UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA

Escuela de Ingeniería Informática

**TRABAJO FIN DE GRADO:**

# “Estimación del estado del flujo de tráfico mediante preprocesado y minería de datos.

# Aplicación de Dataset de posiciones GPS de taxis de Porto”


Autor: Aitor del Pino Saavedra Hernández

Tutores: Javier Jesús Sánchez Medina y Luis Moreira Matías

Grado en Ingeniería Informática

Escuela de Ingeniería Informática

Universidad de Las Palmas de Gran Canaria


Julio 2016

*Si buscas resultados distintos,*

*no hagas siempre lo mismo.*

Albert Einstein.

# AGRADECIMIENTOS

A mis padres, Pepe y Begoña, quienes siempre me han apoyado y me han aconsejado de la mejor manera posible para lograr todas mis metas propuestas.

A mi hermano, Adrián, por ayudarme siempre en todo lo que ha podido.

A mi compañero de piso, Jorge, por aguantarme y apoyarme durante el último año de mí carrera el cual no ha sido fácil debido a la carga que este supone.

Al grupo de la especialidad de computación, porque entre todos hemos intentado que este último año fuera más sencillo y más confortante.

También he de agradecer la labor y el tiempo prestado a mi tutor por aconsejarme y ayudarme con el proyecto.

Por último le doy las gracias a mi novia por haberme ayudado estos últimos cuatros años de mi vida. Ella ha sido fundamental, ya que siempre ha estado a mi lado sobretodo en los peores momentos donde los resultados de las asignaturas no eran los mejores. Así como le agradezco la ayuda prestada en la realización de este proyecto.

# ÍNDICE

# ÍNDICE DE FIGURAS

# ÍNDICE DE TABLAS

# 1. CAPÍTULO I: INTRODUCCIÓN

El presente trabajo consiste en realizar un pre-procesado de datos con el objetivo de dividir la red de Oporto en celdas uniformes por las posiciones del *Global Positioning System*, Sistema de Posicionamiento Global (GPS). Una vez dividido el espacio, dividiríamos el tiempo, y por cada celda, dividiríamos los datos por tramos temporales. Para una celda concreta y tramo temporal predeciremos el estado de ocupación entre libre, sincronizado o congestionado. Para ello haremos uso de algoritmos de clasificación de minería de datos.

Con el fin de comprender la realización de dicho proyecto lo dividiremos en 7 capítulos.

En primer lugar veremos una contextualización teórica de los fundamentos más relevantes del trabajo, haciendo una breve introducción al concepto de minería de datos y al proceso *Knowledge Discovery in Databases*, Descubrimiento de conocimiento en bases de datos (KDD), junto a ello también veremos algunos tipo de problemas resueltos con minería de datos, con el fin de comprender la importancia que tiene. Además veremos un segundo apartado realizaremos una introducción al flujo de tráfico indicando qué es y la importancia que este tiene. Finalmente nos centraremos en una de las categorías con la que predeciremos el estado de ocupación de cada celda, la congestión.

El tercer capítulo consiste en comprender los datos que vamos a utilizar con la intención de situarnos en el problema que queremos resolver.

En el siguiente capítulo se explica cómo se ha aplicado la preparación de datos sobre el dataset que vamos a utilizar para predecir el flujo del tráfico en la ciudad de Oporto. Esta parte contiene la selección de los datos, la limpieza de los datos, así como una transformación de los datos con el objetivo de obtener un fichero al que poder aplicar minería de datos. Además dedicaremos un apartado para comprender el formato *Attribute-Relation file format*, Formato de archivo de atributo-relación (ARFF), ya que este tipo de formato es el que se utilizar en *Waikato Environment for Knowledge Analysis*, entorno para análisis del conocimiento de la Universidad de Waikato (Weka).

Este capítulo lo consideramos fundamental, ya que, obtenemos por cada celda y tramo temporal el estado de ocupación de vehículos.

El programa Weka ocupará el quinto capítulo del proyecto. Weka es una plataforma de software para el aprendizaje automático y la minería de datos escrito en Java y desarrollado en la Universidad de Waikato. Haciendo uso de este programa aplicaremos minería de datos sobre el fichero que hemos obtenido al aplicar la preparación de los datos con el objetivo de obtener un modelo que nos predice el flujo del tráfico de la ciudad de Oporto. Como veremos en este capítulo tendremos tres experimentos donde se realizarán una serie de pruebas con la finalidad de construir los mejores modelos.

El sexto capítulo se explica cómo se crea la interfaz la cual se utiliza para visualizar el modelo que nos ha construido Weka. De esta forma podremos interpretar de una manera gráfica los resultados obtenidos.

En el último capítulo se expone las conclusiones más importantes y las propuestas de trabajos futuros que podrían ampliar lo estudiado en el presente proyecto.

Con todo esto perseguimos el objetivo de predecir el tránsito vehicular en un instante de tiempo concreto para la red de Oporto. De esta manera podríamos lograr una serie de beneficios para la sociedad como por ejemplo una mayor fluidez del tráfico, disminuir el tiempo que tarda en realizar un trayecto un vehículo y disminuir la emisión de CO2 de los vehículos.

**COMPETENCIAS**

Las competencias adicionales que cumple el presente proyecto son las siguientes:

- CII015 Conocimiento y aplicación de los principios fundamentales y técnicas básicas de los sistemas inteligentes y su aplicación práctica.
- CP01 Capacidad para tener un conocimiento profundo de los principios fundamentales y modelos de la computación y saberlos aplicar para interpretar, seleccionar, valorar, modelar, y crear nuevos conceptos, teorías, usos y desarrollos tecnológicos relacionados con la informática.
- CP03 Capacidad para evaluar la complejidad computacional de un problema, conocer estrategias algorítmicas que puedan conducir a su resolución y

recomendar, desarrollar e implementar aquella que garantice el mejor rendimiento de acuerdo con los requisitos establecidos.

- CP04 Capacidad para conocer los fundamentos, paradigmas y técnicas propias de los sistemas inteligentes y analizar, diseñar y construir sistemas, servicios y aplicaciones informáticas que utilicen dichas técnicas en cualquier ámbito de aplicación.
- CP07 Capacidad para conocer y desarrollar técnicas de aprendizaje computacional y diseñar e implementar aplicaciones y sistemas que las utilicen, incluyendo las dedicadas a extracción automática de información y conocimiento a partir de grandes volúmenes de datos.

# 2. CAPÍTULO 2: ESTADO DEL ARTE

En este capítulo explicaremos el proceso KDD, el cual se aplica en su plenitud al proyecto. Posteriormente hablaremos de una de las etapas de este proceso denominada minería de datos. Para concluir definiremos que es el flujo del tráfico y que es la cogestión vehicular, en esta última indicaremos una serie de desventajas que afectan a la sociedad.

## 2.1. PROCESO KDD

El término proceso KDD se utiliza para referirse al proceso de extracción automatizada de conocimiento a partir de grandes volúmenes de datos.

El conocimiento extraído por el proceso KDD ha de poseer las cuatro características que se presentan a continuación:

- No trivial: no sirve de nada extraer conocimiento que ya es conocido por todas las personas.
- Implícito: se encuentra oculto en los datos.
- Previamente desconocido: este conocimiento extraído no aporta nada si ya ha sido descubierto anteriormente.
- Útil: el conocimiento extraído debe servir para algo.

De la misma manera según Usama Fayyad (1996) el proceso KDD es "El proceso no trivial de identificar patrones válidos, nuevos, potencialmente útiles y en última instancia comprensible en los datos".

El proceso KDD está compuesto por diferentes fases:

- Recopilación de datos. En esta etapa se determinan las fuentes de datos y el tipo de información a utilizar. Además todos los datos procedentes de diferentes fuentes, se integran en un mismo y único repositorio de datos, conocido como almacén de datos o data warehouse. El resultado de esta fase es precisamente el almacén de datos.
- Selección, limpieza y transformación de datos. Sobre los datos almacenados en el data warehouse todavía no se puede aplicar minería de datos debido a que puede haber datos que no estén limpios, atributos irrelevantes, etc. Por ello en esta fase se seleccionan los datos que vamos a utilizar del almacén de datos. El resultado de esta fase es la denominada vista minable, que es un subconjunto

limpio y transformado de datos sobre el que ya se puede aplicar diferentes técnicas de minería de datos.

- Minería de datos. Una vez obtenido el conjunto de datos al cual le podemos aplicar minería de datos, el siguiente paso consiste en aplicar técnicas concretas de minería de datos para obtener modelos. El resultado de esta fase son los modelos que obtenemos.
- Interpretación y evaluación de modelos. Los modelos obtenidos en la minería de datos son evaluados con el objetivo de comprobar la calidad de los mismos. Además estos modelos son interpretados y a partir de ellos se obtiene el conocimiento.

El proceso KDD es iterativo, pues consta de varios pasos que pueden llegar a tener que repetirse para extraer la información óptima. Además existe la necesidad de una persona experta ya que su intervención es imprescindible en algunos de los pasos en los que se divide el proceso KDD.

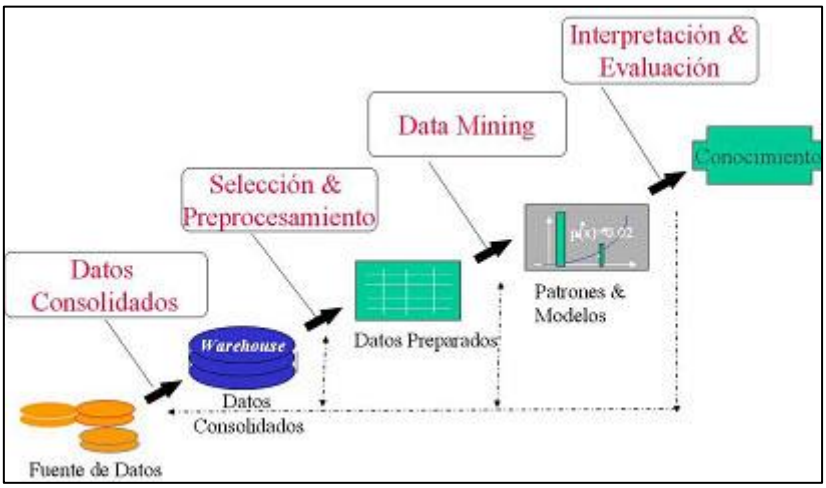



Figura 1: Proceso KDD.

## 2.2. MINERÍA DE DATOS

En las últimas décadas la cantidad de datos almacenados en bases de datos ha aumentado exponencialmente. Gran parte de esta información es histórica, es decir, representa transacciones o situaciones que se han producido en el pasado. Las empresas, instituciones, organizaciones, etc suelen tomar decisiones a partir experiencias pasadas, por tanto esta información almacenada puede ser muy útil. El problema es que ser humano no tiene la capacidad de analizar estos grandes volúmenes de datos por ello apareció el concepto de minería de datos.

La minería de datos es una disciplina informática que estudia el análisis de grandes cantidades de datos con el objetivo de obtener conocimiento a partir de ellos. De igual forma, que mediante la minería tradicional, se persigue la obtención de minerales a partir de la corteza terrestre o del mal, la minería de datos persigue la obtención de conocimiento a partir de los datos. Siguiendo el símil, los datos analizados serían la corteza terrestre y el conocimiento obtenido sería el mineral valioso.

Para conocer el origen del término minería de datos hay que remontarse años atrás. En la década de 1960, los estadísticos utilizaron los términos data fishing (buscar o "pescar" datos) o data dredging (dragado de datos) para referirse al análisis de datos. Con el tiempo, esos términos desaparecieron, dejando lugar, en la década de 1990, al término data mining (minería de datos), con el que se conoce actualmente el campo de la informático que estudia la obtención de conocimiento a partir de los datos por medio de su análisis.

Las técnicas de minería de datos pueden ser de dos tipos:

- Métodos descriptivos: buscan patrones interpretables para describir datos. Son los siguientes: clustering, descubrimiento de reglas de asociación y descubrimiento de patrones secuenciales.
- Métodos predictivos: usan algunas variables para predecir valores futuros o desconocidos de otras variables. Son los siguientes: clasificación, regresión y detección de la desviación.

Nos centraremos en el método de la clasificación ya que es el método que aplicamos en el proyecto. La tarea del clasificador es encontrar un modelo que, aplicado a un nuevo ejemplo sin clasificar, lo clasifique dentro de un conjunto predefinido de clases.

En la actualidad, la minería de datos es aplicada en diferentes campos, desde el terrorismo hasta el análisis de la cesta de la compra. Por ello, destacaremos algunos de los ejemplos en los que se ha utilizado:

- Fraudes: detección de transacciones de lavado de dinero o de fraude en el uso de tarjetas de crédito o de servicios de telefonía móvil e, incluso, en la relación de los contribuyentes con el fisco.
- Recursos humanos: la minería de datos puede ser útil en este campo para la identificación de las características de sus empleados de mayor éxito.

- Terrorismo: mediante uso de la minería de datos en EE.UU había identificado al líder de los atentados del 11 de septiembre de 2001, Mohammed Atta, y a otros tres secuestradores del "11-S" como posibles miembros de una célula de Al Qaeda que operan en los EE.UU, más de un año antes del ataque.
- Análisis de la cesta de la compra: se ha utilizado para la detección de hábitos de compra en supermercados.

## 2.3. INTRODUCCIÓN AL ANÁLISIS DEL TRÁFICO

### 2.3.1. Flujo del tráfico.

Hoy en día, el tránsito vehicular es la consecuencia de múltiples factores sociales, culturales, económicos y políticos que se presentan en las principales ciudades del mundo ya que el número de habitantes en la población está aumentando y esto conlleva a un mayor número de coches.

El tránsito vehicular (también llamado tráfico vehicular, o simplemente tráfico) se define como el fenómeno causado por el flujo de vehículos en una vía, calle o autopista.

En nuestro proyecto predeciremos el flujo del tráfico para las diferentes celdas en la que hemos dividido el mapa de Oporto. Estas celdas podrán contener más de una calle, vía o autopista por tanto predeciremos el tránsito vehicular para un conjunto de calles, vías o autopistas. El flujo del tráfico lo hemos categorizado en tres tipos: libre, sincronizado y congestionado.

### 2.3.2. Congestión.

La congestión vehicular se define como un exceso de vehículos en una vía, la cual trae como consecuencia que cada vehículo avance de forma lenta e irregular en comparación a las condiciones normales.

Las causas más recurrentes por las que se produce la congestión vehicular son la mala gestión de las señales, la insuficiente capacidad vial y la descontrolada demanda de carreteras. Como resultado, se dan situaciones en las que se necesita más espacio debido al gran número de vehículos que sobrepasa la capacidad real o posible de la vía. Además existen otras causas secundarias que provocan la congestión como por ejemplo los fenómenos climáticos, obras viales y accidentes de tráfico.

Además de ser uno de los principales contaminantes atmosféricos, existen una serie de desventajas que provocan la congestión vehicular:

- Desgastes de ciertos componentes de unos vehículos debido a los recurrentes frenados y aceleraciones.
- Mayor cantidad de combustible necesitada para realizar un trayecto.
- Pérdida de tiempo por parte de los conductores y pasajeros. Esto puede provocar que lleguemos al trabajo, a la universidad, etc... tarde lo cual en ciertas intentamos evitar saliendo antes y dedicando menos horas de sueño a nuestro cuerpo. Esta decisión puede repercutir en nuestro rendimiento, así como en nuestra salud física y mental.

# 3. CAPÍTULO III: DESCRIPCIÓN DE LOS DATOS

En esta parte del proyecto se presentan las distintas tareas realizadas con el objetivo de familiarizarse con los datos. De esta manera comprenderemos el problema y tendremos los conocimientos suficientes para afrontar con garantías la siguiente fase.

## 3.1. RECOLECCIÓN INICIAL DE LOS DATOS

El primer paso fue encontrar un dataset que nos permitiera predecir el flujo de tráfico de alguna ciudad. Gracias a Luis Moreira Matías conseguimos un conjunto de datos de entrenamiento de la ciudad de Oporto.

Los datos se obtuvieron de 442 Taxis en la ciudad de Oporto, Portugal, los cuales fueron equipados de manera que pudo obtenerse cada 15 segundos la posición GPS de los mismos, así como un conjunto de otras variables. En total se capturaron secuencias de posiciones del 1 de Julio de 2013 al 30 de Junio de 2014, cada vez que un pasajero contrataba un viaje en uno de esos 442 taxis.

## 3.2. DESCRIPCIÓN DE LOS DATOS INICIALES

Nuestra base de datos está compuesta por nueve campos, 1710671 instancias y tiene un tamaño de 1.8GB aproximadamente. A continuación explicaremos lo que significa cada campo de la base de datos:

- *TRIP_ID*: Identificador único para cada trayecto de un taxi.
- *CALL_TYPE*: Identifica la manera que se usó para contratar el servicio del taxi. Existen tres posibles valores.
  - ‘A’. Este viaje fue contratado desde la central.
  - ‘B’. Este viaje fue contratado en una parada de taxis.
  - ‘C’. De otro modo como por ejemplo parar un taxi en una calle.
- *ORIGIN_CALL*: Identificador único para cada número de teléfono que se utilizó para contratar al menos, un servicio de taxis. Se identifica al cliente de viaje si *CALL_TYPE* = 'A'. De lo contrario, se toma el valor *NULL*.

