Universidad de la República
Facultad de Ingeniería

Predicción de tráfico en la red del Plan Ceibal

Memoria de proyecto presentada a la Facultad de Ingeniería de la Universidad de la República por

Marcos Pastorini

en cumplimiento parcial de los requerimientos
para la obtención del título de
Ingeniero en Computación.

Tutor Alberto Castro ..... Universidad de la República

Tribunal

Eduardo Grampin .....
Fiorella Haim .....
Guillermo Moncecchi .....

Montevideo
viernes 2 septiembre, 2022

Predicción de tráfico en la red del Plan Ceibal, Marcos Pastorini.

Esta tesis fue preparada en LaTeX usando la clase iietesis (v1.1+extras).
Contiene un total de 122 páginas.
Compilada el martes 20 septiembre, 2022.
http://inco.fing.edu.uy/

Resumen

En Uruguay, Plan Ceibal tiene desplegados más de 10.000 puntos de acceso WiFi en centros educativos y organizaciones sociales, esto implica una de las infraestructuras de red más grandes del país, con dimensiones comparables a las que tienen los operadores de redes celulares comerciales. Como todo operador de red, Plan Ceibal necesita conocer las características del tráfico de la red, incluida la clasificación y los volúmenes asociados por aplicación. La clasificación de tráfico tradicional es un proceso costoso, que implica herramientas de software específicas y hardware distribuido para la recopilación y el análisis de datos centralizados.

En esto proyecto se propuso evitar el despliegue masivo de una solución de monitoreo y clasificación de tráfico, buscando inferir esos datos a partir de alguna otra fuente también disponible de la propia red. En tal sentido, dado que actualmente los centros que forman parte de la red del Plan Ceibal utilizan un servicio de DNS en la nube (Cisco Umbrella), el objetivo de este proyecto fue diseñar y desarrollar algoritmos de predicción de tráfico para las distintas aplicaciones a partir de los datos de las consultas DNS aplicando técnicas de aprendizaje automático profundo.

La metodología implementada en este proyecto extiende la capacidad de análisis de tráfico del Plan Ceibal, permitiéndole abordar diferentes casos de uso como planificación de la red a mediano y largo plazo, o para predecir eventos de multitudes repentinas (flash crowd), que exigen una reconfiguración rápida de la red para satisfacer la demanda y evitar colapsos.

Durante el desarrollo del proyecto se generó conocimiento sobre el manejo de grandes volúmenes de datos, y se procesaron y perfeccionaron los conjuntos de datos generados por las herramientas del Plan Ceibal. También se investigó el funcionamiento del machine learning automatizado (AutoML) para generar modelos de redes neuronales capaces de predecir la cantidad de consultas DNS y transformarlas en volumen de tráfico, expandiendo la capacidad de las herramientas preexistentes sin agregar costos de infraestructura.

Tabla de contenidos

Resumen I

Índice de tablas V

Índice de figuras VII

1 Introducción **1**
- 1.1 Contexto . . . 1
- 1.2 Objetivos . . . 2
- 1.3 Organización del documento . . . 2

2 Marco teórico **3**
- 2.1 Conceptos de redes de computadoras . . . 3
  - 2.1.1 Domain Name System . . . 3
  - 2.1.2 Cisco Umbrella . . . 5
  - 2.1.3 ntopng . . . 6
- 2.2 Procesamiento de datos . . . 6
  - 2.2.1 Hortonworks Data Platform, Cloudera . . . 7
  - 2.2.2 Extract, Transform and Load . . . 7
  - 2.2.3 Serie temporal . . . 8
- 2.3 Aprendizaje automático . . . 9
  - 2.3.1 Aprendizaje supervisado . . . 9
  - 2.3.2 Aprendizaje profundo . . . 9
  - 2.3.3 *Random Forest* . . . 16
  - 2.3.4 Tensorflow . . . 18
  - 2.3.5 Scikit-Learn . . . 18
- 2.4 Búsqueda de hiperparámetros . . . 18
  - 2.4.1 Métodos de selección de hiperparámetros . . . 19
  - 2.4.2 Métodos de *pruning* de soluciones . . . 20
  - 2.4.3 Optuna . . . 22
- 2.5 Trabajo relacionado . . . 22
  - 2.5.1 Predicción de consultas DNS y tráfico . . . 22
  - 2.5.2 *Machine Learning* automatizado . . . 23

3 Metodología de trabajo **25**
- 3.1 Estructura del proyecto . . . 25
- 3.2 Ambiente de trabajo . . . 25

3.2.1 Acceso al servidor . . . 26
3.2.2 Conda . . . 26
3.3 Actividades realizadas . . . 27

4 Procesamiento y análisis de datos **29**
4.1 Descripción de los datos . . . 29
4.1.1 Conjunto de consultas DNS . . . 29
4.1.2 Conjunto de tráfico . . . 30
4.2 Manejo de datos corruptos . . . 31
4.3 Procesamiento de los datos . . . 31
4.3.1 Conjunto de consultas DNS . . . 32
4.3.2 Conjunto de tráfico . . . 38
4.4 Información adicional agregada . . . 39
4.4.1 Información geoespacial y socioeconómica . . . 39
4.4.2 Datos de variabilidad temporal . . . 46
4.5 Codificación de variables . . . 46
4.6 Agregación y remuestreo de datos . . . 47
4.6.1 Agregación temporal por ruee . . . 48
4.6.2 Agregación temporal por departamento y subsistema . . . 48
4.6.3 Agregación temporal por subsistema . . . 48
4.7 Predicción . . . 49
4.8 Particionado . . . 49
4.9 Normalización . . . 49
4.10 Pipeline . . . 50
4.10.1 Conjunto de tráfico . . . 50
4.10.2 Conjunto de consultas DNS . . . 53
4.11 Análisis de los datos . . . 59
4.11.1 Análisis individual de las columnas . . . 60
4.11.2 Análisis individual del comportamiento . . . 63

5 Sistema de *Machine Learning* automatizado **67**
5.1 Caracterización de un sistema de autoML . . . 67
5.2 Modelo de aprendizaje profundo . . . 70
5.2.1 Arquitectura general planteada . . . 70
5.2.2 Implementación . . . 72
5.2.3 Entrenamiento de redes neuronales sobre Spark . . . 76
5.3 Búsqueda de hiperparámetros . . . 78
5.3.1 Implementación . . . 78

6 Problemas y experimentos **81**
6.1 Descripción de los problemas y soluciones . . . 81
6.2 Modelo *baseline* . . . 83
6.3 Predicción temporal por subsistema . . . 83
6.3.1 Experimento 1 . . . 83
6.3.2 Experimento 2 . . . 85
6.3.3 Conclusiones . . . 88

6.4 Predicción temporal por departamento y subsistema . . . 88
6.4.1 Experimento 1 . . . 88
6.4.2 Experimento 2 . . . 90
6.4.3 Conclusiones . . . 93
6.5 Predicción temporal por ruee . . . 93
6.5.1 Experimento 1 . . . 93
6.5.2 Experimento 2 . . . 95
6.5.3 Conclusiones . . . 97

7 Conclusiones y trabajo futuro **99**
7.1 Sobre el procesamiento de *Big Data* . . . 99
7.2 Sobre el sistema de autoML . . . 99
7.3 Sobre la predicción de cantidad de consultas DNS . . . 100
7.4 Sobre la transformación de cantidad de consultas DNS en volumen de tráfico . . . 100
7.5 Trabajo futuro . . . 100

Referencias **103**

Índice de tablas

3.3.1 Resumen de ejecución de actividades del proyecto y sus resultados. 28

4.1.1 Ejemplo de logs de una consulta DNS. 30
4.1.2 Ejemplo de logs de tráfico de las consultas DNS realizadas en 5 minutos. 31
4.2.1 Ejemplo de datos con filas inconsistentes. 31
4.2.2 Cantidad de datos corruptos por día. 31
4.3.1 Ejemplo de procesamiento de la columna **timestamp**. 32
4.3.2 Ejemplo de procesamiento de las columnas **mostGranularIdentity**, **identities** y **externalIP**. 33
4.3.3 Ejemplo de procesamiento de la columna **queryType** 33
4.3.4 Lista de dominios de interés de Ceibal, URL y valor asignado al parsear. 34
4.3.5 Ejemplo de procesamiento de la columna **domain**. 36
4.3.6 Ejemplo de procesamiento de la columna **categories** 38
4.3.7 Ejemplo de imputación de datos en las columnas **category_1**, **category_2** y **category_3** 38
4.3.8 Ejemplo de procesamiento de la columna **timestamp**. 38
4.3.9 Ejemplo de procesamiento de la columna **application** 39
4.4.1 Identificadores inconsistentes detectados durante la combinación de las tablas. 43
4.4.2 Cantidad de valores faltantes por columna de la tabla combinada. 43
4.4.3 Cantidad de valores faltantes por columna para el conjunto de ruees utilizados. 44
4.4.4 Ejemplo de adición de información geoespacial y sociocultural. 45
4.5.1 Ejemplo de *encoding* de la columna **departamento** 46
4.5.2 *Encoding* de las columnas que representan datos temporales 46
4.6.1 Ejemplo de re-muestreo de la columna **minute** a 5 minutos 47
4.6.2 Ejemplo de índice resultante del proceso de agregación de datos para el período temporal de 15 minutos, en la agregación temporal por ruee. 48
4.6.3 Ejemplo de índice resultante del proceso de agregación de datos para el período temporal de 15 minutos, en la agregación temporal por departamento y subsistema. 48
4.6.4 Ejemplo de índice resultante del proceso de agregación de datos para el período temporal de 15 minutos, en la agregación temporal por subsistema. 49
4.8.1 Ejemplo de particionado usando las columnas **date** y **hour** conjuntos *train*, *val* y *test* respectivamente. 50

5.2.1 Comparación de las principales características de las bibliotecas para entrenamiento distribuido. 77

6.3.1 Comparación de experimentos para la predicción de consultas DNS sobre el conjunto agregado por subsistema. 87
6.3.2 Comparación de experimentos para la transformación de consultas DNS en tráfico sobre el conjunto agregado subsistema. 87

6.4.1 Comparación de experimentos para la predicción de consultas DNS sobre el conjunto agregado por departamento y subsistema. 92
6.4.2 Comparación de experimentos para la transformación de consultas DNS en tráfico sobre el conjunto agregado por departamento y subsistema. 92
6.5.1 Comparación de experimentos para la predicción de consultas DNS sobre el conjunto agregado por ruee. 97
6.5.2 Comparación de experimentos para la transformación de consultas DNS en tráfico sobre el conjunto agregado ruee. 97

Índice de figuras

2.1.1 Ejemplo de los componentes del hostname `www.fing.edu.uy`. 3
2.1.2 Esquema de la organización jerárquica de los servidores DNS. (Figura extraída de [4]) 4
2.1.3 Ejemplo de consulta DNS del dominio `gaia.cs.umass.edu` del dispositivo identificado como `cis.poly.edu`. (Figura extraída de [4]) 5
2.1.4 Esquema de funcionamiento de *Cisco Umbrella*. (Figura extraída de [3]) 6
2.2.1 Arquitectura de la plataforma HDP. (Figura extraída de [14]) 8
2.3.1 Representación en forma de grafo de la función $f*$ que define una red neuronal. 10
2.3.2 Comparación de la curva de la función *Huber* (izquierda) y la función *Reversed Huber* (derecha). (Figura extraída de [22]) 11
2.3.3 Esquema del algoritmo de descenso por gradiente. (Figura extraída de [23]) 12
2.3.4 A la izquierda se muestra un esquema del algoritmo de descenso por gradiente con *learning rate* muy alto y a la derecha el caso opuesto. (Figura extraída de [23]) 12
2.3.5 Ejemplo de análisis de función de costo durante la variación de *learning rate*. (Figura extraída de [26]) 13
2.3.6 Ejemplo de aplicación de la función *conv* usada por una capa convolucional. 14
2.3.7 A la izquierda diagrama de una red neuronal con dos capas densas ocultas, a la derecha la misma red durante el entrenamiento con *dropout* aplicado en las entradas de las capas. 15
2.3.8 Esquema de paralelismo *all-reduce*. (Figura extraída de [34]) 16
2.3.9 Esquema de *all-reduce* durante el entrenamiento. (Figura extraída de [34]) 17
2.3.10 Esquema ejemplo de la estructura de un *Decision Tree*. 17
2.4.1 Comparación de los métodos *grid search* y *random search* en la búsqueda de los valores posibles de dos hiperparámetros. En el eje x, se muestra un hiperparámetro importante cuya evaluación es mayor (en rojo) y en el eje y otro hiperparámetro de menor importancia. (Figura extraída de [41]) 20
2.4.2 Comparación de los métodos *random search* y TPE en la optimización de una función objetivo sobre dos conjuntos. Los puntos grises representan el error más bajo obtenido de varias ejecuciones de *random search* y los puntos verdes representan el error más bajo obtenido por una ejecución de TPE. (Figura extraída de [44]) 21
2.4.3 Comparación de los algoritmos *Hyperband* y SHA en la optimización de un modelo de redes neuronales sobre un conjunto de datos de prueba. Cada valor de s representa una configuración particular de n para SHA. (Figura extraída de [47]) 22

3.3.1 Diagrama de *Gantt* detallando el tiempo de ejecución de las principales actividades del proyecto. 28

4.4.1 *Boxplots* de latitud y longitud antes de procesarlas. 42
4.4.2 Boxplots de latitud y longitud luego de procesarlas. 42
4.4.3 Interfaz de la aplicación creada para la imputación de datos de locales. 43

4.4.4 Cantidad de datos imputados según la distancia de imputación. 45
4.10.1 Etapa 1 del pipeline del conjunto de tráfico 52
4.10.2 Etapa 2 del pipeline del conjunto de tráfico (izquierda) y etapa 2 de re-muestreo (derecha) 53
4.10.3 Etapa 1 del pipeline 55
4.10.4 Etapa 2 del pipeline 57
4.10.5 Etapa 3 del pipeline del conjunto de consultas DNS (izquierda) y etapa 3 de re-muestreo (derecha) 58
4.10.6 Etapa 4 del pipeline del conjunto de datos 59
4.11.1 Distribución de las consultas DNS por tipo. 60
4.11.2 Distribución de las consultas DNS por código de respuesta. 61
4.11.3 Comparación de la distribución de dominios parseados en el conjunto de datos completo (izquierda) y el conjunto de datos reducido (derecha). 61
4.11.4 Distribución de los dominios por tráfico consumido. 62
4.11.5 Distribución de las consultas DNS por categoría. 63
4.11.6 Cambios en la distribución de las categorías en la columna **category_1** luego de la imputación. . 63
4.11.7 Cambios en la distribución de las categorías en la columna **category_2** luego de la imputación. . 64
4.11.8 Comparación del promedio (normalizado) de consultas DNS en el período marzo-setiembre (*global mean*) y el período setiembre-diciembre (*subset mean*). A la izquierda comparación por hora, a la derecha comparación por día. 64
4.11.9 Comparación de la cantidad de consultas DNS realizadas en el período reducido (*total traffic*) y el tráfico generado (*total queries*). 65
4.11.10 Boxplots de datos normalizados de tráfico total (izquierda) y datos normalizados de cantidad de consultas DNS (derecha). 65

5.1.1 *Pipeline* de un proyecto de *machine learning*. 67
5.1.2 Clasificación de sistemas de autoML según su nivel de automatización y ejemplos de sistemas en cada nivel. (Figura extraída de [73]) 69
5.1.3 Diagrama de funcionamiento del sistema de autoML implementado 69
5.2.1 Comparación de representaciones de variables categóricas, cada columna representa un valor del conjunto de colores. 71
5.2.2 Diagrama de arquitectura base planteada. 72
5.2.3 Transformaciones aplicadas a los datos antes del entrenamiento. 73
5.3.1 Comparación de las distintas combinaciones de métodos de selección de hiperparámetros y de *pruning*. A la izquierda se usa *random search* y a la derecha TPE, en cada gráfica se compara la versión sin *pruning* (nop) contra la versión con *Median Stopping Rule* (median) y la versión con *Hyperband* (hyperband). (Figura extraída de [78]) 79
5.3.2 Diagrama de despliegue de sistema de autoML en nodos de ClusterUY. 79

6.3.1 Comparación de datos observados en el conjunto de *test* (en azul) con el promedio de los observados (en rojo), las predicciones del *baseline* (en naranja) y del experimento 1 (en verde) para la predicción de consultas DNS sobre el conjunto agregado por subsistema. 85
6.3.2 Comparación de datos observados en el conjunto de *test* (en azul) con el promedio de los observados (en rojo), las predicciones del *baseline* (en naranja) y del experimento 1 (en verde) para la transformación de consultas DNS en tráfico sobre el conjunto agregado por subsistema. Se resaltan casos particulares en rojo. 85

6.3.3 Comparación de datos observados en el conjunto de *test* (en azul) con el promedio de los observados (en rojo), las predicciones del *baseline* (en naranja) y del experimento 2 (en verde) para la transformación de consultas DNS en tráfico sobre el conjunto agregado por subsistema. Se resaltan casos particulares en rojo. 86
6.4.1 Comparación de datos observados en el conjunto de *test* (en azul) con el promedio de los observados (en rojo), las predicciones del *baseline* (en naranja) y del experimento 1 (en verde) para la predicción de consultas DNS sobre el conjunto agregado por departamento y subsistema. 90
6.4.2 Comparación de datos observados en el conjunto de *test* (en azul) con el promedio de los observados (en rojo), las predicciones del *baseline* (en naranja) y del experimento 1 (en verde) para la transformación de consultas DNS en tráfico sobre el conjunto agregado por departamento y subsistema. Se resaltan casos particulares en rojo. 90
6.4.3 Comparación de datos observados en el conjunto de *test* (en azul) con el promedio de los observados (en rojo), las predicciones del *baseline* (en naranja) y del experimento 2 (en verde) para la transformación de consultas DNS en tráfico sobre el conjunto agregado por departamento y subsistema. 91
6.5.1 Comparación de datos observados en el conjunto de *test* (en azul) con el promedio de los observados (en rojo), las predicciones del *baseline* (en naranja) y del experimento 1 (en verde) para la predicción de consultas DNS sobre el conjunto de agregado por ruee. 95
6.5.2 Comparación de datos observados en el conjunto de *test* (en azul) con el promedio de los observados (en rojo), las predicciones del *baseline* (en naranja) y del experimento 1 (en verde) para la transformación de consultas DNS en tráfico sobre el conjunto agregado por ruee. 95
6.5.3 Comparación de datos observados en el conjunto de *test* (en azul) con el promedio de los observados (en rojo), las predicciones del *baseline* (en naranja) y del experimento 2 (en verde) para la transformación de consultas DNS en tráfico sobre el conjunto agregado por ruee. 96

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 1

Introducción

En este capítulo se introduce la temática abordada a lo largo de este proyecto, comentando el contexto sobre el que se desarrolla, los objetivos propuestos, resultados esperados y la organización del documento.

1.1. Contexto

En Uruguay, el Plan Ceibal[1] ha desplegado infraestructuras de información y comunicación (TICs) en la totalidad de las escuelas y liceos públicos del país además de muchos espacios públicos. El desarrollo más importante es la conectividad inalámbrica, que permite a un usuario conectarse a diferentes recursos, incluyendo el acceso a Internet en uno de los más de 10000 puntos de acceso WiFi instalados. En los últimos años se agregó a esta infraestructura el soporte al Plan Ibirapitá, que busca superar la barrera digital para personas de la tercera edad. Esto ha llevado a que el Plan Ceibal cuente hoy en día con una infraestructura de red de las más grandes del país, con dimensiones comparables a las que tienen los operadores de redes celulares comerciales.[2] Por lo tanto, el diseño, despliegue, operación y mantenimiento de esta red, es sin lugar a dudas una de las tareas que mayor trabajo y recursos le demanda al Centro Ceibal dentro de sus responsabilidades.

Para una correcta operación y mantenimiento de la red, se han agregado funcionalidades de clasificación de tráfico. Actualmente, la gran mayoría de las redes WiFi que gestiona Ceibal cuentan con una solución propietaria de clasificación de tráfico (*Cisco Umbrella*[3]), esta herramienta genera datos a partir de las consultas *Domain Name System* (DNS)[4] realizadas en cada local. Por ser propietaria no puede ser extendida a dispositivos que no sean de *Cisco* ni puede ser personalizada. Por estos motivos, se ha incorporado en algunos centros educativos una solución abierta basada en *ntopng*[5] que genera datos de tráfico en términos de bytes transferidos y que permite la personalización pero cuyo despliegue es costoso. Esto último se debe a que implica desplegar el sistema de clasificación en cada centro además de la instalación de un mecanismo para centralizar los datos recolectados, su correspondiente almacenamiento y análisis y el mantenimiento de la herramienta en si.

Parte de este proyecto de grado se enmarca y en el proyecto FSDA_1_2018_1_154853 “Predicción de tráfico en redes del Plan Ceibal”financiado por el Fondo sectorial de investigación a partir de datos de la ANII cuyo objetivo es el de extender la capacidad de análisis de tráfico a todos los centros mediante el uso de modelos de aprendizaje automático, utilizar los registros de redirección del DNS para hacer análisis predictivo del tráfico de red, aplicando técnicas de aprendizaje automático profundo, por ejemplo para prever necesidades de infraestructura, y/o para prevenir ataques o aumento sorpresivo del tráfico (eventos del tipo *flash crowd*), entre otras posibilidades.

1.2. Objetivos

En este proyecto se propone diseñar, implementar y validar un modelo de redes neuronales profundas que sea capaz de transformar datos de consultas DNS en volumen de datos y también predecir el tráfico de la red WiFi del Plan Ceibal a mediano plazo a partir del uso que los usuarios hacen de la misma (con los datos de consultas DNS generados por la herramienta *Cisco Umbrella*). Para saber cómo los usuarios usan la red del Plan Ceibal, se plantea el estudio de las consultas DNS realizadas durante un año lectivo. Para alcanzar este objetivo general, se definen los siguientes objetivos específicos y resultados esperados de cada uno:

- **OE1**: Procesamiento y análisis de los datos de consultas DNS y tráfico generados por las distintas herramientas de la red Ceibal, estudio y uso de plataformas de trabajo para grandes volúmenes de datos (*Big data*). Los resultados esperados de este objetivo son la generación de un conjunto de datos útil para el análisis de comportamiento y modelado de la red, y el conocimiento sobre el uso correcto de plataformas de *big data*.
- **OE2**: Estudio del estado del arte del *machine learning* automatizado (autoML) e implementación de un sistema de autoML para la generación de modelos, estudio de redes neuronales convolucionales y recurrentes. Los resultados esperados de este objetivo son el diseño e implementación de un modelo de aprendizaje para la resolución de los problemas de este proyecto de grado, el conocimiento sobre el estado del arte de autoML y el diseño e implementación de un sistema de autoML.
- **OE3**: Predicción de datos de cantidad de consultas DNS futuras a partir de datos actuales. El resultado esperado de este objetivo es la obtención de al menos un modelo capaz de predecir datos de consultas DNS.
- **OE4**: Transformación de datos de cantidad de consultas DNS en datos de tráfico (en términos de bytes transferidos). El resultado esperado de este objetivo es la obtención de al menos un modelo capaz de transformar datos de consultas DNS en datos de tráfico.

1.3. Organización del documento

El contenido restante de este documento se organiza de la siguiente manera:

En el capítulo 2 se introducen conceptos de redes de computadoras, procesamiento de datos, aprendizaje automático y trabajo previo.

Luego, en el capítulo 3 se detalla la estructura del repositorio de este proyecto, metodologías de trabajo usadas y las actividades realizadas.

En el capítulo 4 se comentan detalles de los datos usados y se detalla el trabajo realizado para procesarlos y su análisis posterior.

En el capítulo 5 se caracteriza un sistema de *machine learning* automatizado, su implementación en este proyecto y sus componentes.

En el capítulo 6 se describen los experimentos realizados y los modelos obtenidos.

Finalmente, en el capítulo 7 se discuten las conclusiones obtenidas y el posible trabajo futuro.

Capítulo 2

Marco teórico

En este capítulo se presenta el marco teórico sobre el que se trabajó, se desarrollan conceptos generales de redes de computadoras, se abordan temas relacionados con el manejo de grandes volúmenes de datos (*big data*) y por último temas relacionados con el aprendizaje automático. También se detalla el trabajo relacionado obtenido de la revisión bibliográfica.

2.1. Conceptos de redes de computadoras

Como los datos utilizados en este proyecto pertenecen al dominio de las redes de computadoras, es necesario detallar los siguientes conceptos.

2.1.1. Domain Name System

Internet es una red global de dispositivos interconectados, cuando un dispositivo busca conectarse con otro, el primer paso a ejecutar es la identificación del dispositivo de destino. Un dispositivo se identifica con un *hostname* (también conocidos como nombres de dominio) que es una cadena de caracteres alfanuméricos de largo variable (por ejemplo `www.google.com`), como estos identificadores pueden ser difíciles de procesar por un *router* (dispositivo utilizado como intermediario para acceder a internet) y no contienen información sobre la ubicación del dispositivo, se usa otro identificador, la dirección IP (*Internet Protocol*).[4]

Un nombre de dominio se divide en varios componentes listados de forma jerárquica (siendo el más a la derecha el de más arriba en la jerarquía) mostrados en la figura 2.1.1.

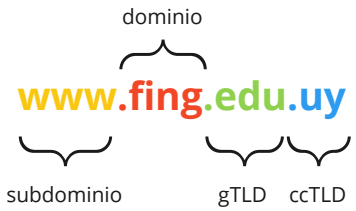


Figura 2.1.1: Ejemplo de los componentes del hostname `www.fing.edu.uy`.

Una dirección IP consiste de una cadena de largo fijo de bloques de números, por ejemplo `121.7.106.83`, donde cada bloque (en este caso separado por puntos) representa información de la ubicación del dispositivo identificado.

La información de cada bloque funciona de manera jerárquica pues, si se lee la dirección de izquierda a derecha, se obtienen datos cada vez más específicos de la ubicación del dispositivo identificado.

Para traducir *hostnames* en direcciones IP, se usa el *Domain Name System* (DNS), este sistema está compuesto por una base de datos distribuida y un protocolo que permite consultas esa base de datos.[4]

Base de datos distribuida y jerárquica

La base de datos distribuida se compone por varios dispositivos llamados servidores DNS categorizados en tres tipos: *root*, *top-level domain* (TLD) y *authoritative* y organizados en forma jerárquica como se muestra en la figura 2.1.2.

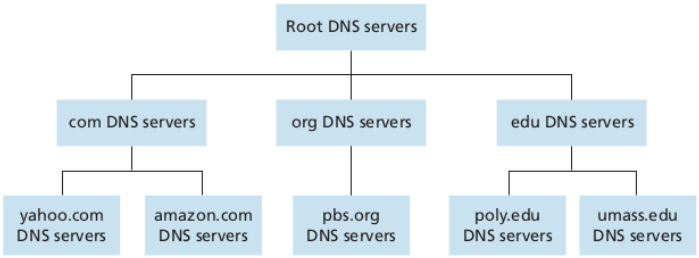


Figura 2.1.2: Esquema de la organización jerárquica de los servidores DNS. (Figura extraída de [4])

Un servidor de tipo *root* contiene las direcciones IP de todos los servidores TLD y cada servidor TLD es responsable de las IPs bajo uno de los dominios de nivel superior que representan países (ccTLD, como `co` o `uy`) o los de uso general (gTLD, como `com`). Por último, un servidor *authotitative* es el encargado de almacenar todos los dominios que una organización haga públicos (por ejemplo `facebook.com`).

También existe otro tipo de servidor DNS (que no pertenece estrictamente a la jerarquía) llamado servidor DNS local (también llamado *default name server*), estos servidores son usados por los proveedores de servicio de Internet (*Internet Service Provider*, ISP) y, cuando un dispositivo se conecta a un ISP, este le brinda las direcciones IP de uno o más de sus servidores locales.[4]

Dentro de la base, los registros DNS se clasifican según un tipo y los principales son:

A y **AAAA** (*Address Mapping Record*): Contiene el mapeo de un *hostname* a una dirección IP.

PTR (*Reverse-lookup Pointer Record*): Contiene el mapeo de una dirección IP a un *hostname*, esto permite realizar una consulta reversa y obtener un *hostname* a partir de una dirección IP.

NS (*Name Server Record*): Contiene el mapeo de un dominio y el hostname del servidor *authoritative* que sabe como obtener las direcciones IP de los dispositivos del dominio.

CNAME (*Canonical Name Record*): Contiene el mapeo de un dominio y sus alias, esto permite que un mismo dispositivo pueda ser identificado con varios *hostnames*.

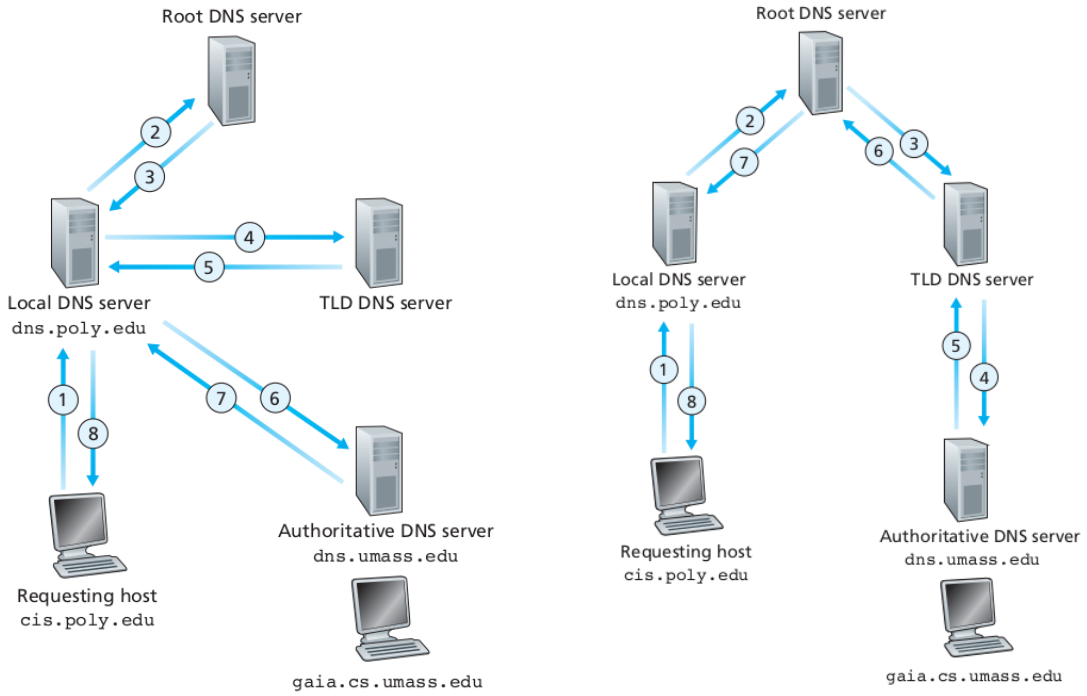
MX (*Mail exchanger Record*): Contiene el mapeo de un dominio de correo electrónico y sus alias, esto permite que un servidor de correo electrónico pueda tener identificador simplificado.

Consulta DNS

Para lograr la traducción de un *hostname* en una dirección IP un dispositivo realiza una consulta DNS a un servidor local, luego este se encarga de continuar con la consulta de forma iterativa o recursiva. En el caso de una

consulta iterativa (figura 2.1.3a), el servidor local consulta primero a un servidor *root* (2) que devuelve una lista con las direcciones IP de los servidores TLD responsables del sufijo más general (3), luego el servidor local realiza la consulta a un servidor TLD de la lista (4) que devuelve la dirección IP del servidor *authoritative* (5), después de esto el servidor local envía la consulta al servidor con la dirección recibida (6) y este le devuelve la dirección del dominio buscado (7), por último, el servidor local envía la dirección obtenida al dispositivo consultante (8).

En el caso de una consulta recursiva (figura 2.1.3b), cada uno de los servidores consultados se encarga de realizar la consulta al servidor siguiente en la jerarquía (2, 3, 4) y devolver el resultado de esa consulta (5, 6, 7, 8).[4]



(a) Ejemplo de consulta DNS iterativa

(b) Ejemplo de consulta DNS recursiva

Figura 2.1.3: Ejemplo de consulta DNS del dominio `gaia.cs.umass.edu` del dispositivo identificado como `cis.poly.edu`. (Figura extraída de [4])

2.1.2. Cisco Umbrella

Cisco Umbrella[3] es una solución basada en la nube cuya función es la de proteger de amenazas a quienes acceden a internet a través de un punto de acceso provisto por una empresa que haya contratado el servicio. *Umbrella* funciona analizando las consultas DNS realizadas y determinando si la web a la que se quiere acceder es segura o no, además registra la actividad, la categoriza y la marca como permitida o bloqueada, permitiendo a quienes administran la red tomar acciones en vivo o analizar los datos para decisiones posteriores.

Este servicio se encuentra en uso en todas las redes administradas por Ceibal y es el que se encarga de generar un parte de los datos usados para este proyecto.

El funcionamiento de *Umbrella* se esquematiza en la figura 2.1.4 y se describe a continuación:

1. Una persona intenta acceder a una web o usar un servicio basado en internet, generando una consulta DNS que se envía a los servidores de *Umbrella*.

2. *Umbrella* analiza la consulta para determinar si el dominio es malicioso o seguro, si es seguro responde con la dirección IP del dominio.
3. El dispositivo continua con la conexión de forma trasparente para la persona.
4. Si *Umbrella* determina que el dominio no es seguro, responde con la dirección IP de una web donde muestra un mensaje de error, este mensaje previene a la persona de acceder al sitio malicioso.

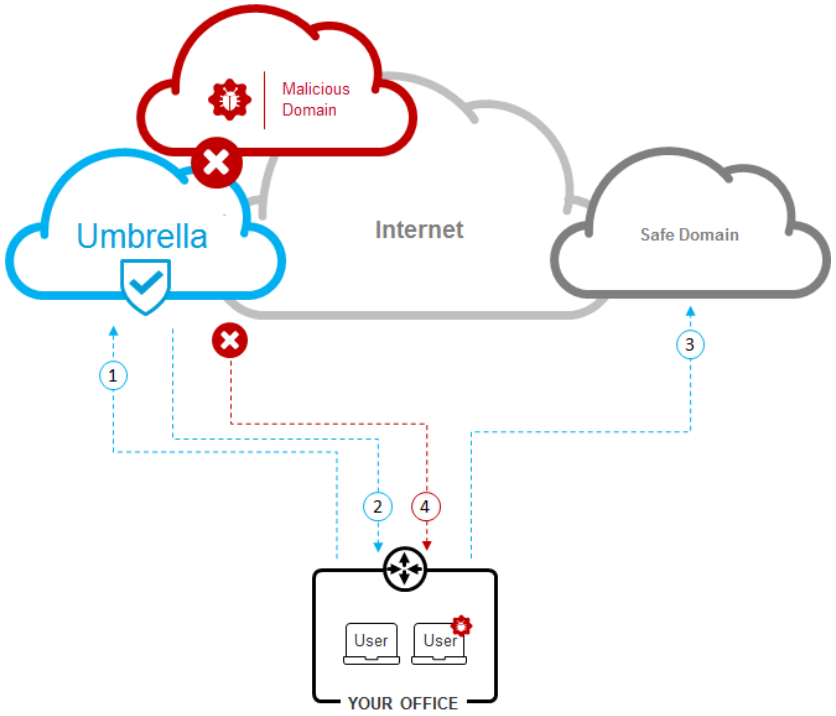


Figura 2.1.4: Esquema de funcionamiento de *Cisco Umbrella*. (Figura extraída de [3])

2.1.3. ntopng

ntopng[5] es una solución basada en la web para monitorizar el volumen de tráfico, es decir, la cantidad de bytes transmitidos entre dos dispositivos a través de intenet. Este programa analiza todos los paquetes implicados en la comunicación entre dos dispositivos capturando información de datos enviados (*uplink*) y recibidos (*downlink*) para luego clasificarlos en una de las distintas categorías definidas (llamadas aplicaciones) dependiendo de los dominios consultados.

Este servicio se encuentra en uso en algunos centros educativos de las redes administradas por Ceibal y es la que se encarga de generar una parte de los datos en este proyecto.

2.2. Procesamiento de datos

Un volumen de datos masivo como el de los utilizados en este proyecto requiere el uso de plataformas especializadas para su manejo y transformación, además requiere procesos estandarizados para poder abstraer la asimilación y transformación de datos.

2.2.1. Hortonworks Data Platform, Cloudera

Hortonworks Data Platform (HDP)[6] es una plataforma que centraliza y despliega un conjunto de servicios que permiten el manejo correcto de grandes cantidades de datos (*big data*), esta plataforma brinda servicios de *cluster-computing*, los principales componentes son:

Apache *Hadoop Distributed File System* (HDFS)[7] es un sistema de archivos distribuido que se diferencia de otros sistemas distribuidos en su diseño centrado en la tolerancia a fallos y la ejecución en hardware de bajo coste. HDFS está pensado para ser usado por aplicaciones con grandes conjuntos de datos que realizan trabajos en *batches*.

Apache *Yet Another Resource Negotiator* (YARN)[8] es un sistema operativo y gestor de recursos distribuidos, la idea detrás de su diseño es la de separar la gestión de recursos de la planificación y monitorización de las aplicaciones o tareas a ejecutar en el sistema.

Apache Hive[9] es un software de *data warehouse* que facilita la lectura, escritura y manejo de grandes conjuntos de datos que residan en sistemas de archivos distribuidos, permite acceder a los datos usando sintaxis SQL. Una particularidad de este sistema de bases de datos es que el formato utilizado para almacenar los datos es independiente de Hive pudiendo ser por ejemplo CSV, TSV, Parquet u ORC, por defecto se usa el último.

Apache Spark[10] es un sistema de *cluster-computing* de propósito general. Provee APIs de alto nivel para Java, Scala, Python y R, su principal funcionalidad es la abstracción DataFrame que permite leer y manipular datos sobre HDFS de forma transparente.

Apache Zeppelin[11] es un *notebook* web multi propósito que soporta varios lenguajes de programación y permite visualizar datos y ejecutar código de forma interactiva.

Apache Arrow[12] es una plataforma desarrollada para permitir a sistemas de *big data* procesar y mover datos rápidamente. Especifica un formato columnar estandarizado que no depende de ningún lenguaje de programación (Parquet) y es especialmente útil para el pasaje de datos entre la maquina virtual de Java (JVM) y Python.

Ambari[13] es una herramienta diseñada para manejar y monitorizar los distintos componentes dentro de un *cluster* HDP, brinda una interfaz gráfica web desde la que se puede configurar recursos y monitorizar aplicaciones en ejecución.

La integración de estos componentes es transparente para el usuario y su arquitectura se muestra en la figura 2.2.1.

2.2.2. Extract, Transform and Load

El proceso *Extract*, *Transform and Load* (ETL) divide el procesamiento de datos en tres fases: extracción, transformación y carga de la información, estas fases permiten obtener y mover datos de distintas fuentes y normalizar o transformar la información obtenida delimitando y modularizando el proceso, logrando facilitar la integración de nuevas etapas o cambios en las ya existentes.

