INFORME

Predicción de tráfico en redes del Plan Ceibal

Junio 2021

Marcos Pastorini, Alexis Arriola, Eduardo Grampín, Germán Capdehourat, Alberto Castro

acastro@fing.edu.uy

ÍNDICE

1 | INTRODUCCIÓN

En este informe se documenta el trabajo realizado en el marco del Proyecto ANII **FSDA_1_2018_1_154853** titulado Predicción de tráfico en redes del Plan Ceibal.

A continuación se comentan las secciones que componen este informe.

Llegado a este punto, el código del proyecto supera las 4000 líneas de código, por esto, la estructura del proyecto se explica en la sección 2.

La cantidad masiva de datos (decenas de miles de millones) indica que el problema tiene escala de *big data*, determinando un ambiente de trabajo distinto específico, en la sección 3 se comentan el ambiente y herramientas utilizadas.

En la sección 4 se describen los datos usados, los cambios hechos sobre los datos iniciales y su almacenamiento.

Al momento de decidir a qué técnica de aprendizaje automático recurrir para crear un modelo capaz de predecir el siguiente estado de la red, las redes neuronales fueron la elección más coherente por la cantidad de datos disponibles, los detalles de la arquitectura concebida se encuentran en la sección 5.

2 | ESTRUCTURA DEL PROYECTO

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 (figura 1), 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 | AMBIENTE DE TRABAJO

Para el procesamiento de datos mediante *big data* se decidió usar la herramienta *Spark*. Se crearon dos instancias del ambiente, uno para trabajar en Facultad de Ingeniería y otro para trabajar en Ceibal. El primero se preparó desde cero durante la etapa más temprana del proyecto y, en esta sección, se explica cómo se compone, sus configuraciones específicas y la ejecución de código en él.

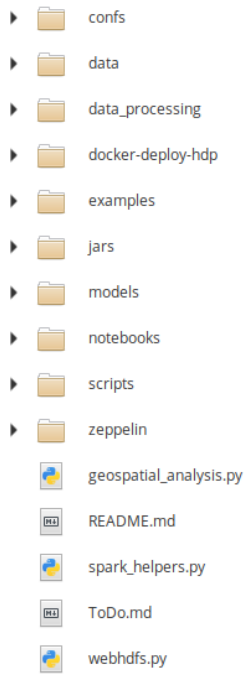


FIGURA 1 Estructura del proyecto.

3.1 | Hortonworks Data Platform Sandbox

Hortonworks Data Platform (**HDP**) es la plataforma que se usó para hacer uso de la herramienta *Spark*, esta plataforma brinda servicios de *cluster-computing*, los principales componentes usados en el proyecto fueron:

Apache HDFS[6] *Hadoop Distributed File System*, 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 de a batches.

Apache YARN[9] *Yet Another Resource Negotiator* 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[7] 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. El formato de los datos es independiente de *Hive* pudiendo ser por ejemplo CSV, TSV, Parquet u ORC, por defecto se usa el último.

Apache Spark[8] es un sistema de *cluster-computing* de proposito general. Provee APIs de alto nivel para Java, Scala, Python y R, su principal funcionalidad es la de abstracción **DataFrame** que permite leer y manipular datos sobre *HDFS* de forma transparente.

Apache Zeppelin[10] notebook web multi propósito que soporta varios lenguajes de programación y permite visualizar datos y ejecutar código de forma interactiva.

Apache Arrow[4] 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. Especialmente útil para el pasaje de datos entre *JVM* y *Python*.

Apache Zeppelin[10] notebook web multi propósito que soporta varios lenguajes de programación y permite visualizar datos y ejecutar código de forma interactiva.

Ambari[3] 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 hasta monitorizar aplicaciones en ejecución.

La integración de estos componentes es transparente para el usuario y su arquitectura se muestra en la figura 2.

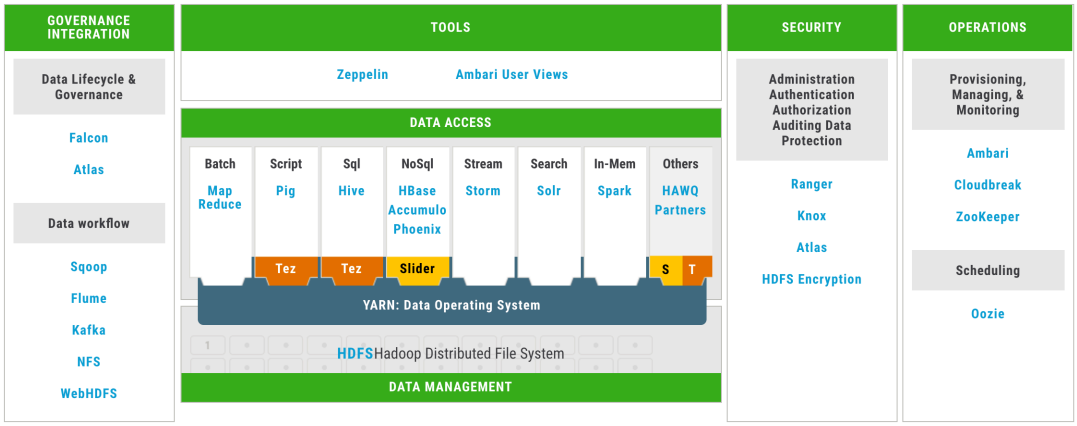


FIGURA 2 Arquitectura de la plataforma **HDP**.

En el ambiente de Facultad de Ingeniería se usó su 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*[37] y en el archivo `README.md` del repositorio en *Gitlab*.

Como los datos a utilizar se encuentran 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. Una copia de todos los datos se encuentra en prod y otra copia reducida (solo mayo de 2019) se encuentra en el ambiente de desarrollo.

3.2 | 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`, su descripción y forma de uso se encuentra en la sección 7.1.1.

Para acceder a uno de los ambientes de trabajo en la plataforma Ceibal se debe configurar una VPN con un cliente Forti[21] con servidor `201.221.46.30:11443` y credenciales válidas.

Una vez conectada la VPN se puede acceder a los servicios a través de `https://ambari.desa.pdm.ceibal.edu.uy` o `https://ambari.prod.pdm.ceibal.edu.uy`, también se puede acceder a través de ssh a uno de los servidores `head-desa.pdm.ceibal.edu.uy` o `head-prod.pdm.ceibal.edu.uy` con un usuario habilitado para desplegar aplicaciones.

3.3 | Conda

La versión usada de *HDP* tiene preinstalado 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[42] 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 **tf** y **data-analysis**, el primero enfocado en el entrenamiento de modelos de *ML* y el segundo en el análisis y procesamiento de *big data*. Para crear cada uno de los ambientes se creó el script cuya descripción y forma de uso se encuentra en la sección 7.1.2.

3.4 | Ejecución de código en *Spark*

Para entender como ejecutar código en *Spark*, primero es necesario introducir algunas nociones básicas de la plataforma. En el nivel más alto de abstracción, cada aplicación que se ejecuta en *Spark* tiene un programa ***driver*** que se encarga de distribuir el trabajo (***tasks***) entre programas ***ejecutores*** ejecutándose en los nodos del cluster. La comunicación y distribución de datos entre ***driver*** y ***ejecutores*** se hace a través del módulo ***cluster manager***, en el marco de este proyecto se usó **Apache YARN**[9]. En la figura 3 se presenta un esquema del modelo de ejecución.

El *driver* se compone por el código que define las transformaciones y acciones aplicadas al conjunto de datos. En su núcleo, el *driver* instancia un objeto de la clase *SparkSession*, este objeto permite al *driver* comunicarse con el cluster, reclamar recursos, dividir la aplicación en *tasks* y planificar y lanzar *tasks* en los *ejecutores*. La configuración del contexto es una tarea fundamental para que una aplicación funcione correctamente, por esto, se trata este tema en las siguientes secciones.

Los *ejecutores* no solo ejecutan las tareas enviadas por el *driver*, también almacenan datos localmente.

Para ejecutar una aplicación, el *driver* organiza la aplicación en *jobs*, cada uno es dividido en *etapas* que consisten en un conjunto de *tasks* independientes que se ejecutan en paralelo. Una *task* es la unidad de trabajo más pequeña en *Spark* y ejecuta la misma fracción de código en cada partición del conjunto de datos.

La plataforma usada cuenta con dos formas de ejecutar código, en las siguientes secciones se discuten junto con sus adaptaciones para el proyecto.

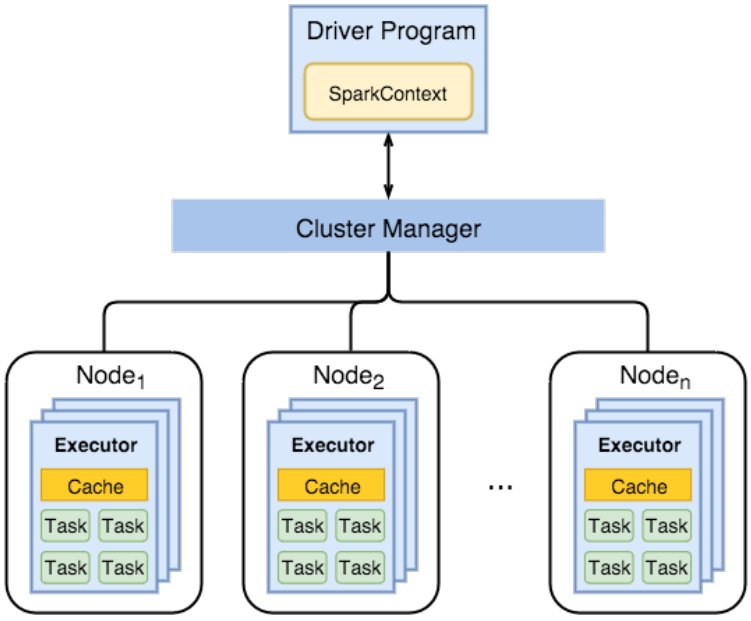


FIGURA 3 Arquitectura del modelo de ejecución de *Spark*

3.4.1 | Ejecución de código con *spark-submit*

Una forma de ejecutar código en *Spark* es a través del comando *spark-submit*, este método es útil para aplicaciones que se ejecutan por batches y no requieren interacción. Cada vez que se ejecuta una aplicación con este script se crea un nuevo contexto.

Para desplegar una aplicación en el entorno de trabajo, se creó un script que se encarga de preparar el sistema para una correcta ejecución, este script se describe en 7.1.4.

Todas las propiedades de *Spark* y sus configuraciones se pueden ver en más detalle en la documentación oficial[40]. Estas pueden ser configuradas mediante una *flag* del comando *spark-submit*, mediante un archivo o mediante la creación de un objeto *SparkSession* dentro del módulo *Python* a ser desplegado. Las propiedades configuradas mediante la creación de un objeto *SparkSession* toman la mayor precedencia, seguidas por las configuradas usando *flags* y por último las configuradas mediante un archivo.

Todos los métodos de configuración se explican a continuación.

Configuración mediante *flag* --conf

Este método configura de forma dinámica las propiedades de la aplicación a ejecutar, estas propiedades deben ser pasadas directamente al comando *spark-submit* en forma **–conf clave=valor**. Un ejemplo se muestra a continuación:

```
spark-submit --conf spark.executor.instances=4 ejemplo.py
```

Las propiedades que no se configuren se reemplazan por los valores por defecto del ambiente.

Configuración mediante archivo

Este método usa un archivo con extensión `.conf` para configurar las propiedades de la aplicación a ejecutar, las configuraciones en se encuentran en forma **clave valor**, para que el proceso funcione correctamente existen algunas propiedades que deben configurarse de forma obligatoria:

spark.master el *cluster manager* que se encargará del despliegue, en este caso se usa *yarn*.
spark.submit.deployMode el modo en que se hace el despliegue de la aplicación, para el ambiente de Facultad de Ingeniería se usa *client* y para el ambiente en Ceibal se usa *cluster*.
spark.pyspark.python la ruta al ejecutable *Python* a usar como interpreter para *PySpark*, en este caso es la ruta al ejecutable de uno de los ambientes virtuales creados.

Al usar este tipo de configuración es importante saber que se omiten las configuraciones por defecto del ambiente *Spark*, para solventar esto existen dos métodos: extraer las configuraciones por defecto (que se encuentran en el archivo `SPARK_CONF/spark-defaults.conf`) o extraer las configuraciones del archivo para pasarlas como parámetros al comando *spark-submit* usando la *flag* **–conf** como se vio anteriormente, en nuestro script preferimos la segunda opción por ser independiente del ambiente de ejecución.

En la sección 2 se comenta el directorio donde se encuentran almacenados los archivos de configuración del proyecto, se creó un archivo para cada ambiente virtual.

El despliegue de una aplicación se realiza en el momento en que se crea un objeto *SparkSession*, a partir de ese momento, solo algunas configuraciones se podrán modificar de forma dinámica. En caso de ejecutar una sesión interactiva de *PySpark* se creará un objeto *SparkSession* con las propiedades definidas en el archivo de propiedades, por esto, se recomienda que además de las propiedades obligatorias se configuren propiedades que aseguren un rendimiento mínimo.

Un ejemplo de archivo de configuración se muestra a continuación:

```
spark.master yarn
spark.submit.deployMode client
spark.pyspark.python /opt/conda/envs/data-analysis/bin/python
spark.sql.execution.arrow.enabled True
spark.executor.instances 4
spark.executor.cores 4
spark.driver.cores 2
spark.executor.memory 20g
spark.driver.memory 2g
spark.yarn.am.memory 2g
spark.yarn.tags data_analysis
```

Configuración mediante *SparkSession*

Dentro del módulo `spark_helpers.py` se creó una función que se encarga de configurar propiedades de cada aplicación en tiempo de ejecución y retornar un objeto de clase **SparkSession**, a través del cual comunicarse con el entorno *Spark*, detalles de su implementación se encuentran en 7.2.1.

Para desplegar un módulo en *Spark* con las configuraciones ya mencionadas, se debe importar dentro del módulo

la función antes mencionada:

```
from spark_helpers import create_spark

if __name__ == "__main__":
    spark = create_spark(app_name='example')
```

Una vez agregado lo anterior, el módulo se debe ejecutar con el script `spark-submit.sh` (detalles en 7.1.4).

3.4.2 | Ejecución de código en *Zeppelin*

Otra forma de ejecutar código en *Spark* es dentro de un ***notebook*** de *Zeppelin*, plataforma accesible a través de `htttp://localhost:9995`. Por su forma de ejecución interactiva, *Zeppelin* puede ser de ayuda en momentos donde se está experimentando con código, para mostrar información sobre datos mientras se procesan o realizar gráficas.

La interacción entre el código dentro de un *notebook* y el ambiente *Spark* se lleva a cabo a través de un ***interpreter***. Esta abstracción propia de *Zeppelin* indica qué lenguaje de programación se usa en un fragmento de código (aportando autocompletado y coloreado de sintaxis), la forma en que debe ejecutarse ese código y como debe configurarse la aplicación que haya por debajo.

En la configuración por defecto, todos los trabajos de los *notebooks* se ejecutan usando la misma instancia para un *interpreter* dado, es decir que se comparten la misma instancia de *SparkContext* y los recursos asociados.

Configuración

En el caso particular de *PySpark*, se debe crear un nuevo *interpreter* perteneciente al grupo *spark2*, esto se puede hacer desde la web de *Zeppelin* (`http://localhost:9995/#/interpreter`) o mediante el script creado con ese fin, la diferencia entre uno y otro método es que el script (que se describe en 7.1.6) se usa uno de los archivos de configuración discutidos en la sección anterior.

Para un correcto funcionamiento, las siguientes propiedades deben ser configuradas de forma obligatoria:

zeppelin.pyspark.python la ruta al ejecutable *Python* a usar como interpreter para *PySpark*, en este caso es la ruta al ejecutable de uno de los ambientes virtuales creados.
SPARK_HOME la ruta a la instalación de *Spark*, este caso es `/usr/hdp/current/spark2-client/`.