- ORIGIN_STAND: Identificador único para cada parada de taxis. Identifica el punto de partida del viaje si el tipo de llamada es de tipo 'B'. De lo contrario, se toma el valor NULL.
- TAXI _ID: Identificador único para cada taxista.
- *TIMESTAMP*: Identifica el comienzo del viaje mediante una marca de tiempo en Unix.
- *DAYTYPE*: Identifica el tipo de día en el que se inició el viaje. Existen tres posibles valores.
  - 'B'. Este viaje comenzó en un día de fiesta.
  - 'C'. Este viaje comenzó un día antes de un día tipo B.
  - 'A'. Este viaje comenzó un día normal como por ejemplo un día laborable o fin de semana.
- *MISSING_DATA*: Este campo es falso cuando el flujo de datos del GPS está completo mientras que es verdadero si una o más localizaciones están perdidas.
- *POLYLINE*: Contiene una lista de coordenadas del GPS en formato WGS84 mapeadas como una cadena. El principio y final de la cadena se identifica con un abre corchetes y cierra corchetes respectivamente. Cada par de coordenadas (longitud, latitud) se identifican también con abre corchetes y cierra corchetes. Esta cadena contiene un contiene un conjunto de par de coordenadas las cuales representan la posición del taxi cada 15 segundos del viaje.

# 4. CAPÍTULO IV: PREPARACIÓN DE LOS DATOS

En este capítulo se describirán todas las tareas que se realizaron hasta llegar a conseguir el fichero al cual posteriormente le aplicamos minería de datos. Varias de las actividades que describiremos tuvieron que ser realizadas más de una vez, por tanto, se cumplió la propiedad iterativa del proceso KDD.

## 4.1. SELECCIÓN DE LOS DATOS

En este apartado se describe cuáles fueron los datos que se seleccionaron. Para ello se aplicó un filtrado a nivel de atributos con el objetivo de eliminar aquellos atributos que no nos van aportar nada en el futuro. Se eliminaron los siguientes atributos: *TRIP_ID*, *CALL_TYPE*, *ORIGIN_CALL*, *ORIGIN_STAND*, *DAYTYPE*. El resto de atributos se mantendrán ya que contienen información valiosa. Tras seleccionar los datos útiles hubo que limpiarlos y depurarlos.

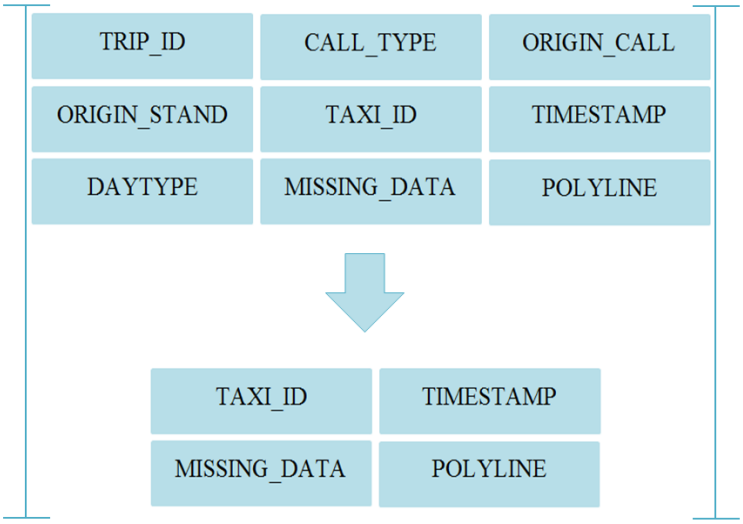



Figura 2: Selección de atributos.

## 4.2. LIMPIEZA DE LOS DATOS

Se aplica la limpieza de los datos con la función de conseguir aumentar la calidad de los datos. Las tareas de limpieza de datos van, normalmente encaminadas a resolver dos

problemas habituales: la ausencia de valores en algunos registros y la existencia de datos erróneos.

En la base de datos aplicamos la limpieza por ausencia de valores en el campo *POLYLINE* cuando solo contiene dos corchetes, pues es como si el taxi no hubiera hecho un trayecto y en el caso de que si lo hiciera fue en menos de 15 segundos. Estas instancias fueron eliminadas ya que no se consideraban útiles. Además, se eliminaron aquellas instancias que tenían el campo "*POLYLINE*" incompleto debido a que consideramos este trayecto como erróneo. Para saber si dicho campo estaba incompleto nos fijamos en el valor booleano que tenía el campo "MISSING_*DATA*", ya que si el valor es verdadero significa que la cadena del GPS está incompleta.

Una vez aplicada la selección y limpieza de datos obtuvimos un fichero de tamaño 1.73GB aproximadamente. La aplicación de la selección y limpieza de datos a la base de datos se puede ver en el proyecto en el método *dbModificate* de la clase *DBModification*.

## 4.3. TRANSFORMACIÓN DE LOS DATOS

En este apartado veremos cuáles son las tareas que se realizan con el objetivo de transformar los datos y así obtener un fichero al cual se le puede aplicar minería de datos.

La primera tarea que se realiza es calcular la latitud y longitud mínima y máxima con el objetivo de acotar el problema a una zona concreta de Oporto. Estos valores que mostraremos a continuación los conseguimos ejecutando el método calculateLatAndLonMinMax de la clase DataPreprocessing.

| Latitud mínima | Latitud máxima | Longitud mínima | Longitud máxima |
|---|---|---|---|
| 36.886 | 44.352 | -15.630 | -3.930 |

Tabla 1: Latitud y longitud mínima y máxima

Haciendo uso de una página que nos permitía representar un mapa acotado por estas longitudes y latitudes nos dimos cuenta que estos valores no eran los más correctos ya que nos salíamos del mapa de Oporto. Por ello haciendo uso del mapa decidimos acotar la zona que realmente queríamos.


Figura 3: Mapa de Oporto.

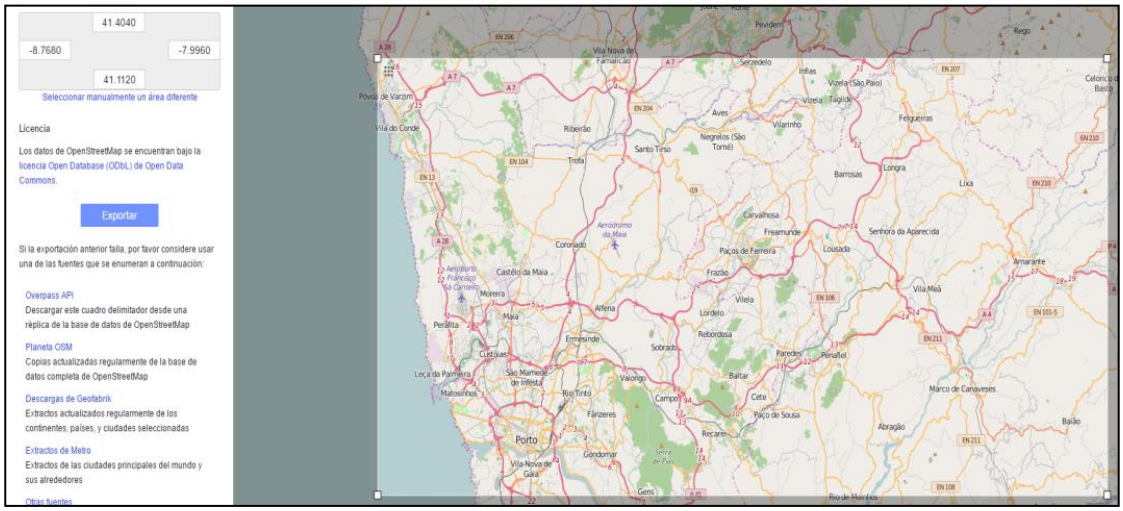



Figura 4: Mapa acotado por una longitud y latitud mínima y máxima.

Por tanto la longitud y latitud mínima y máxima que usaremos son los siguientes:

| Latitud mínima | Latitud máxima | Longitud mínima | Longitud máxima |
|---|---|---|---|
| 41.112 | 41.404 | -8.768 | -7.996 |

Tabla 2: Latitud y longitud mínimas y máximas que usaremos.

Tras tener los valores de longitud y latitud ajustados, nuestra siguiente tarea es construir un conjunto de celdas con el objetivo de representar el flujo del tráfico de Oporto. Para ello hacemos uso de la librería *OpenCSV* puesto que cada celda será un fichero csv y esta librería nos permite leer y escribir en ficheros con extensión csv de manera sencilla.

Cada uno de las celdas representará una parte del mapa de Oporto. Lo primero que tuvimos que calcular fueron las diferencias entre la longitud máxima y mínima y la latitud máxima y mínima ya que así sabremos la distancia que tenemos que repartir. Teniendo estos dos valores podremos saber cuántas celdas necesitamos ya que solo nos hace falta dividirla por el tamaño que tendrá cada celda y así obtendremos celdas uniformes que mediante índices representan un conjunto de valores de latitudes y longitudes. Mediante la siguiente tabla se entenderá mejor.

| Tamaño de Celda | DiferenciaLat | DiferenciaLong | ÍndiceMaxLat | ÍndiceMinLat | Número de celdas |
|---|---|---|---|---|---|
| 0,004 | 0,292 | 0,772 | 73 | 193 | 73*193 = 14089 |

Tabla 3: Representación de la creación de celdas.

De esta manera obtenemos una serie de índices los cuales utilizamos para nombrar los ficheros ya que así será más sencillo enviar cada par de coordenadas de GPS al fichero que le toque. La creación de los ficheros que representan al conjunto de celdas se realiza con el método *DataPreprocessing.CreateCell*.

Sabiendo el número de celdas que nos hacen falta para representar el mapa de Oporto, nos disponemos a explicar cuáles son los atributos que contiene en su interior.

- *Date*: indica el día y la hora en la que almacenó una coordenada de GPS. Para representar el día usamos número sucesivos del 0 al 6 donde el 0 representa el domingo y el 6 el sábado. En cuanto a la hora será en formato digital de 00:00h a las 23:59h.
- *Latitude*: representa la latitud de una coordenada de GPS.
- *Longitude*: representa la longitud de una coordenada de GPS.
- *DPreviousPosition*: distancia euclidea respecto a la posición anterior en la que se encontraba el taxi en su trayecto.
- ID_TAXIS: indica el identificador del taxi.

Al tener creados los ficheros que representan cada una de las celdas en las que vamos a dividir el mapa de Oporto, nos disponemos a introducir cada par de coordenadas de GPS a la celda que le corresponde. Esta acción la realizamos cogiendo cada par de coordenadas GPS que nos encontramos en la String del atributo *POLYLINE* y mediante

el uso de las siguientes fórmulas decidimos a qué fichero, el cual representa una celda, debe ir.

$$IndexLatitud = LatitudGPS - LatitudMinima / TamañoDeCelda$$

$$IndexLongitud = LongitudGPS - LongitudMinima/TamañoDeCelda$$

En cuanto al campo *Date*, lo conseguimos haciendo una conversión del campo *TIMESTAMP*. Además tenemos en cuenta que para calcular este campo, cada posición GPS que se encuentra en el atributo *POLYLINE* es almacenada por el paso de 15 segundos. Por ello la primera posición GPS hace uso del valor concreto del *TIMESTAMP*, pero para cada uno de los siguientes habría que sumarle 15 · i donde i es la posición de un par de coordenadas GPS en el campo *POLYLINE*. Finalmente nos quedaría el campo ID_TAXI, el cual es una copia del atributo TAXI_ID de la base de datos. Este proceso se repite para cada una de las filas de dicha base de datos y lo podemos ver en el proyecto con el método calculateAtributteLatitudeLongitudeTime de la clase DataPreprocessing.

```
"Date","Longitude","Latitude","DPreviousPosition","ID_TAXI"
"1 1:00","-8.681436","41.15664","0.002","20000060"
"1 1:00","-8.682336","41.157711","0.001","20000060"
"1 1:00","-8.683794","41.159457","0.002","20000060"
"1 2:00","-8.683938","41.159601","0.003","20000657"
"1 2:00","-8.683668","41.159196","0.0","20000657"
"1 2:00","-8.683371","41.158836","0.0","20000657"
"1 2:00","-8.681751","41.15691","0.003","20000657"
"1 3:00","-8.681247","41.15637","0.002","20000367"
```

Figura 5: Ejemplo de un fichero que representa una celda.

Nuestro siguiente paso tras tener todas las instancias de la base datos ya procesadas es conseguir un fichero único con la información obtenida en cada una de las celdas. Por ello tendremos por cada celda 24·7 instancias en el fichero final, 24 instancias por día. Además añadiremos una serie de atributos y modificaremos los atributos *longitude* y *latitude* ya que nos interesa tener el índice que representa a ese conjunto de valores. Estos son los atributos que tendremos en este último fichero en el cual aplicaremos minería de datos posteriormente:

- *Date*: indica el día de la semana y una hora mediante un valor numérico. El 0 representa el domingo a las 00:00h, el 1 el domingo a las 01:00h y así

sucesivamente hasta completar todos las horas de todos los días de la semana. Por ello el máximo valor que tendremos en este atributo es el 167 que representa el sábado a las 23:00h.

- *Latitude*: índice de la latitud que representa esa celda.
- *Longitude*: índice de la longitud que representa esa celda.
- *MDPreviousPosition*: distancia media respecto a la posiciones anteriores de los taxis a un día y una hora concreta, que se encuentra en una celda.
- *MediumDistance*: distancia euclidea media de un conjunto de taxis a un día y una hora concreta, que se encuentra en una celda.
- *InflowCar*: flujo de entrada de taxis en una celda a un día y una hora concreta.
- *OutflowCar*: flujo de salida de taxis en una celda a un día y una hora concreta.
- *TotalInstancesThisHour*: número de instancias que tenemos en una celda para un día y una hora concreta.
- *Class*: Clase con la que se va etiquetar esta instancia. Puede ser congestionado, libre o sincronizado.

Estos nuevos atributos que tenemos en el fichero final los conseguimos de la siguiente forma. Para calcular cada uno de ellos nos creamos un *array* de dimensión 7·24 donde las filas se corresponden con los días de la semana y las columnas con las horas del día. Tras ello vamos leyendo las filas que componen una celda y realizamos las siguientes acciones dependiendo del atributo que queramos conseguir:

- *MDPreviousPosition*: almacenamos el valor que se encuentra en el atributo denominado *DPreviousPosition* en la posición del *array* que le corresponda dependiendo del día y la hora.
- *MediumDistance*: almacenamos la latitud y longitud de cada fila en la posición del *array* que le corresponda. Destacar que cada posición del *array* almacena en su interior dos listas, una para almacenar los valores de longitud y otra para almacenar los valores de latitud.
- *InflowCar* y *OutflowCar*: almacenamos el identificador del taxi en la posición del *array* que le corresponda. Cada posición del *array* está compuesta por una lista ya que necesitamos todos los identificadores de los taxis que estuvieron en un día a una hora concreta.

- *TotalInstancesThisHour*: aumentamos en uno el valor que se encuentra en la posición del *array* que le corresponda.

Cuando ya tenemos almacenado toda la información de una celda en los *array*, como hemos explicado anteriormente, realizamos un último paso para calcular estos atributos. En el caso del atributo *MDPreviousPosition* para obtener el valor de un día y una hora concreta dividimos el valor almacenado en la posición del *array* por el número de instancias. En el atributo *MediumDistance* tenemos que aplicar el cálculo de la distancia media euclidea para el conjunto de coordenadas almacenadas en cada posición del *array*. Finalmente nos faltaría el cálculo de flujo de entrada (*InflowCar*) y el flujo de salida (*OutflowCar*) de vehículos a una hora concreta. Al tener almacenado los identificadores de los taxis para cada hora de cada día de la semana, lo que realizamos para calcular el flujo de entrada de taxis para una hora concreta es el cálculo de la diferencia entre los identificadores de taxis de la hora actual y la hora anterior mientras que para el flujo de salida de taxis sería la diferencia entre los identificadores de la hora actual y la hora posterior.

Llegados a este punto, solo nos faltaría describir la obtención del atributo *class*, el cual consideramos el atributo más importante. Para calcular este atributo lo primero que realizamos es obtener un parámetro que denominamos "N crítica" cuya función es indicarnos cual es el mayor flujo de salida de vehículos de dicha celda. De esta forma sabremos cual es el número de taxis que soporta una celda. Tras ello etiquetamos las 7·24 instancias de esta celda, haciendo uso de "N critica" y de "N", donde N es el número de taxis que pasaron por esa celda en ese día y hora concreta. Al querer predecir el estado futuro del flujo de tráfico lo que hacemos es etiquetar la instancia actual con la etiqueta de la hora siguiente, por tanto, estaríamos utizando la N de la hora posterior. Este etiquetado lo hacemos categorizando en tres umbrales la "N crítica":

- 0-25% : libre.
- 75-90% : sincronizado.
- > 90%: congestionado.

Por ello si N es menor o igual que el 25% del valor de N crítica se etiqueta como libre, si está entre el 75-90% del valor N crítica se etiqueta como sincronizado y si es mayor del 90% del valor de N crítica se etiqueta como congestionado.

Programáticamente la N crítica la hemos calculado tomando como máximo 4 muestras de N, en la que se obtenían el mayor flujo de salida de taxis. Posteriormente hacemos una media de ese conjunto de valores. El cálculo de N crítica lo podemos ver en el proyecto en el método calculateCriticN de la clase CalculateFlushCar.

