Facultade de Informática
UNIVERSIDADE DA CORUÑA

TRABALLO FIN DE GRAO
GRAO EN ENXEÑARÍA INFORMÁTICA
MENCIÓN EN TECNOLOXÍAS DA INFORMACIÓN

Técnicas de aprendizaje máquina para la optimización de características de una red de datos

Estudante: Manuel Orois García
Dirección: Víctor Manuel Carneiro Díaz

A Coruña, setembro de 2020.

Os lo dedico Papá, Mamá y Olalla por ayudarme a llegar hasta aquí.

Agradecimientos

Quiero darle las gracias a mi profesor, Víctor Carneiro Díaz, por toda la ayuda que he recibido a lo largo de este trabajo.

A toda mi familia por apoyarme para ser capaz de llegar hasta donde estoy.

A Olalla por aguantarme todos los días y estar siempre a mi lado.

Resumen

Dentro del campo de la Inteligencia Artificial, cuya finalidad es proporcionarle a los sistemas la capacidad de adquirir conocimientos por sí mismos para resolver problemas, nos encontramos con un subcampo como es el *Machine Learning*. Este tipo de *Aprendizaje Automático*, aplicado a las redes, permite realizar una mejor gestión así como un análisis de estas.

Este proyecto se centra en el estudio de diversas técnicas de ML además de realizar una comparativa de los resultados obtenidos por cada una. Todo esto sirve para encontrar soluciones de optimización u operación de redes eficientes, como podría ser el retraso en el tráfico de extremo a extremo. Estas técnicas de aprendizaje automático son capaces de crear modelos de red ligeros con una buena precisión, por lo que facilitarían mucho el trabajo a los operadores de red actuales.

Para ser capaces de conseguir esto, llevamos a cabo varias fases que incluyen la selección y posterior análisis de un dataset público, así como el estudio, implementación y aplicación de diferentes técnicas de machine learning (Random Forest, K-NN y Redes Neuronales)

A partir de diversas pruebas y una buena preparación previa de los datos, se consiguieron los resultados óptimos para cada una de las técnicas mencionadas anteriormente. A partir de estos hemos comprendido que el que mejor resultados arrojó fue K-NN seguido muy de cerca por Random Forest. En cuanto a la Redes Neuronales, cabe decir que obtuvimos unos resultados buenos pero, en comparación con las otras técnicas, todavía hay mucho que mejorar.

Abstract

In the field of the Artificial Intelligence, which main purpose is to make the sistems able to adquire knowledge by themselves to solve problems, we find ourselves with the subfield as machine learning. This kind of authomatic learning, applied to webs, let you make a better management and analysis of them.

This project focus on studying several techniques of ML, moreover to make a comparative of the results obtained by each one. All this is useful to find optimization or operation solutions of efficient webs, as it could be the delay on the extreme-to-extreme traffic. These techniques of authomatic learning are able to create models of light web with good accuracy, which would make the job of the current web operators easier.

To be able to do this, we make several stages, that include the selection and the next analysis of a public dataset, as the studying, implementation and application of the different machine learning techniques. (Random forest, K-NN and neuronal networks.)

To start with several tries, and a good previous preparation of the data, there had been optimal results for each of the techniques previously named. From this, we had understood that the best of the results was of the K-NN, followed close by Random Forest. Speaking of neuronal webs, we could say we had good results, but comparing with the other techniques there is a lot of improvement looking forward.

Palabras clave:

- Machine Learning
- Inteligencia Artificial
- K-NN
- Random Forest
- Redes Neuronales Recurrentes
- Regresión

Keywords:

- Machine Learning
- Artificial Intelligence
- K-NN
- Random Forest
- Recurrent neural network
- Regression

Índice general

Índice de figuras

Índice de tablas

Listings

Capítulo 1

Introducción

El aprendizaje automático (AA), también conocido como machine learning (ML), consiste en un conjunto de técnicas de la informática que se relacionan con la inteligencia artificial (IA), y que sirven para crear sistemas que pueden resolver problemas por sí solos sin que las personas tengan que intervenir. Cabe destacar que sobre estas, el aprendizaje automático puede utilizarse para mejorar los análisis, la gestión y la seguridad, entre otros muchos factores, de las redes [1].

Actualmente, es mucho mejor construir un sistema (nueva generación) que sea capaz de conocer y manejar por sí mismo los problemas que los expertos en redes presentan que continuar con los antiguos sistemas basados en reglas. Estos problemas son el mantenimiento y actualización en sistemas de administración de redes (antigua generación) que conlleva tener a varios expertos al cargo. La clave radica en el mantenimiento, ya que a medida que se encuentren nuevos problemas y soluciones, dicho sistema aprenderá los síntomas y las acciones resultantes a tomar sin ser necesaria la intervención humana, lo cual conllevaría una gran reducción del esfuerzo.

La aplicación de algoritmos de ML a la gestión del tráfico de las redes permite reducir considerablemente el trabajo que tiene que llevar a cabo el personal de Tecnologías de la Información (TI), a pesar de que dichos algoritmos no sean capaces de comprender la repercusión sobre el negocio en sí. En los últimos años se han logrado muchos avances en el software de administración de redes mediante la aplicación de técnicas de aprendizaje automático [2].

1.1 Objetivos

La finalidad del trabajo es construir modelos de red livianos para reducir considerablemente la carga que tienen los operadores de red mediante el uso de varias técnicas de ML como son Random Forest, K-NN (k-nearest neighbors) y RNN (redes neuronales recurrentes). Concretamente vamos a elegir un dataset público, realizar un análisis y tratamiento previo

de los datos, construir los modelos con cada una de las técnicas mencionadas y comparar resultados para ver qué modelo es el que arroja una calidad superior frente al resto. Para ser capaces de evaluar la calidad vamos a utilizar las métricas más usadas para realizar este tipo de mediciones (MSE, RMSE, MAE y R^2), las cuales serán explicadas en detalle en secciones posteriores del trabajo.

1.2 Estructura de la memoria

La presente memoria ha sido estructurada en los siguientes capítulos:

- Capítulo 1. Introducción.
- Capítulo 2. Planificación del proyecto.
  - Abarca las tareas realizadas en cada iteración.
- Capítulo 3. Metodología.
  - Constituye la metodología empleada.
- Capítulo 4. Entender el negocio.
  - Estudio de las técnicas de machine learning aplicadas al tráfico de red (Random Forest, K-NN y Redes Neuronales)
- Capítulo 5. Fundamentos tecnológicos.
  - Análisis de la herramienta y librerías empleadas.
- Capítulo 6. Comprensión de los datos.
  - Elección y análisis del dataset público. También incluye la exploración de este para conseguir una mejor comprensión de los datos y así poder obtener una calidad mayor.
- Capítulo 7. Preparación de los datos.
  - Incluye la limpieza, selección y transformación de los datos. Es una de las fases más importantes del proyecto ya que los resultados que obtengamos dependen en gran medida de la calidad de nuestro conjunto de datos.

- Capítulo 8. Modelado.
  - Análisis de los parámetros más influyentes en cuanto a los resultados del modelo. Además se llevará a cabo el entrenamiento del modelo, su validación y la experimentación pertinente.
- Capítulo 9. Evaluación.
  - Se evaluarán los modelos construidos con un conjunto de datos independiente al de entrenamiento. Es aquí donde realizaremos la comparativa final entre las técnicas utilizadas.
- Capítulo 10. Conclusiones.
  - Objetivos alcanzados y lecciones aprendidas.
  - Líneas futuras

Capítulo 2

Planificación del proyecto

Esta sección abarca la planificación del proyecto tomando como referencia la metodología CRISP-DM [3]. Se ha decidido usar dicha metodología ya que especifica las tareas a realizar en cada una de las fases del proyecto, asignando las tareas concretas y definiendo lo que es deseable obtener en cada fase. En este caso no se ha seguido de una manera estricta pero si se han tenido en cuenta las principales características de esta. En resumen, es un modelo de desarrollo de software que se caracteriza por:

- No es una metodología propietaria, por lo que cualquiera puede usarla fácilmente.
- No es dependiente de tecnologías o herramientas.
- Proporciona una descripción normalizada del ciclo de vida del proyecto.
- Nos permite movernos entre diferentes fases del ciclo de vida si así fuera necesario.

A continuación vamos a tratar todas las iteraciones que se llevan a cabo en el proyecto, así como especificar las tareas que engloba cada una y los perfiles necesarios para llevarla a cabo. Dicho perfiles son los que se pueden observar a continuación:

- Product Manager: es el encargado de definir la estrategia que se va a llevar a cabo en el proyecto, así como de planificar y ejecutar cada una de las etapas establecidas.
- Analista: se dedica a la obtención y diseño de los variados algoritmos. Tiene una amplia visión del negocio.
- Desarrollador: va a dedicarse a construir los diversos modelos que serán usados.
- Ingeniero de datos: se dedica a transformar los datos crudos que serán usados por los algoritmos de aprendizaje automático.

Historias de usuario	Tareas	Perfil	Estimaciones
Entender el negocio	Estudio de las técnicas de Machine Learning aplicadas a las características de tráfico de red	Analista	10
	Estudio de Random Forest	Analista	10
	Estudio de K-NN	Analista	10
	Estudio de Redes Neuronales	Analista	10
		Total	40

Tabla 2.1: Tareas de la iteración 0.

Historias de usuario	Tareas	Perfil	Estimaciones
Metodología y herramientas	Análisis de la metodología	Analista	10
	Estudio herramienta R	Desarrollador	10
		Total	20

Tabla 2.2: Tareas de la iteración 1.

Historias de usuario	Tareas	Perfil	Estimaciones
Modelo Random Forest	Algoritmo en R	Desarrollador	5
	Preprocesado de datos	Ingeniero de datos	15
	Entender los datos	Ingeniero de datos	10
	Preparar los datos	Ingeniero de datos	15
	Modelar	Desarrollador	15
	Experimentación	Desarrollador	10
	Evaluación y comparativa entre modelos Random Forest	Desarrollador	10
		Total	80

Tabla 2.3: Tareas de la iteración 2.

Historias de usuario	Tareas	Perfil	Estimaciones
Modelo K-NN	Algoritmo en R	Desarrollador	5
	Preprocesado de datos	Ingeniero de datos	15
	Entender los datos	Ingeniero de datos	10
	Preparar los datos	Ingeniero de datos	15
	Modelar	Desarrollador	15
	Experimentación	Desarrollador	10
	Evaluación y comparativa entre modelos K-NN	Desarrollador	10
		Total	80

Tabla 2.4: Tareas de la iteración 3.

Historias de usuario	Tareas	Perfil	Estimaciones
Modelo RouteNet	Estudio del modelo	Analista	15
	Preparación de los datos	Ingeniero de datos	5
	Evaluar y comparativa con Random Forest	Desarrollador	5
		Total	25

Tabla 2.5: Tareas de la iteración 4.

Historias de usuario	Tareas	Perfil	Estimaciones
Elaboración de la documentación	Resumen	Analista	2
	Introducción	Analista	6
	Planificación del proyecto	Product Manager	10
	Metodología	Analista	4
	Entender el negocio	Analista	8
	Fundamentos tecnológicos	Desarrollador	5
	Comprensión de los datos	Ingeniero de datos	5
	Preparación de los datos	Ingeniero de datos	5
	Modelado	Desarrollador	20
	Evaluación	Desarrollador	5
	Conclusiones	Analista	5
		Total	75

Tabla 2.6: Tareas de la iteración 5.

2.1 Estimaciones económicas

	Precio	Horas	Total
Analista	22€/hora	90	2085€
Desarrollador	27€/hora	125	3375€
Ingeniero de datos	20€/hora	95	1900€
Product Manager	35€/hora	10	350€
Equipos informáticos	0,40€/hora	320	128€
		Subtotal	7838€
		IVA 21%	1645,98€
		Total	**9483,98€**

Tabla 2.7: Estimaciones económicas

Capítulo 3

Metodología

A la hora de abordar un proyecto utilizando técnicas de ML, contamos con diversas metodologías para llevarlo a cabo con éxito. En general, todas tienen como objetivo definir las fases necesarias para realizar un proyecto de manera eficaz y exitosa. En este caso concreto nos hemos decantado por la utilización de CRISP-DM (Cross Industry Standard Process for Data Mining) [4], ya que especifica concretamente las fases del proyecto, así como lo que es requerido obtener en cada una de ellas.

Si lo vemos desde un punto de vista más global, CRISP-DM se acerca al concepto de un proyecto real. Sabiendo todo esto y ya que todas sus fases se adecúan perfectamente a nuestro proyecto, nos hemos decantado por utilizar esta metodología para el proyecto.

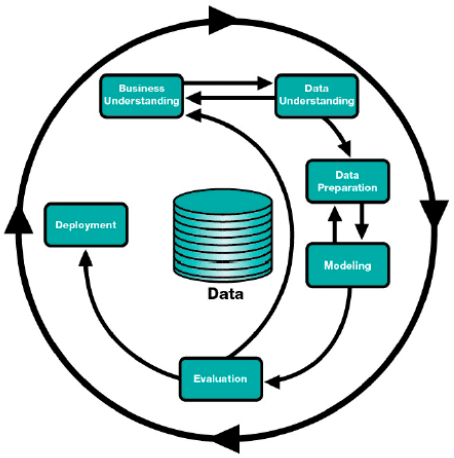

Figura 3.1: Diagrama CRISP-DM

Fuente: ftp.software.ibm.com

3.1 CRISP-DM

CRISP–DM es la guía de referencia más ampliamente utilizada en el desarrollo de proyectos de minería de datos ya que permite mostrar el ciclo de vida de estos. Dicha metodología es tan utilizada gracias a su carácter genérico, indistintamente del tipo de herramienta que se utilice para la elaboración del proyecto y por ser distribuida de manera gratuita.[5]

Está dividido en 4 niveles de abstracción organizados de forma jerárquica (**figura** 3.2) en tareas que van desde el nivel más general, hasta los casos más específicos y organiza el desarrollo de un proyecto de DM (data mining), en una serie de seis fases (**figura** 3.3).

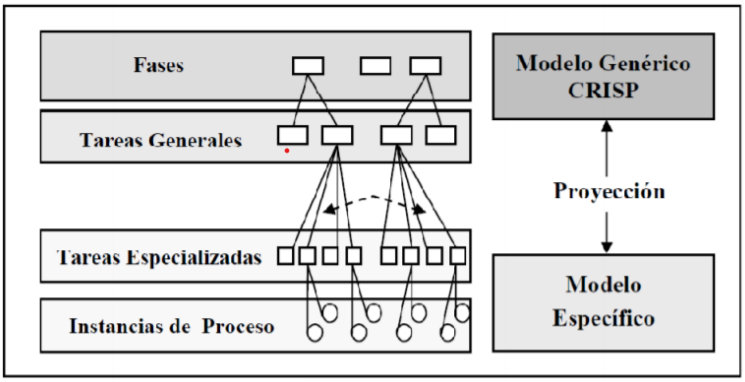

Figura 3.2: Niveles de abstracción en CRISP-DM

Fuente: ftp.software.ibm.com

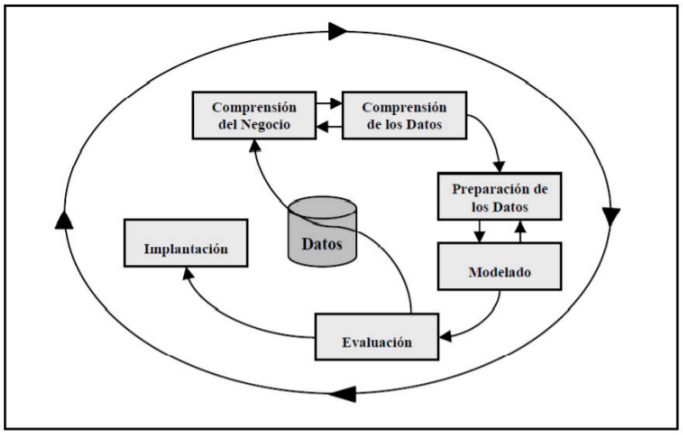

Figura 3.3: Fases en CRISP-DM

Fuente: ftp.software.ibm.com

Ahora se va a describir de una manera breve cada una de las fases:

- **Entender el negocio**: se trata de entender tanto los objetivos como los requisitos del proyecto desde el punto de vista del negocio. Una vez entendido seremos capaces de marcar los requisitos y poner objetivos claros de lo que se pretende alcanzar.
- **Comprensión de los datos**: abarca tanto la recopilación inicial de los datos como el estudio de estos, de manera que seamos capaces de comprenderlos y de obtener conocimiento a partir de dichos datos.
- **Preparación de los datos**: engloba todos los procesos necesarios para crear el conjunto de datos utilizado para construir el modelo. Todo esto incluye la limpieza, transformación y selección de los datos en la herramienta de modelación.
- **Modelado**: es la etapa en la que se llevan a cabo las tareas necesarias para crear el modelo, las cuales abarcan la hiperparametrización, el entrenamiento y la validación. El objetivo de esta fase es conseguir un modelo que cumpla con los requisitos del proyecto.
- **Evaluación**: una vez construido el modelo es importante saber lo bueno que es y para ello hemos de evaluarlo, por lo que en esta fase se lleva a cabo esta tarea para saber si el modelo alcanzó correctamente los objetivos establecidos.
- **Despliegue**: la fase de despliegue es en la que se pone en marcha el proyecto dentro de la organización. Incluye la realización de una adecuada documentación, presentación de los resultados y mantenimiento.