Ejecución de código

Para usar las funciones de *Spark* y *Python* dentro de un *notebook*, se debe iniciar cada bloque con la línea:

```
%pyspark
```

Se puede interactuar con el contexto creado usando las variables ya inicializadas al momento de la ejecución **spark** y **sc**.

Carga de módulos *Python*

Importar módulos locales dentro de un *notebook* de *Zeppelin* es una tarea necesaria para evitar duplicar código y siempre estar en iguales condiciones en todo el proyecto, para esto es necesario primero cargar los módulos en la sesión de *Spark* activa, en 7.2.2 se detalla la función implementada con este fin.

3.4.3 | Distribución del ambiente virtual

Para desplegar una aplicación en un ambiente distribuido (como el de Ceibal), es necesario que todos los nodos cuenten con el ambiente virtual usado por esa aplicación. Para cumplir con este fin, se usa el paquete *conda-pack*[15] que se encarga de empaquetar un ambiente virtual de *Conda* y generar un archivo tar.gz listo para ser distribuido. Este proceso (que se hace automáticamente en el script de despliegue creado 7.1.4) se enumera a continuación:

1. Se empaqueta el ambiente virtual activo usando el comando *conda pack*.
2. Se guarda el archivo creado en el directorio `hdfs:///user/USER/envs/` en HDFS al que todos los nodos tienen acceso.
3. Se configuran las variables de entorno **PYSPARK_DRIVER_PYTHON** y **PYSPARK_PYTHON** con la ruta `./environment/bin/python`.
4. Ejecutar el comando *spark-submit* con la siguiente *flag*:

```
--files HDFS_ENVIRONMENT_PATH#environment
```

Esto último provoca que *Spark* descomprima los archivos del ambiente virtual en la ruta `./environment` antes de la ejecución de la aplicación.

3.5 | Despliegue con compilación extra de Spark

La versión de *Spark* que ejecuta la plataforma al momento de desplegar una aplicación puede ser alterada (si el usuario tiene permisos suficientes), este proceso no modifica ninguno de los demás componentes con los que interactúa el proceso en la plataforma pero sí modifica, en tiempo de ejecución y solo durante esa ejecución, las bibliotecas usadas para interactuar con esos componentes; al momento de ejecutar una aplicación se empaquetan todas las bibliotecas compiladas para la versión de *Spark* usada y se envían a todos los nodos de la plataforma. Por esto último, es recomendable que antes de ejecutar una aplicación se hagan pruebas para asegurar que todo funciona correctamente.

Para desplegar una aplicación con una compilación *Spark* que no es la por defecto, se debe descargar desde los servidores de *Apache* y borrar algunas de las bibliotecas descargadas que interactúan con *Hadoop*, este último paso se hace para reducir el tamaño de las bibliotecas distribuidas. Para facilitar este proceso se creó un script descrito en la sección 7.1.5.

Una vez listo el proceso anterior, se deben configurar las variables de entorno **SPARK_HOME** y **SPARK_CONF_DIR** para que apunten a la distribución descargada y la configuración de *Spark* del ambiente de ejecución respectivamente. Esto se hace automáticamente usando el script de **spark-submit** implementado anteriormente.

3.6 | Problemas, soluciones y optimizaciones

Durante la extensión del proyecto se han encontrado diversos problemas al momento de interactuar con el ambiente *Spark*, por esto se crea esta sección donde se enumeran junto con las soluciones que se han encontrado.

Problema

Por defecto *Zeppelin* activa la función *Dynamic Allocation* de *Spark*, esto hace que cuando una aplicación requiere más recursos de los que le fueron asignaron se creen nuevos *ejecutores* a demanda. En un ambiente donde se comparten los recursos este comportamiento puede ser no deseado.

Solución

Al momento de ejecutar una aplicación, se deben configurar las propiedades **spark.dynamicAllocation.enabled** y **spark.shuffle.service.enabled** según se crea necesario[19].

Problema

Cuando a una tabla cargada en memoria se le aplica una transformación que requiere que se particione y se distribuyan los datos entre los *ejecutores*, o cuando se guardan los datos en HDFS, la partición realizada por *Spark* puede no ser óptima (tablas muy pequeñas demasiado particionadas o tablas muy grandes muy poco particionadas).

Solución

En tiempo de ejecución se deben configurar las propiedades **spark.default.parallelism** (en caso de trabajar con RDDs) o **spark.sql.shuffle.partitions** (en caso de trabajar con *DataFrames*) según el tamaño de la tabla sobre el que se trabaja, la documentación oficial recomienda 2 o 3 particiones por CPU[27], este valor determinará cuantos archivos se generarán al momento de guardar los datos en disco.

Problema

La transformación de *DataFrames* de *Spark* a *DataFrames* de *pandas* puede provocar un cuello de botella.

Solución

Para solucionar este problema, se creó el formato de datos *Apache Arrow*, este formato es columnar y se usa en *Spark* para transferir eficientemente datos entre *JVM* y *Python*, suponiendo su uso una mejora general en la ejecución de código *Python*.

Para activarlo se deben seguir los siguientes pasos[5]:

1. Instalar la última versión de la biblioteca **pyarrow** en el ambiente virtual usado.
2. Configurar la variable de entorno **ARROW_PRE_0_15_IPC_FORMAT** en 1 si la versión de *Spark* es 2.x.
3. Configurar las propiedades **spark.sql.execution.arrow.enabled** y **spark.sql.execution.arrow.fallback.enabled** en **True**.
4. Si la versión de *Spark* usada es mayor o igual a 3, se puede configurar **spark.sql.execution.pandas.convertToArrowArraySafely** para que se haga comprobación de tipos.

Problema

Cuando *tasks* distribuidas entre los *ejecutores* tardan mucho tiempo en terminar su ejecución, se excede el tiempo de espera y se pierde la conexión entre *driver* y *ejecutores*.

Solución

Al momento de ejecutar una aplicación, se deben configurar la propiedad **spark.network.timeout** en un valor suficientemente grande para evitar la desconexión, este valor es puramente experimental.

Problema

Al momento de usar *DataFrames* de *Koalas* tareas sencillas (como mostrar algunos elementos) puede tardar mucho en ejecutarse.

Solución

Por defecto *Koalas* crea un índice de manera global para asegurarse la consistencia de las operaciones realizadas, este comportamiento requiere ejecuciones extras que pueden provocar cuellos de botellas. El tipo de índice se puede cambiar para que se calcule independientemente en cada *ejecutor* configurando la propiedad **compute.default_index_type** en **distributed**[17].

Optimizaciones

A continuación se muestran configuraciones que ayudan en la optimización del rendimiento de *Spark*.

spark.sql.adaptive.enabled a partir de *Spark* 3, esta configuración permite que los planes de ejecución se optimicen basados en estadísticas generadas en tiempo de ejecución a medida que se van ejecutando las distintas etapas.[1]
spark.sql.adaptive.coalescePartitions.enabled a partir de *Spark* 3, esta configuración permite que evitar la configuración personalizada de la cantidad de particiones a usar pues se hace automáticamente usando estadísticas generadas en tiempo de ejecución.[1]
spark.sql.autoBroadcastJoinThreshold esta configuración permite regular el tamaño máximo en el que una tabla se envía a los ejecutores al momento de hacer un *join*.

4 | PROCESAMIENTO DE DATOS

En esta sección se detalla el procesamiento que se debió aplicar sobre el conjunto de datos inicial para llegar a un conjunto consistente y con información útil, también se muestran los análisis estadísticos que se hicieron con los datos finales.

Para procesar los datos, se uso el ambiente virtual **data-analysis**, las principales bibliotecas instaladas en él son:

Koalas[26] port de la biblioteca para el análisis y procesamiento de datos *pandas*[36] 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[35] biblioteca que incluye *bindings* de *Apache Arrow* permitiendo a *PySpark* utilizar sus optimizaciones, además aporta un medio de interacción con *HDFS*.
plotly[32] biblioteca usada para generar gráficos sobre los datos, su principal diferencia con otras es la capacidad de generar gráficos interactivos.
PyDomainExtractor[33] 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[22] 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).

4.1 | Conjuntos de datos iniciales

En el proyecto se trabajó sobre dos conjuntos de datos para obtener el conjunto final.

Conjunto de consultas DNS

Conjunto que contiene **logs de consultas a DNS** de los dispositivos conectados a las redes de Ceibal de todo el país, en el período comprendido entre el 09/02/2019 y el 31/12/2019, en total se cuenta con **más de treinta y dos mil millones de datos** (32,777,748,989).

Los logs 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.

En el entorno de Facultad de Ingeniería, los datos se encuentran en el directorio `/opt/datos/datos_umbrella/dnslogs/` del sistema de archivos local donde se generó un directorio por cada día de datos recabados donde cada uno contiene un conjunto de archivos CSV comprimidos con la información separada en columnas. Al principio los datos fueron descomprimidos para comprobar la validez de los archivos, se creó un script para llevar a cabo esta tarea y tener un registro de los archivos corruptos, este script se describe en 7.1.3. La estructura de este directorio de archivos se muestra en la figura 5.

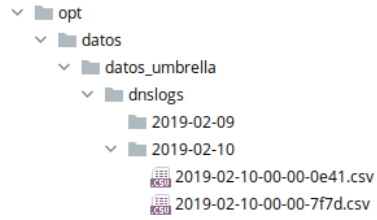


FIGURA 4 Estructura del directorio donde se encuentra el conjunto de datos inicial.

En el entorno de Ceibal, los datos se encuentran 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 el cuadro 1. A continuación se describe cada una de las columnas[30]:

timestamp Cuándo se realizó la consulta en UTC.
mostGranularIdentity La primera identidad correspondiente con la consulta en orden de granularidad.
identities Todas las identidades asociadas con la consulta.
internalIP La dirección IP del dispositivo del que se hizo la consulta.
externalIP La dirección IP del router que procesó la consulta.
action Acción tomada por *Umbrella*, Allowed, Blocked o Proxied dependiendo de si la consulta fue permitida o no.
queryType El tipo de consulta DNS realizada.
responseCode La respuesta a la consulta DNS.
domain El dominio que se consultó.
categories Las 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*

CUADRO 1 Ejemplo de logs de una consulta DNS.

Conjunto de tráfico

Conjunto que contiene **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 cuenta con **más de cuatrocientos cincuenta millones de datos** (467,598,022). Cada dato de este conjunto representa el tráfico acumulado cada 5 minutos.

Los logs fueron generados por la herramienta *NTOP* instalada sobre la red Ceibal con el fin de monitorizar el tráfico.

En el entorno de Facultad de Ingeniería, los datos se encuentran en el directorio `/opt/datos/datos_NTOP/` del sistema de archivos local que contiene un conjunto de archivos CSV comprimidos por semana con la información separada en columnas. Al principio los datos fueron descomprimidos para comprobar la validez de los archivos, se creó un script para llevar a cabo esta tarea y tener un registro de los archivos corruptos, este script se describe en 7.1.3. La estructura de este directorio de archivos se muestra en la figura 5.

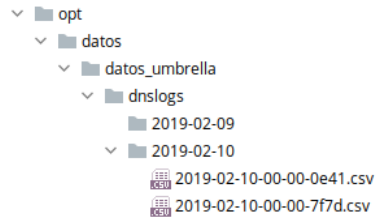


FIGURA 5 Estructura del directorio donde se encuentra el conjunto de datos inicial.

En el entorno de Ceibal, los datos se encuentran 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 el cuadro 2. A continuación se describe cada una de las columnas:

timestamp Cuándo se realizaron las consultas en UTC-3.
mac La dirección MAC del router desde donde se realizaron las consultas.
application El 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

CUADRO 2 Ejemplo de logs de tráfico de las consultas DNS realizadas en 5 minutos.

4.2 | Manejo de datos corruptos

Las funciones y métodos usados en esta sección se describen en 7.2.3.

Se detectaron 5 archivos corruptos en el conjunto de consultas DNS, se intentaron leer desde Spark resultando en datos con filas inconsistentes, al intentar leer esos datos, se encontraron filas inconsistentes como se muestra en el cuadro 3, para esto se ejecutó la función **count_corrupted** que cuenta la cantidad de valores faltantes en la columna *timestamp*.

timestamp	mostGranularIdentity	identities	⋯
NaT	$śCg/4	!%_xV	⋯

CUADRO 3 Ejemplo de datos con filas inconsistentes.

Para tomar una decisión sobre como manejar estos datos, se analizó la cantidad de filas inconsistentes por día (el análisis se encuentra en el *notebook* de *Zeppelin* `Marcos/Analysis/corrupted`), los resultados obtenidos se muestran a en el cuadro 4.

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

CUADRO 4 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*, esta opción descarta todas las filas donde existan datos que no pueden ser parseados de forma correcta al leerlos.

4.3 | Procesamiento de las columnas

A continuación se detalla el procesamiento de cada columna para llegar a los datos finales. Para explicar la lógica de procesamiento de algunas columnas se emplea una notación donde el carácter * representa cualquier carácter 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

Las funciones y métodos usados en esta sección se describen en 7.2.4.

La columna **timestamp** contiene la información temporal del momento en que se realizó la consulta (en formato *yyyy-mm-dd hh:mm:ss*), esta información es de vital importancia para la meta del proyecto, por esto, se decidió separar la información en las columnas **year**, **month**, **day**, **dayofweek** (el lunes se denota con 0 y el domingo con 6), **hour**, **minute** y **second** para 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. También se agregó la columna **date** (en formato *yyyy-mm-dd*) para ayuda en etapas posteriores del procesamiento (4.8).

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

Se utiliza la función ***process_timestamp***. En el cuadro 5 se muestra un ejemplo del resultado del procesamiento.

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

CUADRO 5 Ejemplo de procesamiento de la columna **timestamp**.

mostGranularIdentity, identities, externalIP

Las funciones y métodos usados en esta sección se describen en 7.2.5.

Las columnas **mostGranularIdentity**, **identities** y **externalIP** aportan información sobre el centro donde se realizó cada consulta, la información principal es el identificador dado a ese centro, por esto se decidió extraer esa información y usar como reemplazo de las columnas mencionadas, creando una nueva llamada **ruee** (identificador asignado a los centros educativos por parte cada órgano estatal que los administra).

Se utiliza la función ***process_ruee***, donde se aplican expresiones regulares sobre la columna **mostGranularIdentity** cumpliendo la siguiente lógica de reemplazo:

1. Identificador de la forma **TP-{RUEE}.*** ⟶ **{RUEE}**
2. Identificador de la forma **{RUEE}-.*** ⟶ **{RUEE}**
3. Identificador de la forma **{RUEE},.*** ⟶ **{RUEE}**
4. Identificador de la forma **{RUEE} .*** ⟶ **{RUEE}**
5. Identificador de la forma **1108019179.27.20.70** ⟶ **1108019**. Este es un caso especial donde se encontró un identificador mal asignad.
6. Identificador de la forma **70030589** ⟶ **7003058**. Este es un caso especial donde se encontró un identificador

mal asignado.

7. Identificador con otra forma queda igual.

Luego de ejecutado el procesamiento se hace un control de calidad con la función **ruee_safety_checks**, comprobando que todos los identificadores sean numéricos a excepción de:

- **Locales EDGE - 3G - LTE** identificador representa 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 queda activo, el equipo de soporte de Ceibal lo agrega al sistema asignándole el identificador que corresponda a ese local.
- **NO BORRAR- Locales EDGE - 3G - LTE** otro identificador para el caso anterior.
- **Maqueta Umbrella** identificador de local de prueba.
- **maqueta-umb-ADv4.1-80-UMBRELLA** identificador de local de prueba.
- **escuela-produccion** identificador de local de prueba.