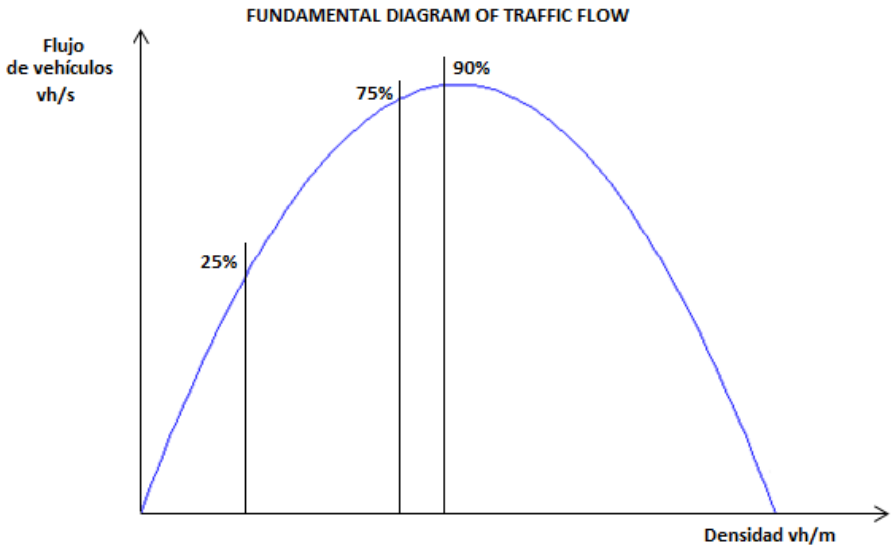



Figura 6: Diagrama fundamental del flujo del tráfico.

En el proyecto se aplica la gráfica anterior para realizar el etiquetado de la clase, el cual se ha explicado anteriormente. En este caso la densidad vehicular es la N crítica calculada para cada celda, mientras que el flujo de vehículos es el número de taxis que han estado dicha celda en un día a una hora concreta.

Cabe destacar que la N crítica que obtenemos es respecto al conjunto de taxis que pasaron por esa celda y por tanto el etiquetado que estamos llevando a cabo es del flujo del tráfico de los taxis y no el del flujo del tráfico general. Esto se debe tener en cuenta ya que podría haber una hora en la que haya mucho más tráfico de particulares que de taxis. Por ello nosotros asumimos que el flujo del tráfico general sigue un patrón similar al flujo del tráfico de los taxis. Esta simplificación se ha realizado debido a que solo teníamos un dataset con trayectorias de taxis y no de vehículos de habitantes de Oporto.

Este proceso donde se calculan los atributos explicados anteriormente lo podemos ver en el proyecto en el método *calculateAttributesEndFile* de la clase *DataPreprocessing*. También es importante comprender que los cálculos de estos atributos se realizan para una única celda y que cuando se finaliza la lectura de dicha celda se escribe en el fichero "final" 7·24 instancias con los correspondientes valores de

cada atributo. Por ello, para comenzar con la lectura de los datos almacenados en otra celda, se reinician los objetos donde está guardada la información de los atributos de la celda anterior.

```
0,0,0,0.001,0,49,39,226,Synchronized
1,0,0,0.001,0,69,55,320,Congested
2,0,0,0.001,0.001,75,51,394,Congested
3,0,0,0.001,0,64,50,474,Congested
4,0,0,0.002,0,61,56,466,Congested
5,0,0,0.002,0.001,58,63,438,Congested
6,0,0,0.002,0,58,77,381,Congested
7,0,0,0.002,0.001,41,68,324,Synchronized
```

Figura 7: Muestras del fichero final.

### 4.3.1. Modificación de la transformación de datos.

El dataset que obtenemos, al cual le vamos aplicar minería de datos, contiene un 88% de instancias con ausencia de valor del atributo *class*. Esto se debe a que no hay muestras de taxis a ciertas horas en algunas de las celdas. Estas instancias en un futuro se etiquetarán como libre ya que no hay muestras de trayecto de taxis en esas zonas. Esto significaría que tendríamos más del 88% del dataset etiquetado como libre lo cual no nos conviene, puesto que no tendríamos clases balanceadas. Además, tener 2366952 instancias nos hace muy difícil obtener un clasificador idóneo ya que la única manera de clasificar es hacer uso de algoritmos incrementales o de *Massive Online Analysis*, Análisis masivo online (MOA).


Figura 8: Número de instancias con valor perdido.

Por estas razones hemos decidido acotar las dimensiones de latitud y longitud mínimas y máximas con el objetivo de obtener clases más balanceadas y estar en la zona de Oporto donde se encuentran la mayoría de las muestras de la base de datos.

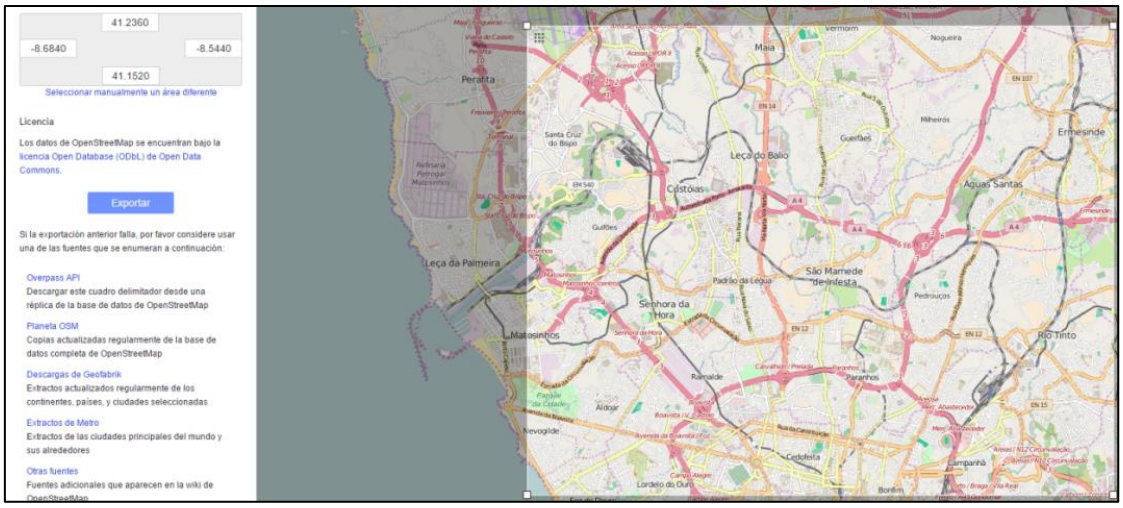



Figura 9: Nuevo mapa acotado por una longitud y latitud mínima y máxima.

Por tanto la longitud y latitud mínima y máxima que usaremos en este nuevo preprocesado de datos son los siguientes:

| Latitud mínima | Latitud máxima | Longitud mínima | Longitud máxima |
|---|---|---|---|
| 41.152 | 41.236 | -8.684 | -8.544 |

Tabla 4: Modificación de la latitud y longitud mínima y máxima.

Al realizar las modificaciones en longitud y latitud mínima y máxima obtenemos un menor número de celdas que se verán reflejados en la siguiente tabla.

| Tamaño de Celda | DiferenciaLat | DiferenciaLong | ÍndiceMaxLat | ÍndiceMinLat | Número de celdas |
|---|---|---|---|---|---|
| 0,004 | 0,084 | 0,140 | 22 | 36 | 22*36 = 792 |

Tabla 5: Representación de la creación de celdas.

Tras haber realizado la etapa de transformación de datos con las modificaciones hechas obtenemos un fichero donde el 12% de las instancias tienen valor perdido lo cual es mucho menor del 88% obtenido en el fichero anterior. Además el fichero contiene 133056 instancias, un número mucho más asequible para los diferentes tipos de clasificadores que contiene Weka. Finalmente indicar que el tiempo de ejecución del preprocesado de datos disminuye considerablemente al realizar estas modificaciones en

la transformación de datos ya que anteriormente tardaba más de un día y ahora tarda 7 horas aproximadamente.

## 4.4. FORMATO DE LOS DATOS

Como se especificó anteriormente, el software que se utilizará para realizar la minería de datos es Weka. Tras la transformación de datos, nosotros tenemos un fichero que tiene extensión csv. Este tipo de fichero los acepta Weka pero decidimos pasarlo a formato ARFF ya que es con lo que se suele trabajar en Weka. Además el tamaño del fichero disminuye considerablemente respecto al fichero que obtenemos con extensión csv.

La estructura de un fichero con formato ARFF es muy sencilla. Los ficheros con formato ARFF se dividen en tres partes: @relation, @attribute y @data.

- @relation <nombre>: Todo fichero ARFF debe comenzar con esta declaración en su primera línea (no se pueden dejar líneas en blanco al principio). <nombre> será una cadena de caracteres y si contiene espacios se pondrá entre comillas.
- @attribute <nombre> <tipo_de_datos>: En esta sección se incluye una línea por cada atributo (o columna) que se vaya a incluir en el conjunto de datos, indicando su nombre y el tipo de dato. Con <nombre> se indicará el nombre del atributo, que debe comenzar por una letra y si contiene espacios tendrá que estar entrecomillado. Con <tipo_de_datos> se indicará el tipo de dato para este atributo (o columna) que puede ser:
    - numeric que representa que el atributo es de tipo numérico.
    - integer indica que este atributo contiene números enteros.
    - string indica que este atributo representa cadenas de texto
    - date [<formato-fecha>]. En <formato-fecha> indicaremos el formato de la fecha, que será del tipo "yyyy-MM-dd'T'HH:mm:ss".
    - <nominal-specification>. Estos son tipo de datos son definidos por el usuario y pueden tomar los valores indicados por él. Cada una de los valores debe ir separados por coma.
- @data: En este apartado se incluyen los datos propiamente dichos. Cada columna es separada por comas y todas las filas deben tener el mismo número de columnas, número que coincide con el de declaraciones @attribute, añadidas en la sección anterior. Si no se dispone de algún dato, se escribe un signo de

interrogación (?) en su lugar, ya que dicho carácter representa la ausencia de valor. El separador de decimales tiene que ser obligatoriamente el punto y las cadenas de tipo string tienen que estar entre comillas simples.

```
@relation DBTrain

@attribute Date numeric
@attribute Latitude numeric
@attribute Longitude numeric
@attribute MDPreviousPosition numeric
@attribute MediumDistance numeric
@attribute InflowCar numeric
@attribute OutflowCar numeric
@attribute TotalInstancesThisHour numeric
@attribute Class {Synchronized,Congested,Free}

@data
0,0,0,0.001,0,49,39,226,Synchronized
1,0,0,0.001,0,69,55,320,Congested
2,0,0,0.001,0.001,75,51,394,Congested
3,0,0,0.001,0,64,50,474,Congested
4,0,0,0.002,0,61,56,466,Congested
5,0,0,0.002,0.001,58,63,438,Congested
6,0,0,0.002,0,58,77,381,Congested
7,0,0,0.002,0.001,41,68,324,Synchronized
```

Figura 10: Fichero al que se va aplicar minería de datos en formato ARFF.

# 5. CAPÍTULO V: MINERÍA DE DATOS Y EVALUACIÓN

En este capítulo, se realizará una breve introducción a Weka con la finalidad de comprender este programa. Posteriormente se describirán los distintos algoritmos de clasificación seleccionados ya que con ellos se realizarán una serie de experimentos con el objetivo de obtener un modelo que nos genera una predicción del flujo del tráfico.

## 5.1. INTRODUCCIÓN A WEKA

Weka es una plataforma de software para el aprendizaje automático y la minería de datos escrito en Java y desarrollado en la Universidad de Waikato.

Podemos hacer uso de Weka mediante una interfaz gráfica o utilizando la librería en un proyecto Java.

Centrándonos en la interfaz gráfica de Weka, nos podemos encontrar con cuatro formas de acceso a las diferentes funcionalidades de la aplicación, las cuales describiremos a continuación:

- *Simple CLI* (Simple command-line interface): es una consola que nos permite acceder a todas las opciones de Weka.
- *Explorer*: permite visualizar y aplicar distintos algoritmos de aprendizaje a un dataset. Cada una de las tareas de minería de datos viene representada por una pestaña en la parte superior. Son las siguientes :
    - *Preprocess*: visualización y preprocesado de los datos. En esta pestaña tendremos una serie de filtros que podremos aplicar al conjunto de datos.
    - *Classify*: Algoritmos de clasificación y regresión.
    - *Cluster*: Algoritmos de agrupamiento.
    - *Associate*: Algoritmos para obtener reglas de asociación.
    - *Select Attributes*: Selección de atributos.
    - *visualize*: Nos permite visualizar pares de atributos.

- *Experimenter*: sirve para aplicar varios algoritmos de minería de datos sobre distintos conjuntos de datos.
- *Knowledge* Flow: nos permite realizar lo mismo que el *explorer* pero de manera gráfica.


Figura 11: Interfaz Weka.

De estas cuatro funcionalidades haremos uso de la denominada *Explorer*, en concreto, de la pestaña *Classify*. Además explicaremos un popurrí de características que podremos encontrar al hacer uso de cualquiera de los algoritmos que contiene weka.

### 5.1.1. Métodos de evaluación.

Existen cuatros formas de evaluar los diferentes algoritmos de minería de datos utilizados:

- *Use training set*: esta opción de Weka nos permite entrenar el método con todos los datos disponibles y a posteriori realizar la evaluación sobre los mismos datos.
- *Supplied test set*: esta opción de Weka nos permite cargar un conjunto de datos, normalmente diferentes a los de aprendizaje, con los cuales se realizará la evaluación.
- *Cross-validation*: evaluación con validación cruzada. Se dividirán las instancias en tantas carpetas como indica el parámetro *Folds*, en cada evaluación se tomarán las instancias de cada carpeta como datos de test y el resto como datos de entrenamiento para construir el modelo. Los errores calculados serán el promedio de todas las ejecuciones.

- *Percentage split*: se define un porcentaje con el que se aprende el modelo. La evaluación se realiza con los datos restantes.

### 5.1.2. Resumen del modelo obtenido.

Tras haber construido un modelo haciendo uso de cualquiera de los algoritmos que contiene Weka, obtenemos un resumen de dicho modelo.

```
=== Summary ===

Correctly Classified Instances      118474               89.0407 %
Incorrectly Classified Instances     14582               10.9593 %
Kappa statistic                          0.8058
Mean absolute error                      0.1144
Root mean squared error                  0.2391
Relative absolute error                 29.9679 %
Root relative squared error             54.7357 %
Total Number of Instances           133056
```

Figura 12: Resumen de un clasificador.

Como podemos observar en la ilustración, obtenemos una serie de datos que significan lo siguiente:

- Número de instancias clasificadas correctamente.
- Numero de instancias clasificadas incorrectamente.
- El estadístico Kappa es un índice que compara el nivel de coincidencia entre varios expertos con el nivel de coincidencia que se podría dar por casualidad.
- Error cuadrático medio: es un estimador que mide el promedio de los errores al cuadrado, es decir, la diferencia entre el estimador y lo que se estima al cuadrado.
- Error absoluto: La diferencia entre el valor obtenido y su valor real.
- Error absoluto relativo: La fórmula [(xi - xt)/xt] x100, donde xi – xt es el error absoluto y xt es el valor real.
- Número total de instancias.

### 5.1.3. Resumen del porcentaje de aciertos por clase.

Al igual que obtenemos un resumen completo del clasificador, también conseguimos un resumen detallado del porcentaje de aciertos por clase.

```
=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.923     0.148      0.885     0.923     0.904      0.949    Synchronized
                 0.788     0.008      0.934     0.788     0.854      0.972    Congested
                 0.871     0.056      0.885     0.871     0.878      0.972    Free
Weighted Avg.    0.89      0.101      0.891     0.89      0.889      0.96
```

Figura 13: Resumen detallado del porcentaje de aciertos por clase.

En la imagen anterior podemos observar los diferentes parámetros que se calculan. Por ello vamos a explicar que significan cada uno de ellos:

- TP significa *true positive* y nos refleja la cantidad media de muestras que el modelo clasificó correctamente para esa clase.
- FP significa *false positive* y nos refleja la cantidad media de muestras que el modelo clasificó de esta clase cuando en realidad no lo eran.
- *Precision* se utiliza para indicar la fracción de ejemplares que se han clasificado como de la clase correspondiente y que, en realidad, son de esa clase.

$$Precision = TP / (TP + FP)$$

- *Recall* significa sensibilidad y se utiliza para indicar la fracción de ejemplos de la clase de todo el conjunto que se clasifican correctamente.

Finalmente tenemos una fila que nos refleja un promedio de cada uno de los parámetros donde cada clase tiene un peso concreto dependiendo del número de instancias que la componen.

### 5.1.4. Matriz de confusión.

La matriz de confusión, o también tabla de contingencia, está formada por tantas filas y columnas como clases hay. Para cada fila tenemos: el número de instancias que se ha clasificado con esa clase de forma correcta y el número de instancias que se ha clasificado para cada de una de las otras clases, es decir, instancias clasificadas incorrectamente, ya que se deberían haber clasificado con la clase que está en la diagonal de dicha fila. Por ello el número de instancias clasificadas correctamente es la suma de la diagonal de la matriz mientras que el resto están clasificadas de forma incorrecta.

Esta matriz es muy útil para saber dónde está fallando fundamentalmente el clasificador ya que podemos saber cuál es la clase con la que más está fallando la predicción y con qué otra clase lo está prediciendo.

```
    a     b     c   <-- classified as
67829   851  4814 |     a = Synchronized
 3184 12404   162 |     b = Congested
 5622    29 38161 |     c = Free
```

Figura 14: Ejemplo de matriz de confusión.

