles 0, 0.2, 0.4, 0.6, 0.8 y 1 respectivamente de X *restringiendonos al conjunto de los fraudes.*
- Se asignó una transacción de índice n al conjunto i tal que $q_{i-1} < n \leq q_i$.

De esta manera, nos aseguramos que la cantidad de fraudes total haya quedado equitativamente repartida en cada subconjunto. Sin embargo, la cantidad de transacciones legítimas si podría variar, por lo que cambiarían las diluciones de los conjuntos. En la tabla 3.2 mostramos cómo resultaron las diluciones de los conjuntos determinados.

El desempeño de los scores y los modelos se midió graficando el error medido como 1 - AUC(ROC) (como fue discutido en la sección 2.4.1 del capítulo 2).

3.3.1. Rareza

Para determinar cual es la mejor manera de calcular la rareza, se construyó un score $w = \sum_{i=1}^{k} w_j^i$ donde $j = 1, 2, 3$ son cada una de

Base	1.1	1.2	2.1	2.2	3
$(0, q_{0.2}]$	503 : 1	587 : 1	228 : 1	915 : 1	144 : 1
$(q_{0.2}, q_{0.4}]$	854 : 1	994 : 1	199 : 1	1009 : 1	104 : 1
$(q_{0.4}, q_{0.6}]$	750 : 1	1300 : 1	215 : 1	1236 : 1	129 : 1
$(q_{0.6}, q_{0.8}]$	819 : 1	1001 : 1	255 : 1	1405 : 1	148 : 1
$(q_{0.8}, \infty)$	1.193 : 1	1494 : 1	378 : 1	1556 : 1	170 : 1

Tabla 3.2: Dilución en los conjuntos utilizados para cada experimento. Cada columna representa una base, y cada fila las cantidades de histórico consideradas.

las definiciones que vimos en la sección 3.2.1. Como queremos estimar no solamente desempeños medios sino que también su variabilidad, para cada una de las bases y para cada uno de los 5 subconjuntos de entrenamiento determinados según la cantidad de histórico, se sortearon 10 muestras de la mitad de las tarjetas. Luego, se evaluó el desempeño del score en las tarjetas pertenecientes a cada una de la muestras usando la curva inversa a la ROC (definida como el gráfico del *review* en función de la efectividad en la sección 2.4.1). El área debajo de esta gráfica es una medida de error del score pues a menor área, menor *review* para valores altos de efectividad, y por ende mejor desempeño. Los resultados se muestran en las figuras 3.4, 3.5, 3.6, 3.7 y 3.8.

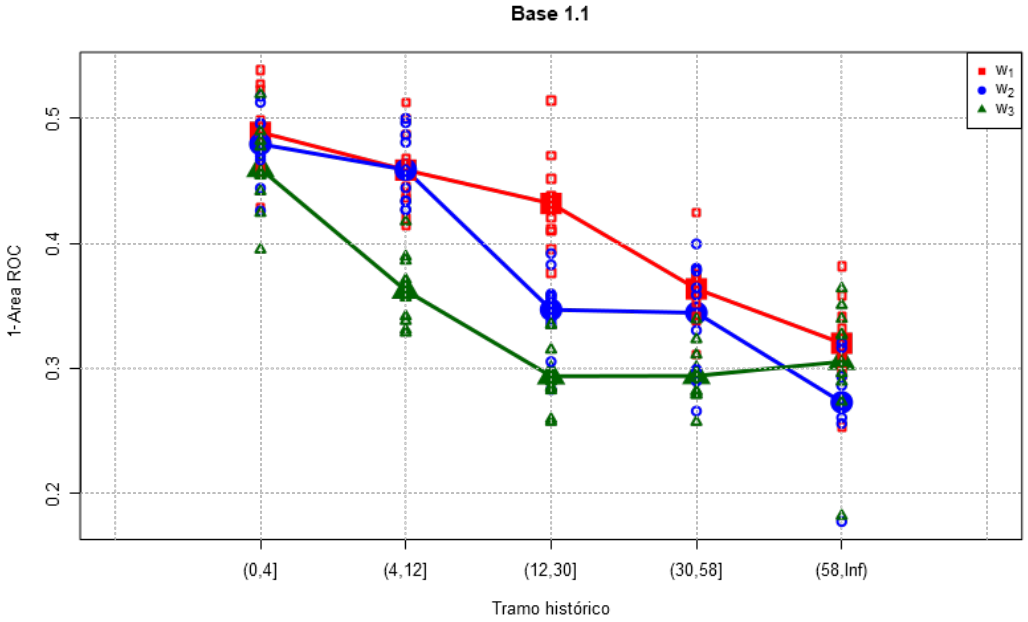


Figura 3.4: Error en función de la cantidad de histórico previo de scores de fraude definidos como $s_j(\vec{t}) = w_j(\vec{t}) = \sum_{i=1}^{k} w_j^i(x_i)$ para las funciones de rareza w_j, $j = 1, 2, 3$ definidas anteriormente. Se realizó el experimento en la base 1.1.

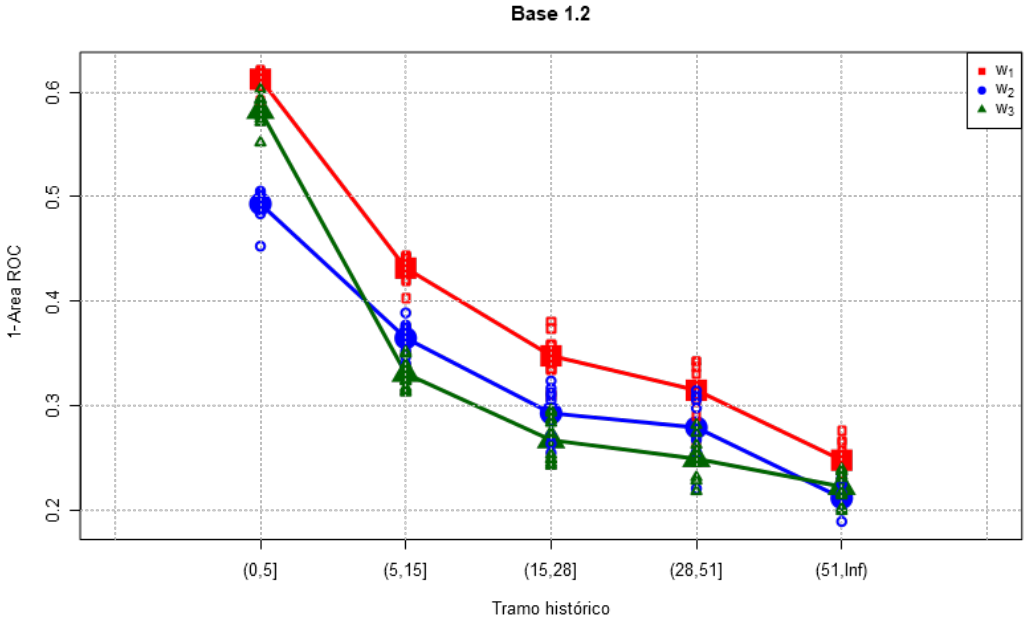


Figura 3.5: Repetición del experimento anterior en la base 1.2.

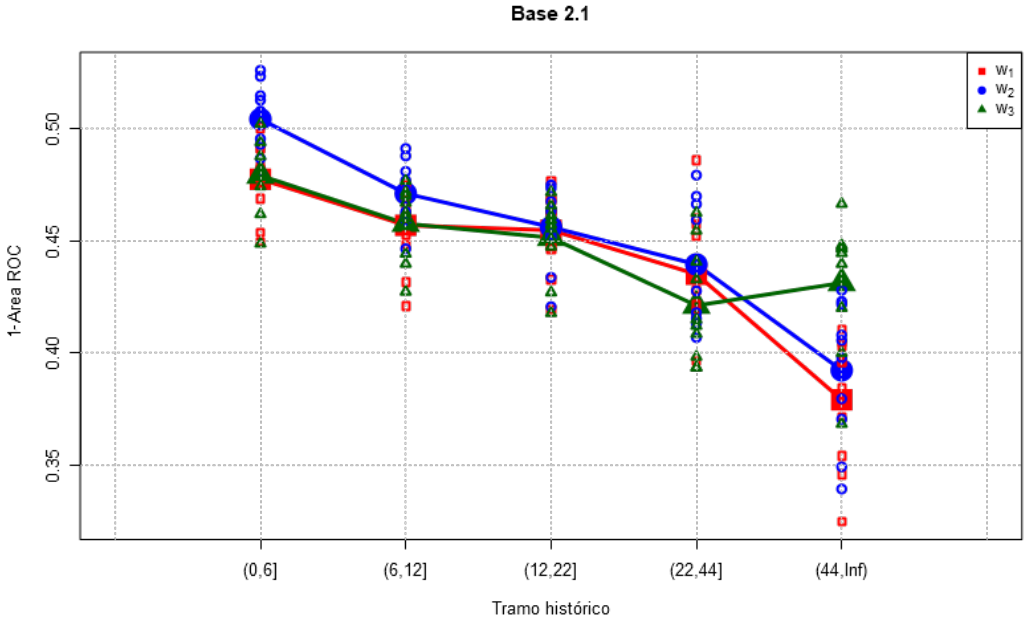


Figura 3.6: Repetición del experimento anterior en la base 2.1.

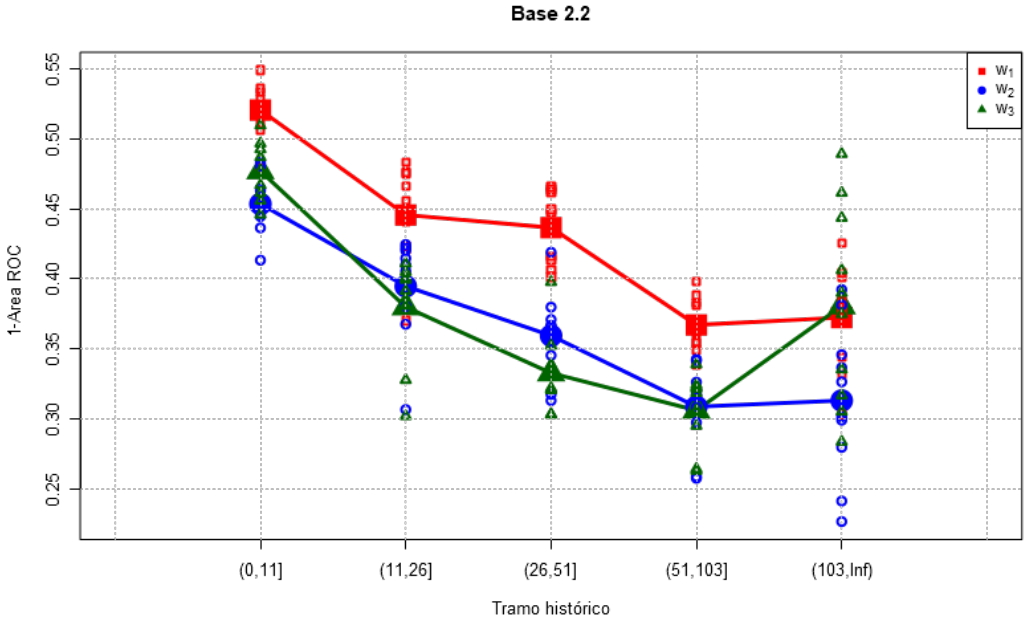


Figura 3.7: Repetición del experimento anterior en la base 2.2.

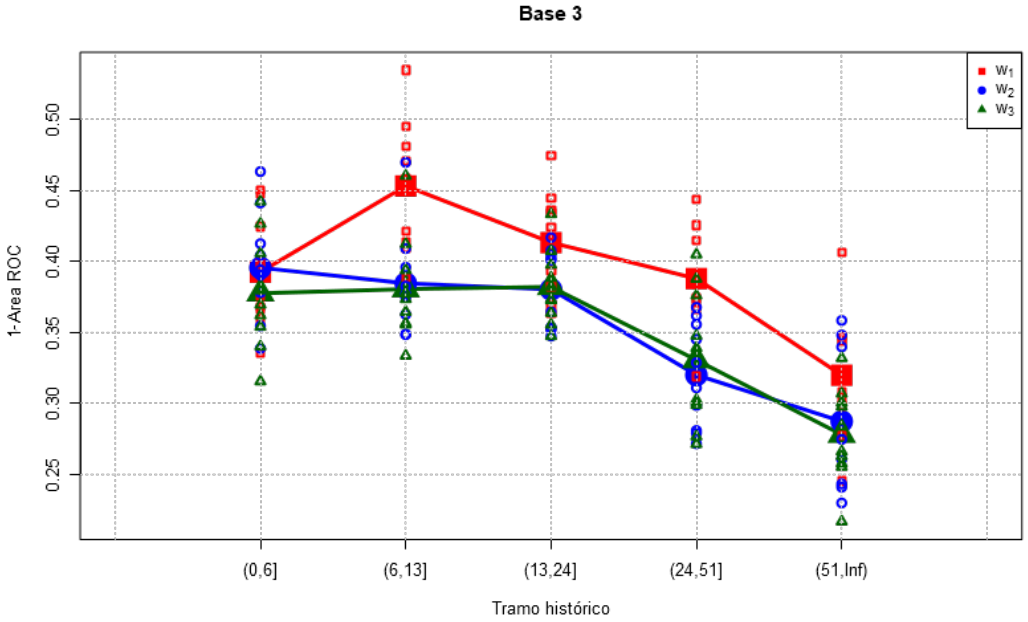


Figura 3.8: Repetición del experimento anterior en la base 3.

De observar los gráficos anteriores concluimos que:

1. En líneas generales, el error promedio de la rareza tiende a disminuir cuando se cuenta con mayor cantidad de histórico. En algunos casos (curvas verdes de las figuras 3.4 y 3.6) ocurre que el error promedio aumenta ligeramente de un tramo a otro con más histórico, pero no de manera estadísticamente significativa teniendo en cuenta la dispersión.
2. No existe una única función de rareza que en todos los casos supere a las demás en cuanto al desempeño. Por lo tanto, tomaremos la elección de la función de rareza como un metaparámetro a optimizar en cada caso.

3.3.2. Distancia

Para comparar el desempeño de los scores construidos con la distancia, al igual que con las rarezas, se construyó un score como suma de las distancias en cada coordenada (usando las dos formas de cálculo presentadas anteriormente). La manera en que se construyeron los conjuntos y se midió el desempeño es análoga. Los resultados se muestran en las figuras 3.9, 3.10, 3.11, 3.12 y 3.13.

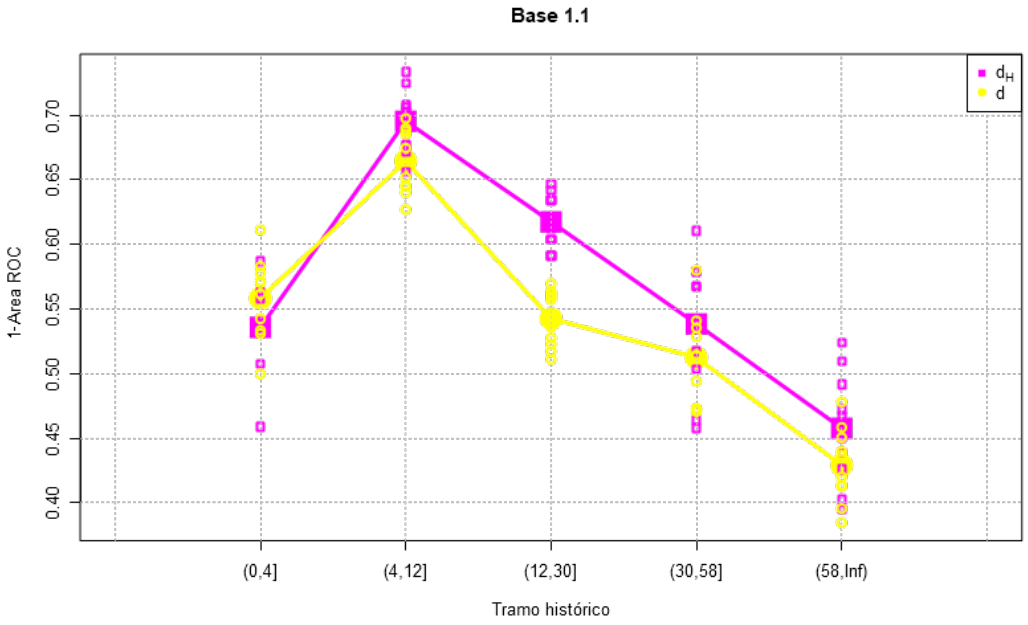


Figura 3.9: Error en función de la cantidad de histórico previo de dos modelos cuyos scores de fraude fueron definidos para cada transacción $\vec{t}$ como la distancia al vecino más cercano de $T(\vec{t})$, donde se tomaron las dos distancias d_H y d definidas anteriormente. Se realizó el experimento en la base 1.1.

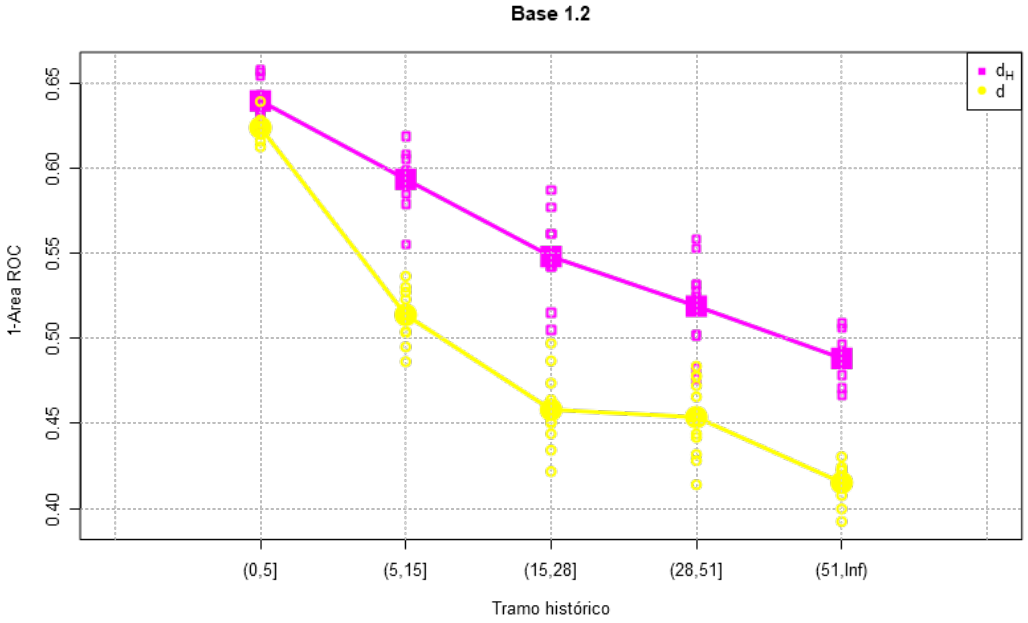


Figura 3.10: Repetición del experimento anterior en la base 1.2.

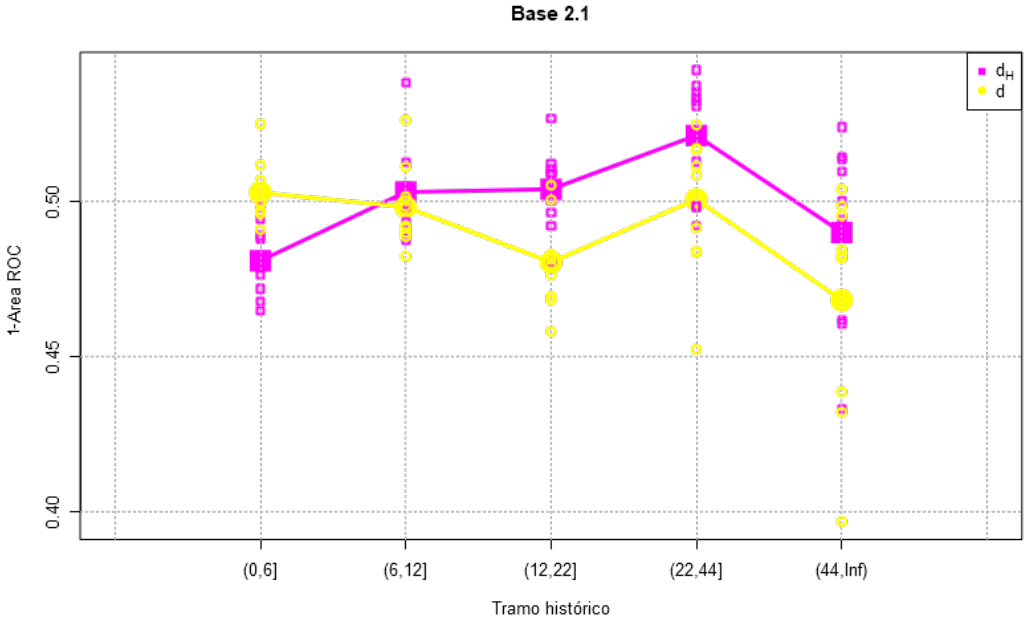


Figura 3.11: Repetición del experimento anterior en la base 2.1.

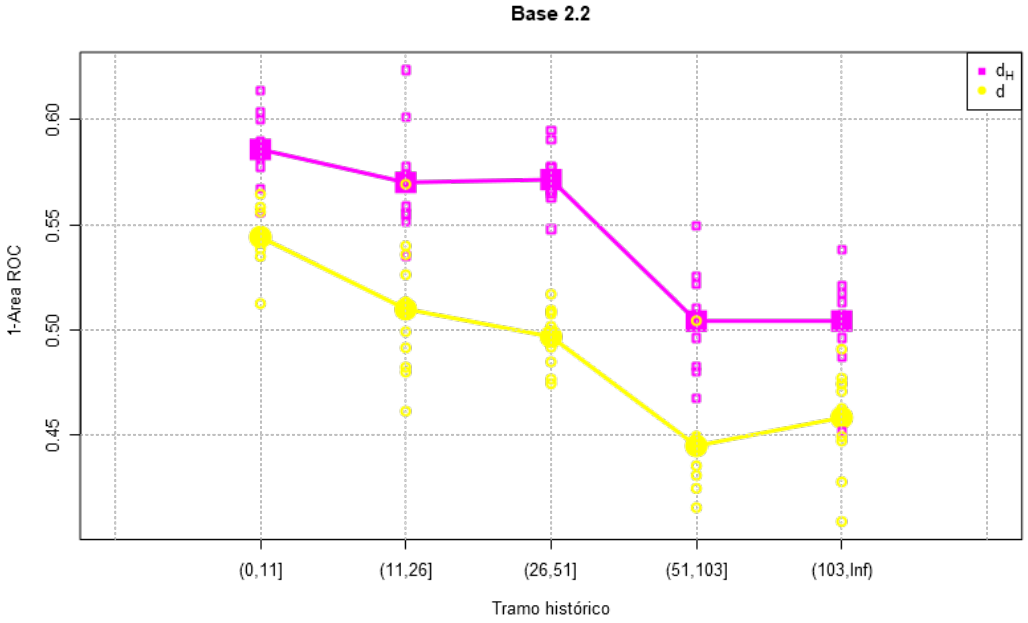


Figura 3.12: Repetición del experimento anterior en la base 2.2.