Todas las consultas realizadas desde estos ruees se descartan. En el cuadro 6 se muestra un ejemplo del resultado del procesamiento.

mostGranularIdentity	identities	externalIP	⟶	ruee
1111054	1111054	179.27.13.66		1111054

CUADRO 6 Ejemplo de procesamiento de las columnas **mostGranularIdentity**, **identities** y **externalIP**.

queryType

Las funciones y métodos usados en esta sección se describen en 7.2.6.

La columna **queryType** identifica 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 renombra a **query_type** para seguir la misma notación usada en el resto.

Se utiliza 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 hace un control de calidad con la función **query_type_safety_checks** comprobando que los resultados pertenecieran a la lista de *query types* válidos[29].

En el cuadro 7 se muestra un ejemplo del resultado del procesamiento.

domain

Las funciones y métodos usados en esta sección se describen en 7.2.7.

La columna **domain** identifica el dominio al que se consultó, para contener un número más acotado de dominios posibles, se decidió realizar parsing sobre esta columna creando una nueva llamada **parsed_domain**, también se agregó

queryType		query_type
1 (A)	⟶	A
Other		Other

CUADRO 7 Ejemplo de procesamiento de la columna **queryType**

la columna booleana **ceibal_domain** que identifica si un dominio es de interés de Ceibal y la columna **ceibal_category** donde se clasifican los dominios de interés de Ceibal.

También se creó la columna booleana **dns_from_windows** para ayuda en etapas posteriores del procesamiento (4.3.1).

Para crear las columnas **ceibal_domain** y **ceibal_category** se usa una lista de dominios provista por Ceibal y cuyos valores se muestran en el cuadro 8. Esta lista también se usa para distinguir una consulta proveniente de Windows, información que se registra en forma de booleano en la columna **dns_from_windows**.

Dominio de interés	URL	Valor parseado	Categoría
CREA	`ceibal.schoology.com`	CREA	plataforma_ceibal
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	

Dominio de interés	URL	Valor parseado	Categoría
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	

CUADRO 8 Lista de dominios de interés de Ceibal, URL, valor asignado al parsear y categoría de pertenencia.

También se cuenta con otra lista de dominios de interés general que se utiliza para agrupar todos los dominios que contengan un dominio de interés, la búsqueda del dominio se aplica siguiendo el orden de precedencia antes listado. Esta lista se compone por los dominios:

- `youtube`
- `google`
- `instagram`
- `whatsapp`
- `wikipedia`
- `mega`
- `spotify`
- `netflix`
- `twitter`
- `amazon`
- `facebook`
- `snapchat`

La columna **domain** contiene varios elementos singulares que debieron ser corregidos antes de realizar el *parsing*, a continuación se listan:

- Cuando se realiza una consulta DNS desde un equipo Windows se agrega al dominio el sufijo **.ceibal.edu.uy.**, por ejemplo `www.gmail.ceibal.edu.uy.`
- Dominios contienen signos de interrogación (?), por ejemplo `mundo?ingles.ceibal.edu.uy.`
- Dominios corresponden con el nombre de dispositivos móviles, por ejemplo `redminote5-sr.k.`, `huawei_y6_2018-5525a8182d`, `iphone-de-lucas`
- Dominios contienen el prefijo **/**, por ejemplo `/ceibal.edu.uy.`
- Dominios contienen el sufijo **.getcacheddhcpresultsforcurrentconfig.**, por ejemplo `instagram.fmvd2-1.fna.fbcdn.net.getcacheddhcpresultsforcurrentconfig.`
- Dominios contienen **._**, por ejemplo `_minecraft._tcp.42945.`
- Dominios relacionados a una resolución DNS inversa tienen la forma **IP.in-addr.arpa**, por ejemplo `120.38.239.216.in-addr.arpa.`

Se utiliza la función ***process_domain***, donde se aplican varias expresiones regulares encadenadas cumpliendo la siguiente lógica:

1. Dominio de la forma **/{DOMAIN}** ⟶ **{DOMAIN}**

2. Dominio de la forma **d?oma?in** ⟶ **{DOMAIN}**
3. Dominio de la forma **_minecraft.*** ⟶ **minecraft**
4. Dominio de la forma ***._tcp.{DOMAIN}** ⟶ **{DOMAIN}**
5. Dominio de la forma ***._udp.{DOMAIN}** ⟶ **{DOMAIN}**
6. Dominio de la forma ***._msfcs.{DOMAIN}** ⟶ **{DOMAIN}**
7. Dominio de la forma **._{DOMAIN}** ⟶ **{DOMAIN}**
8. Dominio de la forma **.{DOMAIN}** ⟶ **{DOMAIN}**
9. Dominio de la forma **www.{DOMAIN}** ⟶ **{DOMAIN}**
10. Dominio de la forma ***.www.{DOMAIN}** ⟶ **{DOMAIN}**
11. Dominio de la forma **{DOMAIN}.ceibal.edu.uy.*** ⟶ **{DOMAIN}.**
12. Dominio de la forma ***.bibliotecadigital.ceibal.edu.uy** ⟶ **bibliotecadigital**
13. Dominio que contiene un dominio de interés ⟶ **dominio_de_interes**
14. Dominio de la forma **{DOMAIN}/{QUERY}** ⟶ **{DOMAIN}**
15. Dominio que identifica un dispositivo móvil **android** (con nombre android, samsung, galaxy, redmi o huawei) ⟶ **android-phone**
16. Dominio que identifica un dispositivo móvil **iphone** ⟶ **iphone-phone**
17. Dominio de la forma **{DOMAIN}.getcacheddhcpresultsforcurrentconfig** ⟶ **{DOMAIN}**
18. Dominio de la forma **{DOMAIN}.** ⟶ **{DOMAIN}**

Luego de estos pasos se realiza *parsing* sobre los dominios pre-procesados para extraer el *generic top-level domain* usando la lisa pública de sufijos.

Finalmente, se clasifican los dominios en las categorías explicadas en los cuadros 8 y 9, en caso de no existir una clasificación para un dominio parseado, se usa el valor **sin_categoria**.

Dominio parseado	Categoría
facebook	redes_sociales
instagram	
whatsapp	
twitter	
tiktok	
youtube	
netflix	
ingles	ceibal_ingles
leanenglish	
learnenglishkids	
learnenglishteens	
scratch	pensamiento_computacional
code	
pilasbloques	
blockly	
lightbot	
appinventor	
tynker	

Dominio parseado	Categoría
open-roberta-lab	
microbit-makecode	laboratorios_digitales
microbit	
microbit-python	
tinkercad	
desafioprofundo	red_aprendizajes
deepchallenge	
AAP	
uruguayeduca	formacion
geogebra	
alterceibal	
tigtag-es	otros
twig-es	
khanacademy	
edmodo	
kahoot	
quizizz	
wikipedia	

CUADRO 9 Clasificación Ceibal de dominios parseados.

En el cuadro 10 se muestra un ejemplo del resultado del procesamiento.

domain		parsed_domain	dns_from_w	ceibal_domain	ceibal_category
brother.		brother	False	False	sin_categoria
_minecraft._tcp.42945.		minecraft	False	False	sin_categoria
www.juegos.		juegos	False	False	sin_categoria
/ceibal.		ceibal	False	False	sin_categoria
www.google.ceibal.edu.uy.		google-ceibal	False	True	plataforma_ceibal
youtube.com.ceibal.edu.uy.	⟶	youtube	True	False	redes_sociales
180.com.uy.		180	True	False	sin_categoria
android-cf46f99d82ced781.		android-phone	False	False	sin_categoria
redminote5-sr.k.		android-phone	False	False	sin_categoria
iphone-de-lucas.		iphone-phone	False	False	sin_categoria
cursos.ceibal.edu.uy		cursos	False	True	plataforma_ceibal
xx.xx.xx.xx.in-addr.arpa.		arpa	False	False	sin_categoria

CUADRO 10 Ejemplo de procesamiento de la columna **domain**.

categories

Las funciones y métodos usados en esta sección se describen en 7.2.8.

La columna **categories** indica la categorización de cada dominio aplicada por *Umbrella*, cada log contiene una lista separada por comas de categorías que lo identifican (una lista exhaustiva se encuentra en [14]).

Como ya se mencionó en la sección anterior, las consultas DNS realizadas desde dispositivos Windows agregan el sufijo `.ceibal.edu.uy.`, esto provoca una mala clasificación aumentando artificialmente la cantidad de logs etiquetados como *Software/Technology*,*Educational Institutions*, para tratar esta situación se usa la columna **dns_from_windows** y se elimina la clasificación en los casos donde esa columna lo indique.

Luego de analizar el contenido de la columna (en la sección **??**), se decidió mantener las primeras tres categorías consideradas por *Umbrella*, generando tres nuevas columnas: **category_1**, **category_2** y **category_3**. También se decidió reordenar las categorías de forma tal que *Search Engines*, *Software/Technology* y *Application* estén siempre al final de la lista.

Se detectaron algunos dominios pertenecientes a servicios de infraestructura que no tienen clasificación, estos dominios se clasifican a posteriori con la categoría *Infrastructure* y son:

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

Luego de realizar análisis sobre los agrupamientos que se dan entre las distintas categorías, se decidió agregar una nueva categoría principal en caso de cumplirse algunas condiciones, a continuación se listan las nuevas categorías junto con sus condiciones necesarias:

- Si las categorías *Chat*, *Instant Messaging*, *Social Networking*, *Photo Sharing* o *Blogs* están en la clasificación o el dominio es `tiktok` ⟶ se indica que la categoría principal es *Social Networkig*
- Si las categorías *Video Sharing*, *Movies* o *Radio* están en la clasificación ⟶ 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, se decidió normalizarlas para llevarlas a un formato uniforme, el proceso de normalización consiste en:

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

Se utiliza la función ***process_categories*** y un ejemplo del resultado del proceso se muestra en el cuadro 11.

Para completar la información faltante, se realiza un proceso de imputación de datos donde para cada dominio se elige 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 usa el valor **sin_categoria**, para este proceso utiliza la función ***categories_imputation*** y un ejemplo del resultado se muestra en el cuadro 12.

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	

CUADRO 11 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

CUADRO 12 Ejemplo de imputación de datos en las columnas **category_1**, **category_2** y **category_3**

4.3.2 | Conjunto de tráfico

timestamp

Las funciones y métodos usados en esta sección se describen en 7.2.11.

La columna **timestamp** contiene 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, 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, se descartó crear una columna de segundos porque dentro de una misma ventana temporal de registro se notaron desfaces producto del tiempo de procesado. También se agregó la columna **date** (en formato *yyyy-mm-dd*) para ayuda en etapas posteriores del procesamiento.

Estos datos se encontraban en formato horario UTC-3 por lo que no fue necesario un corrimiento.

Se utiliza la función ***process_timestamp***. En el cuadro 13 se muestra un ejemplo del resultado del procesamiento.

timestamp
2019-11-09 14:10:01

↓

year	month	day	hour	minute	date
2019	4	9	11	10	2019-04-09

CUADRO 13 Ejemplo de procesamiento de la columna **timestamp**.

application

Las funciones y métodos usados en esta sección se describen en 7.2.12.

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

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

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

En el cuadro 14 se muestra un ejemplo del resultado del procesamiento.

application		application
SSL No cert	⟶	ssl_no_cert
Youtube		youtube

CUADRO 14 Ejemplo de procesamiento de la columna **application**

uplink, downlink

Las funciones y métodos usados en esta sección se describen en 7.2.13.

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

4.4 | Información adicional agregada

Al conjunto de logs se le agrega información extra considerada a priori de ayuda para su clasificación. Desde Ceibal se proveyó en dos instancias con información geoespacial y sociocultural respecto a cada centro educativo, estos datos se encuentran en los archivos `tabla_locales_v1.csv` y `tabla_locales_v2.csv` del directorio de datos como se detalla en 2. Las columnas dentro de estos archivos son:

Tabla **tabla_locales_v1**:

ruee El identificador del centro desde donde se hizo la consulta.

departamento El departamento al que pertenece el local.
localidad La localidad del departamento al que pertenece el local.
subsistema El subsistema educativo al que pertenece el local (por ejemplo: CEIP, CES, Universidad).
contexto_sociocultural El quintil al que corresponde el local.

Tabla **tabla_locales_v2**:
num_local El identificador del centro desde donde se hizo la consulta.
razon_social El nombre por el que se identifica el centro.
latitud La coordenada geográfica latitud del local.
longiutd La coordenada geográfica longitud del local.
departamento El departamento al que pertenece el local.
localidad La localidad del departamento al que pertenece el local.
zona La zona a la que pertenece el local (Rural o Urbana).
subsistema El subsistema educativo al que pertenece el local (por ejemplo: CEIP, CES, Universidad).
desc_tipo_centro_depend La 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 aplican los siguientes pasos:

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

4.4.1 | Procesamiento de tabla_locales_old

Al momento de cargar los datos de esta tabla se realiza 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) los valores de las columnas **departamento**, **localidad** y **subsistema**.

La columna **contexto_sociocultural** contiene información tanto del quintil como de la zona a la que pertenece un local, por esto se desagrega la información reduciendo los posibles valores. Las nuevas columnas formadas son **quintil** que toma valores enteros entre 1 y 5 y **zona_quintil** que toma los valores *urbana* o *rural*. La columna original se descarta.

Se buscan locales con ruees no numéricos encontrando los siguientes:

- 13017651-P
- 1302xxxx
- 1202xxxx

Se descartan estos locales por no contener mayor información identificatoria.

4.4.2 | Procesamiento de tabla_locales

Al momento de cargar los datos de esta tabla se realiza un primer procesado que consiste 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) los valores de las columnas **departamento**, **localidad**, **subsistema** y **tipo_centro**.

Se buscan 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 actualizan los datos de **longitud** y **latitud**, seguido de esto se normalizan las coordenadas y se grafican en busca de *outliers*, en la figura 6 se muestran los *boxplots* realizados.

Para la latitud se observan valores por fuera del rango de coordenadas donde se encuentra Uruguay, se analizan estos valores (latitud > -33) y se nota que todos los centros a los que pertenecen están fuera del país, por lo tanto, no se hacen más procesamientos en esta variable. Para el caso de la longitud se repite el mismo proceso y se llegan a resultados iguales. En la figura 7 se muestran los *boxplots* de las coordenadas reducidas a los valores de Uruguay.

4.4.3 | Procesamiento de datos en conjunto

Las funciones y métodos usados en esta sección se describen en 7.2.9.

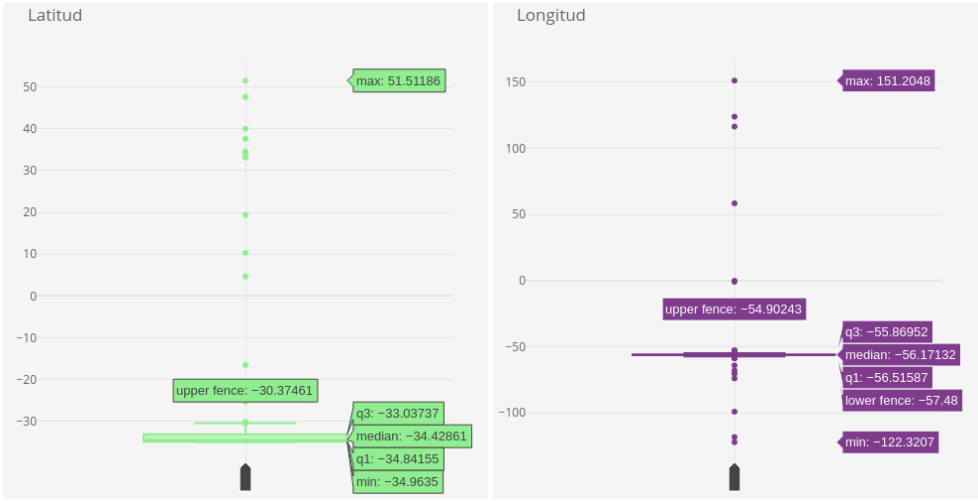


FIGURA 6 *Boxplots* de latitud y longitud antes de procesarlas.