Por último, en este capítulo resalta el hecho de que las fases en CRISP-DM no son rígidas, es decir, se puede saltar entre fases en cualquier momento. El resultado de cada fase determina lo que hay que hacer a continuación. Al tratarse de un proceso iterativo, las flechas solo indican las dependencias más importantes y frecuentes (**figura** 3.3).

Capítulo 4

Entender el negocio

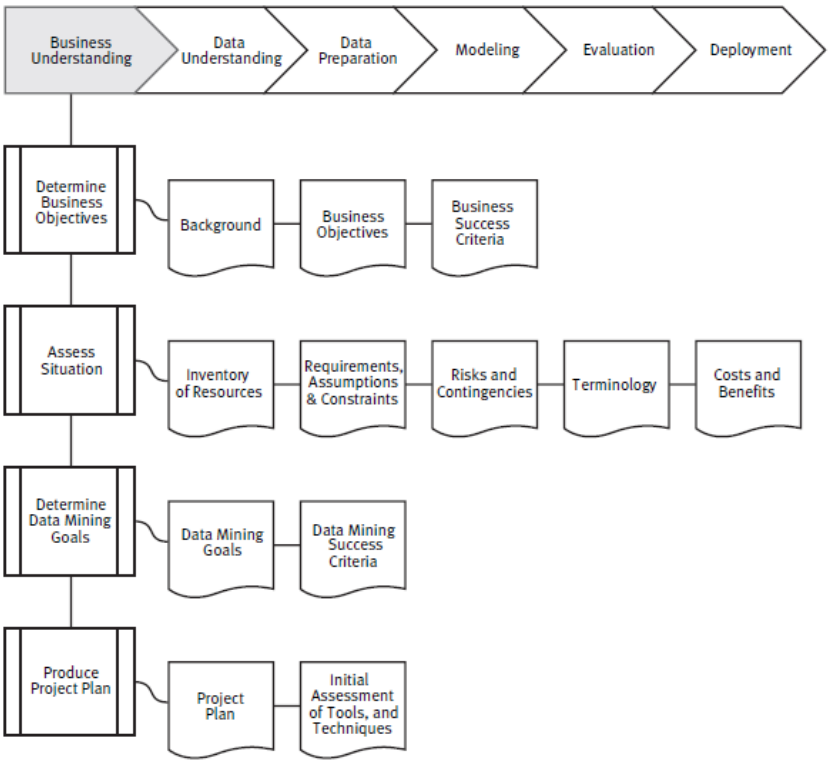

Figura 4.1: CRISP-DM (1)

Fuente: ftp.software.ibm.com

El Machine Learning, en adelante ML, es una disciplina científica del ámbito de la Inteligencia Artificial que crea sistemas que aprenden automáticamente para que sean capaces de resolver problemas. Quien realmente aprende es un algoritmo que revisa los datos y es capaz de predecir comportamientos futuros. Esto implica que los sistemas se mejoran de forma

autónoma con el tiempo, sin intervención humana.

Si queremos que una máquina pueda adquirir la capacidad de aprender por sí misma, esta debe ser entrenada para que a partir de los datos con los que se entrenó sea capaz de tomar sus propias decisiones. Esta es la manera por la que es posible que los ordenadores den soluciones a determinados problemas sin que el ser humano necesite intervenir.

Los algoritmos de ML son capaces de extraer patrones de comportamiento de grandes conjuntos de datos. Una vez hecho podrán generar un modelo capaz de encontrar las respuestas adecuadas a un problema específico.

4.1 Tareas Machine Learning

En el aprendizaje automático nos encontramos con tareas cuya función es dividir un problemas complejo en tareas asequibles. Dependiendo del problema que estamos tratando de resolver será mejor utilizar unas tareas u otras.

Las tareas más habituales que se suelen usar en ML y las más ligadas con nuestro proyecto son las siguientes:

- **Clasificación**: su finalidad es predecir a qué clase pertenece un conjunto de datos.
- **Regresión**: predicen o estiman el valor numérico de alguna de sus variables.
- **Clustering**: identifican grupos de elementos en un conjunto de datos teniendo en cuenta una medida de similitud.

4.2 Tipos de aprendizaje Machine Learning

Las diversas técnicas que hay de Machine Learning suelen dedicarse a la automatización de la identificación de patrones a partir de los datos proporcionados. Con esto lo que se pretende es ir generando y ajustando, a partir de los diferentes algoritmos existentes, un modelo capaz de resolver un problema específico. El problema que se pretende resolver es lo que dictaminará el mejor algoritmo para este proceso [6].

Podemos abordar un problema desde diferentes puntos y por lo tanto se debe investigar detenidamente cuál será la mejor estrategia. Podemos distinguir varias categorías de aprendizaje automático:

- **Aprendizaje supervisado:** en los algoritmos de aprendizaje supervisado se genera un modelo predictivo, basado en datos de entrada y salida. La palabra clave "supervisado" viene de la idea de tener un conjunto de datos previamente etiquetado y clasificado, es

decir, tener un conjunto de muestra, el cual ya se sabe a qué grupo, valor o categoría pertenecen los ejemplos. Con este grupo de datos que llamamos datos de entrenamiento, se realiza el ajuste al modelo inicial planteado. De esta forma es como el algoritmo va "aprendiendo" a clasificar las muestras de entrada comparando el resultado del modelo, y la etiqueta real de la muestra, realizando las compensaciones respectivas al modelo de acuerdo a cada error en la estimación del resultado. Por último una vez que el modelo se considere ajustado se aplicará al conjunto de pruebas para ver la exactitud de la predicción realizada por este.

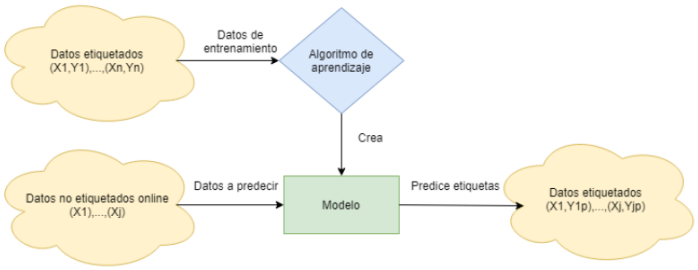

Figura 4.2: Aprendizaje supervisado

Fuente: aspgems.com

- **Aprendizaje no supervisado:** los algoritmos de aprendizaje no supervisado trabajan de forma muy similar a los supervisados, con la diferencia de que estos sólo ajustan su modelo predictivo tomando en cuenta los datos de entrada, sin importar los de salida. Es decir, a diferencia del supervisado, los datos de entrada no están clasificados ni etiquetados y no son necesarias estas características para entrenar el modelo.

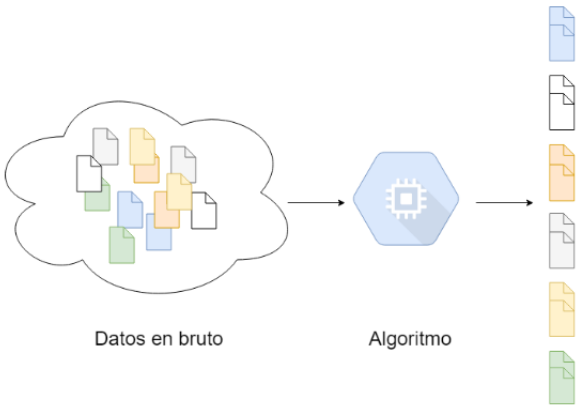

Figura 4.3: Aprendizaje no supervisado

Fuente: aspgems.com

- **Aprendizaje por refuerzo:** los algoritmos de aprendizaje por refuerzo definen modelos y funciones enfocadas en maximizar una medida de "recompensas", basados en "acciones" y al ambiente en el que el agente inteligente se desempeñará. Es decir, son algoritmos que se basan en el modelo de prueba o error que les permite analizar y aprender del entorno en el que se encuentra realizando determinadas acciones mediante las cuales será capaz de llegar al objetivo final. A lo largo de todo el proceso se recibirán recompensas o penalizaciones en función de la acción realizada hasta conseguir la mejor puntuación posible y resolver el problema.

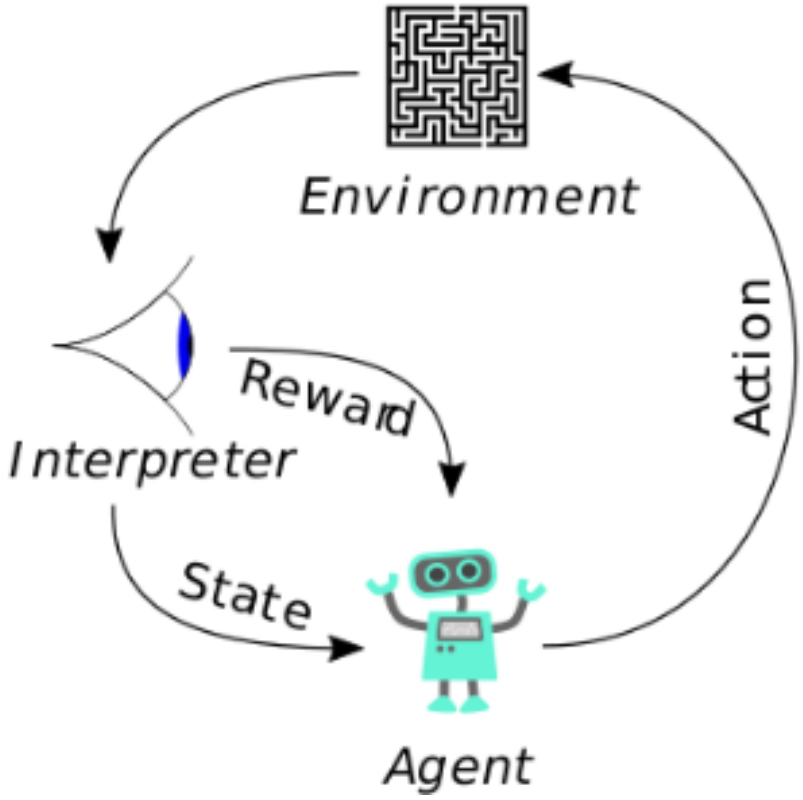

Figura 4.4: Aprendizaje por refuerzo

Fuente: es.wikipedia.org

- **Aprendizaje profundo:** utiliza redes neuronales que replican el funcionamiento de las neuronas del cerebro humano. Estas redes pueden estar formadas por miles o millones de nodos de procesamiento interconectados. A todo esto se le llama aprendizaje profundo porque se lleva a cabo a través de un gran número de capas. En cada una de estas se realizan continuos ajustes hasta alcanzar un determinado punto final. Lo normal es que se use este tipo de aprendizaje en conjuntos de datos que no están ni estructurados ni etiquetados. También cabe destacar que cuanta mayor sea la complejidad del problema mayor cantidad de capas ocultas necesitaremos en la red neuronal para poder resolverlo.

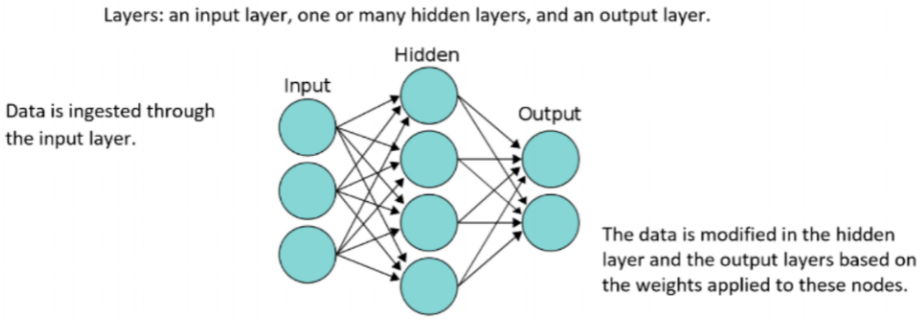

Figura 4.5: Aprendizaje profundo

Fuente: www.researchgate.net

De manera general, el aprendizaje supervisado se utiliza en problemas de regresión y predicción estadística mientras que el no supervisado lo normal es utilizarlo en clasificación de variables.

4.3 Modelos de aprendizaje

En esta sección vamos a hablar sobre algunos modelos de aprendizaje, concretamente sobre KNN, Random Forest y Redes Neuronales que son los que vamos a usar a lo largo del desarrollo del proyecto.

4.3.1 K-NN

K nearest neighbours o en español, K vecinos más cercanos, es un algoritmo que se puede usar tanto para clasificar como para predecir, no obstante, como en este trabajo lo utilizaremos para predecir, vamos a centrar la explicación teniendo esto en cuenta. Su funcionamiento es muy sencillo, ya que va a predecir el valor que nosotros deseemos según tenga k vecinos más

cerca de uno de los valores de referencia. K-NN es un algoritmo de aprendizaje supervisado, es decir, que a partir de un juego de datos inicial su objetivo será el de predecir correctamente la variable objetivo. El juego de datos típico de este tipo de algoritmos está formado por varios atributos descriptivos y un solo atributo objetivo. En otras palabras, lo que hace de manera concreta es calcular las distancias del elemento que se quiere predecir a cada uno de los existentes, y ordena dichas distancias de menor a mayor para seleccionar el valor de la predicción. Esta será, por tanto, la de mayor frecuencia con menores distancias.

En contraste con otros algoritmos de aprendizaje supervisado, K-NN no genera un modelo fruto del aprendizaje con datos de entrenamiento, sino que el aprendizaje sucede en el mismo momento en el que se prueban los datos de test [7].

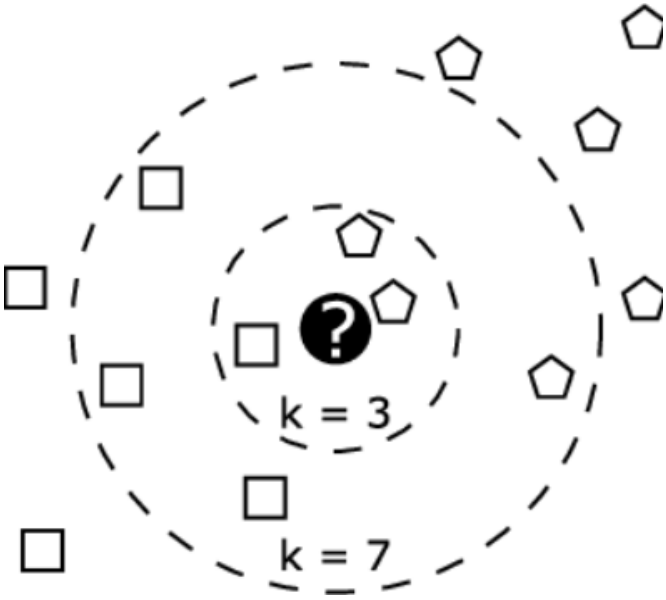

Figura 4.6: K-NN

Fuente: www.researchgate.net

Pasos de K-NN

- **1º**: calcular la distancia entre la variable objetivo y el resto de items del dataset de entrenamiento.
- **2º**: seleccionar los "k" elementos más cercanos (con menor distancia, según la función que se use)
- **3º**: realizar una "votación de mayoría" entre los k puntos. Los de una clase/etiqueta que «dominen» decidirán la predicción final.

Para decidir cual va a ser la predicción es muy importante el valor de k, pues este terminará casi de definir cual será la variable objetivo. Otra cosa reseñable es elegir valores impares de k para desempatar. No es lo mismo tomar para decidir 3 valores que 13. Esto no quiere decir que necesariamente tomar más puntos implique mejorar la precisión. Lo que es seguro es que cuantos más "puntos k", más tardará nuestro algoritmo en procesar y darnos respuesta, ya que a partir de cierto valor de k no se consigue mejorar la calidad de un modelo y el tiempo computacional necesario es muy elevado.

Algunas de las formas más populares de medir la cercanía entre puntos son la distancia Euclidiana, la distancia Coseno o la distancia Jaccard. Por último destacar que este algoritmo funciona mejor con varias características de las que tomemos datos (las columnas de nuestro dataset) [8].

4.3.2 Random Forest

Un Random Forest es un conjunto de árboles de decisión en los que cada árbol es dependiente de los valores de un vector aleatorio muestreado independientemente y con la misma distribución para todos los árboles del bosque. Estos árboles son creados por un algoritmo, de manera aleatoria, para que la correlación entre estos sea mucho menor. Por otro lado, contamos con el error de generalización, el cual nos muestra cómo de bien ha aprendido nuestro modelo de ML el patrón existente entre los datos de entrada y los resultados. Dicho error en un bosque de árboles es totalmente dependiente de cada árbol individual y de la correlación existente entre estos. Otra de las características destacables de random forest es que usa bagging, es decir, que distintos árboles ven distintas porciones de los datos. Ningún árbol ve todos los datos de entrenamiento. Esto hace que cada árbol se entrene con distintas muestras de datos para un mismo problema. De esta forma, al combinar sus resultados, unos errores se compensan con otros y tenemos una predicción que generaliza mejor [9].