### 5.1.5. Curva ROC.

La curva *Receiver Operating Curve*, Característica Operativa del Receptor (ROC) se forma graficando el valor de TP contra el FP, por lo que es un punto en el espacio que muestra el desempeño de un algoritmo de clasificación. Esta curva es muy útil, pues nos ofrece una representación visual entre los beneficios reflejados por los TP y los costos reflejados por los FP de la clasificación.

El clasificador puede ser de dos tipos: *hard-type* y *soft-type*. El primer tipo es aquel en el que la predicción indica solamente la clase a la que pertenece una instancia, no su grado de pertenencia a dicha clase, por tanto, producirá un punto (TP, FP) que corresponde a una coordenada en el espacio ROC. El punto (1,0) corresponde a una clasificación perfecta ya que supondría beneficios máximos a un coste cero, por tanto un clasificador será mejor que otro mientras más cerca se encuentre su punto del (1,0).

El segundo tipo, *soft-type*, produce un valor numérico continuo para representar la confianza de una instancia que pertenece a la clase predicha. Para este tipo se suele utilizar un umbral que produce una serie de puntos en el espacio ROC, por lo que se genera una curva en lugar de tener un único punto en el plano. En este tipo, el clasificador que tenga una mayor área bajo su curva, será el mejor.

Todos los algoritmos de clasificación que utilizaremos son de tipo *soft-type*, por tanto, siempre obtendremos una curva para representar el ROC de este clasificador. Cada vez que utilicemos un clasificador obtendremos un resumen de los porcentajes de aciertos por clase. En dicho resumen se encuentra un parámetro denominado "*ROC Area*", el cual nos indica el área que se encuentra bajo la curva ROC, por lo tanto, este parámetro nos ayudará a saber si un clasificador es mejor que otro.

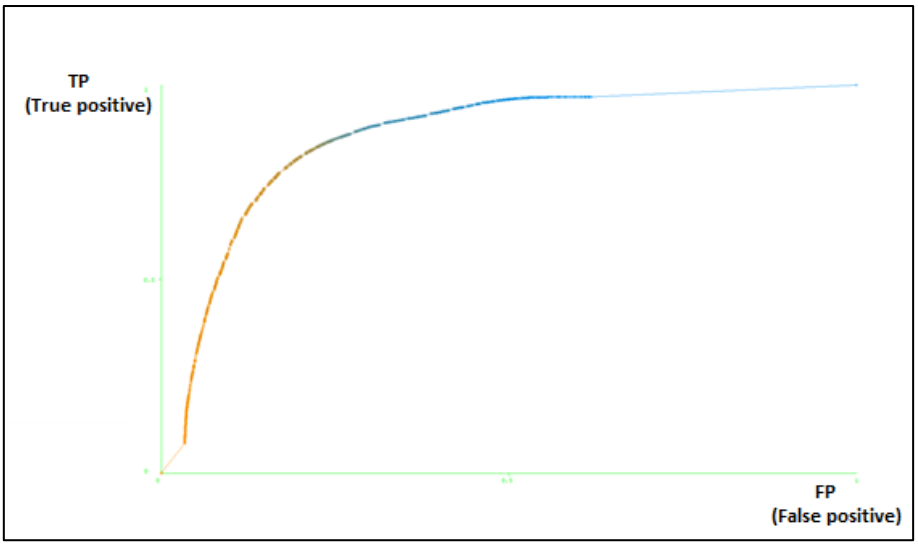



Figura 15: Ejemplo de curva ROC para una clase en Weka.

## 5.2. ALGORITMOS DE CLASIFICACIÓN EN WEKA

Como comentamos en capítulos anteriores, solo haremos uso de algoritmos de clasificación ya que estos nos construyen modelos con el objetivo de clasificar nuevas instancias dentro de un conjunto predefinido de clases. Además nuestra clase es de tipo cualitativo ya que tenemos categorías: libre, sincronizado y congestionado. Esta característica de la clase nos lleva a hacer uso de algoritmos de clasificación.

Existe un conjunto de algoritmos clasificadores en Weka pero solo haremos uso de estos siete:

- *OneR*. Es un clasificador de los más sencillos y rápidos. Sus resultados pueden ser bastantes buenos en comparación con algoritmos mucho más complejos. La idea es hacer reglas que prueban un solo par atributo-valor, probar todos los pares atributo-valor y seleccionar el atributo que ocasione el menor número de errores, por tanto, sólo se utiliza un atributo para predecir la salida.
- *C4.5 (J48).* Este es uno de los algoritmos de minería de datos más utilizado. Su funcionamiento se basa en generar un árbol de decisión a partir de los datos mediante particiones realizadas recursivamente, según la estrategia de profundidad-primero (*depth-first*). Antes de cada partición de datos, el algoritmo considera todas las pruebas posibles que pueden dividir el conjunto de datos y selecciona la prueba que produce la mayor ganancia de información. Para cada atributo discreto, se considera una prueba con "n" resultados, siendo "n" el número de valores posibles que puede tomar el atributo. Para cada

atributo continuo se realiza una prueba binaria sobre cada uno de los valores que toma el atributo en los datos.

- *PART*. Este algoritmo forma reglas a partir de árboles de decisión parcialmente podados, los cuales son construidos usando los heurísticos de *C4.5*. Por ello, las opciones disponibles para este algoritmo son un subconjunto de las disponibles para *C4.5*. Este algoritmo evita la generalización precipitada.
- *Hoeffding Tree*. El clasificador pertenece a los algoritmos denominados incrementales que procesan los ejemplos uno a uno, de tal manera que cada nuevo ejemplo obliga a modificar las reglas obtenidas para que clasifiquen correctamente el nuevo ejemplo. Este tipo de árbol aprovecha la idea de que cualquier árbol de decisión es capaz de aprender un conjunto masivo de datos, asumiendo que la distribución generadora de ejemplos no cambia en el tiempo. Además, este algoritmo explota el hecho de que una pequeña muestra a menudo puede ser suficiente para elegir un óptimo atributo. Esta idea es apoyada matemáticamente por *Hoeffding*, ya que según él es necesario cuantificar el número de observaciones para estimar algunas estadísticas. Más precisamente, *Hoeffding* vincula dicho estado con probabilidad $1 - ɣ$ y la media real de una variable aleatoria de rango R no será diferente de la media estimada después de "n" observaciones independientes en más de:

$$e = \sqrt{\frac{R^2 \ln(1 - ɣ)}{2n}}$$

Una característica atractiva de este algoritmo es que tiene buenas garantías de rendimiento, algo que no comparte con los otros árboles de decisión incrementales.

- *Bagging*. Este algoritmo es un meta-clasificador. La función de los meta-clasificadores es agrupar los clasificadores en conjuntos (*Dietterich*, 1997) con el objetivo de mejorar la precisión. La idea básica que subyace en el algoritmo *Bagging* es que el error del modelo construido por un clasificador se debe en parte a la selección de un conjunto de entrenamiento específico. Por tanto, si se crean varios conjuntos de datos tomando muestras con reemplazo y se crean sendos clasificadores, se reducirá el componente de varianza del error de salida

(*Peter Buhlmann*, 2002). Cuando tenemos una instancia para clasificar, cada modelo emite una predicción sobre dicha instancia, la cual cuenta como un voto. Finalmente el clasificador *Bagging* cuenta los votos y asigna la clase más votada para esa instancia.

- *Random Forest*. Este algoritmo es una combinación de árboles predictores de tal forma que cada árbol depende de los valores de un vector aleatorio probado independientemente y con la misma distribución para cada uno de estos. *Random Forest* hace uso de una técnica de agregación desarrollada por *Leo Breiman*, que mejora la precisión en la clasificación mediante la incorporación de aleatoriedad en la construcción de cada clasificador individual. Esta aleatorización se introduce tanto en la construcción del árbol, como en las muestras de entrenamiento. Este clasificador al ser simple de entrenar y ajustar se suele utilizar bastante.

## 5.3. SELECCIÓN DE ATRIBUTOS

Otra parte fundamental de la minería de datos es la selección de atributos ya que dependiendo del conjunto de atributos que tengamos obtendremos mejores o peores resultados. Además hay que tener en cuenta que si se tiene un número excesivo de atributos, el modelo puede ser demasiado complejo, y por tanto, se puede producir *overfitting*. En Weka, una de las maneras que hay para realizar la selección de atributos es mediante la pestaña "*attribute selection*". De esta forma para realizar la evaluación de atributos tendremos que elegir un método de búsqueda y un método de evaluación. A grandes rasgos existen tres posibilidades:

| Método de búsqueda | Método de evaluación |
|---|---|
| *Ranker* | *InfoGainAttributeEval* |
| *Greedy Stepwise* | *CfsSubsetEval* |
| *Greedy Stepwise* | *ClassifierSubsetEval* |

Tabla 6: Métodos de selección de atributos.

Existe una gran diferencia en entre el primer método y los otros dos, puesto que el primero realiza una evaluación de atributos de forma individual mientras que los otros dos buscan los subconjuntos de atributos más relevantes.

Haciendo uso del método de búsqueda *Ranker* y el método de evaluación *InfoGainAttributeEval* se evalúa cada atributo de manera independiente, calculando medidas de correlación del atributo con la clase. Un atributo está correlacionado con la clase si al conocer su valor podemos predecir la clase con cierta probabilidad. Una vez evaluados todos los atributos se quitan los "k" más deficientes. La principal ventaja de este método es que es rápido. Sin embargo, tiene una serie de desventajas debido a que no elimina los atributos redundantes ni detecta atributos que funcionan correctamente de manera conjunta, pero incorrectamente de manera separada.

El segundo método evalúa los subconjuntos de atributos considerando el valor predictivo (correlación) de cada atributo con la clase y correlaciones por redundancias entre atributos. Este método es realmente rápido y además elimina atributos redundantes, sin embargo, puede eliminar atributos que por sí solos no están correlacionados con la clase pero con otro atributo sí que lo están. Existen otros métodos de búsqueda que se le pueden aplicar a este método de evaluación como por ejemplo algoritmos genéticos, el primero el mejor, búsqueda exhaustiva, etc.

Por último, el tercer método evalúa los subconjuntos de atributos ejecutando un algoritmo de minería de datos concreto sobre el conjunto de datos, por tanto, obtenemos los atributos adecuados para un método concreto. El gran problema de este método es que son verdaderamente lentos. Al igual que el método anterior se le pueden aplicar otros métodos de búsqueda, en concreto, el primero el mejor o *ranker*.

Para los tres métodos podemos usar dos formas distintas de evaluar los atributos, utilizando el dataset completo o usando validación cruzada de un número de hojas concreto. La principal diferencia entre ambos métodos es que si empleamos el dataset completo será más rápido aunque menos preciso que si hacemos uso de la validación cruzada.

## 5.4. CONSTRUCCIÓN DE MODELOS

En este apartado se llevará a cabo la descripción de los experimentos que se van a realizar con el de objetivo de comprender por qué llegamos a la conclusión de que un modelo concreto es el mejor. Siempre haremos uso del fichero que obtenemos al pre-procesar los datos, como conjunto de datos de entrenamiento en todos los experimentos.

A continuación se describirán los tres experimentos llevados a cabo en el presente proyecto. Para cada uno de los experimentos se detallarán los aspectos más importantes de los modelos construidos.

### 5.4.1. Experimento 1: Algoritmos de clasificación por defecto.

Este experimento se realizó con el objetivo de saber cuál es el porcentaje de aciertos en el que nos encontramos. Para este experimento no se seleccionó los mejores atributos ni se ajustó los parámetros de cada algoritmo.

Utilizaremos como método de evaluación el fichero de entrenamiento y la validación cruzada de 10 hojas para todos los algoritmos de clasificación. Se construirán dos tablas: una para los algoritmos que construyen reglas y otra para los algoritmos que construyen árboles. En la primera tabla tendremos información sobre el tipo de evaluación, el número de reglas que construye el modelo y la precisión en el porcentaje de aciertos del clasificador, mientras que en la segunda tabla, además de tener el tipo de evaluación que se realiza y la precisión en el porcentaje de aciertos, mostramos el número de hojas y el tamaño del árbol que construye ese clasificador.

| Clasificador | Tipo de test | Número de Reglas | Precisión (% Aciertos) |
|---|---|---|---|
| *OneR* | Fich. Entrenamiento | 270 | 74,72 |
| *OneR* | Validación cruzada | 270 | 73,79 |
| *PART* | Fich. Entrenamiento | 2149 | 80,31 |
| *PART* | Validación cruzada | 2149 | 80,17 |

Tabla 7: Resultados de los algoritmos que construyen reglas.

| Clasificador | Tipo de test | Num.Hojas | Tam.Árbol | Precisión (%Aciertos) |
|---|---|---|---|---|
| *C4.5 (J48)* | Fich.Entrenamiento | 4961 | 9921 | 88,99 |
| *C4.5(J48)* | Validación cruzada | 4961 | 9921 | 80,77 |
| *Hoeffding Tree* | Fich.Entrenamiento | 41 | 82 | 70,20 |
| *Hoeffding Tree* | Validación cruzada | 41 | 82 | 75,89 |
| *Random Forest* | Fich.Entrenamiento | NA | 36855 | 100 |

| | | | | |
|---|---|---|---|---|
| *Random Forest* | Validación cruzada | Desconocido | 36855 | 83,54 |
| *Bagging(HT)* | Fich.Entrenamiento | 36·10* | 71·10* | 76,78 |
| *Bagging(HT)* | Validación cruzada | 36·10* | 71·10* | 76,61 |
| *Bagging(C4.5)* | Fich.Entrenamiento | 8040 | 16079 | 94,75 |
| *Bagging(C4.5)* | Validación cruzada | 8040 | 16079 | 82,58 |

Tabla 8: Resultados de los algoritmos que construyen árboles.

Viendo los resultados obtenidos, se puede comprobar que este problema no es trivial ya que no se obtiene resultados superiores al 90%, excepto cuando se utiliza los algoritmos de clasificados denominados *Random Forest* y *Bagging*, lo cual no es fiable porque en ambos algoritmos se hace uso del conjunto de datos de entrenamiento como método de evaluación. Cabe destacar que el asterisco en el resultado del algoritmo *Hoeffding Tree* quiere decir que es una aproximación propia.

Cabe destacar como varía el porcentaje de aciertos de un algoritmo dependiendo del tipo de evaluación que se realice. Normalmente, cuando utilizamos el fichero de entrenamiento como método de evaluación obtenemos una mayor precisión en el porcentaje de aciertos que cuando hacemos uso de la validación cruzada. Esto se debe a que mediante la validación cruzada se utiliza un conjunto de datos para entrenar y otro conjunto de datos para testear, por lo que obtenemos un porcentaje de aciertos más real. Sin embargo, haciendo uso del otro tipo de evaluación se entrena y se testea con el mismo conjunto de datos.

Expondremos la matriz de confusión y el resumen estadístico por clase de cada uno de los modelos construidos con la finalidad de tener más información con la que enriquecer la decisión final sobre el mejor modelo construido.

```
=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0,868    0,396    0,730      0,868    0,793      0,495    0,736     0,707     Synchronized
                 0,398    0,019    0,739      0,398    0,517      0,501    0,689     0,365     Congested
                 0,670    0,088    0,789      0,670    0,725      0,610    0,791     0,638     Free
Weighted Avg.    0,747    0,250    0,751      0,747    0,738      0,533    0,749     0,644

=== Confusion Matrix ===

    a     b     c   <-- classified as
 63792  2216  7486 |     a = Synchronized
  9122  6261   367 |     b = Congested
 14439     0 29373 |     c = Free
```

Figura 16: algoritmo *OneR*, método evaluación: fichero entrenamiento.

```
=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure  MCC      ROC Area  PRC Area  Class
               0,860     0,406     0,723       0,860   0,786      0,475    0,727     0,699     Synchronized
               0,358     0,024     0,665       0,358   0,466      0,442    0,667     0,314     Congested
               0,670     0,088     0,789       0,670   0,725      0,610    0,791     0,638     Free
Weighted Avg.  0,738     0,256     0,738       0,738   0,728      0,515    0,741     0,633

=== Confusion Matrix ===

    a     b     c   <-- classified as
 63173  2835  7486 |     a = Synchronized
  9740  5643   367 |     b = Congested
 14437     2 29373 |     c = Free
```

Figura 17: algoritmo *OneR*, método evaluación: validación cruzada 10 hojas.

```
=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure  MCC      ROC Area  PRC Area  Class
               0,893     0,192     0,851       0,893   0,872      0,706    0,945     0,955     Synchronized
               0,711     0,012     0,885       0,711   0,789      0,770    0,980     0,899     Congested
               0,838     0,074     0,848       0,838   0,843      0,767    0,969     0,943     Free
Weighted Avg.  0,854     0,132     0,854       0,854   0,853      0,733    0,957     0,945

=== Confusion Matrix ===

    a     b     c   <-- classified as
 65654  1447  6393 |     a = Synchronized
  4364 11206   180 |     b = Congested
  7100     5 36707 |     c = Free
```

Figura 18: algoritmo *PART*, método evaluación: fichero de entrenamiento.

```
=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure  MCC      ROC Area  PRC Area  Class
               0,852     0,255     0,805       0,852   0,827      0,602    0,882     0,876     Synchronized
               0,648     0,023     0,792       0,648   0,713      0,683    0,924     0,762     Congested
               0,773     0,095     0,799       0,773   0,786      0,684    0,930     0,875     Free
Weighted Avg.  0,802     0,175     0,801       0,802   0,800      0,639    0,903     0,862

=== Confusion Matrix ===

    a     b     c   <-- classified as
 62582  2639  8273 |     a = Synchronized
  5296 10210   244 |     b = Congested
  9891    35 33886 |     c = Free
```