Extracción

Es la primera de las etapas del proceso y se basa en obtener los datos a partir de distintas fuentes. La extracción de información requiere un enfoque distinto dependiendo de la plataforma (como las bases de datos), el sistema operativo y el método de comunicación (REST, SQL, etc) usados. Se pueden identificar dos sub-etapas dentro del proceso de extracción, la primera es la extracción inicial y se realiza una única vez obteniendo (generalmente)

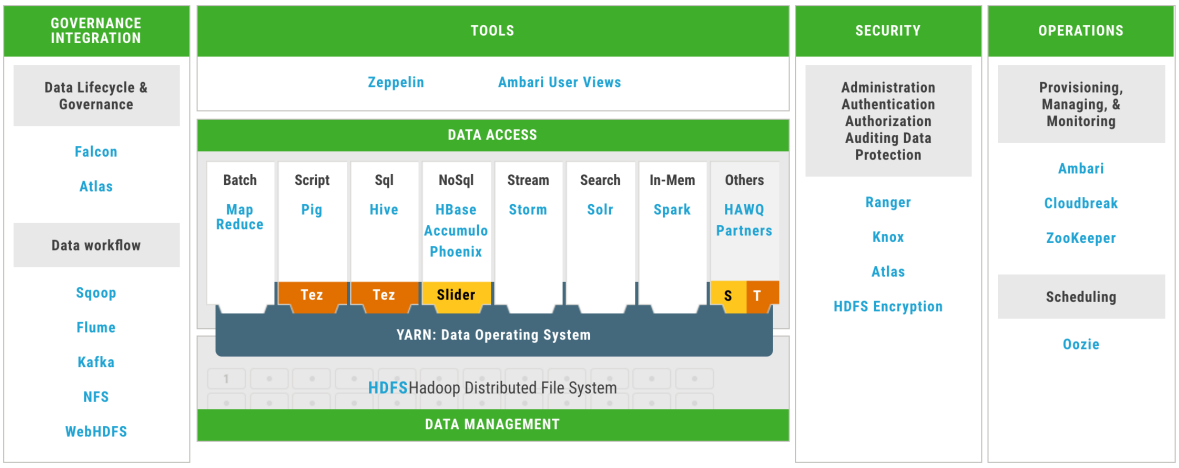


Figura 2.2.1: Arquitectura de la plataforma HDP. (Figura extraída de [14])

una gran cantidad de información, la segunda etapa es la extracción realizada para actualizar la información ya registrada, puede ser realizada cada cierto intervalo de tiempo o ser invocada según cierto evento definido.[15]

Transformación

Es la segunda etapa del proceso y se basa en transformar la información obtenida en la etapa anterior, procesando, normalizando y filtrando los datos para asegurar su correctitud, consistencia y completitud. Los procesos de normalizado se encargan de convertir y estandarizar tipos de datos (por ejemplo fechas).[15]

Carga

Es la última etapa del proceso y consiste en almacenar los datos transformados en algún sistema como puede ser una base de datos o un archivo plano y algún formato de archivo.[15]

2.2.3. Serie temporal

Una serie temporal es una secuencia de observaciones típicamente medidas en intervalos uniformes de tiempo. Este tipo de datos existe en muchas situaciones, incluyendo el valor de las acciones en el mercado, cadenas industriales, controles del estado de salud en pacientes, monitoreo de red e indicadores económicos.[16]

Promedio móvil exponencial ponderado

El promedio móvil exponencial ponderado (*Exponentially Weighted Moving Average*, EWMA) es un indicador estadístico usado para describir series temporales, este indicador está diseñado de forma que las observaciones más recientes tengan más peso que las anteriores, esto se da porque los pesos asignados a cada observación decrecen exponencialmente para las observaciones más antiguas. Es decir que en el cálculo del indicador se promedian exponencialmente los puntos de una ventana temporal (t) usada según su distancia con el punto actual.[17]

La formula empleada para el cálculo de EWMA de la serie temporal $\{X_i\}$ es:

$$\begin{cases} \text{EWMA}(x_n) & = \dfrac{\sum_{i=0}^{t} x_{n-i} w_i}{\sum_{i=0}^{t} w_i} \\ \alpha & = \dfrac{2}{t+1} \\ w_i & = (1-\alpha)^i \end{cases}$$

Donde w_i es el peso asignado a cada observación dependiendo de su distancia al punto actual y α el factor de suavisado.

2.3. Aprendizaje automático

En general, un algoritmo de aprendizaje automático (*machine learning*, ML) puede definirse como "un programa de computación que aprende de la experiencia E con respecto a una tarea T y alguna medida de rendimiento P, si es que el rendimiento en T, medido por P, mejora con la experiencia E"[18].

Estos algoritmos se pueden clasificar, dependiendo de su funcionamiento, en dos categorías: algoritmos de aprendizaje supervisado y algoritmos de aprendizaje no supervisado.

2.3.1. Aprendizaje supervisado

Los algoritmos de aprendizaje supervisado son aquellos cuya tarea es la de inferir una función $f : \mathbb{R}^n \to \mathbb{R}^m$ que mapea una entrada a una salida. El conjunto X de datos de entrada está compuesto por puntos con posiblemente varias variables $X_1, \ldots, X_n$ llamadas *features* mientras que el conjunto Y de datos de salida (o predicciones) está compuesto por puntos denominados *labels* (etiquetas), a una instancia de un algoritmo de aprendizaje supervisado (llamada *modelo*) se la alimenta con pares de datos de entrada y salida (a cada par se le denomina *ejemplo*) para luego adaptar sus parámetros internos (proceso llamado *entrenamiento*) de forma de poder generar una predicción frente a una entrada nunca antes vista (*inferencia*).

Dependiendo de las etiquetas usadas, la tarea que realiza un modelo puede definirse como regresión o clasificación. En el caso de los modelos de clasificación, los valores de salida son categóricos, la función a aproximar es $f : \mathbb{R}^n \to 1, \ldots, k$ y los valores de las etiquetas representan un conjunto de clases, la tarea de un modelo es determinar a que clase pertenece un dato de entrada (por ejemplo, determinar si un email es spam o no). En el caso de los modelos de regresión, los valores de salida usados son continuos, la función a aproximar es $f : \mathbb{R}^n \to \mathbb{R}$ y la tarea del modelo es predecir uno de esos valores (por ejemplo, determinar el precio del dólar a partir de datos históricos).[19]

2.3.2. Aprendizaje profundo

El aprendizaje profundo (*deep learning*, DL) es un sub-área del aprendizaje automático donde se utiliza una una red neuronal prealimentada (*feedforward neural network*) como modelo que aproxima una función $f*$ definiendo un mapeo $y = f*(x; \theta)$ y aprende el valor de los parámetros θ de forma de lograr la mejor aproximación posible.

Este tipo de modelo se llama red porque típicamente se representa como una composición de varias funciones que se asocia con un grafo acíclico dirigido. Por ejemplo, como se observa en la figura 2.3.1, $f*$ puede estar compuesta de tres funciones f_1, f_2, f_3 compuestas (es decir que $f(x) = f_3(f_2(f_1(x)))$), cada una de estas funciones se denomina *capa* de la red (f_1 es la primera capa, f_2 la segunda, etc), la cantidad de capas se denomina profundidad de la red y es aquí donde aparece el término aprendizaje profundo (un modelo debe tener más de dos capas para denominarse así).

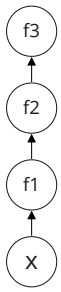


Figura 2.3.1: Representación en forma de grafo de la función $f*$ que define una red neuronal.

La red neuronal adapta sus parámetros internos en el proceso llamado entrenamiento, durante el cual el modelo aplica $f*$ a una entrada x siendo la última capa de la red (llamada capa de salida) la encargada de producir una salida $\overline{y}$ que aproxima a la etiqueta y, esta capa no recibe directamente la información de x sino que determina su valor a partir de los valores producidos por las capas anteriores (llamadas capas ocultas). Cada capa de una red se compone por una función (no lineal) de activación (a_i) y un conjunto de parámetros internos vectoriales llamados pesos (W_i) y bias (b_i), estos parámetros también son llamados unidades y generan una activación aplicando la fórmula $f_i = a_i(x \cdot W_i + b_i)$ sobre una entrada, el conjunto de todos los parámetros internos de las capas de una red componen a θ.

Las fórmulas anteriormente descritas se generalizan a su forma matricial permitiendo el entrenamiento en paralelo con un conjunto de ejemplos reduciendo tiempos, esto se conoce como entrenamiento por lotes (*batches*).

El término neuronal aparece porque la idea detrás de estos modelos está vagamente inspirada en como funcionan las neuronas de un cerebro, cada una de unidades de una capa puede verse como una neurona en el sentido en que una señal de entrada provoca una activación que sirve de señal para las demás unidades directamente conectadas.[18]

Funciones de pérdida

Un modelo de redes neuronales se entrena usando un proceso de optimización que requiere una función de pérdida (*loss*) que calcule el error entre una predicción $\overline{y}$ y el valor que se intenta aproximar y. En el caso de los modelos implementados en este proyecto se probaron varias funciones que se comentan a continuación.

Error cuadrático medio El error cuadrático medio (MSE) es el promedio de los errores elevados al cuadrado entre la predicción y el valor a predecir. En este caso, $y = [y_1, \ldots, y_n]$ es un vector de valores observados al que se quiere aproximar usando $\overline{y} = [\overline{y}_1, \ldots, \overline{y}_n]$. Esta función se calcula como:

$$\text{MSE}(y, \overline{y}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \overline{y}_i)^2$$

Esta función varía en el rango $[0, 1]$ y vale 0 solo si $\overline{y} = y$.[20]

Huber La función de pérdida *Huber* mide la diferencia entre la predicción $\overline{y}$ y la etiqueta y de forma que es cuadrática para valores con magnitud menor a un umbral δ y lineal para valores con magnitud mayor a ese umbral, de esta forma se vuelve resistente a que diferencias muy grandes que puedan influir demasiado en el error. Esta función se calcula como:

$$\text{Huber}_\delta(y, \overline{y}) = \begin{cases} \dfrac{(y_i - \overline{y}_i)^2}{2} & \text{si } |y_i - \overline{y}_i| \leq \delta \\ \dfrac{\delta^2}{2} + \delta \cdot (|y_i - \overline{y}_i| - \delta) & \text{si } |y_i - \overline{y}_i| > \delta \end{cases}$$

Esta función varía en el rango $[0, \infty)$ y vale 0 solo si $\overline{y} = y$.[21]

Reversed Huber La función de pérdida *Reversed Huber* es la función que intercambia los comportamientos internos vistos la función *Huber*, es decir que mide la diferencia entre la predicción $\overline{y}$ y la etiqueta y de forma que es cuadrática para valores con magnitud mayor a un umbral δ y lineal para valores con magnitud menor a ese umbral. Esta función se calcula como:

$$\text{ReversedHuber}_\delta(y, \overline{y}) = \begin{cases} |y_i - \overline{y}_i| & \text{si } |y_i - \overline{y}_i| \leq \delta \\ \dfrac{(y_i - \overline{y}_i)^2 + \delta^2}{2\delta} & \text{si } |y_i - \overline{y}_i| > \delta \end{cases}$$

Esta función varía en el rango $[0, \infty)$ y vale 0 solo si $\overline{y} = y$.[22]

En la figura 2.3.2, se comparan las curvas de las funciones *Huber* y *Reversed Huber* para distintos valores de δ, se observa como una tiene el comportamiento opuesto a la otra.

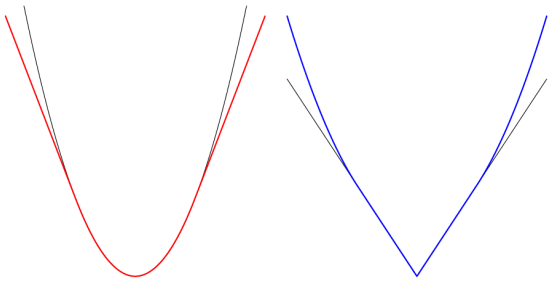


Figura 2.3.2: Comparación de la curva de la función *Huber* (izquierda) y la función *Reversed Huber* (derecha). (Figura extraída de [22])

Entrenamiento

Para entrenar un modelo de redes neuronales se usa el algoritmo de descenso por gradiente, la idea general de este es la de ajustar los parámetros θ de forma iterativa buscando minimizar una función de costo J, para esto se usa la siguiente ecuación de actualización:

$$\theta^{(i+1)} = \theta^{(i)} - \eta \nabla J(\theta^{(i)})$$

Donde $\theta^{(i)}$ son los parámetros internos del modelo durante la iteración i, η es un valor que determina la capacidad de convergencia del algoritmo (denominado *learning rate*) y $\nabla J(\theta^{(i)})$ es el vector gradiente que contiene todas las derivadas parciales de la función de costo para cada parámetro. θ se inicializa de forma aleatoria y luego, durante el paso $i + 1$, se calcula el gradiente de la función de costo usando todos los ejemplos del conjunto de entrenamiento y se modifica θ en la dirección opuesta al gradiente hasta que el algoritmo alcanza el mínimo (converge), este proceso se ejemplifica en la figura 2.3.3.[23]

Un parámetro importante en este algoritmo es el tamaño de los pasos, este valor se determina usando el *learning rate* y su elección puede determinar la convergencia o no, si η es muy pequeño, el algoritmo deberá iterar

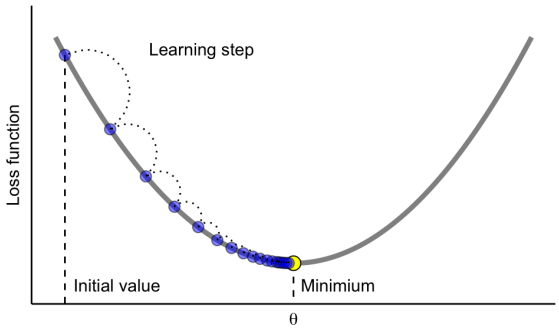


Figura 2.3.3: Esquema del algoritmo de descenso por gradiente. (Figura extraída de [23])

muchas veces para alcanzar el mínimo lo que puede consumir una gran cantidad de tiempo, en cambio, si η es muy grande, en cada paso se puede superar el mínimo y alejarse de él desencadenando la divergencia del algoritmo y fallando en la obtención de una buena solución. Ambos fenómenos se muestran en la figura 2.3.3.

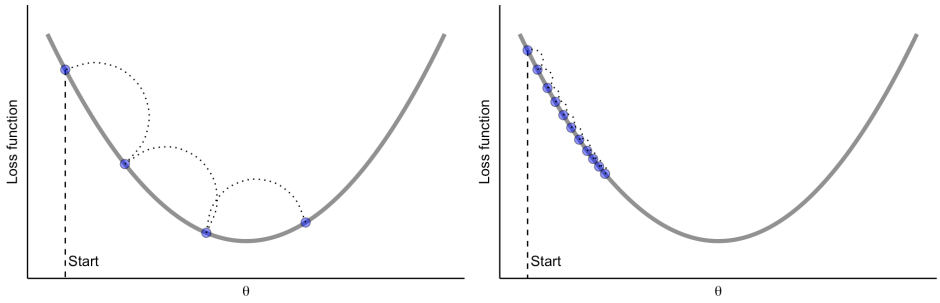


Figura 2.3.4: A la izquierda se muestra un esquema del algoritmo de descenso por gradiente con *learning rate* muy alto y a la derecha el caso opuesto. (Figura extraída de [23])

El mayor problema del algoritmo de descenso por gradiente es el hecho de que usa todos los ejemplos del conjunto de entrenamiento para computar el gradiente en cada iteración, esto lo vuelve muy lento cuando el conjunto de datos es grande. Para solucionar esto se usa el algoritmo de descenso por gradiente en *mini-batches*, en esta versión se usa un subconjunto de ejemplos (*batch*) en cada iteración, una vez que se utilizó todo el conjunto de datos se dice que se completó un *epoch*. Este algoritmo es menos regular que su versión original, acercándose al valor mínimo de J en promedio pero utilizando los recursos de manera más eficiente.[23]

Overfitting El fenómeno llamado *overfitting* se da cuando un modelo de ML ajusta sus parámetros internos perfectamente para el conjunto utilizado durante su entrenamiento, esto a costa de perder capacidad de generalización para nuevos datos. Para evitar que suceda el *overfitting* se pueden usar varias técnicas basadas en cambios en la estructura del modelo entrenado o en el uso de datos de validación durante el entrenamiento. El *early stopping* es una técnica que se basa en el uso de un conjunto de validación, distinto del conjunto de entrenamiento y el conjunto de test; el modelo es evaluado sobre el conjunto de validación luego de cada *epoch* y si el valor de la función de *loss* se muestra peor en los siguientes n *epochs* se detiene el entrenamiento, sin importar el valor sobre el conjunto de entrenamiento.[24]

Selección de *learning rate* Como la selección del *learning rate* determina en gran manera el éxito del algoritmo de descenso por gradiente y depende de los parámetros de cada modelo (y su arquitectura), existe un método

para obtener de manera automática un η óptimo para un modelo sobre un conjunto de datos. En este método se usa el análisis del comportamiento de J mientras se varía η, se inicializa el análisis con un valor pequeño que va gradualmente en aumento luego de cada iteración, este proceso se lleva a cabo por unos pocos *epochs* resultando en una gráfica como la mostrada en la figura 2.3.5, en la gráfica se distinguen tres fases: al principio η es muy bajo y la función de pérdida varía muy poco, luego J decrece rápidamente (y por lo tanto η es óptimo) y finalmente se llega a una fase donde η es demasiado grande y J comienza a diverger.[25]

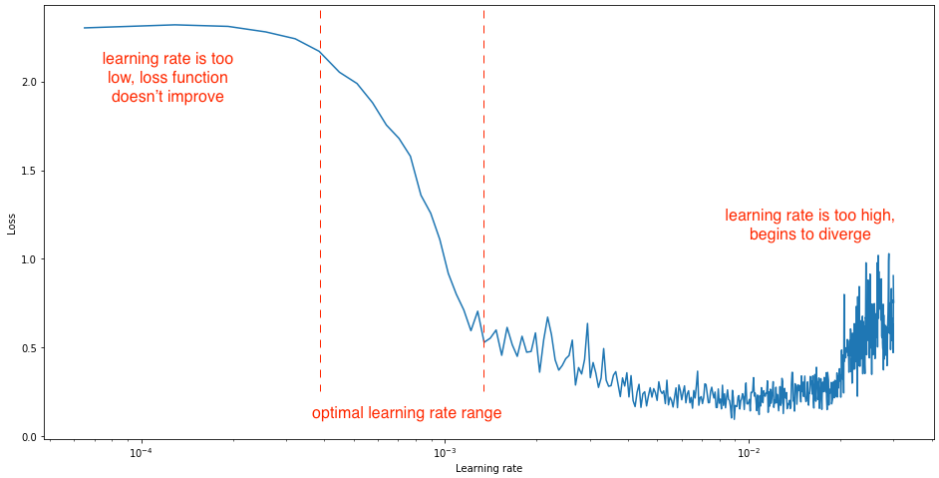


Figura 2.3.5: Ejemplo de análisis de función de costo durante la variación de *learning rate*. (Figura extraída de [26])

Una vez determinado el umbral de *learning rate* óptimo, se usan los extremos del umbral para variar η de formar cíclica durante el entrenamiento, la intuición detrás de este procedimiento es que al comienzo del algoritmo de descenso por gradiente rápidamente se pueden convertir de los valores de θ elegidos aleatoriamente en valores mejores usando η alto y luego refinar los valores con η más bajo, se repite esto de forma cíclica para permitir al algoritmo escapar de mínimos locales de J, estos mínimos suelen ser comunes por la alta dimensionalidad de la función de pérdida.[25]

Tipos de capas

Existen varios tipos de capas que pueden usarse en estos modelos, a continuación se describen las usadas en este proyecto.

Capa densa Este tipo de capa es el más básico de todos y está compuesto por una función de activación a y dos vectores, $W \in \mathbb{R}^{n \times m_o}$ llamado *weights* y $b \in \mathbb{R}^n$ llamado *bias*. Implementa la función $a(X^\top \cdot W + b)$ donde $X \in \mathbb{R}^{n \times m_i}$ son los valores de la entrada de la capa, n es la cantidad de ejemplos, m_i la cantidad de *features* de entrada y m_o la cantidad de unidades de la capa, la salida tiene dimensión $n \times m_o$ y durante el entrenamiento se ajustan los valores de W y b.[18]

Capa de *embeddings* Este tipo de capa está compuesto por una función de mapeo de valores naturales a vectores de tamaño fijo $m : \mathbb{N} \to \mathbb{R}^{m_o}$ y un vector $W \in \mathbb{R}^{w+1 \times m_o}$ llamado *weights*, este vector contiene la representación vectorial de cada uno de los w valores que puede tomar la entrada, se puede pensar en esta capa como un diccionario que contiene las representaciones de un número fijo de palabras, y si una palabra no se encuentra en el

mapeo se le asigna un valor fijo extra. Implementa la función $m(X)$ donde $X \in \mathbb{R}^n$ son los valores de la entrada de la capa, n es la cantidad de ejemplos y m_o el tamaño de las representaciones vectoriales, la salida tiene dimensión $n \times m_o$ y durante el entrenamiento se ajustan los valores de W.[27]

Capa convolucional Este tipo de capa está compuesto por una función de activación a y F vectores, $K_i \in \mathbb{R}^{n_k \times m_o}$ llamados *kernels* con $i \in \{1, \cdots, F\}$. Implementa la función $a(conv(X, K))$ donde $X \in \mathbb{R}^{n \times m_i}$ son los valores de la entrada de la capa, n es la cantidad de ejemplos, m_i la cantidad de *features* de entrada, F la cantidad de filtros, n_k y m_o la cantidad de unidades de la capa. La función *conv* (convolución) divide X en varios fragmentos del tamaño de K (X_k) y realiza el producto interno entre cada X_k y K, concatenando el resultado de utilizar los F *kernels*, en la figura 2.3.6 se muestra un diagrama de su aplicación, durante el entrenamiento se ajustan los valores de K_i, para mantener n ejemplos en la salida, se agregan los ceros necesarios en esa dimensión en X.[28]

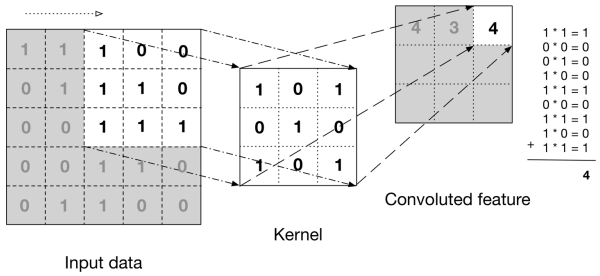


Figura 2.3.6: Ejemplo de aplicación de la función *conv* usada por una capa convolucional.

Capa recurrente Este tipo de capa está diseñada para modelar datos secuenciales como series temporales o el lenguaje natural. En términos generales, una capa recurrente itera sobre cada paso de una secuencia manteniendo un estado interno que codifica información sobre los pasos por los que ya ha iterado y usa la información del paso actual y los anteriores para generar la salida. Este tipo de capa tiene varias implementaciones, en el caso de este proyecto, se usan los siguientes:

- **LSTM** (*Long short term memory*) internamente maneja tres vectores llamados *gates*: *input*, *forget* y *output* que regulan cuanta información de la entrada y del estado interno mantener en cada iteración y cuanta información interna debe ser expuesta a la siguiente capa (respectivamente).[29]
- **GRU** (*Gated Recurrent Unit*) internamente maneja dos *gates*: *reset* y *update* que regulan cuando se debe olvidar el estado interno anterior y cuanta información de la entrada debe usarse para actualizar ese estado (respectivamente).[30]

Además, para cada uno de las implementaciones descritas, se ha agregado la posibilidad de usar cada capa recurrente en modo bidireccional, en este modo, se procesa la secuencia de entrada desde el principio hasta el final y viceversa para luego combinar estas dos salidas.[30]

Capa de *dropout* Este tipo de capa selecciona valores de la entrada de la capa de forma aleatoria según un valor $r \in [0, 1]$ y los transforma en 0 durante el entrenamiento, este proceso es ilustrado en la figura 2.3.7, esto ayuda a prevenir el *overfitting*.[31]

Capa de normalización Este tipo de capa normaliza las entradas de la capa para mantener la media cerca de 0 y la desviación estándar cerca de 1, este procedimiento permite que el tiempo de entrenamiento se reduzca y se prevenga el *overfitting*.[32]

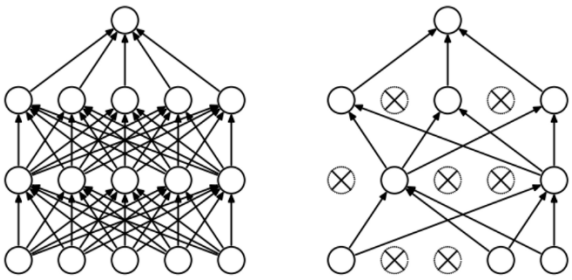


Figura 2.3.7: A la izquierda diagrama de una red neuronal con dos capas densas ocultas, a la derecha la misma red durante el entrenamiento con *dropout* aplicado en las entradas de las capas.

Funciones de activación Las funciones de activación de las capas antes descritas y que fueron usadas en este proyecto son:

Sigmoid: función con rango $(0, 1)$ calculada con la fórmula:

$$\text{sigmoid}(x) = \frac{1}{1 + e^{-x}}$$

ReLU (*Rectified Linear Unit*): función con rango $(0, +\infty)$ calculada con la fórmula:

$$\text{ReLU}(x) = \text{máx}(0, x)$$

ELU (*Exponential Linear Unit*): función con rango $(-\infty, +\infty)$ calculada con la fórmula:

$$\text{ELU}(x) = \begin{cases} x & \text{si } x \geq 0 \\ \alpha(e^x - 1) & \text{si } x < 0 \end{cases}$$

Swish: función con rango $(-\infty, +\infty)$ calculada con la fórmula:

$$\text{swish}(x) = x \cdot \text{sigmoid}(\beta x)$$

Tanh (*Hyperbolic tangent*): función con rango $(-1, 1)$ calculada con la fórmula:

$$\tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$$

Entrenamiento distribuido

Un modelo de redes neuronales se puede entrenar de manera distribuida (en varios nodos de procesamiento) permitiendo, en teoría, aumentar la capacidad del entrenamiento y reducir tiempos. Existen dos formas de distribuir el entrenamiento:

- **Paralelismo de datos** donde un mismo modelo se replica en múltiples nodos, cada uno procesa distintas particiones del conjunto de datos y luego obtienen los resultados del resto. El entrenamiento usando este tipo de paralelismo se puede categorizar en sincrónico, donde todos los nodos agregan los gradientes luego de cada paso, y asincrónico, donde los nodos actualizan los parámetros de la red sin esperar al resto. Típicamente el entrenamiento sincrónico es implementado usando la técnica *all-reduce* y el entrenamiento asincrónico mediante una arquitectura de centralización de parámetros.

- **Paralelismo de la arquitectura** donde diferentes partes del modelo se entrenan en distintos nodos entrenando sobre el mismo conjunto de datos. Este tipo de paralelismo funciona cuando un modelo tiene una arquitectura naturalmente paralela, es decir, cuando un modelo se puede dividir en varias ramas independientes que se juntan al final.

En el marco de este proyecto se experimentó con el **paralelismo de datos sincrónico**, en este modo de paralelismo, a cada nodo trabajador se le asigna una parte del entrenamiento y, luego de cada etapa, se distribuye la información recabada al resto de los trabajadores para que cada uno pueda agregarla y mantener los parámetros de la red en actualizados, en la figura 2.3.8 se muestra un esquema de este comportamiento.[33]

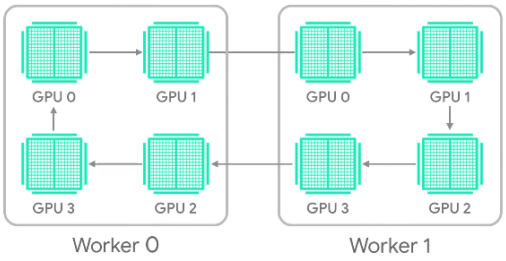


Figura 2.3.8: Esquema de paralelismo *all-reduce*. (Figura extraída de [34])

En este tipo de entrenamiento se dan las siguientes etapas:

1. **Configuración del cluster** se configura el cluster donde se ejecutará el entrenamiento desplegando $N-1$ nodos trabajadores y un nodo jefe, para cada uno de estos nodos se asigna un rol y se registra su dirección IP para permitir la comunicación. La diferencia entre los roles de los nodos es que el nodo jefe es el encargado de guardar en disco las estadísticas del entrenamiento y el modelo entrenado.
2. **Ensamblado del modelo** se ensambla el modelo en cada uno de los nodos del cluster.
3. **Distribución de datos** a cada nodo se le asigna una partición del conjunto de datos, esto se hace dividiendo el tamaño de *batch* (b) entre la cantidad de trabajadores y particionando el conjunto según el resultado obtenido, para aprovechar la potencia de los trabajadores se escala b según N mantentiendo el b original en cada nodo.
4. **Entrenamiento** cada nodo realiza una predicción sobre los datos obtenidos, luego de cada etapa, distribuye los resultados a los nodos del cluster y espera a recibir los resultados del resto. Una vez que obtiene todos los resultados, completa una iteración del algoritmo de descenso por gradiente y continua con el entrenamiento. Esta etapa se muestra en la figura 2.3.9.

2.3.3. *Random Forest*

Otro tipo de modelo usado para el aprendizaje supervisado es el *Random Forest*[35] que pertenece a conjunto de algoritmos de ML llamados tradicionales. Un modelo de este tipo está compuesto por un conjunto de *Decision Trees*[36], cada uno de estos modelos contiene un árbol cuyo proceso para obtener una predicción consta en consultar en el nodo inicial el valor de una *feature* de entrada y, en base al valor reportado, descender por una de sus ramas, repitiendo estas acciones hasta alcanzar una hoja y retornando el valor de salida en ella, este valor se calcula como el promedio de los ejemplos encontrados durante el entrenamiento que pertenecen al subconjunto representado por ese nodo. En la figura 2.3.10 se muestra un ejemplo de un árbol generado, en cada nodo se

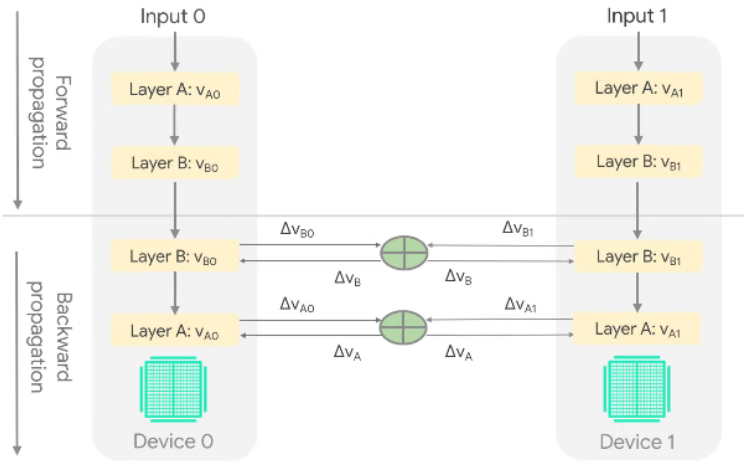


Figura 2.3.9: Esquema de *all-reduce* durante el entrenamiento. (Figura extraída de [34])

retiene información sobre el entrenamiento como por ejemplo la cantidad de muestras usadas y el valor del error hasta ese punto.

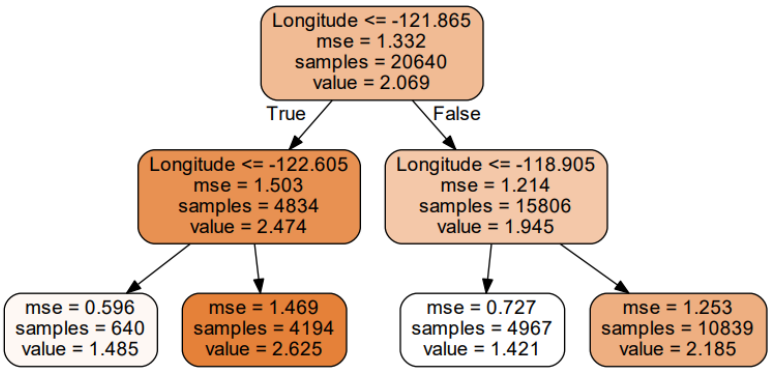


Figura 2.3.10: Esquema ejemplo de la estructura de un *Decision Tree*.

Durante el entrenamiento de un árbol, se deciden los cortes de cada nodo buscando una *feature* f y un umbral f_k (en el ejemplo, para el nodo raíz la *feature* f seleccionada es *Longitude* y $f_t = -121{,}865$) de forma que se minimice la función de costo J:

$$\begin{cases} J(f, f_t) & = \frac{m_l}{m}\text{MSE}_l + \frac{m_r}{m}\text{MSE}_r \\ \text{MSE}_{\text{node}} & = \frac{1}{|\text{node}|}\sum_{i \in \text{node}} (y_i - \overline{y}_i)^2 \end{cases}$$

Donde m es la cantidad total de ejemplos, m_{node} es la cantidad de ejemplos a la izquierda o a la derecha y MSE_{node} es el valor de MSE para los ejemplos a la izquierda o a la derecha.

El algoritmo se detiene al alcanzar un máximo permitido de profundidad (cantidad de niveles) o al no poder encontrar otro corte que minimice J.[36]

En el caso de los árboles que componen un modelo de *Random Forest*, cada uno es entrenado en un subconjunto

del conjunto de entrenamiento original, en cada uno de estos subconjuntos se usa solo algunas de las *features* que se eligen aleatoriamente. Para obtener una predicción, se promedian las predicciones de todos los árboles.[35]

Otra variante de este modelo es el *Extremely Randomized Trees ensemble*, este se diferencia del *Random Forest* en la forma en la que se decide el corte f_t para una *feature*, pasando a ser una decisión aleatoria frente a la búsqueda del mejor corte como se explicó anteriormente, este cambio permite la aceleración del proceso de entrenamiento obteniendo resultados iguales o mejores.[37]

2.3.4. Tensorflow

Tensorflow[38] es una biblioteca *open-source* usada para la implementación de los modelos de redes neuronales. Esta biblioteca se caracteriza por proveer múltiples APIs que se clasifican en dos categorías:

- **APIs de bajo nivel**: dan a quienes desarrollan modelos control total de lo que sucede en cada capa del modelo durante el entrenamiento y evaluación. Este conjunto de métodos se usa principalmente por investigadores y desarrolladores de nuevos modelos.
- **APIs de alto nivel**: construidas usando las APIs de bajo nivel, permiten menor control de los componentes de un modelo pero la curva de aprendizaje es menor y la implementación de modelos complejos se puede realizar en pocas lineas de código. Este conjunto de métodos se usa principalmente por desarrolladores que emplean modelos pre-entrenados o capas ya existentes.

Tensorflow también contiene otros módulos útiles para el procesamiento de los datos de entrada usados durante el entrenamiento y evaluación de los modelos.

2.3.5. Scikit-Learn

Scikit-Learn[39] es una de las bibliotecas *open-source* de ML más usadas en Python, provee la implementación de modelos usados para resolver muchos problemas de aprendizaje supervisado y no supervisado. Todos los modelos utilizan una interfaz común para facilitar su aprendizaje y uso.

Las APIs provistas por esta biblioteca se pueden clasificar en los siguientes grupos:

- **Modelos de aprendizaje supervisado**: conjunto de modelos para realizar regresión o clasificación en conjuntos de datos etiquetados (como regresión lineal o modelos basados en árboles).
- ***Clustering***: conjunto de modelos usados para agrupar datos sin etiqueta (como KMeans).
- ***Cross Validation***: métodos usados para estimar el desempeño de modelos de aprendizaje supervisado en conjuntos de datos no antes utilizados.
- **Reducción de dimensionalidad**: métodos usados para reducir la cantidad de atributos en un conjunto de datos con la finalidad de visualización o la selección de *features* (como PCA).
- ***Ensembles***: métodos usados para combinar las predicciones de varios modelos de aprendizaje supervisado y obtener nuevas predicciones.
- ***Feature selection***: métodos usados para identificar el subconjunto de *features* más relevante de un conjunto de datos.

2.4. Búsqueda de hiperparámetros

Cada tipo de modelo de ML se caracteriza por un conjunto de parámetros que determinan su estructura y desempeño frente a un conjunto de datos particular, estos hiperparámetros deben ajustarse realizando pruebas

sobre un conjunto de datos de entrenamiento, este proceso puede darse de forma manual o automática y para ello, deben definirse el método de selección de los hiperparámetros a probar y el método de poda (*pruning*) de modelos con desempeño peor a otros ya explorados.

2.4.1. Métodos de selección de hiperparámetros

Al momento de probar distintas configuraciones de hiperparámetros, la elección de cada combinación a probar es fundamental, por esto existen varios métodos de selección, los más populares son:

Grid Search

Es un método de fuerza bruta donde se prueban todas las posibles combinaciones de parámetros existentes. Este método asegura encontrar la mejor combinación existente para los valores seleccionados pero tiene una desventaja fundamental, cuando se tiene un conjunto de parámetros con muchos valores posibles, el número de pruebas realizadas se puede volver inmenso. Por ejemplo, si se tienen 10 hiperparámetros con 4 posibles valores cada uno, y en promedio el entrenamiento de un modelo se realiza en 30 minutos, se tardaría 21 días en encontrar el mejor conjunto de hiperparámetros.

Otro problema de este método es que solo se prueban los valores especificados, por lo tanto, si no se considera un valor para un hiperparámetro que obtendría la mejor solución, no se llegaría nunca a esa combinación.

Random Search

Este método elije aleatoriamente los valores a probar en las combinaciones de hiperparámetros a partir de las distribuciones estadísticas seleccionadas para cada uno. Este método ha demostrado ser más eficiente que el *grid search* tanto en la práctica como en la teoría.[40]

Parte de las razones por las que *random search* supera a *grid search* es que, típicamente, solo unos pocos hiperparámetros infieren en el rendimiento de un modelo. Por lo tanto, encontrar valores óptimos en esos hiperparámetros tendrá mayor impacto que encontrar una combinación óptima de todos los hiperparámetros (esto se ejemplifica en la figura 2.4.1), el subconjunto de los parámetros más importantes depende de cada problema y no es tarea sencilla encontrarlo. *Random search* tiene más probabilidades que *grid search* de encontrar el valor óptimo para los hiperparámetros importantes porque realiza la búsqueda en un dominio más amplio.

Tree-structured Parzen Estimator (TPE)

Este método de búsqueda es un algoritmo basado en *sequential model-based optimization* (SMBO)[42], en estos algoritmos se construye un modelo probabilístico de la función objetivo ($p(y|x)$) que se usa para seleccionar el conjunto de hiperparámetros más prometedores y evaluarlos en la función objetivo verdadera.

La función de selección usada para elegir el siguiente conjunto de hiperparámetros a evaluar es la función de *Expected Improvement* (EI) definida como:

$$EI_{y^*}(x) = \int_{-\infty}^{y^*} (y^* - y)p(y|x)dy$$

Donde y^* determina cierto umbral de la función objetivo, x es el conjunto de hiperparámetros propuesto, y es el valor verdadero de la función objetivo evaluada en x y $p(y|x)$ es el modelo probabilístico (también llamada función de subrogación) que expresa la probabilidad de y dado x. El algoritmo busca maximizar la EI con respecto a x.