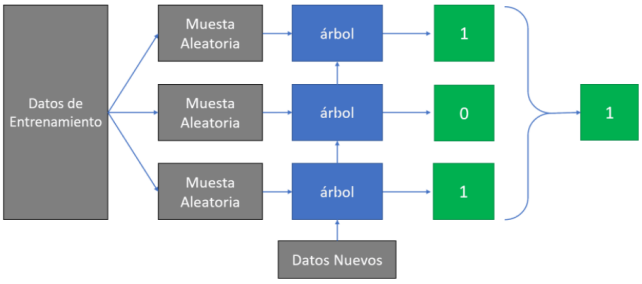

Figura 4.7: Random Forest (1)

Fuente: iartificial.net
donghwa-kim.github.io

Creación de los árboles

En Random Forest los árboles son construidos de la siguiente manera: [10]

- Dado que el número de casos en el conjunto de entrenamiento es N, una muestra de esos N casos se toma aleatoriamente pero con reemplazo. Esta muestra será el conjunto de entrenamiento para construir el árbol i.
- Si existen M variables de entrada, un número m<M se especifica tal que para cada nodo, m variables se seleccionan aleatoriamente de M. La mejor división de estos m atributos es usada para ramificar el árbol. El valor m se mantiene constante durante la generación de todo el bosque.
- Cada árbol crece hasta su máxima extensión posible y no hay proceso de poda.
- Nuevas instancias se predicen a partir de la agregación de las predicciones de los x árboles.

Una cosa de gran importancia es reducir la correlación existente entre árboles, como bien se ha comentado anteriormente. Si utilizamos muestras ligeramente distintas las unas de las otras y además utilizamos diferentes variables para el proceso de ramificación vamos a conseguir una mayor aleatoriedad en la creación de los árboles (lo cual los hace diferentes entre si), por lo que se consigue una menor correlación entre ellos.

En la siguiente figura podemos apreciar un ejemplo de bosque aleatorio. En este se llevan a cabo un promedio de las predicciones que se obtienen en cada árbol del bosque. Este promedio va a ser la variable objetivo.

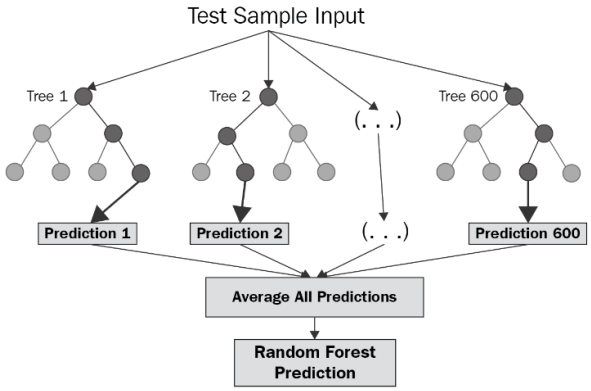

Figura 4.8: Random Forest (2)

Fuente: towardsdatascience.com

Algunos de los parámetros más reseñables de Random Forest son los siguientes:

- **ntree**: indica la cantidad de árboles que forman el bosque.
- **mtry**: indica la cantidad de variables candidatas que se utilizan en cada ramificación del árbol. Cabe destacar que son seleccionadas de manera aleatoria por el algoritmo pero siempre se usa el mismo número de ellas.

Para obtener resultados diferentes en cuanto al uso de Random Forest basta con cambiar estos valores, ya que son los que más influyen en la elaboración del modelo. Por ejemplo, si reducimos el valor de mtry vamos a tener menos variables para elegir (aunque sea de forma aleatoria), por lo que las probabilidades de que algunas de estas se repita es mucho menor. Como todo, esto tiene un punto medio ya que si reducimos demasiado el valor de mtry podemos no contar con la suficiente información dando lugar a una menor precisión en cada árbol. Al conseguir el punto de equilibrio lo que lograremos es tener árboles diferentes y poco correlacionados. En general para tareas de predicción, como es nuestro caso, se recomienda un valor de mtry de M/3, aunque en la práctica todo dependerá del problema a resolver [11]. Por otra parte, el número de árboles también afecta a lo preciso que va a ser el modelo. Los modelos de Random Forest no presentan problemas en generar árboles (se caracterizan por su rapidez en la generación), por tanto tener un gran número de árboles no produce efectos adversos, sino al contrario, son útiles ya que pueden proporcionar información auxiliar (proximidad e importancia de variables entre otras). Lo normal sería pensar que cuantos más árboles utilicemos la predicción obtenida será mejor, pero no es así. Llega un punto en el que a pesar de incrementar los árboles de nuestro bosque no obtenemos mejora y es aquí donde nos encontramos con el valor óptimo de ntree (si siguiéramos aumentado los árboles tendríamos un mayor costo computacional sin ningún sentido) [12].

Out Of Bag Error

Por último, vamos a hablar sobre el Out of bag error (OOB). Se utiliza para medir el error de predicción de diversas técnicas de ML. Para ello, se lleva a cabo la técnica de bootstrap que submuestrea los datos utilizados para la construcción del modelo. Como ya comentamos, los bosques aleatorios se construyen a partir de N observaciones con reemplazamiento y al usar bootstraping, en promedio, cada árbol usa 2/3 de los datos. En conclusión, el OOB se encuentra estrechamente relacionado con el número de árboles que conforman el bosque [13].

4.3.3 Redes de neuronas

Las Redes Neuronales Artificiales (ANN) son un modelo inspirado en el funcionamiento del cerebro humano. Está formado por un conjunto de nodos conocidos como neuronas artificiales que están conectadas y transmiten señales entre sí. Estas señales se transmiten desde la entrada hasta generar una salida [14].

El objetivo principal de este modelo es aprender modificándose automáticamente a sí mismo de forma que puede llegar a realizar tareas complejas que no podrían ser realizadas mediante la clásica programación basada en reglas. De esta forma se pueden automatizar funciones que en un principio solo podrían ser realizadas por personas. Es decir, la parte importante de una Red de Neuronas Artificiales es el aprendizaje, puesto que este va a ser el que determine los problemas que la red va a poder resolver. Las ANN basan su aprendizaje en ejemplos, por lo que el que sean capaces de resolver un problema o no estará estrechamente relacionado con los ejemplos de los que se disponen en el proceso de aprendizaje. Todo esto da a entender que debemos de utilizar tanto suficientes ejemplos como muy variados, ya que si esto no se cumple la red se especializará en un conjunto de datos específico y no se podrá usar de manera general. [15].

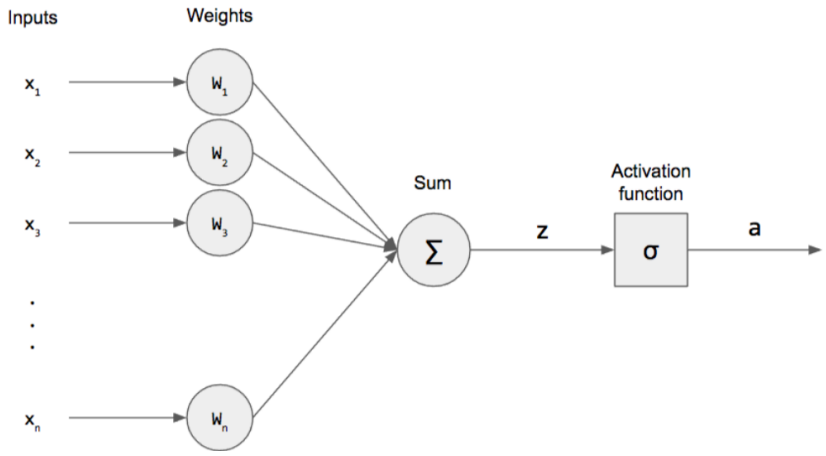

Figura 4.9: Perceptrón simple

Fuente: pythonmachinelearning.pro

Funcionamiento de las ANN

La manera en que funcionan es muy sencilla, ya que lo único que hacen es leer los valores de entrada, sumarlos en función de los pesos de estos e introducir los resultados en una función de activación que genera el resultado final. El entrenamiento realizado se basa en calcular

tanto el umbral que mejor adapte la entrada a la salida como los pesos sinápticos. Para que las ANN sean capaces de calcular estos valores se lleva a cabo un proceso de acondicionamiento en el que los valores parten de un estado inicial aleatorio y se modifican según los valores calculados por la red, así como por los deseados.

En general estas redes constan de una capa de entrada, compuesta por neuronas a las que se les asocia una de las variables del problema, una capa de salida, que es la encargada de darnos el resultado obtenido una vez procesada toda la información y, como mínimo, una capa oculta, que son las que sirven de unión entre las de entrada y salida además de auto-configurar la función global esperada por las Redes Neuronales. A su vez, todas estas capas conforman lo que se conoce como perceptrón multicapa. Este consta de dos fases diferentes: la de propagación, en la cual se propagan hacia las capas siguiente los valores de entrada para conseguir calcular el resultado final y por otro lado, la de aprendizaje, que es su antónimo, ya que los resultados de salida se propagan hacia atrás para poder modificar los pesos de las conexiones y así conseguir un resultado mucho más parejo al real.

Algunas de las ventajas que se obtienen al usar las Redes de Neuronas Artificiales son [16]:

- **Aprendizaje:** tienen la habilidad de aprender mediante una etapa que se llama etapa de aprendizaje. Esta consiste en proporcionar a las redes datos como entrada a su vez que se le indica cuál es la salida esperada.
- **Auto-organización:** crean su propia representación de la información en su interior, obviando al usuario de esto.
- **Tolerancia a fallos:** almacenan la información de forma redundante, por lo que pueden seguir respondiendo de manera aceptable aun si se daña parcialmente.
- **Flexibilidad:** pueden manejar cambios no importantes en la información de entrada, como señales con ruido u otros cambios en la entrada (ej. si la información de entrada es la imagen de un objeto, la respuesta correspondiente no sufre cambios si la imagen cambia un poco su brillo o el objeto cambia ligeramente)
- **Tiempo real:** su estructura es paralela, por lo que si esto es implementado con computadoras o en dispositivos electrónicos especiales, se pueden obtener respuestas en tiempo real.

También cabe mencionar algunas de las desventajas que tiene usar este tipo de redes:

- **Complejidad de aprendizaje:** para grandes tareas, cuantas más cosas se necesite que aprenda una red, mas complicado será enseñarle.
- **Tiempo de aprendizaje elevado:** el cual depende de dos factores. Primero, si se incrementa la cantidad de patrones a identificar o clasificar y segundo si se requiere mayor flexibilidad o capacidad de adaptación de la red neuronal para reconocer patrones que sean sumamente parecidos, se deberá invertir mas tiempo en lograr que la red converja a valores de pesos que representen lo que se quiera enseñar.
- **Elevada cantidad de datos:** se necesitan muchos datos para el entrenamiento. Cuanto más flexible se requiera que sea la red neuronal, mas información habrá que enseñarle para que realice de forma adecuada la identificación.

4.4 Conclusiones

En este trabajo se van a tratar problemas de regresión cuyo objetivo es predecir el retraso extremo a extremo de una red. Para ello llevaremos a cabo un proceso de aprendizaje automático supervisado utilizando las tres técnicas mencionadas anteriormente. En secciones posteriores se explicará en mayor profundidad todos los pasos seguidos durante el desarrollo de este proyecto.

Capítulo 5

Fundamentos tecnológicos

Actualmente, el incremento de software libre hace que el ML sea mucho más accesible para cualquier usuario. La dificultad de implementación actual no tiene nada que ver con la que existía hace años. Debido a esto nos encontramos con una gran variedad de herramientas enfocadas al aprendizaje automático, por lo que hemos tenido que hacer un análisis en profundidad para saber cual elegir.

Estas herramientas utilizadas para el análisis de datos aumentan la precisión y la eficiencia de la investigación en el ámbito de la ciencia de datos. En nuestro caso hemos tenido en cuenta las siguientes herramientas:

- **Azure Machine Learning Studio:** esta opción de Microsoft cuenta con una interfaz de arrastrar y soltar que no requiere experiencia en codificación. Pero se necesita un enfoque aplicado al aprendizaje automático. Además nos permite integrar la tecnología en el trabajo rápidamente. La interfaz visual también permite exportar datos relacionados con el análisis predictivo. El principal problema es que solo se proporciona una versión gratuita del programa y luego pasa a ser de pago mensual.
- **Amazon Machine Learning:** este servicio de aprendizaje automático de Amazon presenta la misma tecnología utilizada internamente por sus científicos de datos. En este momento se encuentra disponible para los clientes que se suscriben al servicio. Esta tecnología ofrece tres capacidades de predicción de aprendizaje automático, por lo que no es necesario conocer ningún método de aprendizaje automático antes de importar datos. La herramienta analiza la información y elige la mejor. En este caso volvemos a tener la restricción del precio por el uso de la tecnología, por lo que también descartamos esta opción.
- **RStudio:** es un entorno de desarrollo integrado (IDE) open source para el lenguaje de programación R, dedicado a la computación estadística y gráficos. Incluye una con-

sola, editor de sintaxis que apoya la ejecución de código, así como herramientas para el trazado, la depuración y la gestión del espacio de trabajo. Una cosa que nos pareció interesante es que podemos encontrar la herramienta en casi cualquier plataforma existente (Windows, Mac y Linux). Las características más reseñables son: el resaltado de sintaxis, auto completado de código, poder ejecutar código R directamente desde el editor de código fuente y salto rápido a las funciones definidas. Debemos resaltar que este entorno cuenta con múltiples paquetes que refuerzan la capacidad de aprendizaje automático en R, ofreciendo un conjunto de funciones que aumentan la eficiencia y creación de los modelos predictivos.

- **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. Es una opción interesante, ya que es open source y cubre todas las necesidades de un proyecto de Machine Learning. Destaca por su portabilidad debido a su implementación sobre Java, además de poder correr en casi cualquier plataforma. Por otro lado, contiene una extensa colección de técnicas para preprocesamiento de datos y modelado y, por último, destacar que es fácil de utilizar por un principiante gracias a su interfaz gráfica de usuario.
- **Orange:** es un programa informático para realizar minería de datos y análisis predictivo. Consta de una serie de componentes desarrollados en C++ que implementan algoritmos de minería de datos, así como operaciones de preprocesamiento y representación gráfica de datos. Sus características más reseñables son la visualización interactiva de datos gracias a su inteligente visualización, la existencia de varios complementos para extraer datos de fuentes de datos externas y que la interfaz gráfica de usuario permite centrarse en el análisis exploratorio de datos en lugar de en la codificación.

Una vez que analizamos en profundidad estas herramientas, nos hemos decantado por el uso de RStudio para llevar a cabo el proyecto. Esto fue decidido así gracias a la pequeña curva de aprendizaje de la herramienta así como de la gran cantidad de documentación existente para esta.

5.1 Librerías utilizadas en R

RStudio cuenta con una multitud de librerías orientadas a ML, por lo que en esta parte de la memoria expondremos las más importantes que se han utilizado en las diferentes fases del proyecto.

5.1.1 Librerías de procesado de datos:

- **Caret:** proporciona funciones precisas para poder particionar los datos.

- **Solitude:** es una implementación de Isolation Forest que se utiliza para la limpieza de los datos.
- **Philentropy:** nos permite utilizar las funciones cosinedist() y jaccard() para ser capaces de cuantificar tanto las distancia como la similitud entre los vectores del conjunto de datos.
- **FSelector:** nos proporciona varias funciones para seleccionar atributos de nuestro dataset. La selección de subconjuntos es el proceso de identificar y eliminar la mayor cantidad de información irrelevante y redundante posible.
- **Mltools:** son una colección de funciones auxiliares de aprendizaje automático, que ayudan particularmente en la fase de análisis de datos exploratorios.
- **Dplyr:** un paquete rápido y consistente para trabajar con data frames como objetos, tanto en memoria como fuera de ella.

5.1.2 Librerías para la elaboración de informes:

- **Rocr:** librería que nos aporta la posibilidad de elaborar muchas de las gráficas relacionadas con las técnicas de ML utilizadas en RStudio.
- **Knitr:** proporciona una herramienta de uso general para la generación de informes dinámicos en R utilizando técnicas de programación alfabetizada.

5.1.3 Librerías para Random Forest:

- **Random Forest:** es la librería que se va a utilizar para crear el modelo con la función randomForest(), así como llevar a cabo la obtención de resultados al usar esta técnica.

5.1.4 Librerías para K-NN:

- **Class:** incluye varias funciones dentro de las cuales destaca la de KNN.
- **Fnn:** nos da acceso a las utilidades de Kd-tree y Cover-tree, que son algoritmos de búsqueda rápida de KNN. En general, se implementan en esta librería medidas de clasificación, regresión e información KNN.