Una vez se combinaron las tablas, se verificó que no se encontraran datos inconsistentes, se encontró en la columna **localidad** identificadores diferentes para mismos lugares, en el cuadro 15 se los identificadores de ambas tablas que representan la misma localidad, se procedió a usar los identificadores provenientes de la tabla **tabla_locales_old** por ser más fidedignos.

Localidad tabla_locales_old	Localidad tabla_locales
paso_carrasco	paso_de_carrasco
villa_rodriguez	rodriguez

CUADRO 15 Identificadores inconsistentes detectados durante la combinación de las tablas.

En el cuadro 16 se muestra la cantidad de valores faltantes para cada una de las columnas de la tabla resultante, se observa un alto porcentaje de valores faltantes en las columnas que contienen la información socio-cultural (**quintil** y **zona_quintil**) y también en las columnas que contienen las coordenadas geográficas.

En el cuadro 17 se repite el conteo pero reducido solo al conjunto de los ruees usados en el conjunto de datos, como el porcentaje de valores faltantes sigue siendo alto (30 % en algunos casos) y todos los datos son 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 la 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 8 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:

1. **Selección de un centro** se selecciona un centro según departamento y localidad, los departamentos están orde-

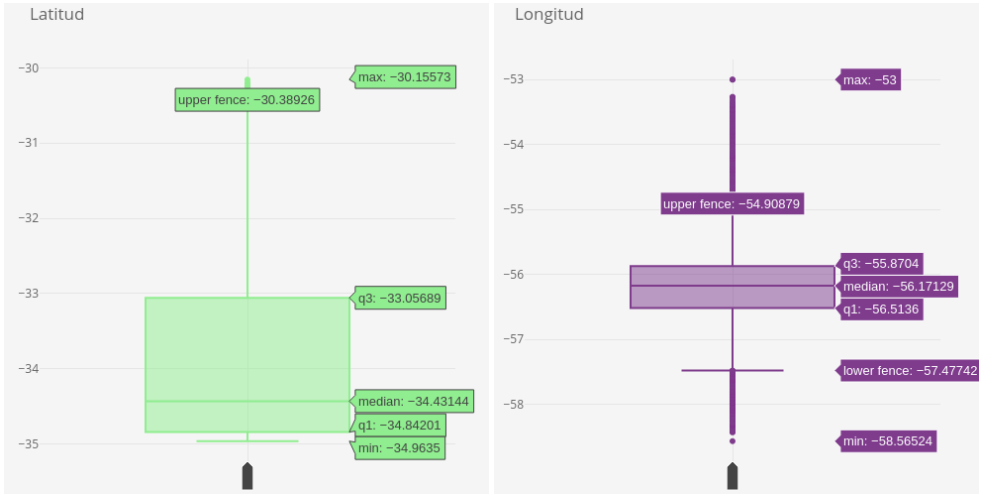


FIGURA 7 Boxplots de latitud y longitud luego de procesarlas.

nados 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.

Para mostrar locales cercanos al analizar en el mapa usado 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 agrega una nueva columna denominada **nivel_fiabilidad** donde queda registrada la confianza que se tiene en un registro de los datos, esta columna tiene 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.

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

CUADRO 16 Cantidad de valores faltantes por columna de la tabla combinada.

- 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 9 se muestra el porcentaje de datos imputados para cada 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**.

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

CUADRO 17 Cantidad de valores faltantes por columna para el conjunto de ruees utilizados.

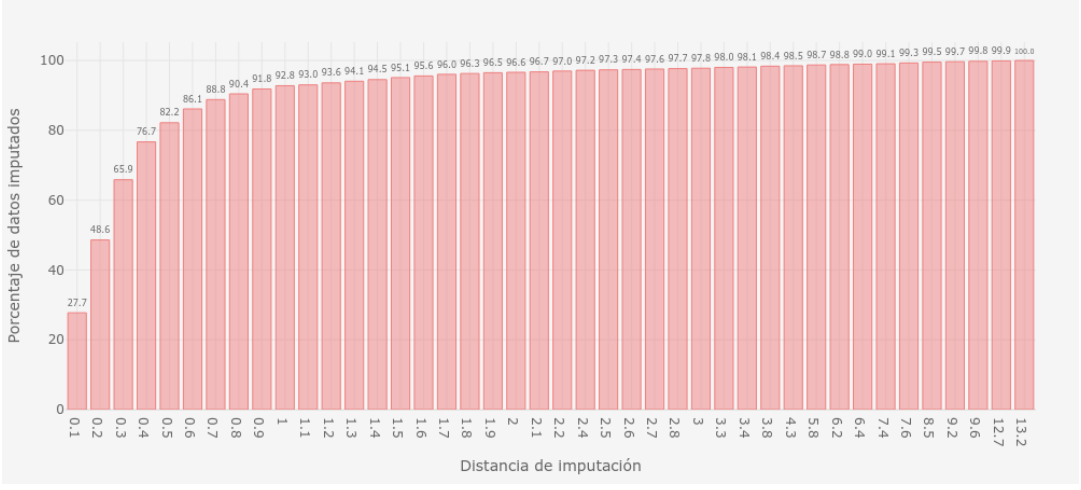


FIGURA 9 Cantidad de datos imputados según la distancia de imputación.

Luego de la imputación se descartaron las columnas **razon_social** y **distancia_imputacion** pues solo son 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 18 se muestra un ejemplo del resultado de este proceso.

FIGURA 8 *Boxplots* de latitud y longitud antes de procesarlas.

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

CUADRO 18 Ejemplo de adición de información geoespacial y sociocultural.

4.5 | Codificación de variables

Las funciones y métodos usados en esta sección se describen en 7.2.10.

Los modelos de predicción solo pueden trabajar con variables de tipo numérico, por lo tanto todas las variables que no son de ese tipo 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 memoria, en este proyecto se decidió seguir el segundo camino.

El proceso de codificación para una variable se describe a continuación:

1. Se seleccionan los n valores más frecuentes de la variable con $0 < n \leq 2000$, si $n < 2000$ se usan todos los valores posibles.
2. A cada valor se le asigna un identificador i único tal que $1 \leq i \leq n$, a todo valor que no se encuentre dentro de

los elegidos para conformar el mapeo se le asigna el identificador $n + 1$.

Se usa la función ***encode_features*** y en el cuadro 19 se muestra un ejemplo del resultado del procesamiento para la columna **query_type**.

query_type		query_type
A		1
Other	⟶	3
AAAA		2
Other		3

CUADRO 19 Ejemplo de *encoding* de la columna **query_type**

Las columnas que representan datos temporales se codifican específicamente para entender rápidamente el valor codificado y que estos sean mayores o iguales a 1, la codificación se explica en el cuadro 20.

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

CUADRO 20 *Encoding* de las columnas que representan datos temporales

4.6 | Agregación y remuestreo de datos

Al intentar procesar todo el conjunto de datos con un modelo no provisto en el entorno *Spark*, los tiempos de entrenamiento resultaron ser inviables para los tiempos del proyecto (más de 300 horas), esto llevó a la búsqueda de algún tipo de agregación que pudiera 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 clave de cada agregación realizada.

Para realizar la 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 menor escala, 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 el cuadro 21 se muestra un ejemplo de re-muestreo de la columna **minute**.

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

CUADRO 21 Ejemplo de re-sampleo de la columna **minute** a 5 minutos

4.6.1 | Agregación temporal por ruee

En la agregación temporal se usan como columnas clave las columnas que representan datos temporales junto con la columna **ruee**. Es decir que, en este caso, se obtiene una agregación donde se resume 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, se probaron los períodos de 5, 10 y 15 minutos.

En el cuadro 22 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

CUADRO 22 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 usan como columnas clave las columnas que representan datos temporales junto con las columnas **departamento** y **subsistema**. Es decir que, en este caso, se obtiene una agregación donde se resume 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 el cuadro 23 se muestra un ejemplo del índice resultante.

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

CUADRO 23 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 usan como columnas clave las columnas que representan datos temporales junto con la columna **subsistema**. Es decir que, en este caso, se obtiene una agregación donde se resume 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 el cuadro **??** se muestra un ejemplo del índice resultante.

4.7 | Registro de variabilidad temporal

Para ayudar en la predicción, se agrega a las columnas que presentan valores numéricos, una columna extra que registra 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 realiza usando el promedio móvil exponencial ponderado (EWMA), la ventana temporal (t) usada, que representa la cantidad de tiempo en el que se observa la variación, se elige dependiendo de la frecuencia temporal de los datos de modo que t represente una hora, por ejemplo, si los datos tienen frecuencia

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

CUADRO 24 Ejemplo de índice resultante del proceso de agregación de datos para el período temporal de 15 minutos, en la agregación temporal por subsistema.

de 5 minutos, se usa $t = 12$.

La formula empleada para el cálculo es:

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

4.8 | Predicción

Conjunto de consultas DNS

Para la predicción de la cantidad de consultas DNS, a cada dato puntual se le agregó la cantidad de consultas en la siguiente hora. El conteo de consultas siguientes se modeló de dos formas distintas para poder probar el comportamiento del modelo en distintas situaciones:

- Conteo agregado de consultas: Se cuenta en una sola columna la cantidad total de consultas para todos los dominios.
- Conteo desagregado de consultas: Se cuenta en varias columnas la cantidad de consultas de dominios particulares, también se agrega una columna extra donde se almacena la cantidad de consultas de los dominios no considerados.

En el caso del conteo desagregado, se consideraron los dominios que concentraron la mayor cantidad de consultas y de tráfico durante el período temporal evaluado.

Conjunto de tráfico

Para la transformación de consultas DNS en tráfico, a cada dato puntual se le agregó el tráfico asociado a sus consultas DNS. Las columnas agregadas y desagregadas siguen la lógica de las generadas para el anterior conjunto. Para este caso, se quitaron los dominios `SSL` y `SSL_NO_CERT`, estos dominios representan el tráfico cifrado y por lo tanto no pueden ser clasificados correctamente.

4.9 | Particionado

Una vez preparados los datos y las predicciones, se divide 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 puede fijar al momento de realizar la 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 realiza la partición determinan la forma en que se dividen 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 **hora** 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 el cuadro 25. En cambio, si se particiona usando la columna **date**, todos los valores de cada subconjunto de días estarán solamente en su respectivo subconjunto.

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

CUADRO 25 Ejemplo de particionado usando las columnas **date** y **hour** conjuntos *train*, *val* y *test* respectivamente.

Para evitar que comportamientos irregulares aparezcan solo en un subconjunto (por ejemplo, períodos de vacaciones), se permite 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.

4.10 | Normalización

Aplicar distintas técnicas de normalización de los datos tiene un impacto en el rendimiento y el tiempo de entrenamiento de los modelos de redes neuronales.[39] En este proyecto se optó por escalar las distintas variables numéricas al rango $[0, 1]$, para esto, se obtienen los valores máximo y mínimo de una variable y se aplica 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 deben obtener los valores estadísticos durante la etapa posterior a la generación de las particiones (sección 4.9) y usarlos aplicando este proceso durante la ingesta de los datos.

4.11 | Pipeline

Una vez definidas y probadas todas las funciones a aplicar sobre las columnas para su procesamiento, se decidió realizar un pipeline definido en varias etapas explicadas a continuación.

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*** que divide cada etapa en tres pasos ejecutados secuencialmente sobre los datos:

1. Cargar datos en memoria
2. Procesar datos
3. Guardar datos en disco

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 guardar y leer datos.

4.11.1 | 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 de a semana, 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 Cuándo se realizaron las consultas en UTC-3.
mac La dirección MAC del router desde donde se realizaron las consultas.
application El 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.

La figura 10 es una representación esquemática de esta etapa.

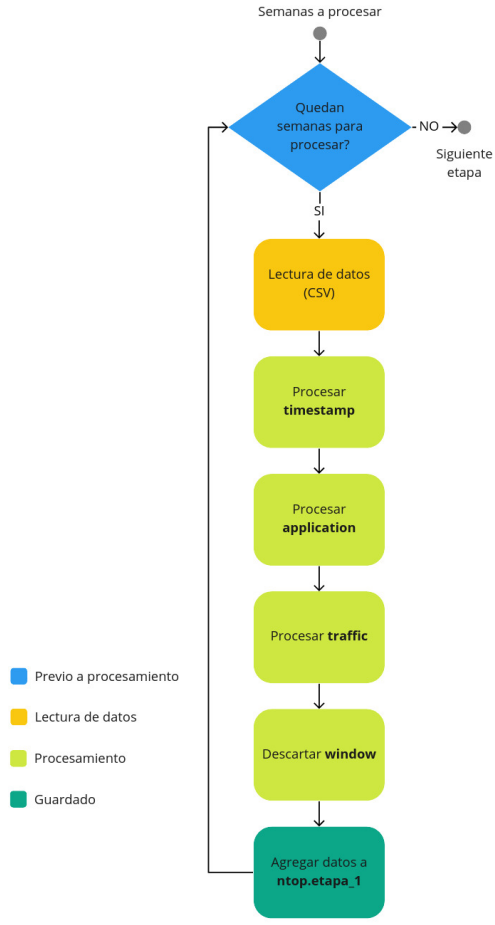


FIGURA 10 Etapa 1 del pipeline del conjunto de tráfico

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_sf_{SAMPLE_FREQUENCY}_agg_{AGGREGRATION_COLUMNS}[_rs]` 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, en caso de que para la agregación de una columna se quiera seleccionar un subconjunto de de los valores posibles, los valores

por 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 11 es una representación esquemática de esta etapa.

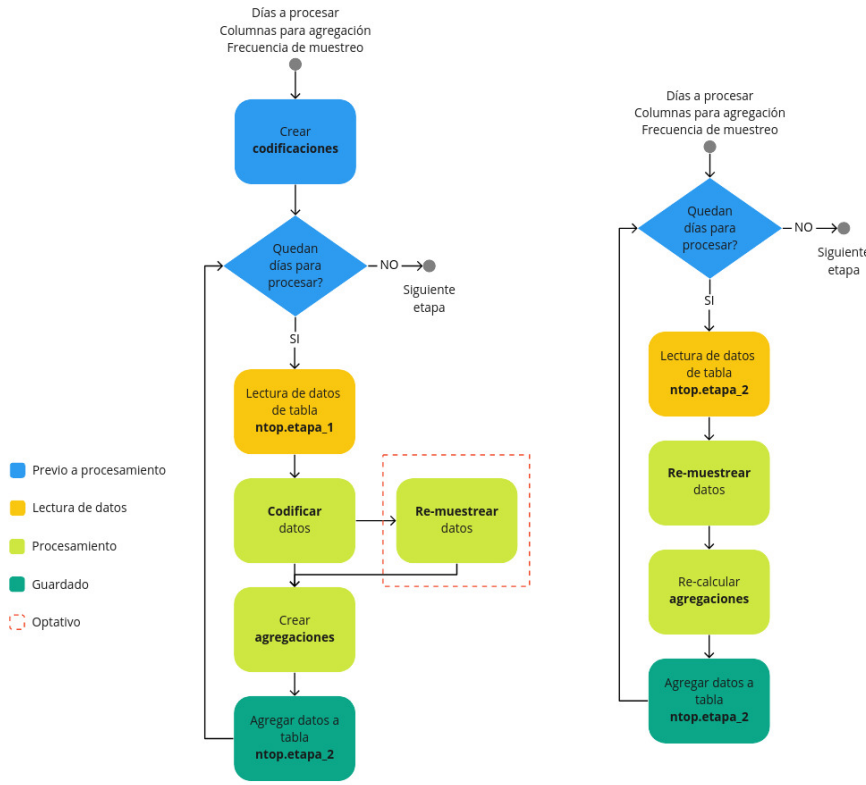


FIGURA 11 Etapa 2 del pipeline del conjunto de tráfico (izquierda) y etapa 2 de re-muestreo (derecha)

Etapa 2 - Re-muestreo

Ubicación datos de entrada tabla `etapa_2_sf_{SAMPLE_FREQUENCY}_agg_{AGGREGRATION_COLUMNS}[_rs]` en base de datos `ntop` en *Hive*.
Formato de lectura ORC
Ubicación datos de entrada tabla `etapa_2_sf_{SAMPLE_FREQUENCY}_agg_{AGGREGRATION_COLUMNS}[_rs]` 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.