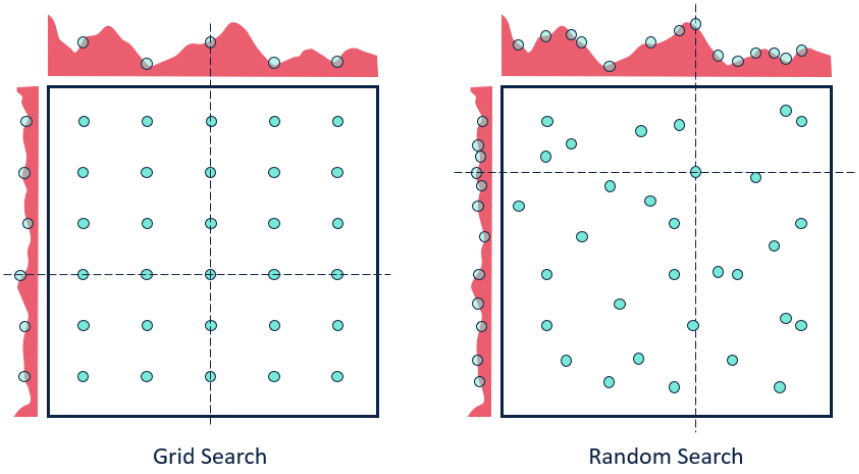


Figura 2.4.1: Comparación de los métodos *grid search* y *random search* en la búsqueda de los valores posibles de dos hiperparámetros. En el eje x, se muestra un hiperparámetro importante cuya evaluación es mayor (en rojo) y en el eje y otro hiperparámetro de menor importancia. (Figura extraída de [41])

La función de subrogación es particular para el caso de TPE, en este estimador se construye usando como base la regla de *Bayes*[43] para calcular $p(y|x)$ en base a $p(x|y)$, que es la probabilidad de un conjunto de hiperparámetros dado un valor de la función objetivo y se define como:

$$p(x|y) = \begin{cases} l(x) & \text{si } y < y^* \\ g(x) & \text{si } y \geq y^* \end{cases}$$

Donde $y < y^*$ representa un valor de la función objetivo menor al umbral determinado. En esta función se hace dos distribuciones distintas para los hiperparámetros, una donde el valor de la función objetivo es menor que el umbral ($l(x)$) y otra donde es mayor ($g(x)$).

El algoritmo mantiene una historia de pares (x, y) y actualiza $l(x)$ y $g(x)$ a medida que se prueban nuevos pares y se agregan a la historia, luego seleccionan nuevos candidatos buscando maximizar el EI.

A diferencia de los métodos anteriores, este selecciona el siguiente conjunto de hiperparámetros basándose en resultados previos, de esta forma se logra que la función objetivo mejore más rápido lo que repercute en la cantidad de conjuntos de hiperparámetros necesarios para lograr buenos resultados (este comportamiento se muestra en la figura 2.4.2).

2.4.2. Métodos de *pruning* de soluciones

Al realizar la búsqueda de arquitecturas, muchas combinaciones probadas presentan desempeño muy bajo desde las primeras iteraciones del entrenamiento, el entrenamiento completo de estos modelos reduce la cantidad de modelos probados en un período limitado de tiempo, para aprovechar los resursos de mejor forma se usa el mecanismo de poda (*pruning*) que permite detener el entrenamiento de modelos que no son prometedores comparándolos con el resto. Los más usados son:

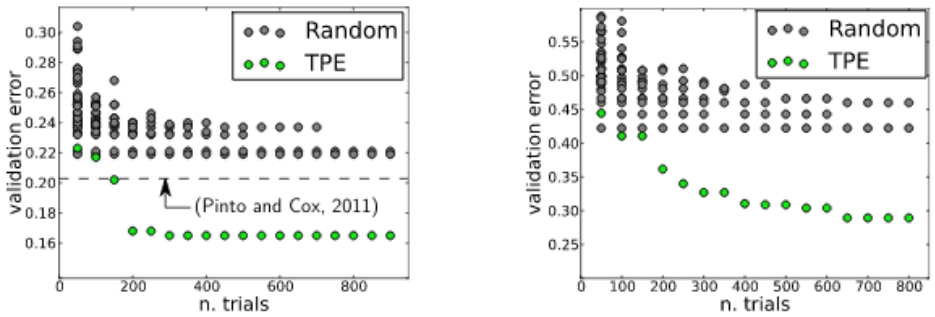


Figura 2.4.2: Comparación de los métodos *random search* y TPE en la optimización de una función objetivo sobre dos conjuntos. Los puntos grises representan el error más bajo obtenido de varias ejecuciones de *random search* y los puntos verdes representan el error más bajo obtenido por una ejecución de TPE. (Figura extraída de [44])

Median Stopping Rule

Este método detiene el entrenamiento de un modelo si, el mejor valor de la función objetivo en el paso s es estrictamente peor que la mediana de los valores de la función objetivo en el paso s para los modelos ya entrenados.[45]

Successive Halving Algorithm (SHA)

Para este algoritmo se define la cantidad total de recursos a usar para el entrenamiento de cada modelo (R) y se asigna una porción (r) de ese total a cada uno de los n modelos a probar, una vez superado ese mínimo y mientras los modelos no consuman el máximo de recursos, se evalúan todos (cada vez que se alcanza el limite de recursos asignados) manteniendo los mejores $\frac{1}{\eta}$ y aumentando sus recursos por un factor de η.[46] La definición de los recursos usados en este algoritmo depende del tipo de modelo a entrenar.

SHA acarrea el llamado "n vs $\frac{R}{n}$"*trade-off*, esto significa que, para un R dado, no es claro si es mejor entrenar muchos modelos con pocos recursos (n grande) o entrenar pocos modelos otorgandoles gran parte de los recuros ($\frac{R}{n}$ grande). El problema de no configurar correctamente estos parámetros resultaría en que, si n es grande entonces todos los buenos modelos que requieran un nivel de recursos importante para converger se perderían en las primeras iteraciones mientras que, si $\frac{R}{n}$ grande entonces se perderán muchos recursos en modelos muy malos que podrían ser detenidos anteriormente.[47]

Hyperband

Este algoritmo soluciona el problema presentado por SHA considerando distintos valores de n y buscando el valor óptimo mediante *grid search*, para cada uno de los valores probados se ejecuta SHA de forma independiente y cada una de estas s pruebas se le llama *bracket*. *Hyperband* comienza con el *bracket* más agresivo que configura n para maximizar la exploración y, a cada *bracket* siguiente, reduce n por un factor de aproximadamente η hasta el último *bracket* en donde a cada modelo se le asigna R recursos, maximizando la explotación. De esta forma el algoritmo es capaz de ajustarse tanto a situaciones donde la explotación es importante como a otras donde la exploración debería ser mayormente utilizada.[47]

En la figura 2.4.3 se observa una comparación entre SHA e *Hyperband* pudiendo comprobar que el segundo obtiene resultados parecidos al primero con la configuración óptima.

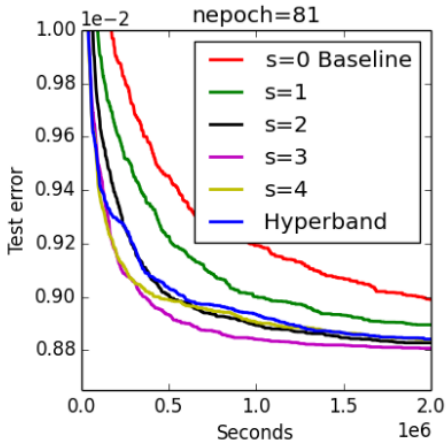


Figura 2.4.3: Comparación de los algoritmos *Hyperband* y SHA en la optimización de un modelo de redes neuronales sobre un conjunto de datos de prueba. Cada valor de s representa una configuración particular de n para SHA. (Figura extraída de [47])

2.4.3. Optuna

Optuna[48] es una biblioteca *open-source* usada para la optimización de hiperparámetros automática.

Esta biblioteca encapsula cada proceso de búsqueda de hiperparámetros en un estudio que puede ser almacenado en una base de datos relacional (RBD). Si la base de datos soporta el acceso concurrente, Optuna permite que varios nodos ejecuten el estudio en paralelo, almacenando los resultados de las evaluaciones realizadas en cada uno en la base de datos y obteniendo el historial centralizado en la base antes de decidir otra configuración a probar.

Además, hace énfasis en la definición del espacio de búsqueda en tiempo de ejecución, esto significa que el conjunto de hiperparámetros y sus respectivos espacios pueden variar entre experimentos e incluso depender unos de otros. Cada proceso de búsqueda se define como una función particular lo que permite gran modularización y extensibilidad.

Por último, Optuna cuenta con un amplio conjunto de integraciones con bibliotecas usadas para la creación y entrenamiento de modelos de ML.

2.5. Trabajo relacionado

En esta sección se presenta un estudio del estado del arte de la predicción de consultas DNS y tráfico y el uso de técnicas para la automatización del *machine learning*.

2.5.1. Predicción de consultas DNS y tráfico

Las tareas de predicción de consultas DNS y de tráfico se definen como tareas de regresión y tradicionalmente se han abordado usando predicción de series temporales (*time series forecasting*, TSF), el objetivo de TSF es construir un modelo de regresión capaz de obtener una predicción correcta para el volumen de tráfico futuro a partir del actual (de igual forma que con las consultas DNS).

Existe bastante trabajo en el área de la predicción y clasificación de tráfico a partir de datos actuales de tráfico usando *machine learning*, estos acercamientos muestran buen rendimiento tanto de predicciones a corto como

a largo plazo, usando una cantidad limitada de datos y *features*, también existe trabajo sobre la predicción del comportamiento de los usuarios usando consultas DNS.

Aceto et al.[49] estudiaron el comportamiento de modelos de aprendizaje profundo en la clasificación de tráfico en redes móviles, evaluaron varios modelos como *multilayer perceptron* (MLP), redes neuronales con capas *Long Short-Term Memory* (LSTM) y capas convolucionales (CNN), encontraron que los modelos que tuvieron mejor desempeño fueron los compuestos por capas CNN.

Azari et al.[50] realizaron un análisis comparativo entre redes neuronales con capas LSTM y modelos lineales ARIMA para la predicción de tráfico, estudiaron el efecto de diferentes parámetros en los modelos y su efecto en su desempeño. Sus simulaciones resultaron en la prueba de la superioridad de las redes LSTM sobre ARIMA, particularmente cuando las series temporales tienen una extensión considerable. Sin embargo, en casos puntuales, ARIMA resultó en un desempeño casi óptimo con un nivel de complejidad mucho menor. Dalgkitsis et al.[51] también compararon métodos basados en redes LSTM con otros modelos como *Support Vector Machines* (SVM), ARIMA y SARIMAX para la predicción de tráfico en redes móviles, el método propuesto mejora el desempeño de los otros en términos de error de predicción, el MSE alcanzado por LSTM fue de $1{,}685$, SARIMAX $11{,}26$, ARIMA $6{,}53$ y SVM $3{,}24$.

Panza et al.[52] analizaron series temporales de consultas DNS usando técnicas estadísticas y de *machine learning* y encontraron que patrones de comportamiento humano se ven reflejados en los datos DNS, concluyendo que estos patrones podrían ser minados y reconocidos usando los métodos apropiados y buen procesamiento de datos.

Madariaga et al.[53] aplicaron modelos de redes neuronales con capas LSTM y capas CNN a series temporales de consultas DNS, concluyendo que el acercamiento es válido tanto para predicción de tráfico y detección de anomalías.

Por otro lado, Chen et al.[54] investigaron la posibilidad de reducir el costo de monitoreo y recolección de datos de volumen de tráfico mediante la predicción del tráfico futuro usando estadísticos (como el conteo de conexiones) como entrada. Ellos usaron modelos de redes neuronales con capas LSTM sobre el conjunto de datos de conteos recolectados cada 5 minutos durante 24 semanas, la red reportó un rendimiento en MSE de $0{,}3$, siendo el doble que el producido usando como entrada el volumen de tráfico y haciendo notoria la dificultad de la predicción mediante otro tipo de valores a los usuales.

2.5.2. *Machine Learning* automatizado

Dentro del *machine learning* varias tareas pueden ser realizadas de forma automática, en este proyecto se estudiarán la automatización de la ingeniería de *features* y la búsqueda de hiperparámetros.

Automatización de ingeniería de *features*

Una de las metodologías más populares de ingeniería de *features* es la de aplicar diferentes transformaciones en las *features* originales para crear nuevas, por ejemplo Katz et al.[55] identificaron algunos operadores básicos que pueden transformar o combinar algunas variables para crear una nueva. La intuición detrás de esta metodología es la siguiente: "*features* con gran carga de información comúnmente surgen de la manipulación de otras más básicas". Usando estos operadores, Katz et al.[55] crearon varias *features* candidatas que fueron usadas en el entrenamiento de modelos y luego comparadas usando los resultados de la evaluación de esos modelos en distintas tareas y conjuntos de datos.

Por otro lado, Dos Santos et al.[56] utilizaron redes convolucionales para extraer *features* a partir de textos usados en tareas de procesamiento de lenguaje natural. Zhang y Wallace[28] también crearon nuevas *features*

usando redes convolucionales para clasificación de textos, en ambos casos compararon las *features* creadas con otras creadas de forma manual mejorando los resultados obtenidos.

Automatización de búsqueda de hiperparámetros

Hay una gran cantidad de literatura respecto a la búsqueda de hiperparámetros de un modelo de ML. Inicialmente se usaron métodos como *grid search* para explorar el espacio de hiperparámetros pero los investigadores notaron que este se vuelve con facilidad demasiado grande para encontrar un buen conjunto de parámetros dentro de un período de tiempo razonable. Bergstra y Bengio[40] descubrieron que el uso de *random search* frecuentemente logra un trabajo decente para encontrar un conjunto óptimo de hiperparámetros, siendo más eficiente que la búsqueda manual y *grid search*, mostraron que *random search* es capaz de encontrar buenos o mejores modelos con una fracción del tiempo necesario en *grid search*. Estos investigadores mostraron que para muchos conjuntos de datos, solo una porción de los hiperparámetros son importantes pero no son los mismos entre los distintos conjuntos.

Por otro lado, investigadores probaron *sequential model based optimization* (SMBO) utilizando *clusters* de computación y GPUs (Bergstra et al.[42]) y levantando las limitaciones de SMBO (Hutter et al.[57]), estos resultados se dieron porque el poder computacional actual permite ejecutar más experimentos durante la misma cantidad de tiempo, resultando en mejores resultados que *random search* en la práctica. Investigaciones recientes muestran que *frameworks* que usan *Bayesian optimization* para la búsqueda de hiperparámetros han mostrado resultados interesantes (Akiba et al. [48]).

Capítulo 3

Metodología de trabajo

En este capítulo se detalla la estructura del repositorio de código de este proyecto, se comentan detalles de la implementación y los ambientes de trabajo. También se comentan las tareas realizadas y su ejecución.

3.1. Estructura del proyecto

Para la implementación del código se usó el lenguaje de programación Python, esta elección se tomó teniendo en cuenta su popularidad y disposición de bibliotecas ampliamente probadas y usadas en el área de ciencia de datos y *machine learning*.[58] Para la implementación de scripts y utilidades se usó Bash ya que se trabajó siempre sobre plataformas con Linux.

El código de todo el proyecto se encuentra en el repositorio de *Gitlab* `https://gitlab.fing.edu.uy/plan-ceibal/supervised-learning.git`. Este repositorio se compone de varios directorios que separan el código según sus funciones, se describen los de mayor importancia:

`/confs` Contiene las configuraciones a usar para desplegar aplicaciones mediante spark-submit.

`/data` Contiene datos necesarios para la correcta ejecución del código, un ejemplo de estos datos son las tablas usadas para imputar datos.

`/data_processing` Contiene los módulos de Python que se utilizan para el procesamiento de los datos del proyecto.

`/models` Contiene los módulos de Python que se utilizan para el entrenamiento de los modelos de predicción.

`/notebooks` Contiene *notebooks* de Jupyter que se utilizan para el análisis de los datos del proyecto.

`/scripts` Contiene scripts de Bash y Python utilizados para tareas de interacción con el ambiente de trabajo.

`/zeppelin` Contiene los módulos de Python que se utilizan para manejar *interpreters* de Zeppelin.

3.2. Ambiente de trabajo

Durante el proyecto, se utilizaron tres ambientes de trabajo, uno instalado en Facultad de Ingeniería, otro en la plataforma de *big data* de Ceibal y el último en el cluster de ClusterUy.[59]

Para el procesamiento de datos mediante *big data* se decidió usar la herramienta *Spark*, esta herramienta se usó en el ambiente de Facultad de Ingeniería y en el de Ceibal. El primero se preparó desde cero durante la etapa más temprana del proyecto, en ambos casos se utilizó la plataforma *Hortonworks Data Platform* (HDP).

En el ambiente de Facultad de Ingeniería se usó la versión lista para funcionar *out of the box* **HDP Sandbox**, esta versión se puede instalar tanto usando *VirtualBox* como *Docker*, en un principio se probó con la instalación con *VirtualBox* pero, luego de comprobar el *overhead* generado y de una posible dificultad para reasignar recursos a medida que se necesitaran, se decidió continuar con la instalación con *Docker*. La plataforma se instaló en un servidor dentro de facultad y se le destinaron 150 GB de memoria RAM, 34 virtual cores y 30 TB de almacenamiento HDD. Los pasos a seguir para la instalación se encuentran en la web de *Cloudera*[14] y en el archivo `README.md` del repositorio en *Gitlab*.

Como los datos a utilizar se encontraban en un directorio fuera del volumen al que tiene acceso *Docker*, se decidió darle acceso a esa carpeta modificando uno de los scripts usados para crear los contenedores. Tales scripts se encuentran en el directorio `/docker-deploy-hdp`. En el script `/docker-deploy-hdp/docker-deploy-hdp30.sh` se agregaron las variables **sharedDir** y **destDir** que indican a *Docker* que carpeta debe compartir con el contenedor a crear y en que directorio debe ser montada.

Una vez instalado y configurado el ambiente, sus servicios quedaron accesibles a través de `http://localhost:1080` y `http://sandbox-hdp.hortonworks.com:1080`.

En la plataforma de Ceibal se usó la versión estándar de HDP y se crearon dos ambientes de trabajo: **desa**, ambiente de desarrollo y pruebas y **prod**, ambiente de producción.

3.2.1. Acceso al servidor

Para poder acceder al servidor donde se desplegó el ambiente de trabajo en Facultad de Ingeniería, se creó un script que agiliza el proceso y permite acceder a los servicios a través de `http://localhost:1080`.

3.2.2. Conda

La versión usada de *HDP* tenía preinstalada la versión 2.7.3 de *Python* junto con paquetes preconfigurados para poder ejecutar algoritmos sobre *Spark*, dado que esa versión dejó de tener soporte en 2020[60] y junto con la necesidad de instalar nuevos paquetes con los que trabajar, se optó por instalar *Conda* y crear varios ambientes virtuales que aseguren portabilidad y aislabilidad. *Conda* es un manejador de paquetes y entornos virtuales para una variedad de lenguajes (Python, R, Ruby, Lua, Scala, Java, JavaScript, C/ C++, FORTRAN, etc) que permite tener un ambiente de trabajo específico y separado del resto sin influir en el sistema donde se ejecuten las tareas.

Los ambientes virtuales creados fueron llamados **data** y **autoML**, el primero enfocado en el análisis y procesamiento de *big data* y el segundo en el despliegue del sistema de autoML.

Los principales paquetes instalados en cada ambiente virtual se describen a continuación:
Ambiente *data*:

- **pandas**[61] biblioteca para el análisis y procesamiento de datos en forma tabular.
- **Koalas**[62] port de la biblioteca *pandas* y sus principales objetos (*DataFrame* y *Series*) sobre *Spark*, el uso de este paquete permite probar el código sobre una porción reducida del conjunto de datos en local y luego migrar al entorno de *big data* sin grandes cambios.
- **PyArrow**[63] biblioteca que incluye *bindings* de *Apache Arrow* permitiendo a *PySpark* utilizar sus optimizaciones, además aporta un medio de interacción con *HDFS*.
- **plotly**[64] biblioteca usada para generar gráficos sobre los datos, su principal diferencia con otras es la capacidad de generar gráficos interactivos.
- **PyDomainExtractor**[65] biblioteca usada para parsear el *gTLD* o *ccTLD* (*generic* o *country code top-level domain*) de un dominio dado usando la lista pública de sufijos (*Public Suffix List*, **PSL**).

- **geopy**[66] biblioteca usada calcular la distancia geodésica entre dos coordenadas geográficas (la distancia geodésica es la distancia más corta sobre la superficie de un modelo elipsoidal de la Tierra).

Ambiente *autoML*:

- **TensorFlow**[38] biblioteca de *machine learning* centrada en redes neuronales, aporta APIs de alto y bajo nivel para desarrollar modelos potentes de forma rápida.
- **Optuna**[48] biblioteca usada para la búsqueda y optimización automática de hyperparámetros de un modelo.

3.3. Actividades realizadas

Las principales actividades llevadas a cabo se enumeran a continuación, cada una con un código que identifica a que OE pertenecen (o T en caso de pertenecer al informe del proyecto):

OE1_1 Investigación sobre uso de HDP y preparación de ambientes de trabajo.

OE1_2 Procesamiento de datos de consultas DNS.

OE1_3 Procesamiento de datos de tráfico.

OE1_4 Análisis estadísticos de los datos procesados.

OE2_1 Investigación del estado del arte de autoML.

OE2_2 Diseño e implementación del sistema de autoML.

OE2_3 Investigación sobre entrenamiento de modelos de redes neuronales sobre HDP.

OE3_1,OE4_1 Ejecución de experimentos y evaluación de resultados.

T_1 Elaboración de informe del proyecto.

En la figura 3.3.1 se muestra el diagrama de *Gantt* donde se muestra la ejecución de las tareas antes descritas, este proyecto se realizó entre abril de 2020 y diciembre de 2021 y no se realizaron tareas relacionadas al proyecto durante los meses de enero, julio y agosto de 2021 por lo que en total se llevo a cabo en 18 meses.

En la tabla 3.3.1 se resume lo mostrado en el diagrama de *Gantt* y se comentan los resultados obtenidos de cada una de las actividades. Se destaca que el 50 % del tiempo se dedicó a tareas de procesamiento y análisis de datos, 44 % a tareas de diseño del sistema de autoML, experimentos utilizando el sistema y análisis de resultados y finalmente 33 % a generar documentación relacionada al proyecto. En particular la tarea OE2_3 se realizó por más tiempo del que debería pues sus resultados no fueron satisfactorios ya que no se pudo usar la plataforma de Ceibal para realizar los experimentos como estaba planeado.

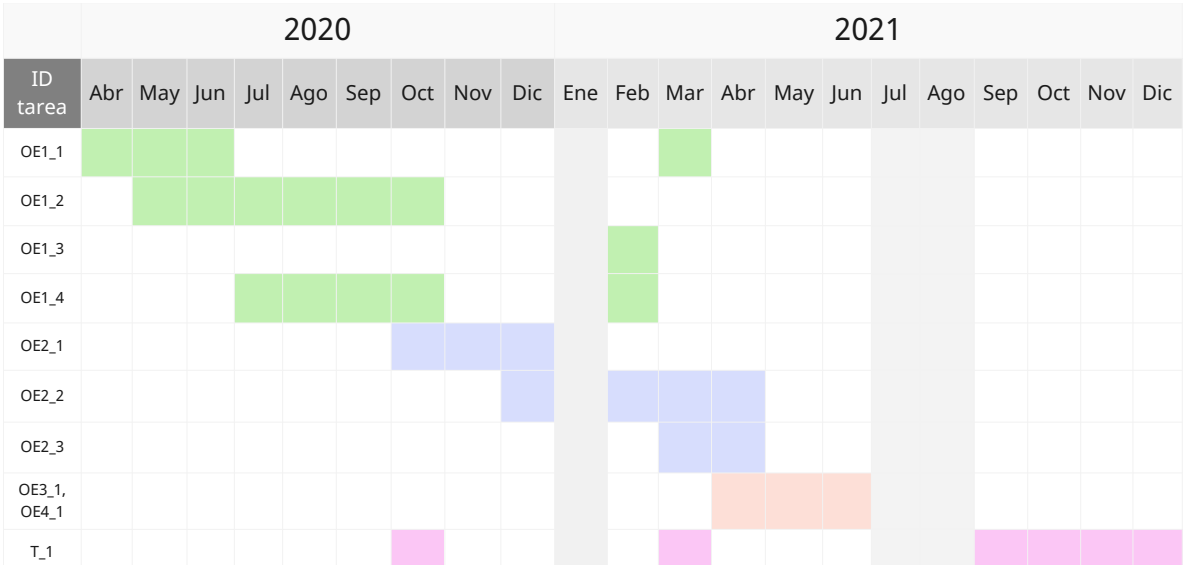


Figura 3.3.1: Diagrama de *Gantt* detallando el tiempo de ejecución de las principales actividades del proyecto.

ID tarea	Inicio-Fin	Extensión en meses	Resultado
OE1_1	04/2020-03/2021	4	Ambientes de trabajo
OE1_2	05/2020-10/2020	6	Conjunto de datos de consultas DNS
OE1_3	02/2021-02/2021	1	Conjunto de datos de tráfico
OE1_4	07/2020-02/2021	5	Análisis de conjuntos de datos
OE2_1	10/2020-12/2021	3	Información sobre autoML y el estado del arte
OE2_2	12/2020-04/2021	4	Sistema de autoML y arquitectura base de modelo de redes neuronales
OE2_3	03/2021-05/2021	2	Comparación de bibliotecas de entrenamiento sobre HDP
OE3_1,OE4_1	04/2021-06/2021	3	Modelos de ML para resolución de problemas
T_1	10/2020-12/2021	6	Informe de proyecto

Tabla 3.3.1: Resumen de ejecución de actividades del proyecto y sus resultados.

Capítulo 4

Procesamiento y análisis de datos

En este capítulo se detalla el trabajo realizado para cumplir con el OE1, es decir, el procesamiento realizado con el fin de conseguir un conjunto con información útil y consistente para trabajar durante este proyecto, puntualizando los motivos de los cambios realizados y su impacto. También se muestra el análisis realizado de los datos para determinar su utilidad y uso en las siguientes etapas del proyecto.

4.1. Descripción de los datos

En este proyecto se trabajó con dos conjuntos de datos brindados por Ceibal, cada uno con información complementaria que se usó para generar los modelos de predicción objetivos. Cabe destacar la importancia del análisis y mejora de los datos siendo esta una etapa sobre la que se iteró constantemente durante la extensión del trabajo.

4.1.1. Conjunto de consultas DNS

Este conjunto contenía logs de consultas a DNS realizadas por todos los dispositivos conectados a las redes de Ceibal montadas en todo el país. Se tenían registros del período comprendido entre el 09/02/2019 y el 31/12/2019, en total se contaba con más de treinta y dos mil millones de datos (32.781.543.191).

Los registros fueron generados por la herramienta *Cisco Umbrella* instalada sobre la red Ceibal con el fin de controlar el tráfico, aumentando la seguridad y bloqueando actividad maliciosa (sección 2.1.2).

En el entorno de facultad de ingeniería, los datos se encontraban en el directorio `/opt/datos/datos_umbrella/dnslogs/` del sistema de archivos local, por cada día de datos se generó un directorio que a su vez contenía un conjunto de archivos CSV comprimidos con la información separada en columnas. Para realizar el análisis y transformación los datos fueron descomprimidos, se creó un script para llevar a cabo esta tarea y tener un registro de los archivos corruptos.

En el entorno de Ceibal, los datos se encontraban en el directorio `/datalake/land/fing-dns/dns/` del sistema de archivos distribuido (HDFS) con la misma estructura que en el entorno de facultad, los datos no fueron descomprimidos antes de procesarlos.

Un ejemplo de una fila de los datos se muestra en la tabla 4.1.1. A continuación se describe cada una de las columnas[67] de un registro:

timestamp Fecha y hora en la que se realizó la consulta en formato UTC.

mostGranularIdentity Identificador principal del dispositivo desde el que se realizó la consulta.

identities Identificadores principales y secundarios del dispositivo desde el que se realizó la consulta.

internalIP Dirección IP del dispositivo del que se hizo la consulta.

externalIP Dirección IP del *router* que procesó la consulta.

action Acción tomada por *Umbrella* al recibir la consulta, *Allowed*, *Blocked* o *Proxied* dependiendo de si la consulta fue permitida o no.

queryType Tipo de consulta DNS realizada.

responseCode Código de respuesta a la consulta DNS.

domain Dominio consultado.

categories Categorías de contenido asignadas por *Umbrella* correspondientes con el dominio de la consulta.

timestamp	mostGranularIdentity	identities
2019-03-04 18:08:00	1202232-Uv1.0-80-umbrella	1202232-Uv1.0-80-umbrella,1202232

internalIP	externalIP	action	queryType	responseCode
10.33.36.77	200.125.60.174	Allowed	1 (A)	NOERROR

domain	categories
`seguridadtd3.ceibal.edu.uy.`	*Software/Technology,Educational Institutions*

Tabla 4.1.1: Ejemplo de logs de una consulta DNS.

4.1.2. Conjunto de tráfico

Conjunto que contenía registros del tráfico consumido por consultas a DNS de un subconjunto de dispositivos conectados a las redes de Ceibal de todo el país, en el período comprendido entre el 01/09/2019 y el 31/12/2019, en total se contaba con más de cuatrocientos cincuenta millones de datos (467.598.022). Cada dato de este conjunto representaba el tráfico acumulado cada 5 minutos.

Los logs fueron generados por la herramienta *ntopng* instalada sobre la red Ceibal con el fin de monitorizar el tráfico (sección 2.1.3).

En el entorno de Facultad de Ingeniería, los datos se encontraban en el directorio `/opt/datos/datos_NTOP/` del sistema de archivos local que en formato CSV comprimido agrupados por semana con la información separada en columnas. Para realizar el análisis y transformación los datos fueron descomprimidos usando el mismo script comentado anteriormente.

En el entorno de Ceibal, los datos se encontraban en el directorio `/datalake/land/fing-dns/ntop/` del sistema de archivos distribuido (HDFS) con la misma estructura que en el entorno de facultad, los datos no fueron descomprimidos antes de procesarlos.

Un ejemplo de una fila de los datos se muestra en la tabla 4.1.2. A continuación se describe cada una de las columnas:

timestamp Fecha y hora en la que se realizó una consulta en formato UTC.

mac Dirección MAC del *router* desde donde se realizaron las consultas.

application Dominio al cual se realizaron las consultas.

downlink Cantidad de bytes enviados al dominio accedido.

uplink Cantidad de bytes recibidos desde dominio accedido.

ruee Identificador del local asociado a las consultas.

window Indicador en segundos de la ventana temporal usada para realizar la agregación de las consultas.

timestamp	mac	application	downlink	uplink	ruee	window
2019-09-05 19:40:01	04:F1:28:06:B1:AE	Facebook	10867	3617	12101017	300

Tabla 4.1.2: Ejemplo de logs de tráfico de las consultas DNS realizadas en 5 minutos.

4.2. Manejo de datos corruptos

Al inicio del procesamiento de datos se detectaron 5 archivos corruptos en el conjunto de consultas DNS, al intentar leer esos datos, se encontraron filas inconsistentes como se muestra en la tabla 4.2.1.

timestamp	mostGranularIdentity	identities	···
`Not a Time`	$śCg/4	!%_xV	···

Tabla 4.2.1: Ejemplo de datos con filas inconsistentes.

Para tomar una decisión sobre como manejar estos datos, se analizó la cantidad de filas inconsistentes por día, para esto se creó y ejecutó la función `count_corrupted` que cuenta la cantidad de valores faltantes en la columna **timestamp**, los resultados obtenidos se muestran a en el cuadro 4.2.2.

Fecha	Cantidad de filas corruptas
2019-03-14	144905
2019-03-21	240020
2019-04-22	1
2019-04-23	1
2019-03-25	1

Tabla 4.2.2: Cantidad de datos corruptos por día.

Como los datos corruptos representaron el 0.001 % del total, se optó por descartarlos usando la opción `DROPMALFORMED` de *PySpark*, que descarta todas las filas donde existan datos que no pueden ser parseados de forma correcta al leerlos.

Para los datos de tráfico no se detectaron archivos corruptos al aplicar la misma técnica.

4.3. Procesamiento de los datos

En esta sección se detalla el procesamiento de cada columna para llegar a los datos con los que se trabajó en el resto del proyecto.

Notación Para explicar la lógica de procesamiento de algunas columnas se emplea una notación donde el carácter * representa cualquier carácter con 0 o más repeticiones y donde una variable se representa entre corchetes y en mayúscula, por ejemplo, el patrón {**DOMAIN**}.* representa el conjunto de dominios ({**DOMAIN**}) seguidos por un punto (.) y cualquier otro conjunto de caracteres (*).

4.3.1. Conjunto de consultas DNS

timestamp

La columna **timestamp** contenía la información temporal del momento en que se realizó una consulta (en formato *yyyy-mm-dd hh:mm:ss*), esta información era de gran importancia para la meta del proyecto por lo que se decidió separar la información en las columnas **year**, **month**, **day**, **dayofweek** (lunes denotado como 0 y domingo como 6), **hour**, **minute** y **second** para lograr un manejo más granular, de esta forma se pasó de tener una sola variable categórica con un número muy alto de categorías a varias variables con a lo sumo 60 categorías cada una. También se agregó la columna **date** (en formato *yyyy-mm-dd*) para ayuda en otras etapas del procesamiento (4.7).

Como los datos se encontraban en formato horario UTC, se realizó un corrimiento de -3 horas para equipararlos a la hora uruguaya y facilitar su análisis.

Se creó y utilizó la función `process_timestamp` para el procesamiento y en la tabla 4.3.1 se muestra un ejemplo del resultado de aplicarla.

timestamp
2019-04-09 14:16:50

↓

year	month	day	dayofweek	hour	minute	second	date
2019	4	9	1	11	16	50	2019-04-09

Tabla 4.3.1: Ejemplo de procesamiento de la columna **timestamp**.

mostGranularIdentity, identities, externalIP

Las columnas **mostGranularIdentity**, **identities** y **externalIP** contenían información sobre el centro donde se realizó una consulta, la información necesaria para identificar cada consulta a nivel de centro educativo era el ruee (nombre del identificador asignado a los centros por parte cada órgano estatal que los administra) y se encontraba en la columna **mostGranularIdentity** por lo que se extrajo de esa columna (creando una nueva llamada **ruee**). Las dos columnas restantes se descartaron por contener información adicional pero innecesaria para la identificación.

Se creó y utilizó la función `process_ruee`, donde se aplican expresiones regulares sobre la columna cumpliendo la siguiente lógica de reemplazo:

1. Identificador de la forma **TP-**{**RUEE**}**.*** $\longrightarrow$ {**RUEE**}
2. Identificador de la forma {**RUEE**}**-.*** $\longrightarrow$ {**RUEE**}
3. Identificador de la forma {**RUEE**}**,.*** $\longrightarrow$ {**RUEE**}
4. Identificador de la forma {**RUEE**} **.*** $\longrightarrow$ {**RUEE**}
5. Identificador **1108019179.27.20.70** $\longrightarrow$ **1108019**.
6. Identificador **70030589** $\longrightarrow$ **7003058**.
7. Identificador con otra forma queda igual.

Los casos 5 y 6 son casos especiales donde se asignaron identificadores incorrectamente dentro de la herramienta que generó los datos.

Luego de ejecutado el procesamiento se realizó un control de calidad con la función `ruee_safety_checks`, comprobando que todos los identificadores fueran numéricos a excepción de los siguientes:

- **Locales EDGE - 3G - LTE** o **NO BORRAR- Locales EDGE - 3G - LTE** identificadores que representan los servicios EDGE/3G/LTE, para estos casos, por el router y el direccionamiento IP que tienen, es necesario definir esta categoría común para inicializar el servicio, una vez activo, el equipo de soporte de Ceibal lo agrega al sistema asignándole el identificador que corresponda a ese centro.
- **Maqueta Umbrella**, **maqueta-umb-ADv4.1-80-UMBRELLA** o **escuela-produccion** identificadores de locales de prueba.

Todas las consultas realizadas desde estos identificadores se descartaron por representar el 0,004 % de las consultas totales e identificar posiblemente a más de un centro. En la tabla 4.3.2 se muestra un ejemplo del resultado del procesamiento.

mostGranularIdentity	identities	externalIP	⟶	ruee
1111054	1111054	179.27.13.66		1111054

Tabla 4.3.2: Ejemplo de procesamiento de las columnas **mostGranularIdentity**, **identities** y **externalIP**.

queryType

La columna **queryType** identificaba el tipo de consulta realizada, esta columna se normalizó para pasar de contener el tipo y un identificador numérico a solo el tipo de consulta. La columna se renombró a **query_type** para seguir la misma notación usada en el resto.

Se creó y utilizó la función `process_query_type`, donde se aplica una expresión regular cumpliendo la siguiente lógica de reemplazo:

1. *Query type* de la forma {**ID**} **(**{**QUERY_TYPE**}**)** ⟶ {**QUERY_TYPE**}
2. *Query type* con otra forma queda igual.

Luego de procesado se hizo un control de calidad con la función `query_type_safety_checks` comprobando que los resultados pertenecieran a la lista de *query types* válidos[68].

En la tabla 4.3.3 se muestra un ejemplo del resultado del procesamiento.

queryType	⟶	query_type
1 (A)		A
Other		Other

Tabla 4.3.3: Ejemplo de procesamiento de la columna **queryType**

domain

La columna **domain** identificaba el dominio al que se consultó, estos datos contenían URLs que incluían, entre otros ejemplos, parámetros de una consulta y subdominios. Para trabajar con el número más acotado posible de dominios donde a cada uno lo representara un solo valor, se decidió realizar *parsing* y extraer el nombre de dominio de esta columna creando una nueva llamada **parsed_domain**.

Existe un conjunto de dominios que Ceibal consideraba de interés, se decidió usar valores específicos para el parseo de estos dominios, la lista de dominios provista y su respectivo valor parseado se muestran en la tabla 4.3.4.

Dominio de interés	URL	Dominio parseado
CREA	ceibal.schoology.com	CREA
PAM	pam.ceibal.edu.uy	PAM
Matific	www.matific.com	matific
Biblioteca	biblioteca.ceibal.edu.uy	biblioteca
	bibliotecadigital.ceibal.edu.uy	
REA	rea.ceibal.edu.uy	REA
Portal Ceibal	www.ceibal.edu.uy	portal-ceibal
Mi espacio	miespacio.ceibal.edu.uy	miespacio
Ingreso	ingreso.ceibal.edu.uy	ingreso
Portal de Estudiantes	estudiantes.ceibal.edu.uy	portal-estudiantes
Entregas	entregas.ceibal.edu.uy	entregas
Políticas	politicas.ceibal.edu.uy	politicas
Registro biblioteca	registro-biblioteca.ceibal.edu.uy	registro-biblioteca
Google Ceibal	google.ceibal.edu.uy	google-ceibal
Desbloqueo de dispositivos	desbloqueo.ceibal.edu.uy	desbloqueo
Compra de laptops	laptops.ceibal.edu.uy	compra-laptops
Seguimiento de casos	casos.ceibal.edu.uy	seguimiento-casos
Cursos	cursos.ceibal.edu.uy	cursos
Logros	logros.ceibal.edu.uy	logros
CLIC	clic.ceibal.edu.uy	CLIC
Sitio de microbit de Ceibal	microbit.ceibal.edu.uy	microbit-ceibal
Valijas	valijas.ceibal.edu.uy	valijas
Aulas	aulas.ceibal.edu.uy	aulas
Línea de tiempo	lineadetiempo.ceibal.edu.uy	lineadetiempo
Tu clase Uruguay	tuclase.uy	tuclase-uruguay
	tuclase.ceibal.webfactional.com	
Uruguay estudia	uruguayestudia.uy	uruguayestudia
SEA	docentes.sea.edu.uy	SEA
Jóvenes a programar	jovenesaprogramar.edu.uy	jovenesaprogramar
Ibirapitá	ibirapita.org.uy	ibirapita
Programa en DataScience	datascience.edu.uy	datascience
Fundación Ceibal	fundacionceibal.edu.uy	fundacionceibal

Tabla 4.3.4: Lista de dominios de interés de Ceibal, URL y valor asignado al parsear.