Capítulo 6

Comprensión de los datos

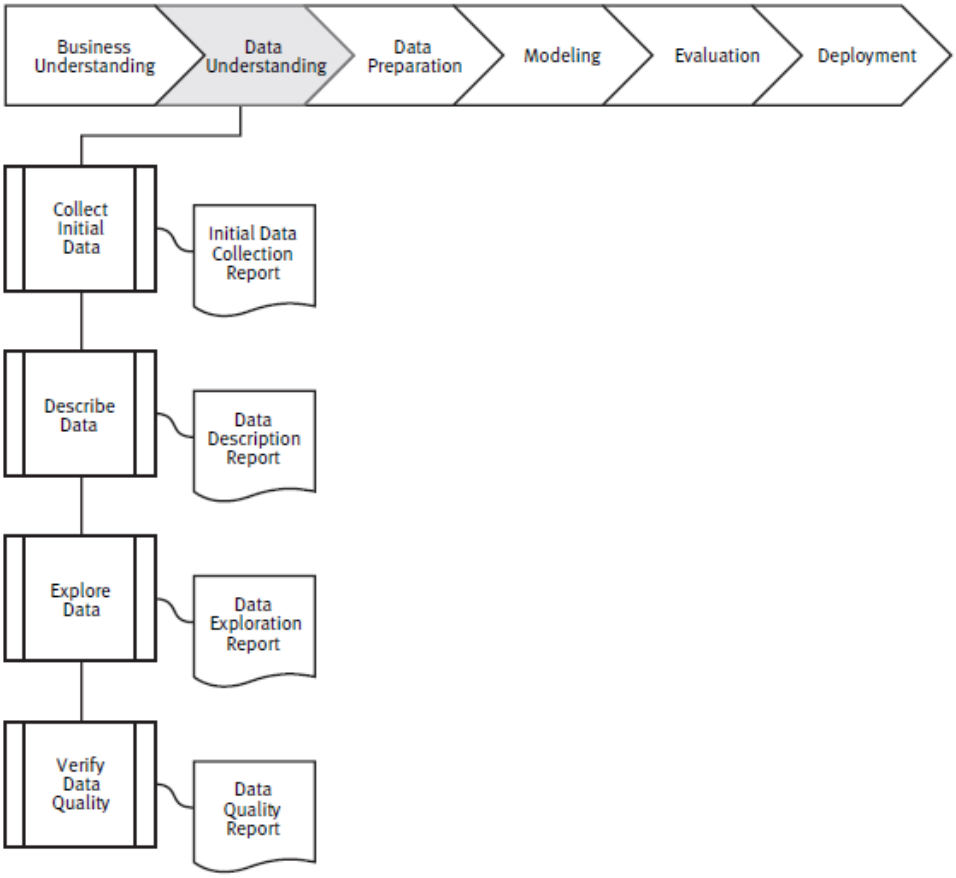

Figura 6.1: CRISP-DM (2)

6.1 Descripción

Como hemos indicado anteriormente vamos a usar aprendizaje supervisado para predecir la variable objetivo (delay) en el tráfico de extremo a extremo. Se utilizó un dataset público etiquetado totalmente, formado por muestras creadas con OMNet++, el cual es un simulador a medida. Dichas muestras que cuentan con varias configuraciones de enrutamiento, patrones de tráfico y diferentes topologías. Cada una contiene mediciones exactas sobre las métricas de rendimiento más relevantes en cuanto al tráfico de extremo a extremo, concretamente del delay (retraso) y del jitter (fluctuación) [17].

6.2 Recolección

Esta etapa se centra en elegir el dataset que se empleará a lo largo del proyecto. Nos hemos decantado por seleccionar el conjunto de datos mencionado anteriormente debido a que se trata de un dataset que ya divide los datos en entrenamiento y pruebas además de estar etiquetados. Otra de las cualidades que tiene y destaca en gran medida es el hecho de que haya sido probado con Redes de Neuronas Artificiales, por lo que podremos comparar resultados con los otros modelos que creemos.

6.3 Exploración

Uno de los problemas que surgieron en la exploración de los datos era que estaban en formato .tfrecord, que es el formato que se utiliza en la herramienta de TensorFlow [18]. Debido a que nos encontramos frente al problema mencionado, era necesario convertir el formato a uno correcto para usarlo con nuestra herramienta, RStudio. Es por ello que se ha procedido a la implementación de un código en Python capaz de convertir el mencionado formato a uno más manejable como es csv.

El código que podemos ver a continuación es el encargado de convertir el formato .tfrecord a .txt para que podamos acceder al contenido de los diferentes archivos que forman nuestro dataset y ser capaces de analizarlos.

```
1
2  import os
3  import sys
4  import tensorflow as tf
5
6  d1 = "/home/tfg/CarpetaCompartidaMV/tfrecords/train"
7  d2 = "/home/tfg/CarpetaCompartidaMV/tfrecords/evaluate"
8  d3 = "/home/tfg/CarpetaCompartidaMV/archivosConvertidosTXT/train/"
9  d4 =
       "/home/tfg/CarpetaCompartidaMV/archivosConvertidosTXT/evaluate/"
10
11 destD1 = os.listdir(d1)
12 destD2 = os.listdir(d2)
13
14 for file1 in destD1:
15   p1 = d1 + "/" + file1
16   r1 = None
17   for ex1 in tf.python_io.tf_record_iterator(p1):
18     r1 = tf.train.Example.FromString(ex1)
19   sys.stdout = open(d3 + file.replace(".tfrecords", ".txt"), "w")
20   print(r1)
21
22 for file2 in destD2:
23   p2 = d2 + "/" + file2
24   r2 = None
25   for ex2 in tf.python_io.tf_record_iterator(p2):
26     r2 = tf.train.Example.FromString(ex2)
27   sys.stdout = open(d4 + file2.replace(".tfrecords", ".txt"), "w")
28   print(r2)
```

Listing 6.1: Código que sirve para pasar de formato .tfrecord a .txt

Una vez concluida la transformación, el siguiente paso es coger todos estos datos y crear un dataframe de manera que pueda ser usado en nuestra herramienta. Esto se hizo de una manera muy sencilla, ya que lo único que tenemos que hacer es recorrer los datos que están en formato .txt, ir metiéndolos en un array y ya cuando se haya acabado, crear el dataframe correspondiente. El siguiente código muestra el fragmento final en el que se convierte a dataframe y luego, la transformación a csv.

```
1
2  array.append(
3      {'delay': delay,
4       'jitter': jitter,
5       'link_capacity' : link_capacity,
6       'links':  array_elements[3][i],
7     'n_links':  n_links,
8     'n_paths':  n_paths,
9     'n_total':  n_total,
10    'paths': array_elements[7][i],
11    'sequences' : array_elements[8][i],
12    'traffic':  traffic})
13
14 data = pd.DataFrame(
15     array, index=None,
16     columns = ['delay', 'jitter', 'link_capacity', 'links',
       'n_links', 'n_paths', 'n_total', 'paths', 'sequences',
       'traffic'])
17
18 data.to_csv(
19
       "/home/tfg/CarpetaCompartidaMV/archivosConvertidosCSV/evaluate/"
       + file1.replace(".txt", ".csv"),
20     ",",
21     index = False)
```

Listing 6.2: Código que sirve para pasar de formato .txt a .csv

Terminada la conversión, toca enfocarse en la importación de los datos a la herramienta, pero nos enfrentamos a un nuevo problema. El dataset que hemos elegido cuenta con aproximadamente 331.800 muestras, por lo que la carga computacional es terriblemente grande. Al comprender que no era viable usar todo el conjunto, se optó por seleccionar un subconjunto representativo del total, compuesto por unas 38.664 muestras que se dividen en entrenamiento (70%) y pruebas (30%), manteniendo la proporción de valores de características .

Ya dándole fin a la sección, vamos a hablar sobre el análisis que se ha hecho sobre el dataset para realizar un correcto tratamiento de los datos. Lo primero es hacernos una idea de la cantidad de datos que tiene cada columna de nuestro conjunto de datos. En la siguiente figura se puede apreciar en profundidad dichos valores:

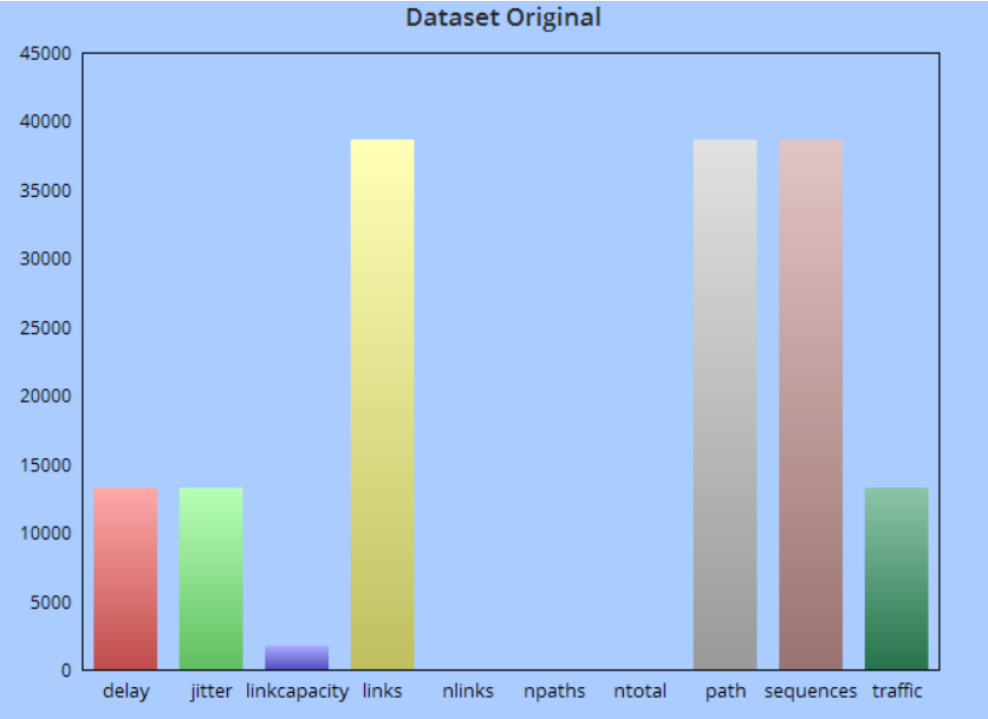

Figura 6.2: Dataset Original

- **delay**: 13248
- **jitter**:13248
- **linkcapacity**: 1.776
- **links**: 38664
- **nlinks**: 24
- **npaths**: 24
- **ntotal**: 24
- **path**: 38664
- **sequences**: 38664
- **traffic**: 13248

Si queremos evitar tanto problemas como errores en los algoritmos de Machine Learning debemos realizar un correcto tratamiento de los valores que faltan en algunas columnas, los cuales son nombrados como NA's (valores nulos o que no existen). Contemplamos dos opciones: la primera, eliminar estas entradas y la segunda, completarlas.

En cuanto a las columnas de nlinks, npaths y ntotal podemos observar que prácticamente no hay entradas. Esto ocurre porque simplemente indican el número total de enlaces y caminos que tiene la topología de la red. Una vez que entendemos estos atributos, la mejor opción es eliminarlos debido a la pequeña aportación de información. Además de los ya comentados, contamos con el atributo linkcapacity, el cual nos muestra el bit rate de transmisión física de un enlace. Podría ser un valor interesante pero debido a su escasez también consideramos su eliminación, ya que no sería productivo dejar un atributo en el dataset con tan poca cantidad de datos. En la siguiente figura podemos apreciar como van quedando los datos una vez realizadas las modificaciones pertinentes:

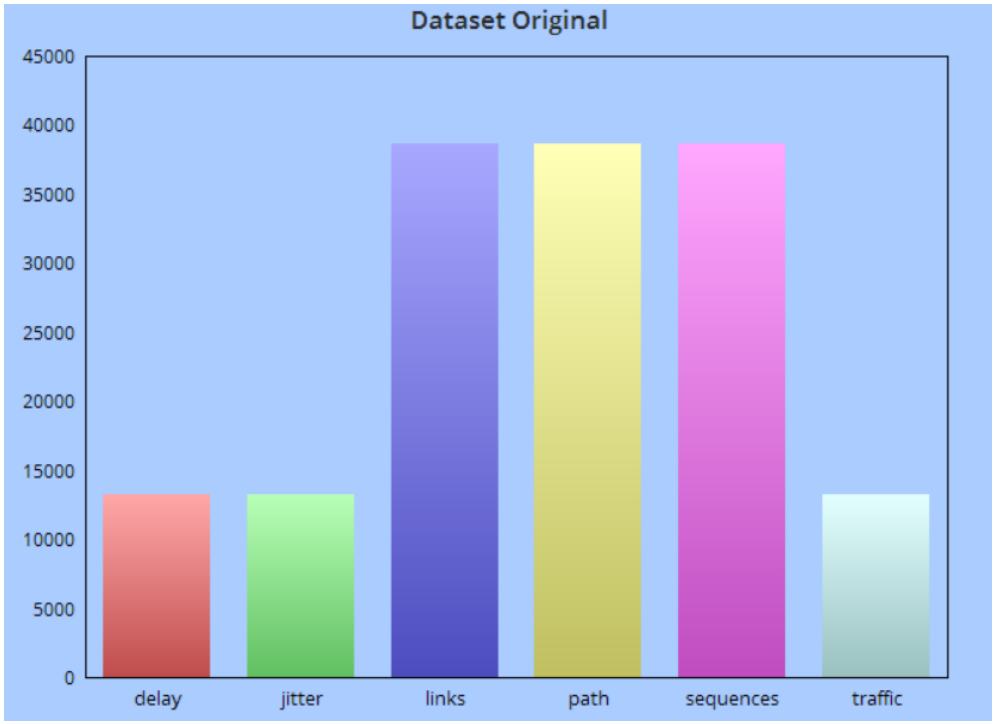

Figura 6.3: Dataset con la primera modificación

- **delay**: 13248
- **jitter**:13248
- **sequences**: 38664
- **traffic**: 13248

- **links**: 38664 **path**: 38664

Por otro lado también tenemos que fijarnos en los valores NA's de las columnas delay, jitter y traffic. Estos valores tienen sentido ya que en determinados caminos y enlaces existe la ausencia de tráfico. Ya que nuestro dataset es muy extenso y que solo estamos interesados en tener en cuenta aquellas muestras en las cuales el delay (la que va a ser nuestra variable dependiente) tenga valor, nos decantamos por la eliminación de las muestras con valores NA's para estas columnas. A pesar de todos los valores afectados, se concluyó que no debería afectar significativamente a los datos gracias a la gran extensión de estos, así como una vez eliminados, nuestros algoritmos de ML podrán desenvolverse correctamente. En la siguiente figura ya se puede ver como queda el dataset después de realizar todos los ajustes que consideramos necesarios:

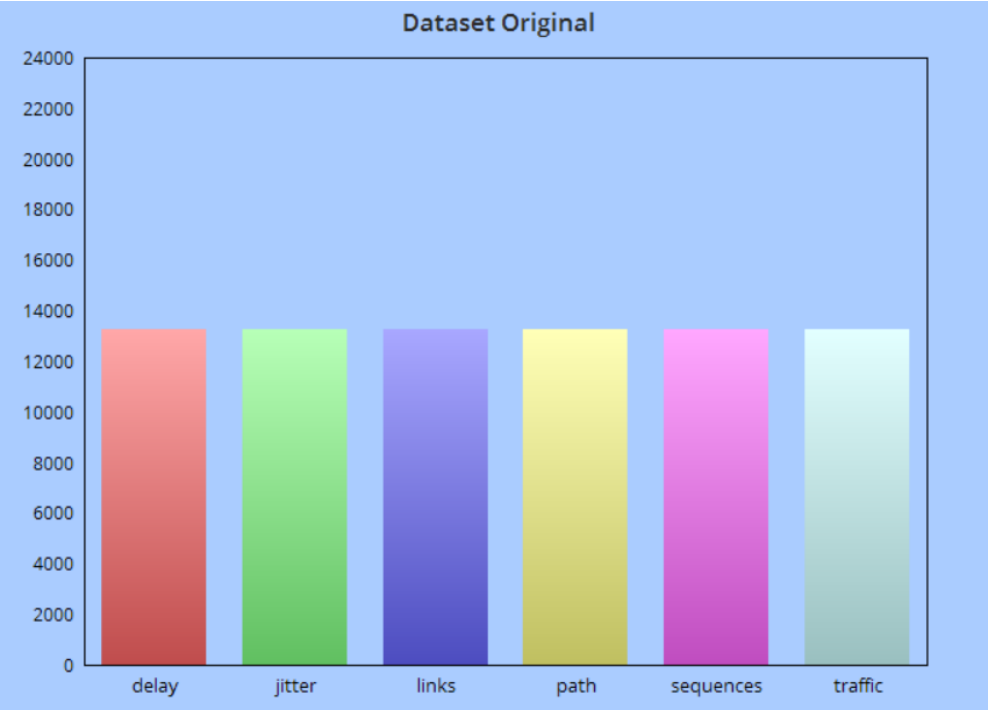

Figura 6.4: Dataset con la segunda modificación

- **delay**: 13248
- **jitter**:13248
- **links**: 13248
- **sequences**: 13248
- **traffic**: 13248
- **path**: 13248

Capítulo 7

Prepararación de los datos

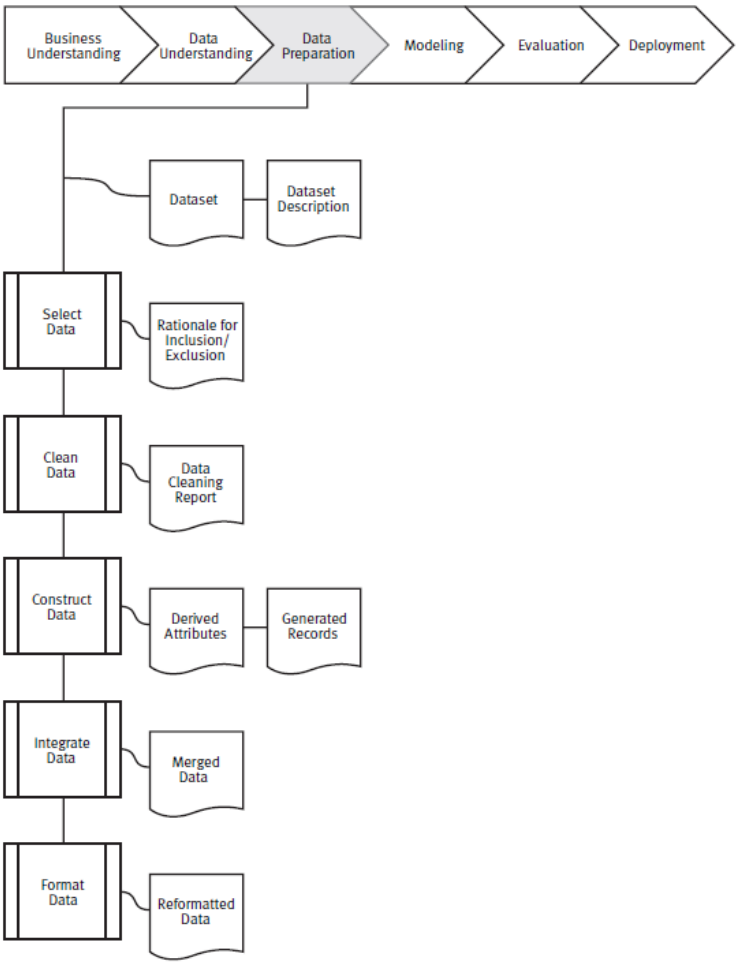

Figura 7.1: CRISP-DM (3)

Fuente: ftp.software.ibm.com

En esta etapa vamos a dedicarnos tanto a la limpieza de los datos como a su transformación y posterior selección de las características que mejor encajan con estos. Cuando procesamos datos tenemos que tener en cuenta la importancia de realizar una buena limpieza y análisis previo de estos para que, posteriormente, puedan ser usados correctamente en los diversos procesos de ML y sea posible conseguir resultados satisfactorios.