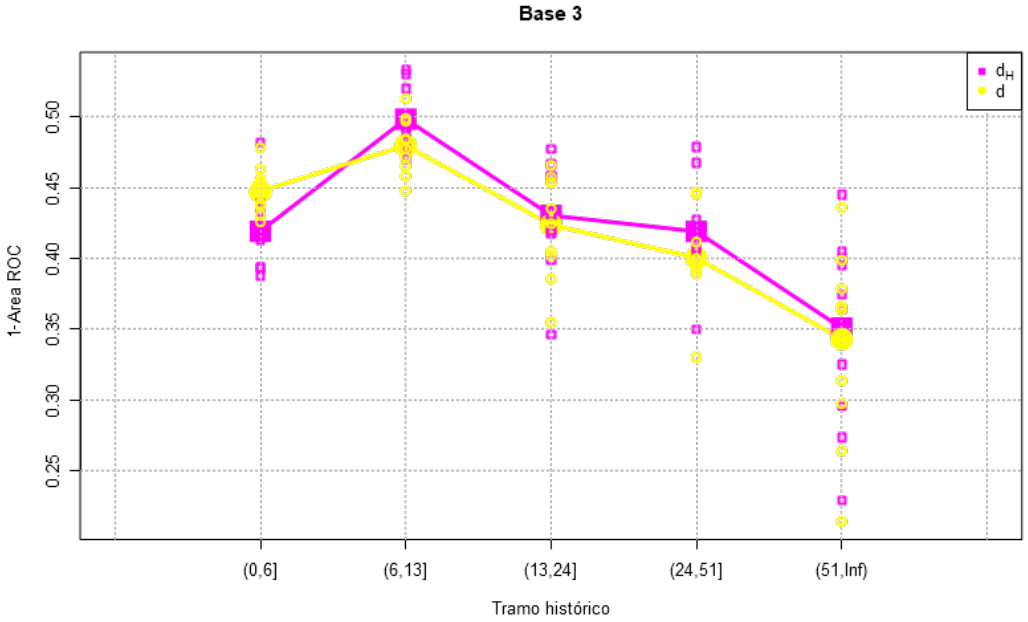


Figura 3.13: Repetición del experimento anterior en la base 3.

De estos gráficos concluimos lo siguiente:

1. El error en todas las bases es elevado, llegando en ocasiones a superar el valor 0.5 (lo que indica que ese score funciona peor que uno aleatorio), por lo que debemos encontrar una manera de mejorar el cálculo (lo veremos en la sección siguiente).
2. No observamos una tendencia marcada a la disminución del error según la cantidad de histórico. En las bases 1.2 y 2.1 esto podría explicarse por la variación en la dilución de los conjuntos, que aumenta en los tramos donde también aumenta el error.
3. En todas las bases, a partir de determinada cantidad de histórico, la distancia definida en la sección 3.2.2 es preferible a la distancia de Hamming. Como veremos en el capítulo siguiente, el caso en que el histórico previo de una transacción es muy bajo será tratado aparte, por lo que en definitiva siempre usaremos nuestra definición de distancia.

3.4. Score de Outlierness supervisado

Hasta el momento hemos definido funciones que nos permiten obtener dos evidencias complementarias de *outlierness* de una transacción con respecto a su histórico. Inicialmente, formulamos el producto de ambas como una combinación posible para consolidarlas en un score único que realice la discriminación. Sin embargo, recordemos que todos los cálculos que hemos realizado han sido no supervisados, pero que en todas las bases contamos con el dato de la etiqueta de fraude. Entonces, sería razonable utilizar esta información para obtener una mejor detección. Por ejemplo: si observamos las ecuaciones (3.2) y (3.4) vemos claramente como hemos definido tanto la rareza como la distancia como la suma de funciones w_j^i y d^i en cada coordenada X_i. Esto nos está diciendo que nuestro score final es una combinación de dos tipos de evidencia por cada variable X_i. Por lo tanto, cabe preguntarse, ¿no podríamos usar la etiqueta de fraude para determinar cual es la combinación óptima de estas evidencias? Un ejemplo muy sencillo podría ser mediante una regresión lineal, la cual nos da explícitamente el score final como una combinación lineal de las w^i y d^i.

Para probar si esta aproximación mejora los resultados se realizó un experimento donde las bases se partieron de la misma manera que en los

experimentos anteriores. Nuevamente, se tomaron 10 muestras de la mitad de las tarjetas[3] para cada subconjunto de entrenamiento obtenido a partir de un tramo según la cantidad de histórico. Esta vez, con cada muestra se calcularon dos scores distintos:

- El primero se obtuvo como $s_1 = \sum_{i=1} k(w_3^i + d^i)$, lo que correspondería a la combinación no supervisada definida en la ecuacion 3.1 [4].
- El segundo se obtuvo evaluando una regresión lineal entrenada en la mitad restante de las tarjetas y cuyas variables de entrada eran las rarezas w_3^i y distancias d^i en cada una de las coordenadas.

Los resultados se muestran en las figuras 3.14, 3.15, 3.16, 3.17 y 3.18 a continuación. De las mismas se desprende que la combinación supervisada de las rarezas y distancias en cada coordenada genera un desempeño notoriamente superior que la no supervisada.

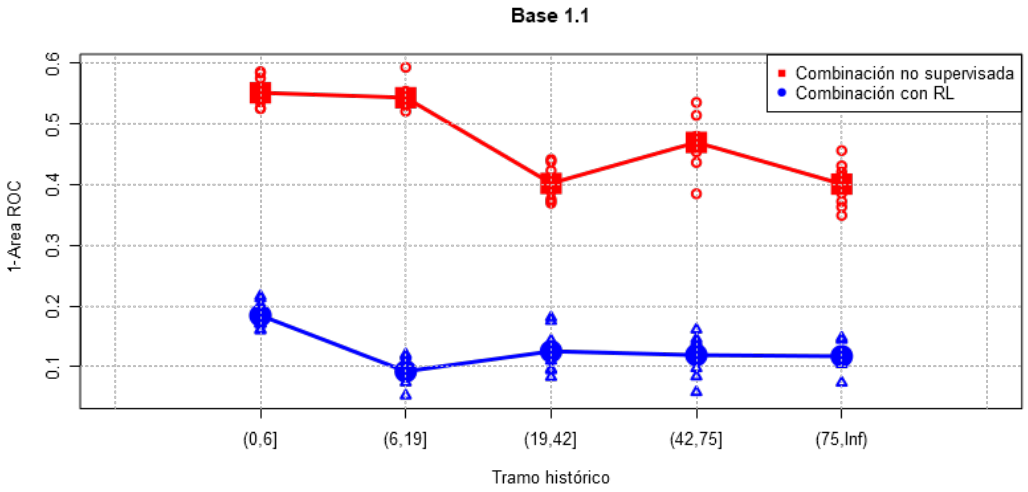


Figura 3.14: Error en función de la cantidad de histórico previo de dos modelos. El primero (rojo) da un score de fraude definido como la combinación no supervisada de rarezas y distancias. El segundo (azul) es una regresión lineal aplicada sobre las características construidas como w^i y d^i para todas las X_i de cada una de las bases (rareza y distancia en cada coordenada i-ésima).

[3] En concordancia con lo visto en la sección 2.4.2 del capítulo 2, el sorteo se realiza por tarjetas para evitar sobreajuste.

[4] Podríamos incluir aquí cualquier combinación *a priori* de las evidencias. No probamos con la suma de las distancias con las rarezas ya que esto es seguro que funcionará mejor con la regresión lineal, pues el algoritmo está diseñado para elegir la mejor combinación lineal.

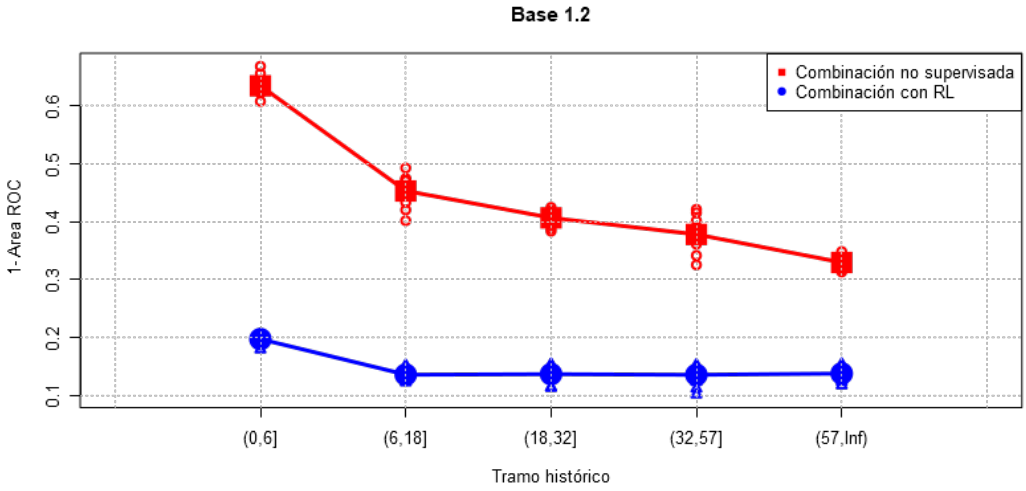


Figura 3.15: Repetición del experimento anterior en la base 1.2.

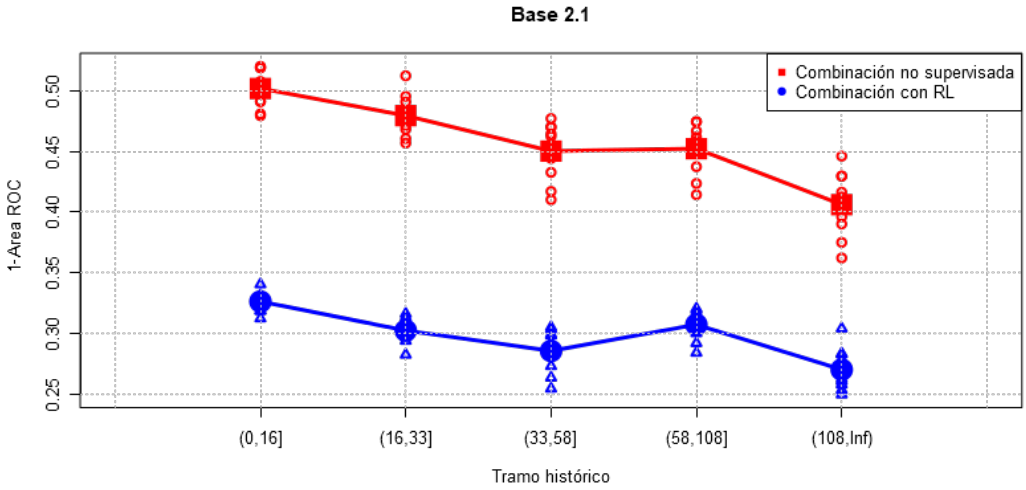


Figura 3.16: Repetición del experimento anterior en la base 2.1.

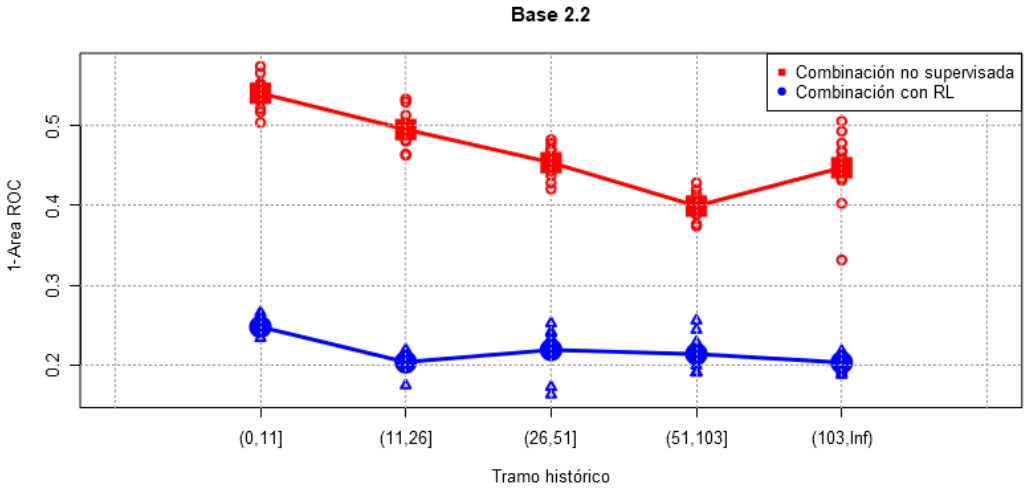


Figura 3.17: Repetición del experimento anterior en la base 2.2.

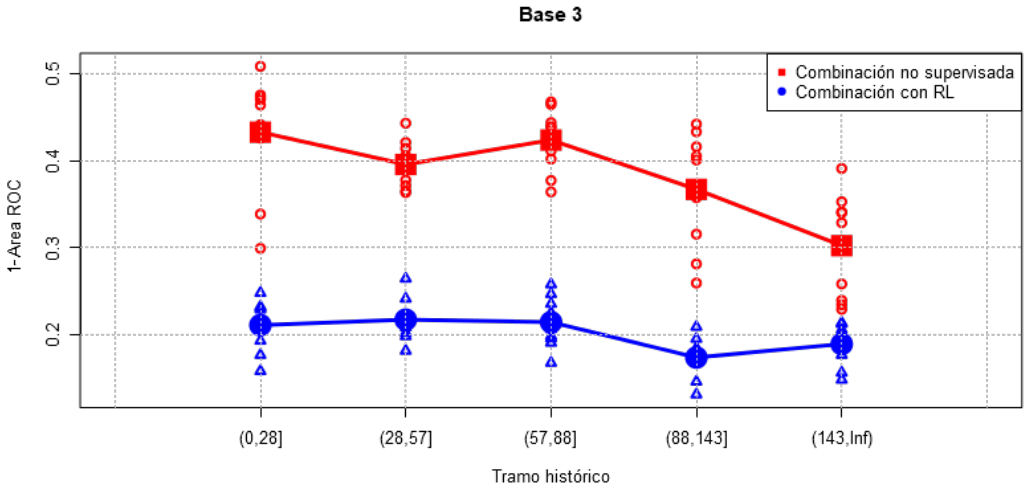


Figura 3.18: Repetición del experimento anterior en la base 3.

En conclusión, dada la mejoría tan pronunciada al utilizar un Score de Outlierness supervisado, durante el resto del trabajo usaremos las funciones de rareza y el cálculo de distancia como un método de extracción de características, que en particular permite obtener Y_i y Z_i continuas a partir de una X_i nominal.

Capítulo 4

Optimización de parámetros del Score de Outliernes

En el capítulo anterior definimos la metodología de cálculo del "Score de Outlierness" de una transacción $\vec{t}_{n+1}$ respecto a determinada variable X bajo ciertas hipótesis, como por ejemplo, que X era nominal o que la cantidad n de transacciones anteriores a $\vec{t}_{n+1}$ de la misma tarjeta eran *suficientes* como para hacer una buena estimación de la distribución de X. Durante este capítulo nos dedicaremos a presentar alternativas para el caso en que estas hipótesis no se cumplan. También plantearemos cuestiones que surgen por la estructura de los datos y que conllevan posibles mejoras en el poder de detección de las variables calculadas. Todas las discusiones presentadas serán tratadas como decisiones de diseño del modelo de detección que podemos reducir a un problema de elección de uno o varios parámetros óptimos.

4.1. El problema de la cantidad de histórico

Una de las hipótesis más fuertes y que hemos mencionado con más frecuencia hasta el momento es la relativa al tamaño del conjunto $T(\vec{t}_{n+1})$ de transacciones anteriores a $\vec{t}_{n+1}$. Es claro que la precisión de los cálculos que hemos presentado está sujeta a que haya una cantidad suficiente de histórico, pero cuánto es esta cantidad depende de varios factores como el período de tiempo que abarca la base de datos, la frecuencia con que los clientes usan su tarjeta o el tamaño del recorrido de las variables nominales. Sin embargo, lo que podemos asegurar es que *siempre* se dan casos donde el histórico es

insuficiente ya que toda tarjeta tiene una primera transacción para la cual no se puede armar ningún perfil (si no se cuentan con datos no financieros del cliente, que es lo que ocurre con la mayoría de las bases que manejamos). Es imprescindible por lo tanto definir una estrategia para ese contexto.

4.1.1. El conjunto de histórico insuficiente

Volviendo a las definiciones y notación del capítulo 3, lo que falla cuando el conjunto de histórico $T(\vec{t}_{n+1})$ tiene cardinal bajo es que no podemos modelar la "normalidad" propia de la tarjeta contra la cual querríamos comparar la transacción $\vec{t}_{n+1}$. Lo más confiable a lo que podríamos recurrir es a las características propias de la transacción y comparar con las del histórico que tengan el mismo valor. En definitiva, proponemos lo siguiente:

Dada X_{i_0} característica i_0-ésima nominal tal que $\text{Rec}(X) = \{x_1^{i_0}, \ldots, x_{k_{i_0}}^{i_0}\}$:

1. Seleccionar aleatoriamente un subconjunto de tarjetas sin fraudes del conjunto de entrenamiento (que llamaremos conjunto de **histórico fijo**).
2. Dividir este subconjunto en k particiones de la forma $S_j = \{X_{i_0} = x_j^{i_0}\},\ j = 1, \ldots, k_{i_0}$
3. Utilizar S_j como el histórico de una transacción $\vec{t}$ cuyo valor en la coordenada i_0-ésima es $x_j^{i_0}$.

La elección de la variable X_{i_0} y la cantidad de tarjetas a sortear para formar el conjunto de histórico fijo son decisiones de diseño que deben optimizarse en cada caso. Lo más sencillo sería realizar un *grid search* usando *wrapping* (ver 2.2). Para elegir el conjunto de valores a explorar convendría tener en cuenta las siguientes consideraciones:

1. Es conveniente explorar variables X_{i_0} cuyo recorrido no sea demasiado pequeño, para que determinen una cantidad suficiente de conjuntos S_j y así permitir una diferenciación más fina entre transacciones.
   Para ilustrar por qué esto es importante, supongamos el caso extremo en que existe un único conjunto S_j pues $\text{Rec}(X_{i_0}) = \{x_j^{i_0}\}$. Si observamos la manera en que definimos las funciones de rareza en la sección 3.2.1, podemos observar que para cualquiera de ellas, su valor en la coordenada h-ésima sólo depende del valor de la transacción evaluada en X_h y de la

distribución de X_h en el conjunto de histórico que se esté utilizando. Como estamos suponiendo que hay un único conjunto S_j de histórico fijo para todas las transacciones, la distribución de X_h en el conjunto de histórico es siempre la misma. Por lo tanto, calcular la rareza de una transacción es equivalente a codificar la variable X_h a un conjunto finito de números. Como vimos en la sección 2.6, nuestro objetivo es utilizar árboles de decisión como algoritmo de clasificación, por lo que una reordenación de los valores de X_h no tiene ningún efecto en la clasificación[1]

2. Tanto para la distancia como para la rareza, considerar una variable X_{i_0} que genera subconjuntos S_j demasiado pequeños conlleva los mismos problemas para la estimación que usar el propio histórico de la tarjeta (que era precisamente lo que se quería evitar). Por lo tanto, conviene explorar algunas X_{i_0} que generen conjuntos S_j no demasiado pequeños.

En definitiva, lo que conviene considerar es que existe una tensión a resolver entre los valores extremos de los parámetros que queremos optimizar. Convendría por lo tanto llevar a cabo una exploración que genere conjuntos S_j de características variadas.

4.1.2. Optimización del parámetro m_H

Con la estrategia definida anteriormente, estamos en condiciones calcular el score de Outlierness para cualquier transacción: queda por determinar ahora el valor óptimo para el parámetro m_H tal que si $\left|T(\vec{t}_{n+1})\right| > m_H$ entonces comparamos a $\vec{t}_{n+1}$ contra las transacciones de su histórico $T(\vec{t}_{n+1})$ y si $\left|T(\vec{t}_{n+1})\right| \leq m_H$ entonces recurrimos al conjunto de histórico fijo. Para optimizar el valor de m_H proponemos asignarle secuencialmente los valores de un conjunto M de valores y para cada uno de ellos:

- Partir
  el conjunto de entrenamiento D en $D_{suf} = \left\{\vec{t} \in D : \left|T(\vec{t}_{n+1})\right| > m_H\right\}$ (conjunto de histórico suficiente) y $D_{bajo} = \left\{\vec{t} \in D : \left|T(\vec{t}_{n+1})\right| \leq m_H\right\}$ (conjunto de histórico bajo).
- Generar las variables calculadas como el Score de Outlierness respecto a cada característica X en D_{suf} y D_{bajo} por separado.

[1] Esto no es necesariamente cierto para otros clasificadores.

- Calcular un modelo que use ambas variables y estimar los desempeños usando validación cruzada (ver 2.2).
- Estimar el desempeño global de tener ambos modelos funcionando en simultáneo (ver 4 para los detalles).

De esta manera podemos elegir el valor de m_H que genere variables más útiles para la detección. Los valores a explorar se deben determinar de manera que los conjuntos D_{suf} y D_{bajo} tengan una cantidad significativa de fraudes como para poder ejecutar validación cruzada. Para esto, sugerimos determinarlos mediante los cuantiles de la distribución de la variable $I = |T(\cdot)| = \textit{tamaño del histórico anterior de la tarjeta}$ en las transacciones fraudulentas del conjunto de entrenamiento.

4.2. Ventanas Superior e Inferior

Para esta sección, nos centraremos en el conjunto de histórico suficiente. Una consideración que debemos tener en cuenta cuando queremos armar algún tipo de perfil del cliente para detectar anomalías es que no todos los comportamientos atípicos tienen por qué ser fraudulentos. De hecho, los clientes pueden irse de viaje, aumentar sus ingresos o tener otros cambios de conducta que se reflejarían en su perfil. Lo que esperaríamos es que la primera transacción extraña que llegue al banco sea alertada, pero a medida que vayan apareciendo más casos parecidos, podamos aprender el nuevo comportamiento y dejar de emitir falsas alarmas. Sin embargo, como usamos histogramas para calcular el score de outlierness en el caso de la rareza, cuantos más casos previos existan más lento es el reaprendizaje del nuevo perfil, pues tienen que ingresar muchas transacciones del nuevo comportamiento para balancear el histórico previo. Por lo tanto, sería razonable implementar un límite de tiempo para las transacciones a considerar en el histórico. Llamaremos a este parámetro la *ventana inferior* de tiempo v_i.