También se contaba con otra lista de dominios de interés general que se utilizó para agrupar todos los subdominios que contuvieran uno de estos, esta lista se compone por los dominios:

- youtube
- google
- instagram
- whatsapp
- wikipedia
- mega

- `spotify`
- `netflix`
- `twitter`
- `amazon`
- `facebook`
- `snapchat`

La columna **domain** contenía varios elementos que provocaban una extracción errónea y que debieron ser corregidos, a continuación se listan:

- Cuando se realizó una consulta DNS desde un equipo con sistema operativo Windows se agregó al dominio el sufijo **.ceibal.edu.uy.**, por ejemplo `www.gmail.ceibal.edu.uy`. Para distinguir estos casos de los correctos y poder corregir la clasificación, se usó la lista de dominios de interés y se registró en forma de booleano si un dominio pertenecía a este caso en la columna **dns_from_windows**.
- Dominios contenían signos de interrogación (?), por ejemplo `mundo?ingles.ceibal.edu.uy`.
- Dominios que referenciaban el nombre de dispositivos móviles, por ejemplo `redminote5-sr.k.`, `huawei_y6_2018-5525a8182d`, `iphone-de-lucas`
- Dominios contenían el prefijo **/**, por ejemplo `/ceibal.edu.uy`.
- Dominios contenían el sufijo **.getcacheddhcpresultsforcurrentconfig.**, por ejemplo `instagram.fmvd2-1.fna.fbcdn.net.getcacheddhcpresultsforcurrentconfig`.
- Dominios contenían los caracteres **._**, por ejemplo `_minecraft._tcp.42945`.
- Dominios relacionados a una resolución DNS inversa tenían la forma {**IP**}**.in-addr.arpa**, por ejemplo `120.38.239.216.in-addr.arpa`.

Se creó y utilizó la función `process_domain`, donde se aplican varias expresiones regulares encadenadas cumpliendo la siguiente lógica:

1. Dominio de la forma **/**{**DOMAIN**} $\longrightarrow$ {**DOMAIN**}
2. Dominio de la forma **d?oma?in** $\longrightarrow$ {**DOMAIN**}
3. Dominio de la forma **_minecraft.*** $\longrightarrow$ **minecraft**
4. Dominio de la forma ***._tcp.**{**DOMAIN**} $\longrightarrow$ {**DOMAIN**}
5. Dominio de la forma ***._udp.**{**DOMAIN**} $\longrightarrow$ {**DOMAIN**}
6. Dominio de la forma ***._msfcs.**{**DOMAIN**} $\longrightarrow$ {**DOMAIN**}
7. Dominio de la forma **._**{**DOMAIN**} $\longrightarrow$ {**DOMAIN**}
8. Dominio de la forma **.**{**DOMAIN**} $\longrightarrow$ {**DOMAIN**}
9. Dominio de la forma **www.**{**DOMAIN**} $\longrightarrow$ {**DOMAIN**}
10. Dominio de la forma ***.www.**{**DOMAIN**} $\longrightarrow$ {**DOMAIN**}
11. Dominio de la forma {**DOMAIN**}**.ceibal.edu.uy.*** $\longrightarrow$ {**DOMAIN**}**.**
12. Dominio de la forma ***.bibliotecadigital.ceibal.edu.uy** $\longrightarrow$ **bibliotecadigital**
13. Dominio de la forma {**DOMAIN**}**.getcacheddhcpresultsforcurrentconfig** $\longrightarrow$ {**DOMAIN**}
14. Dominio que contiene un dominio de interés $\longrightarrow$ **dominio_de_interes**, en este caso, si contiene más de un dominio de interés se elige el primero según el orden en la lista

15. Dominio de la forma {**DOMAIN**}/{**QUERY**} ⟶ {**DOMAIN**}
16. Dominio que identifica un dispositivo móvil **android** (con nombre android, samsung, galaxy, redmi o huawei) ⟶ **android-phone**
17. Dominio que identifica un dispositivo móvil **iphone** ⟶ **iphone-phone**
18. Dominio de la forma {**IP**}**.in-addr.arpa** ⟶ **arpa**
19. Dominio de la forma {**DOMAIN**}**.** ⟶ {**DOMAIN**}

Luego de estos pasos se realizó el *parsing* sobre los dominios pre-procesados para extraer el *generic top-level domain* usando la lisa pública de sufijos. En la tabla 4.3.5 se muestra un ejemplo del resultado del procesamiento.

domain
brother.
_minecraft._tcp.42945.
www.juegos.
/ceibal.
www.google.ceibal.edu.uy.
youtube.com.ceibal.edu.uy.
180.com.uy.
android-cf46f99d82ced781.
redminote5-sr.k.
iphone-de-lucas.
cursos.ceibal.edu.uy
xx.xx.xx.xx.in-addr.arpa.

↓

parsed_domain	dns_from_windows	ceibal_domain	ceibal_category
brother	False	False	sin_categoria
minecraft	False	False	sin_categoria
juegos	False	False	sin_categoria
ceibal	False	False	sin_categoria
google-ceibal	False	True	plataforma_ceibal
youtube	True	False	redes_sociales
180	True	False	sin_categoria
android-phone	False	False	sin_categoria
android-phone	False	False	sin_categoria
iphone-phone	False	False	sin_categoria
cursos	False	True	plataforma_ceibal
arpa	False	False	sin_categoria

Tabla 4.3.5: Ejemplo de procesamiento de la columna **domain**.

categories

La columna **categories** indicaba la categorización de cada dominio aplicada por *Umbrella*, cada registro contenía una lista separada por comas de categorías que lo identifican (una lista exhaustiva se encuentra en [69]).

Como ya se mencionó en la anteriormente, las consultas DNS realizadas desde dispositivos Windows agregan el sufijo `.ceibal.edu.uy.`, esto provocaba una mala clasificación aumentando artificialmente la cantidad de logs etiquetados como *Software/Technology,Educational Institutions*, para tratar esta situación se usó la información de la columna **dns_from_windows**, eliminando la clasificación de los registros que esa columna indicara.

Luego de analizar el contenido de la columna (en la sección 4.11.1), se decidió mantener las primeras dos categorías consideradas por *Umbrella*, generando dos nuevas columnas: **category_1** y **category_2**. También se decidió reordenar las categorías de forma tal que *Search Engines*, *Software/Technology* y *Application* estuviesen siempre al final de la lista, esto debido a que estas categorías podrían aparecer delante de otras más específicas y que resultasen de mayor utilidad.

Se detectaron algunos dominios pertenecientes a servicios de infraestructura que no tenían clasificación, estos dominios se re-clasificaron con la categoría *Infrastructure* y son:

- `notificaciones`
- `local`
- `cisco-capwap-controller`

Al analizar los posibles valores que podían tomar las categorías, se descubrió que algunas podrían ser agrupadas, a continuación se listan los agrupamientos realizados:

- Si las categorías *Chat*, *Instant Messaging*, *Social Networking*, *Photo Sharing* o *Blogs* están en la clasificación o el dominio es `tiktok` $\longrightarrow$ se indica que la categoría principal es *Social Networkig*
- Si las categorías *Video Sharing*, *Movies* o *Radio* están en la clasificación $\longrightarrow$ se indica que la categoría principal es *Streaming*

Debido a que se detectaron casos donde algunas categorías diferían de las esperadas por el uso de mayúsculas (esto por el uso de distintas versiones de la herramienta usada para generar los datos), se decidió normalizarlas para llevarlas a un formato uniforme, el proceso de normalización consistió en:

1. Se reemplazan las mayúsculas por minúsculas.
2. Se reemplazan los espacios por el carácter guión bajo (_).
3. Se reemplaza el carácter / por los caracteres _or_.

Se utilizó la función `process_categories` para el procesamiento y un ejemplo del resultado de aplicarlo se muestra en la tabla 4.3.6.

Para completar la información faltante, se realizó un proceso de imputación de datos donde para cada dominio se eligió el conjunto de categorías más representativo (la decisión tomada en este punto fue seleccionar para cada columna el valor que más veces se repitiese entre todas las consultas) o en caso de no existir dicho conjunto se usó el valor **sin_categoria**, para este proceso se utilizó la función `categories_imputation` y un ejemplo del resultado se muestra en la tabla 4.3.7, los resultados de este proceso también se muestran en la sección 4.11.1.

categories	parsed_domain	dns_from_windows
Chat,Instant Messaging,Search Engines,Application	whatsapp	False
Software/Technology,Educational Institutions	notificaciones	True
Blogs,Social Networking	twitter	False

↓

categories	category_1	category_2	category_3
[social_networking, search_engines, application]	social_networking	search_engines	application
[infrastructure]	infrastructure		
[social_networking, blogs]	social_networking	blogs	

Tabla 4.3.6: Ejemplo de procesamiento de la columna **categories**

parsed_domain	category_1	category_2	category_3
google	search_engines		
facebook	social_networking	application	
local			

↓

parsed_domain	category_1	category_2	category_3
google	search_engines	software_or_technology	application
facebook	social_networking	application	saas_and_b2b
local	sin_categoria	sin_categoria	sin_categoria

Tabla 4.3.7: Ejemplo de imputación de datos en las columnas **category_1**, **category_2** y **category_3**

4.3.2. Conjunto de tráfico

timestamp

La columna **timestamp** contenía la información temporal del momento en que se registró el tráfico (en formato *yyyy-mm-dd hh:mm:ss*), al igual que con las consultas DNS, se decidió separar la información en las columnas **year**, **month**, **day**, **hour** y **minute** para un manejo más granular, se descartó crear una columna de segundos porque dentro de una misma ventana temporal de registro se notaron desfasajes producto del tiempo de procesado de la herramienta utilizada para la creación de los registros. También se agregó la columna **date** (en formato *yyyy-mm-dd*) para ayuda en etapas posteriores del procesamiento.

Como los datos se encontraban en formato horario UTC, se realizó un corrimiento de -3 horas para equipararlos a la hora uruguaya y facilitar su análisis.

Se utilizó la función ***process_timestamp***. En la tabla 4.3.8 se muestra un ejemplo del resultado del procesamiento.

timestamp	→	year	month	day	hour	minute	date
2019-11-09 14:10:01		2019	4	9	11	10	2019-04-09

Tabla 4.3.8: Ejemplo de procesamiento de la columna **timestamp**.

application

La columna **application** identificaba el dominio para el cual se registró el tráfico, esta columna se normalizó para evitar errores de codificación en las siguientes etapas.

Se utilizó la función `process_application`, donde se se aplica el proceso de normalización que consiste en:

1. Se reemplazan las mayúsculas por minúsculas.
2. Se reemplazan los espacios por el carácter guión bajo (_).
3. Se remueven caracteres especiales (,;()=).

En la tabla 4.3.9 se muestra un ejemplo del resultado del procesamiento.

application		application
SSL No cert	⟶	ssl_no_cert
Youtube		youtube

Tabla 4.3.9: Ejemplo de procesamiento de la columna **application**

uplink, downlink

Las columnas **uplink** y **downlink** se utilizaban para registrar el tráfico parcial de cada conjunto de consultas, *uplink* registraba la cantidad de bytes enviados en la consulta a un dominio y *downlink* la cantidad de bytes recibidos, estas columnas se sumaron para obtener la columna **traffic** que registraba el tráfico total.

4.4. Información adicional agregada

Al conjunto de datos se le agregó información adicional considerada de ayuda para la resolución de los problemas, en esta sección se describe esa información, su generación y su procesamiento.

4.4.1. Información geoespacial y socioeconómica

Ceibal brindó en dos instancias información geoespacial y socioeconómica respecto a cada centro educativo, estos datos se encontraban en los archivos `tabla_locales_v1.csv` y `tabla_locales_v2.csv` del directorio de datos como se detalla en 3. Las columnas dentro de estos archivos se describen a continuación:

Tabla **tabla_locales_v1**:

ruee Identificador del centro desde donde se realizó una consulta.

departamento Departamento al que pertenece el local.

localidad Localidad del departamento al que pertenece el local.

subsistema Subsistema educativo al que pertenece el local (por ejemplo: CEIP, CES, Universidad).

contexto_sociocultural Quintil al que corresponde el local.

Tabla **tabla_locales_v2**:

num_local Identificador del centro desde donde se realizó una consulta.

razon_social Nombre por el que se identifica el centro.

latitud/longitud Coordenadas geográficas del local.

departamento Departamento al que pertenece el local.

localidad Localidad del departamento al que pertenece el local.

zona Zona a la que pertenece el local (Rural o Urbana).

subsistema Subsistema educativo al que pertenece el local (por ejemplo: CEIP, CES, Universidad).

desc_tipo_centro_depend Descripción de tipo de dependencia del local (por ejemplo: Escuela Privada, Dependencia Administrativa, Utu).

num_aps_cisco Información extra sobre Umbrella.

Estas tablas se procesaron para eliminar información incorrecta y completar datos faltantes, el proceso se encuentra en el *notebook* `geoespatial_analysis.ipynb` y se detalla a continuación.

Normalización

Para normalizar los valores de las tablas se aplicaron los siguientes pasos:

1. Se reemplazan las mayúsculas por minúsculas.
2. Se reemplazan los espacios por el carácter guión bajo (_).
3. Se eliminan los acentos.

Procesamiento de tabla_locales_v1

Al momento de cargar los datos de esta tabla se realizó un primer procesado que consiste en:

1. Reemplazar mayúsculas por minúsculas en los nombres de las columnas.
2. Eliminar locales con ruees duplicados.
3. Eliminar locales con ruees faltantes.
4. Marcar como valor faltante el valor 'Sin Dato' y el valor 'Sin clasificar'.
5. Normalizar (4.4.1) los valores de las columnas **departamento**, **localidad** y **subsistema**.

La columna **contexto_sociocultural** contenía información conjunta del quintil y la zona a la que pertenece un local, por esto se desagregó la información reduciendo los posibles valores. Las nuevas columnas formadas fueron **quintil** que tomaba valores enteros entre 1 y 5 y **zona_quintil** que tomaba los valores *urbana* o *rural*. La columna original se descartó.

Se buscaron locales con ruees no numéricos encontrando los siguientes:

- 13017651-P
- 1302xxxx
- 1202xxxx

Se descartaron estos locales por no contener mayor información identificatoria.

Procesamiento de locales_v2

Al momento de cargar los datos de esta tabla se realizó un primer procesado que consistió en:

1. Reemplazar mayúsculas por minúsculas en los nombres de las columnas.
2. Renombrar las columnas **num_local** por **ruee**, **desc_tipo_centro_depend** por **tipo_centro** y **zona** por **zona_quintil**.
3. Descartar la columna **num_aps_cisco** por no aportar información relevante.
4. Extraer ruee de la columna **razon_social** para locales con ruees faltantes.
5. Eliminar locales con ruees duplicados.
6. Eliminar locales con ruees faltantes.
7. Marcar como valor faltante el valor 'Sin Dato'.
8. Normalizar (4.4.1) los valores de las columnas **departamento**, **localidad**, **subsistema** y **tipo_centro**.

Se buscaron locales con ruees no numéricos encontrando los siguientes:

- 13017651-P
- 1302xxxx
- 1301ZZZZ
- 1218XXX
- 1101362 (6170)
- _x_x_x_x
- 1309XXXX
- 1201XXXX
- 1301XXXX

Se descartan estos locales por no aparecer en los datos de logs.

A partir de los mapas de SIGANEP (que cuentan con el ruee de los locales), se actualizaron los datos de **longitud** y **latitud**, seguido de esto se normalizaron las coordenadas y se graficaron en busca de *outliers*, en la figura 4.4.1 se muestran los *boxplots* realizados.

Para la latitud se observaron valores por fuera del rango de coordenadas donde se encuentra Uruguay (latitud > -33), se analizaron estos valores y se encontró que todos los centros a los que pertenecen están fuera del país, por lo tanto, no se hicieron más procesamientos en esta variable. Para el caso de la longitud se repitió el mismo proceso y se llegó a resultados iguales. En la figura 4.4.2 se muestran los *boxplots* de las coordenadas reducidas a los valores de Uruguay.

Procesamiento de datos en conjunto

Una vez se combinaron las tablas, se verificó que no se existiesen datos inconsistentes pero se encontró en la columna **localidad**, identificadores diferentes representando el mismo local, en la tabla 4.4.1 se muestran los identificadores de ambas tablas que representan la misma localidad, se procedió a usar los identificadores provenientes de la tabla **tabla_locales_v1**.

En la tabla 4.4.2 se muestra la cantidad de valores faltantes para cada una de las columnas de la tabla resultante, se observó un alto porcentaje de valores faltantes en las columnas que contenían la información socio-económica (**quintil** y **zona_quintil**) y también en las columnas que contenían las coordenadas geográficas.

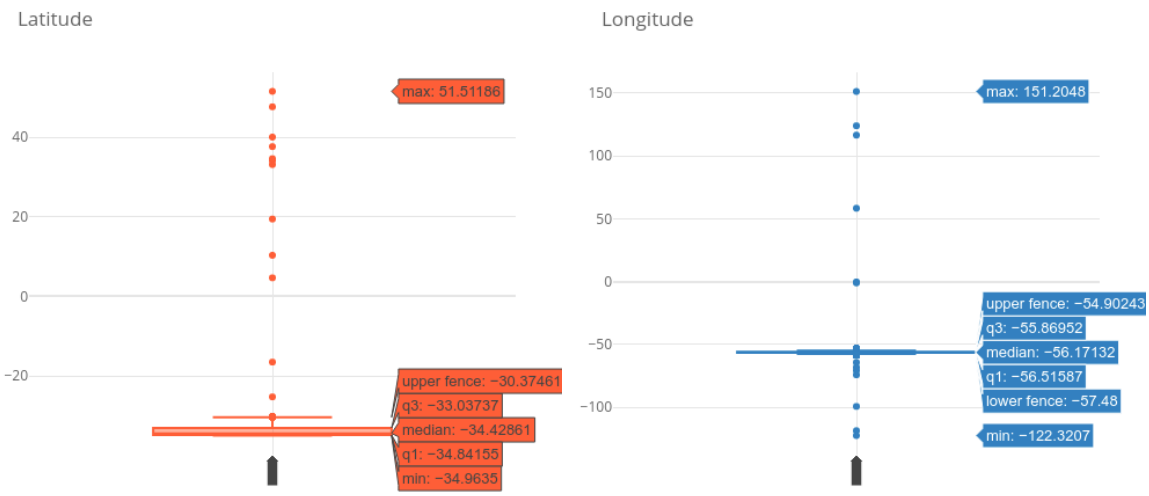


Figura 4.4.1: *Boxplots* de latitud y longitud antes de procesarlas.

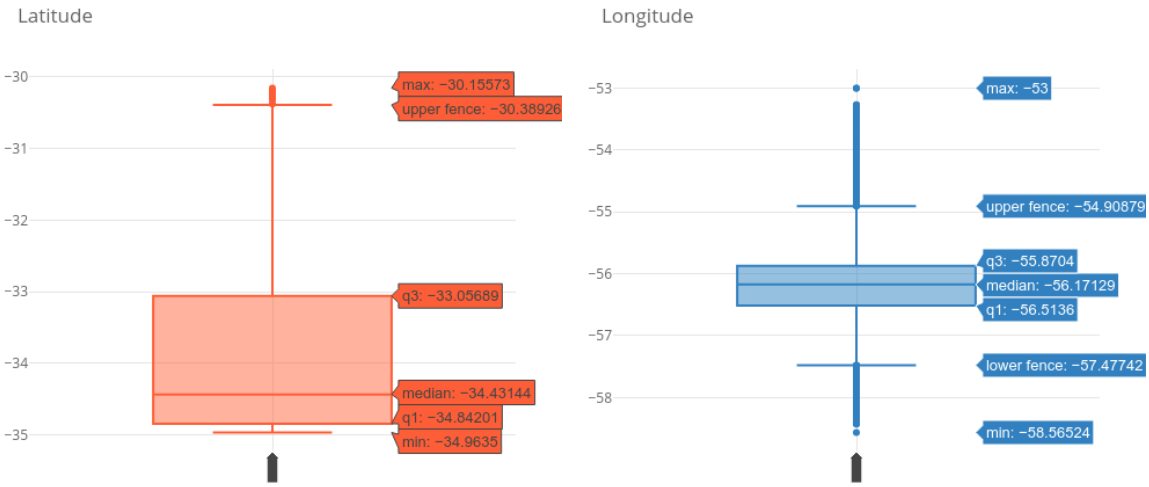


Figura 4.4.2: Boxplots de latitud y longitud luego de procesarlas.

En la tabla 4.4.3 se repitió el conteo pero considerando solo al conjunto de los ruees usados en el conjunto de datos de logs, como el porcentaje de valores faltantes continuó siendo alto (más de 30 % en algunos casos) y todos los datos se consideraron necesarios para el modelo de predicción, se decidió aplicar una técnica de imputación de datos; además se decidió aplicar una técnica de validación de los datos para asegurar su correctitud.

Imputación y validación de los datos

Para la asegurar tanto la completitud como la correctitud de los datos, se creó una aplicación web que muestra los datos de cada centro de forma visual y permite corregir y agregar los datos de cada local usando otros de referencia.

En la figura 4.4.3 se muestra la interfaz de la aplicación creada y se indican las secciones que corresponden a cada uno de los pasos necesarios para realizar el análisis y corrección de los datos, a continuación se explican esos pasos:

Localidad tabla_locales_v1	Localidad tabla_locales_v2
paso_carrasco	paso_de_carrasco
villa_rodriguez	rodriguez

Tabla 4.4.1: Identificadores inconsistentes detectados durante la combinación de las tablas.

Columna	Cantidad de valores faltantes	Porcentaje de valores faltantes
ruee	0	0
razon_social	0	0
departamento	136	1.6
localidad	180	2.2
zona_quintil	5977	71.6
subsistema	0	0
tipo_centro	0	0
quintil	6043	72.4
latitud	1890	22.7
longitud	1890	22.7

Tabla 4.4.2: Cantidad de valores faltantes por columna de la tabla combinada.

1. **Selección de un centro** se selecciona un centro según departamento y localidad, los departamentos están ordenados en forma creciente según la cantidad de centros.
2. **Localización y comparación con otros** se muestra la ubicación del centro seleccionado según sus coordenadas y se compara con centros cercanos para corroborar que la ubicación sea la correcta.
3. **Modificación de los datos** se modifican los datos del centro seleccionado según sea necesario.

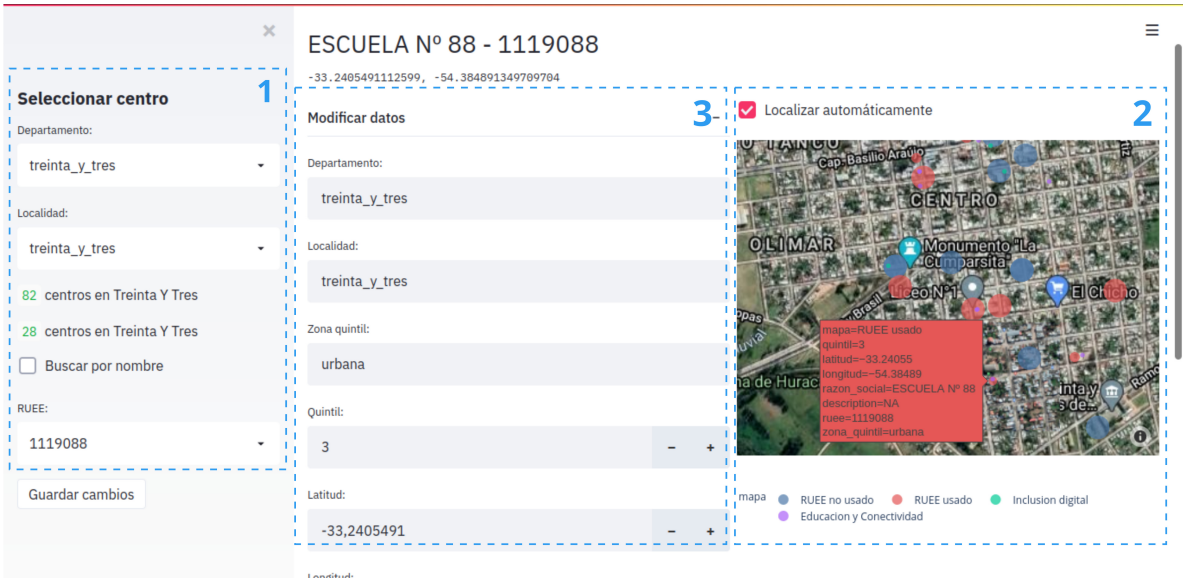


Figura 4.4.3: Interfaz de la aplicación creada para la imputación de datos de locales.

Columna	Cantidad de valores faltantes	Porcentaje de valores faltantes
ruee	0	0
razon_social	0	0
departamento	0	0
localidad	0	0
zona_quintil	840	30.3
subsistema	0	0
tipo_centro	0	0
quintil	883	31.9
latitud	40	1.4
longitud	40	1.4

Tabla 4.4.3: Cantidad de valores faltantes por columna para el conjunto de ruees utilizados.

Para indicar en el mapa los locales cercanos al seleccionado en la aplicación, se descargaron varios mapas de referencia y se listan a continuación:

- **Consejo de Educación Inicial y Primaria** extraído del Sistema de Información Geográfica de ANEP (**SIGANEP**), se considera de confiabilidad alta.
- **Consejo de Educación Secundaria** extraído del Sistema de Información Geográfica de ANEP (**SIGANEP**), se considera de confiabilidad alta.
- **Consejo de Educación Técnico profesional** extraído del Sistema de Información Geográfica de ANEP (**SIGANEP**), se considera de confiabilidad alta.
- **Consejo de Formación en Educación** extraído del Sistema de Información Geográfica de ANEP (**SIGANEP**), se considera de confiabilidad alta.
- **Educación y Conectividad** extraído de Google Maps, cuenta con información de instalación de antenas de ANTEL en diversos locales, se considera de confiabilidad baja y se usa de referencia.
- **Inclusión digital** extraído de Google Maps, cuenta con información de locales del MEC y escuelas, se considera de confiabilidad baja y se usa de referencia.

Durante esta etapa se agregó una nueva columna denominada **nivel_fiabilidad** donde quedó registrada la confianza que se tenía en un registro, esta columna tenía tres posibles valores:

- Nivel de fiabilidad **alto**: significa que los datos fueron analizados y posiblemente corregidos, además se sabe que son correctos y no existe ningún valor faltante en ese registro.
- Nivel de fiabilidad **medio**: significa que los datos fueron analizados y posiblemente corregidos pero existen valores faltantes en ese registro.
- Nivel de fiabilidad **bajo**: significa que los datos no fueron analizados ni corregidos.

Una vez que terminado el proceso de corrección de datos, se realizó la imputación de los valores faltantes usando la función `make_geospatial_imputation_data` que sigue la siguiente lógica:

1. Se calcula la distancia geodésica de los centros donde faltan datos al resto.
2. Se hacen barridos de a 100m iniciando en 100m hasta 25km, si no se encuentran centros en el barrido se continúa hasta encontrar.

3. Con los centros encontrados en un barrido, se calcula la moda del **quintil** y la **zona** y se completa la información, también se guarda en la columna **distancia_imputacion**, la información de la distancia usada para hacer la imputación.

En la figura 4.4.4 se muestra el porcentaje de datos imputados según distancia de imputación usada, se observa que el 97.8 % de los centros fueron imputados en una distancia menor o igual a 3km, por esto, se decidió que los centros imputados con datos a más de 3km se descarte esta información. Los datos faltantes se completaron con el valor *sin_dato* para la columna **zona** y -1 para la columna **quintil**.

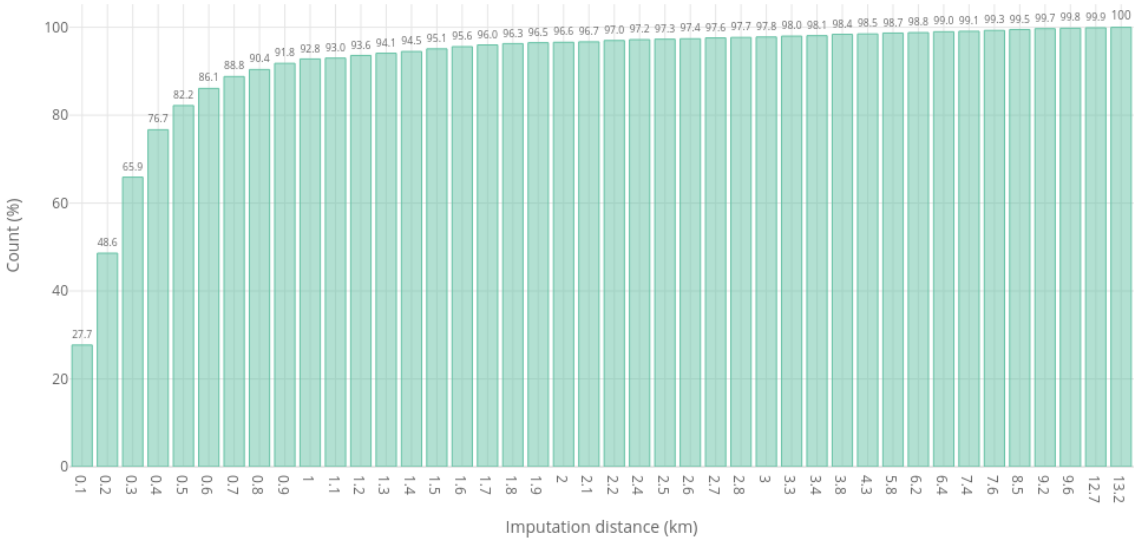


Figura 4.4.4: Cantidad de datos imputados según la distancia de imputación.

Luego de la imputación se descartaron las columnas **razon_social** y **distancia_imputacion** porque solo fueron relevantes para la búsqueda de datos faltantes.

Se utilizó la función `geospatial_imputation`, donde a través de la operación *merge* entre tablas se agregó la información antes descrita, se decidió eliminar las consultas provenientes de locales sin información. En el cuadro 4.4.4 se muestra un ejemplo del resultado de este proceso.

ruee
1101297

↓

ruee	departamento	localidad	subsistema	tipo_centro
1101297	montevideo	casabo	ceip	jardin_publico

zona_quintil	quintil	latitud	longitud
urbana	1	-34.88559	-56.27426

Tabla 4.4.4: Ejemplo de adición de información geoespacial y sociocultural.

4.4.2. Datos de variabilidad temporal

También se agregó a las columnas que presentan valores numéricos una columna extra con el registro de la variabilidad temporal durante un período anterior, para cada columna **C**, la columna nueva se nombra como **C_EWMA**. El cálculo de la variabilidad se realizó usando el promedio móvil exponencial ponderado (EWMA). La ventana temporal representa la cantidad de tiempo en el que se observa la variación y se elige dependiendo de la frecuencia temporal de los datos de modo que t represente una hora, por ejemplo, para datos con frecuencia de 5 minutos, la ventana temporal debería ser $t = 12$.

4.5. Codificación de variables

Los modelos de predicción solo pueden trabajar con variables de tipo numérico, por lo tanto todas las variables de tipo categórico deben ser codificadas. Este proceso (llamado *encoding*) confiere a cada valor posible de una variable un identificador entero, este mapeo se puede hacer en forma dinámica al momento de realizar el entrenamiento o en la etapa de pre-procesamiento guardando los datos transformados en disco, en este proyecto se decidió seguir el segundo camino.

El proceso de codificación para una variable consiste en asignar a cada valor posible un identificador i único tal que $1 \leq i \leq n$ con n igual a la cantidad de valores posibles.

Se usó la función `encode_features` y en la tabla 4.5.1 se muestra un ejemplo del resultado del procesamiento para la columna **departamento**.

departamento		parsed_domain
florida		1
montevideo	$\longrightarrow$	3
flores		2
montevideo		3

Tabla 4.5.1: Ejemplo de *encoding* de la columna **departamento**

Al usar el método antes explicado los valores de una columna se vuelven difusos, para el caso de las columnas que representan datos temporales es conveniente entender la codificación rápidamente para su análisis, por este motivo se creó una codificación especifica que se explica en la tabla 4.5.2.

Columna	Valor codificado
year	year - 2019
month	month
day	day
dayofweek	dayofweek + 1
hour	hour + 1
minute	minute + 1
second	second + 1

Tabla 4.5.2: *Encoding* de las columnas que representan datos temporales

4.6. Agregación y remuestreo de datos

Los datos de consultas DNS y tráfico presentaban distintas frecuencias temporales (1 segundo y 5 minutos respectivamente), esto llevó a la búsqueda de algún tipo de agregación para reducir la cantidad de consultas sin perder información importante.

Varias agregaciones fueron realizadas buscando la mejor para el problema a resolver, a continuación se explica el método general aplicado y luego se comentan los puntos claves de cada agregación realizada.

Para realizar una agregación se elige un conjunto de **columnas clave** que se usan para crear los grupos de agregación y un conjunto de **columnas agregadas**. A partir de una columna agregada **C** que tiene valores $c_0, \ldots, c_n$, se crean las columnas **C**_$c_0, \ldots,$**C**_c_n donde cada valor de C_c_i representa el valor agregado de c_i, la función de agregación utilizada es específica del conjunto de datos, para el conjunto de datos de consultas DNS se cuenta la cantidad de consultas por cada valor y, para el conjunto de datos de tráfico, se usa el tráfico total por cada valor. Los valores c_i usados para crear las nuevas columnas pueden representar un subconjunto de todos los valores posibles de la columna C, en este caso, se agrega la columna **C_OTHER** que representa la agregación de todos los valores no considerados. Se agrega la columna **TOTAL** que representa el valor total por fila.

Frente a una agregación usando columnas que representan datos temporales, el muestreo final de los datos será igual al representado por la columna temporal de mayor granularidad, por ejemplo, si se agrega usando las columnas **date**, **hour** y **minute**, el conjunto resultante representará los valores para cada minuto de cada hora de cada día. Para poder tener muestreos intermedios, por ejemplo, cada 5 minutos, se debe re-muestrear la columna **minute** de forma que los valores 0 a 4 se mapeen al valor 5, los valores 5 a 9 se mapeen al valor 10, etc. De esta forma, al realizar nuevamente la agregación del ejemplo anterior, el conjunto resultante representará los valores cada 5 minutos para cada hora de cada día. Este proceso se realiza con la función `resample_col` y se aplica la siguiente fórmula:

$$\text{resample}(val, freq) = val + freq - (val \mod freq)$$

En la tabla 4.6.1 se muestra un ejemplo de re-muestreo de la columna **minute**.

minute		minute
1		5
2		5
5		10
19		20
15	$\longrightarrow$	20
42		45
31		35
47		50
22		25
50		55

Tabla 4.6.1: Ejemplo de re-muestreo de la columna **minute** a 5 minutos

4.6.1. Agregación temporal por ruee

En esta agregación temporal se usaron como columnas clave las columnas que representan datos temporales junto con la columna **ruee**. Es decir que se obtuvo una agregación donde se resumió la actividad de un centro en un período temporal.

Para el período de tiempo que representa la agregación temporal se probaron varios valores buscando reducir la cantidad de datos sin perder la calidad de la predicción, los períodos probados fueron 5, 10 y 15 minutos.

En la tabla 4.6.2 se muestra un ejemplo del índice resultante para el período temporal de 15 minutos.

date	hour	minute	ruee
2019-05-05	0	15	1001934
			1101006
			...
		30	1101012

Tabla 4.6.2: Ejemplo de índice resultante del proceso de agregación de datos para el período temporal de 15 minutos, en la agregación temporal por ruee.

4.6.2. Agregación temporal por departamento y subsistema

En esta agregación temporal se usaron como columnas clave las columnas que representan datos temporales junto con las columnas **departamento** y **subsistema**. Es decir que se obtuvo una agregación donde se resumió la actividad de todos los locales de cada subsistema dentro de cada departamento en un período temporal.

Para el período de tiempo que representa la agregación temporal se uso el período de 15 minutos y en la tabla 4.6.3 se muestra un ejemplo del índice resultante.

date	hour	minute	departamento	subsistema
2019-05-05	0	15	Montevideo	CEIP
				CES
			...	...
		30	Florida	CEIP

Tabla 4.6.3: Ejemplo de índice resultante del proceso de agregación de datos para el período temporal de 15 minutos, en la agregación temporal por departamento y subsistema.

4.6.3. Agregación temporal por subsistema

En esta agregación temporal se usaron como columnas clave las columnas que representan datos temporales junto con la columna **subsistema**. Es decir que se obtuvo una agregación donde se resumió la actividad de todos los locales de cada subsistema en un período temporal.

Para el período de tiempo que representa la agregación temporal se uso el período de 15 minutos y en la tabla 4.6.4 se muestra un ejemplo del índice resultante.

date	hour	minute	subsistema
2019-05-05	0	15	Universidad
			CEIP
			...
		30	CES

Tabla 4.6.4: Ejemplo de índice resultante del proceso de agregación de datos para el período temporal de 15 minutos, en la agregación temporal por subsistema.

4.7. Predicción

Para la predicción de la cantidad de consultas DNS, a cada dato puntual se le agregó la cantidad total de consultas en el siguiente momento como etiqueta a predecir.

Para la transformación de consultas DNS en tráfico, a cada dato puntual se le agregó el tráfico asociado al total de consultas DNS de ese momento como etiqueta a predecir.

4.8. Particionado

Una vez preparados los datos y las predicciones, se dividió el conjunto de datos en tres subconjuntos:

1. ***Train*** conjunto usado para el entrenamiento de un modelo, el modelo ajusta sus parámetros a partir de este conjunto.
2. ***Val*** conjunto usado para comprobar el rendimiento del modelo durante su entrenamiento y tomar acciones.
3. ***Test*** conjunto usado para comprobar el rendimiento del modelo después del entrenamiento, este conjunto se usa para determinar la capacidad de generalización de un modelo con datos que no usó antes.

El tamaño de cada conjunto se fijó al momento de realizar una partición, normalmente el conjunto de entrenamiento (*train*) contiene gran parte de los datos para que el modelo pueda aprender de una muestra lo suficientemente variada del espacio de posibilidades, por esto, para este proyecto se decidió que este conjunto contendría el 70 % de los datos mientras que cada uno de los otros contendría 15 %.

Las columnas por las que se realizó la partición determinaron la forma en que se dividieron los datos ya que los conjuntos generados son disjuntos. Por ejemplo, si se decide realizar la partición usando las columnas **date** y **hour**, el 75 % de los valores que puede tomar la columna **hour** siempre pertenecerá al conjunto de entrenamiento mientras que el restante 30 % se dividirá en los otros, los datos de cada día estarán en cada conjunto pero solo en parte, este ejemplo se muestra en la tabla 4.8.1. En cambio, si se particiona usando la columna **date**, todos los valores de un día seleccionado para un conjunto se encontrarán en ese conjunto y no en el resto.

Para evitar que comportamientos irregulares aparecieran solo en un subconjunto (por ejemplo, períodos de vacaciones), se agregó la opción de hacer la selección de valores de forma aleatoria, retomando el ejemplo anterior, si se usa esta opción, un conjunto podría tener ejemplos de varios días no consecutivos.