En cuanto a la fase de transformación de los datos, va a centrarse en elaborar nuevos atributos a partir de los que ya tenemos. Para ser capaces de seleccionar las características que más benefician al modelo, se llevará a cabo un análisis de los diversos métodos de cálculo de similitud entre los vectores que componen el dataset.

7.1 Limpieza

Como ya hemos mencionado anteriormente, los datos pueden presentar irregularidades o estar dañados por lo que comprometerían la calidad del conjunto de datos. Los problemas de calidad de datos más habituales que podemos resolver en la etapa de limpieza son:

- **Incompletos**: en los datos no hay atributos o contienen valores que faltan.
- **Con ruido**: los datos contienen registros erróneos o valores atípicos.
- **Incoherentes**: los datos contienen discrepancias o registros en conflicto.

Los datos de calidad son un requisito previo para los modelos predictivos de Machine Learning. Para mejorar la calidad de los datos y, por tanto, el rendimiento del modelo, es fundamental llevar a cabo dicha limpieza para así ser capaces de detectar problemas prematuros. La presencia de datos atípicos en el conjunto de datos puede suponer que los modelos no se adecúen a la realidad del problema a resolver, destacando el hecho de que no siempre es necesaria su eliminación aunque siempre es aconsejable hacerlo porque se puede disminuir considerablemente la carga computacional y el volumen del dataset que vayamos a utilizar.

Es muy común que los datos presenten inconsistencias relacionadas, como ya se vio anteriormente, con datos incompletos o inconsistentes que no encajan dentro de los valores esperados, los llamados datos anómalos u ”outliers”.

Después de un análisis de diversas técnicas para la limpieza de datos, nos hemos decidido por el uso de Isolation Forest [19].

7.1.1 Isolation Forest

Isolation Forest es una método no supervisado para identificar anomalías (outliers). Su funcionamiento está inspirado en el algoritmo de clasificación y regresión Random Forest. Al igual que en Random Forest, un modelo Isolation Forest está formado por la combinación de múltiples árboles llamados isolation trees, en los cuales la selección de los puntos de división se hace de forma aleatoria. Aquellas observaciones con características distintas al resto, quedarán aisladas en las divisiones iniciales, por lo que el número de nodos necesarios para llegar a estas observación desde el inicio del árbol (profundidad) es menor que para el resto. Con esto lo que se quiere decir es que las anomalías son más susceptibles al aislamiento. En las siguientes imágenes nos podemos hacer una idea de como funciona esta técnica además de su implementación en R y de los resultados que arrojó sobre el dataset seleccionado:

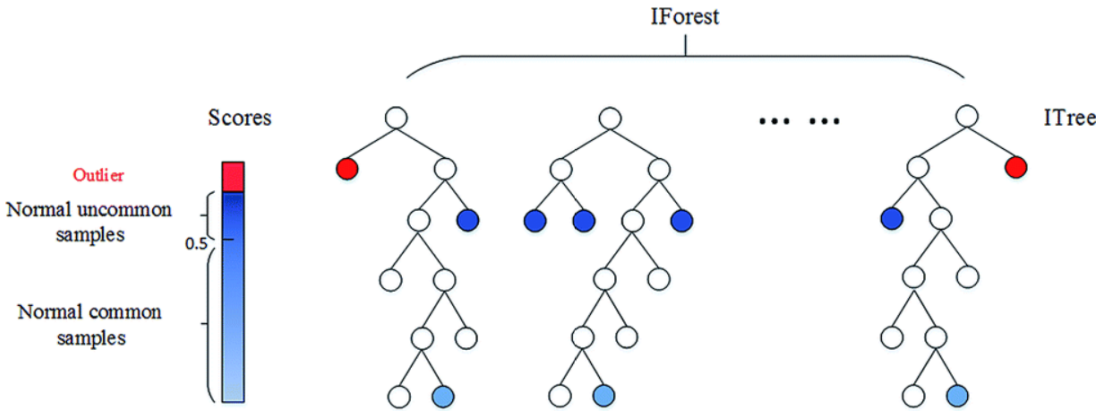

Figura 7.2: Técnica de limpieza Isolation Forest

Fuente: donghwa-kim.github.io

```r
library("solitude")

iforest <- isolationForest$new()
iforest$fit(train)
print(iforest$scores)

train$pred <- iforest$predict(train)
train$outlier <- as.factor(ifelse(train$pred$anomaly_score >=0.50,
    "outlier", "normal"))
```

Listing 7.1: Código elaborado para realizar la técnica de Isolation Forest

```
pred.average_depth   pred.anomaly_score    outlier
Min.   :11.896802    Min.   :0.2093091   normal:13248
1st Qu.:17.987559    1st Qu.:0.2485216
Median :19.442291    Median :0.2683566
Mean   :19.124206    Mean   :0.2766869
3rd Qu.:20.577206    3rd Qu.:0.2961132
Max.   :23.115209    Max.   :0.4471231
```

Figura 7.3: Resultado de Isolation Forest sobre el dataset

Vamos a comentar un poco los datos que vemos en la última figura. El valor que nos permite saber si hay datos anómalos es ”anomaly score”. Cuanto menores de 0.5 sean los valores, las probabilidades de que sean considerados datos normales aumentan drásticamente, mientras que si se superara dicho umbral ocurriría lo inverso, teniendo mayor probabilidad de considerarse una anomalía cuanto más cerca de 1 se encuentren estos. Por lo tanto, los resultados devueltos por Isolation Forest indican que no existen anomalías en nuestro conjunto de datos.

7.2 Transformación

En esta fase es donde vamos a crear los nuevos atributos a partir de los ya existentes, calculando las distancias entre los diferentes vectores que forman nuestro conjunto de datos. Concretamente utilizaremos la distancia coseno y el índice de jaccard, los cuales vamos a explicar a continuación:

- **Índice de Jaccard**: mide el grado de similitud entre dos conjuntos, sea cual sea el tipo de elementos. Los resultados que arroja son 0 si los vectores seleccionados en la función no tienen ninguna característica pareja y, en caso de tener muchas en común, tendería a 1 [20].
- **Distancia Coseno**: es una medida de la similitud existente entre dos vectores en un espacio que posee un producto interior con el que se evalúa el valor del coseno del ángulo comprendido entre ellos. Esta función trigonométrica proporciona un valor igual a 1 si el ángulo comprendido es cero, es decir si ambos vectores apuntan a un mismo lugar [21].

Ya obtenidos los resultados (tarda bastante debido a la carga computacional generada por tantos datos y al ordenador personal en el que se ejecuta), somos capaces de observar las similitudes existentes entre los distintos vectores. Ahora toca calcular los estadísticos a partir de los resultados que obtuvimos porque los usaremos en la generación de los modelos de Machine Learning. En las siguientes imágenes podemos ver los estadísticos que se calcularon a partir de los resultados arrojados por el índice jaccard y la distancia coseno:

delay	jitter	links	paths	sequences	traffic	FirstQuCosine	medianCosine	meanCosine	ThirdQuCosine	maxCosine	sdCosine	varCosine
0.5491320	0.238221005	0	0	0	0.4929860	0.002278202	0.003688803	0.01087788	0.00725905	0.9819031	0.04282674	0.001834130
0.1623640	0.012087000	1	1	0	0.2506140	0.773418121	0.804466231	0.82638895	0.88630842	0.9990538	0.08005872	0.006409399
0.3431960	0.044449799	1	2	0	0.3429000	0.918613619	0.929719798	0.92434584	0.95410999	0.9999846	0.06071280	0.003686044
0.3065250	0.034241501	7	2	1	0.2489860	0.388491181	0.436964883	0.51235214	0.59485374	0.9999827	0.18767608	0.035222312
0.3638500	0.047058899	1	3	0	0.2265860	0.964421136	0.974115603	0.95040541	0.98449060	0.9998535	0.08562368	0.007331414
0.6096910	0.284900993	8	3	1	0.1451710	0.460945860	0.507310001	0.57523097	0.65685354	0.9999663	0.17193135	0.029560391
0.3311110	0.039278898	1	4	0	0.4030290	0.975146086	0.985341533	0.95107018	0.98946687	0.9999828	0.10102623	0.010206299
0.6113750	0.253879994	7	4	1	0.3370290	0.594036106	0.635594592	0.68782698	0.76588098	0.9999453	0.13884526	0.019278007
0.6064830	0.271793991	11	4	2	0.3481290	0.449146081	0.495903945	0.56496944	0.64599368	0.9998906	0.17394528	0.030256959
0.8932640	0.350796998	0	5	0	0.3726710	0.915994532	0.964784908	0.90177077	0.97253893	0.9993751	0.14998637	0.022495913
0.6875550	0.295812011	5	5	1	0.1223710	0.778032770	0.810041997	0.83315297	0.89404136	0.9983152	0.08423207	0.007095042
0.6359670	0.269073009	1	6	0	0.2710570	0.973219912	0.990954830	0.94521016	0.99244651	0.9994708	0.11800081	0.013924191
0.6983620	0.283872992	8	6	1	0.0660429	0.690024386	0.727313459	0.76601802	0.84062493	0.9998381	0.11281117	0.012726361
0.6832700	0.264046013	14	6	2	0.3819430	0.500665520	0.546050095	0.60961799	0.69058004	0.9997432	0.16307363	0.026593009
0.8809280	0.314657003	0	7	0	0.4031710	0.925259175	0.974568336	0.91050049	0.98232058	0.9997743	0.15206944	0.023125115
0.6890330	0.261828989	5	7	1	0.1591290	0.869120960	0.889887387	0.89966665	0.93771257	0.9998435	0.06162302	0.003797396
0.6556610	0.272426993	17	7	2	0.3439140	0.489924272	0.535410322	0.60049524	0.68211546	0.9999059	0.16603366	0.027567178

Figura 7.4: Atributos generados mediante la distancia coseno

delay	jitter	links	paths	sequences	traffic	FirstQuJaccard	medianJaccard	meanJaccard	ThirdQuJaccard	maxJaccard	sdJaccard	varJaccard
0.5491320	0.238221005	0	0	0	0.4929860	0.9999058	0.9999698	0.9983249	0.9999867	0.9999981	0.02532713	0.0006414633
0.1623640	0.012087000	1	1	0	0.2506140	0.9772851	0.9869206	0.9721815	0.9916644	0.9944752	0.06443384	0.0041517199
0.3431960	0.044449799	1	2	0	0.3429000	0.9606902	0.9766517	0.9529704	0.9844261	0.9890976	0.08720931	0.0076054643
0.3065250	0.034241501	7	2	1	0.2489860	0.9213782	0.9615962	0.9204154	0.9773095	0.9983429	0.12078505	0.0145890281
0.3638500	0.047058899	1	3	0	0.2265860	0.9436454	0.9667827	0.9346129	0.9773639	0.9904542	0.10575661	0.0111844608
0.6096910	0.284900993	8	3	1	0.1451710	0.8958648	0.9474008	0.8991489	0.9683074	0.9982146	0.13555852	0.0183761130
0.3311110	0.039278898	1	4	0	0.4030290	0.9256363	0.9559199	0.916 0.8991489	0.9701437	0.9932646	0.12241056	0.0149843440
0.6113750	0.253879994	7	4	1	0.3370290	0.8846211	0.9395364	0.8859210	0.9619522	0.9975962	0.14295201	0.0204352759
0.6064830	0.271793991	11	4	2	0.3481290	0.8579193	0.9282864	0.8724686	0.9569160	0.9988635	0.15082470	0.0227480916
0.8932640	0.350796998	0	5	0	0.3726710	0.9154935	0.9493151	0.9069501	0.9648262	0.9916580	0.13023623	0.0169614755
0.6875550	0.295812011	5	5	1	0.1223710	0.8768919	0.9318817	0.8768922	0.9570370	0.9973211	0.14881332	0.0221454031
0.6359670	0.269073009	1	6	0	0.2710570	0.8912343	0.9340142	0.8825402	0.9556771	0.9958617	0.15063413	0.0226906401
0.6983620	0.283872992	8	6	1	0.0660429	0.8361950	0.9125559	0.8494147	0.9457095	0.9988302	0.16252159	0.0264132672
0.6832700	0.264046013	14	6	2	0.3819430	0.8081464	0.8976311	0.8339497	0.9377532	0.9992438	0.16811394	0.0282622954
0.8809280	0.314657003	0	7	0	0.4031710	0.8783795	0.9277750	0.8739081	0.9501150	0.9956388	0.15578803	0.0242699117
0.6890330	0.261828989	5	7	1	0.1591290	0.8364857	0.9082451	0.8439100	0.9410111	0.9980353	0.16990602	0.0288680543
0.6556610	0.272426993	17	7	2	0.3439140	0.7768420	0.8782923	0.8135148	0.9263812	0.9995071	0.17704955	0.0313465443

Figura 7.5: Atributos generados mediante el índice Jaccard.

7.3 Selección

Esta va a ser la última fase que realizaremos en cuanto a la preparación de los datos de nuestro conjunto. Va a consistir en determinar qué atributos del conjunto son los que aportan una mayor cantidad de información. Una vez visto cuales son los mas importantes en cuanto a ganancia de información, podríamos prescindir de los que menos aportan para así aumentar la calidad de nuestros modelos de Machine Learning.

En este trabajo decidimos utilizar la técnica de ganancia de información que se basa en el cálculo de la entropía de cada atributo con respecto a la variable dependiente (en nuestro caso delay). Cuanto mas grandes sean los valores obtenidos, mayor será la información aportada por el atributo pertinente [22].

En la siguiente figura se muestra la ganancia de información obtenida para nuestro conjunto de datos inicial:

```
            attr_importance
jitter           1.397016604
links            0.074764270
paths            0.568554147
sequences        0.002289328
traffic          0.040297681
```

Figura 7.6: Ganancia de información del conjunto de datos inicial.

Se ve que los atributos que más resaltan son el jitter y paths. Esto se debe a que ambos están muy relacionados con el retraso de extremo a extremo. El jitter hace referencia al tiempo que ha transcurrido entre dos paquetes distintos recibidos, por lo que está directamente relacionado con nuestra variable objetivo y paths es el número de caminos que hay entre extremos, por lo que cuantos más caminos se tengan menor será la probabilidad de que ocurra un fallo en la red.

Como bien podemos ver, los atributos restantes aportan menos información, por lo que podrían ser prescindibles en nuestro proyecto. Vamos a explicar en mayor profundidad lo que es cada uno:

- **Links**: hace referencia al número de nodos intermedios entre extremos. Cabría pensar que cuanto más nodos haya por el medio, mayor será el retraso obtenido.
- **Sequences**: es la cantidad de bits enviados por paquete. Podríamos pensar que si enviamos una mayor cantidad de información en cada paquete obtendremos un retraso menor aunque a su vez depende de muchos otros factores.
- **Traffic**: se refiere al tráfico que tiene la red.

Hay que comentar que cuando se trata de temas de red no siempre los factores de esta actúan como nosotros originalmente pensamos. Es por este motivo por lo que siempre es bueno llevar a cabo técnicas de ganancia de información para hacernos una idea de qué atributos tienen una mayor importancia aunque los resultados obtenidos no siempre están acertados. En nuestro caso decidimos utilizar todos los atributos dado que los tiempos computacionales obtenidos en la ejecución de Random Forest y K-NN son aceptables. Además como se va a realizar una comparativa con las Redes Neuronales, se prefirió no eliminar ningún atributo para mantener en igualdad de condiciones a los algoritmos anteriormente mencionados
.