Figura 19: Algoritmo *PART*, método de evaluación: validación cruzada 10 hojas.

```
=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure  MCC      ROC Area  PRC Area  Class
               0,724     0,315     0,739       0,724   0,731      0,408    0,719     0,750     Synchronized
               0,589     0,105     0,429       0,589   0,497      0,424    0,834     0,314     Congested
               0,706     0,096     0,784       0,706   0,743      0,629    0,895     0,804     Free
Weighted Avg.  0,702     0,218     0,717       0,702   0,707      0,482    0,791     0,716

=== Confusion Matrix ===

    a     b     c   <-- classified as
 53178 12142  8174 |     a = Synchronized
  6117  9284   349 |     b = Congested
 12649   211 30952 |     c = Free
```

Figura 20: Algoritmo *Hoeffding Tree*, método de evaluación: fichero entrenamiento.

```
=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0,854    0,351    0,750      0,854    0,799      0,518    0,829     0,820     Synchronized
                 0,507    0,023    0,750      0,507    0,605      0,577    0,894     0,641     Congested
                 0,689    0,095    0,781      0,689    0,732      0,615    0,909     0,851     Free
Weighted Avg.    0,759    0,228    0,760      0,759    0,754      0,557    0,863     0,809

=== Confusion Matrix ===

     a     b     c   <-- classified as
 62785  2620  8089 |     a = Synchronized
  7369  7990   391 |     b = Congested
 13563    42 30207 |     c = Free
```

Figura 21: Algoritmo *Hoeffding Tree*, método de evaluación: validación cruzada 10 hojas.

```
=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 1,000    0,000    1,000      1,000    1,000      1,000    1,000     1,000     Synchronized
                 1,000    0,000    1,000      1,000    1,000      1,000    1,000     1,000     Congested
                 1,000    0,000    1,000      1,000    1,000      1,000    1,000     1,000     Free
Weighted Avg.    1,000    0,000    1,000      1,000    1,000      1,000    1,000     1,000

=== Confusion Matrix ===

     a     b     c   <-- classified as
 73494     0     0 |     a = Synchronized
     0 15750     0 |     b = Congested
     0     0 43812 |     c = Free
```

Figura 22: Algoritmo *Random Forest*, método de evaluación: fichero entrenamiento.

```
=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0,892    0,231    0,826      0,892    0,858      0,670    0,920     0,929     Synchronized
                 0,688    0,013    0,877      0,688    0,771      0,751    0,961     0,862     Congested
                 0,793    0,074    0,841      0,793    0,816      0,731    0,957     0,922     Free
Weighted Avg.    0,835    0,154    0,837      0,835    0,834      0,699    0,937     0,919

=== Confusion Matrix ===

     a     b     c   <-- classified as
 65571  1501  6422 |     a = Synchronized
  4747 10835   168 |     b = Congested
  9039    15 34758 |     c = Free
```

Figura 23: Algoritmo *Random Forest*, método de evaluación: validación cruzada 10 hojas.

```
=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0,923    0,148    0,885      0,923    0,904      0,780    0,949     0,952     Synchronized
                 0,788    0,008    0,934      0,788    0,854      0,841    0,972     0,902     Congested
                 0,871    0,056    0,885      0,871    0,878      0,819    0,972     0,951     Free
Weighted Avg.    0,890    0,101    0,891      0,890    0,889      0,800    0,960     0,946

=== Confusion Matrix ===

     a     b     c   <-- classified as
 67829   851  4814 |     a = Synchronized
  3184 12404   162 |     b = Congested
  5622    29 38161 |     c = Free
```

Figura 24: Algoritmo *C4.5 (J48)*, método de evaluación: fichero de entrenamiento.

```
=== Detailed Accuracy By Class ===

               TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
               0,851    0,238    0,815      0,851    0,833      0,617    0,862     0,832     Synchronized
               0,675    0,022    0,804      0,675    0,734      0,705    0,905     0,696     Congested
               0,786    0,096    0,800      0,786    0,793      0,693    0,917     0,850     Free
Weighted Avg.  0,809    0,166    0,809      0,809    0,808      0,652    0,885     0,822

=== Confusion Matrix ===

     a     b     c   <-- classified as
 62580  2517  8397 |     a = Synchronized
  4901 10639   210 |     b = Congested
  9304    78 34430 |     c = Free
```

Figura 25: Algoritmo *C4.5 (J48)*, método de evaluación: validación cruzada 10 hojas.

```
=== Detailed Accuracy By Class ===

               TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
               0,971    0,081    0,936      0,971    0,953      0,894    0,990     0,991     Synchronized
               0,853    0,003    0,977      0,853    0,911      0,902    0,997     0,980     Congested
               0,939    0,022    0,955      0,939    0,947      0,921    0,994     0,990     Free
Weighted Avg.  0,946    0,052    0,947      0,946    0,946      0,904    0,992     0,989

=== Confusion Matrix ===

     a     b     c   <-- classified as
 71348   318  1828 |     a = Synchronized
  2187 13441   122 |     b = Congested
  2663     3 41146 |     c = Free
```

Figura 26: Algoritmo *Bagging + C4.5 (J48)*, método de evaluación: fichero entrenamiento.

```
=== Detailed Accuracy By Class ===

               TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
               0,878    0,230    0,825      0,878    0,851      0,654    0,912     0,921     Synchronized
               0,685    0,015    0,856      0,685    0,761      0,739    0,954     0,848     Congested
               0,796    0,083    0,826      0,796    0,810      0,720    0,952     0,916     Free
Weighted Avg.  0,828    0,156    0,829      0,828    0,827      0,686    0,930     0,911

=== Confusion Matrix ===

     a     b     c   <-- classified as
 64512  1791  7191 |     a = Synchronized
  4776 10796   178 |     b = Congested
  8920    19 34873 |     c = Free
```

Figura 27: Algoritmo *Bagging + C4.5 (J48)*, método de evaluación: validación cruzada 10 hojas.

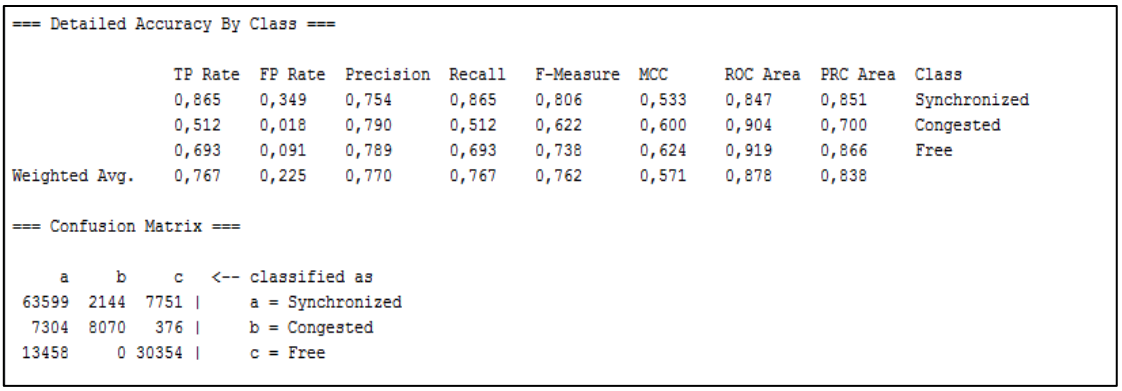
```
=== Detailed Accuracy By Class ===

               TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
               0,874    0,352    0,754      0,874    0,809      0,541    0,852     0,857     Synchronized
               0,528    0,019    0,789      0,528    0,633      0,609    0,909     0,701     Congested
               0,684    0,083    0,801      0,684    0,738      0,628    0,922     0,870     Free
Weighted Avg.  0,770    0,224    0,774      0,770    0,765      0,578    0,882     0,843

=== Confusion Matrix ===

     a     b     c   <-- classified as
 64201  2219  7074 |     a = Synchronized
  7082  8318   350 |     b = Congested
 13857     1 29954 |     c = Free
```

Figura 28: Algoritmo *Bagging + Hoeffding Tree*, método de evaluación: fichero entrenamiento.



```
=== Detailed Accuracy By Class ===

               TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
               0,865    0,349    0,754      0,865    0,806      0,533    0,847     0,851     Synchronized
               0,512    0,018    0,790      0,512    0,622      0,600    0,904     0,700     Congested
               0,693    0,091    0,789      0,693    0,738      0,624    0,919     0,866     Free
Weighted Avg.  0,767    0,225    0,770      0,767    0,762      0,571    0,878     0,838

=== Confusion Matrix ===

    a     b     c   <-- classified as
 63599  2144  7751 |     a = Synchronized
  7304  8070   376 |     b = Congested
 13458     0 30354 |     c = Free
```

Figura 29: *Algoritmo Bagging + Hoeffding Tree*, método de evaluación: validación cruzada 10 hojas.

Centrándonos en la matriz de confusión y el resumen estadístico por clase de cada uno de los modelos obtenidos, descartaremos aquellos modelos en los que se utilizó el fichero de entrenamiento como método de evaluación ya que como dijimos anteriormente no obtenemos resultados fiables. Además del resumen estadístico utilizaremos el parámetro *ROC Area* ya que entre mayor sea el área que está bajo la curva ROC, mejor será el clasificador. Además, este parámetro muestra los beneficios reflejados por TP y los costos reflejados por FP en la clasificación.

Como podemos ver en la matriz de confusión de cada modelo construido, no hay ninguna clase que sea clasificada correctamente en su totalidad. Además, hay que destacar que si queremos mejorar aún más el clasificador tendríamos que conseguir algún atributo que permita a dicho clasificador distinguir correctamente la clase libre de sincronizado, ya que como podemos ver en la matriz, donde más se equivoca el clasificador es en clasificar instancias que tiene que clasificar como libre y este las clasifica como sincronizado y viceversa.

Si únicamente tuviéramos en cuenta el parámetro *ROC Area* para indicar que clasificador es el mejor, tendríamos que decidir que el mejor modelo es construido por el clasificador *Random Forest*. Sin embargo, tendremos otros parámetros como la precisión en el porcentaje de aciertos, el tamaño del árbol o el número de reglas, dependiendo del tipo de modelo que se construya. Por ello, indicaremos que es lo que más nos ha llamado la atención de alguno de los algoritmos utilizados para posteriormente tomar una decisión final sobre que algoritmo es el mejor.

- *OneR*: haciendo sólo uso del atributo denominado "*TotalInstancesThisHour*" se obtiene un porcentaje de aciertos bastante alto.

- *Random Forest*: este algoritmo construye un árbol con un número de hojas considerables, en comparación con los algoritmos que construyen árboles. El tamaño del árbol y el porcentaje de acierto que se obtiene utilizando como método de evaluación el conjunto de datos completo, nos hace sospechar que este modelo está sufriendo *overfitting*. Para concluir con este clasificador, es necesario indicar que este algoritmo no proporciona el número de hojas que contiene el modelo construido.
- *Hoeffding Tree*: no es el algoritmo con el que se obtiene el mejor porcentaje de acierto, pero si es el mejor algoritmo en relación porcentaje de aciertos y tamaño del árbol.
- *Bagging*: de este meta-clasificador hay que subrayar que obtenemos un porcentaje de acierto mejor para los dos algoritmos bases utilizados, pero haciendo que el tamaño del árbol aumente bastante.

La gráfica que mostramos a continuación nos refleja el porcentaje de aciertos para cada uno de los algoritmos utilizados.

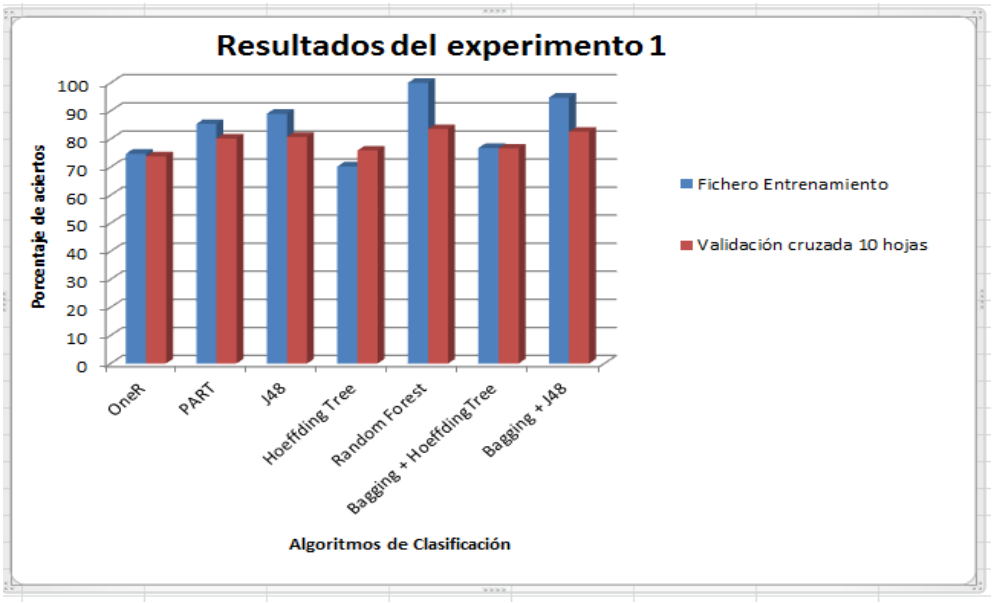



Figura 30: Resultados del experimento 1.

Tras realizar estas pruebas para cada uno de los algoritmos seleccionados, creemos que el mejor modelo es el obtenido por el meta-clasificador *Bagging* utilizando como clasificador base *C4.5 (J48)* ya que obtiene un porcentaje de acierto bastante bueno. Y aunque el parámetro *ROC Area* no sea el mejor, la diferencia con el que obtiene el mejor valor, *Random Forest*, es mínima, en concreto, 7 milésimas. También hemos

tenido muy en cuenta el tamaño del árbol para decidirnos por este meta-clasificador en lugar de por el algoritmo *Random Forest*, ya que el meta-clasificador construye un árbol con un tamaño de 16079 nodos mientras que con *Hoeffding Tree* el tamaño del árbol es de 36855 nodos. Para concluir este experimento resaltaremos que hay que prestar atención al algoritmo *Hoeffding Tree* para los futuros experimentos debido a que proporciona resultados bastantes satisfactorios para un árbol con un tamaño de 82 nodos.

### 5.4.2. Experimento 2: Algoritmos de clasificación modificando parámetros.

En este segundo experimento haremos uso de los mismos algoritmos de clasificación pero ajustando los parámetros de cada uno de ellos. Este segundo experimento tiene el objetivo de conseguir mejorar los resultados del experimento 1, tanto en la precisión del porcentaje de acierto, valor del *ROC Area*, como en disminuir el número de hojas, tamaño del árbol y número de reglas.

En este experimento haremos sólo uso de la validación cruzada de 10 hojas ya que como hemos visto en el primer experimento se obtiene porcentaje de aciertos más reales.

A continuación describiremos y ajustaremos los parámetros más influyentes para cada uno de los algoritmos:

- *OneR*. Este algoritmo tiene un único parámetro principal denominado "*minBucketSize*", el cual tiene la función de indicar cuantas instancias deben ser cubiertas por cada regla generada.

  Hemos realizado pruebas modificando el parámetro "*minBucketSize*" y hemos obtenido los siguientes resultados.

| *minBucketSize* | Precisión (% Aciertos) |
|---|---|
| 6 | 73,79 |
| 10 | 73,96 |
| 30 | 74,17 |
| 45 | 74,23 |
| 100 | 74,28 |

Tabla 9: Ajuste del parámetro para el algoritmo *OneR*.

Como podemos ver en los resultados, el porcentaje de acierto no mejora sustancialmente a medida que aumentamos el número de instancias que debe ser cubierta por cada regla. Sin embargo, el número de reglas si disminuye

considerablemente a medida que aumentamos este parámetro ya que, por ejemplo, si le indicamos que cada regla debe cubrir 6 instancias obtenemos 250 reglas mientras que si le indicamos que cada regla debe cubrir 100 instancias obtenemos únicamente 6 reglas.

```
=== Classifier model (full training set) ===

TotalInstancesThisHour:
        < 7.5   -> Free
        < 1744.5        -> Synchronized
        < 1778.5        -> Congested
        < 1932.5        -> Synchronized
        >= 1932.5       -> Congested
(98883/133056 instances correct)
```

Figura 31: Modelo de 6 reglas construido por el algoritmo *OneR*.

- *C4.5 (J48)*. Este algoritmo tiene una serie de parámetros que se pueden modificar pero hay dos que son fundamentales:
  - *minNumObj*: Indicamos el número mínimo de instancias permitido en cada hoja.
  - *confidence level*: es el factor de confianza para la poda, que influye en el tamaño y la capacidad de predicción del árbol construido. Entre menor sea este número, más operaciones de poda se permiten.

| minNumObj | confidence Level | Num Hojas | Tam árbol | Precisión (% Aciertos) |
|---|---|---|---|---|
| 2 | 0,1 | 1997 | 3993 | 81.40 |
| 5 | 0,1 | 1261 | 2521 | 81.48 |
| 3 | 0,05 | 1228 | 2455 | 81.55 |

Tabla 10: Ajuste de parámetros para el algoritmo *C4.5 (J48)*.