En esta etapa se mantienen todas las columnas previamente creadas.

La figura 11 es una representación esquemática de esta etapa.

4.11.2 | 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

Para las primeras etapas, los resultados se guardan en tablas en *Hive* 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 guardar y leer datos.

En esta etapa, al comienzo se realiza el **manejo de datos corruptos** (4.2) y luego se lleva a cabo todo el **procesamiento de columnas** que fuera independiente del resto de los datos (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 Cuándo se realizó la consulta en UTC.
mostGranularIdentity La primera identidad correspondiente con la consulta en orden de granularidad.
identities Todas las identidades asociadas con la consulta.
internalIP La dirección IP del dispositivo del que se hizo la consulta.
externalIP La dirección IP del router que procesó la consulta.
action Acción tomada por *Umbrella*, Allowed, Blocked o Proxied dependiendo de si la consulta fue permitida o no.
queryType El tipo de consulta DNS realizada.
response_code La respuesta a la consulta DNS.
domain El dominio que se consultó.
categories Las categorías de contenido asignadas por *Umbrella* correspondientes con el dominio de la consulta.
year El año en que se hizo la consulta.
month El mes en que se hizo la consulta.
day El día del mes en que se hizo la consulta.

dayofweek El día de la semana en que se hizo la consulta.
hour La hora en que se hizo la consulta.
minute El minuto en que se hizo la consulta.
second El segundo en que se hizo la consulta.
ruee El identificador del centro desde donde se hizo la consulta.
query_type El tipo de consulta DNS realizada.
parsed_domain El dominio que se consultó.
category_1 La primera categoría de contenido que corresponde con el dominio de la consulta.
category_2 La segunda categoría de contenido que corresponde con el dominio de la consulta.
category_3 La tercera categoría de contenido que corresponde con el dominio de la consulta.

La figura 12 es una representación esquemática de esta etapa.

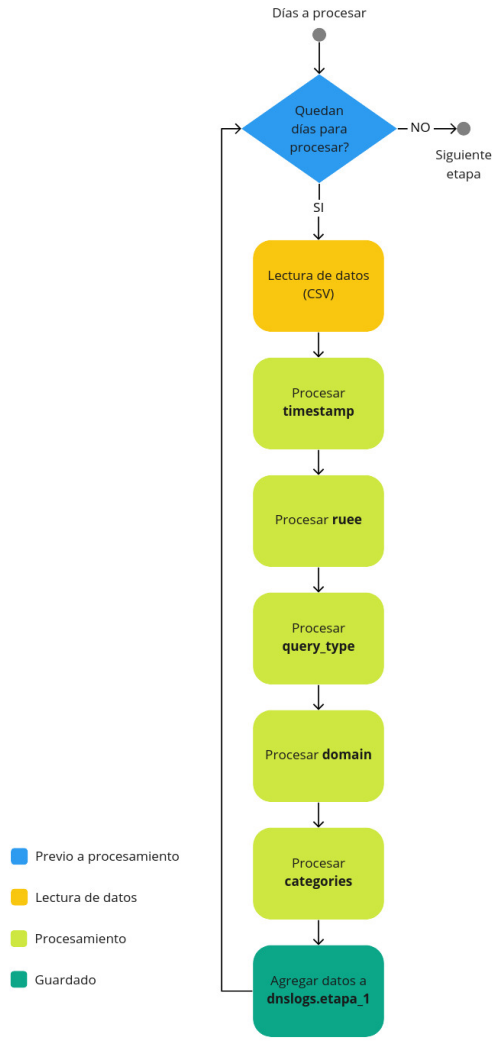


FIGURA 12 Etapa 1 del pipeline

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 La coordenada geográfica longitud del local.
latitud La coordenada geográfica latitud del local.
departamento El departamento al que pertenece el local.
localidad La localidad del departamento al que pertenece el local.
quintil El quintil al que corresponde el local.
zona_quintil La zona a la que pertenece el local (Rural o Urbana).
subsistema El subsistema educativo al que pertenece el local (por ejemplo: CEIP, CES, Universidad).
tipo_centro La descripción de tipo de dependencia del local (por ejemplo: Escuela Privada, Dependencia Administrativa, Utu).
nivel_fiabilidad El nivel de fiabilidad de los datos geoespaciales.

La figura 13 es una representación esquemática de esta etapa.

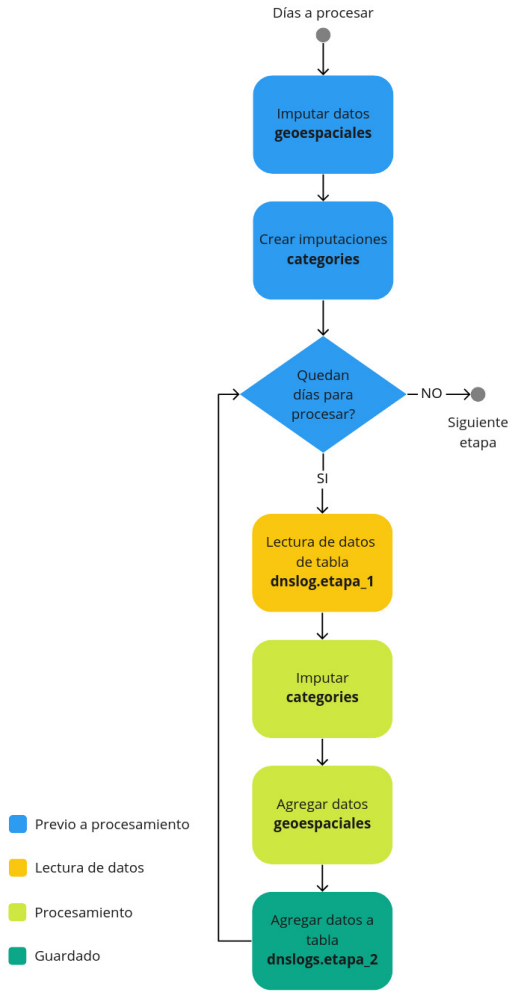


FIGURA 13 Etapa 2 del pipeline

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_sf_{SAMPLE_FREQUENCY}_agg_{AGGREGRATION_COLUMNS}[_rs]` 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, en caso de que para la agregación de una columna se quiera seleccionar un subconjunto de de los valores posibles, los valores

por columna, en este caso se agrega el sufijo _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 (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 14 es una representación esquemática de esta etapa.

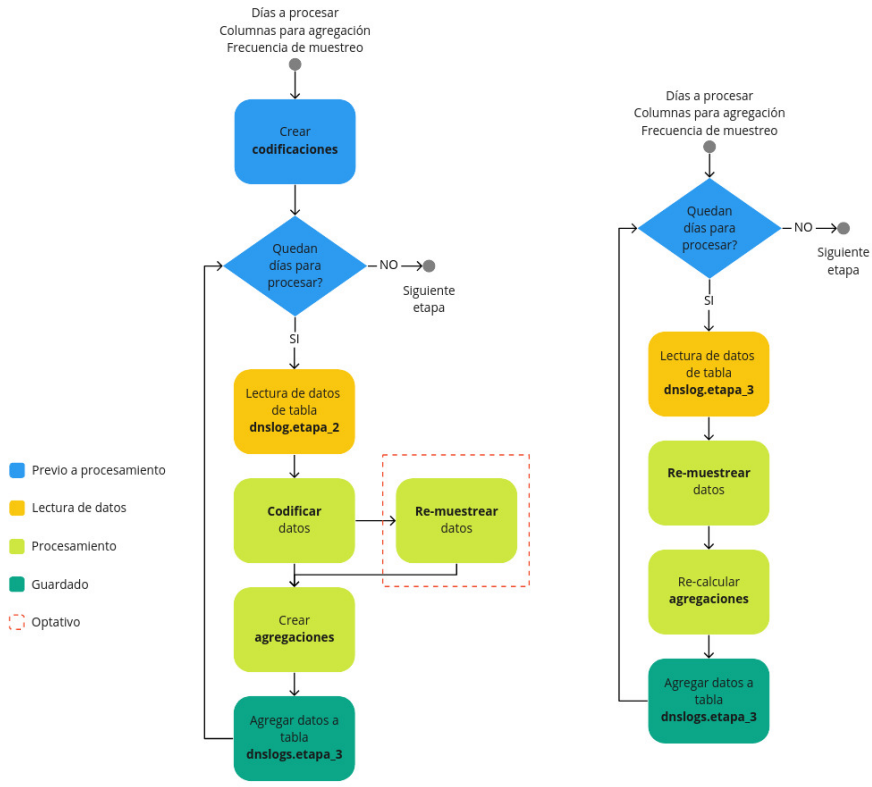


FIGURA 14 Etapa 3 del pipeline del conjunto de consultas DNS (izquierda) y etapa 3 de re-muestreo (derecha)

Etapa 3 - Re-muestreo

Ubicación datos de entrada tabla `etapa_3_sf_{SAMPLE_FREQUENCY}_agg_{AGGREGRATION_COLUMNS}[_rs]` en base de datos `dnslogs` en *Hive*.
Formato de lectura ORC
Ubicación datos de entrada tabla `etapa_3_sf_{SAMPLE_FREQUENCY}_agg_{AGGREGRATION_COLUMNS}[_rs]` en base de datos `dnslogs` en *Hive*.
Formato de lectura ORC

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 14 es una representación esquemática de esta etapa.

Etapa 4

Ubicación datos de entrada tabla `etapa_3_sf_{SAMPLE_FREQUENCY}_agg_{AGGREGRATION_COLUMNS}[_rs]` en base de datos `dnslogs` en *Hive* y tabla `etapa_2_sf_{SAMPLE_FREQUENCY}_agg_{AGGREGRATION_COLUMNS}[_rs]` en base de datos `ntop` en *Hive*.
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.[46]

En esta etapa, antes de iniciar el procesamiento por día se realiza el particionado de los datos (4.9) y se generan las predicciones de cada partición (4.8), 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 4.7), 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 (4.10). 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 4.7.

La figura 15 es una representación esquemática de esta etapa.

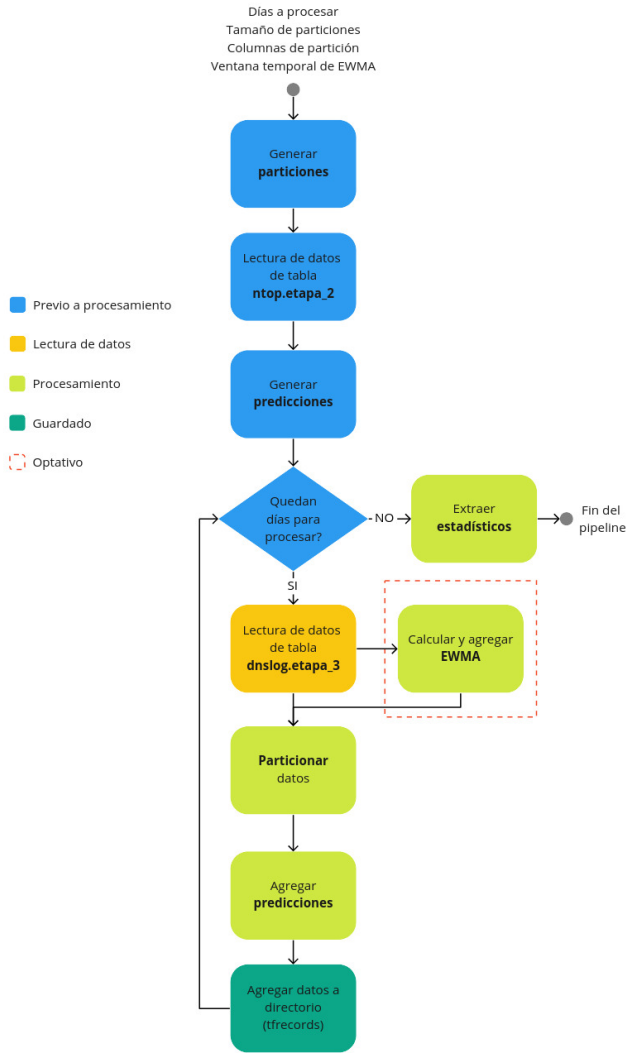


FIGURA 15 Etapa 4 del pipeline del conjunto de datos

5 | APRENDIZAJE SUPERVISADO

En esta sección se comentan los detalles de la arquitectura diseñada para el modelo predictor y su implementación.

Para crear y entrenar el modelo de redes neuronales, se usó el ambiente virtual **tf**, las principales bibliotecas instaladas en él son:

TensorFlow[44] biblioteca de *machine learning* centrada en redes neuronales, aporta APIs de alto y bajo nivel para desarrollar modelos potentes de forma rápida.

TensorFlowOnSpark[45] biblioteca que funciona de nexo entre *TensorFlow* y *Spark*, habilita el uso de redes pro-

fundas en clusters de nodos GPUs y CPUs sin tener que hacer grandes cambios en el código ya implementado para trabajar en un equipo local.
Spark Tensorflow Distributor[41] biblioteca que se encarga de distribuir el entrenamiento en un cluster *YARN*, usa mecanismos de la versión 3 de *Spark* que aseguran mayor estabilidad en el entrenamiento.
Horovod[25] biblioteca que se encarga del entrenamiento distribuido en varios *frameworks* distintos, funciona sobre muchos tipos de cluster, en particular, *YARN*.
Optuna biblioteca usada para la búsqueda y optimización automática de hyperparámetros de un modelo.

5.1 | Arquitectura planteada

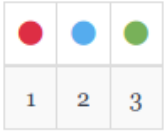
Al analizar los datos se encontraron ciertas características que determinaron el uso de algunos componentes específicos en la arquitectura del modelo, a continuación se listan:

Tipo de variable

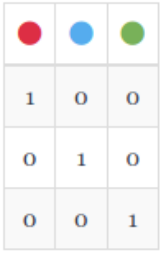
A cada una de las variables de un conjunto de datos se la puede categorizar de muchas formas que ayudan a entender su tratamiento particular, la categorización más básica es entre variables continuas y variables categóricas. La distinción entre ambas es la siguiente: las variables continuas son aquellas 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).

Por su naturaleza numérica, las variables continuas pueden ser procesadas por el modelo de redes neuronales de forma automática, por lo tanto no se aplica 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:

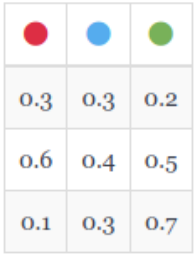
- **Etiqueta numérica**: a cada categoría se le asigna un valor entero. Este método le asigna un orden arbitrario a las categorías lo cual es indeseado, a modo de ejemplo, si se considera la variable color con valores rojo y azul, un modelo consideraría que el valor 1 (representando el rojo) sería menor que el valor 2 (azul) pero este razonamiento no tiene sentido en el conjunto que representa la variable. En la figura 16a 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 16b 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.[24] En la figura 16c se ve un ejemplo de esta representación.