Por último, mencionar que se ha llevado a cabo la técnica de ganancia de información sobre los dataset que cuentan con los valores estadísticos calculados a partir del índice jaccard y de la distancia coseno. A continuación podemos apreciar los resultados obtenidos para estos conjuntos:

	attr_importance
jitter	1.397016604
links	0.074764270
paths	0.568554147
sequences	0.002289328
traffic	0.040297681
minCosine	0.000000000
FirstQuCosine	0.927197860
medianCosine	0.878884978
meanCosine	1.013579119
ThirdQuCosine	0.813719086
maxCosine	0.106322657
sdCosine	0.961179603
varCosine	0.961179603

Figura 7.7: Ganancia de información del conjunto de datos coseno.

	attr_importance
jitter	1.397016604
links	0.074764270
paths	0.568554147
sequences	0.002289328
traffic	0.040297681
minJaccard	0.000000000
FirstQuJaccard	1.204975468
medianJaccard	1.175794906
meanJaccard	1.150759473
ThirdQuJaccard	1.166503877
maxJaccard	0.131909558
sdJaccard	1.193300850
varJaccard	1.193300850

Figura 7.8: Ganancia de información del conjunto de datos jaccard.

Vemos que estos nuevos valores aportan bastante información. Más adelante veremos los resultados obtenidos y sabremos qué algoritmos y con qué datos ofrecen mejores resultados.

Capítulo 8

Modelado

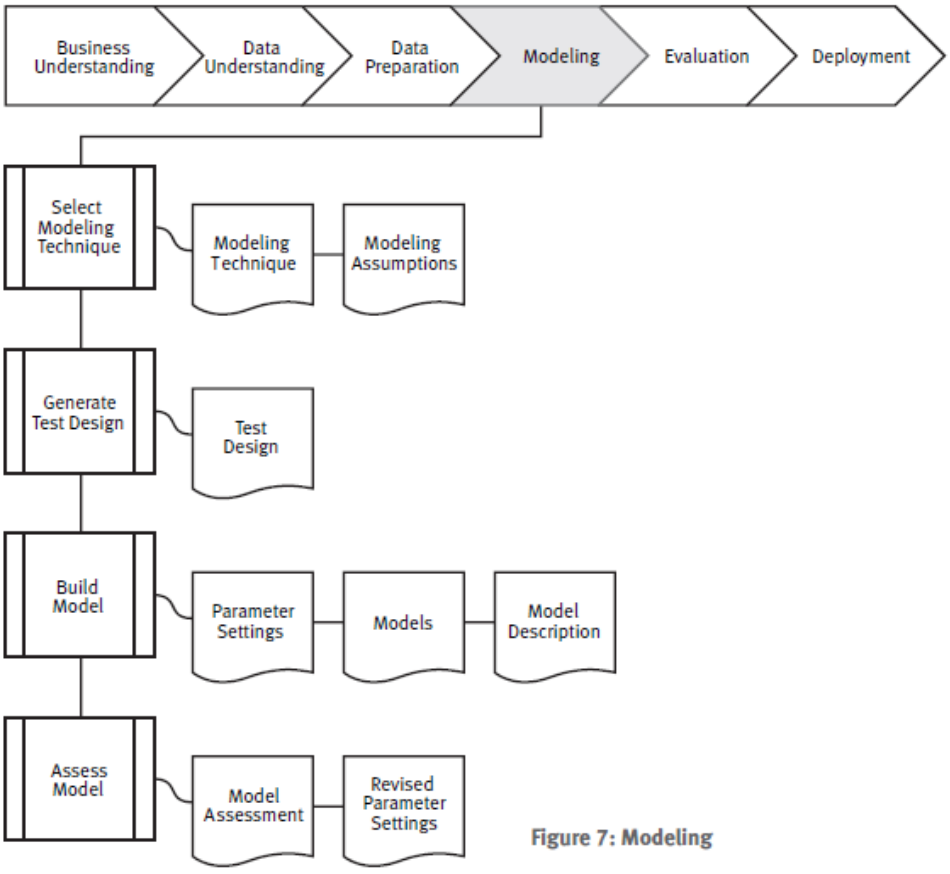

Figura 8.1: CRISP-DM (4)

En este capítulo se van a definir las métricas de rendimiento que serán usadas para evaluar los modelos construidos y también se definirá la metodología de validación mediante la cual seremos capaces de encontrar los valores óptimos para los hiperparámetros más importantes. Todo esto nos vale para conseguir obtener la máxima calidad posible en nuestros modelos.

8.1 Métricas

Las métricas que normalmente se utilizan en cuanto a la resolución de problemas de regresión y que sirven para ver el rendimiento de los modelos de machine learning son las siguientes:

- **Error Cuadrático Medio**: esta métrica, también conocida como Mean Square Error (MSE), mide el error que hay entre dos conjuntos de datos. En otras palabras, compara un valor predicho y un valor observado o conocido. Cuanto mayor sea el valor obtenido, peor será el modelo. No se puede devolver valores negativos pero si el modelo fuera perfecto entonces lo que obtendríamos sería un 0 [23]. Se define por la siguiente ecuación:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{8.1}$$

- **Raíz del Error Cuadrático Medio**: también conocido como Root Mean Square Error (RMSE), mide el error que hay entre dos conjuntos de datos. En otras palabras, compara un valor predicho y un valor observado o conocido. La raíz cuadrada se introduce para hacer que la escala de los errores sea igual a la escala de los objetivos. Como pasaba en el MSE, nunca es negativo y cuanto mayor sea el resultado devuelto peor será la calidad del modelo evaluado [24].

- **Error Absoluto Medio**: es una medida de la diferencia entre dos variables continuas. Considerando dos series de datos (unos calculados y otros observados) relativos a un mismo problema, el error absoluto medio sirve para cuantificar la precisión de una técnica de predicción (en este caso se aplica a nuestros modelos) comparando los valores predichos frente a los observados [25]. Se define por la siguiente ecuación:

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i| \tag{8.2}$$

- **R^2**: es una medida estadística de cuan cerca están los datos de la línea de regresión. También se conoce como coeficiente de determinación, o coeficiente de determinación múltiple si se trata de regresión múltiple. Es decir, es el porcentaje de la variación en la variable de respuesta que es explicado por un modelo lineal (R^2 = Variación explicada / variación total). Los valores suelen estar entre 0 y 1 y, en general, cuanto mayor es resultado devuelto por R^2, mejor se ajusta el modelo a los datos [26]. Se define por la siguiente ecuación:

$$R^2 = 1 - \frac{MSE(model)}{MSE(baseline)} \tag{8.3}$$

8.2 Random Forest

Como punto de partida tenemos que hablar sobre algunas cosas a tener en cuenta para la creación de los modelos con este algoritmo. Vamos a poder seleccionar los hiperparámetros más importantes gracias al análisis realizado por el proceso de validación. Dicho proceso solo es empleado para lo comentado anteriormente debido a que Random Forest valida por sí mismo el modelo y no es necesario llevar a cabo validaciones externas.

8.2.1 Hiperparametrización

Los hiperparámetros son variables a las que les establecemos un valor antes de llevar a cabo el proceso de entrenamiento para poder afinar la calidad final del modelo. Estos deben tenerse en cuenta para la creación óptima de los árboles que forman el bosque, entre los cuales destacan los que ya hemos mencionado anteriormente:

- **Mtry**: es el número de atributos seleccionados para cada división. Se recomienda establecer un valor estándar de M/3 pero teniendo en cuenta que, en función del problema a resolver, debemos de ajustarlo. En este trabajo para ser capaces de identificar el valor más adecuado, se realizó un análisis del error OOB (tercio restante de los datos utilizado para pruebas) en función de los atributos que se eligieron en cada división de los árboles para cada uno de los datasets. Esto se aplicó sobre el dataset original (con su preparación de datos previa), así como sobre los datasets que cuentan con los atributos

creados a partir del uso del índice jaccard y de la distancia coseno. A continuación podemos ver los resultados obtenidos mediante el uso de la técnica de validación cruzada (la explicaremos en la sección de validación 8.2.2):

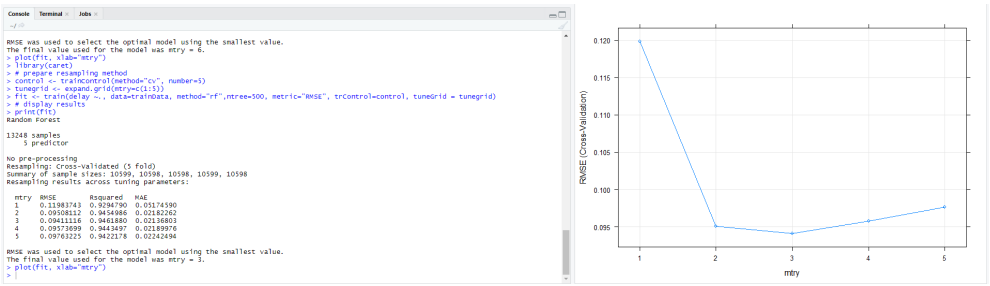

Figura 8.2: Mtry para el dataset original.

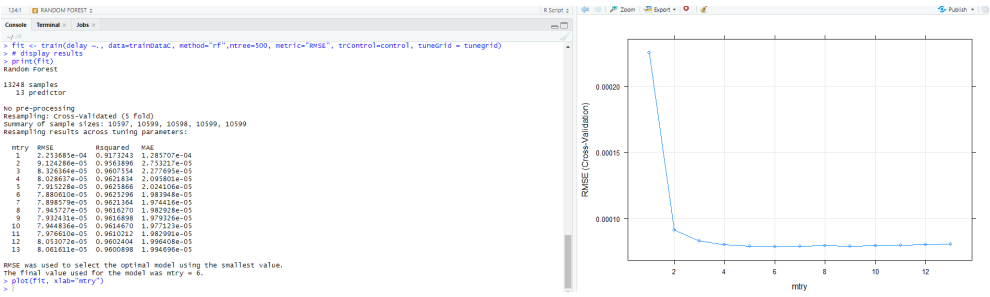

Figura 8.3: Mtry para el dataset coseno.

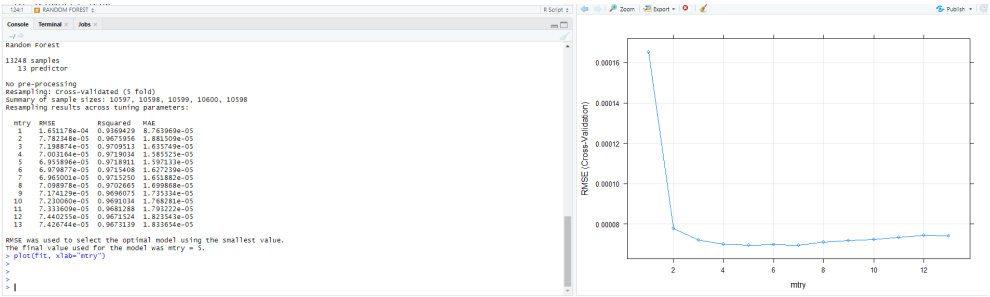

Figura 8.4: Mtry para el dataset jaccard.

- **Ntree**: es el número de árboles que se utilizan para construir el bosque. En RStudio, la función randomforest() contenida en la librería randomForest establece este hiperparámetro a 500 árboles. En las siguientes gráficas se puede ver como varía el mse en función del número de árboles:

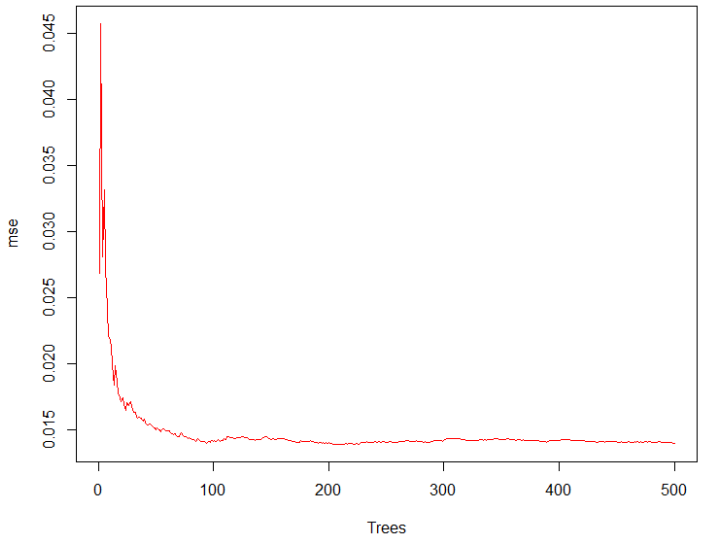

Figura 8.5: MSE en función del número de arboles para el dataset original.

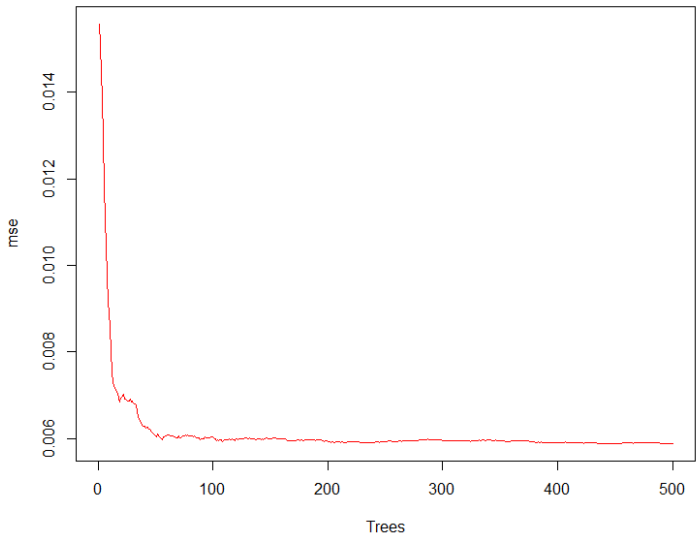

Figura 8.6: MSE en función del número de arboles para el dataset coseno.

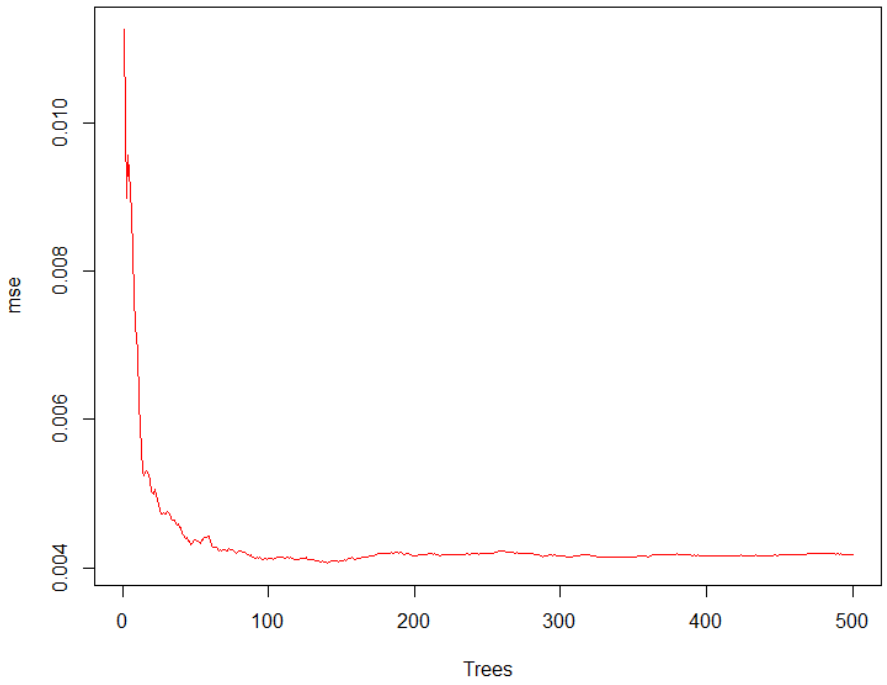

Figura 8.7: MSE en función del número de arboles para el dataset jaccard.

Se observa en las gráficas anteriores que a partir de 100 árboles el mse queda estable. Esto significa que el modelo no mejora de manera significativa su precisión, pero la ventaja que aporta es la aleatoriedad en la creación de los árboles por parte de Random Forest, lo que favorece la existencia de poca correlación entre ellos. Todo esto hace que sea muy difícil que se produzca sobreentrenamiento, es decir, que dicho algoritmo sea capaz de realizar predicciones satisfactorias con datos diferentes a los de entrenamiento. Finalmente, destacar que el coste computacional aumenta a partir del número de árboles mencionado, lo cual es otra característica a tener en cuenta de cara a la creación de los modelos. Más adelante, en la etapa de entrenamiento y resultados, veremos los valores de ntree óptimos seleccionados para la creación de cada uno de los modelos Random Forest.

8.2.2 Validación

La validación del modelo es el proceso que busca la independencia entre el conjunto de entrenamiento y el de prueba. Esto proporciona la capacidad de generalización de un modelo entrenado. Como estamos utilizando Random Forest no necesitamos llevar a cabo por nosotros mismos la validación, debido a que el propio algoritmo se encarga de ello. Además, como ya se comentó, Random Forest utiliza bootstraping para construir los árboles en función de un número X de observaciones con reemplazamiento, por lo que cada árboles emplea 2/3 del

dataset siendo el 1/3 restante utilizado para validar el modelo (OOB).