En este algoritmo variando los valores que vienen por defecto se mejoran el porcentaje de aciertos pero en menos de un 1%. También disminuye el número de hojas y el tamaño del árbol. Por ello, tras haber realizado las pruebas diríamos que los mejores valores parámetros son confidenceLevel 0.05 y minNumObj 3, puesto que es el que mejor porcentaje de aciertos nos proporciona y construye un árbol de un tamaño menor respecto a los otros modelos.

- *PART***.** Este clasificador tiene los mismos parámetros principales que el algoritmo *C4.5 (J48)* y tienen la misma funcionalidad.

  Los resultados que hemos obtenido modificando estos parámetros son los siguientes:

| *minNumObj* | *confidence Level* | Num reglas | Precisión (% Aciertos) |
|---|---|---|---|
| 2 | 0,1 | 2123 | 80,20 |
| 5 | 0,1 | 1329 | 80,52 |
| 3 | 0,05 | 984 | 81,54 |

Tabla 11: Ajuste de parámetros para el algoritmo *PART*.

Al igual que en el algoritmo *C4.5 (J48)*, conseguimos mejorar el porcentaje de acierto un 1% aproximadamente, lo cual es despreciable. Lo más destacable es que conseguimos que el número de reglas disminuya a la mitad, lo cual es bastante bueno ya que conseguimos un modelo más general y, por tanto, será más robusto a la hora de clasificar instancias en un futuro. Tras las pruebas realizadas creemos que los mejores valores de los parámetros son *confidenceLevel* 0.05 y *minNumObj* 3, los mismos valores que hemos decidido para el algoritmo *C4.5 (J48)*.

- *Hoeffding Tree*. Los parámetros fundamentales en la configuración de este algoritmo son los siguientes:
  - *gracePeriod*: el número de instancias que una hoja debe observar entre cada intento de división.
  - *splitCriterion*: criterio de división usado.
  - *minimunFractionOfWeightInfoGain*: el umbral por debajo del cual una división se verá obligado a romper los vínculos.
  - *leafPredictionStrategy*: clasificador usado para clasificar las hojas.

| *gracePeriod* | *splitCriterion* | *minimunFractionOf - WeightInfoGain* | *leafPrediction Strategy* | Precisión (% Aciertos) |
|---|---|---|---|---|
| 300 | *Gini Split* | 0,05 | *Naives Bayes Adaptative* | 76,08 |
| 400 | *Gini split* | 0,05 | *Naives Bayes Adaptative* | 76,43 |
| 450 | *Gini Split* | 0.05 | *Naives Bayes Adaptative* | 76,56 |
| 500 | *Gini Split* | 0,05 | *Naives Bayes* | 76,58 |

| | | | *Adaptative* | |
|---|---|---|---|---|
| 300 | *Gini Split* | 0,05 | *Majority Class* | 75,29 |
| 300 | *Gini Split* | 0,05 | *Naives Bayes* | 76,11 |
| 400 | *Gini Split* | 0,05 | *Naives Bayes* | 75,76 |
| 400 | *Info Gain Split* | 0,05 | *Naives Bayes Adaptative* | 75,77 |
| 300 | *Info Gain* | 0,01 | *Naives Bayes Adaptative* | 76,56 |

Tabla 12: Ajuste de parámetros para el algoritmo *Hoeffding Tree*.

Tras haber hecho ajustes en los parámetros de este algoritmo, hemos comprobado que el porcentaje de aciertos mejora en un 1%, igualando a los algoritmos anteriores. En cuanto al tamaño del árbol se mantiene respecto a los valores obtenidos para este algoritmo en el experimento.

- *Random Forest*. Los parámetros que consideramos fundamentales en este algoritmo son los siguientes:
  - *maxDepth*: máxima profundidad de los árboles.
  - *numIterations*: número de árboles que se construirán.
  - *Seed*: valor aleatorio para la elección de número de variables y de instancias que se usarán para la construcción de cada árbol.

| *maxDepth* | *numIterations* | *Seed* | Num Hojas | Tam Árbol | Precisión (% Aciertos) |
|---|---|---|---|---|---|
| Indefinido | 100 | 5 | Desconocido | 35013 | 83,56 |
| Indefinido | 150 | 1 | Desconocido | Desconocido | 83,66 |
| 20000 | 100 | 1 | Desconocido | 36089 | 83,61 |

Tabla 13: Ajuste de parámetros para el algoritmo *Random Forest*.

Para ajustar los parámetros de este algoritmo hemos tenido problemas por falta de memoria de Random Access Memory, Memoria de acceso aleatorio (RAM). Por ello, con los resultados que tenemos de las pruebas hemos decidido que el mejor ajuste es el que viene por defecto al hacer uso del algoritmo.

- *Bagging*. En este algoritmo las parámetros más importantes son "*numIterations*" donde marcamos el número de modelos base que queremos crear, y "*Classifier*", donde seleccionamos el método base con el cual deseamos crear los modelos base.

Primero obtendremos resultados ajustando el parámetro *numIterations* para los dos clasificadores base que utilizamos, *Hoeffding Tree* y *C4.5 (J48).*

Posteriormente mostraremos los resultados que nos proporciona ajustando tanto el algoritmo *Bagging* como el clasificador base que utilicemos.

| Clasificador | *numIterations* | Num Hojas | Tam Árbol | Precisión (% Aciertos) |
|---|---|---|---|---|
| *Hoeffding Tree* | 20 | 32·20 | 70·20 | 76,69 |
| *Hoeffding Tree* | 30 | 32·30 | 70·30 | 76,70 |
| *C4.5(J48)* | 20 | 7678 | 15355 | 83,26 |
| *C4.5(J48)* | 30 | 7797 | 15593 | 83,80 |

Tabla 14: Ajuste de parámetros para el meta-clasificador *Bagging*.

Analizando los resultados obtenidos llegamos a la conclusión de que ajustar el parámetro *numIterations* con el clasificador base *Hoeffding Tree* no nos aporta nada positivo ya que no mejora la precisión del porcentaje de aciertos y el tamaño del árbol aumenta, por tanto, el mejor ajuste del parámetro denominado *numIterations* es el que viene por defecto que es 10. En cuanto al uso del algoritmo *C4.5 (J48)* obtenemos mejor precisión de porcentaje de aciertos ajustando el parámetro y además disminuye el tamaño del árbol, por tanto, el mejor modelo lo obtenemos cuando el parámetro *numIterations* tiene el valor de 30.

Ajustando tanto los parámetros del meta-clasificador como los del clasificador base, obtenemos los siguientes resultados.

| Clasificador | Num Hojas | Tam árbol | Precision (% Aciertos) |
|---|---|---|---|
| *Hoeffding Tree* | 36·10 | 71·10 | 76,78 |
| *C4.5(J48)* | 3231 | 6461 | 83,35 |

Tabla 15: Ajuste de parámetros para el meta-clasificador *Bagging* y los clasificadores base.

Incluiremos la matriz de confusión y el resumen estadístico por clase para aquellos modelos que consideramos los mejores tras ajustar los parámetros del clasificador, excepto para el algoritmo *OneR* ya que es el mismo que el experimento 1.

```
=== Detailed Accuracy By Class ===

               TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
               0,867    0,244    0,814      0,867    0,840      0,629    0,882     0,872     Synchronized
               0,642    0,016    0,841      0,642    0,728      0,705    0,919     0,751     Congested
               0,792    0,090    0,811      0,792    0,801      0,706    0,937     0,893     Free
Weighted Avg.  0,816    0,167    0,816      0,816    0,814      0,663    0,905     0,865

=== Confusion Matrix ===

     a     b     c   <-- classified as
 63729  1902  7863 |     a = Synchronized
  5436 10104   210 |     b = Congested
  9123     3 34686 |     c = Free
```

Figura 32: Algoritmo *C4.5 (J48).*

```
=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.892     0.236       0.824     0.892      0.856      0.916    Synchronized
                 0.667     0.012       0.882     0.667      0.76       0.952    Congested
                 0.796     0.075       0.839     0.796      0.817      0.955    Free
Weighted Avg.    0.834     0.156       0.836     0.834      0.832      0.933

=== Confusion Matrix ===

     a     b     c   <-- classified as
 65553  1400  6541 |     a = Synchronized
  5090 10506   154 |     b = Congested
  8952     4 34856 |     c = Free
```

Figura 33: Algoritmo *Bagging + C4.5 (J48).*

```
=== Detailed Accuracy By Class ===

               TP Rate  FP Rate  Precision  Recall  F-Measure  MCC      ROC Area  PRC Area  Class
               0,854    0,252    0,807      0,854   0,829      0,606    0,889     0,887     Synchronized
               0,643    0,021    0,802      0,643   0,714      0,685    0,932     0,784     Congested
               0,779    0,096    0,800      0,779   0,789      0,688    0,936     0,890     Free
Weighted Avg.  0,804    0,173    0,804      0,804   0,802      0,643    0,910     0,876

=== Confusion Matrix ===

     a     b     c   <-- classified as
 62728  2472  8294 |     a = Synchronized
  5371 10131   248 |     b = Congested
  9664    30 34118 |     c = Free
```

Figura 34: *PART*.

```
=== Detailed Accuracy By Class ===

               TP Rate  FP Rate  Precision  Recall  F-Measure  MCC      ROC Area  PRC Area  Class
               0,865    0,337    0,760      0,865   0,809      0,543    0,852     0,854     Synchronized
               0,524    0,019    0,790      0,524   0,630      0,607    0,904     0,696     Congested
               0,704    0,091    0,792      0,704   0,746      0,634    0,922     0,870     Free
Weighted Avg.  0,772    0,218    0,774      0,772   0,767      0,581    0,882     0,841

=== Confusion Matrix ===

     a     b     c   <-- classified as
 63563  2188  7743 |     a = Synchronized
  7127  8257   366 |     b = Congested
 12946     1 30865 |     c = Free
```

Figura 35: *Bagging + Hoeffding Tree*.

```
=== Detailed Accuracy By Class ===

               TP Rate  FP Rate  Precision  Recall  F-Measure  MCC      ROC Area  PRC Area  Class
               0,857    0,340    0,757      0,857   0,804      0,531    0,838     0,828     Synchronized
               0,511    0,020    0,774      0,511   0,616      0,591    0,891     0,650     Congested
               0,703    0,096    0,782      0,703   0,741      0,625    0,914     0,861     Free
Weighted Avg.  0,766    0,222    0,767      0,766   0,761      0,569    0,869     0,818

=== Confusion Matrix ===

     a     b     c   <-- classified as
 63003  2329  8162 |     a = Synchronized
  7293  8054   403 |     b = Congested
 12978    20 30814 |     c = Free
```

Figura 36: *Hoeffding Tree*.

```
=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0,893    0,231    0,827      0,893    0,859      0,671    0,921     0,929     Synchronized
                 0,688    0,013    0,874      0,688    0,770      0,750    0,961     0,862     Congested
                 0,795    0,073    0,843      0,795    0,818      0,733    0,957     0,923     Free
Weighted Avg.    0,836    0,153    0,838      0,836    0,835      0,701    0,938     0,919

=== Confusion Matrix ===

     a     b     c   <-- classified as
 65605  1548  6341 |     a = Synchronized
  4757 10830   163 |     b = Congested
  8979    12 34821 |     c = Free
```

Figura 37: *Random Forest*.

Al igual que en el experimento 1, en la matriz de confusión se puede ver que el problema al clasificar este dataset se encuentra principalmente en la distinción de la clase sincronizada y libre, ya que en esas dos clases es donde más falla el clasificador. En cuanto al parámetro "*ROC Area*", se sigue obteniendo su mayor valor con el algoritmo *Random Forest* aunque hemos conseguido que los otros algoritmos mejoren este parámetro ligeramente.

Llegados a este punto del experimento 2, solo nos falta realizar una comparación entre el experimento 1 y el 2. Para ello haremos uso del porcentaje de aciertos de cada algoritmo, usando como método de evaluación la validación cruzada de 10 hojas. Para los resultados del experimento 2 escogeremos aquellos que han dado mejores resultados en cuanto a la precisión del porcentaje de aciertos.

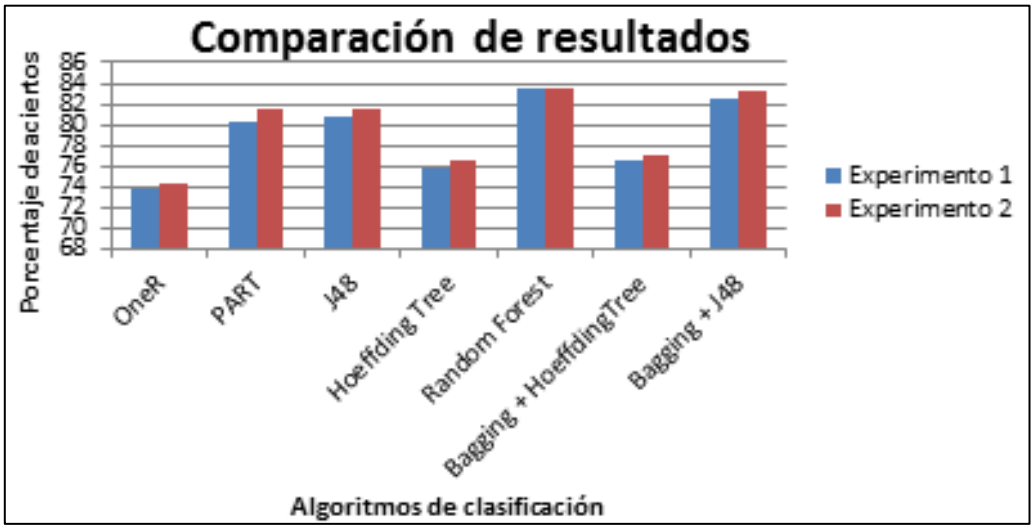



Figura 38: Comparación de resultados entre el experimento 1 y 2.

Para concluir este experimento, daremos nuestro punto de vista sobre los resultados obtenidos. Antes de realizar este experimento, estábamos seguros de que el porcentaje

de aciertos de cada uno de los algoritmos utilizados iban a mejorar pero de una forma más impactante. Lo máximo que hemos conseguido aumentar el porcentaje de aciertos es un 1%, aproximadamente, mientras que nuestras expectativas estaban en subir un 5%. Aunque no hayamos alcanzado ese objetivo estamos satisfechos ya que dependiendo del modelo, tanto el tamaño del árbol como el número de reglas, ha disminuido considerablemente excepto cuando hacemos uso del meta-clasificador *Bagging* con el clasificador base *Hoeffding Tree*.

Tras obtener estos resultados, creemos que el mejor algoritmo es el meta-clasificador *Bagging* con el clasificador base *C4.5 (J48)*. Esta decisión no se ha tomado por el porcentaje de aciertos del clasificador únicamente, ya que hay algoritmos con un porcentaje de aciertos mínimamente mejor, sino que también ha influido el tamaño del árbol debido a que se ha conseguido reducir más de la mitad. También hemos tenido en cuenta que el parámetro *ROC Area* ha mejorado para este meta-clasificador y aunque no tenga el mayor valor, su valor está bastante próximo. Además, mediante el uso de este meta-clasificador disminuimos el riesgo de que se haya producido *overfitting* ya que ninguno de los clasificadores que lo compone tiene todos los datos de entrenamiento.

### 5.4.3. Experimento 3: Selección de atributos más algoritmos de clasificación.

En este experimento realizaremos una selección de atributos y después aplicaremos los algoritmos de clasificación. Como explicamos en los apartados anteriores, existen tres métodos de selección de atributos. Por ello iremos usando cada uno de los métodos y obteniendo resultados con los algoritmos clasificadores. Tanto en la selección de atributos como en el uso de clasificadores utilizaremos como método de evaluación la validación cruzada de 10 hojas.

Para cada uno de los algoritmos clasificadores que utilicemos, ajustaremos sus parámetros a los valores que hemos considerado mejores en el experimento 2.

A grandes rasgos, estos son los tres tipos de métodos que se utilizan para la selección de atributos.

| Método de búsqueda | Método de evaluación |
|---|---|
| *Ranker* | *InfoGainAttributeEval* |
| *Greedy Stepwise* | *CfsSubsetEval* |
| *Greedy Stepwise* | *ClassifierSubsetEval* |

Tabla 16: Métodos de selección de atributos.

Comenzaremos por el primer método, el cual utiliza como método de búsqueda el algoritmo *Ranker* y como método de evaluación el algoritmo *InfoGainAttributeEval*. Para el primer método de selección de atributos escogeremos aquellos atributos que obtengan más de un 0,40 de "mérito" para ser elegidos.

```
=== Attribute selection 10 fold cross-validation (stratified), seed: 1 ===

average merit      average rank  attribute
 0.556 +- 0.001     1   +- 0       8 TotalInstancesThisHour
 0.522 +- 0.001     2   +- 0       6 InflowCar
 0.457 +- 0.001     3   +- 0       7 OutflowCar
 0.231 +- 0.001     4   +- 0       5 MediumDistance
 0.2   +- 0         5   +- 0       4 MDPreviousPosition
 0.179 +- 0.001     6   +- 0       2 Latitude
 0.146 +- 0         7   +- 0       1 Date
 0.037 +- 0         8   +- 0       3 Longitude
```

Figura 39: Selección de atributos del primer método.