(a) Ejemplo etiqueta numérica



(b) Ejemplo one-hot encoding



(c) Ejemplo embeddings

FIGURA 16 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, entonces, a cada columna de una consulta que representa una variable categórica, se le aplica una capa independiente de **embeddings** que es entrenada junto con el resto de la red neuronal. Se configura el mismo tamaño e de la representación vectorial de cada capa de *embedding*, al hacer esto, se crea una representación matricial de tamaño $n \cdot e$ donde n son la cantidad de variables; esta representación puede ser procesada por las siguientes capas de la red sin problemas. Detalles de implementación y decisiones tomadas se comentan en 5.3.1.

Representatividad de los datos

A priori no se conocen las relaciones entre las variables que componen el conjunto de datos, tampoco se sabe si algunas de ellas determinan o no el estado de la red en el siguiente momento temporal, por lo tanto es necesario buscar un modelo de extracción de *features*.

Para obtener este modelo, a la representación matricial conformada por las variables de una consulta se le aplican 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 los datos usados en problemas de procesamiento de lenguaje natural, donde recientemente se han usado las redes convolucionales para extraer *features* de representaciones vectoriales de palabras[18, 47], 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 agrupan las consultas en períodos de tiempo (*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.[28]

El período de tiempo a usar es un parámetro a definir, todas las consultas de ese período deben estar asociadas a la misma predicción para que la red aprenda las relaciones correctamente, por esto se realizaron conteos de la cantidad de consultas por hora, debería

Teniendo en cuenta lo planteado anteriormente se llega una arquitectura como la mostrada en la figura 17, donde la entrada se procesa según sus características particulares, luego un conjunto de capas convolucionales se encargan de generar representaciones internas especializada, 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.

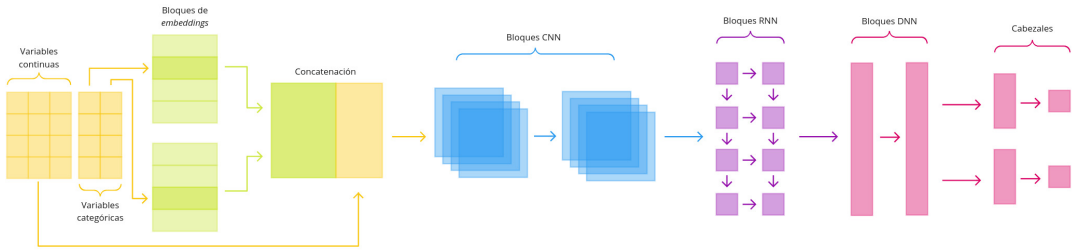


FIGURA 17 Esquema de arquitectura base planteada.

5.2 | Lectura de datos

Para alimentar a la red con datos, se cargan los archivos en formato tfrecord creados en la etapa final del pipeline (14) y se preprocesan 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 repercuta en el tiempo de entrenamiento del modelo, se mantienen los datos en memoria luego del primer epoch del entrenamiento.

En la figura 18 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 describen 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 el tamaño 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 4.10).
5. ***Batching*** los ejemplos se dividen en conjuntos del tamaño de la muestra sobre la que se entrenará al mismo tiempo en un instante dado (b), 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 guarda en memoria para evitar volver a ejecutar el procesamiento en las iteraciones faltantes.

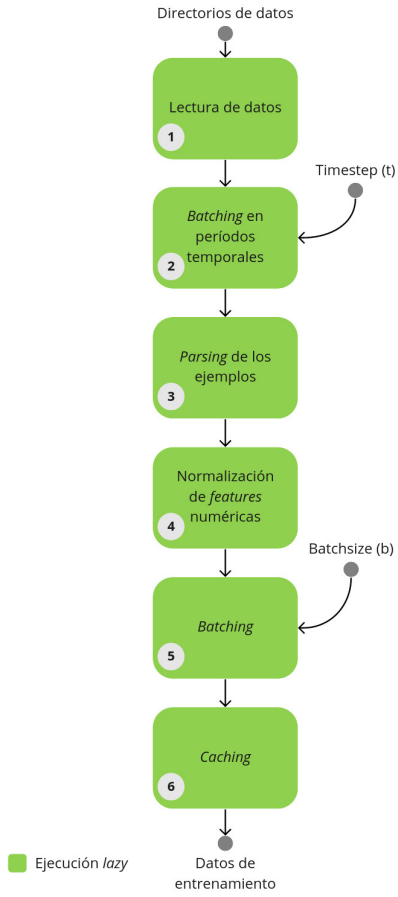


FIGURA 18 Transformaciones de los datos antes del entrenamiento.

5.3 | Implementación en Tensorflow

Esta implementación 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*[16] llamada ***Params*** cuya instancia almacena atributos de una arquitectura de red (por ejemplo cantidad de capas densas y cantidad de neuronas por capa) la descripción completa de sus atributos se encuentra en la sección 7.2.14. Para crear la arquitectura planteada se usa la siguiente configuración paramétrica:

```
Params(
    num_cnn_layers=2,
    num_cnn_filters=32,
    cnn_kernel_size=2,
    cnn_dropout=0.1,
```

```
        cnn_batch_norm=1,
        cnn_activation='linear',
        cnn_pooling=None,

        num_rnn_layers=1,
        rnn_hidden_dim=128,
        rnn_dropout=0.1,
        rnn_layer_norm=1,
        rnn_layer_type='lstm',

        num_dnn_layers=1,
        dnn_hidden_dim=128,
        dnn_dropout=0.1,
        dnn_layer_norm=1,
        dnn_kernel_initializer='auto',
        dnn_activation='relu',
    )
```

Dentro de la red se usan 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, los detalles de implementación de las funciones se encuentran en la sección 7.2.15.

5.3.1 | Procesamiento de la entrada

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

Variables categóricas

Una variable categórica puede ser de tamaño 1 (como por ejemplo la variable **month**) o mayor (por ejemplo la variable **domains**), a partir de ahora el tamaño de una variable i categórica se denominará f_i. Para el uso de *embeddings* dos parámetros deben definirse: la cantidad de valores posibles que puede tomar una variable y el tamaño de los vectores densos creados. Para asignar la cantidad de valores posibles para cada variable, se cuentan los valores y se usan en tiempo de creación de los *embeddings*. Para asignar el tamaño de los vectores densos (d_i) se plantean dos posibles acercamientos:

- Para todas las variables usar el mismo tamaño vectorial, es decir $d_i = D, \forall i$. Esta estrategia genera vectores de dimensiones $f_i \times D$ y puede ocasionar que no todas las variables obtengan la mejor representación vectorial posible, por ejemplo, una variable con alta cardinalidad seguramente requiera una representación vectorial de mayor tamaño que otra variable con cardinalidad muy baja y, como se usa el mismo tamaño para todas las variables, algunas pueden salir menos beneficiadas que otras.

- Para todas las variables usar un tamaño vectorial que dependa de su cardinalidad, es decir $d_i = f_i \cdot r, \forall i, 0 < r < 1$. Esta estrategia genera vectores de dimensiones $f_i \times d_i$ y requiere redimensionar los vectores para poder concatenarlos más tarde, este proceso se realiza seleccionando $d_{min} = \min_{\forall i} d_i$ y reordenando los valores del vector de modo que sus dimensiones sean $f'_i \times d_{min}$, donde $f'_i = \frac{d_i}{d_{min}} \cdot f_i$. Esta estrategia presenta dos inconvenientes: todo d_i debe ser divisible entre d_{min} (una solución a esto es eligir d_i potencia de 2) y se pierde interpretabilidad de la salida de esta sección de la red, pues se altera el tamaño de la dimensión de las variables de una muestra (f_i). Esta estrategia presentaría en mucho menor medida el problema presentado en la estrategia anterior.

Ambas estrategias presentan parámetros que deben ser seleccionados para obtener el mejor rendimiento posible (D y r en cada caso), a priori no se descarta ninguna de las dos estrategias para poder comparar cual obtiene mejor desempeño.

Cualquiera sea la estrategia utilizada, las representaciones vectoriales deben concatenarse en la dimensión de las *features* para obtener un nuevo vector de dimensiones $\sum_{\forall i} f_i \times d$ con d seleccionado según la estrategia utilizada. La función encargada de crear los *embeddings* se llama **create_embeddings**.

Variables continuas

Las variables continuas deben ser procesadas para concatenarse a la representación vectorial generada por el uso de *embeddings*, para esto, se debe igualar el tamaño de estas variables con el tamaño de las representaciones (d). Esto se lleva a cabo mediante el uso de un bloque DNN (descrito en la sección 5.3.4) cuyo tamaño de salida es d, aquí se genera un vector de dimensiones $f \times d$ donde f es la cantidad de variables continuas y este vector se concatena al generado para representar a las variables categóricas.

5.3.2 | Bloque CNN

La función encargada de crear un bloque CNN se llama ***create_cnn_block***, este tipo de bloque esta compuesto por:

Capa convolucional este tipo de capa está compuesto por una función de activación a y dos vectores, K de dos dimensiones llamado *kernel* y b llamado *bias*. Implementa la función $a(conv(I, K) + b)$ donde i son los valores de la entrada de la red. La función *conv* divide I en varios fragmentos del tamaño de K (I_k) y realiza el producto interno entre cada I_k y K, en la figura 19 se muestra un diagrama de su aplicación.
Capa de regularización (*batch normalization*) este tipo de capa normaliza las activaciones de la capa anterior de cada ejemplo dependiendo del resto del *batch*, en resumen, aplica una transformación que mantiene la media del conjunto de ejemplos cerca de 0 y la desviación estándar cerca de 1.
Capa de *dropout* este tipo de capa selecciona valores de la activación de la capa anterior de forma aleatoria y los transforma en 0 durante el entrenamiento, esto ayuda a prevenir el *overfitting*. En este bloque se usa *dropout* espacial, donde no se seleccionan valores individuales sino vectores enteros.
Capa de *pooling* este tipo de capa reduce las dimensiones de la activación de la capa anterior aplicando una función a un conjunto de valores, esta función puede ser 'max' (máximo), 'avg' (promedio) o 'mix' (máximo y promedio seguido de concatenación de los valores). Esta capa es opcional.

Todas las capas de este bloque se agregan dentro de un componente *TimeDistributed* que permite aplicar una

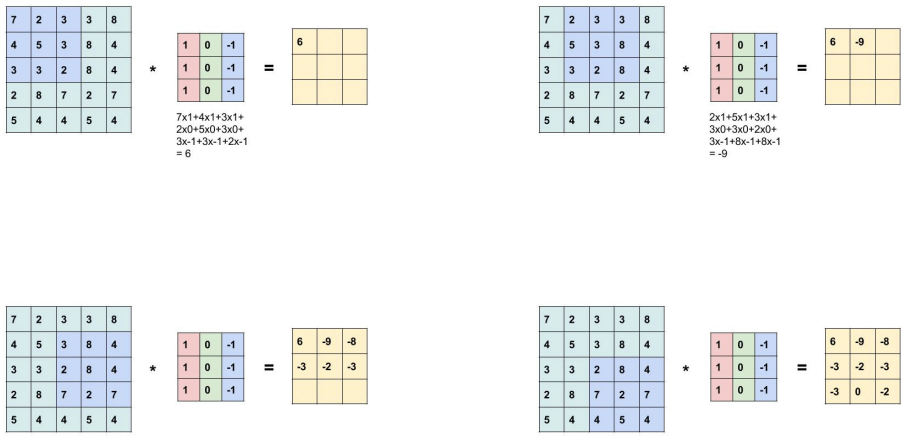


FIGURA 19 Ejemplo de aplicación de la función *conv* usada por una capa convolucional.

capa a cada partición temporal de la entrada.

5.3.3 | Bloque RNN

La función encargada de crear un bloque RNN se llama ***create_rnn_block***, este tipo de bloque está compuesto por:

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 usa un bucle para iterar sobre cada paso de una secuencia, manteniendo un estado interno que codifica información sobre los pasos por los que ya ha iterado. 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 *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).
- **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).

Además, para cada uno de las implementaciones descriptas, se agrega la posibilidad de usar la 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. Esta capa ya cuenta con *dropout*.

Capa de regularización (*layer normalization*) este tipo de capa normaliza las activaciones de la capa anterior para cada ejemplo de forma independiente (no dependiendo de los valores de cada ejemplo dentro de un *batch*), en resumen, aplica una transformación que mantiene la media de la activación de cada ejemplo cerca de 0 y la desviación estándar cerca de 1.

5.3.4 | Bloque DNN

La función encargada de crear un bloque DNN se llama ***create_dnn_block***, este tipo de bloque está compuesto por:

Capa densa este tipo de capa está compuesto por una función de activación a y dos vectores, W llamado de pesos y b llamado *bias*. Implementa la función $a(i \cdot W + b)$ donde i son los valores de la entrada de la red.
Capa de regularización (*layer normalization*) este tipo de capa normaliza las activaciones de la capa anterior para cada ejemplo de forma independiente (no dependiendo de los valores de cada ejemplo dentro de un *batch*), en resumen, aplica una transformación que mantiene la media de la activación de cada ejemplo cerca de 0 y la desviación estándar cerca de 1.
Capa de *dropout* este tipo de capa selecciona valores de la activación de la capa anterior de forma aleatoria y los transforma en 0 durante el entrenamiento, esto ayuda a prevenir el *overfitting*.

5.3.5 | 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*** que se describe en la sección 7.2.14. En esta función se usa la API funcional de Tensorflow[43] que permite personalizar el modelo para poder manejar múltiples entradas y múltiples salidas.

En esta función 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 entrenamiendo, por defecto la lista solo incluye el error absoluto medio (mae).
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.

5.4 | Entrenamiento

Un modelo de redes neuronales tiene un alto número de parámetros (también llamados pesos de cada capa) que deben ser adaptados sobre un conjunto de datos para que, al recibir una entrada X y aplicarle un varias transformaciones matemáticas, se llegue a un valor de salida $\overline{y}$ que aproxime el valor y verdadero correspondiente con X.

5.4.1 | 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 el proyecto se probaron varias funciones que se comentan a continuación.

MSE

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

Huber

La función de pérdida Huber mide la diferencia entre la predicción y el valor a predecir de forma que es cuadrática para valores pequeños y lineal para valores fuera de cierto umbral, de esta forma que se vuelve resistente a que diferencias muy grandes influyan demasiado en el error. Esta función se calcula como:

$$\text{Huber}_\delta(y, \overline{y}) = \begin{cases} 0{,}5 \cdot (y_i - \overline{y}_i)^2 & \text{si } |y_i - \overline{y}_i| \leq \delta \\ 0{,}5 \cdot \delta^2 + \delta \cdot (|y_i - \overline{y}_i| - \delta) & \text{si } |y_i - \overline{y}_i| < \delta \end{cases}$$

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

Predicción de varios objetivos

En los casos en los que un modelo se entrena para predecir varios objetivos al mismo tiempo, la función de pérdida se corresponde con la suma promediada de las funciones de pérdida específicas de cada uno de los objetivo.

5.4.2 | Entrenamiento distribuido

En un principió se intentó entrenar los modelos sobre el cluster de Ceibal, a continuación se comentan todas las bibliotecas probadas para realizar el entrenamiento distribuido de un modelo de *Tensorflow*. Hay dos formas de distribuir el entrenamiento sobre muchos nodos:

- **Paralelismo de datos** donde un mismo modelo se replica en múltiples nodos, cada uno procesa distintas particiones del conjunto de datos y luego juntan sus resultados. 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 *all-reduce* y el entrenamiento asincrónico mediante una arquitectura 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 tiene varias ramas que se juntan al final.

En el marco del proyecto se usa **paralelismo de datos sincrónico**, en este modo de paralelismo, cada nodo 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 sincronía, en la figura 20 se muestra un esquema de este comportamiento.

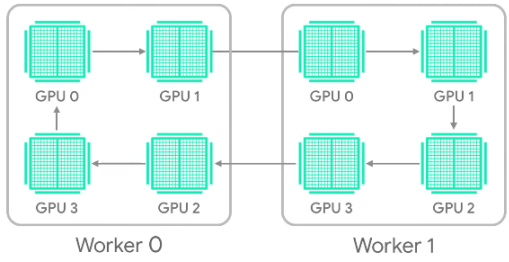


FIGURA 20 Esquema de paralelismo *all-reduce*.