Por otro lado, utilizar información demasiado nueva también puede ser un problema: es sabido que una vez que una tarjeta es comprometida, es usual que se intenten cometer la mayor cantidad de fraudes posibles antes que la tarjeta sea bloqueada. En un caso como este, si la primera transacción inusual no provoca el bloqueo de la tarjeta, las siguientes podrían tener un score de outlierness más bajo que la original debido a que los fraudes previos entran

dentro del conjunto de histórico previo a comparar. De hecho, para evitar este problema, no podemos recurrir tampoco a la etiqueta de fraude pues muchas veces la misma es colocada ya cuando es demasiado tarde y la tarjeta ya se bloqueó. La solución que planteamos para este problema es introducir un parámetro que llamaremos a *ventana superior* de tiempo v_s de manera que no se consideren para el histórico aquellas transacciones que se han hecho hace menos de v_s unidades de tiempo.

En definitiva, tenemos dos parámetros a optimizar: v_s y v_i de manera que en el conjunto de histórico superior, las transacciones que se utilizan para calcular el score de outlierness de una transacción $\vec{t}_{n+1}$ son las $\vec{t}_i,\ i = 1, \ldots, n$ de la misma tarjeta tales que $v_s \leq d_F(\vec{t}_{n+1}, \vec{t}_i) \leq v_i$ donde d_F es la diferencia de fechas (medida en la misma unidad de tiempo que los parámetros v_s y v_i). El conflicto que queda por resolver es la interacción de este filtro con la elección del parámetro m_H, pues si se sacan demasiadas transacciones de $T(\vec{t}_{n+1})$, $\vec{t}_{n+1}$ puede pasar del conjunto $Fsuf$ al $Fbajo$. El criterio que proponemos por simplicidad y para evitar descartar excesiva información es el siguiente:

Dado el histórico $T(\vec{t}_{n+1})$ de $\vec{t}_{n+1}$:

- Descartar todas aquellas $\vec{t}_i \in T(\vec{t}_{n+1})$ tales que $d_F(\vec{t}_{n+1}, \vec{t}_i) < v_s$
- Descartar las k transacciones $\vec{t}_i$ más antiguas tales que $d_F(\vec{t}_{n+1}, \vec{t}_i) > v_i$ donde $k = \left|T(\vec{t}_{n+1})\right| - m_H$

Con este mecanismo estamos dando prioridad al filtro de la ventana superior, que es el más crítico en cuanto a la detección de fraude (según el conocimiento experto). Además, no dejamos que se descarten tantas transacciones antiguas como para disminuir el histórico a un tamaño menor que el mínimo histórico. Por lo tanto, solamente al variar los valores de v_s pueden cambiar los conjuntos $Dsuf$ y $Dbajo$. En consecuencia, la optimización de este parámetro debe hacerse adecuadamente, evaluando el impacto de un cada valor de v_s en los dos modelos[2].

4.3. Inclusión de variables continuas

En el capítulo 3 nos dedicamos a definir una distancia en espacios de variables nominales que funciona mejor en nuestro contexto que la distancia de

[2]Un valor de v_s que mejora levemente el modelo de histórico suficiente pero empeora notoriamente el modelo de histórico bajo no sería deseable.

Hamming. El caso en que alguna variable es continua es fácilmente reducible al caso nominal mediante alguna técnica de discretización. Sin embargo, esto implicaría perder información, y al mismo tiempo, desaprovechar la riqueza de posibilidades que ofrece el trabajar con variables naturalmente continuas. En esta sección presentaremos cómo podríamos trabajar con ambas ópticas:

4.3.1. Mezcla de variables continuas y nominales

Vamos a considerar un conjunto de características $X_1, \ldots, X_m$ nominales y otras $Y_1, \ldots, Y_k$ continuas. Usaremos una distancia en las variables continuas de manera que la distancia d entre dos transacciones $\vec{t_1} = (x_1^1, \ldots, x_m^1, y_1^1, \ldots, y_k^1)$ y $\vec{t_2} = (x_1^2, \ldots, x_m^2, y_1^2, \ldots, y_k^2)$ en este espacio mixto sea:

$$D\left(\vec{t_1}, \vec{t_2}\right) = \sum_{i=1}^{m} \mathbb{1}\left(x_i^1 \neq x_i^2\right) H(X) + \sum_{j=1}^{k} d\left(y_j^1, y_j^2\right)$$

Para mantener una analogía con la distancia definida por la ecuación (3.4) en el capítulo 3, estamos considerando D definida por coordenadas. Sin embargo, el combinar las métricas nominales con las continuas mediante la suma puede modificar drásticamente el peso que tiene una variable en la distancia total: por ejemplo, si X es una variable nominal con n categorías, el peso del término $\mathbb{1}\left(x_i^1 \neq x_i^2\right) H(X)$ está acotado por $log_2(n)$. Sin embargo, si Y es una variable como el monto y usamos la distancia euclídea, es altamente probable que $d(y_{j1}, y_{j2})$ supere en varios órdenes de magnitud a cualquiera de los términos provenientes de variables nominales, por lo que el vecino más cercano se determinaría casi solamente con el monto. Para remediar esto, podemos probar con distintas normalizaciones que transformen la distancia d a una escala más comparable con la de las variables nominales. Las definiciones de d que exploramos en nuestros experimentos son:

1. $d_1(x, y) = |x - y|^{1/b}, \ b \geq 1$
2. $d_2(x, y) = \left(\frac{|x-y|}{s_n}\right)^{1/b}, \ b \geq 1$
3. $d_3(x, y) = \log_b\left(|x - y| + 1\right), \ b \geq 2$
4. $d_4(x, y) = \log_b\left(\frac{|x-y|}{s_n} + 1\right), \ b \geq 2$

donde s_n es la raíz de la varianza muestral de la variable Y en el conjunto de histórico contra el que se está comparando.

4.3.2. Discretización global

Por otro lado, para evitar el problema de las escalas, sería razonable discretizar las variables continuas y así solo trabajar con variables del mismo tipo, volviendo al caso visto en el capítulo 3. El mayor problema que trae esta aproximación es que criterios distintos pueden tener como resultado variables con distribuciones muy distintas. Por ejemplo: consideremos una variable Y continua y un conjunto de puntos $P = \{p_1, \ldots, p_k\}$ tal que Y se transforma en una variable Y_d nominal con recorrido $\{0, 1, \ldots, k\}$ mediante la siguiente fórmula

$$Y_d = \begin{cases} 0 & \text{si } Y \leq p_1 \\ i & \text{si } Y \in (p_i, p_{i+1}],\ i = 1, \ldots, k \\ k & \text{si } Y > p_k \end{cases}$$

Consideremos también las siguientes elecciones para P:

- Si $p_i = q_{i/k}$ (cuantil de nivel i/k de Y) entonces Y_d tiene distribución uniforme discreta en $\{1, \ldots, k\}$ y forzamos a que $H(Y_d) = \log_2(k)$.
- Si $k = 2$, $p_1 < Y$ y $p_2 > Y$ entonces $Y_d = 1$ constante por lo que $H(Y_d) = 0$.

En definitiva: la elección de los intervalos de discretización determina la entropía de la variable resultado y en consecuencia, su peso en el cálculo de la distancia. Teniendo esto en cuenta proponemos los siguientes criterios:

1. **Intervalos uniformes:** $p_i = m + \frac{i-1}{k-1}(M - m),\ i = 1, \ldots, k$ donde m y M son dos cotas máximas y mínimas para Y. Esta discretización garantiza que los intervalos de discretización tengan la misma longitud, pero no hace ninguna asunción sobre la distribución de Y. De hecho, en los casos más usuales (en que Y es el monto o una variable temporal) la distribución de la variable Y_d resultante no es uniforme discreta.
2. **Intervalos logarítmicamente uniformes:**
   $p_i = \exp\left(\log m + \frac{i-1}{k-1}\log(M/m)\right),\ i = 1, \ldots, k$ donde m y M son dos cotas máximas y mínimas para Y. Con este mecanismo conseguimos que la longitud de los intervalos sea uniforme en la escala logarítmica. Esto puede ser de utilidad en los casos en que Y tiene una distribución aproximadamente exponencial, que es lo que usualmente ocurre con el monto.

3. **Por cuantiles**: $p_i = q_{i/n}$ donde los cuantiles $q_{i/n}$ son aprendidos mediante la distribución global de Y en todas las transacciones del conjunto de entrenamiento. En el caso del conjunto de histórico suficiente, esto determina la distribución de Y_d en el conjunto global pero no necesariamente en la distribución de cada tarjeta, por lo que no estaríamos necesariamente forzando Y_d a ser uniforme. En el caso del conjunto de histórico insuficiente, es más probable que esto sí ocurra dependiendo del tamaño y tipo de particiones que se hagan del conjunto de histórico fijo (por ejemplo, si los conjuntos se determinan según una X independiente de Y, Y_d sería uniforme en cada subconjunto).

Para ilustrar cómo funcionan estos métodos, mostraremos en la figura 4.1 un ejemplo en particular. Para construirlo, elegimos al azar en una de las bases una tarjeta con 200 transacciones en su histórico y consideramos Y = monto de la transacción. Vamos a discretizar la variable Y usando los tres criterios anteriormente presentados. En este ejemplo, el valor de los parametros en cada discretización es el siguiente:

- **Uniforme:** $m = 0$, $M = 1000$ y $k = 4$ puntos. Los puntos de corte resultado son $p_1 = 0$,$p_2 = 333$, $p_3 = 666$ y $p_4 = 1000$.
- **Uniforme logarítmica:** $m = 1$, $M = 1000$ y $k = 4$ puntos. Los puntos de corte resultado son $p_1 = 1, p_2 = 10$, $p_3 = 100$ y $p_4 = 1000$.
- **Por cuantiles:** niveles $0.2, 0.4, 0.6$ y 0.8. Los puntos de corte resultado son $p_1 = 19,65$, $p_2 = 28,35$, $p_3 = 45,50$ y $p_4 = 150$.

Como podemos ver en la figura 4.1, los métodos van produciendo variables progresivamente con más entropía. Particularmente en el último caso, al haber elegido los cuantiles a partir de la propia Y, la variable Y_d resulta uniforme. Sin embargo, como mencionamos anteriormente, esto no tiene por qué ser así cuando se considera Y restringida a una tarjeta en particular pero los cuantiles han sido determinados globalmente.

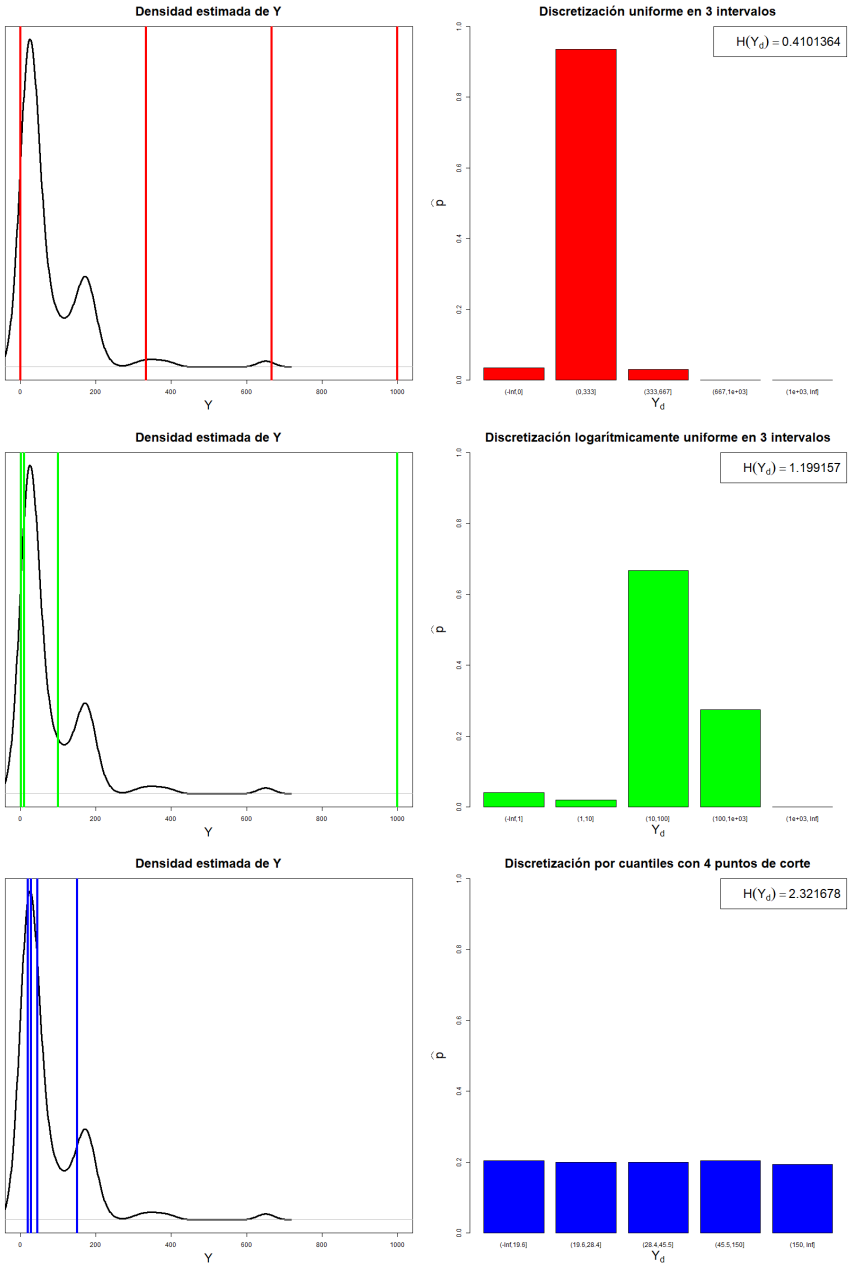


Figura 4.1: Discretización de la variable Y = "monto de la transacción" en un conjunto de 200 transacciones obtenidas de una tarjeta al azar. En la columna izquierda se grafica la densidad de Y estimada mediante kernels (en negro), mientras que las líneas verticales coloreadas indican los puntos p_i de corte determinados por cada mecanismo de discretización. En la columna derecha se muestran los histogramas de las variables Y_d discretizadas usando los puntos de corte p_i de la gráfica a la izquierda correspondiente. En la esquina superior derecha de cada histograma se muestra la entropía de la variable Y_d. Los colores indican los criterios de discretización: intervalos uniformes (rojo), uniformes logarítmicos (verde) y por cuantiles (azul).

En la figura 4.2 a continuación, comparamos los tres métodos según la entropía por tarjeta de las variables Y_d que producen (el caso de interés es el conjunto de histórico suficiente).

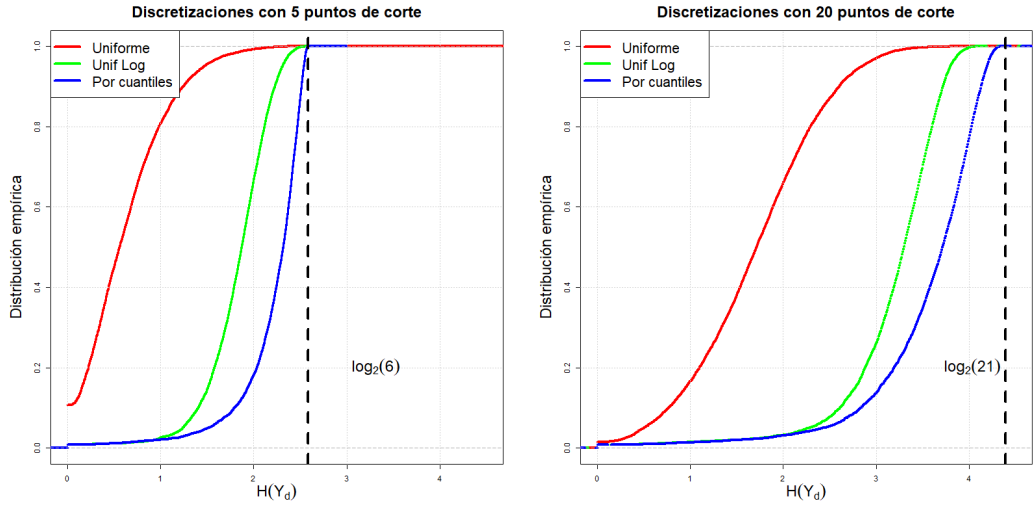


Figura 4.2: Distribución empírica de $H(Y_d)$ (entropía del monto discretizado) en 10.000 tarjetas elegidas al azar en el conjunto de entrenamiento con un histórico mayor a 10 transacciones. Se usaron los tres criterios de discretización descritos anteriormente usando 5 puntos de corte (izquierda) y 20 puntos de corte (derecha). Ambas gráficas tienen la misma escala. La línea vertical punteada indica la máxima entropía alcanzable que vale $log_2(|\text{Sop}(Y_d)|)$.

De las gráficas se desprende que en este caso, el método 2 produce una Y_d más uniforme que el método 1, pero eso depende fuertemente de la distribución inicial de Y. Nuevamente, por la complejidad de la interacción entre la discretización y el poder predictivo del Score de Outlierness como distancia es que proponemos elegir la aproximación que funcione mejor mediante validación cruzada.

4.3.3. Discretización adaptativa

Presentaremos ahora una última alternativa que busca mitigar un efecto que siempre esta presente en cualquier discretización. Consideremos el siguiente ejemplo, focalizándonos en el caso de histórico suficiente:

- En el histórico de un cliente hay 100 transacciones de montos entre \$50 y \$100.

- La variable Y =monto de la transacción se discretizó mediante intervalos logarítmicamente uniformes determinados por $P = \{10, 100, 1000, 10000\}$.
- Llega una transacción nueva $\vec{t}_{101}$ por valor \$101.

En un caso como este, la transacción $\vec{t}_{101}$ será considerada como un outlier usando el criterio de la rareza. Sin embargo, su valor según Y es muy próximo a valores en el histórico, por lo que una pequeña modificación en P hubiera determinado que $\vec{t}_{101}$ cayera dentro del mismo intervalo de Y que el resto del histórico, lo cual parece razonable.

Lo que estamos viendo aquí es que toda discretización cuyos intervalos se mantengan fijos puede resultarsubóptima a medida que van llegando nuevas transacciones de una tarjeta. Presentaremos entonces una última alternativa, cuyo objetivo es utilizar algún tipo de *clustering* (ver 2.1) para determinar categorías de discretización adaptatias según como se distribuyen las transacciones en el histórico. Por cuestiones de velocidad y facilidad en la implementación, proponemos estas dos simplificaciones:

1. Usar *k-means* como algoritmo de clusterización.
2. Para obtener mayor velocidad de cálculo, no ejecutar una clusterización cada vez que se evalúa una transacción nueva $\vec{t}_{n+1}$, sino que aprender solo cuando el fitness de la clusterización se degrada. En el caso de usar *k-means* como algoritmo, el fitness es fácilmente calculable usando la suma de cuadrados intra-cluster (Kovács et al., 2006).

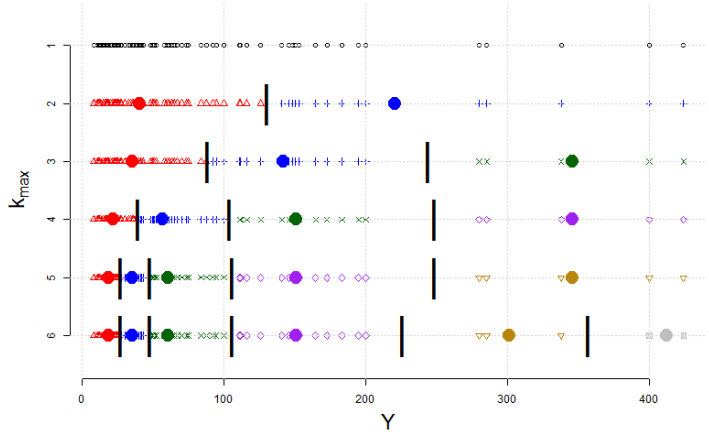
De esta manera tenemos dos parámetros a controlar: la cantidad de clusters k y el fitness mínimo necesario para mantener una discretización. Dado un conjunto de valores $y_1, \ldots, y_n$ de una variable continua Y en el histórico de la misma tarjeta, este es el mecanismo propuesto:

Algoritmo 2 Discretización de variable Y continua por valores en histórico de una tarjeta

function DISCRETIZAR($k_{\text{mín}}, k_{\text{máx}},\ fitness_{\text{mín}},\ y_1, \ldots, y_n$)
  $k = k_{\text{mín}}$; $fitness = 0$
  while $(k < k_{\text{máx}} + 1)$ & $(fitness < fitness_{\text{mín}})$ **do**
    $\{S_i\}_{i=1}^{m} = \text{clustersKMeans}(y_1, \ldots, y_n, k)$
    $\mu_i = \dfrac{1}{|S_i|} \displaystyle\sum_{y_j \in S_i} y_j, \quad i = 1, \ldots, h$
    $betweenss = \displaystyle\sum_{i=1}^{k} \sum_{y_j \in S_i} \|y_j - \mu_i\|^2$
    $totalss = \displaystyle\sum_{y_j \in S} \|y_j - \mu\|^2$ donde $\mu = \dfrac{1}{n} \displaystyle\sum_{i=1}^{n} y_i$
    $fitness = 1 - betweenss/totalss$
    $k = k + 1$
  end while
end function

El mismo busca en cada caso que k sea el *menor* valor posible que genere una clusterización con fitness determinado, ysa que elegir un valor demasiado grande (como caso extremo, k igual a la cantidad de transacciones en el histórico) generaría una clusterización de fitness muy bueno o perfecto, pero con poca utilidad práctica. Como para valores de k demasiado bajos no se puede garantizar que encontremos una partición que cumpla los requisitos, el valor de k se incrementa en 1 en cada etapa hasta alcanzar el fitness requerido.

En la figura 4.3 ilustramos el funcionamiento del algoritmo mediante un ejemplo. Además, en la figura 4.4, mostramos cómo los clusters pueden reorganizarse al llegar transacciones nuevas. Como podemos ver, las variables Y_d resultantes del algoritmo no tienen por qué tener una distribución determinada. Por otro lado, el problema motivador se resuelve con este algoritmo, pues el cálculo del fitness evitará que una transacción nueva quede aislada en un cluster pequeño que podría fusionarse con otros cercanos más numerosos.



$k_{\text{máx}}$	$fitness$	$H(Y_d)$
2	0.689	0.57
3	0.895	0.86
4	0.936	1.62
5	0.942	2.07
6	0.960	2.13

Figura 4.3: Demostración del funcionamiento del algoritmo. En la gráfica de la izquierda, sobre cada recta horizontal se muestran los mismos 117 valores de una variable Y continua (montos de la transacción) obtenidas seleccionando al azar una tarjeta en el conjunto de entrenamiento de una de las bases. Desde arriba hacia abajo se muestra el resultado de la ejecución del algoritmo con parámetros $fitness = 0.99$, $k_{\text{mín}} = 2$ y $k_{\text{máx}}$ variando desde 2 hasta 6 (excepto en la primera recta horizontal que presenta los datos sin procesar). El color de los puntos representa el cluster asignado. Los puntos circulares rellenos de mayor tamaño son los centroides de cada cluster y las barras negras verticales los bordes de los mismos. El cuadro de la derecha muestra el fitness alcanzado por cada discretización ejecutada y la entropía de la variable discretizada resultante.