Como podemos ver en la ilustración, solo hay tres atributos que obtengan más de un 0,40 de "mérito". Mediante Weka eliminaremos los atributos no elegidos y aplicaremos los algoritmos de clasificación. Estos son los resultados que obtenemos

.

| Clasificador | Num reglas | Precision (% Aciertos) |
|---|---|---|
| *OneR* | 5 | 74,28 |
| *PART* | 128 | 76,27 |

Tabla 17: Resultados de los algoritmos que construyen reglas.

| Clasificador | Num Hojas | Tam árbol | Precision (% Aciertos) |
|---|---|---|---|
| *C4.5(J48)* | 53 | 105 | 76,37 |
| *Hoeffding Tree* | 8 | 14 | 75,75 |
| *Random Forest* | Desconocido | 32055 | 73,98 |
| *Bagging(HF)* | 30·10* | 50·10* | 76,13 |
| *Bagging(C4.5(J48))* | 256 | 511 | 76,49 |

Tabla 18: Resultados de los algoritmos que construyen árbol.

Después de conseguir los resultados para cada uno de los algoritmos de clasificación que utilizamos, realizaremos una comparativa con los resultados del experimento 2 para ver si esta selección de atributos ha sido beneficiosa.

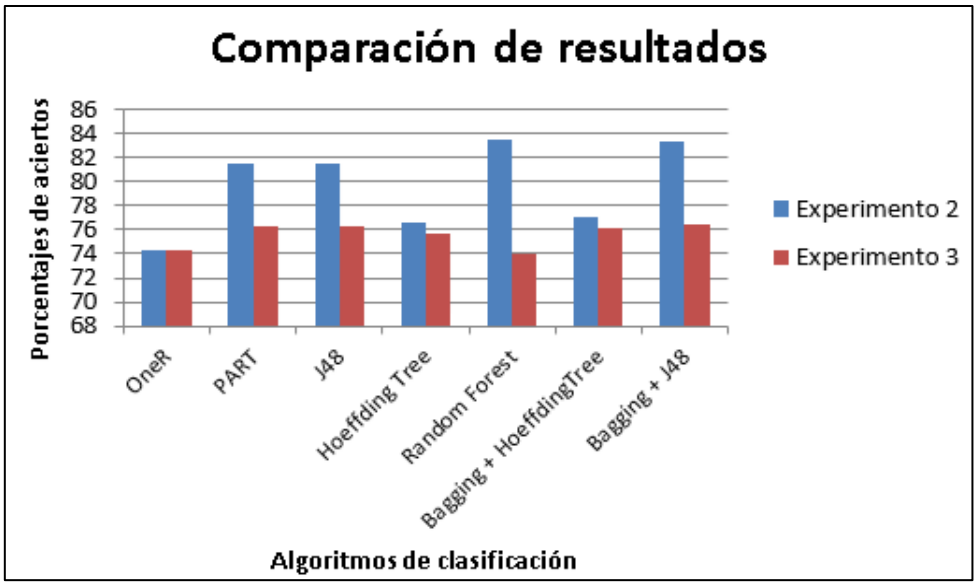



Figura 40: Comparación de resultados entre el experimento 2 y 3.

Como podemos ver en la gráfica, la selección de atributos ha hecho que disminuyera en varios puntos en la precisión del porcentaje de aciertos de algunos algoritmos como por ejemplo el algoritmo *PART* o *C4.5 (J48).* El único algoritmo que ha mantenido el porcentaje de aciertos para ambos experimentos es el que se denomina *OneR* y esto se debe a que, como dijimos anteriormente, sólo hace uso del atributo *TotalInstancesThisHour* para construir el modelo por lo que la selección de atributos no le ha afectado.

Estos resultados se deben a que este método de selección de atributos sólo evalúa los atributos de manera individual y por tanto no detecta atributos que funcionan bien de manera conjunta. Para concluir, creemos que esta selección de atributos no nos aporta nada ya que el porcentaje de aciertos de algunos algoritmos disminuye varios puntos y, por tanto, estamos construyendo modelos peores que los que construíamos en el experimento 2. Al obtener estos resultados hemos decidido no realizar un análisis a la matriz de confusión y el resumen estadístico por clase de estos modelos ya que no nos van a ser útil.

El segundo método utiliza como método de búsqueda el algoritmo *Greedy Stepwise* y como método de evaluación el algoritmo *CfsSubsetEval*. Para este método de selección de atributos escogeremos aquellos atributos que sean elegidos en más del 50% de las hojas utilizadas en la validación cruzada.

```
=== Attribute selection 10 fold cross-validation (stratified), seed: 1 ===

number of folds (%)  attribute
          10(100 %)    1 Date
           0(  0 %)    2 Latitude
           0(  0 %)    3 Longitude
          10(100 %)    4 MDPreviousPosition
          10(100 %)    5 MediumDistance
          10(100 %)    6 InflowCar
          10(100 %)    7 OutflowCar
          10(100 %)    8 TotalInstancesThisHour
```

Figura 41: Selección de atributos del segundo método.

| Clasificador | Num reglas | Precision (% Aciertos) |
|---|---|---|
| *OneR* | 5 | 74,28 |
| *PART* | 1041 | 79,64 |

Tabla 19: Resultados de los algoritmos que construyen reglas.

| Clasificador | Num Hojas | Tam árbol | Precision (% Aciertos) |
|---|---|---|---|
| *C4.5(J48)* | 428 | 855 | 80,56 |
| *Hoeffding Tree* | 10 | 18 | 76,01 |
| *Random Forest* | Desconocido | 44177 | 80,26 |
| *Bagging(HF)* | 35·10* | 65·10* | 76,56 |
| *Bagging(C4.5(J48))* | 1854 | 3707 | 81,31 |

Tabla 20: Resultados de los algoritmos que construyen árbol.

Tras obtener los resultados para cada uno de los algoritmos de clasificación que usamos, realizaremos una comparativa con los resultados del experimento 2 para ver si esta selección de atributos ha sido beneficiosa. También compararemos estos resultados con los que hemos obtenido después de aplicar el primer método de selección de atributos.

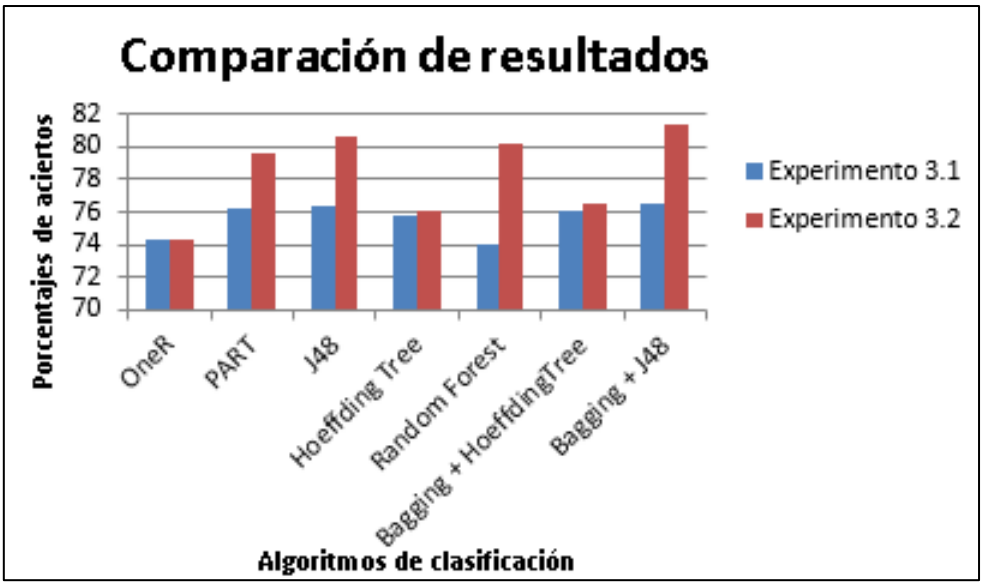



Figura 42: Comparación de resultados tras la selección de atributos.

Esta gráfica nos muestra que la selección de atributos realizada mediante el método 2 es mejor que la del método 1 ya que obtenemos un porcentaje de acierto mayor o igual para cada uno de los algoritmos clasificadores que utilizamos.

Esta mejora en los resultados se debe a que este método de selección de atributos evalúa los subconjuntos de atributos más relevantes mientras que el otro sólo los evalúa individualmente.

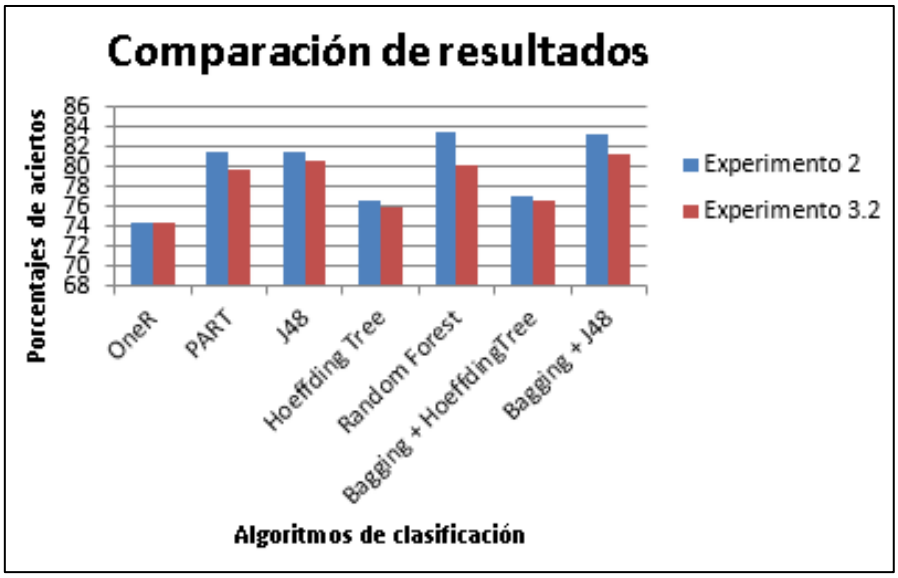



Figura 43: Comparación de resultados con entre el experimento 2 y 3.2.

Como podemos ver en esta otra comparativa, la selección de atributos mediante el método 2 no nos permite conseguir mejor porcentaje de aciertos que los obtenidos con

el experimento 2. Este hecho puede venir provocado por una de las desventajas de este método de selección de atributos ya que durante el proceso de selección puede eliminar atributos que por sí solos no están correlacionados con la clase, pero con otro atributo sí que lo están.

La conclusión que alcanzamos tras haber utilizado este método de selección de atributos es que con dicho método obtenemos mejores resultados que con el primer método de selección de atributos pero no llegamos a superar los resultados obtenidos en el experimento 2. Por ello, centrándonos en el algoritmo que consideramos el mejor hasta ahora, meta-clasificador *Bagging* con clasificador base *C4.5 (J48)*, decidimos que esta selección de atributos no es la más idónea ya que no conseguimos mejorar el porcentaje de aciertos de una manera considerable.

El tercer y último método de selección de atributos no será aplicado a nuestro dataset ya que necesitamos un supercomputador para poder ejecutarlo. Esto se debe a que este método de selección evalúa los subconjuntos de atributos ejecutando un algoritmo de minería de datos concreto sobre el conjunto de datos, lo cual hace que sea muy lento.

# 6.CAPÍTULO VI: IMPLEMENTACIÓN DE UNA INTERFAZ PARA VISUALIZAR MODELOS

En este capítulo explicaremos como se realizó la interfaz la cual tiene la finalidad de visualizar la predicción del estado actual y futuro del flujo del tráfico en Oporto. De esta manera sabremos si los resultados que obtenemos son coherentes.

## 6.1. OBJETIVOS

El objetivo de esta interfaz es representar el conjunto de celdas en las que hemos dividido el mapa de Oporto. De esta manera podremos visualizar e interpretar el mejor modelo construido en Weka para predicción del flujo del tráfico en Oporto.

La aplicación también contiene una especie de mapa cuya funcionalidad es mostrarnos que parte representa cada una las celdas en la cual estamos diviendo el mapa de Oporto.

## 6.2. DISEÑO

Nuestra interfaz estará compuesta por un gestor de pestañas el cual contendrá dos pestañas estáticas, una para predecir el estado actual del flujo del tráfico y otro para predecir el estado futuro del flujo del tráfico. La primera pestaña contendrá los siguientes componentes:

- Un *JLabel* para dar información que en el campo contiguo se introduce una fecha.
- Un *JTextField* donde tenemos que introducir una fecha. Dicha fecha debe ser de la siguiente manera: Día de la semana hora digital. Por ejemplo: Lunes 10.
- Un *JButton* el cual tiene la función de que cuando lo pulsemos se calcule el flujo del tráfico de Oporto para esa fecha dada.
- Un *JPanel* el cual contiene un conjunto de botones que representan cada una de las celdas en las que hemos dividido el mapa de Oporto.
- Un *JPanel* donde almacenamos una leyenda con los diferentes colores que se pueden pintar cada uno de los botones que representan las celdas. Existen tres

tipos de colores: rojo para indicar que el flujo de tráfico para esa celda está cogestionado, naranja para indicar que está sincronizado y verde para mostrar que está libre. Estos tres colores los representamos en la leyenda con tres botones y cada uno de ellos tiene al lado un *JLabel* para indicar que significa ese color.

- Un *JMapViewer* el cual utilizamos para mostrar que parte de la zona de Oporto representa una celda en concreto.

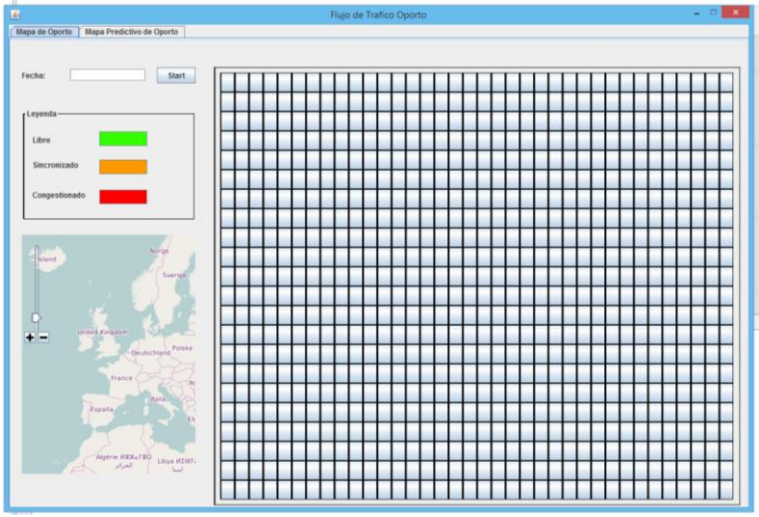



Figura 44: Pestaña para predecir el estado actual del flujo del tráfico.

La segunda pestaña la utilizamos para predecir el estado futuro del flujo del tráfico y contiene únicamente dos componentes:

- El *JPanel* que nos muestra una leyenda como en la pestaña anterior.
- El *JPanel* el cual contiene un conjunto de botones que representan cada una de las celdas en las que hemos dividido el mapa de Oporto.

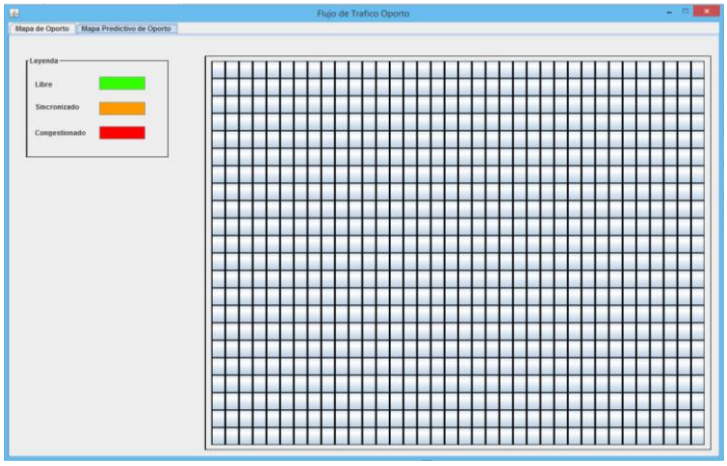



Figura 45: Pestaña para predecir el estado futuro del flujo de tráfico.

## 6.3. FUNCIONALIDADES

Existe una serie de funcionalidades en esta interfaz que nos permiten ser más intuitiva y amigable. Por ejemplo, si introducimos una fecha no correcta nos saldrá un panel indicándonos que dicha fecha no es correcta.

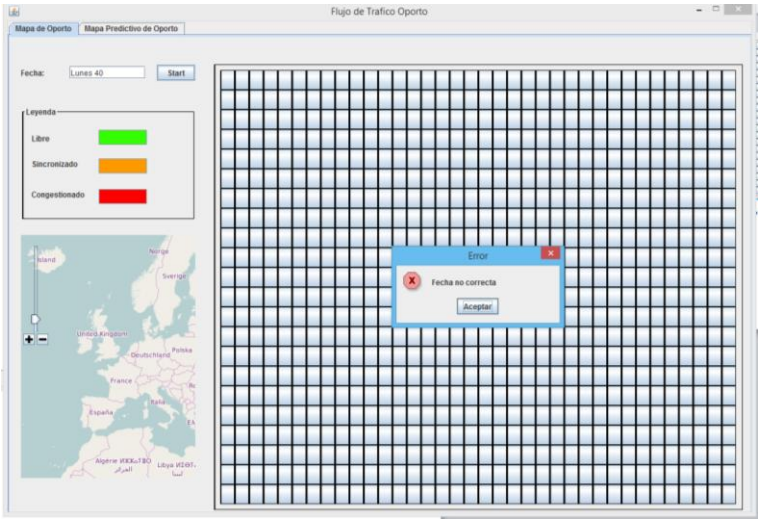



Figura 46: Introducción incorrecta de una fecha.