4.9. Normalización

Aplicar distintas técnicas de normalización a los datos tiene un impacto en el rendimiento y el tiempo de entrenamiento de los modelos de redes neuronales.[70] En este proyecto se optó por escalar las distintas variables

date	hour	date	hour	date	hour
2019-05-05	1	2019-05-05	19	2019-05-05	22
	...		20		23
	18		21		24
2019-05-06	1	2019-05-06	19	2019-05-06	22
	...		20		23
	18		21		24

Tabla 4.8.1: Ejemplo de particionado usando las columnas **date** y **hour** conjuntos *train*, *val* y *test* respectivamente.

numéricas con dominio continuo al rango $[0, 1]$, este proceso consta en obtener los valores máximo y mínimo de una variable y aplicar la siguiente formula:

$$\text{Normalización}(X) = \frac{X - X_{min}}{X_{max} - X_{min}}$$

Es importante que los valores usados para la normalización se obtengan del subconjunto de datos usado para entrenar el modelo y no del conjunto total, de esta forma la evaluación de la capacidad de generalización del modelo es más confiable ya que no se introduce información de los conjuntos de validación en el de entrenamiento. Para llevar a cabo este proceso de forma correcta, se obtuvieron los valores estadísticos durante la etapa posterior a la generación de las particiones (sección 4.8) y se usaron aplicando este proceso durante la ingesta de los datos.

4.10. Pipeline

Una vez definidas y probadas todas las funciones a aplicar sobre los datos para su procesamiento, se decidió realizar un pipeline definido en varias etapas explicadas a continuación, esta decisión se dio por dos motivos, el primero, la posibilidad de realizar comprobaciones intermedias para validar los datos procesados y el segundo, para evitar tener que volver a ejecutar pasos con muy alto tiempo de ejecución (días, concentrados en la etapa 1) cada vez que se quería generar un conjunto final.

En todas las etapas se cargan y procesan los datos de forma iterativa de a un día por vez. Cada etapa se implementó usando una clase base `StageExecutor` ejecuta cada etapa siguiendo la metodología ETL (sección 2.2.2) dividida en tres pasos:

1. Cargar datos en memoria (*extract*)
2. Procesar datos (*transform*)
3. Persistir datos (*load*)

Los datos de las etapas se almacenan en *HDFS* en formato ORC (*Optimized Row Columnar*), este formato fue diseñado para superar limitaciones de otros formatos de *Hive* y provee una forma altamente eficiente de almacenar y leer datos.

4.10.1. Conjunto de tráfico

Etapa 1

Ubicación datos de entrada

Ambiente Facultad de ingeniería: `/opt/datos/datos_NTOP/` en el sistema de archivos local

Formato de lectura CSV

Ubicación datos de salida tabla `etapa_1` en base de datos `ntop` en *Hive*.

Formato de escritura ORC

En esta etapa, se lleva a cabo todo el **procesamiento de columnas** (4.3.2, 4.3.2, 4.3.2). Los datos se encuentran comprimidos por semana, por esto se procesan con esa granularidad, distribuyendo los datos en el cluster y luego, se agregan a la tabla correspondiente particionando por fecha y hora.

Se descarta la columna **window** pues siempre tiene le mismo valor.

Las columnas al final de esta etapa son los siguientes:

timestamp Fecha y hora en la que se realizó una consulta en UTC-3.

mac Dirección MAC del *router* desde donde se realizaron las consultas.

application Dominio al cual se realizaron las consultas.

downlink Cantidad de bytes enviados al dominio accedido.

uplink Cantidad de bytes recibidos desde dominio accedido.

traffic Cantidad de bytes total transmitidos en las consultas con el dominio accedido.

ruee Identificador del local asociado a las consultas.

year Año en que se hizo la consulta.

month Mes en que se hizo la consulta.

day Día del mes en que se hizo la consulta.

dayofweek Día de la semana en que se hizo la consulta.

hour Hora en que se hizo la consulta.

minute Minuto en que se hizo la consulta.

La figura 4.10.1 es una representación esquemática de esta etapa.

Etapa 2

Ubicación datos de entrada tabla `etapa_1` en base de datos `ntop` en *Hive*.

Formato de lectura ORC

Ubicación datos de entrada tabla `etapa_2[_rs]_sf_{SAMPLE_FREQUENCY}_agg_{AGGREGRATION_COLUMNS}` en base de datos `ntop` en *Hive*.

Formato de lectura ORC

En esta etapa, antes de iniciar el procesamiento por día se selecciona una frecuencia de muestreo temporal (SAMPLE_FREQUENCY), las columnas clave que se usarán en la agregación (AGGREGATION_COLUMNS) (4.6) y los valores a usar en caso de que se quiera seleccionar un subconjunto de los valores posibles para la agregación de una columna, en este caso se agrega el prefijo `\_rs` al nombre de la tabla. También se crean las codificaciones a usar a partir de todo el conjunto de datos (4.5). Luego, el procesamiento iterativo consiste en re-muestrear los datos (en caso de ser necesario), codificar las variables categóricas y realizar la agregación según las columnas seleccionadas anteriormente. Esta etapa es de codificación y agregación.

En esta etapa se mantienen las columnas seleccionadas como claves de la agregación y se crean las columnas agregadas según se explica en la sección 4.6.

La figura 4.10.2 es una representación esquemática de esta etapa.

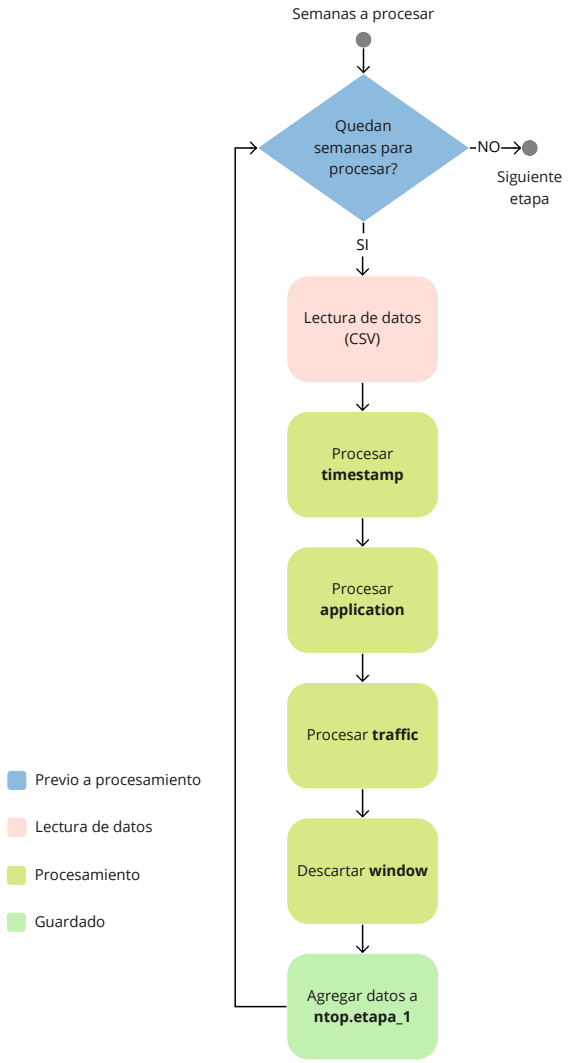


Figura 4.10.1: Etapa 1 del pipeline del conjunto de tráfico

Etapa 2 - Re-muestreo

Ubicación datos de entrada tabla `etapa_2[_rs]_sf_{OLD_SAMPLE_FREQUENCY}_agg_{AGGREGRATION_COLUMNS}` en base de datos `ntop` en *Hive*.

Formato de lectura ORC

Ubicación datos de entrada tabla `etapa_2[_rs]_sf_{NEW_SAMPLE_FREQUENCY}_agg_{AGGREGRATION_COLUMNS}` en base de datos `ntop` en *Hive*.

Formato de lectura ORC

Esta etapa se creó para evitar ejecutar la etapa 2 en caso de que ya se haya ejecutado anteriormente para las mismas columnas clave de agregación y una frecuencia de muestreo menor a la deseada. El procesamiento iterativo consiste en re-muestrear los datos y volver a calcular la agregación, agrupando según los nuevos índices y los valores previamente calculados. Esta etapa es de re-muestreo.

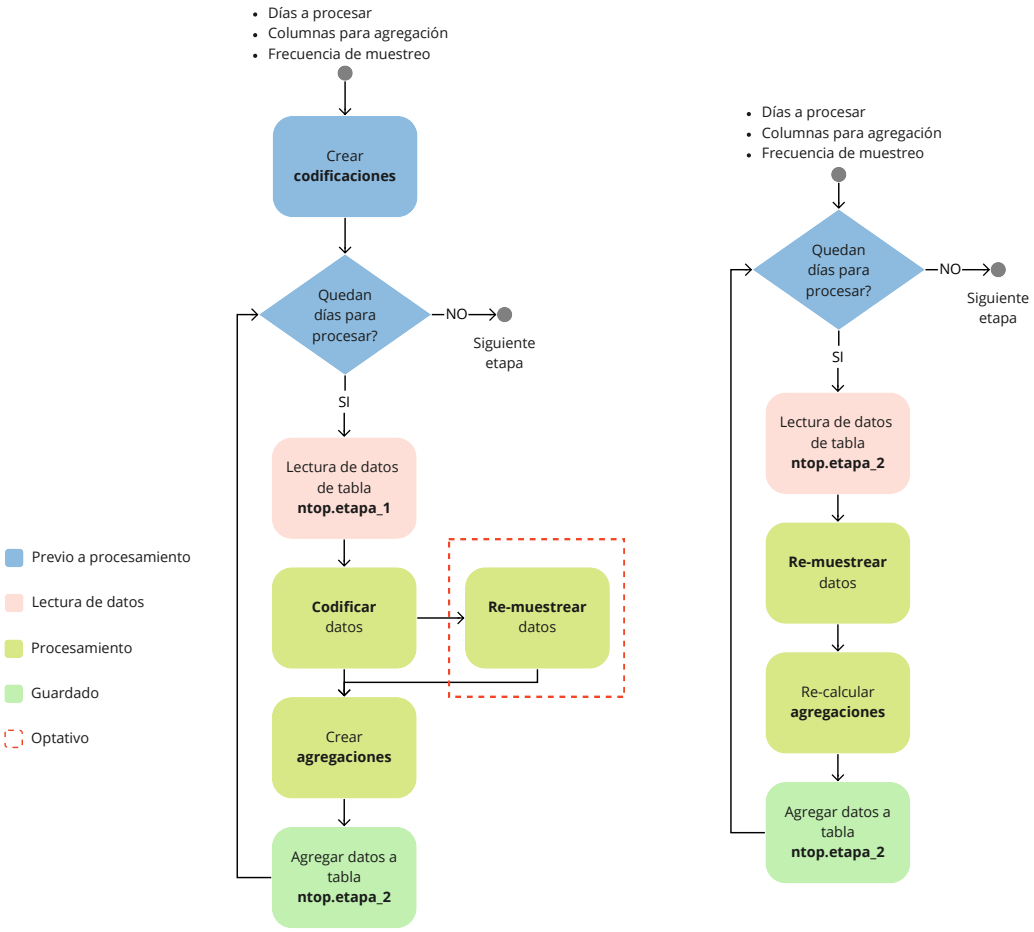


Figura 4.10.2: Etapa 2 del pipeline del conjunto de tráfico (izquierda) y etapa 2 de re-muestreo (derecha)

En esta etapa se mantienen todas las columnas previamente creadas.
La figura 4.10.2 es una representación esquemática de esta etapa.

4.10.2. Conjunto de consultas DNS

Etapa 1

Ubicación datos de entrada

Ambiente Facultad de ingeniería `/opt/datos/dsnlogs/yyyy-mm-dd/` en el sistema de archivos local

Ambiente Ceibal `/datalake/land/fing-dns/dns/yyyy-mm-dd/` en HDFS

Formato de lectura CSV

Ubicación datos de salida tabla `etapa_1` en base de datos `dnslogs` en *Hive*.

Formato de escritura ORC

En esta etapa, al comienzo se realiza el manejo de datos corruptos (4.2) y luego se lleva a cabo todo el procesamiento independiente de columnas (4.3.1, 4.3.1, 4.3.1, 4.3.1, 4.3.1). A partir del 18/12/2019 los logs guardados agregaron las columnas **mostGranularIdentityType**, **identityTypes** y **blockedCategories**, como en el proyecto se trabaja en un período donde el 96.7 % de los datos no contienen esas columnas, se decidió descartarlas.

Luego de procesar los datos, se agregan a la tabla correspondiente manteniendo la granularidad de lectura y también particionando por hora.

Las columnas al final de esta etapa son los siguientes:

timestamp Fecha y hora en la que se realizó la consulta en formato UTC.

mostGranularIdentity Identificador principal del dispositivo desde el que se realizó la consulta.

identities Identificadores principales y secundarios del dispositivo desde el que se realizó la consulta.

internalIP Dirección IP del dispositivo del que se hizo la consulta.

externalIP Dirección IP del *router* que procesó la consulta.

action Acción tomada por *Umbrella* al recibir la consulta, *Allowed*, *Blocked* o *Proxied* dependiendo de si la consulta fue permitida o no.

query_type Tipo de consulta DNS realizada.

response_code Código de respuesta a la consulta DNS.

domain Dominio consultado.

parsed_domain Dominio consultado parseado.

categories Categorías de contenido asignadas por *Umbrella* correspondientes con el dominio de la consulta.

year Año en que se hizo la consulta.

month Mes en que se hizo la consulta.

day Día del mes en que se hizo la consulta.

dayofweek Día de la semana en que se hizo la consulta.

hour Hora en que se hizo la consulta.

minute Minuto en que se hizo la consulta.

second Segundo en que se hizo la consulta.

ruee Identificador del centro desde donde se hizo la consulta.

category_1 Primer categoría de contenido que corresponde con el dominio de la consulta.

category_2 Segunda categoría de contenido que corresponde con el dominio de la consulta.

La figura 4.10.3 es una representación esquemática de esta etapa.

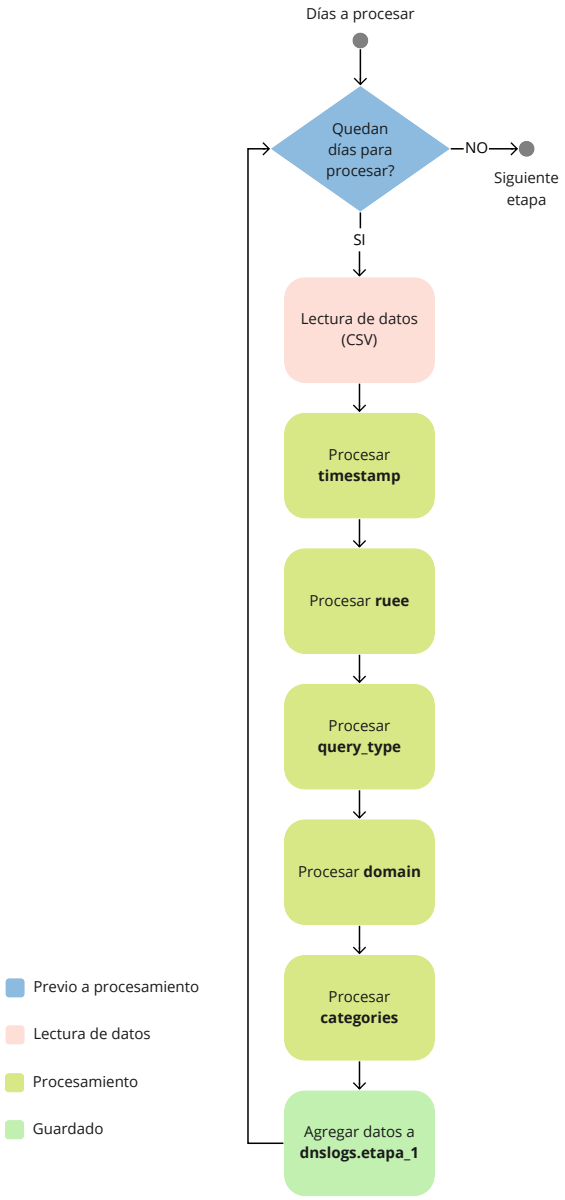


Figura 4.10.3: Etapa 1 del pipeline

Etapa 2

Ubicación datos de entrada tabla `etapa_1` en base de datos `dnslogs` en *Hive*.

Formato de lectura ORC

Ubicación datos de salida tabla `etapa_2` en base de datos `dnslogs` en *Hive*.

Formato de escritura ORC

En esta etapa, antes de iniciar el procesamiento por día se realiza la imputación de la información geoespacial y sociocultural (4.4) y se genera la información para realizar la imputación de las categorías (4.3.1). Luego, el procesamiento iterativo consiste en imputar los datos faltantes de las categorías (4.3.1) y agregar la información extra

al conjunto de datos (4.4), es decir, esta etapa es de imputación de datos.

A las columnas resultantes de la etapa anterior se le agregan las siguientes:

longitud Coordenada geográfica longitud del local.

latitud Coordenada geográfica latitud del local.

departamento Departamento al que pertenece el local.

localidad Localidad del departamento al que pertenece el local.

quintil Quintil al que corresponde el local.

zona_quintil Zona a la que pertenece el local (Rural o Urbana).

subsistema Subsistema educativo al que pertenece el local (por ejemplo: CEIP, CES, Universidad).

tipo_centro Descripción de tipo de dependencia del local (por ejemplo: Escuela Privada, Dependencia Administrativa, Utu).

nivel_fiabilidad Nivel de fiabilidad de los datos geoespaciales.

La figura 4.10.4 es una representación esquemática de esta etapa.

Etapa 3

Ubicación datos de entrada tabla `etapa_2` en base de datos `dnslogs` en *Hive*.

Formato de lectura ORC

Ubicación datos de entrada tabla `etapa_3[_rs][_rv]_sf_{SAMPLE_FREQUENCY}_agg_{AGGREGRATION_COLUMNS}` en base de datos `dnslogs` en *Hive*.

Formato de lectura ORC

En esta etapa, antes de iniciar el procesamiento por día se selecciona una frecuencia de muestreo temporal (SAMPLE_FREQUENCY), las columnas clave que se usarán en la agregación (AGGREGATION_COLUMNS) (4.6) y los valores a usar en caso de que se quiera seleccionar un subconjunto de los valores posibles para la agregación de una columna, en este caso se agrega el prefijo `\_rs` al nombre de la tabla. En caso de que la columna **ruee** no se encuentre en la clave de la agregación, se puede seleccionar un subconjunto de ruees con los que trabajar (para no incluir locales que no existen en el conjunto de tráfico), en este caso, se agrega el prefijo `_rv` al nombre de la tabla. También se crean las codificaciones a usar a partir de todo el conjunto de datos (4.5). Luego, el procesamiento iterativo consiste en re-muestrear los datos (en caso de ser necesario), codificar las variables categóricas y realizar la agregación según las columnas seleccionadas anteriormente (4.6). Esta etapa es de codificación y agregación.

En esta etapa se mantienen las columnas seleccionadas como claves de la agregación y se crean las columnas agregadas según se explica en la sección 4.6.

La figura 4.10.5 es una representación esquemática de esta etapa.

Etapa 3 - Re-muestreo

Ubicación datos de entrada tabla `etapa_3[_rs]_sf_{OLD_SAMPLE_FREQUENCY}_agg_{AGGREGRATION_COLUMNS}` en base de datos `dnslogs` en *Hive*.

Formato de lectura ORC

Ubicación datos de entrada tabla `etapa_3[_rs]_sf_{SAMPLE_FREQUENCY}_agg_{AGGREGRATION_COLUMNS}` en base de datos `dnslogs` en *Hive*.

Formato de lectura ORC

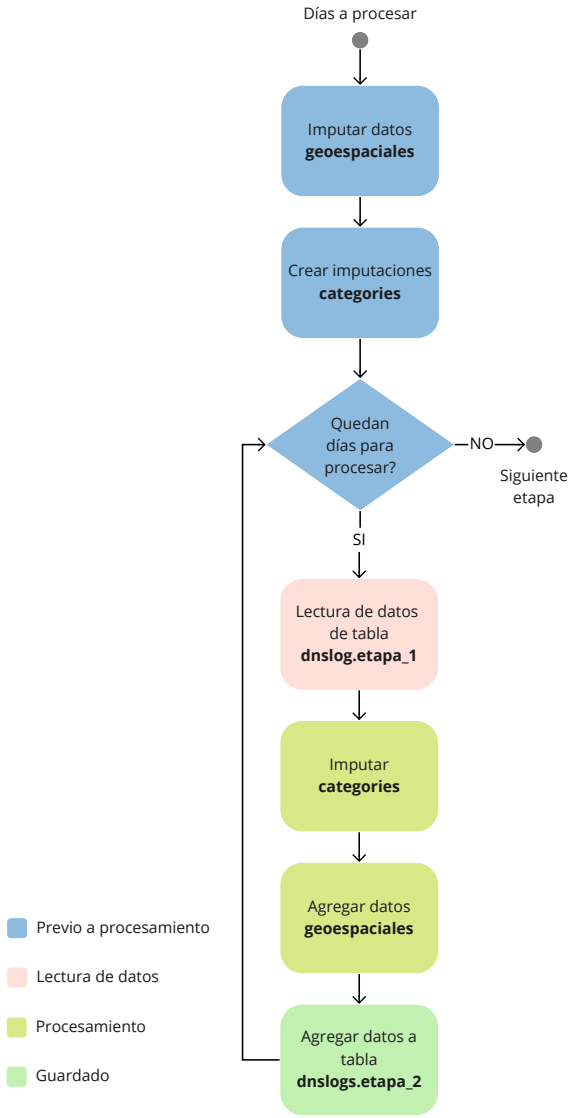


Figura 4.10.4: Etapa 2 del pipeline

Esta etapa se creó para evitar ejecutar la etapa 3 en caso de que ya se haya ejecutado anteriormente para las mismas columnas clave de agregación y una frecuencia de muestreo menor a la deseada. El procesamiento iterativo consiste en re-muestrear los datos y volver a calcular la agregación, agrupando según los nuevos índices y los valores previamente calculados. Esta etapa es de re-muestreo.

En esta etapa se mantienen todas las columnas previamente creadas.

La figura 4.10.5 es una representación esquemática de esta etapa.

Etapa 4

Ubicación datos de entrada tabla `etapa_3[_rs]_sf_{SAMPLE_FREQUENCY}_agg_{AGGREGRATION_COLUMNS}` en base de datos `dnslogs` en *Hive* y tabla `etapa_2[_rs]_sf_{SAMPLE_FREQUENCY}_agg_{AGGREGRATION_COLUMNS}` en base de datos `ntop` en *Hive*.

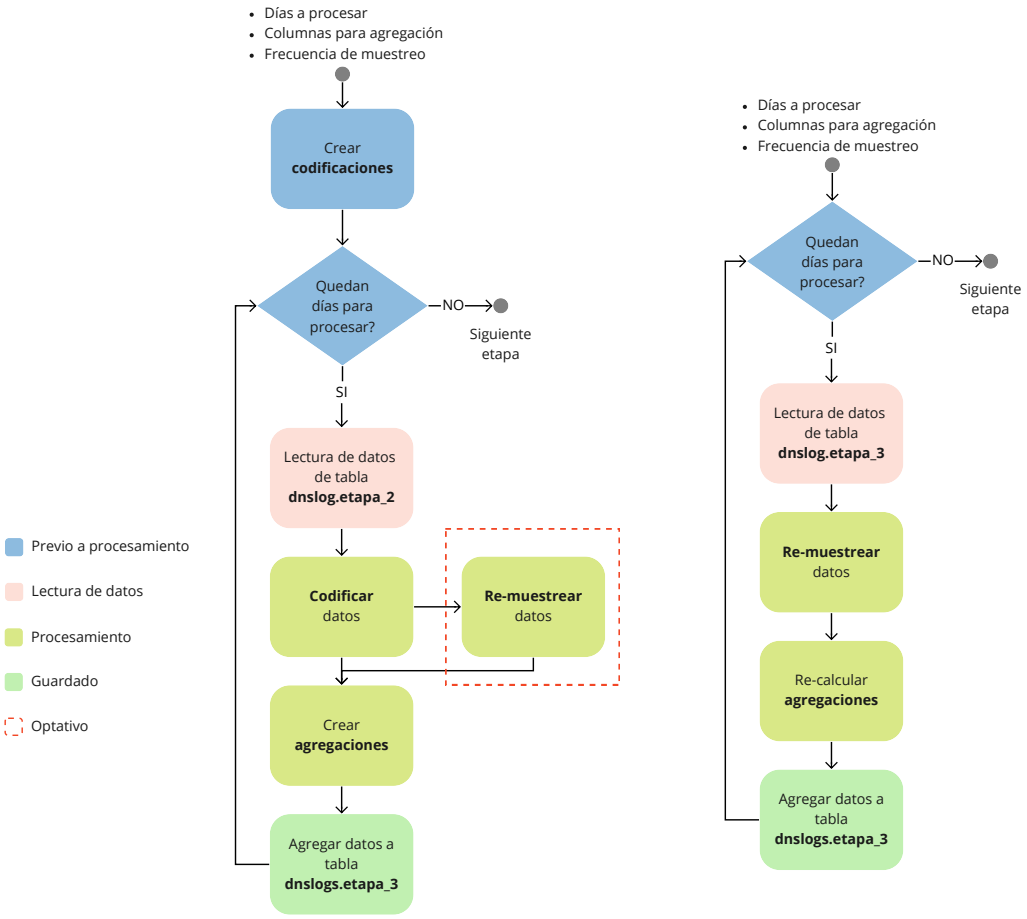


Figura 4.10.5: Etapa 3 del pipeline del conjunto de consultas DNS (izquierda) y etapa 3 de re-muestreo (derecha)

Formato de lectura ORC

Ubicación datos de salida `tfrecord/sf_{SAMPLE_FREQUENCY}_agg_{AGGREGRATION_COLUMNS}/[ruee_subset/][random_split/]partition={PARTITION}/date=yyyy-mm-dd/` en HDFS.

Formato de escritura tfrecords

Para esta etapa, los resultados se guardan en formato tfrecord, este formato fue diseñado para estar completamente integrado con TensorFlow y almacena una secuencia de filas de la tabla de datos como diccionarios en formato binario.[71]

En esta etapa, antes de iniciar el procesamiento por día se realiza el particionado de los datos (4.8) y se generan las predicciones de cada partición (4.7), para estas etapas se usan los valores del conjunto de datos de tráfico, además, se reduce el conjunto de ruees total al contenido en ese conjunto. El procesamiento iterativo consiste en reducir el conjunto de ruees de los datos de consultas DNS, calcular el EWMA para las columnas numéricas (en caso de asignar una ventana temporal válida 4.4.2), particionar los datos y agregar las predicciones. Luego del procesado individual, se calculan los estadísticos necesarios para la normalización de las columnas numéricas (de datos de entrada y de predicciones) usando los datos del conjunto de entrenamiento generado (4.9). Esta etapa es de particionado.

En esta etapa se mantienen las columnas de la etapa anterior y se crean las columnas de según se explica en la sección 4.4.2.

La figura 4.10.6 es una representación esquemática de esta etapa.

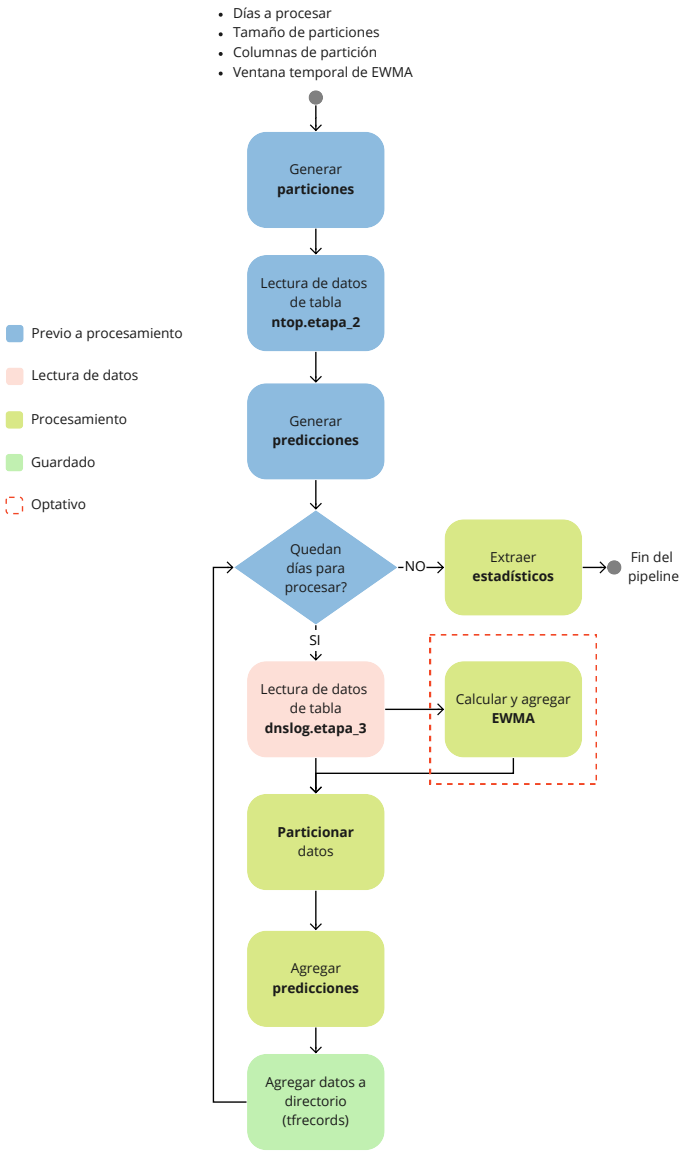


Figura 4.10.6: Etapa 4 del pipeline del conjunto de datos

4.11. Análisis de los datos

En esta sección se presentan, describen y comentan distintos análisis realizados sobre los datos con el fin de entender su comportamiento, elegir el conjunto de columnas a usar para crear los modelos objetivos y comparar la influencia de las transformaciones realizadas durante el procesamiento.

4.11.1. Análisis individual de las columnas

query_type

Luego del proceso de normalización y validación de los datos de esta columna, se realizó un gráfico de tortas (figura 4.11.1) para conocer la distribución de los distintos valores que puede tomar. Como se observa, el 90 % de las consultas corresponden al tipo de registro A (contiene la IP del dominio consultado), por este motivo no se consideró que esta columna sea una fuente de datos interesante y se descartó.

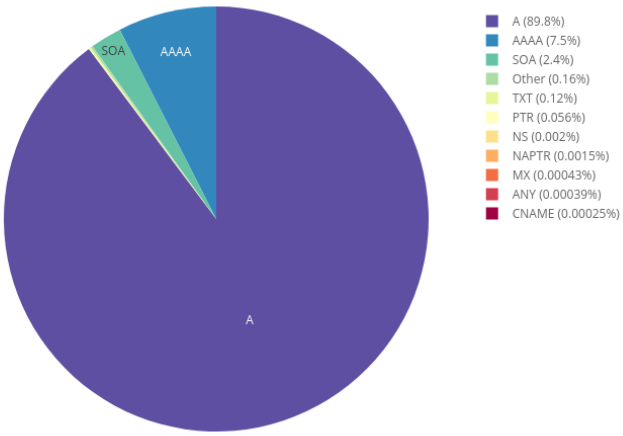


Figura 4.11.1: Distribución de las consultas DNS por tipo.

response_code

Luego del proceso de normalización y validación de los datos de esta columna, se realizó un gráfico de tortas (figura 4.11.2) para conocer la distribución de los distintos valores que puede tomar. Como se observa, el 84 % de las consultas corresponden al código de respuesta NOERROR, por este motivo no se consideró que esta columna sea una fuente de datos interesante y se descartó.

parsed_domain

Una vez aplicado el proceso de normalización de dominios, se analizó la distribución de los dominios más consultados, en la figura 4.11.3 se muestra esta distribución en un gráfico de tortas, todos los dominios con porcentaje menor al 1 % se agruparon en *other*. Como se puede observar los dominios agrupados en *google* contienen el mayor porcentaje individual de consultas (26.3 %), también se destacan los dominios *notificaciones*, *seguridadtd3*, *local* y *cisco-capwap-controller*, todos representan consultas generadas por los dispositivos usados pero no por usuarios finales, a pesar del considerable número de consultas en conjunto (10.9 %), es esperable que no presenten un porcentaje muy alto en el tráfico consumido.

También se muestra la distribución de los dominios en el conjunto de datos reducido al período temporal del conjunto de tráfico, como no se observan grandes diferencias se asume que se respeta la distribución global de esta columna.

Otro análisis interesante a hacer es determinar la cantidad de dominios que resultarían útiles para el entrenamiento de modelos de predicción, para esto, se calculó la cantidad de valores únicos que existen en el dominio

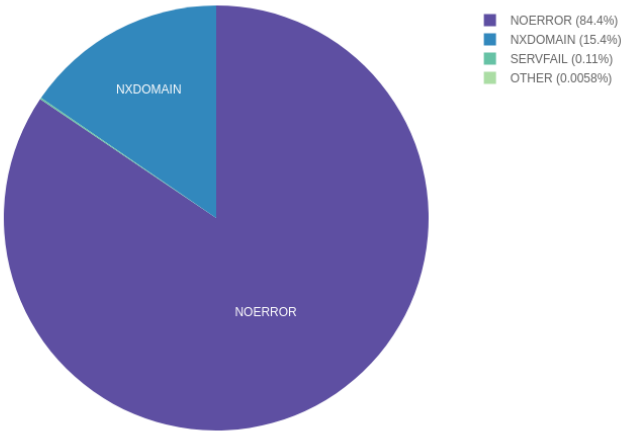


Figura 4.11.2: Distribución de las consultas DNS por código de respuesta.

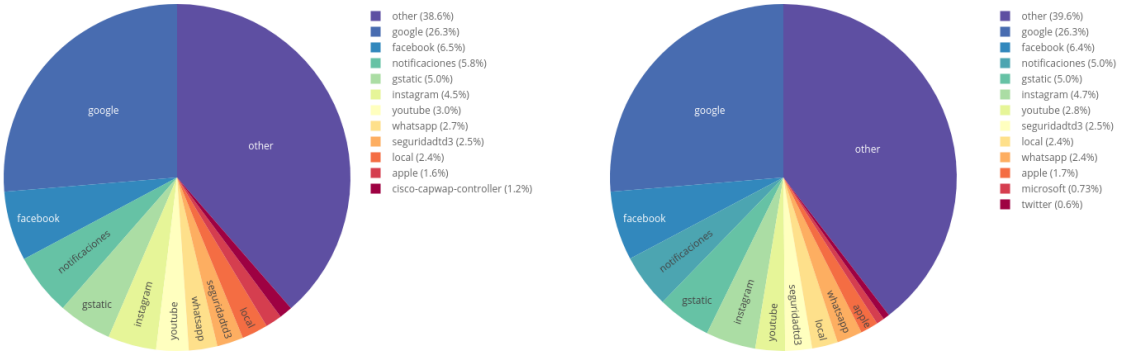


Figura 4.11.3: Comparación de la distribución de dominios parseados en el conjunto de datos completo (izquierda) y el conjunto de datos reducido (derecha).

(695.443.254) y la cantidad de consultas realizadas con cada valor. Al ponderar estos valores, se descubrió que el 60 % de las consultas se realizaron a 11 dominios (como se observa en la figura 4.11.3), luego de estos dominios, todos los valores representan menos del 1 % del total, el 90 % de las consultas se distribuyen entre 800 valores.

application

Para comparar los datos de los dominios consultados con los del tráfico consumido, se analizó la distribución de los dominios más consultados, en la figura 4.11.4 se muestra esta distribución en un gráfico de tortas. Si se compara con los gráficos de la distribución por consultas, se ve una clara diferencia de los dominios *youtube* (aumenta de 2.8 % a 20.3 %) y *google* (disminuye de 26.3 % a 6.3 %) estas diferencias son esperables por el tipo de datos a los que se acceden en los servicios. Sin embargo, otros dominios presentan diferencias que no parecen concordar con la lógica anterior (*facebook* o *instagram*), esto se debe a que la herramienta usada para la generación de datos, no puede etiquetar correctamente el tráfico encriptado, generando los dominios *ssl* y *ssl_no_cert* que consumen el 47.2 % del tráfico total, a pesar de esta situación, no se puede lograr una imputación correcta con los datos

disponibles por lo que se decidió incluir estos dominios dentro del dominio *other* para omitirlos.

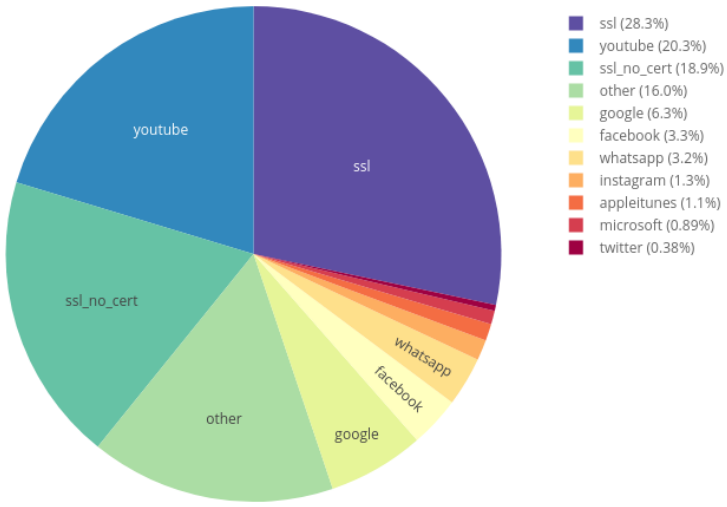


Figura 4.11.4: Distribución de los dominios por tráfico consumido.

categories

En la figura 4.11.5 se muestra la cantidad de registros existentes por categoría, se observa que el 88 % de las consultas tienen al menos una categoría, pero el porcentaje decrece drásticamente a partir de dos o más categorías. Dicho de otra forma, a partir de la lista de categorías existente para cada registro, siempre se puede extraer al menos una categoría pero, al extraer dos o más, la cantidad de valores faltantes es mayor al 40 %. A pesar de lo dicho, se decidió extraer las primeras dos categorías de la lista con la finalidad de permitir análisis posteriores fuera del ámbito del proyecto. El 12 % de valores sin categorías que se muestra en la figura se deben a consultas cuyos dominios no pudieron ser clasificados por la herramienta usada para generar los datos, al agregar los casos derivados de las consultas provenientes de dispositivos con sistema operativo Windows a este grupo, el valor aumentó al 22.8 %, este fenómeno generó la necesidad de diseñar un método de imputación que revirtiese la situación.

Una vez realizada la extracción y procesamiento, se crearon dos gráficos de tortas (figura 4.11.6) donde se observa la distribución de las categorías para la columna **category_1** antes y después de la imputación en los datos faltantes, para facilitar la comparación, todas las categorías con porcentaje menor al 1 % se agruparon en *other*. Luego de realizada la imputación se logró completar el 10 % de los datos faltantes, los cambios más significativos se dieron en categorías que en principio aparecían en menos de 1 % de las consultas, con estos datos se puede afirmar que el proceso de imputación mostró un desempeño excelente para esta columna, recuperando el 98 % de los valores objetivo.

Como se puede observar en la figura 4.11.7, el proceso de imputación logró un efecto aún mejor en cifras en la columna **category_2**, pasando de tener 57 % de valores faltantes a 31 %. Si se analizan en detalle los cambios, se nota que la mayoría de los registros recategorizados se etiquetaron con la categoría *web_page_translation*, este cambio pone en duda la efectividad del proceso en este caso pero, como esta columna no fue utilizada en las etapas siguientes, no se profundizó en la búsqueda de la razón de lo constatado.