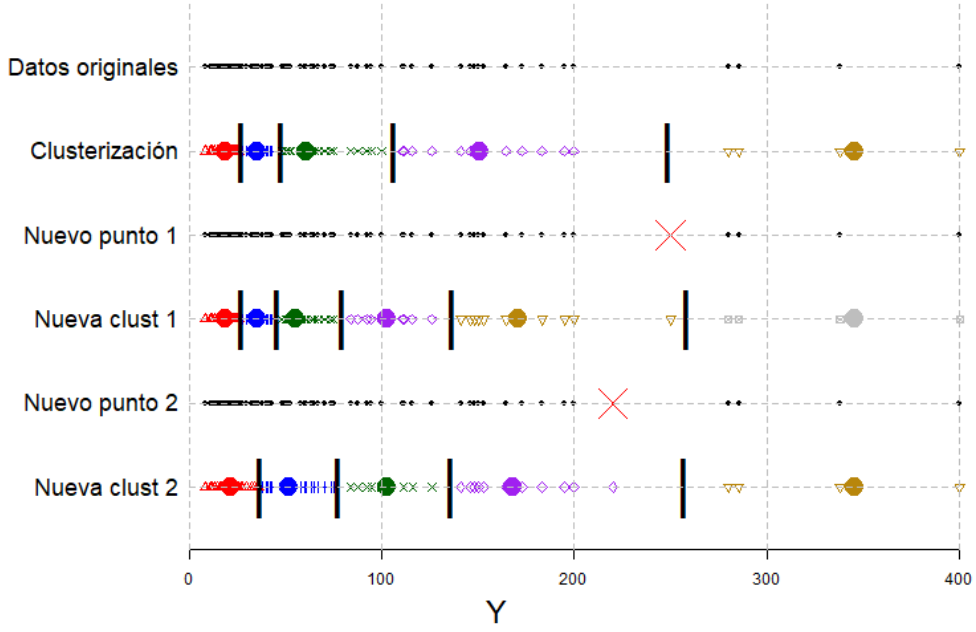


Figura 4.4: Ejemplo de adaptación de los clusters ante nuevas transacciones. En cada recta horizontal se representan las mismas instancias de la variable Y utilizadas en el ejemplo anterior. El color de los puntos representa el cluster asignado. Los puntos circulares rellenos de mayor tamaño son los centroides de cada cluster y las barras negras verticales los bordes de los mismos. Los puntos marcados con una cruz roja son instancias artificiales correspondientes a una nueva transaccion entrante. En orden de arriba a abajo, consideramos los siguientes escenarios: 1) datos originales, 2) clusterización de los datos anteriores, 3) datos originales con una nueva instancia y' 4) clusterización de los datos en 3, 5) datos originales con una nueva instancia y'' 6) clusterización de los datos en 5

Las discusiones presentadas en esta sección fueron pensadas siempre en el conjunto de histórico suficiente. Para el caso de histórico bajo, este algoritmo podría dar una alternativa para discretizar variables en cada subconjunto de histórico fijo y respetar las diferencias en las distribuciones condicionales de Y según la variable X que determine las particiones.

4.4. Agrupación de variables

Una de las posibles mejoras que surge más naturalmente es relativa a los histogramas, ya que los usamos tanto en el cálculo de la rareza como en el de la distancia al vecino más cercano. Es sabido que un histogramas de variables conjuntas es más explicativo que varios histogramas individuales. Esto es particularmente importante al hacer detección de anomalías ya que un

outlier de la distribución conjunta no tiene por qué ser outlier de ninguna de las distribuciones marginales:

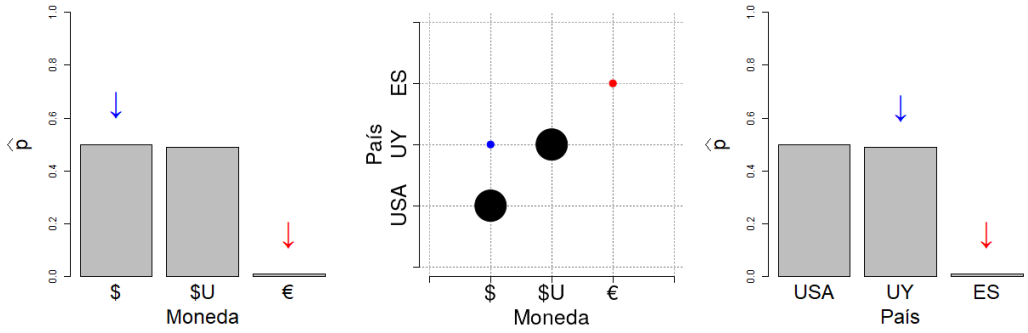


Figura 4.5: Ejemplo artificial donde $X =$ "moneda" e $Y =$ "país". De izquierda a derecha se muestran el histograma de X, la distribución conjunta de (X, Y) y el histograma de Y. En el gráfico central el tamaño de los círculos representa la cantidad de transacciones que toma cada par posible de valores (x, y). Los valores marcados con flechas en las distribuciones marginales corresponden a los marcados en la distrbución conjunta.

En este ejemplo sintético se puede ver como el valor $(x, y) = (\$, \texttt{UY})$ es un outlier en la distribución conjunta de (X, Y) por más que $x = \$$ no lo es según la distribución de X ni $y = \texttt{UY}$ lo es en la distribución de Y.

Esta observación sencilla nos induciría a pensar que quizás, cuando queremos usar histogramas en los cálculos para el Score de Outlierness, sería más conveniente considerar agrupaciones de algunas variables en lugar de las mismas individualmente. Sin embargo, aquí debemos hacer una disgresión según la relación que exista entre X e Y:

- Si X e Y son independientes (o con una dependencia muy baja), no solo la distribución conjunta no aportará información nueva, sino que también se pueden generar problemas de imprecisión si el tamaño de sus recorridos es grande (lo cual ocurre frecuentemente cuando se introducen variables continuas discretizadas según una cantidad muy grande de intervalos).
- Si X e Y están altamente correlacionadas, la estimación del histograma conjunto tiene una robustez similar a la de los histogramas marginales, por lo que la determinación de outliers según la distribución de (X, Y) es más confiable. Sin embargo, en el caso extremo en que ambas variables estén completamente correlacionadas[3], la agrupación (X, Y) está en

[3]Lo cual ocurre por ejemplo con las variables "identificador de comercio" y "MCC"

biyección con alguna de sus coordenadas, por lo que no resulta de gran utilidad.

Teniendo en cuenta estas observaciones, nos enfocaremos en determinar la correlación existente para cada par de variables X e Y, y en base a los resultados, buscaremos candidatos a agrupar. Proponemos agrupar secuencialmente una lista de candidatos, para poder determinar si en cada paso, la modificación mejoró o empeoró la detección. Si la lista se construye ordenando los pares de variables mas correlacionadas en forma descendente, esperaríamos que las primeras agrupaciones no empeoren el desempeño y las últimas si, por lo que buscaríamos el punto óptimo entre las agrupaciones intermedias.

Como estamos en el caso en que las variables a considerar son nominales, evidentemente no podemos recurrir a los criterios más comunes como el coeficiente de correlación de Pearson, aunque sí a la información mutua. Para utilizar una misma escala para cada par de variables, proponemos usar el siguiente cálculo como medida de la correlación entre X e Y nominales:

$$\rho(X,Y) = \frac{I(X,Y)}{\text{mín}\{H(X), H(Y)\}}$$

que por propiedades de la información mutua se encuentra normalizado en el intervalo $[0; 1]$.

Capítulo 5

Procedimiento final e implementación

En los capítulos anteriores nos dedicamos a presentar la metodología para calcular el Score de Outlierness de una transacción y las maneras en que el resultado puede ser optimizado para mejorar el desempeño de un modelo que lo utilice como característica. En este capítulo presentaremos cómo sería el flujo completo de creación de un modelo de detección de fraudes incorporando la metodología presentada.

Como resultado de la investigación que dio lugar a este documento, se generó un conjunto de scripts escritos en los lenguajes SQL, R y C++, los cuáles al ser ejecutados en orden permiten ir diseñando un modelo de detección sobre una base determinada. Los mismos están escritos de tal manera que todas las decisiones manuales de diseño se reducen a la asignación de valores a un conjunto de variables al principio del script. Luego, al ejecutar uno de ellos, se produce una salida (generalmente en forma de tablas o gráficas) que al ser interpretada por el analista le permite identificar cuáles son los valores óptimos y tomar nuevas decisiones en la etapa siguiente. En consecuencia, no es necesario saber programar en ninguno de estos lenguajes para poder ejecutar la secuencia de scripts y obtener el modelo.

5.1. Resumen del procedimiento

Detallamos a continuación cómo es el proceso de ejecución de los scripts:

1. **Limpieza de datos y pre-procesamiento** En la etapa inicial, se estudian los datos crudos para entender su significado y extraer las columnas de la base que aportan valor. Los estudios que se hacen puede dividirse en:

   - **Limpieza:** Estudio de la cantidad de valores faltantes, valores únicos, outliers, etc.
   - **Chequeo de coherencias**: Se verifica el sentido semántico de los valores que toma una variable (por ejemplo: el modo de entrada 5 debe corresponder a tarjetas con chip bajo los estándares de la mayoría de los emisores) y de las relaciones entre variables (por ejemplo: correspondencia inyectiva entre identificador de comercio y MCC).
   - **Análisis de distribuciones temporales:** Se estudia la variación de las variables a lo largo del período que abarca la base. En caso que las distribuciones de las variables por sí mismas o condicionadas a la etiqueta de fraude varíen significativamente, podría haber errores en los datos.

2. **Determinación de cortes**
   Como mencionamos en la sección 2.4.4, los cortes permiten segmentar la base de manera de entrenar modelos más especializados en un tipo de transacciones. Como en cada subconjunto de datos cambia significativamente el valor de las variables y los valores óptimos de los metaparámetros, los cortes se definen inmediatamente después de terminar con la limpieza. Proponemos ejecutar el algoritmo de sugerencias visto en 2.4.4 y que el usuario realice la elección manual en base al resultado. A partir de este paso, se construirán dos modelos para cada corte determinado: uno para el conjunto de histórico suficiente y otro para el conjunto de histórico bajo.
3. **Optimización del parámetro m_H y la función de rareza**
   Una vez creados los cortes y antes de ejecutar la selección de variables, proponemos optimizar los parámetros del Score de Outlierness: de esta manera, podemos extraer variables lo más explicativas posibles sobre el conjunto total de las mismas. Si ejecutáramos la selección antes del cálculo del Score de Outlierness, podríamos perder una variable que por si misma podría no ser útil pero cuyo Score de Outlierness sí podría serlo

bajo los valores adecuados de algún metaparámetro.
Si bien la manera de obtener un óptimo global sería probando con una grilla en el espacio completo de los metaparámetros, esto se vuelve impracticable dada la complejidad y los tiempos de cálculo. Por lo tanto, proponemos hacer una optimización *greedy* secuencial, comenzando en el primer paso con tres parámetros (para no simplificar excesivamente la búsqueda del óptimo global):

a) La cantidad de histórico mínimo m_H (siguiendo el procedimiento visto en la sección 4.1.2), ya que cada valor que probemos modifica el conjunto de transacciones. Una vez que se halle un valor para m_H, podemos fijar los conjuntos de histórico bajo y suficiente, lo que facilita la ejecución de las optimizaciones posteriores.
b) La función de rareza, pues es sencillo desde un punto de vista del cálculo y para poder fijar la fórmula utilizada para el resto del procedimiento.
c) La cantidad de árboles al usar `XGBoost` como clasificador (ver 2.1), ya que de esta manera podemos estudiar rápidamente la interacción entre los parámetros anteriores y la complejidad del modelo.

Las optimizaciones posteriores se realizarán secuencialmente de a un parámetro por vez.

4. **Optimización de parámetros del Score de Outlierness en el conjunto de histórico suficiente**
Cuando termina el paso 3, el único parámetro del Score de Outlierness restante que tiene una influencia sobre la determinación de los conjuntos de histórico bajo y suficiente es la ventana temporal. Por lo tanto, para poder fijar los conjuntos para todo el resto del procedimiento, proponemos seguir la secuencia con este parámetro. Luego, vendrían el resto de los parámetros relativos al conjunto de histórico suficiente que vimos en el capítulo 4:

4.1 Ventana temporal.
4.2 Uso de variables continuas.
4.3 Agrupación de variables.

El orden entre los pasos 4.2 y 4.3 es indistinto *a priori*, pues el primero refiere a variables continuas mientras que el segundo a variables

nominales.

5. **Optimización de parámetros del Score de Outlierness en el conjunto de histórico insuficiente**
   Si bien el orden de los pasos siguientes podría modificarse, proponemos la siguiente secuencia de ejecución para reflejar lo hecho en el paso 4:

   5.1 Conjuntos de histórico fijo.

   5.2 Uso de variables continuas.

   5.3 Agrupación de variables.

   Con la ejecución de este paso finalizaría el proceso de extracción de características usando el Score de Outlierness.

6. **Acumuladores y selección de variables**
   Una vez que se ha fijado los valores óptimos para el cálculo del Score de Outlierness del conjunto de variables, sería momento de ejecutar una selección de variables e incluir a los acumuladores. Proponemos ejecutar los pasos en ese orden para minimizar el número de comparaciones que habrá que hacer para incluir un acumulador.
   La selección de variables se ejecuta en este proceso mediante la utilización de una medida intrínseca de los clasificadores por árboles: la *importancia* de la variable Louppe et al. (2013). La misma puede calcularse como el porcentaje de ganancia de pureza que una variable aportó en el conjunto de nodos del bosque. Nuestra propuesta es:

   - Entrenar un modelo con todas las N variables.
   - Ordenarlas según su importancia.
   - Remover secuencialmente las n peores variables (con $n = i \times \frac{N}{k}$, $i = 1, \ldots, k$).
   - En cada remoción, entrenar otro modelo.

   Finalmente, se elegirá la remoción que obtenga un buen balance entre mejorar (o mantener) el desempeño y descartar el mayor número posible de variables.
   Luego, para finalizar el proceso de extracción de características y obtener el conjunto definitivo de variables, solamente quedaría incluir a los acumuladores. Como se discutió en la sección 2.4.3, proponemos incluir los acumuladores de a uno y velando que mantengan una baja correlación con el resto de las variables y un alto poder de clasificación. El algoritmo

ha sido explicado en detalle en la sección mencionada.
En conclusión, la secuencia de ejecución sería:

6.1 Selección usando remoción secuencial por importancia.
6.2 Introducción secuencial de acumuladores.

7. **Optimización de parámetros del clasificador**
   En esta etapa realizamos una exploración de grilla para optimizar los parámetros del clasificador. En el caso particular de la implementación en R de `XGBoost`, que fue el utilizado, existen decenas de parámetros posibles (xgboost developers, 2016). Por limitaciones de tiempo y memoria, nos restringimos a los más importantes que son los siguientes:

   - **Cantidad de rondas:** cantidad máxima de árboles que se entrenaran. Pueden ser menos, ya que el algoritmo vigila el error en un conjunto de validación cruzada y deja de crear árboles cuando el error en validación cruzada no baja desde los últimos n árboles.
   - η **o *learning rate*:** corrige el aporte de cada nuevo árbol por un factor de η, de manera que valores más pequeños generan más cantidad de árboles.
   - γ: indica la ganancia mínima de pureza que debe haber para que se construya un nodo. Valores más altos generan modelos menos complejos.
   - **Profundidad máxima:** profundidad máxima de cada árbol construido. Valores más altos dan modelos más complejos.

8. **Evaluación en conjunto de test**
   Finalmente, el proceso terminaría con un reporte de los resultados en el conjunto de test de cada uno de los modelos entrenados. En esta etapa proponemos usar la metodología que llamamos *evaluación realista* y que consiste en evaluar el resultado del modelo suponiendo que detectar un fraude acarrea el bloqueo de la tarjeta (después de un período determinado de tiempo) y por lo tanto evita la concreción de todos los fraudes futuros. Consideramos que esta manera de entender los resultados es la que más se aproxima a cómo funcionan los modelos en producción (ver el apéndice 4 para más detalles).

5.2. El paquete `scoreOutlierness`

Dada la gran cantidad de funciones que fue necesario definir (todas las relativas al cálculo del Score de Outlierness y algunas auxiliares) se creó un paquete de R llamado `scoreOutlierness`. El mismo engloba todas las funciones necesarias para la ejecución del procedimiento descrito anteriormente[1] El paquete alcanza velocidades muy altas de procesamiento por hallarse escrito parcialmente en C++ (gracias al paquete `Rcpp` de R). Para tener una idea de cuan rápido se ejecutan los cálculos, mostraremos los resultados del siguiente experimento: para cada una de las bases, se sortearon 1000 tarjetas y se crearon N variables nominales artificiales discretizando el monto de la transacción usando cuantiles aleatorios. El valor de N se hizo variar y se repitieron los sorteos 10 veces para cada uno. Las corridas se realizaron en una computadora con sistema operativo Windows 10, procesador Intel Core i7 de 2.40GHz, memoria de 16GB y disco duro en estado sólido. Los resultados se pueden ver en la gráfica 5.1.

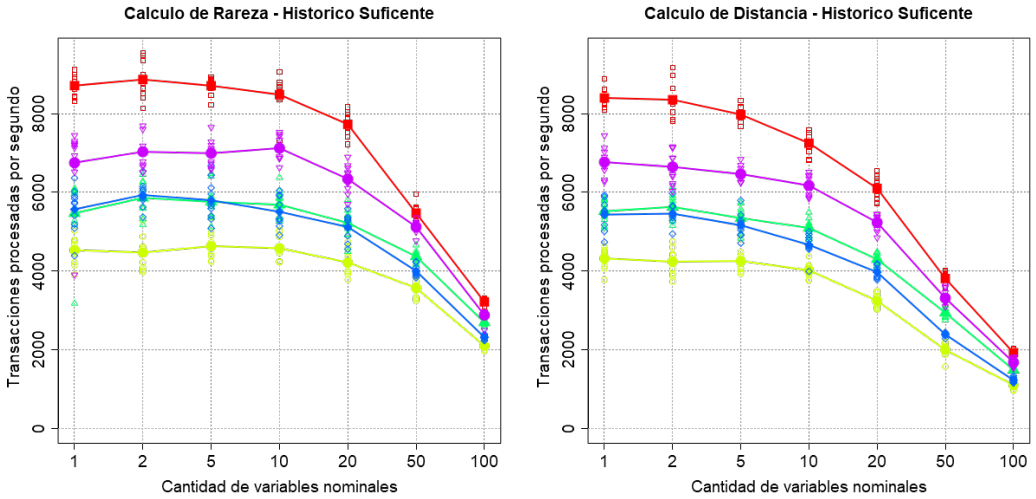


Figura 5.1: Velocidad de cálculo del score de outlierness (medida en transacciones procesadas por segundo) en función de la cantidad de variables nominales en cada una de las bases utilizadas. A la izquierda, se determinaron variables usando el cálculo de rareza. A la derecha, se usó el cálculo de distancia. Las curvas representan los valores promedio de los 10 sorteos realizados por cada base y cada valor de N.

[1] Con excepción del algoritmo de discretización adaptativa visto en 4.3.3, ya que la puesta en producción de este algoritmo implicaría el almacenamiento de información extra para cada cliente (la discretización parcial para compararla con una transacción nueva), lo cual no estaba dentro del alcance del proyecto.

Capítulo 6

Resultados

En esta sección presentaremos y analizaremos el desempeño de los modelos finales en los conjuntos de test.

Recordemos que según el procedimiento resumido en el capítulo anterior, en cada una de las bases disponibles se determinaron una cantidad variable de cortes y se entrenaron dos modelos por cada uno: uno para histórico suficiente y otro para histórico bajo. Lo anterior fue siempre realizado utilizando los conjuntos de entrenamiento como datos de entrada de los clasificadores y de validación para la optimización de parámetros. El conjunto de test fue reservado para esta etapa final y solamente fue utilizado para reportar los resultados que presentaremos en este capítulo. Algunas consideraciones adicionales sobre los resultados presentados son las siguientes:

- En todos los casos, para poder trazar la variabilidad de los resultados, se partió el conjunto de test en 10 subconjuntos y se evaluó sucesivamente en una elección de 9 de ellos.
- Para entender que tan buena es la clasificación de los modelos, graficaremos algunas de las medidas presentadas en la sección 2.4.1. Sin embargo, observaremos que en algunas figuras aparecen nubes de puntos en lugar de curvas continuas (que sería la apariencia esperada). Esto se debe principalmente a que muchos de los conjuntos de test utilizados para la evaluación tienen pocos fraudes, o los mismos presentan muchos valores repetidos de score[1]. En menor medida, este efecto puede influir

[1]Esto puede pasar si las transacciones en el corte tienden a parecerse mucho o sí el modelo óptimo aprendido es muy sencillo (por ejemplo, un conjunto de árboles de poca profundidad).

en las curvas del desempeño global estimado provocando "saltos". Sin embargo, esto no impide que podamos hacernos una idea del desempeño del modelo.

6.1. Desempeños globales

Si bien la manera de construir el sistema de detección de fraudes que proponemos es mediante un ensamble de modelos para los cuales reportamos siempre curvas de desempeño por cada uno de los umbrales posibles[2], cada submodelo tomará una decisión binaria a la hora de ser implementado, por lo que el desempeño global final será uno solo y no parametrizable: se detecterán en promedio una determinada cantidad de fraudes y se elevarán cierto número de falsas alarmas. Esto es en definitiva lo que observará la institución financiera y con lo que evaluará al modelo.

Por lo tanto, debemos encontrar la manera óptima de configurar umbrales para cada uno de los submodelos de forma que se maximice la detección global. Este problema no es evidente a priori, pues los submodelos no funcionan de forma independiente. En el apéndice 4 se discute en detalle como es este procedimiento de estimación de desempeño conjunto, pero a modo de resumen, se utilizan dos técnicas:

1. Como para cada submodelo se tiene un conjunto de umbrales posibles a utilizar, el desempeño global debe ser estimado en un principio estudiando el conjunto de todas las combinaciones posibles de elecciones de umbrales por submodelo. Sin embargo, dentro de esas posibilidades existirán algunas que serán claramente preferibles a otras. La *frontera de Pareto* nos servirá para quedarnos con un subconjunto de las combinaciones de umbrales que permitan describir el desempeño global mediante una curva para ser analizada manualmente.
2. Como en su historia, las transacciones de una misma tarjeta pueden ser evaluadas por distintos submodelos (si realiza transacciones en distintos cortes, o si pasa de tener histórico bajo a suficiente), la detección temprana de un fraude podría prevenir todos los fraudes futuros, suponiendo que todas las alarmas son procesadas por un analista de riesgo en al menos v horas. Por lo tanto, para determinar la cantidad de

[2]Los desempeños de cada submodelo se pueden ver en el apéndice 1.

fraudes detectados por una combinación de umbrales, usaremos distintos valores de v y marcaremos como detectados todos los fraudes cometidos v unidades de tiempo después de un fraude correctamente alertado. El parámetro v será llamado la *ventana de evaluación realista*.