Se han analizado dos técnicas de validación muy comunes que son:

- **Hold-out**: se separa el conjunto de datos en dos para asegurarnos que el algoritmo de machine learning aprende y no simplemente memoriza, lo que implica que el modelo resultante no sufra un sobreajuste. Esta técnica destaca por la rapidez que proporciona en cuanto a la ejecución. Como desventajas destaca el hecho de que se divide nuestro conjunto de datos, por lo que en datasets pequeños podría afectar gravemente a la calidad del modelo construido y, por otra parte está la desventaja de que el porcentaje de datos utilizado en cada parte influye notablemente en la obtención del resultado. Lo normal es utilizar un 70% de los datos para entrenamiento y lo restante para pruebas.

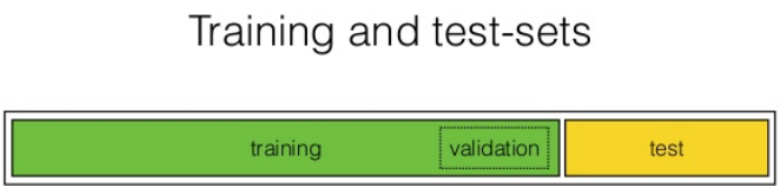

Figura 8.8: Método Hold-out

- **Cross-validation**: debido a las carencias que tiene la técnica anterior se crea la validación cruzada, también conocida como k-fold cross-validation. Lo que hace es dividir el dataset en K subconjuntos y realizar K iteraciones. En cada iteración se utiliza como parte de entrenamiento K-1 subconjuntos siendo el restante utilizado para llevar a cabo las pruebas. Todo esto hace que sea muy costoso desde el punto de vista computacional, ya que hay que llevar a cabo el entrenamiento y la validación K veces. Como ya hemos dicho en la sección de hiperparametrización, esta ha sido la técnica elegida para calcular los mtry óptimos sobre los datasets correspondientes.

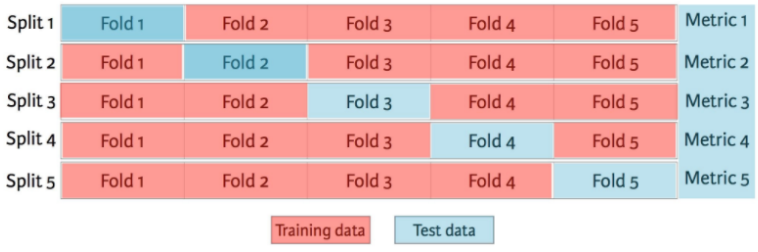

Figura 8.9: Método Cross-validation

8.2.3 Calidad de los Modelos

Esta parte del trabajo va a tratar sobre los diversos modelos de Random Forest que hemos construido, así como en el análisis de los resultados obtenidos para cada uno de los conjuntos de datos construidos.

En primer lugar, vamos a proceder a visualizar los tres modelos construidos con sus hiperparámetros óptimos. En cuanto al número de árboles se han probado diversas combinaciones hasta lograr obtener el mejor resultado:

```
originalModel <- randomForest(x = trainData[, 2:6],
                              y = trainData[, 1],
                              ntree = 200,
                              mtry = 3)
```

Figura 8.10: Modelo construido a partir del dataset original.

```
 randomForest(x = trainData[, 2:6], y = trainData[, 1], ntree = 200,      mtry = 3)
               Type of random forest: regression
                     Number of trees: 200
No. of variables tried at each split: 3

          Mean of squared residuals: 0.009117848
                    % Var explained: 94.54
```

Figura 8.11: Resultados de la construcción del modelo original.

```
cosineModel <- randomForest(x = trainDataC[, 2:14],
                            y = trainDataC[, 1],
                            ntree = 200,
                            mtry = 6)
```

Figura 8.12: Modelo construido a partir del dataset coseno.

```
 randomForest(x = trainDataC[, 2:14], y = trainDataC[, 1], ntree = 200,      mtry = 6)
               Type of random forest: regression
                     Number of trees: 200
No. of variables tried at each split: 6

          Mean of squared residuals: 0.005928573
                    % Var explained: 96.45
```

Figura 8.13: Resultados de la construcción del modelo coseno.

```
jaccardModel <- randomForest(x = trainDataJ[, 2:14],
                             y = trainDataJ[, 1],
                             ntree = 300,
                             mtry = 5)
```

Figura 8.14: Modelo construido a partir del dataset jaccard.

```
randomForest(x = trainDataJ[, 2:14], y = trainDataJ[, 1], ntree = 300,      mtry = 5)
               Type of random forest: regression
                     Number of trees: 300
No. of variables tried at each split: 5

          Mean of squared residuals: 0.00430724
                    % Var explained: 97.42
```

Figura 8.15: Resultados de la construcción del modelo jaccard.

Una vez vistos los modelos construidos y algunos de los resultados arrojados por este, podemos apreciar que el mejor MSE obtenido por los tres es para jaccard, ya que es el que ha obtenido un valor más bajo frente al resto.

Por otro lado contamos con la gráfica que relaciona MSE frente a ntree, la cual hemos visto anteriormente pero ahora se hizo con el número de árboles óptimo. A continuación podemos ver como el modelo jaccard es el que más destaca sobre los otros a pesar de que la diferencia tampoco es radical:

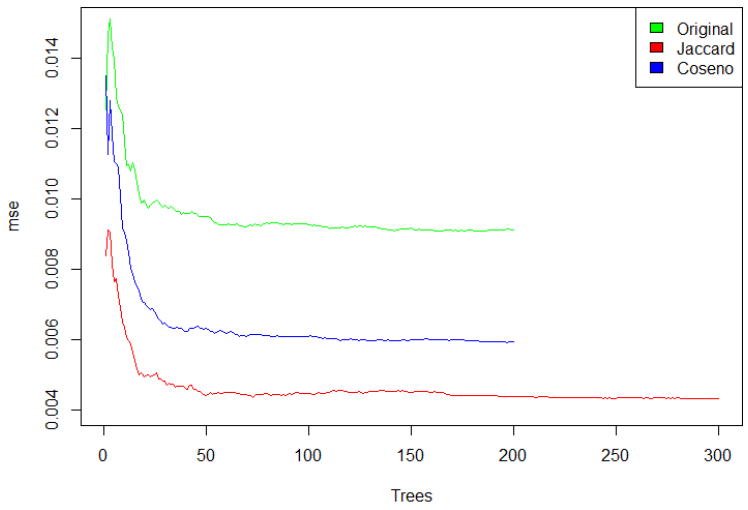

Figura 8.16: MSE en función del ntree óptimo.

También tenemos que hablar sobre la métrica de R^2, la cual es una de las determinantes de la calidad de un modelo. Su medición por lo normal se encuentra entre 0 y 1 mostrando una mayor calidad cuanto más cerca de 1 esté. En la figura 8.17 se muestra una comparativa de los valores de R^2 obtenidos para cada uno de los modelos, sobre los cuales destaca el que utiliza las variables creadas a partir de los resultados del índice jaccard:

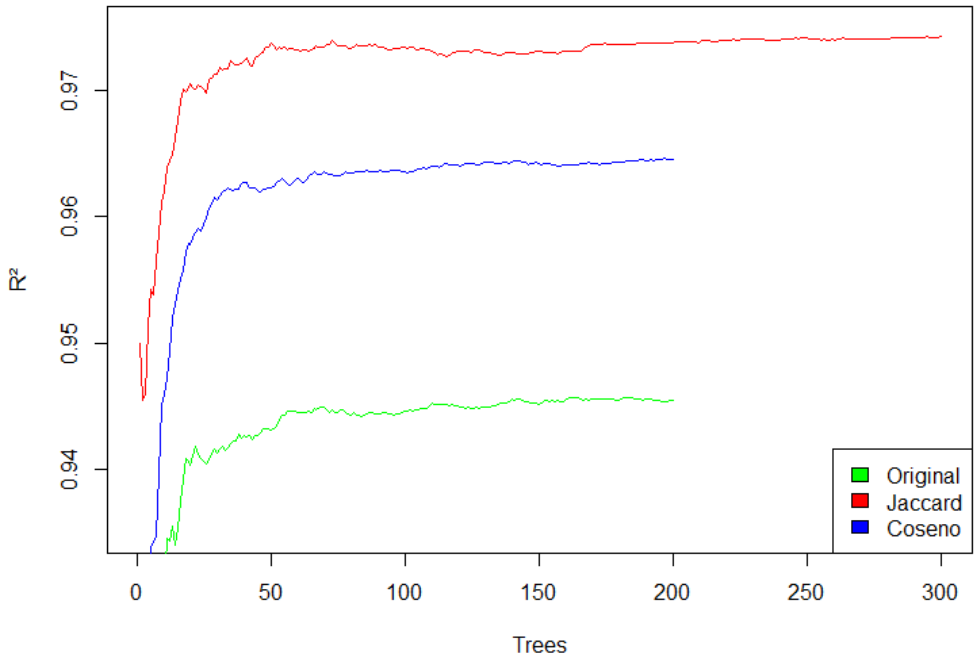

Figura 8.17: Métrica R^2 para cada modelo.

Ya para concluir con Random Forest vamos a ver las predicciones realizadas por cada uno de los modelos, así como visualizar los resultados arrojados por las métricas más usuales en machine learning:

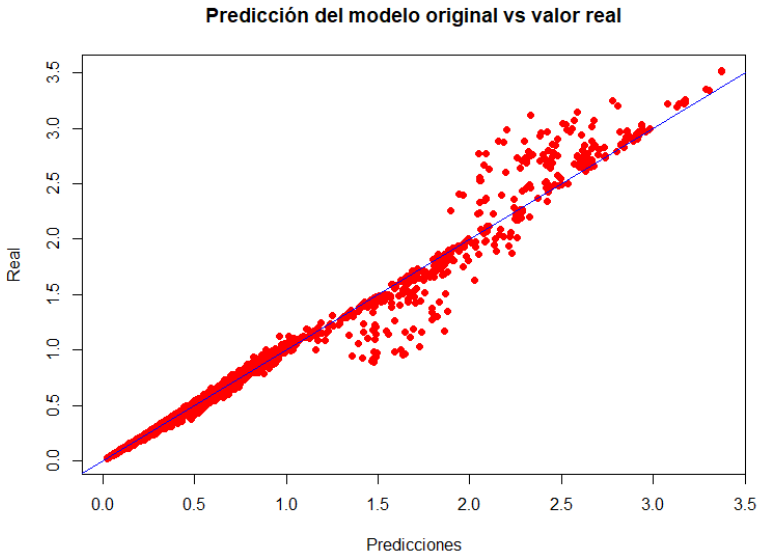

Figura 8.18: Predicciones del modelo original.

```
MSE modelo original: 0.00911784775108463
RMSE modelo original: 0.0954874219522374
MAE modelo original: 0.0203708527194177
R² modelo original: 0.942942996087782
```

Figura 8.19: Resultados utilizadas para medir la calidad del modelo original.

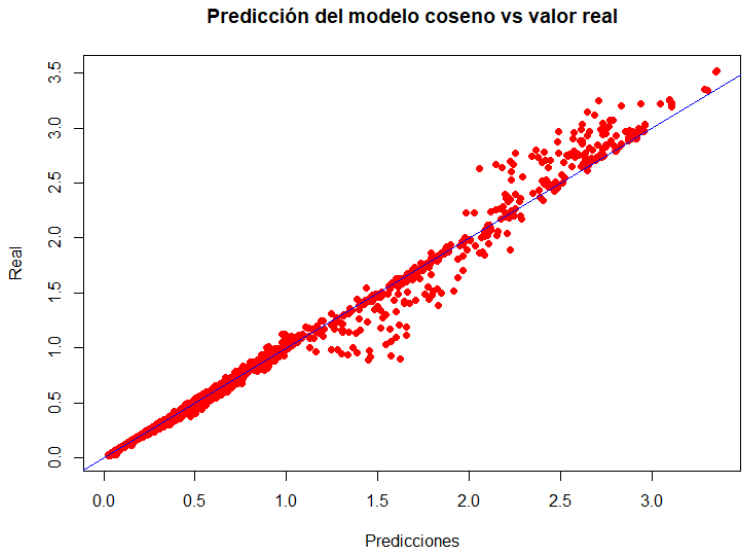

Figura 8.20: Predicciones del modelo coseno.

```
MSE modelo coseno: 0.00592857309067558
RMSE modelo coseno: 0.076997227811627
MAE modelo coseno: 0.018322948609096
R² modelo coseno: 0.961523282467064
```

Figura 8.21: Resultados utilizadas para medir la calidad del modelo coseno.

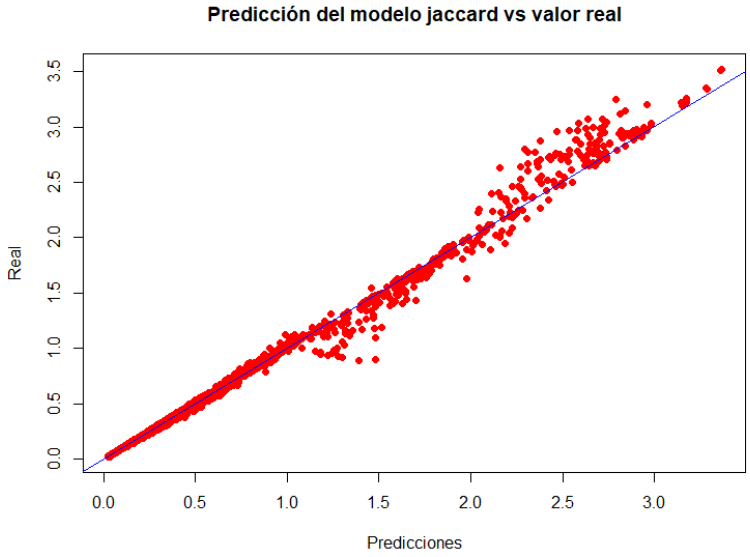

Figura 8.22: Predicciones del modelo jaccard.

```
MSE modelo jaccard: 0.00430724003962586
RMSE modelo jaccard: 0.065629566809677
MAE modelo jaccard: 0.0144700684839065
R² modelo jaccard: 0.972551472892978
```

Figura 8.23: Resultados utilizadas para medir la calidad del modelo jaccard.

Como bien se puede ver en los resultados, el modelo creado a partir del dataset original es el que peores resultados arroja, luego sigue el que se construyó a partir de los resultados obtenidos por la distancia coseno, mejorando ligeramente los resultados obtenidos y por último, siendo el mejor, el modelo jaccard. En este se aprecia una mejora considerable en la calidad de las predicciones realizadas frente a las del modelo original.

8.3 K-NN

K-NN es un algoritmo de tipo supervisado que puede ser usado tanto para clasificar como para predecir. En este caso vamos a utilizarlo para predecir una variable objetivo. Lo primero que debemos hacer es elegir los hiperparámetros adecuados, así como hicimos con Random Forest.

8.3.1 Hiperparametrización

Esta vez nos encontramos con que el único hiperparámetro que tenemos es la K. Para entender lo que es, necesitamos saber cómo funciona K-NN. Lo que hace es que a partir de una serie de variables dadas busca el valor más próximo calculado mediante alguna distancia (Euclídea, etc) y una vez encontrado, esa será la predicción resultante. Este caso solo se da para K=1 ya que cuando tenemos una K mayor se buscan los primeros K resultados más parejos al dado y luego se elige por consenso entre todos el más adecuado.