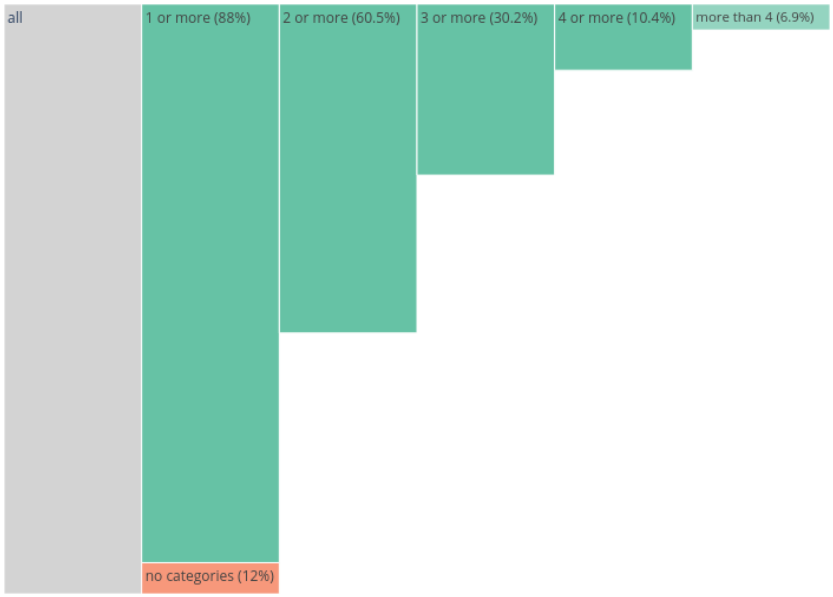


Figura 4.11.5: Distribución de las consultas DNS por categoría.

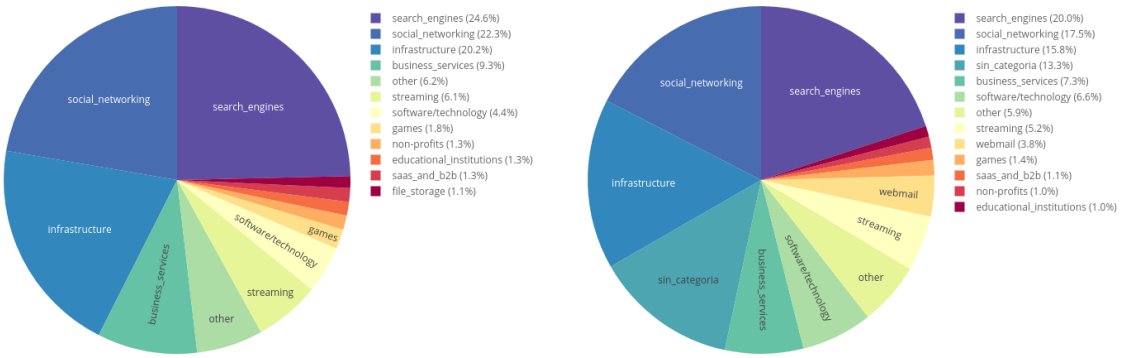


Figura 4.11.6: Cambios en la distribución de las categorías en la columna **category_1** luego de la imputación.

4.11.2. Análisis individual del comportamiento

Para comprobar si la distribución estadística de los datos se vio alterada al reducir el período temporal al de los datos de tráfico, se realizaron los gráficos mostrados en la figura 4.11.8, en ambos se compara el promedio (normalizado) de consultas DNS en el período temporal desde marzo a setiembre (promedio global) y el promedio en el período setiembre a diciembre (promedio reducido). En la primer gráfica se muestra el promedio por hora del día y en la segunda se muestra por día de la semana, en ambas gráficas se observa que ambas distribuciones tienen tendencias similares, en particular se aprecian algunos cambios en el promedio por día, esto se puede deber a períodos de vacaciones existentes en el período reducido.

Estas gráficas ayudan en la comprensión de los datos ya que describen el comportamiento existente, se observan que en promedio, la actividad de consultas DNS comienza a aumentar a partir de las 6 de la mañana, alcanzando su pico a las 10 y descendiendo hasta el medio día. Este comportamiento es coherente con la actividad esperada

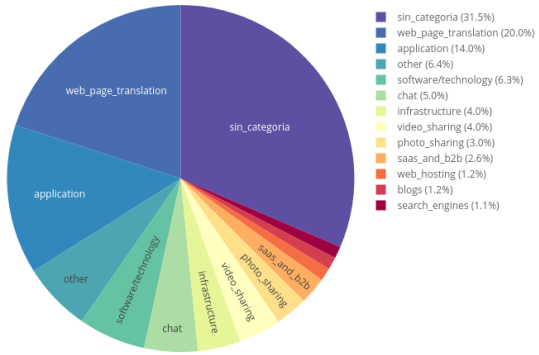


Figura 4.11.7: Cambios en la distribución de las categorías en la columna **category_2** luego de la imputación.

durante el turno matutino de los centros educativos. Un patrón parecido se repite desde las 12 aumentando hasta las 14 y reduciendo a medida que avanzan las horas, este comportamiento es el esperable del turno vespertino de los centros educativos.

Además, cuando se analiza el comportamiento promedio semanal, se ve una clara diferencia entre los días con actividad educativa (lunes a viernes) y los fines de semana. El comportamiento promedio se asemeja mucho en los días entre semana, en el conjunto global, se observa una pequeña tendencia al alza a medida que avanzan los días pero este comportamiento no se mantiene en el conjunto reducido.

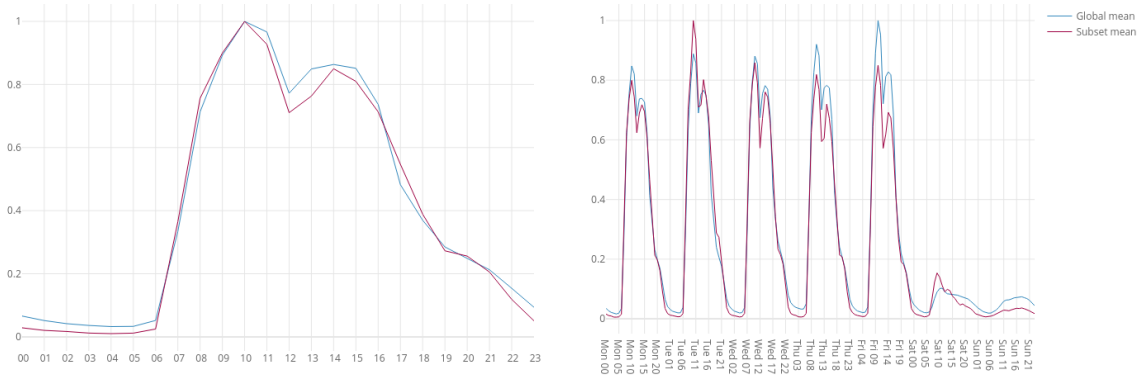


Figura 4.11.8: Comparación del promedio (normalizado) de consultas DNS en el período marzo-setiembre (*global mean*) y el período setiembre-diciembre (*subset mean*). A la izquierda comparación por hora, a la derecha comparación por día.

El siguiente análisis realizado fue la comparación entre la cantidad de consultas DNS y el tráfico generado por esas consultas, en la gráfica 4.11.9 se realiza esta comparación para el período reducido usando ambas series normalizadas (pues el tráfico es varias magnitudes mayor), se observa que el tráfico generado es mayor que el número de consultas (a excepción de algunos hechos puntuales), este tipo de análisis se volverá a realizar para comparar las predicciones de los modelos generados con la realidad.

Algunos fenómenos puntuales que se observan son los feriados y períodos de vacaciones donde, tanto consultas DNS como tráfico disminuye a niveles vistos durante los fines de semana.

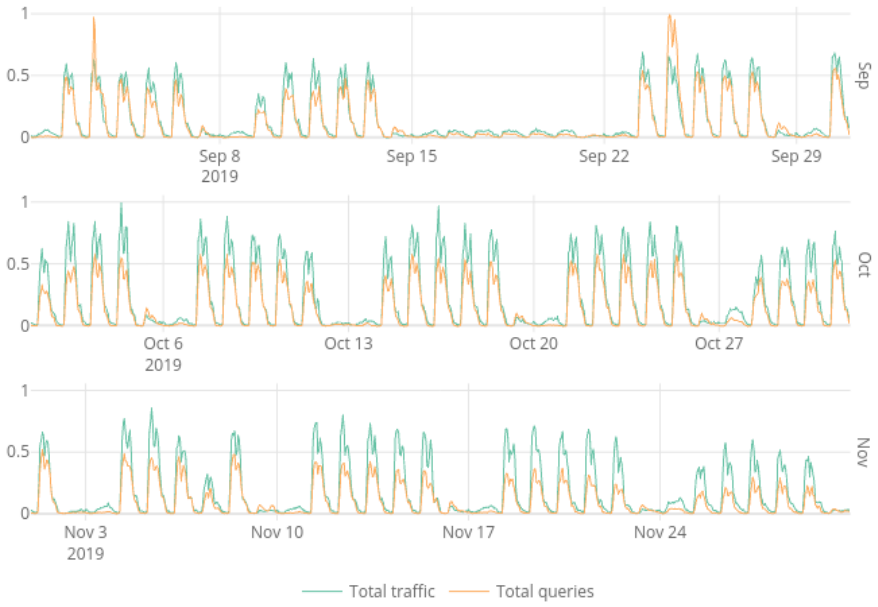


Figura 4.11.9: Comparación de la cantidad de consultas DNS realizadas en el período reducido (*total traffic*) y el tráfico generado (*total queries*).

Por último, en la figura 4.11.10 se muestran estadísticos representativos de las series temporales antes mencionadas en forma de *boxplots*. Como puede observarse, en ambos casos la mitad de los valores observados se encuentran en el rango $[0,0{,}05]$ y el 75 % se encuentran en el rango $[0,0{,}3]$, en ambos casos el valor máximo es 1 por lo que predecir valores mayores a $0{,}3$ puede ser muy difícil para los modelos de aprendizaje automático, en el caso del problema planteado en el proyecto, estos valores clasificados como *outliers* se buscan predecir de manera la más acertada posible.

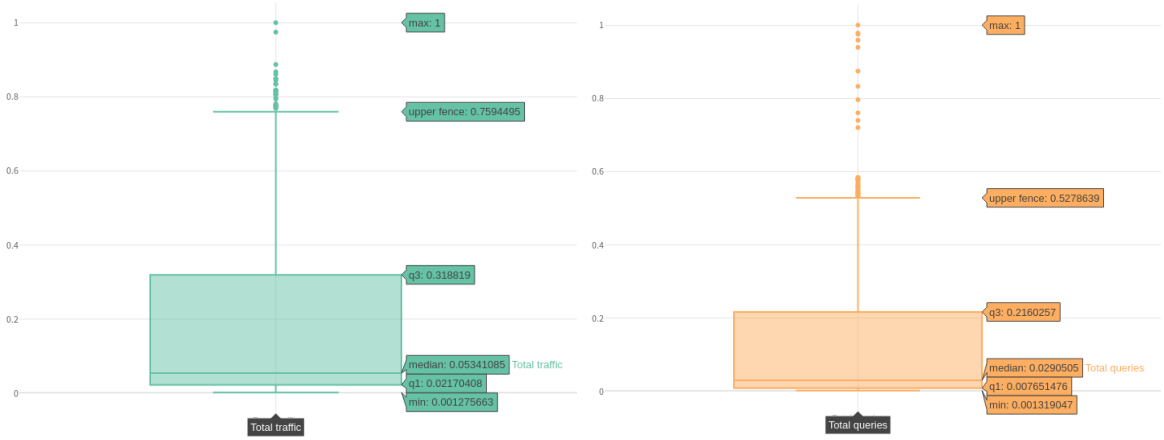


Figura 4.11.10: Boxplots de datos normalizados de tráfico total (izquierda) y datos normalizados de cantidad de consultas DNS (derecha).

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 5

Sistema de *Machine Learning* automatizado

En este capítulo se detalla lo realizado para cumplir con el OE2, se explica el concepto de sistema de *machine learning* automatizado, su implementación en el proyecto y detalles de sus componentes.

5.1. Caracterización de un sistema de autoML

Un proyecto de *machine learning* se compone por varias etapas que se caracterizan dentro de un *pipeline*, sus principales actores son los científicos de datos que se encargan del análisis e implementación y los expertos de dominio que aportan conocimiento y datos sobre los que trabajar. Generalmente, este *pipeline* inicia con la ingesta de nuevos datos, su pre-procesamiento, el entrenamiento de uno o varios modelos y finalmente la evaluación del rendimiento del modelo. Como se ve en la figura 5.1.1, este proceso es un ciclo iterativo donde se revisitan sus etapas hasta alcanzar un modelo que cumpla con el objetivo planteado.[72]

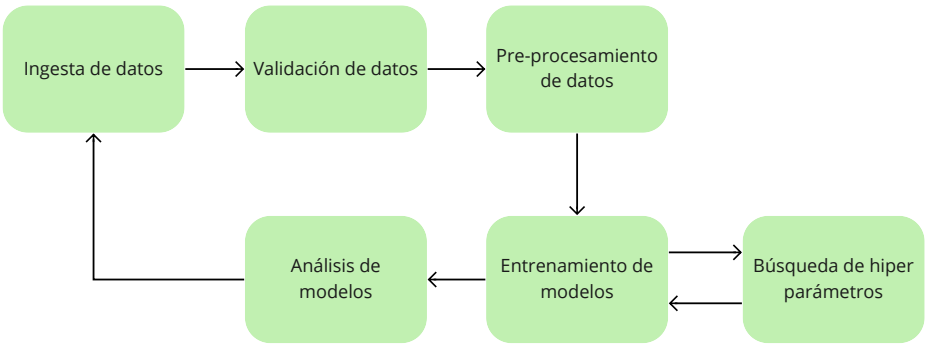


Figura 5.1.1: *Pipeline* de un proyecto de *machine learning*.

De este proceso reiterativo nace la necesidad de la automatización, intentando acelerar el tiempo de obtención de resultados, tratando de mejorar lo pre-existente e incluso, acercando el *machine learning* a personas que no tengan grandes conocimientos del área en general y/o del dominio del problema en particular. Estas automatizaciones de las distintas etapas son comprendidas dentro del campo del *machine learning* automatizado (AutoML),

en este sentido, un sistema de autoML puede ser clasificado según su grado de automatización (medido en cantidad de tareas realizadas sin intervención directa de un científico de datos)[73]. En la figura 5.1.2 se muestran los niveles propuestos en [73] junto con herramientas actuales clasificadas y las tareas automatizadas. Los niveles de clasificación son:

Nivel 0: Sin automatización En este nivel todas las tareas de ML son llevadas a cabo de forma manual siendo implementadas desde su base. No hay automatización posible.

Nivel 1: Solo ML automatizado En este nivel el sistema provee de implementaciones de algoritmos y modelos de ML (como árboles de decisión, redes neuronales, regresiones lineales, entre otros) pero su evaluación y el procesamiento de los datos se hace de forma manual. Este es el nivel mínimo de automatización de un sistema de autoML.

Nivel 2: ML + ATV automatizados de forma separada En este nivel el sistema agrega la exploración de modelos alternativos, su testeo y evaluación (ATV -*Alternative Models Exploration*, *Testing and Validation*-), con estas herramientas, se prueban distintos modelos para obtener el que mejor se adapte a los datos usados, en este nivel las herramientas requieren bastante trabajo manual para ser utilizadas por parte de científicos de datos y no son fácilmente empleables por expertos de dominio.

Nivel 3: ML + ATV automatizados de forma conjunta En este nivel el sistema provee herramientas que entrenan y al mismo tiempo eligen los mejores modelos de forma que se facilita su uso para los científicos de datos.

Nivel 4: ML + ATV + FE automatizados En este nivel el sistema provee herramientas capaces de transformar los datos de entrada en *features* que representen mejor el problema existente (FE -*Feature Engineering*-), sin intervención de los científicos de datos, los expertos de dominios pueden usar los sistemas de este nivel con cierta dificultad.

Nivel 5: ML + ATV + FE + PE automatizados En este nivel el sistema se encarga de generar y asignar las etiquetas usadas para el entrenamiento a partir de los datos de entrada y salida (PE -*Prediction Engineering*-). Expertos de dominio pueden usar los sistemas de este nivel de forma fácil y no se requiere mucho esfuerzo de programación.

Nivel 6: ML + ATV + FE + PE + TF + RSR automatizados En este nivel el sistema se encarga de determinar la tarea a resolver (TF -*Task Formulation*-) y resumir los resultados y realizar recomendaciones (RSR -*Result Summary and Recommendation*-) solo a partir de los datos de entrada, los sistemas de este nivel no requieren intervención alguna por parte de científicos de datos.

En este proyecto, se creó y utilizó un sistema de nivel 4, a continuación se describen los módulos que lo componen y en la figura 5.1.3 se presenta el diagrama de funcionamiento del sistema.

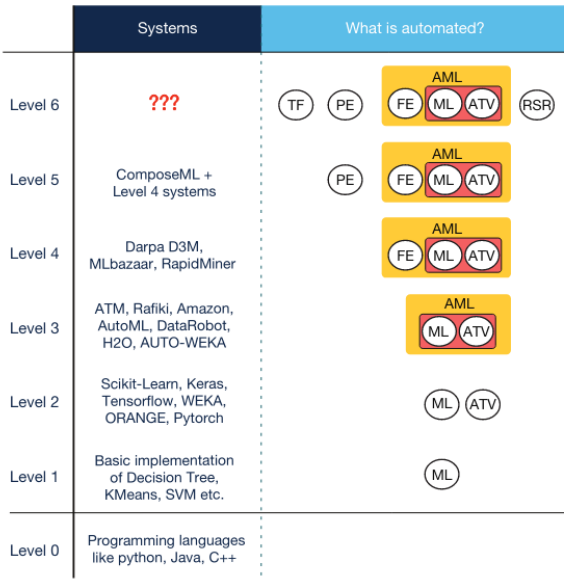


Figura 5.1.2: Clasificación de sistemas de autoML según su nivel de automatización y ejemplos de sistemas en cada nivel. (Figura extraída de [73])

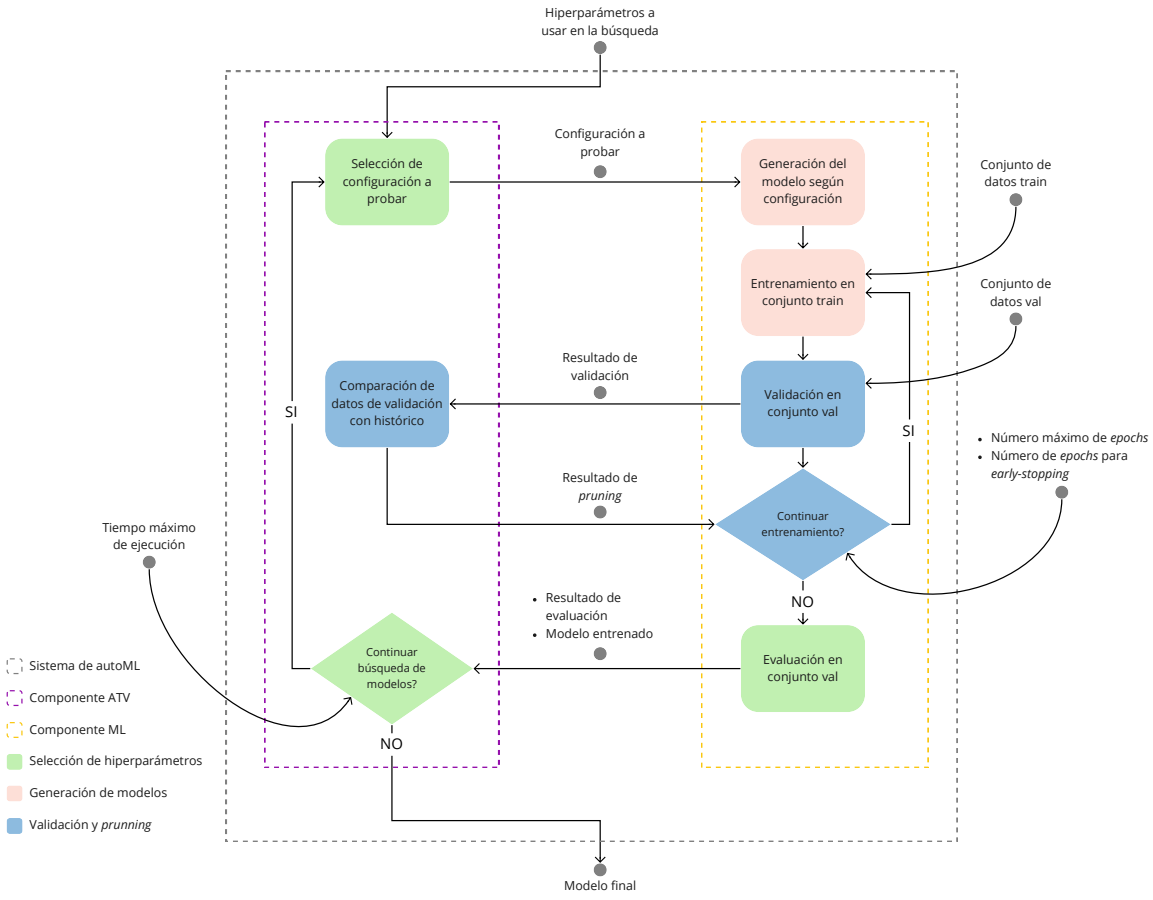


Figura 5.1.3: Diagrama de funcionamiento del sistema de autoML implementado

5.2. Modelo de aprendizaje profundo

En esta sección se comenta el modelo de aprendizaje profundo, la arquitectura general planteada y su implementación.

5.2.1. Arquitectura general planteada

Se describe el proceso de diseño y creación de las arquitecturas probadas en los modelos predictores, este es el componente referido como ML en el sistema. Durante el análisis de los datos se descubrieron ciertas características que determinaron el uso de algunos componentes específicos en la arquitectura de los modelos, a continuación se listan:

Tipo de variable

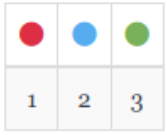
A cada una de las variables de un conjunto de datos se la puede categorizar de muchas formas que ayudan a entender su tratamiento particular, la categorización más básica es entre variables continuas y variables categóricas. La distinción entre ambas es la siguiente: las variables continuas son aquellas para las que existe un número infinito de valores entre cualquier par, por ejemplo, variables cuyos valores representan mediciones (variables físicas, económicas, etc) y variables que representan conteos. Las variables categóricas en contraste tienen un conjunto discreto de valores posibles, es decir que se considera categórica cualquier variable que represente un conjunto de elementos finito (por ejemplo el conjunto de los meses).[74]

Por su naturaleza numérica, las variables continuas pueden ser procesadas por el modelo de redes neuronales de forma automática, por lo tanto no es necesario aplicar ningún procesamiento especial a estas entradas. En cambio, las variables categóricas necesitan ser transformadas a alguna representación numérica antes de ingresar al modelo de redes neuronales, algunas de las posibles representaciones son:

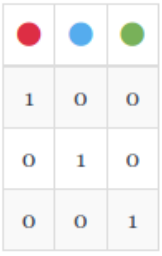
Etiqueta numérica A cada categoría se le asigna un valor entero. Este método le asigna un orden arbitrario a las categorías lo cual es indeseado, a modo de ejemplo, si se considera la variable color con valores rojo y azul, un modelo consideraría que el valor 1 (representando el rojo) sería menor que el valor 2 (azul) pero este razonamiento no tiene sentido en el conjunto que representa la variable. En la figura 5.2.1a se ve un ejemplo de esta representación.

One-hot encoding Para cada variable se produce un vector de tamaño igual a la cantidad de posibles categorías y, si un dato corresponde a la iésima categoría se asigna 1 a la posición i del vector y 0 al resto. Este método crea una alta esparcidad en las variables con alta cardinalidad, este comportamiento es indeseado en términos de memoria, por ejemplo, si se considera una variable con 10000 categorías posibles, cada representación vectorial será un vector donde el 9999 de los valores son cero. En la figura 5.2.1b se ve un ejemplo de esta representación.

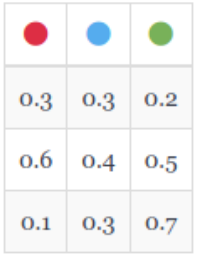
Embeddings A cada categoría se le asigna una representación vectorial densa de largo fijo, la representación vectorial se aprende durante el entrenamiento de un modelo sobre un conjunto de datos, si este método se entrena en una cantidad suficientemente grande de datos, se llegan a representaciones buenas que tienen en cuenta las relaciones entre las categorías.[27] En la figura 5.2.1c se ve un ejemplo de esta representación.



(a) Ejemplo de variable representada con etiquetas numéricas



(b) Ejemplo de variable representada con *one-hot encoding*



(c) Ejemplo de variable representada con *embeddings*

Figura 5.2.1: Comparación de representaciones de variables categóricas, cada columna representa un valor del conjunto de colores.

La última representación mencionada fue elegida por tratarse de un conjunto de datos masivo. Detalles de implementación y decisiones tomadas se comentan en 5.2.2. Este aspecto particular de la arquitectura se encarga de la ingeniería de *features* (FE) que es otro componente fundamental del sistema de autoML.

Representatividad de los datos

Durante el modelado no se conocían las relaciones entre las variables que componían el conjunto de datos, tampoco se sabía si algunas de ellas determinan independientemente del resto la predicción que se quería modelar, por lo tanto fue necesario buscar un modelo de extracción de *features*.

Para obtener este modelo, a la representación matricial de los datos de entrada se le podrían aplicar varias capas convolucionales concatenadas, estas capas (principalmente usadas en problemas de procesamiento de imágenes) funcionan sobre entradas en forma de matrices y extraen la información más importante de ellas, de esta forma se intenta lograr que el modelo se vuelva un 'experto de dominio', por la similaridad de las características de los datos con usados en problemas de procesamiento de lenguaje natural, donde recientemente se han usado las redes convolucionales para extraer *features* de representaciones vectoriales de palabras[56, 28], se elige usar convoluciones de dos dimensiones (a diferencia de convoluciones de tres dimensiones como se usan en el ámbito de procesamiento de imágenes), estas redes han demostrado aprender representaciones correctas en la dimensión temporal de los datos.

Temporalidad

Las consultas son sucesos temporales no independientes, esto significa que para determinar el futuro de la red el pasado debe ser tomado en cuenta. Para tomar en cuenta la temporalidad de los datos se pueden agrupar los datos en ventanas temporales (*timesteps*). Luego de que un *timestep* atraviesa las primeras capas de la red, se le aplican una o varias capas recurrentes concatenadas, estas capas funcionan muy bien para predecir sucesos que cuentan con cierta periodicidad.[75]

Arquitectura

Teniendo en cuenta lo planteado anteriormente se llega una arquitectura como la mostrada en la figura 5.2.2, donde la entrada es procesada según sus características particulares, luego un conjunto de capas convolucionales se encargan de generar representaciones internas especializadas y la temporalidad de las representaciones se maneja

con capas recurrentes, por último se comprime la información con capas densas y luego se genera un conjunto de capas por cada una de las predicciones que debe hacer la red, a cada una de estas salidas se le llama cabezal.

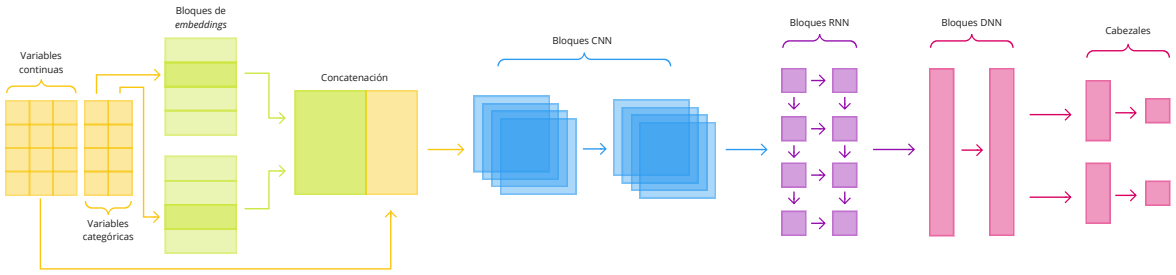


Figura 5.2.2: Diagrama de arquitectura base planteada.

Entrenamiento Durante el entrenamiento, se probaron tres funciones de pérdida: MSE usada frecuentemente para problemas de regresión y que es resistente a *outliers* y Huber y Huber Reversed usadas cuando el comportamiento que se quiere predecir tiene grandes diferencias de magnitud como se vio en la sección 4.11.2, este es el caso de las predicciones a usar en el proyecto. En particular, la función Huber Reversed podría ser útil para el problema a resolver ya que la magnitud del error debería ser muy alta si no se predicen los valores *outliers* muy altos que se buscan predecir de manera correcta.

5.2.2. Implementación

Lectura de datos de entrada

Para alimentar a la red con datos, se cargaron los archivos en formato tfrecord creados en la etapa final del pipeline (4.10.5) y se preprocesaron obteniendo un conjunto de datos de tipo `tf.data.Dataset` propio de *Tensorflow*, este tipo de conjunto aplica cada transformación de forma *lazy* sobre los datos, es decir, un momento antes de que se alimente al modelo con ellos, para evitar que este proceso repercutiera en el tiempo de entrenamiento del modelo, se mantenían los datos en memoria luego del primer *epoch* del entrenamiento.

En la figura 5.2.3 se muestran las transformaciones que se realizan sobre los datos, estas transformaciones se aplican dentro de la función creada para tal fin. A continuación se detallan los pasos aplicados:

1. **Lectura de datos** se selecciona un archivo con datos a partir de una lista de directorios y se cargan esos ejemplos (serializados) en memoria.
2. ***Batching* en períodos temporales** los datos se agrupan por fecha y hora y dividen en conjuntos del tamaño de un período temporal (t), en este momento la dimensión de la matriz que representa de un ejemplo es $t \times n$ donde n es la cantidad de *features*. Si no hay suficientes ejemplos para completar una muestra de n elementos, se completa la ventana temporal con ceros, si hay más de n elementos, se descartan los ejemplos sobrantes de forma aleatoria.
3. ***Parsing* de los ejemplos** los ejemplos se transforman en un formato manejable a partir de su forma serializada.
4. **Normalización de *features* numéricas** usando los estadísticos obtenidos durante el procesamiento, las *features* numéricas y las predicciones son normalizadas (sección 4.9).
5. ***Batching*** los ejemplos se dividen en conjuntos del tamaño de la muestra (b) que se usará para entrenar el modelo en un en un instante dado, en este momento una muestra tiene b ejemplos de tamaño $t \times n$.

6. ***Caching*** durante la primer iteración del entrenamiento, el conjunto de datos se almacena en memoria para evitar volver a ejecutar el procesamiento en las iteraciones faltantes.

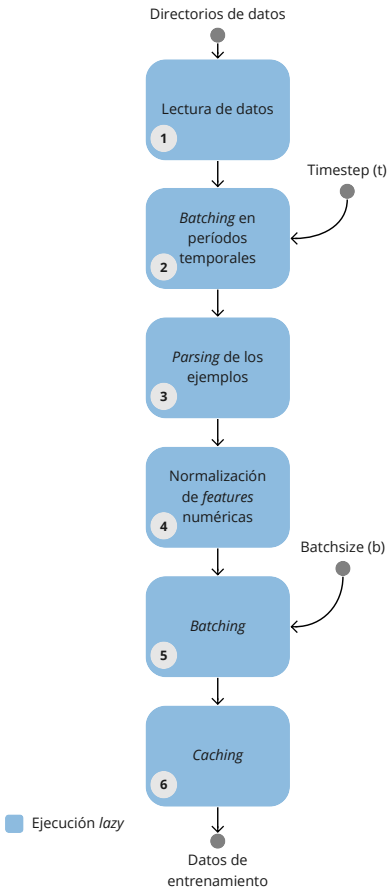


Figura 5.2.3: Transformaciones aplicadas a los datos antes del entrenamiento.

Modelo de redes neuronales

La implementación del modelo de redes neuronales se creó con el principio de poder alterar parámetros de la arquitectura sin tener que cambiar el código, por esto, se creó una clase de tipo `dataclass`[76] llamada `Params` cuya instancia almacenaba atributos de una arquitectura de red (por ejemplo cantidad de capas densas y cantidad de neuronas por capa).

El modelo se implementó usando la biblioteca `tensorflow`. Dentro de la red se usaron varios tipos de bloques estructurales, los tres principales y con mayor reuso son: las redes neuronales densas (DNN), las redes neuronales recurrentes (RNN) y las redes neuronales convolucionales (CNN). Para ensamblar con facilidad distintas arquitecturas, se creó una función específica para crear cada uno de los bloques (implementada dentro del módulo `models.utils`), a continuación se detallan los bloques creados ordenados según su ubicación en el modelo.

Procesamiento de la entrada Como ya se comentó anteriormente, la entrada se componía tanto de variables continuas como de variables categóricas, a continuación se comenta la implementación del procesamiento de ambos tipos.

Variables categóricas Estas variables se procesan usando *embeddings*, para su uso deben definirse dos parámetros: la cantidad de valores posibles que puede tomar una variable f y el tamaño de los vectores densos creados.

Para asignar la cantidad de valores posibles $|f|$ para cada variable, se contaron los valores durante el procesamiento de los datos.

El tamaño de los vectores densos (d_f) se asignó en función de la cantidad de valores y una fracción r, es decir $d_f = |f| \cdot r, 0 \leq r \leq 1$.

Las representaciones vectoriales resultantes debieron concatenarse en la dimensión de las *features* para obtener un nuevo vector de tamaño $\sum_{\forall f} d_f = |f| \cdot r$. La función encargada de crear los *embeddings* se llama `create_embeddings`.

Dentro de la clase `Params` el siguiente parámetro altera el comportamiento descrito anteriormente:

- `embeddings_size` fracción r usada para el cálculo del tamaño de los vectores creados.

Variables continuas Las variables continuas son normalizadas durante la lectura de los datos (sección 5.2.2) y concatenadas al vector que representa las variables categóricas.

Bloque CNN La función encargada de crear un bloque CNN se llama `create_cnn_block`, este tipo de bloque esta compuesto por una capa convolucional, una capa de normalización y una de *dropout*.

Todas las capas de este bloque se agregan dentro de un componente `TimeDistributed` que permite aplicar una capa a cada partición temporal de la entrada.

Dentro de la clase `Params` los siguientes parámetros alteran el comportamiento descrito anteriormente:

- `num_cnn_layers` cantidad de capas convolucionales y bloques CNN.
- `num_cnn_filters` cantidad de filtros F dentro de cada capa convolucional.
- `cnn_kernel_size` tamaño de los kernels K_i dentro de cada capa convolucional.
- `cnn_dropout` fracción r usada dentro de la capa de *dropout* de cada bloque.
- `cnn_batch_norm` índice i de bloque en el que se agrega una capa de normalización, si $b_k \mod i = 0$ se agrega una capa de normalización.
- `cnn_activation` función de activación usada en cada capa convolucional.

Bloque RNN

La función encargada de crear un bloque RNN se llama `create_rnn_block`, este tipo de bloque está compuesto por una capa recurrente, una capa de regularización y una de *dropout*.

Dentro de la clase `Params` los siguientes parámetros alteran el comportamiento descrito anteriormente:

- `num_rnn_layers` cantidad de capas recurrentes y bloques RNN.
- `rnn_hidden_dim` tamaño de la salida de cada capa recurrente.
- `rnn_dropout` fracción r usada dentro de la capa de *dropout* de cada bloque.
- `rnn_layer_norm` índice i de bloque en el que se agrega una capa de normalización, si $b_k \mod i = 0$ se agrega una capa de normalización.
- `rnn_layer_type` tipo de capa recurrente usada en los bloques.

Bloque DNN

La función encargada de crear un bloque DNN se llama `create_dnn_block`, este tipo de bloque está compuesto por una capa densa, una capa de regularización y una de *dropout*.

Dentro de la clase `Params` los siguientes parámetros alteran el comportamiento descrito anteriormente:

- `num_dnn_layers` cantidad de capas densas y bloques DNN.
- `dnn_hidden_dim` tamaño de la salida de cada capa densa.
- `dnn_dropout` fracción r usada dentro de la capa de *dropout* de cada bloque.
- `dnn_layer_norm` índice i de bloque en el que se agrega una capa de normalización, si $b_k \mod i = 0$ se agrega una capa de normalización.
- `dnn_activation` función de activación usada en la cada capa densa.

Cabezal

Un cabeza es un bloque DNN ubicado al final del modelo cuando se tienen varias salidas, cada cabezal correspondiente a cada salida contiene información particular de la misma.

Dentro de la clase `Params` se agregó un conjunto de parámetros análogo al de los bloques DNN para representar a los cabezales, estos parámetros se identifican sustituyendo `dnn` por `head` en el nombre.

Ensamblado del modelo El uso de la biblioteca Tensorflow exige que un modelo se ensamble (*build*), es decir, que se definan las capas usadas en la red para luego compilarse, en esta etapa se configura como será el entrenamiento, asignando función de *loss*, optimizador y métricas a evaluar. Estos procesos se llevan a cabo dentro de la función `build_and_compile`, en esta se usa la API funcional de Tensorflow[77] que permite personalizar el modelo para poder manejar múltiples entradas y múltiples salidas.

Se usan parámetros específicos para la definen el ensamblado y compilación de la red que no pertenecen a los parámetros usados por los métodos de ensamblado de bloques antes descritos, los parámetros específicos del ensamblado son:

- `name` nombre de la red.
- `out_dims` lista compuesta por las dimensiones de las salidas de la red.
- `out_activation` función de activación de la capa de salida usada para las salidas cuyo dominio son los números reales, las demás salidas usan la función *softmax* que genera un vector de valores en el rango $[0, 1]$ que suman 1 y representan porcentajes.

Los parámetros específicos de la compilación son:

- `loss` funciones de *loss* usadas para cada una de las salidas de la red, por defecto se usa el error cuadrático medio (MSE).
- `lr` *learning rate* inicial, por defecto se usa 3^{-4}.
- `adam_epsilon` parámetro específico del optimizador Adam que conviene optimizar, por defecto es $0{,}001$.
- `metrics` lista de metricas a evaluar durante el entrenamiento.
- `loss_weights` diccionario compuesto por el peso que se le asigna a cada una de las salidas de la red, por defecto todas las salidas tienen el mismo peso.

5.2.3. Entrenamiento de redes neuronales sobre Spark

En un principio se intentó entrenar los modelos sobre el cluster de Ceibal, los nodos en este cluster no contaban con GPUs por lo que para equiparar el rendimiento se hizo uso del entrenamiento distribuido, a pesar de no obtener resultados deseables se logró realizar una investigación acerca de varias implementaciones existentes, a continuación se comentan todas las bibliotecas probadas para realizar el entrenamiento distribuido de un modelo de *Tensorflow*.

Implementación en TensorFlowOnSpark

Esta biblioteca fue creada por *Yahoo* para realizar el entrenamiento de forma distribuida integrándolo con *Spark* a partir de su versión 2.3, el uso de su API consiste en varios pasos:

1. **Inicio** se lanza la función de ensamblado y entrenamiento de *TensorFlow* en los ejecutores, junto con procesos escucha para el manejo de mensajes de datos y control entre los nodos.
2. **Carga de datos** permite ingerir los datos en dos modos:
   - **InputMode.TENSORFLOW** usa la API integrada de *TensorFlow* para leer los datos directamente desde HDFS.
   - **InputMode.SPARK** envía los datos en forma de RDD de *Spark* a los nodos, en este modo los procesos no concluyen hasta que todos los datos sean ingeridos, incluso si el modelo terminó su entrenamiento.
3. **Apagado** termina con los procesos de *TensorFlow* creados en los ejecutores.

Los modos de ingesta de datos se diferencian en la cantidad de tareas que recaen sobre *Spark*. InputMode.TENSORFLOW solo usa *Spark* para lanzar los nodos de *TensorFlow* sobre los ejecutores de *Spark*, una vez lanzados, esencialmente se vuelve un cluster de *TensorFlow* y cada nodo lee los datos directamente de HDFS. Con InputMode.SPARK *Spark* no solo lanza los nodos de *TensorFlow* sobre los ejecutores, sino que también se encarga de transferir los datos desde RDDs *Spark* a los ejecutores. Los datos de las particiones de los RDDs se encolan y cada nodo se encarga de sacar su partición de la cola, este proceso provoca *overhead* en la etapa de I/O.