Nuevamente, ambas técnicas se discuten más profundamente en el apéndice 4. Como resultado, tendremos una desempeño global estimado que ilustraremos mediante las curvas ROC y de Efectividad vs $FPPF$ (ver 2.4.1 para la definición de las medidas). Para la curva de ROC incluiremos ademas una estimación[3] del AUC (área bajo la curva).

Línea base

Tal y como mencionamos en la sección 2.4.2, no podemos comparar nuestros resultados contra los obtenidos mediante el procedimiento anterior de armado de modelos ya que los mismos son excesivamente optimistas (por presentar overfitting). Por lo tanto, la línea de base que usaremos será la dada por el negocio. En general, las expectativas dependen del contexto, pero a grandes rasgos, se esperan valores de $FPPF$ menores a 50 para efectividades mayores al 70 %. Para la medida del AUC ROC, existen posibles *rules of thumb* (Tape, 2019) que indican que se pueden considerar buenos aquellos modelos con valores superiores a 0.8. Valores superiores a 0.9 se consideran muy buenos o excelente.

Adicionalmente, en el caso particular de las bases 2.1 y 2.2, se cuenta para cada transacción con el valor de un score de riesgo implementado por la entidad emisora de las tarjetas de crédito. Para usar este score como línea de base en esos conjuntos de datos, graficaremos sus curvas de desempeño usando las mismas medidas que para nuestros modelos.

En las figuras 6.1, 6.2, 6.3, 6.5 y 6.7 a continuación se presentan los resultados discutidos anteriormente. Las gráficas 6.4 y 6.6 refieren a los scores emisores utilizados para compararnos en el contexto de las bases de datos 2.1 y 2.2.

[3]Para esto, se utilizó una aproximación polinomial en los casos en que existen "saltos".

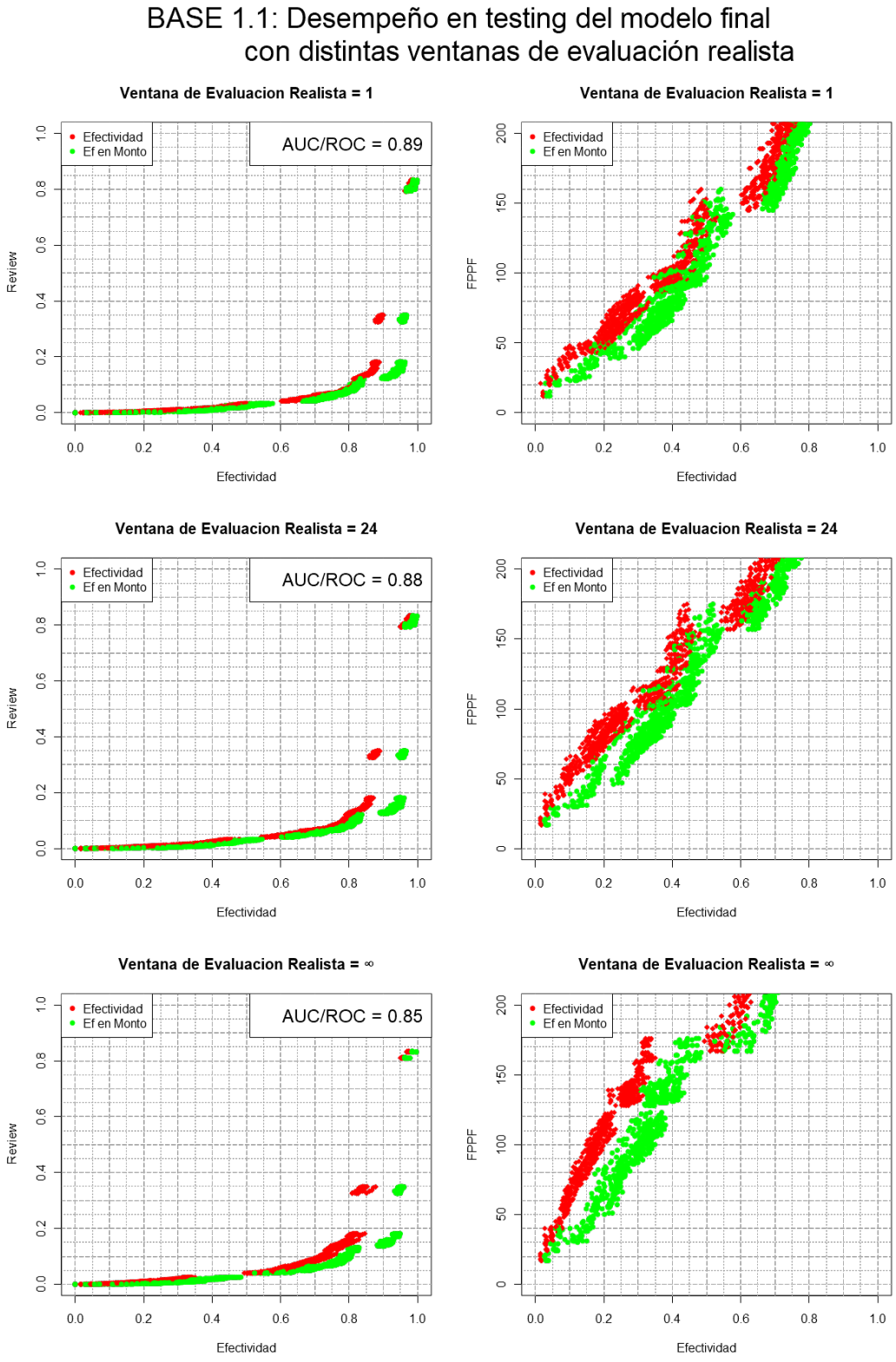


Figura 6.1: Base 1.1: Desempeño en Test de los modelos de histórico bajo y suficiente en todos los cortes funcionando en simultáneo utilizando la evaluación realista con distintas ventanas de tiempo.

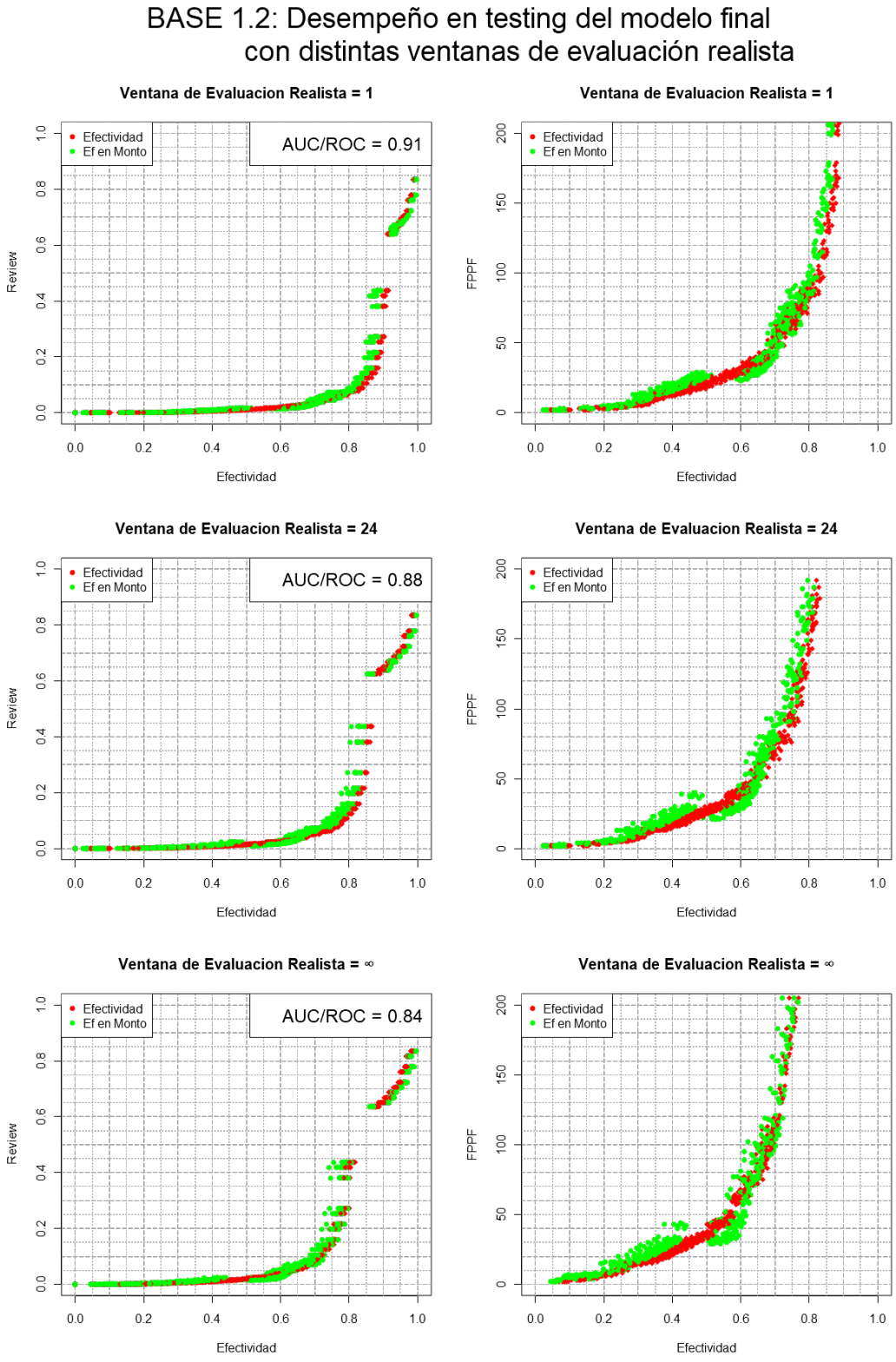


Figura 6.2: Base 1.2: Desempeño en Test de los modelos de histórico bajo y suficiente en todos los cortes funcionando en simultáneo utilizando la evaluación realista con distintas ventanas de tiempo.

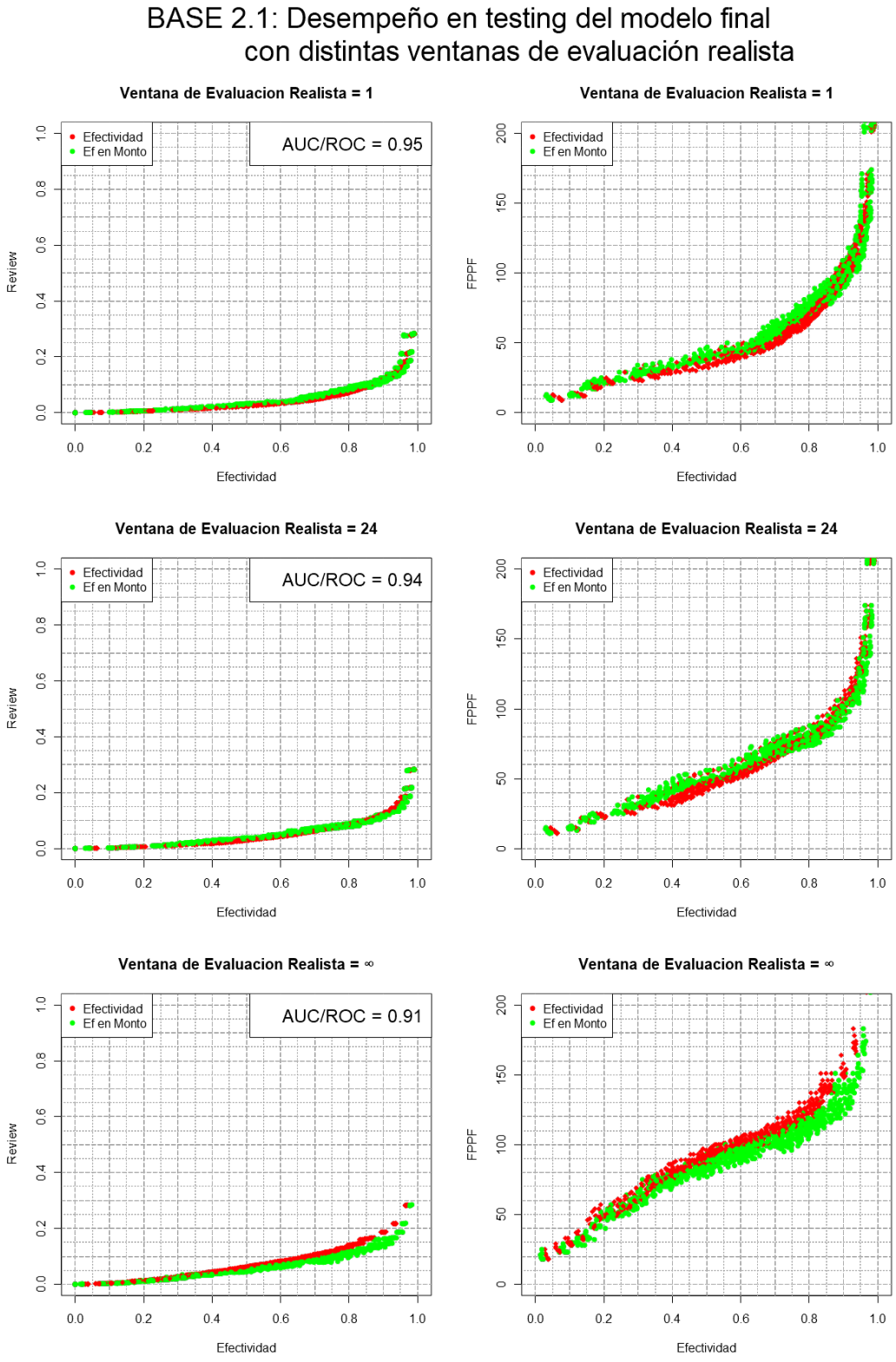


Figura 6.3: Base 2.1: Desempeño en Test de los modelos de histórico bajo y suficiente en todos los cortes funcionando en simultáneo utilizando la evaluación realista con distintas ventanas de tiempo.

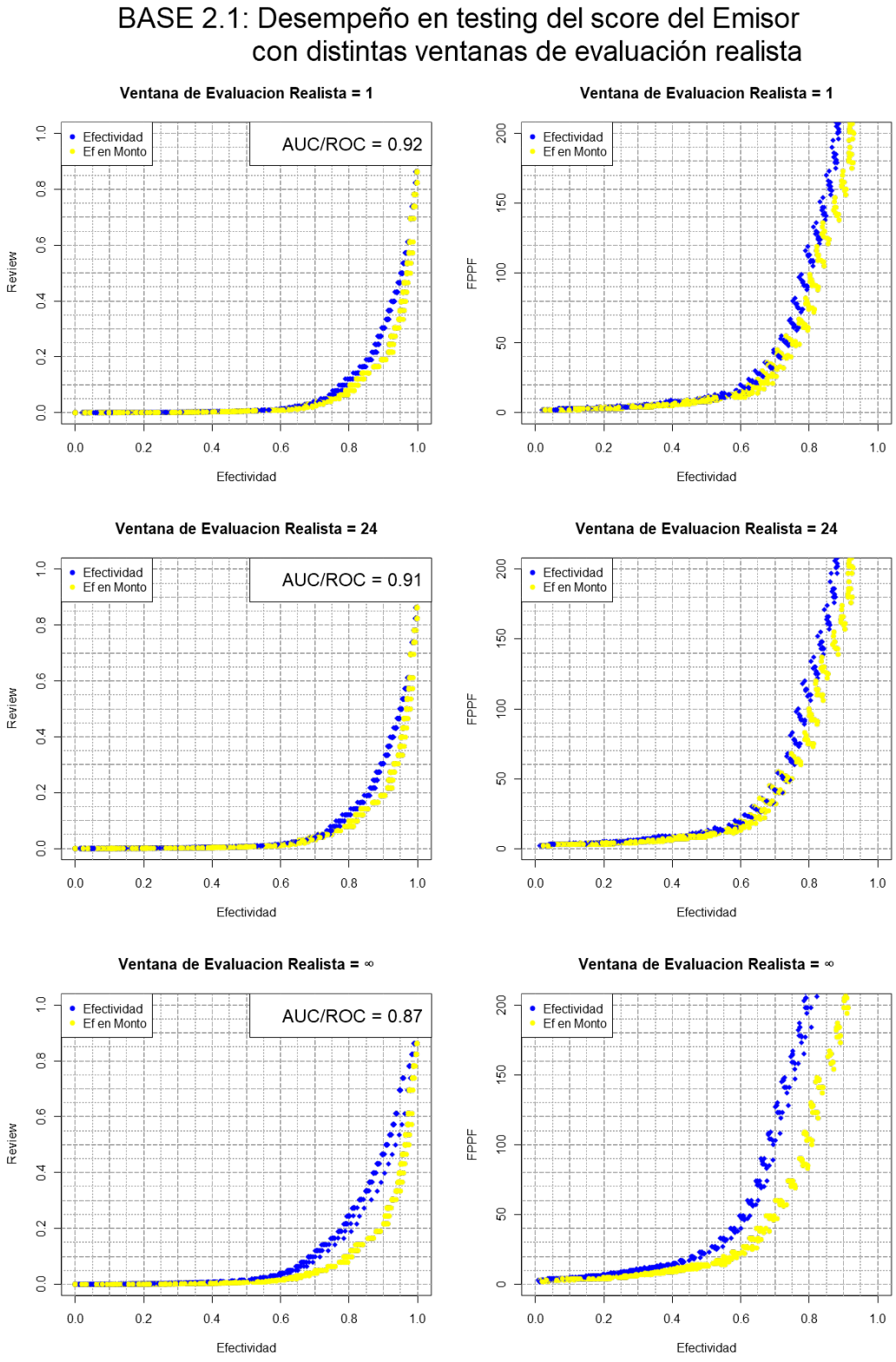


Figura 6.4: Base 2.1: Desempeño en Test del score de riesgo implementado por el emisor utilizando la evaluación realista con distintas ventanas de tiempo.

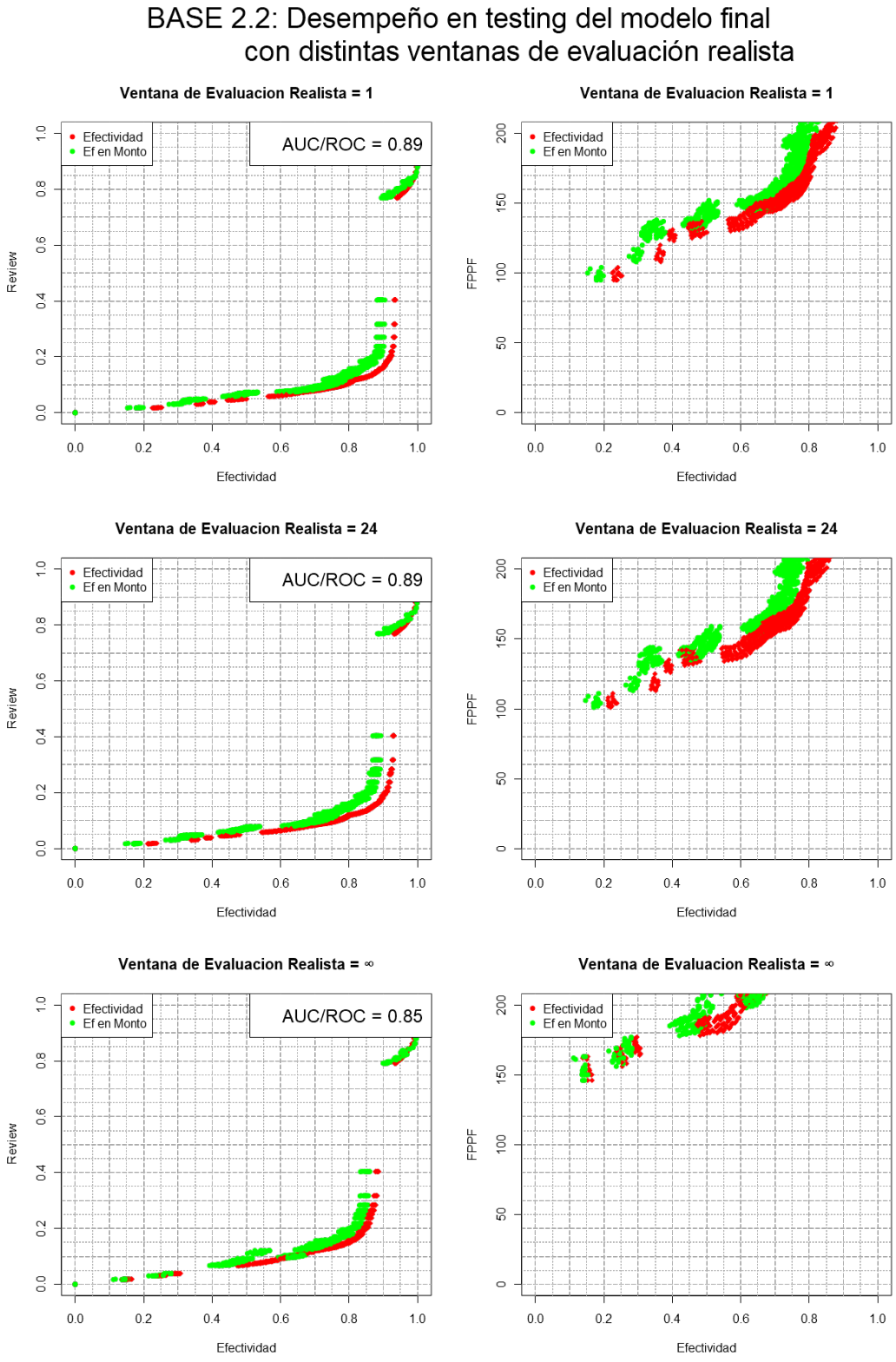


Figura 6.5: Base 2.2: Desempeño en Test de los modelos de histórico bajo y suficiente en todos los cortes funcionando en simultáneo utilizando la evaluación realista con distintas ventanas de tiempo.