A continuación se listan las etapas seguidas por todas las bibliotecas usadas:

1. **Configuración del cluster** se configura el cluster donde se ejecutará el entrenamiento desplegando $N-1$ nodos trabajadores y 1 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, realiza *back-propagation* y continua con el entrenamiento. Esta etapa se muestra en la figura 21.

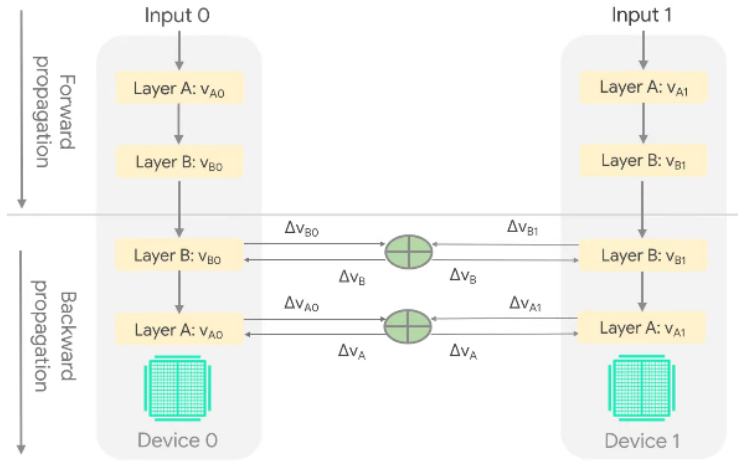


FIGURA 21 Esquema de *all-reduce* durante el entrenamiento.

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 por defecto de *Spark* en el ambiente de Facultad de Ingeniería. Durante su uso se notó que tarda bastante tiempo en reportar errores de ejecución.

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 permite 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*

Esta biblioteca fue creada por *Yahoo* para realizar el entrenamiento de forma distribuida integrándolo con varios ecosistemas y en particular *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*.

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

Comparación de bibliotecas

En las secciones anteriores se describieron las particulares de cada una de las bibliotecas utilizadas, en el cuadro 26 se muestra una comparación entre todas las bibliotecas para poder elegir una de forma fácil dependiendo de las características particulares del ambiente de ejecución.

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 y recuperación del entrenamiento
TensorFlowOnSpark	*MultiWorkerMirroredStrategy*	No
Spark TensorFlow Distributor	*MultiWorkerMirroredStrategy*	Si
Horovod	*DistributedOptimizer*	Si

CUADRO 26 Comparación de bibliotecas de distribución

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[31] que cuenta con GPUs en sus nodos. El entrenamiento también cambió la forma de distribución ya que, en vez de entrenar un mismo modelo en varios subconjuntos de datos, se pasó a entrenar en cada nodo un modelo distinto sobre el conjunto total de datos.

5.5 | Búsqueda de hiperparámetros

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*.

5.5.1 | 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.[12]

Parte de las razones de que *random search* supere a *grid search* es que, tipicamente, 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 22), 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.

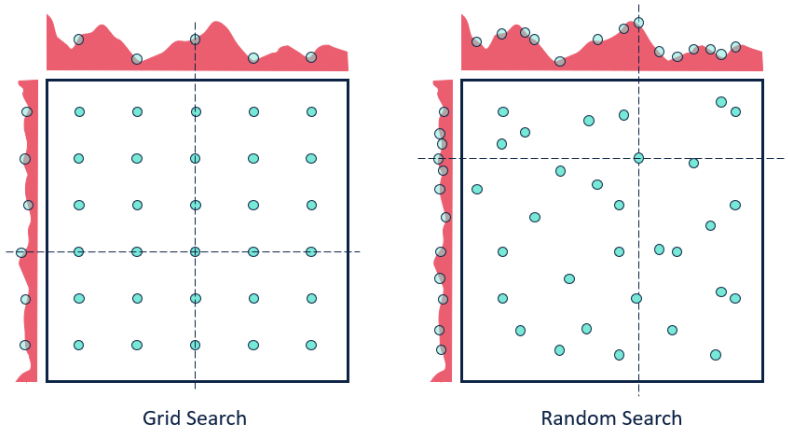


FIGURA 22 Comparación de los métodos *grid search* y *random search* en la búsqueda de los valores posibles de dos hiperparámetros. En el eje x, se muestra un hiperparámetro importante cuya evaluación es mayor (en rojo) y en el eje y otro hiperparámetro de menor importancia.

Tree-structured Parzen Estimator (TPE)

Este método de búsqueda es un algoritmo basado en sequential model-based optimization (SMBO)[13], 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* 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.

La función de subrogación es particular para el caso de TPE, en este estimador se construye usando como base la regla de Bayes 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 menos 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 mejora mucho más rápido lo que repercute en la cantidad de conjuntos de hiperparámetros necesarios (este comportamiento se muestra en la figura 23).

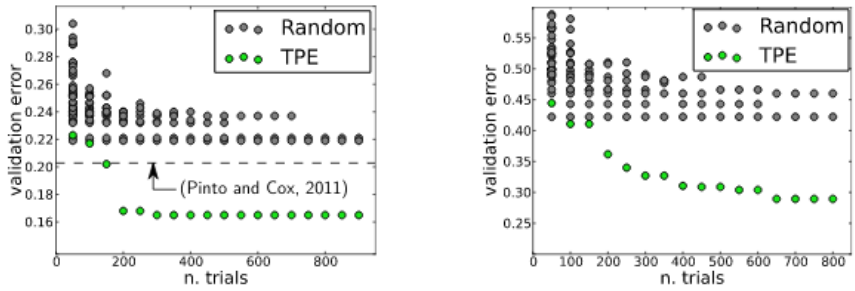


FIGURA 23 Comparación de los métodos *random search* y TPE en la optimización de una función objetivo sobre dos conjuntos, Los puntos grises representan el error más bajo obtenido de varias ejecuciones de *random search* y los puntos verdes representan el error más bajo obtenido por una ejecución de TPE.[11]

5.5.2 | Implementación

La implementación de la búsqueda del mejor modelo posible, se usó la biblioteca Optuna[2] en conjunción con las técnicas ya descritas de selección de hiperparámetros mediante TPE (5.5.1).

Esta biblioteca encapsula cada proceso de búsqueda en un **estudio** que puede ser almacenado en una base de datos relacional (RBD), en particular, se usa la base de datos postgresql que permite el acceso concurrente y en red. Estas características combinadas con la ejecución sobre el cluster de supercomputación ClusterUY[31] permiten la paralelización de la búsqueda en hasta cuatro nodos.

No solo permite obtener el mejor modelo posible de *deep learning*, sino que también se puede usar con algoritmos tradicionales de aprendizaje supervisado (sección 5.6.1).

Esta biblioteca necesita una función objetivo para decidir que modelo es mejor entre los que prueba, en este proyecto, se eligió la misma función que la función de pérdida del modelo de redes neuronales.

5.6 | Experimentos y resultados

Los experimentos que se realizaron para conseguir el mejor modelo predictor posible, se dividieron según el conjunto de datos usado, estas variantes se comentaron en la sección 4.6, además, cada agregación de datos se particionó usando la columna **date** y con selección aleatoria (sección 4.9), por lo tanto, cada subconjunto obtenido contenía todos los registros de de varios días no consecutivos.

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.

5.6.1 | Baseline

Como modelo base contra el que buscar mejoras, se usó el modelo *Extremely Randomized Trees Regressor* (con implementación en la biblioteca sklearn[20]). Este modelo entrena un conjunto de árboles de decisión en varios subconjuntos de datos y promedia las predicciones para mejorar la capacidad predictiva y controlar el sobreajuste, se diferencia del *Random Forest Regressor* en la forma en la que se decide el punto de división del rango de una variable, tomando un subconjunto de *features* (seleccionado aleatoriamente) el rango usado para particionar el espacio de una variable se hace de forma aleatoria, en el caso del *Random Forest Regressor* se hace la división buscando el rango más representativo, este cambio permite la aceleración del proceso de entrenamiento obteniendo resultados iguales o mejores.[23]

Para hacer la comparación lo más justa posible, se realizó una búsqueda de hiperparámetros de este modelo siguiendo la misma metodología que el modelo de redes neuronales (sección 5.5.2).

5.6.2 | Experimentos con conjunto temporal por ruee

En este grupo de experimentos se usó la agregación de datos temporal por ruee (sección 4.6.1).

En primera instancia se buscó el mejor modelo encargado de la transformación de consultas DNS en tráfico y las primeras variantes probadas fueron usando el conjunto con frecuencia temporal de 5 minutos.

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.
ruee El identificador del centro desde donde se hizo la consulta.
departamento El departamento al que pertenece el centro.
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.

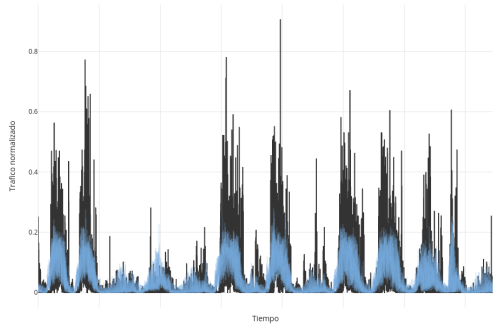
Además se probó una variante (experimento 2) agregando a la entrada:

Agregación de la columna category_1 La cantidad de consultas realizadas para cada una de las categorías más importantes.

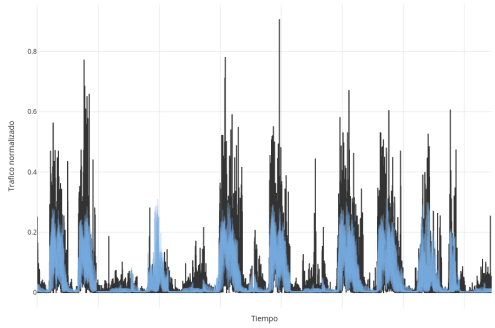
Estos experimentos fueron replicados también usando el conjunto con frecuencia temporal de 15 minutos para evaluar si una reducción de datos influiría negativamente en la predicción, los resultados de estas pruebas se muestran en la figura 24 donde se compara la predicción de los modelos resultantes de cada experimento con el valor real y la predicción del *baseline*.

Cada experimento se realiza con las dos funciones de pérdida antes descritas (sección 5.4.1), para facilitar la lectura, solo se muestra la variante con mejor resultado de cada experimento y se indica que función se usa.

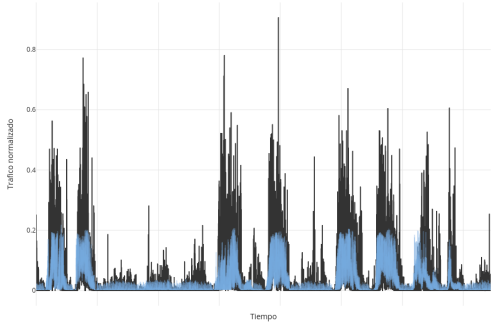
Se observa que el modelo del experimento 3 puede predecir mejor los valores altos que el *baseline* pero a su vez sobrestima algunos valores, por otro lado, el modelo del experimento 4 no presenta diferencias generales frente al *baseline* por esto se intentó mejorar el primero. Se hipotetizó que el comportamiento mostrado por el experimento 3 se debe a la influencia de las *features* temporales en el modelo.



(a) Comparación de datos observados con *baseline*.



(b) Comparación de datos observados con experimento 3 (Huber).

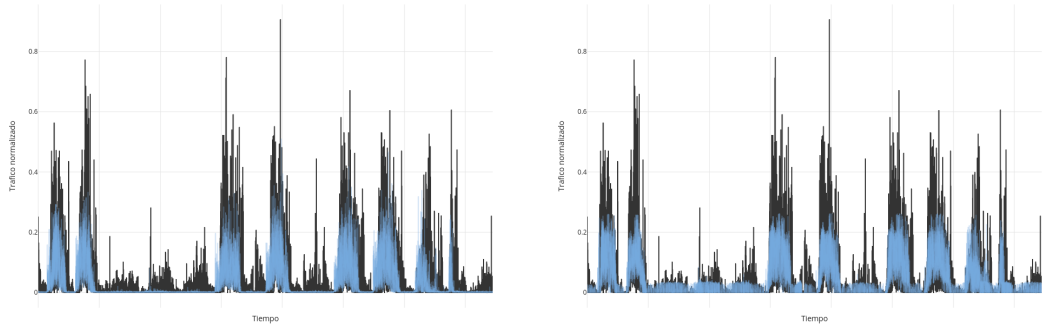


(c) Comparación de datos observados con experimento 4 (Huber).

FIGURA 24 Comparación de datos observados en el conjunto de *test* (en negro) con las predicciones del *baseline* y de los experimentos 3 y 4 (en azul).

Si se analizan los datos del conjunto de *test*, se puede ver que la mayoría de los valores observados se encuentran debajo del $0,15$ (quintil 90), este comportamiento provoca que las métricas usadas para la validación no reflejen los resultados esperados, pues, si la predicción del modelo es muy buena para valores bajos ($\leq 0,15$) el error cuadrático medio será muy bajo, a pesar de que no se logra predecir el objetivo del proyecto, valores altos de tráfico.

Para el experimento 5, se quitaron las *features* temporales (*day*, *hour* y *minute*), en este caso la predicción del modelo generado ya no presenta los valores altos que se mostraban antes pero aparece un nuevo problema, presenta un desfasaje frente a los datos reales y se pierde capacidad de predicción frente a series de valores bajos (menores a $0,15$). Para solucionar esto, se agregan las columnas que registran la evolución dentro de la hora anterior (EWMA4.7) en el experimento 6. En la figura 25 se observan los resultados de los nuevos experimentos, en el experimento 6 se soluciona el desfasaje a costa de perder poder de predicción para series de valores altos.



(a) Comparación de datos observados con experimento 5 (MSE).

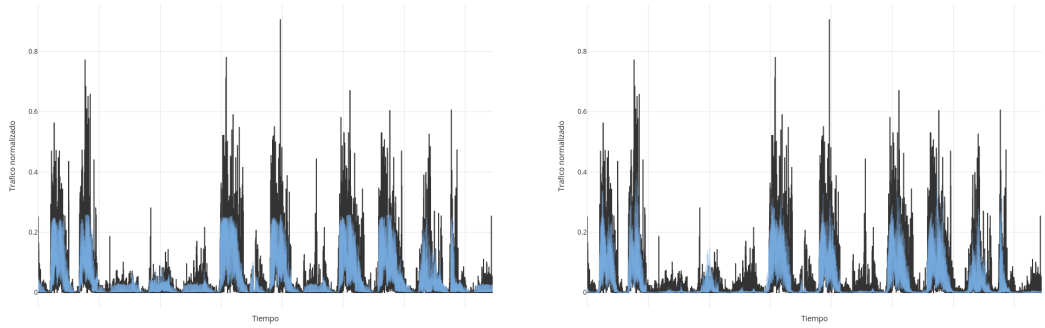
(b) Comparación de datos observados con experimento 6 (Huber).

FIGURA 25 Comparación de datos observados en el conjunto de *test* (en negro) con las predicciones de los experimentos 5 y 6 (en azul).

Buscando solucionar el problema de la predicción de valores altos, para el experimento 7 se volvieron a agregar las *features* temporales y se quitó la *feature* ruee, esto porque el identificador del local podría estar introduciendo demasiado ruido dentro del modelo siendo que el tráfico puede ser independiente del local donde se mide.