Esta biblioteca fue creada en el año 2017 cuando todavía *TensorFlow* y *Spark* estaban en etapas más tempranas de desarrollo, no cuenta con funcionalidades integradas de *Spark* que aseguren la estabilidad del entrenamiento y la recuperación en caso de que falle, tampoco permite que los nodos usen más de un núcleo.

Para el control del cluster de entrenamiento se usa la estrategia de distribución *MultiWorkerMirroredStrategy* nativa de *TensorFlow*.

Está biblioteca se usó en primera instancia por ser compatible con la versión de *Spark* instalada en el ambiente de Facultad de Ingeniería. Durante su uso se observó que no es capaz de reportar errores de ejecución de forma inmediata, dificultando el control de fallas.

Implementación en Spark TensorFlow Distributor

Esta biblioteca forma parte del ecosistema *TensorFlow* y usa funcionalidades nativas de *Spark* en su versión 3.0 para realizar el entrenamiento distribuido.

Spark 3.0 implementa un nuevo modo de ejecución llamado *barrier execution mode* que es diferente al modelo estándar de *Map/Reduce*. En el modelo *Map/Reduce*, todas las tareas de una etapa son independientes y no se comunican entre sí, si una tarea falla, solo esa tarea se vuelve a ejecutar. En *barrier execution mode*, todas las tareas de una etapa se lanzan al mismo tiempo y se comunican entre sí, además, si una falla toda la etapa se vuelve a ejecutar.

La ingesta de datos se hace usando la API nativa de *TensorFlow*, permitiendo la lectura directa desde HDFS.

Esta biblioteca permite el uso de más de un núcleo por nodo y asegura una buena estabilidad y recuperación en caso de que el entrenamiento falle.

Para el control del cluster de entrenamiento se usa la estrategia de distribución *MultiWorkerMirroredStrategy* nativa de *TensorFlow*.

Está biblioteca se usó en el ambiente de Facultad de Ingeniería donde se permitía modificar la versión de *Spark*. La comprobación de errores de ejecución se da de manera automática.

Implementación en Horovod

Esta biblioteca fue creada por *Uber* para realizar el entrenamiento de forma distribuida integrándolo con *Spark* a partir de su versión 2.3,2, el uso de su API consiste en varios pasos:

1. **Inicio** se lanza la función de ensamblado y entrenamiento de *TensorFlow* en los nodos.
2. **Carga de datos y entrenamiento** permite ingerir los datos en dos modos:
   - **Horovod Spark Estimators** permite realizar la ingesta de datos y entrenamiento directamente sobre un *DataFrame Spark*.
   - **Horovod Spark Run** permite lanzar y ejecutar una función de ensamblado y entrenamiento de *TensorFlow* directamente sobre *Horovod*.

Los modos de ingesta de datos y entrenamiento se diferencian en la cantidad de tareas que recaen sobre *Horovod*. *Horovod Spark Estimators* permite el entrenamiento directamente sobre un *DataFrame Spark*, *Horovod* se encarga de todas las tareas de entrenamiento e ingesta, ocultando la complejidad de integrar *DataFrames* y *TensorFlow*, los resultados del entrenamiento se integran directamente como una columna del *DataFrame*. *Horovod Spark Run* permite controlar todos los aspectos del entrenamiento pues se usa la misma función de ensamblado y entrenamiento de *TensorFlow* que se usaría para el entrenamiento local, en este caso la ingesta de datos se hace con la API nativa de *TensorFlow*.

Para el control del cluster de entrenamiento se usa la estrategia de distribución *DistributedOptimizer* específica de *Horovod*.

Comparación de bibliotecas

En las secciones anteriores se describieron las particulares de cada una de las bibliotecas utilizadas, en el cuadro 5.2.1 se muestra una comparación entre todas.

Biblioteca de distribución	Versión mínima de *Spark*	Ingesta de datos
TensorFlowOnSpark	2.3.0	Nativa *TensorFlow* o RDDs *Spark*
Spark TensorFlow Distributor	3.0.0	Nativa de *TensorFlow*
Horovod	2.3.2	Nativa *TensorFlow* o DataFrame *Spark*

Biblioteca de distribución	Control de cluster	Estabilidad del entrenamiento
TensorFlowOnSpark	*MultiWorkerMirroredStrategy*	No
Spark TensorFlow Distributor	*MultiWorkerMirroredStrategy*	Si
Horovod	*DistributedOptimizer*	No

Tabla 5.2.1: Comparación de las principales características de las bibliotecas para entrenamiento distribuido.

En la práctica solo se pudieron ejecutar sobre el cluster de Ceibal las bibliotecas *TensorFlowOnSpark* y *Horovod*, con la primera no se logró un nivel de paralelización suficiente como para que el entrenamiento fuera efectivo a nivel de tiempos de ejecución, la segunda permitió el uso de todos los CPUs de cada nodo pero no se logró un nivel de estabilidad suficiente como para ejecutar todas las pruebas pensadas.

Para solucionar este problema, se cambió la estrategia y se utilizó el cluster de ClusterUY[59] que cuenta con GPUs en sus nodos.

5.3. Búsqueda de hiperparámetros

En esta sección se describe el componente del sistema encargado la de exploración de modelos y selección del mejor para el conjunto de datos dado (ATV).

Un modelo de redes neuronales tiene un conjunto de parámetros (pesos) que se ajustan sobre un conjunto de datos, pero también tiene un conjunto de hiperparámetros que pueden ser ajustados de forma manual o semi-manual para obtener una ganancia potencial en el resultado. Estos hiperparámetros incluyen características de la arquitectura del modelo (como la cantidad de capas usadas o el tamaño de las capas de *embeddings*) y parámetros usados durante el entrenamiento como el *learning rate*. En este proyecto, el proceso se automatizó realizando la búsqueda de arquitecturas e hiperparámetros de forma inteligente, obteniendo como resultado la mejor adaptada al conjunto de datos trabajado.

5.3.1. Implementación

Para la implementación de la búsqueda del mejor modelo posible, se usó la biblioteca Optuna[48] en conjunción con las técnicas ya descritas de selección de hiperparámetros mediante TPE (2.4.1) y poda (*pruning*) de modelos mediante *Hyperband* (2.4.2). Esta decisión fue tomada teniendo en cuenta comparaciones como las mostradas en la figura 5.3.1 donde se observa que es una combinación de métodos que funciona bien para distintos conjuntos de datos, no solo en las mejores situaciones sino que en general.

La poda realizada con *Hyperband* se modificó de forma que iniciase su funcionamiento luego de n epochs, evitando de esta forma que se poden soluciones durante un tiempo razonable que permitiera la evaluación más justa.

Para almacenar los estudios generados por esta biblioteca se usó una base de datos *postgresql* que permite el acceso concurrente. Una vez se decidió una configuración a probar, este componente delega el entrenamiento y evaluación a la implementación del componente ML del sistema, la función que usa para comparar los modelos generados y poder decidir cual es el mejor puede ser independiente de la función elegida para entrenar ese modelo, en este proyecto se decidió usar las utilizadas como funciones de pérdida.

Las características particulares de Optuna combinadas con la ejecución sobre el cluster de supercomputación ClusterUY[59] permitieron la paralelización de la búsqueda en hasta cuatro nodos, cada uno entrenando un modelo en una GPU Nvidia Tesla P100. A estos nodos se les agregó otro que funcionaba a modo de base de datos. En la figura 5.3.2 se muestra un diagrama del despliegue del sistema de autoML en cluster.

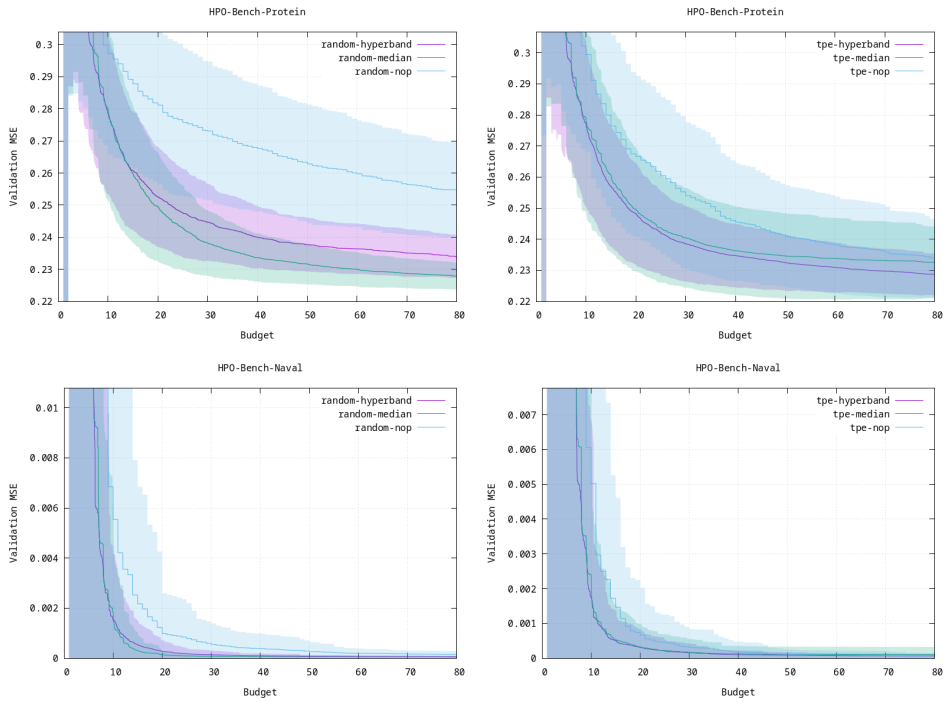


Figura 5.3.1: Comparación de las distintas combinaciones de métodos de selección de hiperparámetros y de *pruning*. A la izquierda se usa *random search* y a la derecha TPE, en cada gráfica se compara la versión sin *pruning* (nop) contra la versión con *Median Stopping Rule* (median) y la versión con *Hyperband* (hyperband). (Figura extraída de [78])

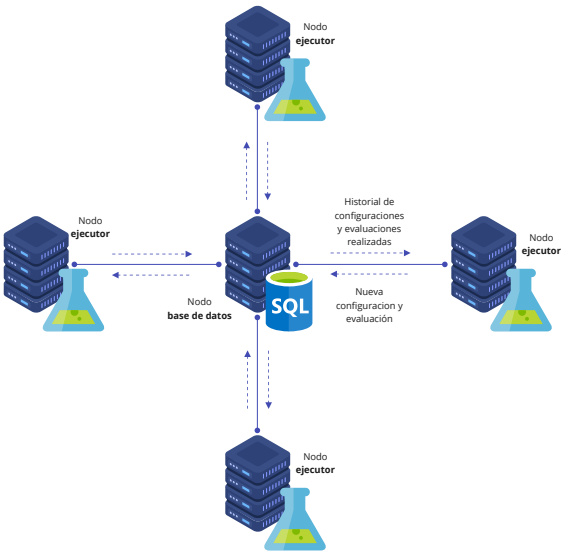


Figura 5.3.2: Diagrama de despliegue de sistema de autoML en nodos de ClusterUY.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 6

Problemas y experimentos

En este capítulo se detallan los experimentos realizados para cumplir con los EO3 y OE4, estos experimentos resultaron en varios modelos capaces de resolver los problemas planteados, para cada uno se detallan los resultados y conclusiones obtenidos.

6.1. Descripción de los problemas y soluciones

En este proyecto se planteó la resolución de dos problemas: la predicción del número de consultas DNS en la siguiente hora a partir de datos actuales y la transformación del número de consultas DNS actuales en el volúmen tráfico consumido.

Para la resolución de cada uno se buscó crear un modelo específico usando los mismos datos de entrada (que se detallan en cada experimento) pero alternando la salida según el problema a resolver, en el caso de la predicción de consultas DNS los datos de salida fueron:

- **total de consultas en la próxima hora** La cantidad de consultas realizadas en la hora siguiente al momento dado en los datos de entrada.

Mientras que para la transformación de consultas DNS en tráfico los datos de salida usados fueron:

- **tráfico total** El volumen de tráfico (en bytes) correspondientes con el momento dado en los datos de entrada.

También se exploraron distintas granularidades espaciales y temporales para hallar la que mejor resultado obtuviese (estas variantes se comentaron en la sección 4.6), esta exploración se verá reflejada en las siguientes secciones, para cada configuración espacial se comentan los experimentos y resultados de cada problema. Además, cada agregación de datos se particionó usando la columna **date** y con selección aleatoria (sección 4.8), por lo tanto, cada subconjunto usado (*train*, *val*, *test*) estuvo compuesto por todos los registros de varios días no consecutivos.

Para la evaluación de los resultados generados por cada modelo, la situación antes observada en los datos (sección 4.11.2) generó el siguiente problema, como el 75 % de los datos a los que se quiere aproximar eran menores a $0{,}3$ métricas usadas comúnmente para la comparación de los datos generados y los originales (como RMSE) resultan en valores muy bajos incluso cuando se compara el promedio de los datos con los originales, por ello se complementa el análisis con comparaciones gráficas.

Durante la búsqueda de arquitectura, todos los modelos se entrenaron durante 500 *epochs*, deteniendose por *early-stopping* si no hubo mejora durante 70 *epochs* e iniciado con la poda (explicada en la sección 5.3.1) luego de los 100 *epochs*, los parámetros explorados fueron los siguientes:

- **Embeddings**:
  - `embeddings_size` entre 0 y $0{,}9$
- **Bloques CNN**:
  - `num_cnn_layers` entre 0 y 10
  - `num_cnn_filters` entre 10 y 500
  - `cnn_kernel_size` entre 1 y 10
  - `cnn_dropout` entre 0 y $0{,}5$
  - `cnn_batch_norm` entre 0 y `num_cnn_layers`
  - `cnn_activation` relu, elu, selu, swish, sigmoid, tanh o linear
- **Bloques RNN**:
  - `num_rnn_layers` entre 0 y 10
  - `rnn_hidden_dim` entre 10 y 512
  - `rnn_dropout` entre 0 y $0{,}5$
  - `rnn_layer_norm` entre 0 y `num_rnn_layers`
  - `rnn_layer_type` GRU, LSTM, bidirectional GRU o bidirectional LSTM
- **Bloques DNN**:
  - `num_dnn_layers` entre 1 y 10
  - `dnn_hidden_dim` entre 10 y 2048
  - `dnn_dropout` entre 0 y $0{,}5$
  - `dnn_layer_norm` entre 0 y `num_head_layers`
  - `dnn_activation` relu, elu, selu, swish, sigmoid, tanh o linear
- **Bloques DNN cabezales**:
  - `num_head_layers` entre 1 y 10
  - `head_hidden_dim` entre 10 y 2048
  - `head_dropout` entre 0 y $0{,}5$
  - `head_layer_norm` entre 0 y `num_head_layers`
  - `head_activation` relu, elu, selu, swish, sigmoid, tanh o linear

Como ya se comentó en la sección 2.3.2, la búsqueda del parámetro *learning rate* se realizó al comienzo del entrenamiento de cada modelo, pudiendo variar entre $0{,}0001$ y 1.

En resumen, se crearon seis modelos distintos, uno por cada configuración espacial y problema a resolver, para cada grupo de experimentos se probaron las variantes necesarias para obtener un modelo que resultara satisfactorio, estas variantes se detallan a continuación en orden cronológico para mostrar la evolución y mejoría de los resultados.

6.2. Modelo *baseline*

Como modelo base contra el que buscar mejoras, se usó el modelo *Extremely Randomized Trees Regressor* (con implementación en la biblioteca sklearn[79]). Para hacer la comparación lo más justa posible, se creó este modelo usando el mismo sistema de autoML reemplazando el componente de ML, los parámetros explorados fueron los siguientes:

- `num_estimators` La cantidad de árboles en el modelo, entre 10 y 1000
- `max_depth` La profundidad máxima de cada árbol en el modelo, entre 10 y 500
- `min_samples_leaf` La cantidad mínima de ejemplos necesarios para marcar un nodo como hoja, entre 1 y 20
- `min_samples_split` La cantidad mínima de ejemplos necesarios para dividir un nodo, entre 2 y 20
- `max_features` La cantidad de *features* usadas para elegir un corte, $\sqrt{n}$ o $\log_2 n$ donde n es la cantidad de $features$ del conjunto

6.3. Predicción temporal por subsistema

En este grupo de experimentos se usó la agregación de datos temporal por subsistema (sección 4.6.3), es decir que es el grupo de experimentos con datos de menor granularidad geográfica, donde cada ejemplo representa la agregación de los datos para un subsistema.

6.3.1. Experimento 1

Para el experimento 1, las *features* usadas fueron:

- **dayofweek** El día de la semana en que se hicieron las consultas.
- **hour** La hora en que se hicieron las consultas.
- **minute** El minuto en que se hicieron las consultas.
- **subsistema** El subsistema educativo al que pertenece el centro (por ejemplo: CEIP, CES, Universidad).
- **agregación de la columna parsed_domain** La cantidad de consultas realizadas para cada uno de los dominios más importantes.
- **total de consultas** La cantidad de consultas realizadas en un momento dado.

La frecuencia temporal del conjunto fue 5 minutos.

El conjunto total contó con 28.864.073 ejemplos de los cuales 20.204.667 se usaron en el conjunto train (aproximadamente 70 %), 4.392.303 en el conjunto val (aproximadamente 15 %) y los restantes 4.267.103 en el conjunto test (aproximadamente 15 %).

Predicción de consultas DNS Los resultados del *baseline* y del modelo elegido para la predicción de consultas DNS se muestran en la figura 6.3.1 donde se comparan con los datos observados, se aprecia que el modelo creado se ajusta muy bien a los datos observados (incluso en los valores altos) y que el *baseline* no ajusta correctamente con valores muy bajos o muy altos, de todas formas, ambos modelos se desempeñan correctamente.

En la tabla 6.3.1 se detallan las métricas RMSE, Huber y Reversed Huber de las series ya mencionadas comparadas con los valores observados, se confirman los resultados ya mencionados y se observa que tanto para RMSE como para Huber, la comparación del error entre el baseline y el modelo generado no son significativas, esto puede dificultar el entrenamiento.

Como los resultados fueron muy buenos, no se realizaron más experimentos para este problema, el modelo final se compone de la siguiente estructura:

- **Embeddings**:
  - `embeddings_size` $0{,}7$
- **Bloques CNN**:
  - `num_cnn_layers` 2
  - `num_cnn_filters` 154
  - `cnn_kernel_size` 3
  - `cnn_dropout` $0{,}03$
  - `cnn_batch_norm` 2
  - `cnn_activation` selu
- **Bloques RNN**:
  - `num_rnn_layers` 2
  - `rnn_hidden_dim` 403
  - `rnn_dropout` $0{,}1$
  - `rnn_layer_norm` 1
  - `rnn_layer_type` bidirectional GRU
- **Bloques DNN**:
  - `num_dnn_layers` 5
  - `dnn_hidden_dim` 1050
  - `dnn_dropout` $0{,}16$
  - `dnn_layer_norm` 2
  - `dnn_activation` tanh
- **Bloques DNN cabezales**:
  - `num_head_layers` 4
  - `head_hidden_dim` 12
  - `head_dropout` $0{,}04$
  - `head_layer_norm` 3
  - `head_activation` swish

Transformación de consultas DNS en tráfico Los resultados del *baseline* y del modelo elegido para la transformación de consultas DNS en tráfico se muestran en la figura 6.3.2 (para el conjunto de *test*), se observa que la predicción del modelo (en verde) no estima correctamente los valores altos ($\geq 0{,}4$) pero igualmente mejora al *baseline* a excepción de las predicciones para el día 27 de noviembre (resaltado en rojo), si se comparan los datos con los observados en el número de consultas DNS de ese día, se observa que la predicción del modelo entrenado se ajusta más al comportamiento observado en esa serie temporal, el mismo fenómeno se registra para el día 7 de noviembre (resaltado en rojo).

En la tabla 6.3.2 se detallan las métricas RMSE, Huber y Reversed Huber de las series ya mencionadas comparadas con los valores observados, se observa que todas las métricas dan un valor de error más bajo para el *baseline* que para el modelo obtenido, se cree que esto se debe a los fenómenos comentados anteriormente.

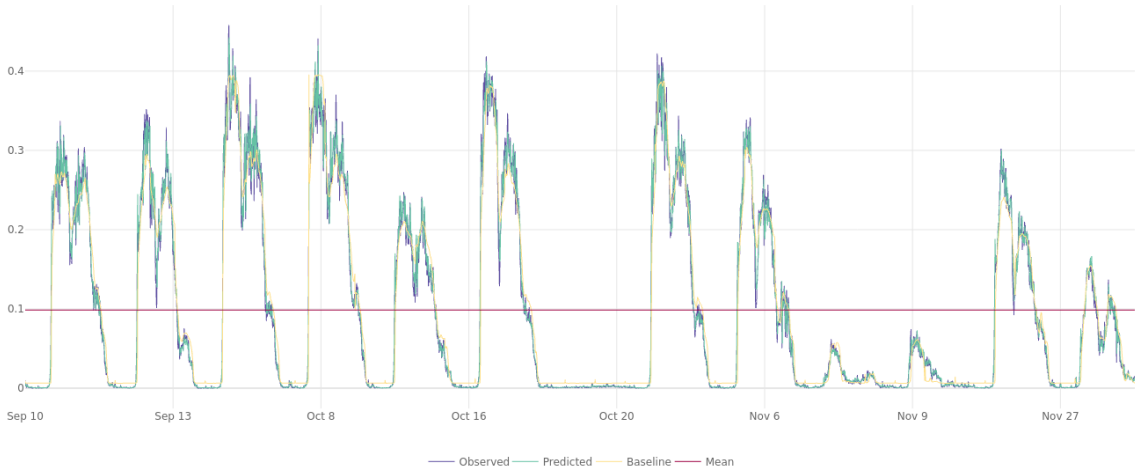


Figura 6.3.1: Comparación de datos observados en el conjunto de *test* (en azul) con el promedio de los observados (en rojo), las predicciones del *baseline* (en naranja) y del experimento 1 (en verde) para la predicción de consultas DNS sobre el conjunto agregado por subsistema.

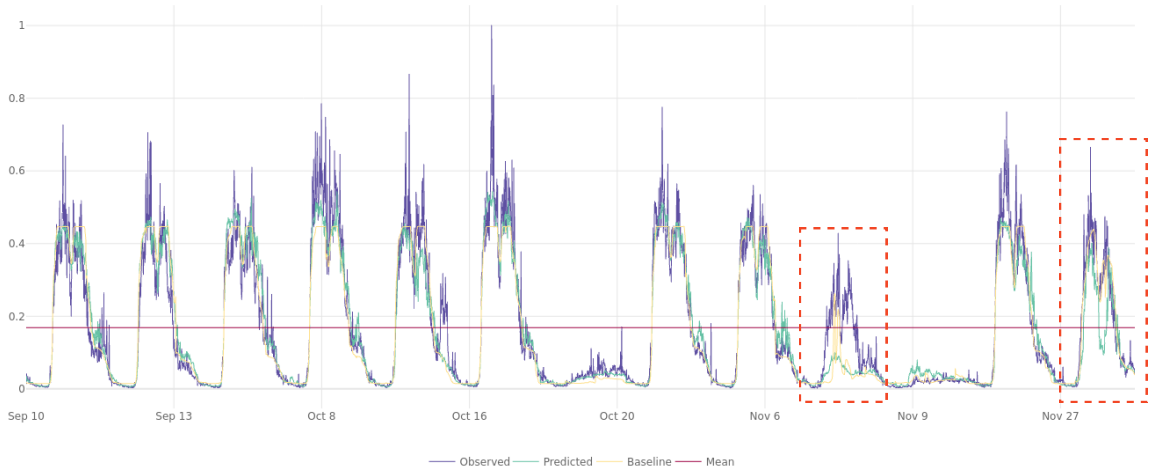


Figura 6.3.2: Comparación de datos observados en el conjunto de *test* (en azul) con el promedio de los observados (en rojo), las predicciones del *baseline* (en naranja) y del experimento 1 (en verde) para la transformación de consultas DNS en tráfico sobre el conjunto agregado por subsistema. Se resaltan casos particulares en rojo.

6.3.2. Experimento 2

El siguiente experimento se hizo replicando el anterior sobre el conjunto de datos con frecuencia temporal de 15 minutos y agregando las *features*:

- **EWMA de la agregación de la columna parsed_domain** EWMA de la cantidad de consultas realizadas para cada uno de los dominios más importantes.
- **EWMA total de consultas** EWMA de la cantidad de consultas realizadas en un momento dado.

El conjunto total contó con 11.631.790 ejemplos de los cuales 8.142.179 se usaron en el conjunto train (aproximadamente 70 %), 1.770.032 en el conjunto val (aproximadamente 15 %) y los restantes 1.719.579 en el conjunto test (aproximadamente 15 %).

Transformación de consultas DNS en tráfico Los resultados del *baseline* y del modelo elegido para la transformación de consultas DNS en tráfico se muestran en la figura 6.3.3 (para el conjunto de *test*), se observa que la predicción del modelo (en verde) estima bien los valores altos ($\geq 0{,}4$) (comparado con los resultados anteriores) mejorando la predicción del *baseline*, el modelo sobreestima algunos valores del 7 de noviembre y lo mismo ocurre con el *baseline* con los valores del 9 de noviembre (resaltados en rojo).

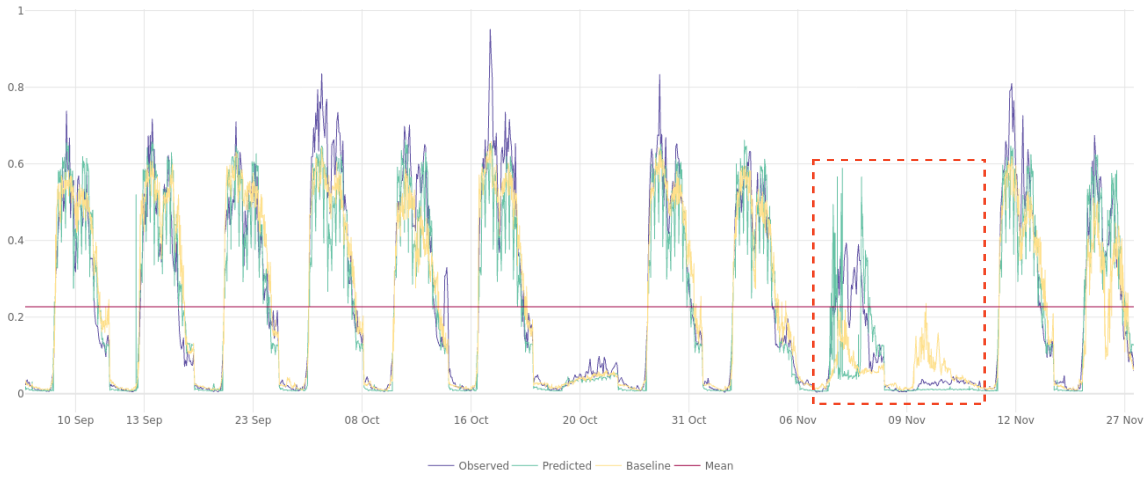


Figura 6.3.3: Comparación de datos observados en el conjunto de *test* (en azul) con el promedio de los observados (en rojo), las predicciones del *baseline* (en naranja) y del experimento 2 (en verde) para la transformación de consultas DNS en tráfico sobre el conjunto agregado por subsistema. Se resaltan casos particulares en rojo.

En la tabla 6.3.2 se detallan las métricas RMSE, Huber y Reversed Huber de las series ya mencionadas comparadas con los valores observados, se observa que para RMSE y Huber el modelo entrenado supera a los otros mientras que para Reversed Huber el valor más bajo lo genera el *baseline*, esto último puede deberse a los fenómenos comentados anteriormente.

Como el resultado del último experimento fue muy satisfactorio, no se ejecutaron más pruebas. La arquitectura seleccionada fue:

- **Embeddings**:
  - `embeddings_size` $0{,}4$
- **Bloques CNN**:
  - `num_cnn_layers` 6
  - `num_cnn_filters` 248
  - `cnn_kernel_size` 3
  - `cnn_dropout` $0{,}4$
  - `cnn_batch_norm` 2

  - `cnn_activation` elu

- **Bloques RNN**:
  - `num_rnn_layers` 2
  - `rnn_hidden_dim` 252
  - `rnn_dropout` $0{,}5$
  - `rnn_layer_norm` 0
  - `rnn_layer_type` bidirectional GRU
- **Bloques DNN**:
  - `num_dnn_layers` 5
  - `dnn_hidden_dim` 1449
  - `dnn_dropout` $0{,}13$
  - `dnn_layer_norm` 3
  - `dnn_activation` swish
- **Bloques DNN cabezales**:
  - `num_head_layers` 5
  - `head_hidden_dim` 393
  - `head_dropout` $0{,}5$
  - `head_layer_norm` 4
  - `head_activation` selu

Modelo	RMSE	Huber	Reversed Huber
Experimento 1 (mean)	$0{,}0140$	$0{,}0069$	$0{,}0817$
Experimento 1 (baseline)	$0{,}0005$	$0{,}0003$	$0{,}0148$
Experimento 1 (predicted)	$0{,}0002$	$0{,}0001$	$0{,}0080$

Tabla 6.3.1: Comparación de experimentos para la predicción de consultas DNS sobre el conjunto agregado por subsistema.

Modelo	RMSE	Huber	Reversed Huber
Experimento 1 (mean)	$0{,}0327$	$0{,}0147$	$0{,}0993$
Experimento 1 (baseline)	$0{,}0036$	$0{,}0018$	$0{,}0321$
Experimento 1 (predicted)	$0{,}0041$	$0{,}0020$	$0{,}0328$
Experimento 2 (mean)	$0{,}0575$	$0{,}0245$	$0{,}0648$
Experimento 2 (baseline)	$0{,}0080$	$0{,}0038$	$0{,}0411$
Experimento 2 (predicted)	$0{,}0067$	$0{,}0033$	$0{,}0437$

Tabla 6.3.2: Comparación de experimentos para la transformación de consultas DNS en tráfico sobre el conjunto agregado subsistema.

6.3.3. Conclusiones

Para este conjunto de datos, se observa que la predicción de consultas DNS es muy buena tanto para el *baseline* como para el modelo entrenado, evitando la necesidad de usar conjuntos con mayor granularidad temporal y prediciendo de forma muy correcta valores altos.

Para la transformación de consultas DNS en tráfico, se observa que el conjunto de datos de frecuencia de 5 minutos no obtiene resultados muy buenos pero que mejora con la frecuencia de 15 minutos. Incluso en ese conjunto, valores mayores a $0{,}6$ presentan dificultad en la predicción pero en general se llega a un modelo bastante bueno.

6.4. Predicción temporal por departamento y subsistema

En este grupo de experimentos se usó la agregación de datos temporal por departamento y subsistema (sección 4.6.3), es decir que es un grupo de experimentos con datos con granularidad espacial intermedia, donde cada ejemplo representa la agregación de los datos para un subsistema dentro de un departamento.

6.4.1. Experimento 1

Para el experimento 1, las *features* usadas fueron:

- **dayofweek** El día de la semana en que se hicieron las consultas.
- **hour** La hora en que se hicieron las consultas.
- **minute** El minuto en que se hicieron las consultas.
- **subsistema** El subsistema educativo al que pertenece el centro (por ejemplo: CEIP, CES, Universidad).
- **departamento** El departamento al que pertenece el centro.
- **agregación de la columna parsed_domain** La cantidad de consultas realizadas para cada uno de los dominios más importantes.
- **total de consultas** La cantidad de consultas realizadas en un momento dado.

La frecuencia temporal del conjunto fue 5 minutos (análogo al experimento 1 de la sección anterior).

El conjunto total contó con 146.124.380 ejemplos de los cuales 102.286.129 se usaron en el conjunto train (aproximadamente 70 %), 22.236.038 en el conjunto val (aproximadamente 15 %) y los restantes 21.602.213 en el conjunto test (aproximadamente 15 %).

Predicción de consultas DNS Los resultados del *baseline* y del modelo elegido para la predicción de consultas DNS se muestran en la figura 6.4.1 donde se comparan con los datos observados, se aprecia que tanto el modelo creado como el *baseline* ajustan muy bien a los datos observados (incluso en los valores altos).

En la tabla 6.4.1 se detallan las métricas RMSE, Huber y Reversed Huber de las series ya mencionadas comparadas con los valores observados, se confirman los resultados ya mencionados y se observa que el modelo generado es el que obtiene valores más bajos en todos los casos.

Como los resultados fueron muy buenos, no se realizaron más experimentos para este problema, el modelo final se compone de la siguiente estructura:

- **Embeddings**:
  - `embeddings_size` $0{,}2$

- **Bloques CNN**:
  - `num_cnn_layers` 6
  - `num_cnn_filters` 163
  - `cnn_kernel_size` 5
  - `cnn_dropout` $0{,}003$
  - `cnn_batch_norm` 6
  - `cnn_activation` relu
- **Bloques RNN**:
  - `num_rnn_layers` 3
  - `rnn_hidden_dim` 251
  - `rnn_dropout` $0{,}14$
  - `rnn_layer_norm` 2
  - `rnn_layer_type` GRU
- **Bloques DNN**:
  - `num_dnn_layers` 2
  - `dnn_hidden_dim` 710
  - `dnn_dropout` $0{,}26$
  - `dnn_layer_norm` 2
  - `dnn_activation` tanh
- **Bloques DNN cabezales**:
  - `num_head_layers` 2
  - `head_hidden_dim` 658
  - `head_dropout` $0{,}15$
  - `head_layer_norm` 2
  - `head_activation` selu

Transformación de consultas DNS en tráfico Los resultados del *baseline* y del modelo elegido para la transformación de consultas DNS en tráfico se muestran en la figura 6.4.2 (para el conjunto de *test*), se observa que la predicción del modelo (en verde) nuevamente no estima correctamente los valores altos ($\geq 0{,}4$) y parecería estimar por debajo muchos valores que (en promedio) el *baseline* sobreestima, se ve un caso particular el día 7 de noviembre (resaltado en rojo) donde el *baseline* estima muy por debajo los valores (siguiendo las tendencias de las consultas DNS) pero que el modelo entrenado estima de mejor forma.

En la tabla 6.4.2 se detallan las métricas RMSE, Huber y Reversed Huber de las series ya mencionadas comparadas con los valores observados, se observa que todas las métricas dan un valor de error más bajo para el modelo obtenido.

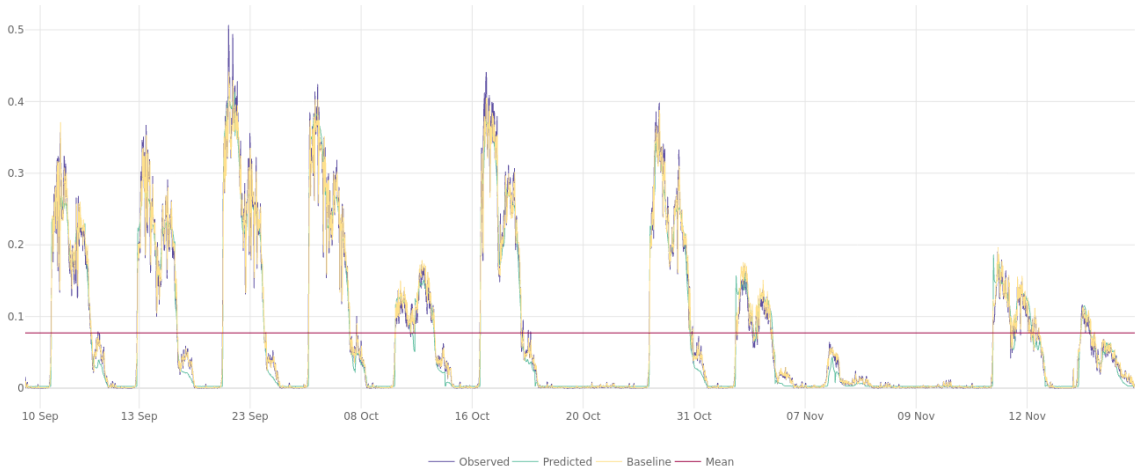


Figura 6.4.1: Comparación de datos observados en el conjunto de *test* (en azul) con el promedio de los observados (en rojo), las predicciones del *baseline* (en naranja) y del experimento 1 (en verde) para la predicción de consultas DNS sobre el conjunto agregado por departamento y subsistema.

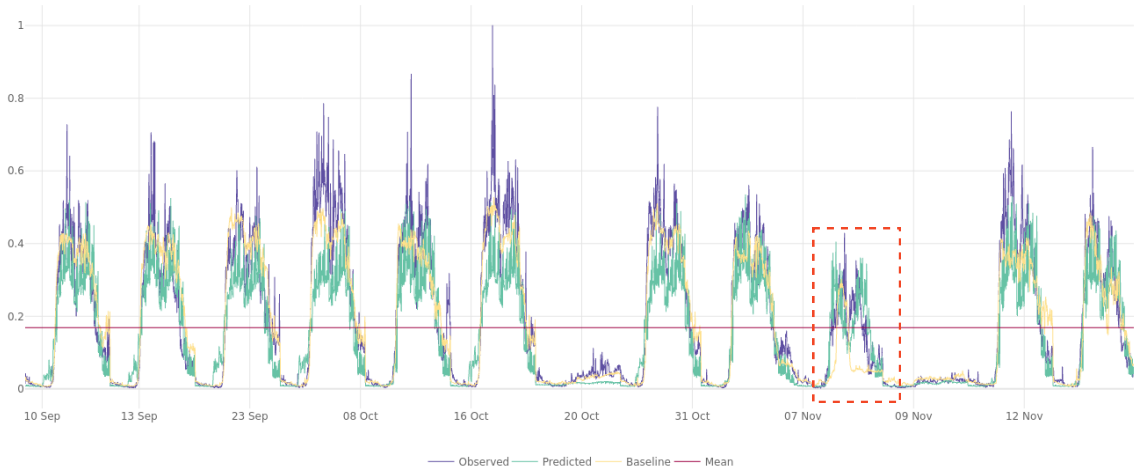


Figura 6.4.2: Comparación de datos observados en el conjunto de *test* (en azul) con el promedio de los observados (en rojo), las predicciones del *baseline* (en naranja) y del experimento 1 (en verde) para la transformación de consultas DNS en tráfico sobre el conjunto agregado por departamento y subsistema. Se resaltan casos particulares en rojo.

6.4.2. Experimento 2

El siguiente experimento se hizo replicando el anterior sobre el conjunto de datos con frecuencia temporal de 15 minutos y agregando las features:

- **EWMA de la agregación de la columna parsed_domain** EWMA de la cantidad de consultas realizadas para cada uno de los dominios más importantes.
- **EWMA total de consultas** EWMA de la cantidad de consultas realizadas en un momento dado.

El conjunto total contó con 48.708.125 ejemplos de los cuales 34.095.376 se usaron en el conjunto train (aproximadamente 70 %), 7.412.012 en el conjunto val (aproximadamente 15 %) y los restantes 7.200.737 en el conjunto test (aproximadamente 15 %).

Transformación de consultas DNS en tráfico Los resultados del *baseline* y del modelo elegido para la transformación de consultas DNS en tráfico se muestran en la figura 6.4.3 (para el conjunto de *test*), se observa que la predicción del modelo (en verde) estima mejor que el *baseline* los valores altos ($\geq 0{,}4$), también estima por debajo los valores que se dan cuando comienza el aumento en la serie temporal. El *baseline* no logra predecir los valores del 7 de noviembre en comparación con el modelo generado.