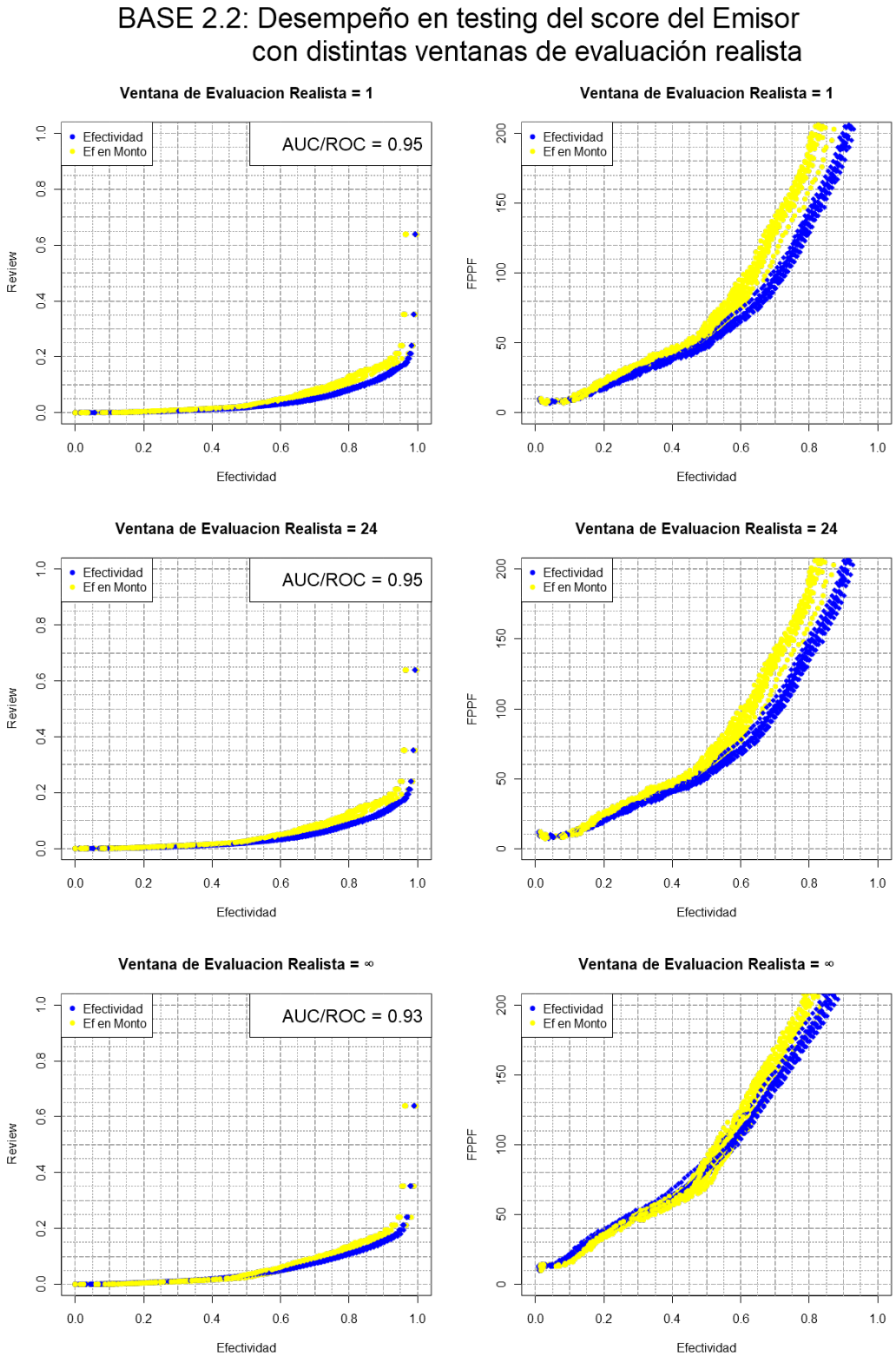


Figura 6.6: Base 2.2: Desempeño en Test del score de riesgo implementado por el emisor utilizando la evaluación realista con distintas ventanas de tiempo.

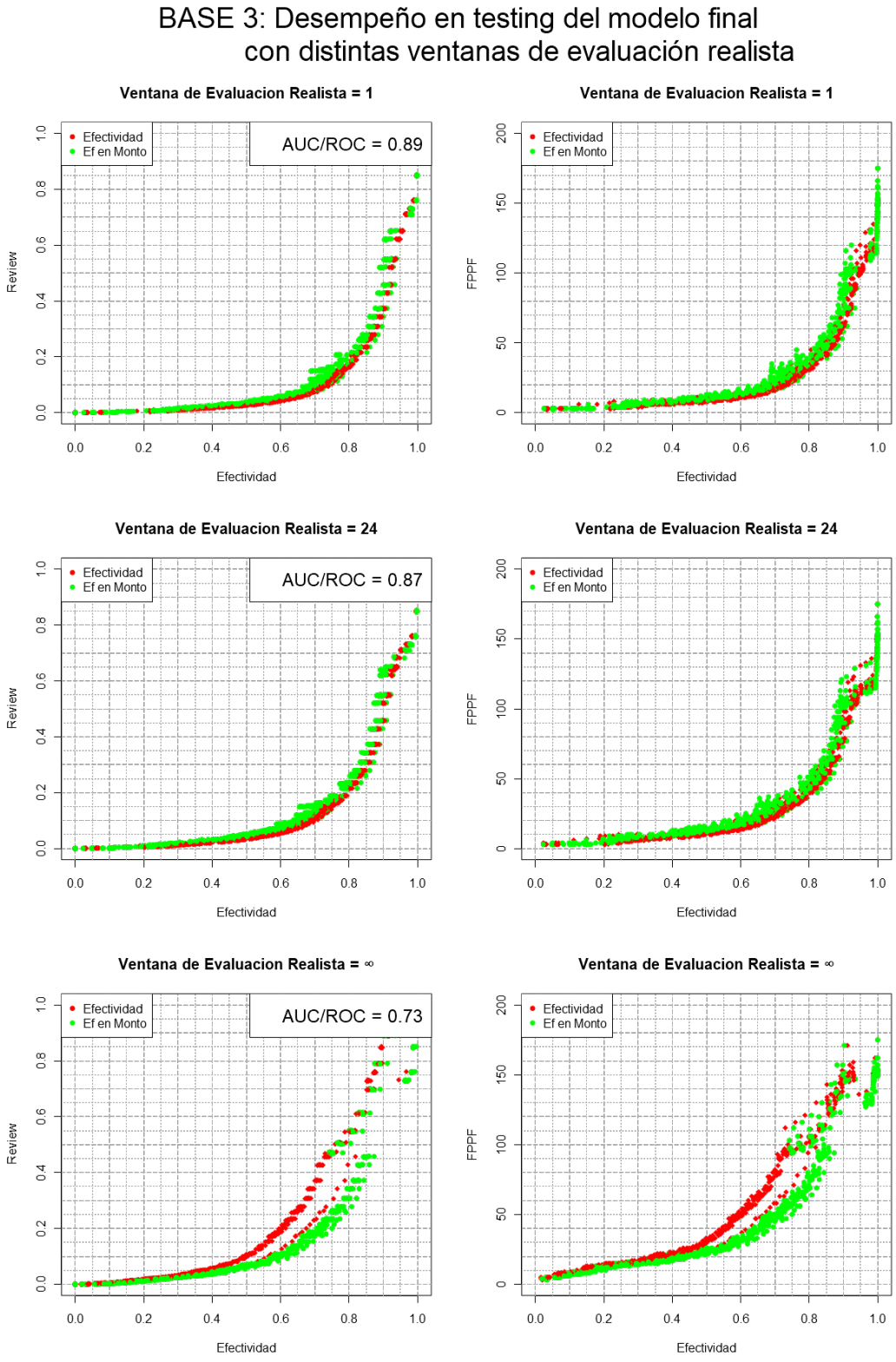


Figura 6.7: Base 3: Desempeño en Test de los modelos de histórico bajo y suficiente en todos los cortes funcionando en simultáneo utilizando la evaluación realista con distintas ventanas de tiempo.

Del análisis de las figuras anteriores observamos lo siguiente:

1. Si usamos las expectativas del negocio como línea base (como dijimos anteriormente: $FPPF$ menores a 50 para efectividades mayores al 70 %), las bases 1.1 y 2.2 tienen desempeños peores a los esperados para todos los valores explorados de ventana realista. En las bases 1.2, 2.1 y 3 los resultados se aproximan a lo pretendido al usar $v = 1$ o $v = 24$ (incluso lo superan en el caso de la base 3).
2. Al compararnos contra el score otorgado por la entidad financiera, vemos que el desempeño del mismo es mejor en ambas bases (2.1 y 2.2) con una diferencia menos pronunciada en la base 2.1.
3. Los valores estimados para la AUC ROC pueden considerarse buenos o muy buenos de acuerdo con la *rule of thumb* presentada (Tape, 2019), ya que se mantienen por encima de 0.84 para todos los modelos si no usamos evaluación realista (salvo para la base 3) y por encima de 0.87 y 0.89 en todos los modelos usando ventanas de 24 horas y 1 hora respectivamente.

Estos resultados son optimistas y nos indican que hemos propuesto un procedimiento prometedor. Para interpretar los resultados, debemos tener en cuenta que las líneas bases con las que nos hemos comparado son altamente exigentes. Adicionalmente, las bases de datos utilizadas son de un período de tan solo un año y no presentan algunos datos básicos (por ejemplo: características del cliente, límites de crédito y disponible, información geográfica precisa, etc). Teniendo en cuenta que los emisores que implementaron el score contra el cual nos comparamos en las bases 2.1 y 2.2. tienen más datos y de un período más extendido, afirmamos que las condiciones de ambos modelos no son equitativas. El habernos aproximado a las líneas bases en el contexto en el cual desarrollamos nuestros modelos puede interpretarse como un resultado favorable y que nos da confianza en que, con datos más ricos, podría mejorarse aún más.

Esto también es apoyado por la evidencia aportada por las AUC ROC obtenidas: si bien esta medida puede tomar valores altos aún cuando la cantidad de errores promedio en términos absolutos es inaceptable para la institución financiera (porque el FPR usado en la curva ROC no tiene en cuenta la dilución del conjunto), el haber obtenido valores considerados buenos indica que nuestros modelos consiguen una buena separación entre clases.

6.2. Desempeños por corte

El estudio del desempeño en cada corte puede ayudarnos a entender como funciona el modelo global en detalle. Para entender el contexto en que se entrenó cada modelo, mostramos en las siguientes tablas las particiones que fueron definidas en cada base:

Índice	Distr. fraudes	Distr. legítimas	Dilución	Descripción
-1	6 %	0.06 %	1:400.000	Pagos
0	50.3 %	3.6 %	1:68.000	Modo de entrada Chip
1	1.7 %	39 %	1:220	Modo de entrada Manual
2	1.3 %	13.4 %	1:500	Modo de entrada Presente / Exterior
3	40.6 %	44 %	1:4.500	Modo de entrada Presente / Nacionales

Tabla 6.1: Resumen de los cortes definidos en la base 1.1.

Índice	Distr. fraudes	Distr. legítimas	Dilución	Descripción
-1	18 %	0 %	∞	Pagos
0	5.3 %	2.7 %	1:2.000	Modo de entrada Chip
1	4 %	49.4 %	1:100	Modo de entrada Manual
2	11 %	20.4 %	1:600	Modo de entrada Presente / Últ. trx. $\leq$ 1 h
3	61.4 %	27.4 %	1:2.500	Modo de entrada Presente / Últ. trx. $>$ 1 h

Tabla 6.2: Resumen de los cortes definidos en la base 1.2.

Índice	Distr. fraudes	Distr. legítimas	Dilución	Descripción
0	6.7 %	0 %	∞	Monto $= 0$
1	93.3 %	100 %	$1:700$	Monto > 0

Tabla 6.3: Resumen de los cortes definidos en la base 2.1.

Índice	Distr. fraudes	Distr. legítimas	Dilución	Descripción
0	11.5 %	4 %	1:4.000	Modo de entrada Chip
1	15.5 %	84 %	1:270	Modo de entrada Manual
2	73 %	12 %	1:8.500	Modo de entrada Presente

Tabla 6.4: Resumen de los cortes definidos en la base 2.2.

Índice	Distr. fraudes	Distr. legítimas	Dilución	Descripción
0	80 %	4 %	1:23.000	Modo de entrada Chip
1	20 %	96 %	1:230	Otros

Tabla 6.5: Resumen de los cortes definidos en la base 3.

En el apéndice 2 se muestran las tablas con el detalle de los valores óptimos obtenidos para cada parámetro. Adicionalmente, los desempeños de cada submodelo se pueden ver en el apéndice 1.

6.2.1. Importancia de las variables del Score de Outlierness

Como fue mencionado en las secciones 2.1 y 5.1, la *importancia* de una variable aleatoria es una medida intrínseca que es calculada en los algoritmos de clasificación que utilizan árboles de decisión. El valor de la misma para una característica X_h se obtiene sumando todas las ganancias en pureza[4] en cada uno de los nodos en que sea haya utilizado X_h. En nuestro caso, al estar utilizando XGBoost como algoritmo de clasificación, contamos con esta medida para cada uno de los clasificadores entrenados.

En esta sección buscamos estudiar los valores experimentales obtenidos para las importancias, de manera de entender cuanto poder predictivo aportan las variables calculadas usando el Score de Outlierness. Como no todas las bases utilizan las mismas características, las agruparemos en cuatro tipos: acumuladores, variables propias de la base (las descritas en la sección 2.3 o características derivadas de forma directa de ellas) y variables calculadas como el Score de Outlierness usando las fórmulas de rareza o distancia respectivamente. De la misma manera, no todos los clasificadores obtienen la misma ganancia en pureza total (dependerá de cada conjunto de datos), por lo que utilizaremos la *importancia relativa* de la variable (medida como la importancia divida entre la ganancia total de pureza del clasificador).

En el conjunto de tablas 6.6 presentamos los valores de importancias relativas según tipo de variable en cada uno de los clasificadores utilizados para armar el modelo final en cada base.

[4]Distintos algoritmos pueden usar distintas funciones de pureza, pero a los efectos de la importancia, no es relevante la función específica utilizada sino solamente la ganancia en pureza que aporta una característica.

Base 1.1

Tipo de variable	Porcentaje de ganancia					
	Corte 1		Corte 2		Corte 3	
	Hist. Bajo	Hist. Suf.	Hist. Bajo	Hist. Suf.	Hist. Bajo	Hist. Suf.
Acumuladores	15 %	22 %	33 %	5 %	3 %	8 %
Var. de la base	54 %	9 %	19 %	6 %	40 %	22 %
Var. Distancia SO	8 %	49 %	32 %	24 %	50 %	48 %
Var. Rareza SO	22 %	20 %	16 %	65 %	7 %	22 %

Base 1.2

Tipo de variable	Porcentaje de ganancia					
	Corte 1		Corte 2		Corte 3	
	Hist. Bajo	Hist. Suf.	Hist. Bajo	Hist. Suf.	Hist. Bajo	Hist. Suf.
Acumuladores	3 %	0 %	0 %	8 %	0 %	2 %
Var. de la base	50 %	23 %	58 %	20 %	62 %	35 %
Var. Distancia SO	16 %	16 %	27 %	38 %	22 %	0 %
Var. Rareza SO	31 %	61 %	15 %	34 %	16 %	63 %

Base 2.1

Tipo de variable	Porcentaje de ganancia	
	Corte 1	
	Hist. Bajo	Hist. Suf.
Acumuladores	13 %	5 %
Var. de la base	55 %	28 %
Var. Distancia SO	16 %	42 %
Var. Rareza SO	16 %	25 %

Base 2.2

Tipo de variable	Porcentaje de ganancia			
	Corte 1		Corte 2	
	Hist. Bajo	Hist. Suf.	Hist. Bajo	Hist. Suf.
Acumuladores	26 %	7 %	42 %	0 %
Var. de la base	64 %	10 %	28 %	3 %
Var. Distancia SO	5 %	12 %	21 %	24 %
Var. Rareza SO	5 %	71 %	9 %	73 %

Base 3

Tipo de variable	Porcentaje de ganancia	
	Corte 1	
	Hist. Bajo	Hist. Suf.
Acumuladores	3 %	0 %
Var. de la base	60 %	58 %
Var. Distancia SO	27 %	24 %
Var. Rareza SO	10 %	18 %

Tabla 6.6: Importancias relativas (medidas como el porcentaje de ganancia relativo a la ganancia total) agrupadas según el tipo de variable en los modelos entrenados para todas las bases.

Observando las tablas, podemos concluir lo siguiente:

- Las variables obtenidas mediante el Score de Outlierness tienen un alto poder predictivo en la mayoría de los modelos: en 12 de 20 casos representan aproximadamente el 50 % o más de la ganancia total en pureza. Esta relación es aún mejor si nos restringimos a los modelos entrenados con histórico suficiente (9 de 10).
- No existe una tendencia clara que indique una preferencia entre características propias de la base y las obtenidas con el Score de Outlierness. Tampoco ocurre que siempre sea mejor utilizar la forma de cálculo de la rareza o de la distancia. Sólamente los acumuladores presentan porcentajes de ganancia notoriamente menores a los otros tipos de variables, aunque esto se debe seguramente a que en el procedimiento final, se busca minimizar el número de acumuladores utilizados en los modelos para evitar su tendencia al overfitting (tal y como fue discutido en la sección 2.4.3).
- En todas las bases, el poder predictivo combinado de las variables del Score de Outlierness es superior en el conjunto de histórico suficiente que en el conjunto de histórico bajo. Este resultado está en concordancia con la idea intuitiva de que los cálculos realizados para determinar el perfil de un cliente deberían ser más precisos a medida que tengo más transacciones en su histórico.

En resumen, el estudio de la importancia de las variables aporta evidencia que indica que las variables calculadas usando el Score de Outliernes resultan útiles y genera una buena separación entre transacciones fraudulentas y legítimas.

Capítulo 7

Conclusiones finales

7.1. Conclusiones

La investigación de esta tesis estuvo focalizada en la aplicación de técnicas de aprendizaje automático en la detección de fraudes en tarjetas de crédito: un problema dificil, con motivaciones en la industria financiera y de alta aplicabilidad. Se dispusieron de datos abundantes en cantidad, pero limitados en cuanto al número y diversidad de variables.

La investigación comenzó con un análisis crítico del estado del arte en detección de fraude y del procedimiento de creación de modelos previo, que concluyó con la propuesta de varias mejoras. Luego, se exploró la posibilidad de utilizar aproximaciones relacionadas con la detección de anomalías para construir un score útil para la detección. Recorriendo ese camino, se produjo un aporte importante de este trabajo: el desarrolló de una función de distancia no trivial entre instancias en un espacio con características nominales, que se probó logra una mayor discriminación que la distancia de Hamming. Tambén se introdujo la noción de rareza de una instancia (con respecto a su histórico previo). Ambas aproximaciones (rareza y distancia) pueden ser usadas como método de extracción de características para obtener como resultado un conjunto de variables nuevas que probaron ser útiles para la detecciíon de fraudes y lograr una buena discriminación entre clases.

Juntando las mejorías al procedimiento anterior y este nuevo método de extracción de características, se construyó un procedimiento nuevo que abarca todas las etapas de construcción de un modelo de clasificación, y se implementó su totalidad en un conjunto de scripts escritos en R. Los mismos se encuentran

completamente documentados y tienen una performance computacional alta por encontrarse parcialmente implementados en C++.

Usando los scripts anteriores, se construyeron modelos que fueron evaluados en 5 bases de datos distintas. Los resultados obtenidos en bases de testing indican un desempeño prometedor que, si bien no llega a la línea base con algunos de los criterios establecidos, resultan muy positivos teniendo en cuenta la dificultad del problema y la calidad de los datos. En particular, el estudio de las importancias de las variables construidas mediante el Score de Outlierness da evidencias de que el método propuesto produce variables útiles, que se pueden beneficiar de más datos y mayor cantidad de histórico por tarjeta.

7.2. Trabajo futuro

Para finalizar este documento, presentamos a continuación las principales líneas de trabajo que quedaron abiertas y podrían surgir a partir de la investigación desarrollada:

1. Por como está definido, el Score de Outlierness, el mismo funciona mejor cuantas más variables haya disponibles, pues permite generar más características y puede determinar perfiles más precisos combinando estas variables. Por lo tanto, sería interesante probar esta herramienta en contextos donde haya más características disponibles y con menor interdependencia.
2. En una línea similar a la anterior, las bases utilizadas tenían la característica de no poseer grandes cantidades de histórico para cada cliente. Si se aplicara el Score de Outlierness en contextos donde los clientes poseen más cantidad de histórico (por ejemplo: operaciones bancarias, logueos en una aplicación móvil, etc), se podría investigar que tan precisos son los perfiles al aumentar la cantidad de datos disponible.
3. Como fue mencionado en 2.4.4, se propuso trabajar en un algoritmo de sugerencia de cortes que permita una optimización más precisa, teniendo en cuenta la interacción entre las variables y librando la menor cantidad de decisiones posibles al usuario. La implementación probablemente sería similar a la de un árbol de decisión con una función de pureza que tenga en cuenta las restricciones vistas anteriormente (para no generar nodos con una cantidad demasiado pequeña de transacciones).

4. Cuando se definió el cálculo del Score de Outlierness mediante la distancia al vecino más cercano en 3.2.2, se omitió la posibilidad de usar las distancias a otros K-vecinos más próximos. Variando el parámetro K o promediando las distancias (como ejemplo de combinación) podríamos generar nuevas variables.
5. La construcción de perfiles tipo para el caso de histórico bajo (ver 4.1.1) se hizo de manera sencilla, pero podrían explorarse alternativas más sofisticadas. Una alternativa sería por ejemplo usar clusterizaciones en el conjunto de los clientes: en un caso tal, al haber variables nominales (por ejemplo el modo de entrada o rubro de compra más frecuente de un cliente), la distancia nominal definida en 3.2.2 podría ser de utilidad para darle una métrica al espacio de los clientes. Esta es otra aproximación que podría verse beneficiada por la presencia de datos más diversos: si se contaran con características propias del cliente (edad, sexo, perfil socioeconómico, domicilio, etc) podrían utilizarse las mismas para agrupar los clientes y reservar la información transaccional solamente para la detección.
6. Los perfiles que determina el Score de Outlierness carecen de una característica importante que podría ser explorada en el futuro: la temporalidad. Utilizando un análisis diferente (series de tiempo, por ejemplo), se podría investigar la existencia de anomalías por el patrón temporal en que se ejecutan las transacciones, algo que no se ve si solo nos restringimos al análisis de histogramas.
7. La combinación no supervisada de las rarezas y distancias para formar un solo score de fraude no ha sido explorada en detalle en este documento ya que en todas las bases se contaba con la etiqueta de fraude, y tal como vimos en 3.4, en tal caso es mejor utilizar una aproximación supervisada. Sin embargo, sería interesante determinar si existe una manera no supervisada de calcular pesos para cada variable de rareza y distancia de manera que la combinación ponderada tenga mejor desempeño que la combinación simple. Si se encontrara una forma de cálculo de pesos que funcionara universalmente mejor, sería posible implantar modelos en instituciones financieras nuevas que aún no contaran con un histórico (o el mismo no estuviera etiquetado).
8. Si bien no todas las bases de datos utilizadas tienen exactamente las mismas características, existen muchas de ellas que son similares e incluso

que siguen un estándar. Por lo tanto, se podría trabajar en utilizar patrones aprendidos en una base de datos para aportar a la detección en otros contextos. Es posible que en casos de instituciones financieras con contextos similares, esto pueda mejorar los resultados.

Referencias bibliográficas

A. C. Bahnsen, et al. (2015). 'Detecting Credit Card Fraud using Periodic Features'. In *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*, pp. 208–213. IEEE.

A. C. Bahnsen, et al. (2016). 'Feature engineering strategies for credit card fraud detection'. *Expert Systems with Applications* **51**:134–142.

S. Bhattacharyya, et al. (2011). 'Data mining for credit card fraud: A comparative study'. *Decision Support Systems* **50**(3):602–613.