Como se ve en la figura 25, usando la función de pérdida Huber se mejora sustancialmente la predicción de valores $\leq 0,15$ a costa de reducir la predicción de valores altos, si se compara con el mismo experimento con función de pérdida MSE, se puede ver el efecto contrario, por esto, se plantea un modelo final que toma lo mejor de estas variantes:

$$\text{Predicción}(X) = \begin{cases} \text{máx}(\text{Predicción}_H(X), \text{Predicción}_M(X)) & \text{si Predicción}_H(X) \geq 0,15 \\ \text{Predicción}_H(X) & \text{si Predicción}_H(X) < 0,15 \end{cases}$$



(a) Comparación de datos observados con experimento 7 (Huber).

(b) Comparación de datos observados con experimento 7 (MSE).

FIGURA 26 Comparación de datos observados en el conjunto de *test* (en negro) con las predicciones del experimento 7 (en azul).

Donde Predicción$_H$ es la predicción dada por el modelo generado usando la función de pérdida Huber y Predicción$_M$ su análoga usando la función de pérdida MSE. La predicción final de este modelo se muestra en la figura 27.

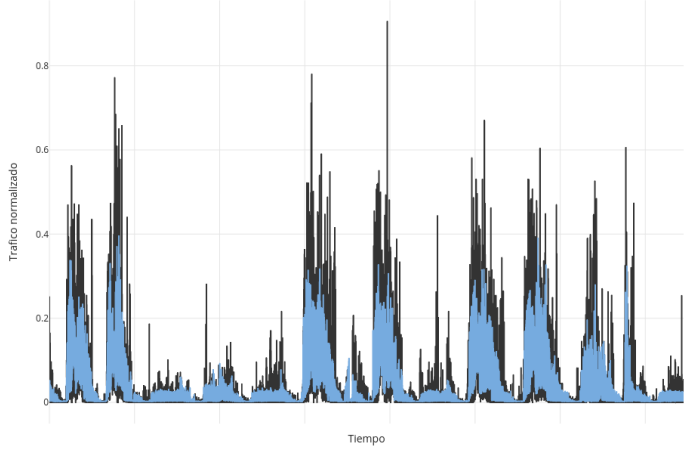
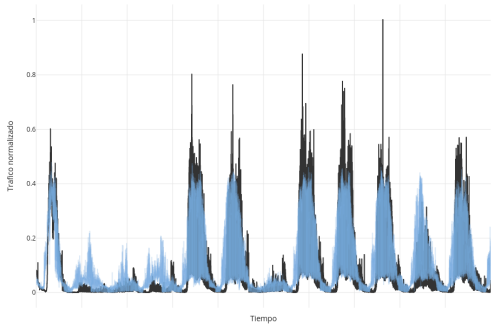


FIGURA 27 Comparación de datos observados en el conjunto de *test* (en negro) con las predicciones del modelo final (en azul).

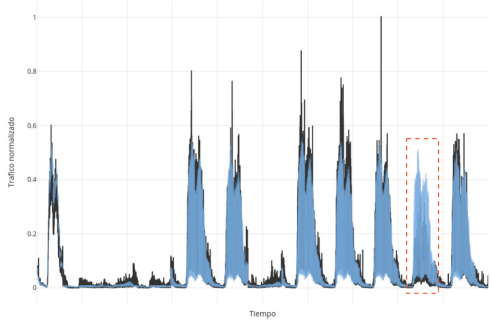
5.6.3 | Experimentos con conjunto temporal por subsistema

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 menos específicos a predecir.

Al igual que en el grupo de experimentos anterior, se probó una variante análoga al experimento 1 quitando las columnas ruee y departamento sobre el conjunto con frecuencia temporal de 5 minutos, los resultados del *baseline* y del modelo elegido se pueden observar en la figura 28, se observa que la predicción del *baseline* sobreestima los valores bajos ($\leq 0{,}15$) y no estima correctamente los valores altos, mientras que la predicción del modelo se ajusta mejor pero sobreestima una secuencia de valores altos (en rojo).



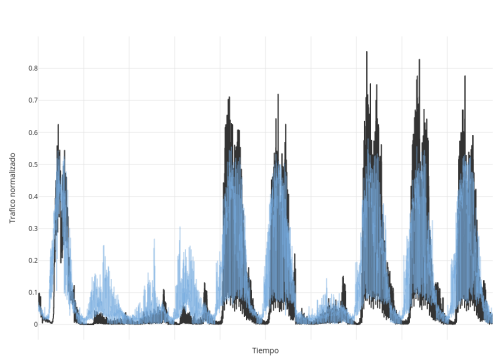
(a) Comparación de datos observados con *baseline*.



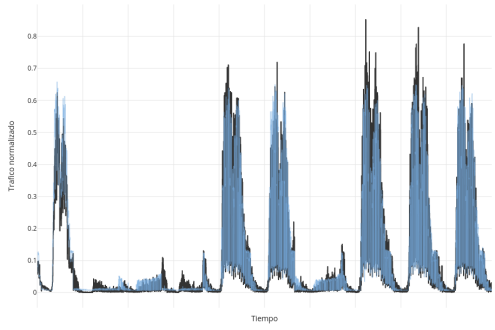
(b) Comparación de datos observados con experimento 1 (MSE).

FIGURA 28 Comparación de datos observados en el conjunto de *test* (en negro) con las predicciones del *baseline* y del experimento 1 (en azul).

El siguiente experimento se hace replicando el anterior sobre el conjunto de datos con frecuencia temporal de 15 minutos, el resultado y comparación de este experimento contra el *baseline* y los datos originales se puede ver en la figura 29, se observa que la predicción del *baseline* mantiene los problemas notados en el experimento anterior mientras que el modelo generado presenta una gran mejoría, no solo mejorando la predicción de valores bajos sino que también ajustandose satisfactoriamente en la predicción de picos.



(a) Comparación de datos observados con *baseline*.



(b) Comparación de datos observados con experimento 2 (Huber).

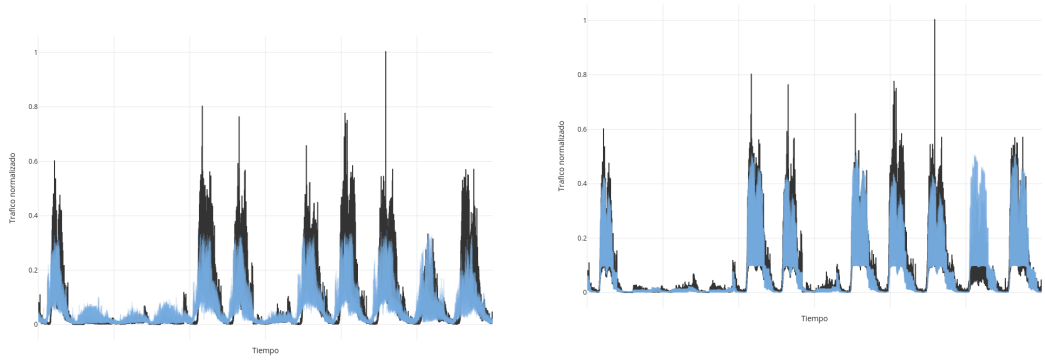
FIGURA 29 Comparación de datos observados en el conjunto de *test* (en negro) con las predicciones del *baseline* y del experimento 2 (en azul).

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

5.6.4 | Experimentos con conjunto temporal por subsistema y departamento

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.

La metodología inicial de los otros experimentos se volvió a repetir, se probó una variante análoga al experimento 1 quitando la columna ruee sobre el conjunto con frecuencia temporal de 5 minutos, los resultados del *baseline* y del modelo elegido se pueden observar en la figura 30, se observa que la predicción del *baseline* sobreestima los valores bajos y no subestima los valores altos, mientras que la predicción del modelo se ajusta de mejor forma pero sobreestima una secuencia de valores altos, repitiendo el comportamiento del grupo de experimentos anterior.

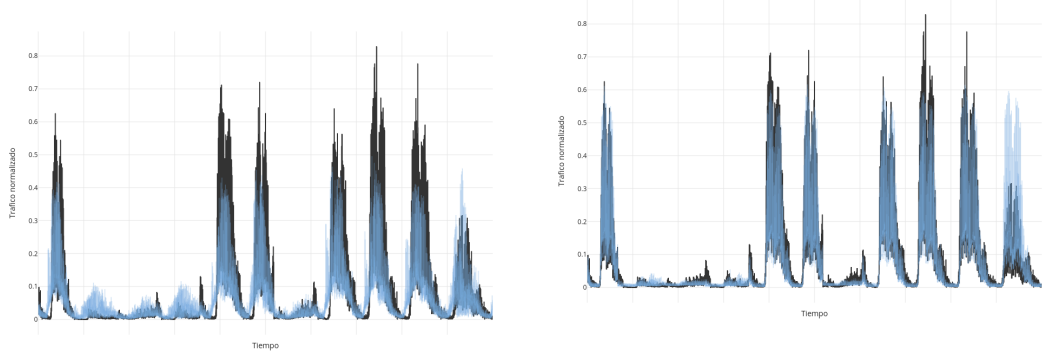


(a) Comparación de datos observados con *baseline*.

(b) Comparación de datos observados con experimento 1 (Huber).

FIGURA 30 Comparación de datos observados en el conjunto de *test* (en negro) con las predicciones del *baseline* y del experimento 1 (en azul).

El siguiente experimento se hizo replicando el anterior sobre el conjunto de datos con frecuencia temporal de 15 minutos, el resultado y comparación de este experimento contra el *baseline* y los datos originales se puede ver en la figura 31, se observa que la predicción del *baseline* mantiene los problemas notados en el experimento anterior mientras que el modelo se ajusta casi perfectamente a los datos observados aunque presenta sobreestimación de la última secuencia de valores altos.



(a) Comparación de datos observados con *baseline*.

(b) Comparación de datos observados con experimento 2 (Huber).

FIGURA 31 Comparación de datos observados en el conjunto de *test* (en negro) con las predicciones del *baseline* y del experimento 2 (en azul).

6 | CONSLUSIONES

Varias conclusiones pudieron obtenerse de este proyecto, tanto del procesamiento de datos en grandes cantidades como de la generación de modelos capaces de transformar consultas DNS en tráfico, a continuación se discuten:

6.1 | 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 y la integración con bibliotecas como Koalas es muy útil para simplificar la curva de aprendizaje necesaria para su uso. 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 que también obliga 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 clusters.

6.2 | Sobre el uso de aprendizaje supervisado

La integración de esta herramienta con otras de *deep learning* no existe de manera oficial (como si lo hace para algoritmos de *machine learning* clásico) y debe ser lograda a través de bibliotecas de terceros. Nuestra 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 sea 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. Frente a la cuestión de la estabilidad, no logramos entender completamente si se debió a que la instancia de la plataforma en el cluster de Ceibal no está correctamente configurada para tareas de muy largo procesamiento y uso intensivo de CPUs o si se debió a las bibliotecas que usamos.

Pasando más específicamente a la creación de modelos capaces de predecir tráfico a partir de consultas DNS, logramos crear modelos que superan ampliamente a cada *baseline* realista utilizado. Vimos que la frecuencia temporal con la que se trabaje puede influir en la dificultad del problema, en particular, concluimos que agregar los datos hasta una frecuencia razonable (15 minutos) ayudó a la obtención de modelos más rápidamente 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 el tráfico para cada local es más difícil que transformar el tráfico de una zona y/o subsistema. Creemos que este fenómeno se da por el aumento desmedido de datos donde el tráfico total es menor al 20 % del tráfico que se da en los picos. Creemos que 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 departamento o por departamento y subsistema) tienen mayor nivel de confiabilidad y los más específicos aportarían una mirada más enfocada en detectar qué local puede ser problemático en vez de dar una cifra concreta de tráfico.

REFERENCIAS

[1] *Adaptative query execution - Spark 3*. Mayo de 2020. URL: https://databricks.com/blog/2020/05/29/adaptive-query-execution-speeding-up-spark-sql-at-runtime.html (visitado 18-06-2021).

[2] 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.

[3] *Apache Ambari documentation*. URL: https://cwiki.apache.org/confluence/display/AMBARI/Ambari.

[4] *Apache Arrow*. URL: https://arrow.apache.org/docs/index.html.

[5] *Apache Arrow - Spark*. URL: https://spark.apache.org/docs/latest/sql-pyspark-pandas-with-arrow.html#enabling-for-conversion-tofrom-pandas.

[6] *Apache HDFS documentation*. URL: https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html.

[7] *Apache Hive documentation*. URL: https://cwiki.apache.org/confluence/display/Hive/Home.

[8] *Apache Spark documentation*. URL: https://spark.apache.org/docs/latest/.

[9] *Apache YARN documentation*. URL: https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html.

[10] *Apache Zeppelin documentation*. URL: https://zeppelin.apache.org/docs/0.8.2/.

[11] 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, pp. I–115–I-123.

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

[13] 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.

[14] *Cisco Umbrella, categories*. URL: https://docs.umbrella.com/deployment-umbrella/docs/content-categories.

[15] *Conda-pack documentation*. URL: https://conda.github.io/conda-pack/.

[16] *Data Classes - Python*. URL: https://docs.python.org/3/library/dataclasses.html.

[17] *Default Index Type - Koalas*. URL: https://koalas.readthedocs.io/en/latest/user_guide/options.html#default-index-type.

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

[19] *Dynamic Allocation - Spark*. URL: https://spark.apache.org/docs/latest/configuration.html#dynamic-allocation.

[20] *ExtraTreesRegressor - sklearn*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.ExtraTreesRegressor.html (visitado 18-06-2021).

[21] *Forticlient*. URL: https://www.forticlient.com/downloads.

[22] *geopy, python module*. URL: https://geopy.readthedocs.io/en/stable.

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

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

[25] *Horovod documentation*. URL: https://github.com/horovod/horovod.

[26] *Koalas: pandas API on Apache Spark*. URL: https://koalas.readthedocs.io/en/latest/.

[27] *Level of Parallelism - Spark*. URL: https://spark.apache.org/docs/latest/tuning.html#level-of-parallelism.

[28] 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).

[29] *List of DNS record types*. URL: https://en.wikipedia.org/wiki/List_of_DNS_record_types.

[30] *Logs Formats and Versioning, DNS Logs*. URL: https://docs.umbrella.com/deployment-umbrella/docs/log-formats-and-versioning#section-dns-logs.

[31] 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.

[32] *Plotly Python Open Source Graphing Library*. URL: https://plot.ly/python/.

[33] *PyDomainExtractor, python module*. URL: https://github.com/Intsights/PyDomainExtractor.

[34] *PySpark User-Defined Functions*. URL: https://docs.databricks.com/spark/latest/spark-sql/udf-python.html.

[35] *Python bindings, Apache Arrow*. URL: https://arrow.apache.org/docs/python/.

[36] *Python Data Analysis Library*. URL: https://pandas.pydata.org/.

[37] *Sandbox Deployment and Install Guide*. URL: https://www.cloudera.com/tutorials/sandbox-deployment-and-install-guide/3.html.

[38] *Sandbox Port Forwards*. URL: https://www.cloudera.com/tutorials/hortonworks-sandbox-guide/3.html.

[39] 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.

[40] *Spark Configuration*. URL: https://spark.apache.org/docs/2.3.1/configuration.html.

[41] *Spark TensorFlow Distributor documentation*. URL: https://github.com/tensorflow/ecosystem/tree/master/spark/spark-tensorflow-distributor.

[42] *Sunsetting Python 2*. URL: https://www.python.org/doc/sunset-python-2/.

[43] *Tensorflow - Functional API*. URL: https://www.tensorflow.org/guide/keras/functional.

[44] *TensorFlow documentation*. URL: https://www.tensorflow.org/api_docs/.

[45] *TensorFlowOnSpark documentation*. URL: https://github.com/yahoo/TensorFlowOnSpark/blob/master/README.md.

[46] *TFRecords - Tensorflow*. URL: https://www.tensorflow.org/tutorials/load_data/tfrecord.

[47] 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).