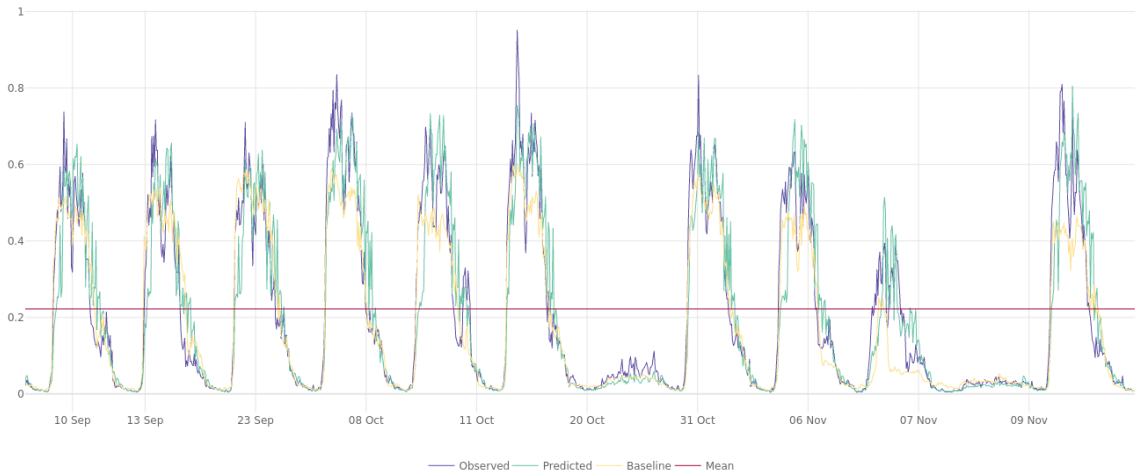


Figura 6.4.3: Comparación de datos observados en el conjunto de *test* (en azul) con el promedio de los observados (en rojo), las predicciones del *baseline* (en naranja) y del experimento 2 (en verde) para la transformación de consultas DNS en tráfico sobre el conjunto agregado por departamento y subsistema.

En la tabla 6.4.2 se detallan las métricas RMSE, Huber y Reversed Huber de las series ya mencionadas comparadas con los valores observados, se observa que para todas las métricas el modelo entrenado supera al resto, en particular para Reversed Huber la diferencia entre el *baseline* y el modelo entrenado es menor que el resto.

Como el resultado del último experimento fue satisfactorio, no se ejecutaron más pruebas. La arquitectura seleccionada fue:

- **Embeddings**:
  - `embeddings_size` $0{,}2$
- **Bloques CNN**:
  - `num_cnn_layers` 6
  - `num_cnn_filters` 163
  - `cnn_kernel_size` 5
  - `cnn_dropout` $0{,}004$
  - `cnn_batch_norm` 6

- `cnn_activation` relu

- **Bloques RNN**:
  - `num_rnn_layers` 3
  - `rnn_hidden_dim` 251
  - `rnn_dropout` $0{,}14$
  - `rnn_layer_norm` 2
  - `rnn_layer_type` GRU
- **Bloques DNN**:
  - `num_dnn_layers` 2
  - `dnn_hidden_dim` 710
  - `dnn_dropout` $0{,}26$
  - `dnn_layer_norm` 2
  - `dnn_activation` tanh
- **Bloques DNN cabezales**:
  - `num_head_layers` 2
  - `head_hidden_dim` 658
  - `head_dropout` $0{,}15$
  - `head_layer_norm` 2
  - `head_activation` selu

Modelo	RMSE	Huber	Reversed Huber
Experimento 1 (mean)	$0{,}0114$	$0{,}0055$	$0{,}0659$
Experimento 1 (baseline)	$0{,}0004$	$0{,}0002$	$0{,}0118$
Experimento 1 (predicted)	$0{,}0003$	$0{,}0002$	$0{,}0118$

Tabla 6.4.1: Comparación de experimentos para la predicción de consultas DNS sobre el conjunto agregado por departamento y subsistema.

Modelo	RMSE	Huber	Reversed Huber
Experimento 1 (mean)	$0{,}0354$	$0{,}0159$	$0{,}0984$
Experimento 1 (baseline)	$0{,}0099$	$0{,}0046$	$0{,}0461$
Experimento 1 (predicted)	$0{,}0043$	$0{,}0021$	$0{,}0351$
Experimento 2 (mean)	$0{,}0407$	$0{,}0219$	$0{,}0667$
Experimento 2 (baseline)	$0{,}0081$	$0{,}0040$	$0{,}0456$
Experimento 2 (predicted)	$0{,}0054$	$0{,}0026$	$0{,}0417$

Tabla 6.4.2: Comparación de experimentos para la transformación de consultas DNS en tráfico sobre el conjunto agregado por departamento y subsistema.

6.4.3. Conclusiones

Para este conjunto de datos, se observa que la predicción de consultas DNS nuevamente muy buena en el primer experimento siendo competitivo tanto el *baseline* como el modelo entrenado.

Para la transformación de consultas DNS en tráfico, se observa que el conjunto de datos de frecuencia de 5 minutos no obtiene resultados muy buenos pero que mejora con la frecuencia de 15 minutos. Incluso en ese conjunto, valores mayores a $0{,}6$ presentan dificultad en la predicción pero en general se llega a un modelo bastante bueno.

6.5. Predicción temporal por ruee

En este grupo de experimentos se usó la agregación de datos temporal por ruee (sección 4.6.1), es decir que es el grupo de experimentos con datos de mayor granularidad geográfica, donde cada ejemplo representa la agregación de los datos para un local.

6.5.1. Experimento 1

Para el experimento 1, las *features* usadas fueron:

- **dayofweek** El día de la semana en que se hicieron las consultas.
- **hour** La hora en que se hicieron las consultas.
- **minute** El minuto en que se hicieron las consultas.
- **subsistema** El subsistema educativo al que pertenece el centro (por ejemplo: CEIP, CES, Universidad).
- **departamento** El departamento al que pertenece el centro.
- **ruee** El identificador del centro educativo.
- **agregación de la columna parsed_domain** La cantidad de consultas realizadas para cada uno de los dominios más importantes.
- **total de consultas** La cantidad de consultas realizadas en un momento dado.

La frecuencia temporal del conjunto fue 5 minutos al igual que en los experimentos anteriores.

El conjunto total contó con 467.598.022 ejemplos de los cuales 327.315.615 se usaron en el conjunto train (aproximadamente 70 %), 71.155.323 en el conjunto val (aproximadamente 15 %) y los restantes 69.127.084 en el conjunto test (aproximadamente 15 %).

Predicción de consultas DNS En primera instancia se buscó el mejor modelo encargado de la predicción de consultas DNS, los resultados del *baseline* y del modelo elegido para la predicción de consultas DNS se muestran en la figura 6.5.1 donde se comparan con los datos observados, se aprecia que tanto el modelo creado como el *baseline* ajustan muy bien a los datos observados (incluso en los valores altos).

En la tabla 6.5.1 se detallan las métricas RMSE, Huber y Reversed Huber de las series ya mencionadas comparadas con los valores observados, en todos los casos el modelo observado no logra superar al *baseline* a pesar de eso, en la figura anterior se aprecia que logra predecir de mejor forma los valores altos, este fenómeno puede darse porque el promedio de los valores es muy bajo ($0{,}038$).

Como los resultados fueron muy buenos, no se realizaron más experimentos para este problema, el modelo final se compone de la siguiente estructura:

- **Embeddings**:
  - `embeddings_size` $0{,}6$
- **Bloques CNN**:
  - `num_cnn_layers` 1
  - `num_cnn_filters` 31
  - `cnn_kernel_size` 1
  - `cnn_dropout` $0{,}09$
  - `cnn_batch_norm` 1
  - `cnn_activation` elu
- **Bloques RNN**:
  - `num_rnn_layers` 5
  - `rnn_hidden_dim` 202
  - `rnn_dropout` $0{,}37$
  - `rnn_layer_norm` 5
  - `rnn_layer_type` GRU
- **Bloques DNN**:
  - `num_dnn_layers` 2
  - `dnn_hidden_dim` 203
  - `dnn_dropout` $0{,}12$
  - `dnn_layer_norm` 2
  - `dnn_activation` swish
- **Bloques DNN cabezales**:
  - `num_head_layers` 2
  - `head_hidden_dim` 43
  - `head_dropout` $0{,}32$
  - `head_layer_norm` 2
  - `head_activation` elu

Transformación de consultas DNS en tráfico Los resultados del *baseline* y del modelo elegido para la transformación de consultas DNS en tráfico se muestran en la figura 6.5.2 (para el conjunto de *test*), se observa que ningún modelo logra predecir de forma correcta valores mayores a $0{,}15$, para intentar mejorar este fenómeno se prueba en el siguiente experimento con los datos con frecuencia de 15 minutos.

En la tabla 6.5.2 se detallan las métricas RMSE, Huber y Reversed Huber de las series ya mencionadas comparadas con los valores observados, se observa que todas las métricas dan un valor de error más bajo para el *baseline* que para el modelo obtenido, se cree que esto se debe a los fenómenos comentados anteriormente.

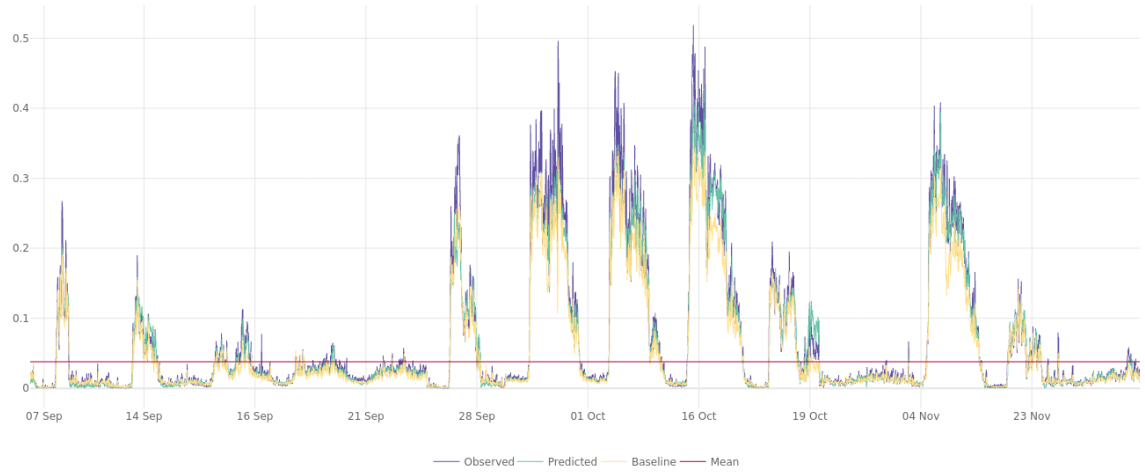


Figura 6.5.1: Comparación de datos observados en el conjunto de *test* (en azul) con el promedio de los observados (en rojo), las predicciones del *baseline* (en naranja) y del experimento 1 (en verde) para la predicción de consultas DNS sobre el conjunto de agregado por ruee.

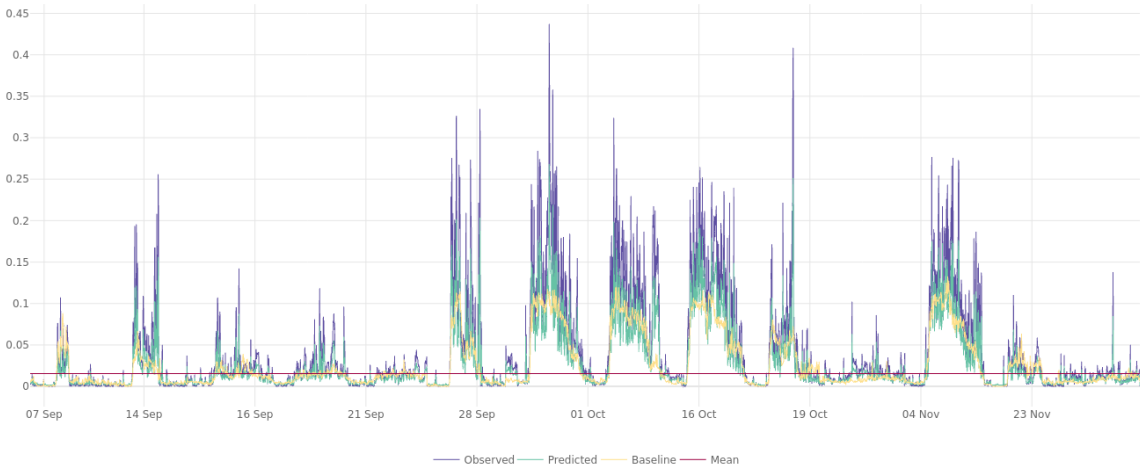


Figura 6.5.2: Comparación de datos observados en el conjunto de *test* (en azul) con el promedio de los observados (en rojo), las predicciones del *baseline* (en naranja) y del experimento 1 (en verde) para la transformación de consultas DNS en tráfico sobre el conjunto agregado por ruee.

6.5.2. Experimento 2

El siguiente experimento se hizo replicando el anterior sobre el conjunto de datos con frecuencia temporal de 15 minutos y agregando las *features*:

- **EWMA de la agregación de la columna parsed_domain** EWMA de la cantidad de consultas realizadas para cada uno de los dominios más importantes.
- **EWMA total de consultas** EWMA de la cantidad de consultas realizadas en un momento dado.

El conjunto total contó con 133.599.434 ejemplos de los cuales 93.518.747 se usaron en el conjunto train (aproximadamente 70 %), 20.330.092 en el conjunto val (aproximadamente 15 %) y los restantes 19.750.595 en el conjunto test (aproximadamente 15 %).

Transformación de consultas DNS en tráfico Los resultados del *baseline* y del modelo elegido para la transformación de consultas DNS en tráfico se muestran en la figura 6.5.3 (para el conjunto de *test*), se observa que la predicción del modelo (en verde) logra estimar más correctamente los valores altos ($\geq 0{,}3$) (comparado con los resultados anteriores) mejorando la predicción del *baseline*, ninguno de los modelos logra llegar al máximo y ambos se comportan correctamente con los valores bajos.

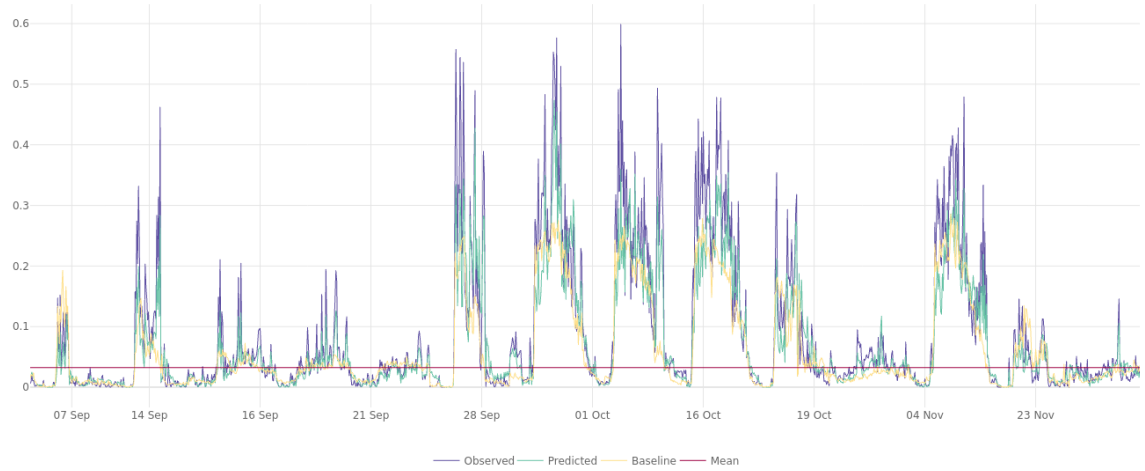


Figura 6.5.3: Comparación de datos observados en el conjunto de *test* (en azul) con el promedio de los observados (en rojo), las predicciones del *baseline* (en naranja) y del experimento 2 (en verde) para la transformación de consultas DNS en tráfico sobre el conjunto agregado por ruee.

En la tabla 6.5.2 se detallan las métricas RMSE, Huber y Reversed Huber de las series ya mencionadas comparadas con los valores observados, nuevamente el *baseline* es el mejor evaluado a pesar de que se observa que predice de peor forma los valores altos, este comportamiento se le atribuye a que los datos (en promedio) son muy bajos.

Como el resultado del último experimento fue satisfactorio, no se ejecutaron más pruebas. La arquitectura seleccionada fue:

- **Embeddings**:
  - `embeddings_size` $0{,}2$
- **Bloques CNN**:
  - `num_cnn_layers` 3
  - `num_cnn_filters` 58
  - `cnn_kernel_size` 7
  - `cnn_dropout` $0{,}21$
  - `cnn_batch_norm` 3

- `cnn_activation` selu

- **Bloques RNN**:
  - `num_rnn_layers` 4
  - `rnn_hidden_dim` 297
  - `rnn_dropout` $0{,}03$
  - `rnn_layer_norm` 2
  - `rnn_layer_type` LSTM
- **Bloques DNN**:
  - `num_dnn_layers` 3
  - `dnn_hidden_dim` 853
  - `dnn_dropout` $0{,}15$
  - `dnn_layer_norm` 2
  - `dnn_activation` elu
- **Bloques DNN cabezales**:
  - `num_head_layers` 0

Modelo	RMSE	Huber	Reversed Huber
Experimento 1 (mean)	$0{,}0045$	$0{,}0022$	$0{,}0393$
Experimento 1 (baseline)	$0{,}0003$	$0{,}0002$	$0{,}0081$
Experimento 1 (predicted)	$0{,}0004$	$0{,}002$	$0{,}0094$

Tabla 6.5.1: Comparación de experimentos para la predicción de consultas DNS sobre el conjunto agregado por ruee.

Modelo	RMSE	Huber	Reversed Huber
Experimento 1 (mean)	$0{,}0010$	$0{,}0005$	$0{,}0190$
Experimento 1 (baseline)	$0{,}0001$	$0{,}0001$	$0{,}0055$
Experimento 1 (predicted)	$0{,}0005$	$0{,}0001$	$0{,}0088$
Experimento 2 (mean)	$0{,}0045$	$0{,}0021$	$0{,}0351$
Experimento 2 (baseline)	$0{,}0007$	$0{,}0004$	$0{,}0135$
Experimento 2 (predicted)	$0{,}0015$	$0{,}0007$	$0{,}0140$

Tabla 6.5.2: Comparación de experimentos para la transformación de consultas DNS en tráfico sobre el conjunto agregado ruee.

6.5.3. Conclusiones

Para este conjunto de datos, se observa ambos problemas presentaron mayor dificultad que en los anteriores. En particular, la transformación de consultas DNS en tráfico fue más desafiante siendo que el *baseline* pareció comportarse de peor forma en las comparaciones gráficas pero no así en las evaluaciones de las métricas.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 7

Conclusiones y trabajo futuro

En este capitulo se discuten las conclusiones que pudieron obtenerse de este proyecto, tanto del procesamiento de grandes volúmenes de datos como de la generación de modelos capaces de resolver los problemas planteados. También se comenta el posible trabajo futuro.

7.1. Sobre el procesamiento de *Big Data*

La plataforma de Hortonworks y más en general la herramienta Spark, hacen que el manejo y transformación de grandes volúmenes de datos sea posible pero tanto el uso de pyspark como la ejecución y configuración de las tareas a ejecutar tienen una curva de aprendizaje elevada, esto se observa en el cronograma de ejecución del proyecto (sección 3.3). La integración con bibliotecas como Koalas es muy útil para simplificar la curva de aprendizaje antes nombrada pero, como la integración no es completa, existen casos de funciones específicas que deben ejecutarse usando pyspark. Dicho esto, lograr un uso de recursos y tiempo óptimos no es tarea sencilla y requiere entender la plataforma y sus configuraciones.

Trabajar en la escala de datos del proyecto obliga no solo a entender el uso de las herramientas de análisis y procesamiento sino también a entender qué formato de archivo, particiones y otras características relativas a la infraestructura producirán un mejor desempeño en la lectura y escritura durante la ejecución de los algoritmos. Esto se reflejó en la necesidad de crear scripts y utilidades que ayudasen a facilitar la tarea del pasaje de datos y comunicación con los distintos clusters.

7.2. Sobre el sistema de autoML

En este proyecto se logró implementar de manera satisfactoria el sistema de autoML diseñado, este sistema no solo permitió obtener el mejor modelo posible de *deep learning*, sino que también se pudo utilizar con algoritmos tradicionales de aprendizaje supervisado (sección 6.2).

La integración de la herramienta de Spark con otras de *deep learning* no existía al momento del proyecto de manera oficial (como si lo hace para algoritmos de *machine learning* tradicional) y debía ser lograda a través de bibliotecas de terceros. La experiencia con estas herramientas no fue satisfactoria tanto por problemas de estabilidad como por problemas de tiempo, en cuanto al tiempo, es esperable que una cantidad moderada de procesadores como la que existe en el cluster de Ceibal no fuera competencia contra GPUs cuya arquitectura se adapta de mejor manera al tipo de trabajo que hacen los modelos de redes neuronales, esta suposición fue confirmada ya que no

se logró igualar la velocidad contra un solo GPU. En cuanto a la cuestión de la estabilidad, no se logró comprender completamente si se debió a que la instancia de la plataforma en el cluster de Ceibal no estaba correctamente configurada para tareas de muy largo procesamiento y uso intensivo de CPUs o si se debió a las bibliotecas que se usaron (sección 5.2.3).

7.3. Sobre la predicción de cantidad de consultas DNS

Sobre la creación de modelos capaces de predecir consultas DNS futuras a partir de las actuales, en todos los casos se logró llegar a modelos muy satisfactorios, la granularidad temporal no aportó complejidad mientras que la granularidad espacial dificultó la tarea solo en el caso más específico (sección 6.5.1). Dado el tiempo de entrenamiento y la cantidad de recursos consumidos tanto por los *baselines* como por los modelos generados, donde, en promedio los modelos de redes neuronales requirieron de al menos 24 horas de entrenamiento y el uso de GPUs mientras que los *baselines* requirieron en promedio 1 hora de entrenamiento y el uso de CPUs, no se considera meritorio el uso de redes neuronales para este problema.

7.4. Sobre la transformación de cantidad de consultas DNS en volumen de tráfico

Respecto a los modelos entrenados para transformar consultas DNS en volumen tráfico, se logró crear modelos que superan a cada *baseline* realista utilizado, existiendo algunos casos donde esta ventaja se contradice en la comparación gráfica y a través de métricas. Se observó que la frecuencia temporal usada puede influir en la dificultad del problema, en particular, se concluye que agregar los datos hasta una frecuencia razonable (15 minutos) ayudó a la obtención de modelos más rápidamente por utilizar menor cantidad de datos y sin perder poder predictivo.

Otro punto de discusión es la influencia del nivel de granularidad espacial de los datos en la dificultad del problema, cuanta mayor especificidad se usa, más problemática se vuelve la predicción. En particular, transformar las consultas para cada local es más difícil que transformar las consultas de un departamento y/o subsistema. Una hipótesis conjeturada es que este fenómeno se da por el aumento importante de datos para los que el tráfico total es menor al 20 % del tráfico que se da en los picos. En conclusión, la solución al problema puede ser vista desde un punto más global, incluyendo varios modelos a distintos niveles que aporten varios focos de atención, dentro de este grupo de modelos, los más generales (datos por subsistema o por departamento y subsistema) tienen mayor nivel de confiabilidad y los más específicos aportan una mirada más enfocada en detectar qué local puede ser problemático en vez de dar una cifra concreta de tráfico.

7.5. Trabajo futuro

En ambos casos, se plantea como trabajo futuro profundizar en la desagregación por dominios específicos tanto la predicción de tráfico como cantidad de consultas DNS, de esta forma se permitiría a los administradores de red entender de mejor forma el comportamiento de los usuarios y prepararse ante posibles sobrecargas, para la realización de este trabajo el problema podría ser planteado de forma distinta pudiendo resolverse como un problema de clasificación.

También se plantea el uso de *Graph Neural Networks* (GNN) para continuar la línea de investigación de transformación de consultas DNS en volúmen de tráfico, estos modelos podrían mejorar la predicción ya que logran

incorporar la topografía de red en su estructura interna[80], se observa que el uso de esta nueva metodología ocasionaría repensar las últimas etapas del procesamiento de los datos.

Esta página ha sido intencionalmente dejada en blanco.

Referencias

[1] *Plan Ceibal*. 2021. url: https://www.ceibal.edu.uy/es/institucional (visitado 15-12-2021).

[2] *Plan Ceibal en cifras*. 2021. url: https://www.ceibal.edu.uy/es/articulo/ceibal-en-cifras (visitado 15-12-2021).

[3] *Cisco Umbrella Documentation*. url: https://docs.umbrella.com/ (visitado 15-12-2021).

[4] James F. Kurose y Keith W. Ross. *Computer Networking: A Top-Down Approach (6th Edition)*. 6th. Pearson, 2012. isbn: 0132856204.

[5] *ntopng Documentation*. url: https://www.ntop.org/guides/ntopng/ (visitado 15-12-2021).

[6] *Hortonworks Data Platform, Cloudera*. url: https://www.cloudera.com/products/hdp.html (visitado 15-12-2021).

[7] *Apache HDFS documentation*. url: https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html (visitado 15-12-2021).

[8] *Apache YARN documentation*. url: https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html (visitado 15-12-2021).

[9] *Apache Hive documentation*. url: https://cwiki.apache.org/confluence/display/Hive/Home (visitado 15-12-2021).

[10] *Apache Spark documentation*. url: https://spark.apache.org/docs/latest/ (visitado 15-12-2021).

[11] *Apache Zeppelin documentation*. url: https://zeppelin.apache.org/docs/0.8.2/ (visitado 15-12-2021).

[12] *Apache Arrow*. url: https://arrow.apache.org/docs/index.html (visitado 15-12-2021).

[13] *Apache Ambari documentation*. url: https://cwiki.apache.org/confluence/display/AMBARI/Ambari (visitado 15-12-2021).

[14] *Sandbox Deployment and Install Guide*. url: https://www.cloudera.com/tutorials/sandbox-deployment-and-install-guide/3.html (visitado 15-12-2021).

[15] Shaker H. Ali El-Sappagh, Abdeltawab M. Ahmed Hendawi y Ali Hamed El Bastawissy. "A proposed model for data warehouse ETL processes". En: *Journal of King Saud University - Computer and Information Sciences* 23.2 (2011), pp. 91-104. issn: 1319-1578. doi: https://doi.org/10.1016/j.jksuci.2011.05.005. url: https://www.sciencedirect.com/science/article/pii/S131915781100019X.

[16] Bruce L Bowerman, Richard T O'Connell y Anne B Koehler. *Forecasting, time series, and regression: an applied approach*. Vol. 4. South-Western Pub, 2005.

[17] W. Schmid. "On EWMA Charts for Time Series". En: *Frontiers in Statistical Quality Control*. Ed. por Hans-Joachim Lenz y Peter-Theodor Wilrich. Heidelberg: Physica-Verlag HD, 1997, pp. 115-137. isbn: 978-3-642-59239-3.

[18] Tom M. Mitchell. *Machine Learning*. New York: McGraw-Hill, 1997. isbn: 978-0-07-042807-2.

[19] Ian Goodfellow, Yoshua Bengio y Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[20] "Mean Squared Error". En: *Encyclopedia of Machine Learning*. Ed. por Claude Sammut y Geoffrey I. Webb. Boston, MA: Springer US, 2010, pp. 653-653. isbn: 978-0-387-30164-8. doi: 10.1007/978-0-387-30164-8_528. url: https://doi.org/10.1007/978-0-387-30164-8_528.

[21] Peter J. Huber. "Robust Estimation of a Location Parameter". En: *Breakthroughs in Statistics: Methodology and Distribution*. Ed. por Samuel Kotz y Norman L. Johnson. New York, NY: Springer New York, 1992, pp. 492-518. isbn: 978-1-4612-4380-9. doi: 10.1007/978-1-4612-4380-9_35. url: https://doi.org/10.1007/978-1-4612-4380-9_35.

[22] Art Owen. "A robust hybrid of lasso and ridge regression". En: *Contemp. Math.* 443 (enero de 2007). doi: 10.1090/conm/443/08555.

[23] Bradley C. Boehmke y Brandon M. Greenwell. "Hands-On Machine Learning with R". En: 2019.

[24] Lutz Prechelt. "Early Stopping-But When?" En: *Neural Networks: Tricks of the Trade*. Ed. por Genevieve B. Orr y Klaus-Robert Müller. Vol. 1524. Lecture Notes in Computer Science. Springer, 1996, pp. 55-69. isbn: 3-540-65311-2. url: http://dblp.uni-trier.de/db/conf/nips/nips1996.html#Prechelt96.

[25] Leslie N. Smith. *Cyclical Learning Rates for Training Neural Networks*. 2017. arXiv: 1506.01186 [cs.CV].

[26] *Setting the learning rate of your neural network*. url: https://www.jeremyjordan.me/nn-learning-rate/ (visitado 15-12-2021).

[27] Cheng Guo y Felix Berkhahn. "Entity Embeddings of Categorical Variables". En: (abril de 2016).

[28] Ye Zhang y Byron Wallace. "A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification". En: *arXiv preprint arXiv:1510.03820* (2015).

[29] Sepp Hochreiter y Jürgen Schmidhuber. "Long Short-term Memory". En: *Neural computation* 9 (diciembre de 1997), pp. 1735-80. doi: 10.1162/neco.1997.9.8.1735.

[30] Junyoung Chung et al. *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling*. 2014. doi: 10.48550/ARXIV.1412.3555. url: https://arxiv.org/abs/1412.3555.

[31] Shaeke Salman y Xiuwen Liu. *Overfitting Mechanism and Avoidance in Deep Neural Networks*. 2019. arXiv: 1901.06566 [cs.LG].

[32] Twan van Laarhoven. "L2 Regularization versus Batch and Weight Normalization". En: *ArXiv* abs/1706.05350 (2017).

[33] *Distributed Training with Tensorflow*. url: https://medium.com/@deepakec1031/distributed-training-with-tensorflow-1e586d7fbcd6 (visitado 15-12-2021).

[34] *Distributed training with TensorFlow and Keras*. url: https://www.tensorflow.org/guide/distributed_training (visitado 15-12-2021).

[35] Leo Breiman. "Random Forests". En: *Machine Learning* 45 (2004), pp. 5-32.

[36] Leo Breiman, Jerome Friedman, Charles J. Stone and R.A. Olshen. *Classification and Regression Trees*. Chapman y Hall/CRC, 1984.

[37] Pierre Geurts, Damien Ernst y Louis Wehenkel. "Extremely randomized trees". En: *Machine learning* 63.1 (2006), pp. 3-42.

[38] *TensorFlow documentation*. url: https://www.tensorflow.org/api_docs/ (visitado 15-12-2021).

[39] *Scikit-Learn, Machine Learning in Python*. url: https://scikit-learn.org/stable/ (visitado 15-12-2021).

[40] James Bergstra y Yoshua Bengio. "Random search for hyper-parameter optimization." En: *Journal of machine learning research* 13.2 (2012).

[41] *Hyperparameter Tuning Black Magic*. url: https://community.alteryx.com/t5/Data-Science/Hyperparameter-Tuning-Black-Magic/ba-p/449289 (visitado 15-12-2021).

[42] James Bergstra et al. "Algorithms for hyper-parameter optimization". En: *25th annual conference on neural information processing systems (NIPS 2011)*. Vol. 24. Neural Information Processing Systems Foundation. 2011.

[43] Geoffrey I. Webb. "Bayes' Rule". En: *Encyclopedia of Machine Learning and Data Mining*. Ed. por Claude Sammut y Geoffrey I. Webb. Boston, MA: Springer US, 2017, pp. 99-99. isbn: 978-1-4899-7687-1. doi: 10.1007/978-1-4899-7687-1_21. url: https://doi.org/10.1007/978-1-4899-7687-1_21.

[44] J. Bergstra, D. Yamins y D. D. Cox. "Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures". En: *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*. ICML'13. Atlanta, GA, USA: JMLR.org, 2013, I–115–I-123.

[45] Daniel Golovin et al., eds. *Google Vizier: A Service for Black-Box Optimization*. 2017. url: http://www.kdd.org/kdd2017/papers/view/google-vizier-a-service-for-black-box-optimization.

[46] Liam Li et al. *A System for Massively Parallel Hyperparameter Tuning*. 2020. arXiv: 1810.05934 [cs.LG].

[47] Lisha Li et al. "Hyperband: A novel bandit-based approach to hyperparameter optimization". En: *The Journal of Machine Learning Research* 18.1 (2017), pp. 6765-6816.

[48] Takuya Akiba et al. "Optuna: A Next-generation Hyperparameter Optimization Framework". En: *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2019.

[49] Giuseppe Aceto et al. “Mobile Encrypted Traffic Classification Using Deep Learning: Experimental Evaluation, Lessons Learned, and Challenges”. En: *IEEE Transactions on Network and Service Management* 16.2 (2019), pp. 445-458. doi: 10.1109/TNSM.2019.2899085.

[50] Amin Azari et al. “Cellular Traffic Prediction and Classification: A Comparative Evaluation of LSTM and ARIMA”. En: *Discovery Science*. Ed. por Petra Kralj Novak, Tomislav Šmuc y Sašo Džeroski. Cham: Springer International Publishing, 2019, pp. 129-144. isbn: 978-3-030-33778-0.

[51] Anestis Dalgkitsis, Malamati Louta y George T Karetsos. “Traffic forecasting in cellular networks using the LSTM RNN”. En: *Proceedings of the 22nd Pan-Hellenic Conference on Informatics*. 2018, pp. 28-33.

[52] Martín Panza, Diego Madariaga y Javier Bustos-Jiménez. “Revealing User Behavior by Analyzing DNS Traffic”. En: *Machine Learning for Networking*. Ed. por Selma Boumerdassi, Éric Renault y Paul Mühlethaler. Cham: Springer International Publishing, 2020, pp. 212-226. isbn: 978-3-030-45778-5.

[53] Diego Madariaga, Martín Panza y Javier Bustos-Jiménez. “DNS Traffic Forecasting Using Deep Neural Networks”. En: *Machine Learning for Networking*. Ed. por Éric Renault, Paul Mühlethaler y Selma Boumerdassi. Cham: Springer International Publishing, 2019, pp. 181-192. isbn: 978-3-030-19945-6.

[54] Zhitang Chen, Jiayao Wen y Yanhui Geng. “Predicting future traffic using Hidden Markov Models”. En: *2016 IEEE 24th International Conference on Network Protocols (ICNP)*. 2016, pp. 1-6. doi: 10.1109/ICNP.2016.7785328.

[55] Gilad Katz, Eui Chul Richard Shin y Dawn Song. “ExploreKit: Automatic Feature Generation and Selection”. En: *2016 IEEE 16th International Conference on Data Mining (ICDM)*. 2016, pp. 979-984. doi: 10.1109/ICDM.2016.0123.

[56] Cicero Dos Santos y Bianca Zadrozny. “Learning Character-level Representations for Part-of-Speech Tagging”. En: vol. 5. Julio de 2014.

[57] Frank Hutter, Holger H. Hoos y Kevin Leyton-Brown. “Sequential Model-Based Optimization for General Algorithm Configuration”. En: *Proceedings of the 5th International Conference on Learning and Intelligent Optimization*. LION’05. Rome, Italy: Springer-Verlag, 2011, pp. 507-523. isbn: 9783642255656. doi: 10.1007/978-3-642-25566-3_40. url: https://doi.org/10.1007/978-3-642-25566-3_40.

[58] *Data Science survey, Jetbrains*. 2018. url: https://www.jetbrains.com/research/data-science-2018/ (visitado 15-12-2021).

[59] Sergio Nesmachnow y Santiago Iturriaga. “Cluster-UY: Collaborative scientific high performance computing in Uruguay”. En: *International Conference on Supercomputing in Mexico*. Springer. 2019, pp. 188-202.

[60] *Sunsetting Python 2*. url: https://www.python.org/doc/sunset-python-2/ (visitado 15-12-2021).

[61] *Python Data Analysis Library*. url: https://pandas.pydata.org/ (visitado 15-12-2021).

[62] *Koalas: pandas API on Apache Spark*. url: https://koalas.readthedocs.io/en/latest/ (visitado 15-12-2021).

[63] *Python bindings, Apache Arrow*. url: https://arrow.apache.org/docs/python/ (visitado 15-12-2021).

[64] *Plotly Python Open Source Graphing Library*. url: https://plot.ly/python/ (visitado 15-12-2021).

[65] *PyDomainExtractor, python module*. url: https://github.com/Intsights/PyDomainExtractor (visitado 15-12-2021).

[66] *geopy, python module*. url: https://geopy.readthedocs.io/en/stable (visitado 15-12-2021).

[67] *Logs Formats and Versioning, DNS Logs*. url: https://docs.umbrella.com/deployment-umbrella/docs/log-formats-and-versioning#section-dns-logs (visitado 15-12-2021).

[68] *List of DNS record types*. url: https://en.wikipedia.org/wiki/List_of_DNS_record_types (visitado 15-12-2021).

[69] *Cisco Umbrella, categories*. url: https://docs.umbrella.com/deployment-umbrella/docs/content-categories (visitado 15-12-2021).

[70] J. Sola y Joaquin Sevilla. "Importance of input data normalization for the application of neural networks to complex industrial problems". En: *Nuclear Science, IEEE Transactions on* 44 (julio de 1997), pp. 1464-1468. doi: 10.1109/23.589532.

[71] *TFRecords - Tensorflow*. url: https://www.tensorflow.org/tutorials/load_data/tfrecord (visitado 15-12-2021).

[72] H. Hapke y C. Nelson. *Building Machine Learning Pipelines: Automating Model Life Cycles with TensorFlow*. O'Reilly Media, Incorporated, 2020. isbn: 9781492053194. url: https://books.google.com.uy/books?id=8hpPzQEACAAJ.

[73] Shubhra Kanti Karmaker Santu et al. *AutoML to Date and Beyond: Challenges and Opportunities*. 2021. arXiv: 2010.10777 [cs.LG].

[74] John Wiley & Sons, Ltd, 2017. isbn: 9781119092919. doi: https://doi.org/10.1002/9781119092919.fmatter. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781119092919.fmatter. url: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119092919.fmatter.

[75] Zachary C Lipton, John Berkowitz y Charles Elkan. "A critical review of recurrent neural networks for sequence learning". En: *arXiv preprint arXiv:1506.00019* (2015).

[76] *Data Classes - Python*. url: https://docs.python.org/3/library/dataclasses.html (visitado 15-12-2021).

[77] *Tensorflow - Functional API*. url: https://www.tensorflow.org/guide/keras/functional (visitado 15-12-2021).

[78] *Optuna benchmarks with Kurobajo*. url: https://github.com/optuna/optuna/wiki/Benchmarks-with-Kurobako (visitado 15-12-2021).

[79] *ExtraTreesRegressor - sklearn*. url: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.ExtraTreesRegressor.html (visitado 15-12-2021).

[80] Zhiyong Cui et al. “Learning traffic as a graph: A gated graph wavelet recurrent neural network for network-scale traffic prediction”. En: *Transportation Research Part C: Emerging Technologies* 115 (2020), p. 102620. issn: 0968-090X. doi: https://doi.org/10.1016/j.trc.2020.102620. url: https://www.sciencedirect.com/science/article/pii/S0968090X19306448.

Esta es la última página.
Compilado el martes 20 septiembre, 2022.
http://inco.fing.edu.uy/