L. Breiman & E. Schapire (2001). 'Random forests'. In *Machine Learning*, pp. 5–32.

V. Chandola, et al. (2007). 'Anomaly Detection: A Survey' .

N. V. Chawla, et al. (2002). 'SMOTE: Synthetic Minority Over-sampling Technique'. *Journal of Artificial Intelligence Research* **16**:321–357.

T. Chen & C. Guestrin (2016). 'XGBoost: A Scalable Tree Boosting System'. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pp. 785–794, New York, NY, USA. ACM.

A. Dal Pozzolo, et al. (2014). 'Learned lessons in credit card fraud detection from a practitioner perspective'. *Expert systems with applications* **41**(10):4915–4928.

A. Daneshpazhouh & A. Sami (2014). 'Entropy-based outlier detection using semi-supervised approach with few positive examples'. *Pattern Recognition Letters* **49**:77–84.

E. Duman & M. H. Ozcelik (2011). 'Detecting credit card fraud by genetic algorithm and scatter search'. *Expert Systems with Applications* **38**(10):13057–13063.

Y. Freund & R. E. Schapire (1999). 'A Short Introduction to Boosting'.

N. S. Halvaiee & M. K. Akbari (2014). 'A novel model for credit card fraud detection using Artificial Immune Systems'. *Applied Soft Computing* **24**:40–49.

A. K. Jain, et al. (2000). 'Statistical pattern recognition: A review'. *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE* **22**(1):4–37.

S. Jha, et al. (2012). 'Employing transaction aggregation strategy to detect credit card fraud'. *Expert systems with applications* **39**(16):12650–12657.

F. Kovács, et al. (2006). 'Cluster Validity Measurement Techniques'.

J. Z. Lei & A. A. Ghorbani (2012). 'Improved competitive learning neural networks for network intrusion and fraud detection'. *Neurocomputing* **75**(1):135–145.

G. Louppe, et al. (2013). 'Understanding variable importances in forests of randomized trees'. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, & K. Q. Weinberger (eds.), *Advances in Neural Information Processing Systems 26*, pp. 431–439. Curran Associates, Inc.

N. Mahmoudi & E. Duman (2015). 'Detecting credit card fraud by modified Fisher discriminant analysis'. *Expert Systems with Applications* **42**(5):2510–2516.

Y. Sahin, et al. (2013). 'A cost-sensitive decision tree approach for fraud detection'. *Expert Systems with Applications* **40**(15):5916–5923.

T. Saito & M. Rehmsmeier (2015). 'RESEARCH ARTICLE The Precision-Recall Plot Is More Informative than the ROC Plot When Evaluating Binary Classifiers on'.

T. G. Tape (2019). 'The Area Under an ROC Curve'. http://gim.unmc.edu/dxtests/ROC3.htm.

V. Van Vlasselaer, et al. (2015). 'APATE: A novel approach for automated credit card transaction fraud detection using network-based extensions'. *Decision Support Systems* **75**:38–48.

U. von Matt & D. Dacunha-Castelle (2014). 'Improving credit card fraud detection with calibrated probabilities' .

J. West & M. Bhattacharya (2016). 'Intelligent financial fraud detection: a comprehensive review'. *Computers & Security* **57**:47–66.

C. Whitrow, et al. (2009). 'Transaction aggregation as a strategy for credit card fraud detection'. *Data Mining and Knowledge Discovery* **18**(1):30–55.

xgboost developers (2016). 'XGBoost Parameters'. https://xgboost.readthedocs.io/en/latest/parameter.html.

F. Zhao, et al. (2018). 'A Filter Feature Selection Algorithm Based on Mutual Information for Intrusion Detection'. *Applied Sciences* **8**:1535.

APÉNDICES

Apéndice 1

Gráficas

En este apéndice se incluyen las gráficas de los modelos individuales construidos según el procedimiento resumido en 5.1. Para cada base de datos, se construyen dos modelos por cada corte determinado: uno para el conjunto de histórico suficiente y otro para el conjunto de histórico bajo. Los resultados reportados en las siguientes figuras son sobre el conjunto de test. En todos los casos, para tener noción de la variabilidad de los resultados, se partió el conjunto de test en 10 subconjuntos y se evaluó sucesivamente en una elección de 9 de ellos. Se grafican las curvas de ROC, es decir, el *Review* en función de la Efectividad (ver sección 2.4.1).

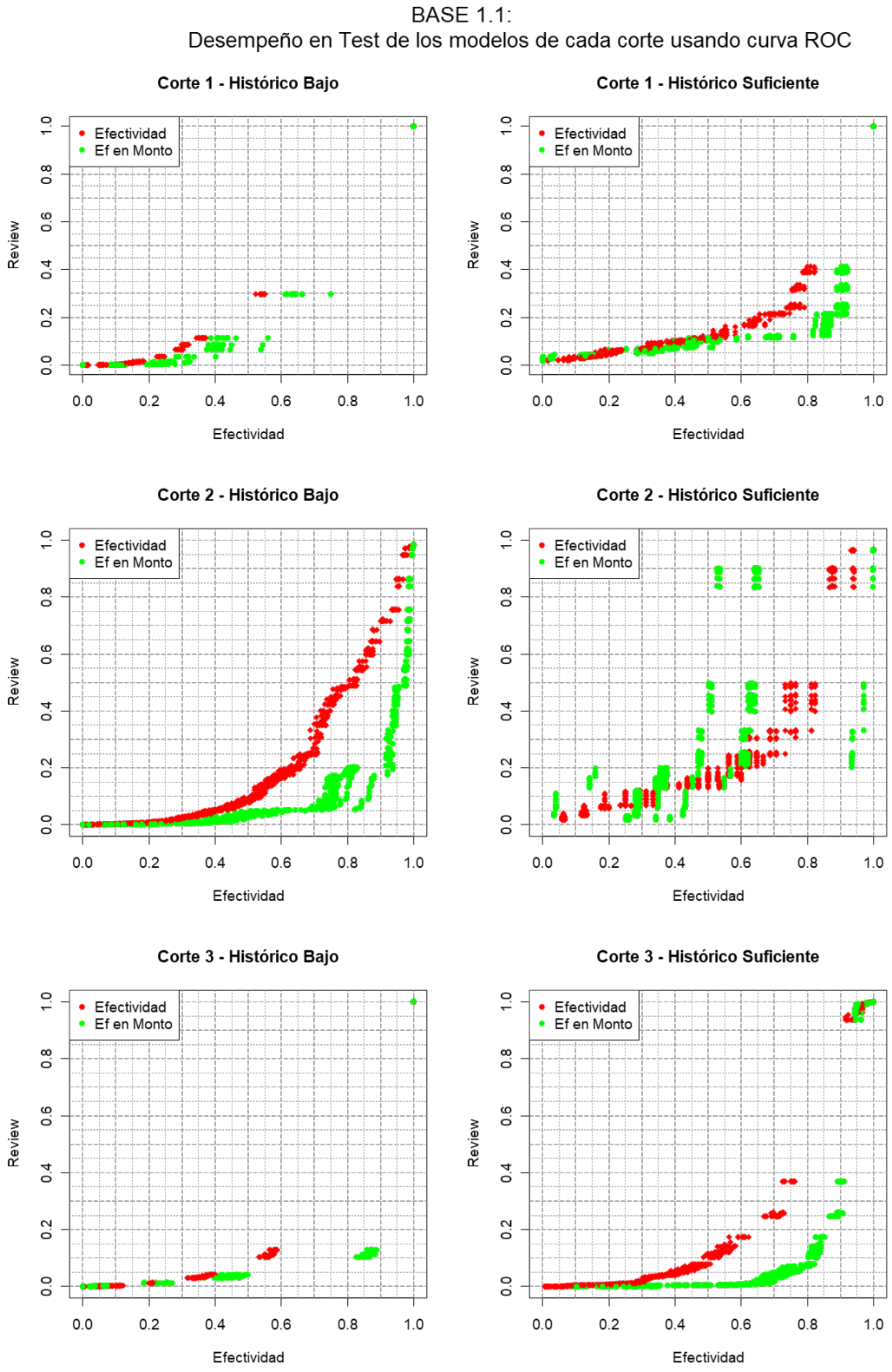


Figura 1.1: Desempeño individual de todos los modelos en la base 1.1 según cantidad de histórico y corte, usando como medida la curva de Efectividad vs Review (curva ROC).

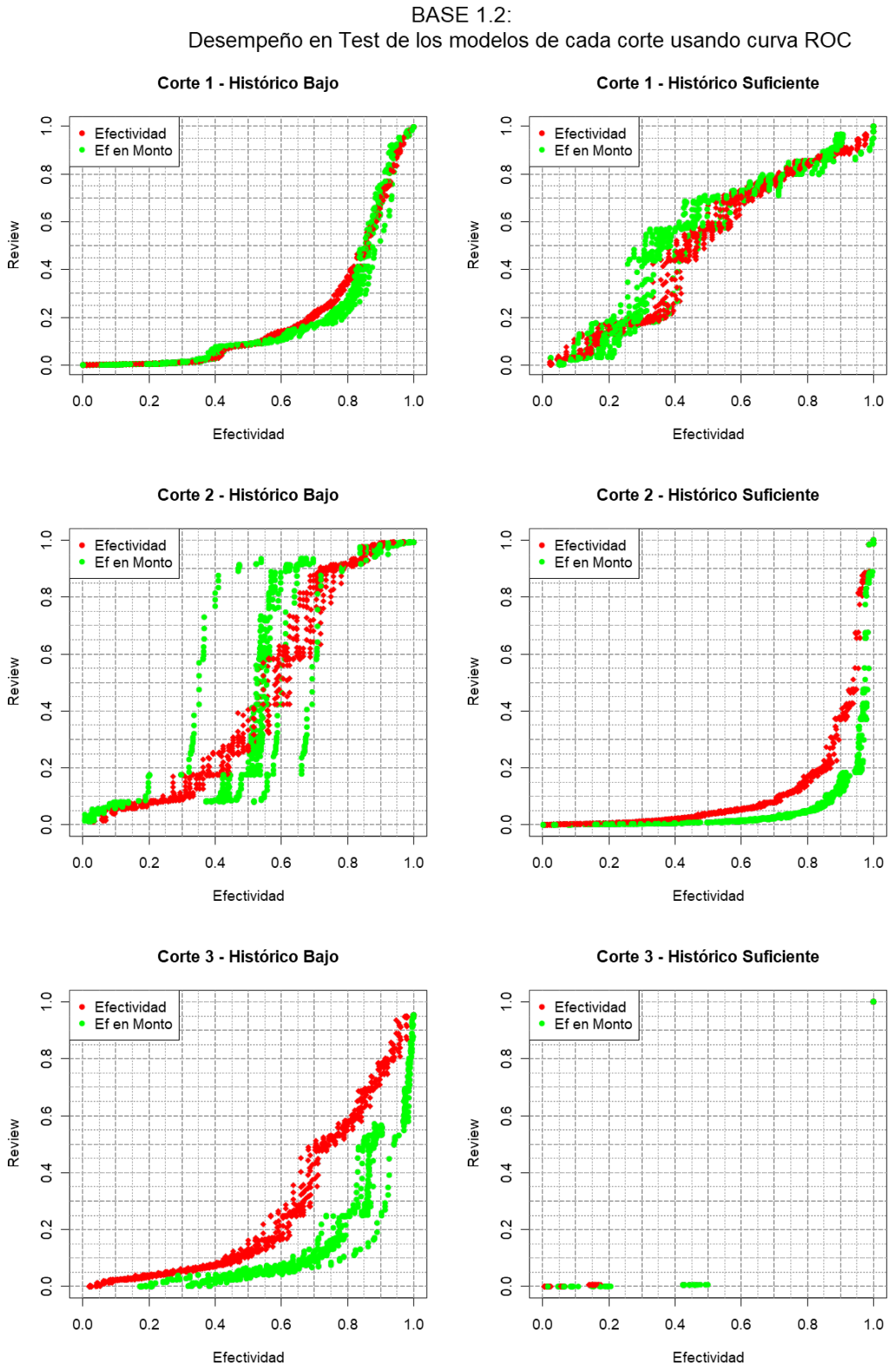


Figura 1.2: Desempeño individual de todos los modelos en la base 1.2 según cantidad de histórico y corte, usando como medida la curva de Efectividad vs Review (curva ROC).

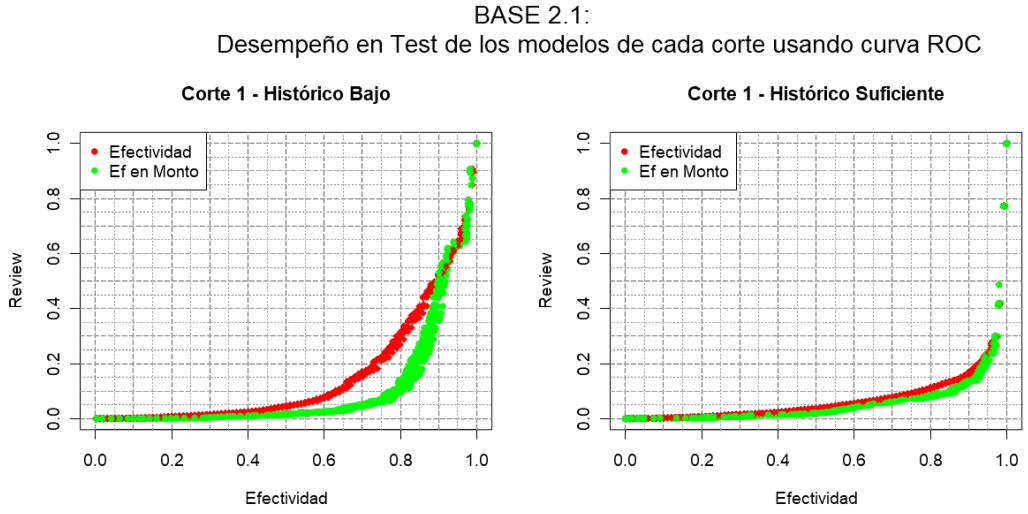


Figura 1.3: Desempeño individual de todos los modelos en la base 2.1 según cantidad de histórico y corte, usando como medida la curva de Efectividad vs Review (curva ROC).

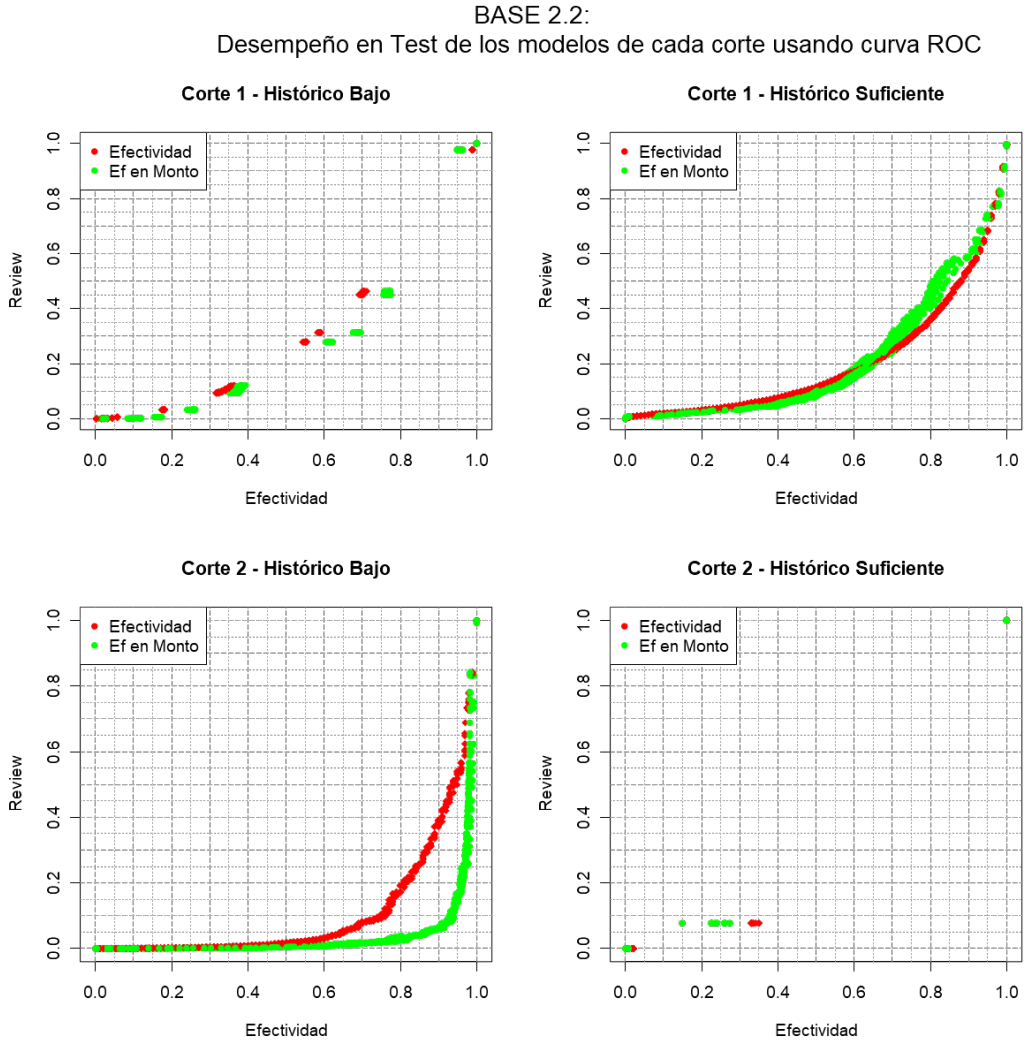


Figura 1.4: Desempeño individual de todos los modelos en la base 2.2 según cantidad de histórico y corte, usando como medida la curva de Efectividad vs Review (curva ROC).

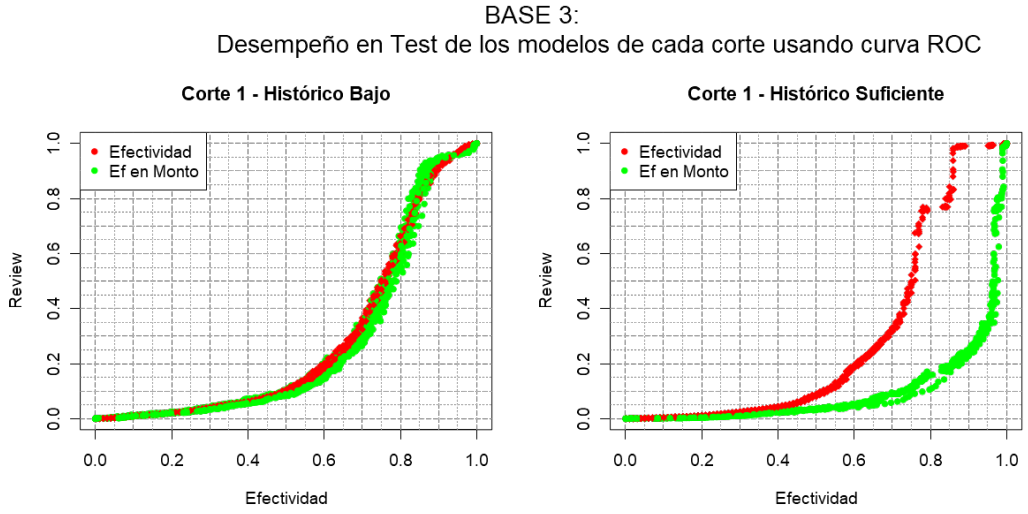


Figura 1.5: Desempeño individual de todos los modelos en la base 3 según cantidad de histórico y corte, usando como medida la curva de Efectividad vs Review (curva ROC).

Apéndice 2

Tablas

En este apéndice se muestran las tablas de parámetros óptimos obtenidos para cada uno de los modelos finales en las bases disponibles La descripción detallada de cada parámetro y su influencia sobre los modelos fue discutida en los capítulos 3 y 4.

En las tablas de resumen de los cortes, se utiliza la notación vista en la sección 2.4.4 para determinar los índices de cada corte.

	Parámetro	Corte 1	Corte 2	Corte 3
Paso 3	m_H	126	184	102
	Func. de rareza Hist. Suf.	3	2	1
	Cant. árboles Hist. Suf.	10	10	10
	Func. de rareza Hist. Bajo	1	1	2
	Cant. árboles Hist. Bajo	50	10	10
Paso 4	Ventana superior (horas)	12	0	0
	Ventana inferior (horas)	360	∞	168
	Dist. var. cont. Hist. Suf.	2, $b = 2$	-	2, $b = 1$
	Cant. agrup. Rareza Hist. Suf.	11	2	7
	Cant. agrup. Dist. Hist. Suf.	0	7	10
Paso 5	Dist. var. cont. Hist. Bajo	4, $b = 10$	3, $b = 10$	4, $b = 10$
	Cant. agrup. Rareza Hist. Bajo	9	11	11
	Cant. agrup. Dist. Hist. Bajo	0	0	10
Paso 6	Cant. var. Removidas Hist. Suf.	7	10	7
	Cant. var. Removidas Hist. Bajo	5	6	5
	Cant. acum. incl. Hist. Suf	2	5	3
	Cant. acum. incl. Hist. Bajo	1	6	1
Paso 7	Cant. rondas Hist. Suf.	50	50	10
	η Hist. Suf.	0.01	0.3	0.3
	γ Hist. Suf.	1.0	0.5	0
	Prof. máx. Hist. Suf.	10	12	10
	Cant. rondas Hist. Bajo	50	50	10
	η Hist. Bajo	0.001	0.5	0.01
	γ Hist. Bajo	0	1.0	0.1
	Prof. máx. Hist. Bajo	4	8	12

Tabla 2.1: Valores óptimos de parámetros explorados en la base 1.1.

	Parámetro	Corte 1	Corte 2	Corte 3
Paso 3	m_H	58	12	13
	Func. de rareza Hist. Suf.	3	2	3
	Cant. árboles Hist. Suf.	10	10	10
	Func. de rareza Hist. Bajo	1	2	2
	Cant. árboles Hist. Bajo	100	10	10
Paso 4	Ventana superior (horas)	0	3	0
	Ventana inferior (horas)	1488	168	1488
	Dist. var. cont. Hist. Suf.	-	-	-
	Cant. agrup. Rareza Hist. Suf.	6	0	2
	Cant. agrup. Dist. Hist. Suf.	8	7	11
Paso 5	Dist. var. cont. Hist. Bajo	-	-	-
	Cant. agrup. Rareza Hist. Bajo	12	0	0
	Cant. agrup. Dist. Hist. Bajo	12	8	0
Paso 6	Cant. var. Removidas Hist. Suf.	3	10	18
	Cant. var. Removidas Hist. Bajo	4	5	6
	Cant. acum. incl. Hist. Suf	1	3	1
	Cant. acum. incl. Hist. Bajo	1	0	0
Paso 7	Cant. rondas Hist. Suf.	50	10	50
	η Hist. Suf.	0.1	0.3	0.001
	γ Hist. Suf.	0.5	0.5	0.1
	Prof. máx. Hist. Suf.	10	10	12
	Cant. rondas Hist. Bajo	50	50	50
	η Hist. Bajo	0.3	0.5	0.3
	γ Hist. Bajo	0	0.1	0.1
	Prof. máx. Hist. Bajo	8	12	6

Tabla 2.2: Valores óptimos de parámetros explorados en la base 1.2.