También tenemos un texto de información por cada uno de los botones que representan las celdas del mapa de Oporto. Este texto nos indica cuál es la longitud y latitud mínima y máxima de esa celda.

Otra de las funcionalidades que nos podemos encontrar en esta interfaz es que si pulsamos cualquier botón del panel que representan las celdas en las que dividimos el mapa de Oporto, el componente *JMapViewer* se ve modificado con el objetivo de mostrarnos que parte de Oporto es en la realidad.

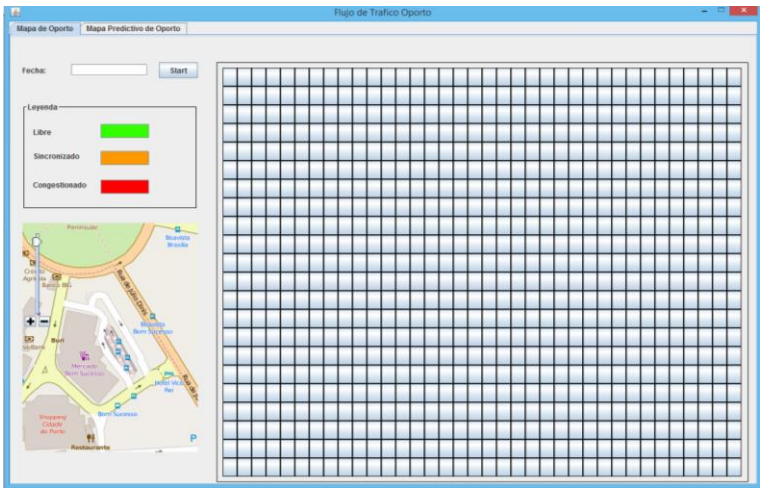



Figura 47: Visualización de una zona concreta de Oporto.

## 6.4. FUNCIONAMIENTO

Tras tener el modelo de la interfaz definido, nos centramos en su funcionamiento con el objetivo de mostrar el flujo del tráfico actual y futuro de todo el mapa de Oporto para una hora de un día concreto.

Utilizamos el modelo obtenido por el meta-clasificador Bagging con la clasificadora base J48 para predecir el flujo del tráfico en Oporto. Este modelo los almacenamos en un fichero con extensión model y los cargamos en java en un objeto del tipo Classifier mediante el uso de la librería Weka.

Para concluir el funcionamiento de esta interfaz activamos el evento que cuando se cliquee el botón Start se comience a predecir el estado actual y futuro del flujo de tráfico. Al activar este evento, lo primero que se realiza es comprobar que el campo fecha contiene una fecha correcta ya que sino no se podrá predecir el flujo de tráfico. En el caso de que la fecha sea correcta se carga la instancia de cada celda que corresponde del dataset y se clasifica. Cuando ya sabemos cómo se encuentra el tráfico tanto para el estado actual como el futuro modificamos el color de la correspondiendo celda. La gestión de este evento se puede ver en el proyecto en el método buttonStartMouseClicked de la clase VisualizeFlowTrafficOporto.

## 6.5. VISUALIZACIÓN DE PREDICCIONES DEL FLUJO DE TRÁFICO

En este apartado visualizaremos varias predicciones del flujo de tráfico en Oporto para comprobar que los resultados que obtenemos son coherentes. Para ello visualizaremos la predicción de las ocho de la mañana de un martes y a las 2 de madrugada del miércoles ya que debería haber un cambio drástico en el flujo del tráfico.

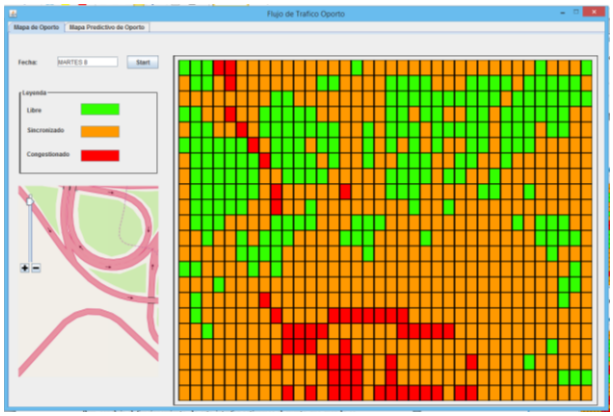



Figura 48: Estado actual del flujo de tráfico a las 8:00 de un martes.

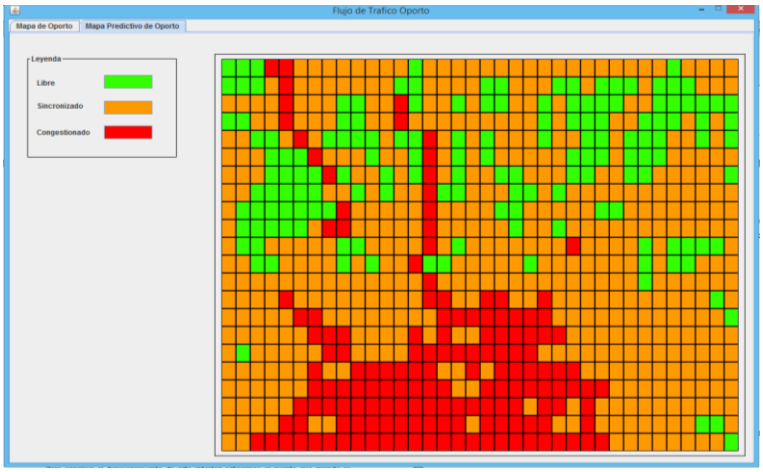



Figura 49: Estado futuro del flujo de tráfico a las 8:00 horas de un martes.

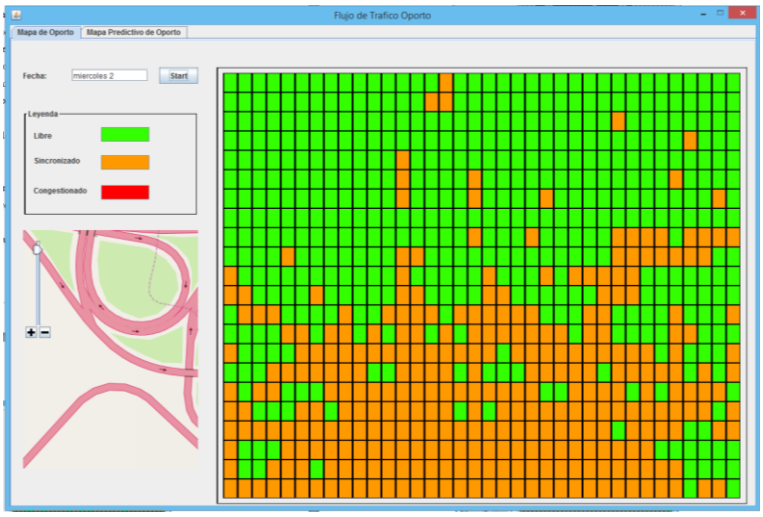



Figura 50: Estado actual del flujo de tráfico para las 2:00 de un miércoles.

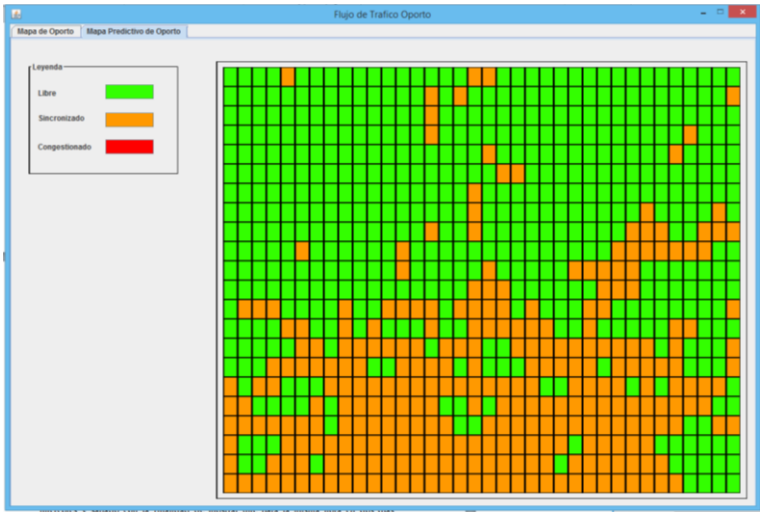



Figura 51: Estado futuro del flujo de tráfico para las 2:00 de un miércoles.

Además visualizaremos el flujo de tráfico para las 10 de la noche para los días miércoles y sábado con la finalidad de mostrar que para la misma hora en dos días diferentes el flujo del tráfico varía notablemente.

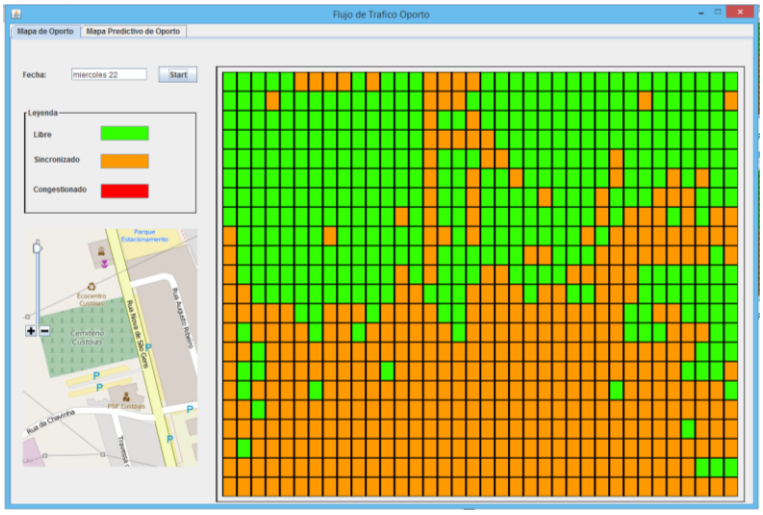



Figura 52: Estado actual del flujo de tráfico para las 22:00 de un miércoles.


Figura 53: Estado futuro del flujo de tráfico para las 22:00 de un miércoles.


Figura 54: Estado actual del flujo de tráfico para las 22:00 sábado.

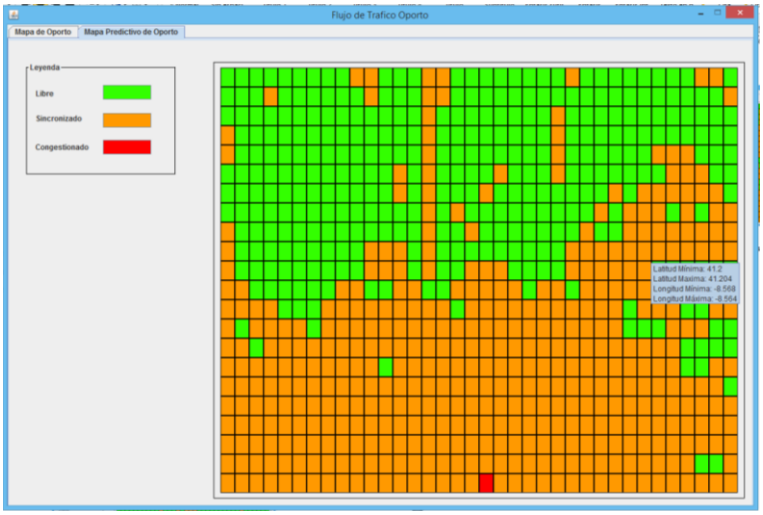



Figura 55: Estado futuro del flujo de tráfico para las 22:00 sábado.

# 7. CAPÍTULO VII: CONCLUSIÓN Y TRABAJOS FUTUROS

## 7.1. CONCLUSIÓN

Tras haber estudiado el proceso KDD y en mayor profundidad la fase denominada minería de datos. Hemos realizado un proyecto fin de grado, el cual pretende predecir el flujo del tráfico con dicha fase, de la ciudad de Oporto en este caso, con el fin de disminuir la congestión, ya que esta produce una serie de desventajas en el medioambiente y en la sociedad. Desventajas tales como: la contaminación, problemas físicos, problemas mentales en las personas, el deterioro de los vehículos, etc.

En cuanto a la aplicación del proceso KDD, la fase denominada preparación de los datos fue la más costosa, ya que tuvimos que dividir el mapa de Oporto en celdas con la finalidad de distribuir cada coordenada de GPS en una celda específica. Además, almacenamos la fecha, contando con el día y hora específica, a la que se había registrado cada coordenada de GPS en el dispositivo y el identificador del taxi que realizó el trayecto.

Por cada celda obtenemos 7·24 instancias, cada instancia corresponde a una hora de un día a la semana. Dichas instancias contienen una serie de atributos, los cuales serán utilizados por diferentes técnicas de minería de datos, en concreto los algoritmos de clasificación, para predecir el flujo del tráfico.

Con el software Weka se realizaron una serie de experimentos en la fase de minería de datos con la finalidad de construir el mejor modelo posible. En primer lugar se ejecutó los siete algoritmos de clasificación para saber cuál era la situación en la que nos encontramos. En el segundo experimento ajustamos los parámetros de cada uno de los algoritmos de clasificación para mejorar la precisión en el porcentaje de aciertos. Pero este experimento no funcionó como esperábamos, ya que el porcentaje de aciertos no aumentó más del 1% en ninguno de los algoritmos utilizados.

En cuanto al tercer experimento, se realizó una selección de atributos con el fin de mejorar el porcentaje de aciertos, y en este caso los resultados tampoco resultaron favorables, e incluso en algunos casos empeoró considerablemente.

Tras haber obtenido los distintos resultados comentados, decidimos que el mejor modelo era construido por el meta-clasificador *Bagging* con un clasificador base *C4.5 (J48)* ajustando sus parámetros de tal forma que conseguíamos un porcentaje de aciertos del 83,35%.

Por último, nos queda hablar de la construcción de una interfaz, la cual se utiliza para evaluar el modelo construido y realizar una interpretación de sus resultados de manera visual sobre el mapa de Oporto.

Con la elaboración del proyecto se ha aprendido a seguir la metodología del proceso de KDD en un trabajo de investigación. Cabe resaltar la necesidad de retornar e iterar continuamente entre las distintas fases, realizando diversas modificaciones para obtener un mayor acierto en la predicción del flujo del tráfico, cumpliendo así nuestro objetivo.

Como conclusión final hemos de declarar que nos encontramos satisfechos con el trabajo realizado, ya que cumple el objetivo principal propuesto, el cual era construir un modelo que predijera el flujo del tráfico para la ciudad de Oporto. Además, cabe destacar que al tratarse de un trabajo de investigación estamos contentos con los resultados obtenidos, ya que nuestro modelo obtiene un porcentaje de acierto de más del 80%.

## 7.2. TRABAJOS FUTUROS

Muchas son las propuestas que se pueden plantear para la continuación o mejora de este proyecto fin de grado, a continuación se exponen algunas de las mismas:

- Aplicación para obtener los trayectos óptimos haciendo uso de este modelo. De esta manera desde los servicios públicos tales como los bomberos o la policía hasta un ciudadano en Oporto podrá llegar a su destino evitando la congestión vehicular, en el caso de que se pudiera.
- Ejecutar el tercer método de selección de atributos que nombramos en el capítulo 5 en un supercomputador. Mediante el uso de este método de selección podríamos obtener un modelo con mejor porcentaje de acierto ya que con este método se consigue el mejor subconjunto de atributos para un algoritmo concreto.
- Modificar la preparación de los datos con el fin de que cada celda sea "alimentada" por la información de las celdas vecinas. De esta forma tendríamos

un dataset que podría dar mejores resultados ya que para cada celda a una hora concreta, sabríamos cuales son los valores de los atributos de sus celdas vecinas a esa misma hora, por tanto, tendríamos más información para clasificar esa instancia.

# BIBLIOGRAFÍA

## REFERENCIAS BIBLIOGRÁFICAS


Corso, C. (2009) Aplicación de algoritmos de clasificación supervisada usando Weka. *Universidad tecnológica nacional, Facultad Regional Córdoba.* Recuperado de: http://www.investigacion.frc.utn.edu.ar/labsis/Publicaciones/congresos_labsis/cynthia/CNIT_2009_Aplicacion_Algoritmos_Weka.pdf

Fayyad, U. M., Piatetsky-Shapiro, G., and Smyth, P. (1996). In U. M. Fayyad, G. Piatetsky-Shapiro, & P. Smyth, From data mining to knowledge discovery in databases. *Al Magazine*, Vol. 17, número 3. (pp. 37-54). Recuperado de: https://www.aaai.org/ojs/index.php/aimagazine/article/viewFile/1230/1131

Hernández, J., Ramírez, M. y Ferri, C. (2004). *Introducción a la minería de datos*. España: Pearson.

Lara, J. (2014). *Minería de datos.* Universidad a distancia de Madrid: CEF.

Witten, I., Frank, E. and Hall M. (2011) *Data mining. Practical machine learning tools and techniques.* Estados Unidos: Morgan Kaufmann Publishers.


## REFERENCIAS WEB


Conjunto de datos de entrenamiento:

https://archive.ics.uci.edu/ml/datasets/Taxi+Service+Trajectory+-+Prediction+Challenge,+ECML+PKDD+2015

Obtención de un mapa: http://www.openstreetmap.org/export#map=14/41.1500/-8.6306&layers=D

Página oficial de Weka: http://www.cs.waikato.ac.nz/ml/weka/