	Parámetro	Corte 1
Paso 3	m_H	3
	Func. de rareza Hist. Suf.	1
	Cant. árboles Hist. Suf.	50
	Func. de rareza Hist. Bajo	1
	Cant. árboles Hist. Bajo	10
Paso 4	Ventana superior (horas)	48
	Ventana inferior (horas)	∞
	Dist. var. cont. Hist. Suf.	3, $b = 10$
	Cant. agrup. Rareza Hist. Suf.	6
	Cant. agrup. Dist. Hist. Suf.	7
Paso 5	Dist. var. cont. Hist. Bajo	2, $b = 2$
	Cant. agrup. Rareza Hist. Bajo	2
	Cant. agrup. Dist. Hist. Bajo	8
Paso 6	Cant. var. Removidas Hist. Suf.	5
	Cant. var. Removidas Hist. Bajo	5
	Cant. acum. incl. Hist. Suf	2
	Cant. acum. incl. Hist. Bajo	1
Paso 7	Cant. rondas Hist. Suf.	10
	η Hist. Suf.	0.3
	γ Hist. Suf.	0.5
	Prof. máx. Hist. Suf.	10
	Cant. rondas Hist. Bajo	50
	η Hist. Bajo	0.3
	γ Hist. Bajo	1
	Prof. máx. Hist. Bajo	8

Tabla 2.3: Valores óptimos de parámetros explorados en la base 2.1.

	Parámetro	Corte 1	Corte 2
Paso 3	m_H	43	16
	Func. de rareza Hist. Suf.	2	2
	Cant. árboles Hist. Suf.	10	10
	Func. de rareza Hist. Bajo	3	2
	Cant. árboles Hist. Bajo	50	50
Paso 4	Ventana superior (horas)	0	0
	Ventana inferior (horas)	∞	∞
	Dist. var. cont. Hist. Suf.	2, $b = 2$	2, $b = 1$
	Cant. agrup. Rareza Hist. Suf.	0	1
	Cant. agrup. Dist. Hist. Suf.	4	3
Paso 5	Dist. var. cont. Hist. Bajo	2, $b = 2$	2, $b = 1$
	Cant. agrup. Rareza Hist. Bajo	0	0
	Cant. agrup. Dist. Hist. Bajo	10	10
Paso 6	Cant. var. Removidas Hist. Suf.	10	6
	Cant. var. Removidas Hist. Bajo	4	8
	Cant. acum. incl. Hist. Suf	4	0
	Cant. acum. incl. Hist. Bajo	1	2
Paso 7	Cant. rondas Hist. Suf.	50	50
	η Hist. Suf.	0.1	0.001
	γ Hist. Suf.	0	1
	Prof. máx. Hist. Suf.	12	10
	Cant. rondas Hist. Bajo	100	50
	η Hist. Bajo	0.001	0.3
	γ Hist. Bajo	1	0
	Prof. máx. Hist. Bajo	4	3

Tabla 2.4: Valores óptimos de parámetros explorados en la base 21.2.

	Parámetro	Corte 1
Paso 3	m_H	93
	Func. de rareza Hist. Suf.	3
	Cant. árboles Hist. Suf.	50
	Func. de rareza Hist. Bajo	3
	Cant. árboles Hist. Bajo	100
Paso 4	Ventana superior (horas)	0
	Ventana inferior (horas)	168
	Dist. var. cont. Hist. Suf.	3, $b = 10$
	Cant. agrup. Rareza Hist. Suf.	4
	Cant. agrup. Dist. Hist. Suf.	6
Paso 5	Dist. var. cont. Hist. Bajo	3, $b = 10$
	Cant. agrup. Rareza Hist. Bajo	4
	Cant. agrup. Dist. Hist. Bajo	0
Paso 6	Cant. var. Removidas Hist. Suf.	4
	Cant. var. Removidas Hist. Bajo	5
	Cant. acum. incl. Hist. Suf	0
	Cant. acum. incl. Hist. Bajo	1
Paso 7	Cant. rondas Hist. Suf.	50
	η Hist. Suf.	0.1
	γ Hist. Suf.	0
	Prof. máx. Hist. Suf.	12
	Cant. rondas Hist. Bajo	150
	η Hist. Bajo	0.5
	γ Hist. Bajo	0.1
	Prof. máx. Hist. Bajo	10

Tabla 2.5: Valores óptimos de parámetros explorados en la base 3.

Apéndice 3

Tratamiento de variables circulares

La fecha y hora de una transacción es un dato importante del que se puede extraer información que puede ser de utilidad: por ejemplo, para saber la hora o el día de la semana en la que suele realizar compras una persona. Sin embargo, tenemos que tener en cuenta que los datos de fecha se representan en un formato que no permite realizar operaciones numéricas

Si un dato tiene fecha $DM/MM/AAAA$ y hora $HH{:}MI{:}SS$ donde el día de la semana es DS (siendo Lunes el día 1 y Domingo el día 7), entonces los valores de las variables circulares que podemos extraer se calculan de la siguiente manera:

- Hora del día $= HH + \frac{MI}{60} + \frac{SS}{60\times60}$
- Día de la semana $= DS + \frac{HH}{24} + \frac{MI}{24\times60} + \frac{SS}{24\times60\times60} - 1$
- Día del mes $= DM + \frac{HH}{24} + \frac{MI}{24\times60} + \frac{SS}{24\times60\times60} - 1$

donde los -1 en las últimas dos variables son para que sus valores mínimos sean 0. Por ejemplo, tomemos la fecha 15/01/2018 a las 12:30:00 (donde $DS = 1$ por ser un Lunes). Los valores calculados para las tres variables anteriores serían 12.5, 14.52083 y 0.520833 respectivamente.

Distancia circular

Ahora bien, debemos recordar que no podemos trabajar con estas variables así calculadas como si fueran datos continuos, ya que considerar distancias

entre puntos no daría un resultado realista. Por ejemplo: sabemos que entre las 23:00:00 y las 01:00:00 transcurrieron solamente 2 horas de diferencia, pero la resta de ambas horas transformadas como hicimos anteriormente es $|1 - 23|$ que nos daría 22 horas de diferencia. Lo que ocurre aquí es que los datos extraídos de la fecha y la hora no se distribuyen en la recta sino que en un círculo cuyo período[1] depende de la variable extraída en particular (24 para la hora del día, 31 para el día del mes[2] y 7 para el día de la semana). Daremos entonces una fórmula para hallar la distancia entre dos valores x e y en un círculo de período p:

$$d_p(x, y) = \text{mín}\{|x - y|, p + \text{mín}\{x, y\} - \text{máx}\{x, y\}\}$$

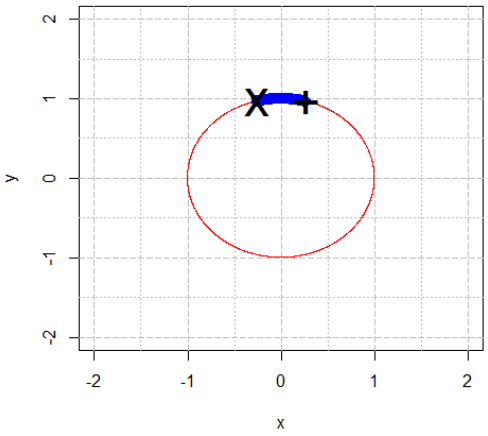


Figura 3.1: Representación de los puntos 1 y 23 (+ y × respectivamente) en un círculo de período 24 (es decir que se parametriza el intervalo [0,24] desde el punto (0,1) en sentido horario). La distancia usual sería la longitud del arco marcado en rojo fino mientras que la distancia $d_p(x, y)$ que nos interesa sería la longitud del arco azul grueso.

[1] Definimos el período como el valor p tal que todos los puntos del círculo se pueden considerar en el intervalo $[0; p]$.

[2] Aquí deberíamos hacer una distinción por el mes que se está considerando, pues podría ser 30, 29 y 28 en otros casos.

Media circular

Para poder usar las variables que extrajimos sin discretizarlas en el cálculo del Score de Outlierness como distancia al vecino más cercano, necesitamos una noción de varianza en la métrica circular, pero esta precisa que definamos una media circular μ_C de los datos distribuidos en el círculo. Es claro que el cálculo usual que hacemos para variables continuas no se extiende directamente al caso circular. Para eso, consideremos el siguiente ejemplo: sea $x = (21, 22, 0, 1)$ un vector de horas del día. Es claro que la hora promedio de x debería ser las 23:00:00, pero hacer el promedio usual daría como resultado $\frac{22+23+1+0}{4} = 11$. Por lo tanto, nos vemos en la necesidad de definir un cálculo diferente para el promedio circular μ_C. El algoritmo que proponemos para calcular μ_C dado un vector $v = (x_1, \ldots, x_n)$ de datos en un círculo de período p es el siguiente:

1. Llevar los datos x_i a puntos y_i del plano mediante la parametrización del círculo $y_i = \left(\sin\left(2\pi\frac{x_i}{p}\right)\ ,\ \cos\left(2\pi\frac{x_i}{p}\right)\right)$
2. Calcular el centroide $\tilde{\mu}_C$ de los puntos y_i como $\tilde{\mu}_C = \left(\frac{1}{n}\sum_{i=1}^{n}(y_i)_1\ ,\ \frac{1}{n}\sum_{i=1}^{n}(y_i)_2\right)$
3. Proyectar $\tilde{\mu}_C = (\tilde{\mu}_1, \tilde{\mu}_2)$ a $\hat{\mu}_C = (\hat{\mu}_1, \hat{\mu}_2)$ mediante una recta que pase por $(0, 0)$ y por $\tilde{\mu}_C$ usando la siguiente fórmula:

$$\hat{\mu}_C = \frac{1}{\sqrt{1 + \left(\frac{\tilde{\mu}_1}{\tilde{\mu}_2}\right)^2}} \times \left(\frac{\tilde{\mu}_1}{|\tilde{\mu}_1|}\ ,\ \frac{\tilde{\mu}_2}{|\tilde{\mu}_1|}\right)$$

4. Mediante la parametrización inversa, recuperar μ_C a partir de $\hat{\mu}_C$. Es decir:

$$\mu_C = \begin{cases} \dfrac{p}{2\pi}\arccos(\hat{\mu}_2) & \text{si } \hat{\mu}_1 \geq 0 \\ p - \dfrac{p}{2\pi}\arccos(\hat{\mu}_2) & \text{si } \hat{\mu}_1 < 0 \end{cases}$$

En la figura 3.2 mostramos gráficamente como se realizan estos cálculos para ilustrar mejor su funcionamiento.

Como puede verse, para el vector $v = (21, 22, 0, 1)$ que mencionábamos anteriormente, el cálculo de la media circular da 23 que es lo que queríamos.

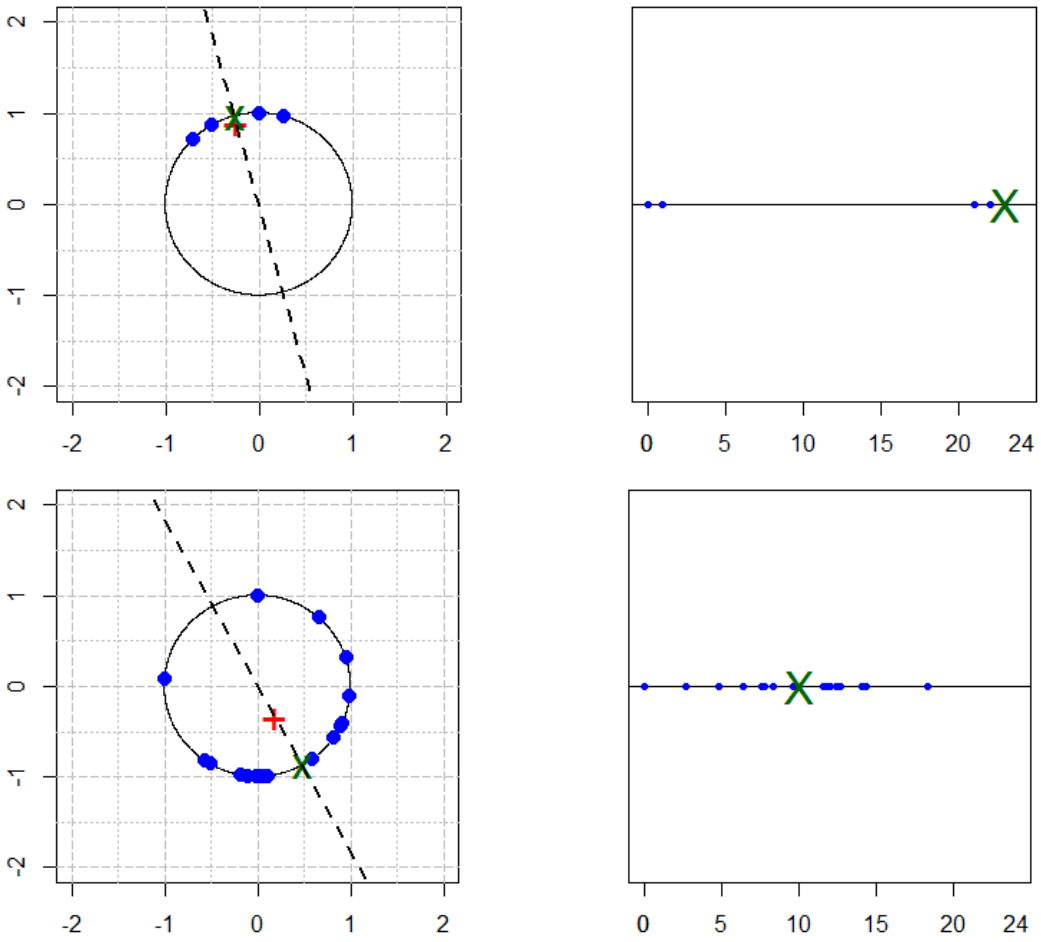


Figura 3.2: Ilustración del cálculo de la media circular para dos vectores v de horas en el círculo de período 24. Arriba se muestra al vector $v = (21, 22, 0, 1)$. Abajo, un vector aleatorio sorteado mediante una distribución normal. A la izquierda, se muestran en azul los puntos y_i transformados de x_i. En rojo, se muestra el centroide $\tilde{\mu}_C$ de los y_i. La línea punteada representa la proyección de $\tilde{\mu}_C$ a $\hat{\mu}_C$ (marcado en verde). A la derecha, se muestran los valores de x_i (azul) y la media circular μ_C calculada (verde).

Varianza circular

Una vez que tenemos definida una distancia y una media, el cálculo de la varianza es sencillo, pues fórmula en el caso continuo es trasladable fácilmente al caso circular: si μ_C es la media circular de un vector $(x_1, \ldots, x_n)$ de datos circulares de período p, entonces definimos su varianza circular σ_C^2 como:

$$\sigma_C^2 = \frac{1}{n-1} \sum_{i=1}^{n} d(x_i, \mu_C, p)^2$$

Habiendo definido una distancia, media y varianza (y por ende, desviación estándar) para las variables circulares, estamos en condiciones de extender las fórmulas vistas en la sección 4.3.1 y así poder usar nuestras variables derivadas de la fecha para el cálculo del Score de Outlierness.

Apéndice 4

Estimación de desempeño de modelos de detección de fraude

Como hemos visto en la sección 2.4.1, nos interesa medir el desempeño de un modelo de detección usando las curvas ROC o PRC, las cuales ilustran el desempeño para cada umbral admisible del modelo. De esta manera, cada institución financiera puede elegir el umbral que alcance el mejor desempeño para las restricciones que decida. En esta sección plantearemos algunas cuestiones particulares sobre los modelos de detección de fraude que deben tenerse en cuenta a la hora de realizar los cálculos para evaluar el desempeño.

Validación cruzada por tarjeta

Siguiendo el espíritu de la sección 2.4.2, tenemos que tener en cuenta que al realizar validación cruzada sobre un conjunto de datos, las particiones o *folds* deben ser sorteadas aleatoriamente *por tarjeta* y no por transacción para evitar problemas de sobreajuste. Como hemos mencionado anteriormente, si se usan instancias de una misma tarjeta para entrenar y evaluar el modelo, el resultado resultaría excesivamente optimista (más aun usando variables acumuladas).

Evaluación realista

Al momento de reportar los resultados de un modelo simplemente contabilizando la cantidad de fraudes que fueron correctamente clasificados, estamos dejando de lado un aspecto fundamental del funcionamiento de estos

modelos en producción: cuando un fraude es detectado correctamente, la institución puede tomar acciones (como bloquear la tarjeta) para prevenir fraudes futuros. Por lo tanto, cuando el modelo detecta el primer fraude cometido en una tarjeta, todos los fraudes posteriores podrían ser evitados independientemente de su score.

Por este motivo es que reportaremos los resultados en el conjunto de testing usando la *evaluación realista*. Esto es: dado un parámetro v (que interpretaremos como el tiempo que tarda la institución en procesar una alerta generada para determinar si es legítima o no) y un umbral de detección u vamos a marcar como detectados:

- Todos los fraudes cuyo score $s(\vec{t})$ cumple que $s(\vec{t}) > u$.
- Todos los fraudes que fueron realizados v unidades de tiempo o más después de un fraude detectado.

De esta manera, podemos realizar una evaluación más ajustada al verdadero funcionamiento del modelo en producción, además de permitir que la institución financiera observe como afecta el tiempo de respuesta v al desempeño del modelo.

Frontera de Pareto

Lo más usual es que no pongamos un solo modelo en producción sino más bien uno por cada corte (como fue mencionado en la sección 2.4.4) que dividen la base en varios conjuntos, cada uno con un modelo y funcionando todos simultáneamente. En un caso como este, la elección de un conjunto de umbrales $(u_1, \ldots, u_n)$ que alcancen un desempeño óptimo globalmente no es equivalente a la elección del umbral óptimo u_i en cada corte, pues en la práctica los modelos interactúan mutuamente. Retomando la idea de la evaluación realista: si un fraude es detectado, puede provocar la prevención de todos los fraudes futuros de la misma tarjeta aunque pertenezcan a cortes distintos. Por lo tanto, para estimar el desempeño global de todos los modelos funcionando en paralelo, deberíamos unir todos los conjuntos de testing de cada corte en uno solo para luego aplicar la evaluación realista.

El aspecto particular de considerar varios modelos al mismo tiempo es que podemos permitir un umbral distinto para cada uno. Por ejemplo: en un corte

donde se cometen fraudes con mayor tendencia a estar entre las primeras transacciones de una tarjeta, puede ser mejor usar un umbral bajo. Al mismo tiempo, en un corte donde el modelo genera un gran número de falsas alarmas, puede convenir usar umbrales altos. Esta elección puede dejarse a criterio de la institución, pero para dar una recomendación, podemos presentar un gráfico como hacemos con la curva PRC para un solo modelo. En este caso, podemos probar presentando en un mismo gráfico todas las combinaciones de umbrales posibles y sus desempeños. Por lo general, eso genera una nube de puntos como la del siguiente ejemplo:

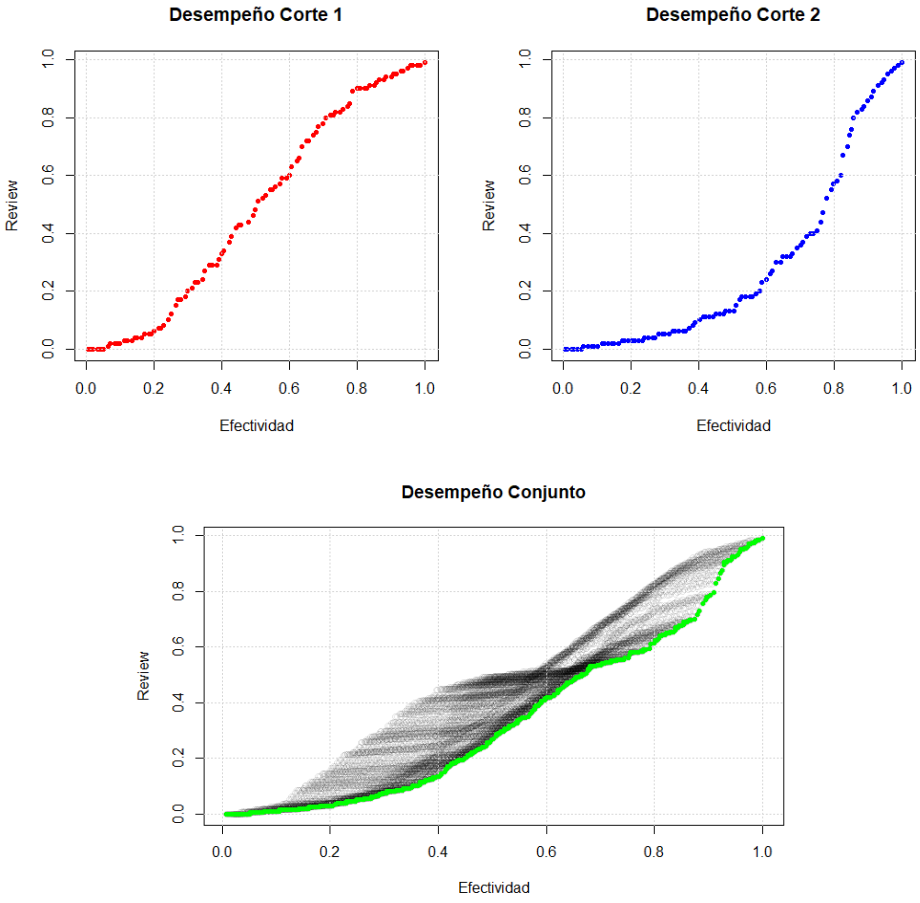


Figura 4.1: Ejemplo artificial de estimación de desempeño conjunto de 2 cortes. Las líneas roja y azul son las curvas PRC para dos cortes de una base. Cada punto gris de la gráfica inferior representa el desempeño de un par de umbrales (u_1, u_2) para los modelos de los cortes 1 y 2 respectivamente funcionando en conjunto. La curva verde en la gráfica inferior es la frontera de Pareto de la nube de puntos.

Como vemos, reportar *todas* las combinaciones posibles de umbrales es innecesario, pues existen muchas que tienen un desempeño claramente peor

que otras. Por eso vamos a buscar la llamada *frontera de Pareto* de la nube de puntos, la cual puede considerarse como la curva óptima para todos los modelos funcionando en simultaneo. Para hallarla, simplemente debemos quedarnos con los puntos de la nube tales que no existe ningún otro punto que lo domine[1]. Algorítmicamente, esto se puede ejecutar de forma sencilla con una recorrida por todos los puntos usando un código que tenga orden $O(n^2)$, que para la cantidad de puntos que solemos manejar suele ser un tiempo sumamente razonable (generalmente son 100^n combinaciones posibles donde n es el número de cortes).

[1]Decimos que un punto $P = (x, y)$ domina a otro $Q = (v, w)$ cuando $x \geq v$ e $y \leq w$ (esto en nuestro caso particular, donde queremos maximizar la medida de desempeño del eje x y minimizar la del eje y).