En este trabajo para ser capaces de elegir una K óptima hemos realizado un proceso de validación similar al que llevamos a cabo en Random Forest, el cual se basa en la validación cruzada anteriormente explicada. También destaca que escalamos los valores de los datasets mediante la función scale() de R para conseguir una estandarización de los valores de nuestros conjuntos de datos, y para nuestro caso conseguimos mejores resultados [27]. Se han probado con valores de K entre 1 y 200 como podemos ver en las siguientes figuras:

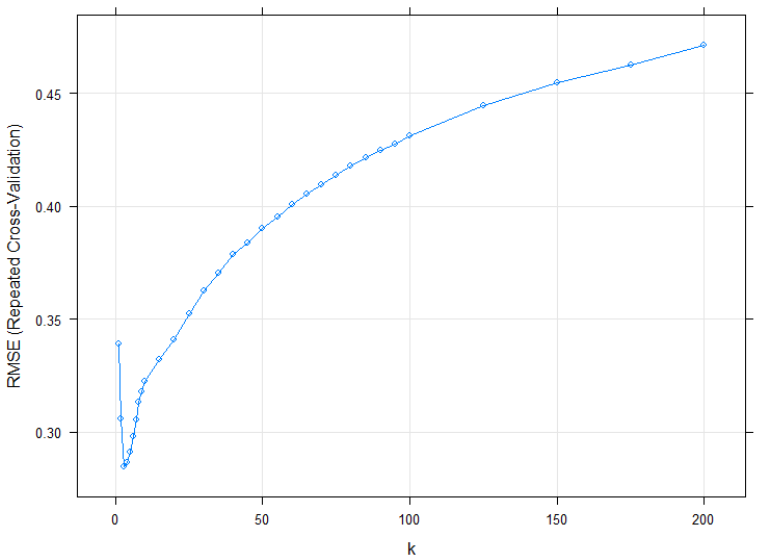

Figura 8.24: Gráfica del RMSE frente a las K para el modelo original.

```
k-Nearest Neighbors

13248 samples
    5 predictor

No pre-processing
Resampling: Cross-Validated (5 fold, repeated 1 times)
Summary of sample sizes: 10600, 10598, 10600, 10597, 10597
Resampling results across tuning parameters:

  k    RMSE       Rsquared   MAE
    1  0.3389300  0.8877241  0.07554405
    2  0.3061565  0.9067020  0.08428279
    3  0.2846123  0.9192060  0.09214363
    4  0.2867795  0.9186589  0.10309923
    5  0.2911277  0.9160572  0.11413283
    6  0.2981149  0.9120405  0.12538374
    7  0.3057265  0.9075431  0.13627567
    8  0.3135132  0.9027451  0.14544927
    9  0.3177559  0.9002283  0.15282953
   10  0.3226504  0.8972810  0.15945001
   15  0.3321454  0.8920381  0.18105639
   20  0.3410145  0.8872716  0.19607116
   25  0.3524815  0.8804690  0.20896836
   30  0.3627555  0.8740769  0.21871426
   35  0.3702854  0.8692583  0.22692661
   40  0.3788907  0.8636175  0.23485503
   45  0.3839657  0.8602291  0.24096254
   50  0.3901163  0.8560875  0.24682979
   55  0.3951018  0.8525234  0.25198955
   60  0.4010284  0.8481936  0.25663952
   65  0.4056473  0.8447771  0.26030499
   70  0.4095272  0.8419388  0.26387506
   75  0.4134935  0.8389874  0.26714262
   80  0.4179321  0.8356356  0.27035471
   85  0.4216222  0.8327039  0.27337894
   90  0.4247060  0.8303895  0.27611357
   95  0.4275262  0.8283909  0.27826180
  100  0.4311528  0.8257748  0.28059389
  125  0.4444608  0.8163841  0.29005489
  150  0.4545350  0.8093392  0.29727670
  175  0.4627303  0.8040370  0.30375147
  200  0.4714554  0.7980511  0.30960811

RMSE was used to select the optimal model using the smallest value.
The final value used for the model was k = 3.
```

Figura 8.25: Mejor K para el modelo original.

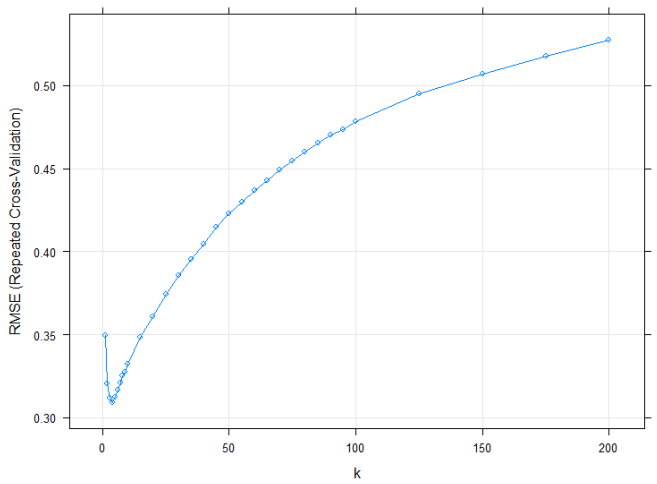

Figura 8.26: Gráfica del RMSE frente a las K para el modelo coseno.

```
k-Nearest Neighbors

13248 samples
   11 predictor

No pre-processing
Resampling: Cross-Validated (5 fold, repeated 1 times)
Summary of sample sizes: 10597, 10600, 10599, 10599, 10597
Resampling results across tuning parameters:

  k    RMSE       Rsquared   MAE
    1  0.3493994  0.8792595  0.08108763
    2  0.3203605  0.8974762  0.08945581
    3  0.3118861  0.9030826  0.10112382
    4  0.3089386  0.9052357  0.11309589
    5  0.3125780  0.9030777  0.12542462
    6  0.3169105  0.9007830  0.13718957
    7  0.3212333  0.8982826  0.14699343
    8  0.3255609  0.8960911  0.15620160
    9  0.3274357  0.8952163  0.16362118
   10  0.3324200  0.8923624  0.17086813
   15  0.3486009  0.8832818  0.19466526
   20  0.3609351  0.8764198  0.21168607
   25  0.3741316  0.8685910  0.22599082
   30  0.3859442  0.8609683  0.23704184
   35  0.3952489  0.8547775  0.24636991
   40  0.4047090  0.8484097  0.25455401
   45  0.4146175  0.8412160  0.26261497
   50  0.4228245  0.8354444  0.26866206
   55  0.4300789  0.8304651  0.27458338
   60  0.4371126  0.8254832  0.27968764
   65  0.4430830  0.8211746  0.28417799
   70  0.4490492  0.8167321  0.28812770
   75  0.4547899  0.8125669  0.29200194
   80  0.4603224  0.8082333  0.29561701
   85  0.4654444  0.8038317  0.29916208
   90  0.4700984  0.7999449  0.30221448
   95  0.4737881  0.7972627  0.30515348
  100  0.4781387  0.7940919  0.30775253
  125  0.4952097  0.7804071  0.31870054
  150  0.5068040  0.7725072  0.32565504
  175  0.5175798  0.7658723  0.33150123
  200  0.5276770  0.7597910  0.33654818

RMSE was used to select the optimal model using the smallest value.
The final value used for the model was k = 4.
```

Figura 8.27: Mejor K para el modelo coseno.

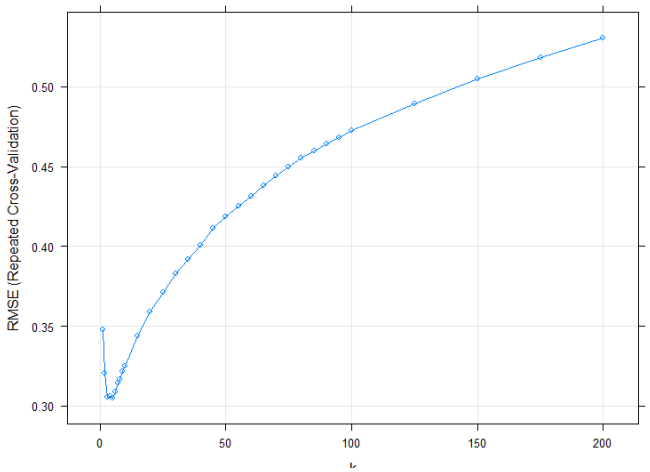

Figura 8.28: Gráfica del RMSE frente a las K para el modelo jaccard.

```
k-Nearest Neighbors

13248 samples
   12 predictor

No pre-processing
Resampling: Cross-Validated (5 fold, repeated 1 times)
Summary of sample sizes: 10599, 10598, 10598, 10599, 10598
Resampling results across tuning parameters:

  k    RMSE       Rsquared   MAE
    1  0.3479999  0.8818564  0.07716978
    2  0.3203833  0.8981884  0.08631387
    3  0.3055049  0.9072102  0.09534430
    4  0.3060883  0.9070485  0.10710190
    5  0.3046421  0.9081537  0.11904228
    6  0.3088379  0.9058965  0.13132480
    7  0.3142425  0.9030552  0.14305802
    8  0.3165300  0.9017698  0.15218083
    9  0.3218378  0.8985537  0.16106919
   10  0.3251401  0.8965013  0.16786242
   15  0.3440184  0.8852398  0.19241788
   20  0.3587147  0.8765163  0.20969463
   25  0.3713335  0.8690967  0.22389275
   30  0.3826450  0.8616425  0.23497907
   35  0.3918304  0.8555961  0.24374293
   40  0.4004451  0.8499557  0.25173950
   45  0.4110926  0.8423582  0.25983085
   50  0.4183338  0.8373750  0.26600672
   55  0.4254694  0.8325629  0.27101142
   60  0.4314641  0.8287247  0.27572717
   65  0.4380271  0.8240037  0.28005352
   70  0.4440254  0.8199206  0.28398583
   75  0.4498894  0.8159075  0.28756010
   80  0.4553777  0.8120087  0.29099950
   85  0.4596097  0.8093038  0.29397701
   90  0.4640440  0.8061777  0.29692104
   95  0.4683794  0.8032801  0.29984166
  100  0.4725002  0.8004212  0.30243436
  125  0.4892253  0.7898451  0.31306712
  150  0.5051288  0.7798522  0.32222838
  175  0.5184592  0.7718660  0.32877206
  200  0.5307762  0.7657232  0.33512523

RMSE was used to select the optimal model using the smallest value.
The final value used for the model was k = 5.
```

Figura 8.29: Mejor K para el modelo jaccard.

8.3.2 Validación

En las gráficas anteriores hemos podido apreciar los resultados obtenidos para el análisis de cuál es la K óptima. Todo esto ha sido posible gracias al uso de la técnica de validación cruzada, tal y como se ha hecho en la sección de Random Forest.

Como ya hemos mencionado en apartados anteriores, no vamos a realizar una validación propia debido a que el propio algoritmo de K-NN, implementado en la librería FNN de R, utiliza validación cruzada por sí mismo como también ocurría con Random Forest.

8.3.3 Calidad de los Modelos

En esta sección mostramos los resultados de las diferentes métricas obtenidas por cada uno de los modelos construidos, que nos permiten validar la calidad de los mismos. Las siguientes gráficas nos muestran cómo de buenas son las predicciones realizadas sobre el conjunto de entrenamiento y los resultados de las métricas más relevantes para cada uno:

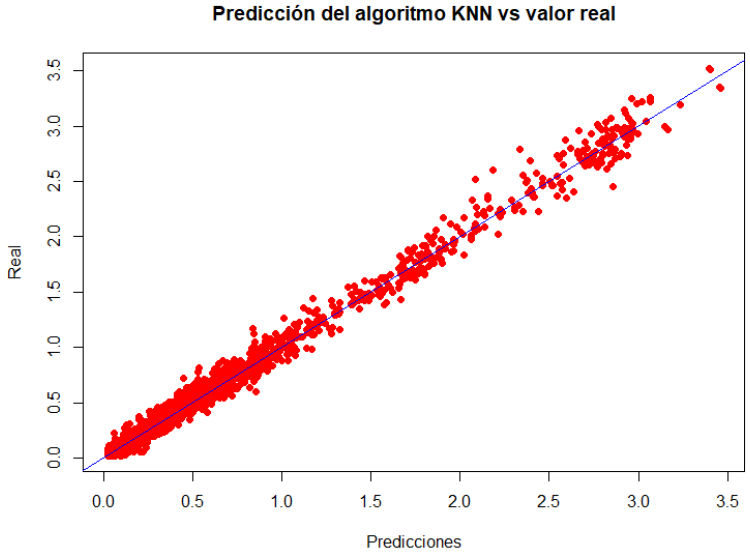

Figura 8.30: Predicciones realizas por el modelo original.

```
MSE modelo original: 0.00101409413169882
RMSE modelo original: 0.0318448446643851
 MAE modelo origina: 0.0145312760333829
  R² modelo original: 0.993931385011242
```

Figura 8.31: Resultados de las métricas para el modelo original.

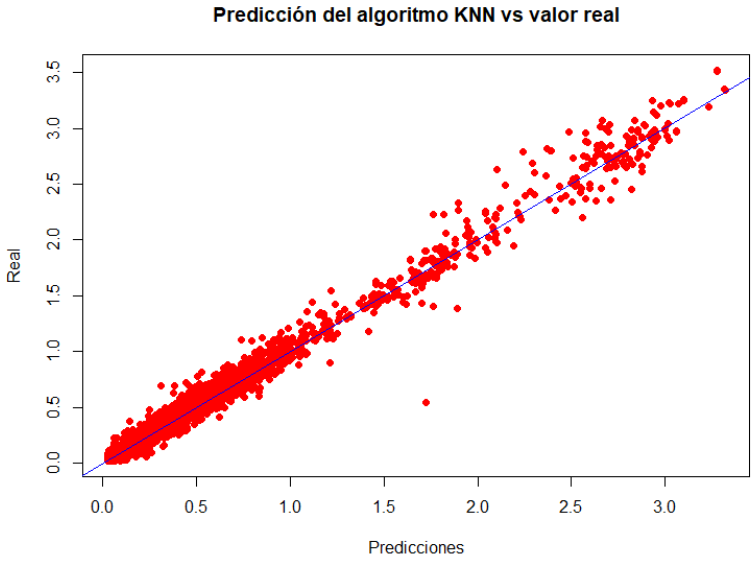

Figura 8.32: Predicciones realizas por el modelo coseno.

```
MSE modelo coseno: 0.00166753547124327
RMSE modelo coseno: 0.0408354682995465
MAE modelo coseno: 0.0184416130560806
R² modelo coseno: 0.990021014382442
```

Figura 8.33: Resultados de las métricas para el modelo coseno.

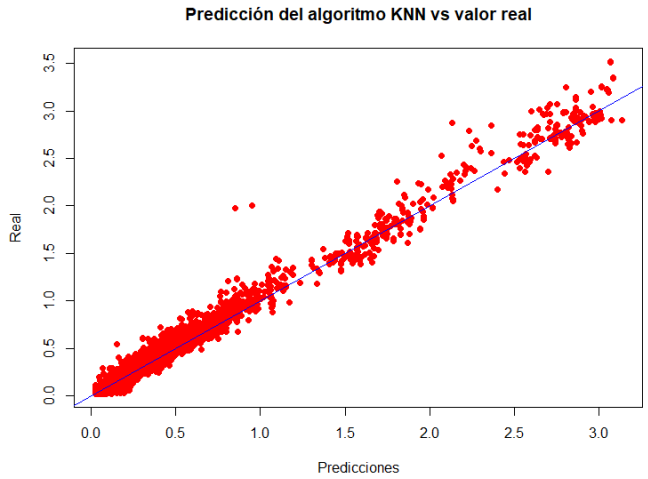

Figura 8.34: Predicciones realizas por el modelo jaccard.

```
MSE modelo jaccard: 0.00262758511234087
RMSE modelo jaccard: 0.0512599757348837
 MAE modelo jaccard: 0.029571227543295
  R² modelo jaccard: 0.984275816318673
```

Figura 8.35: Resultados de las métricas para el modelo jaccard.

Como bien se muestra en las figuras, el modelo que mejores resultados consiguió fue el modelo original, seguido por el modelo coseno y el modelo jaccard. Esta vez no se ha conseguido una mejora mediante la utilización de los atributos construidos a partir de los resultados arrojados por la distancia coseno y el índice jaccard. En secciones posteriores veremos que tal son las predicciones de estos modelos frente a datos externos al conjunto de entrenamiento, para así poder saber si aprendió correctamente o solo memorizó, pero a priori obtenemos unos resultados bastante buenos.

8.4 RouteNet

A día de hoy, los modelos de redes neuronales pueden usarse para construir modelos de red con una buena precisión. Como ya sabemos, los operadores de redes siguen sin contar con modelos de red que sean capaces de hacer predicciones de calidad por lo que hace no mucho que se propuso un modelo de red neuronal gráfica (GNN) llamado RouteNet [28]. Esta es una alternativa desarrollada por un equipo para resolver las carencias que estos operadores presentan y que demostró ser capaz de construir modelos a partir de los datos de enrutamiento, tráfico, etc, de las topologías de las redes y además, arrojó buenos resultados sobre escenarios de red totalmente diferentes a los vistos en los conjuntos de entrenamiento [29]. Este trabajo se realizó para conseguir alternativas a este planteamiento y así también ser capaces de realizar una comparativa entre las diversas técnicas empleadas. En la siguiente imagen podemos ver como es la arquitectura de RouteNet:

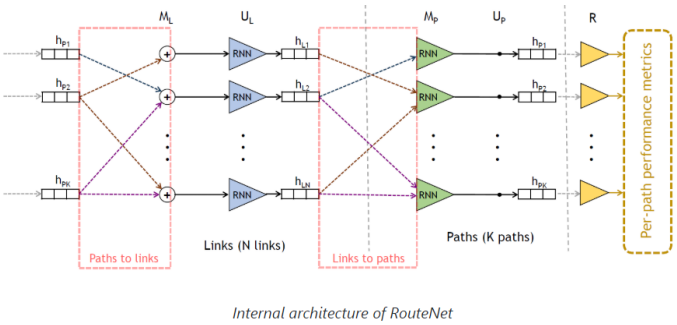

Figura 8.36: Arquitectura interna de RouteNet

Fuente: github.com/knowledgedefinednetworking/demo-routenet

Ahora vamos a comentar los tipos más importantes de redes neuronales de una forma resumida y también a llevar a cabo un análisis de las características más destacadas. Los tipos de redes neuronales según la topología de red son [30]:

- **Redes neuronales Monocapa**: se corresponde con la red neuronal más simple. Está compuesta por una capa de neuronas que proyectan las entradas a una capa de neuronas de salida donde se realizan los diferentes cálculos.
- **Redes neuronales Multicapa**: es una generalización de la red neuronal monocapa. La diferencia reside en que mientras la red neuronal monocapa está compuesta por una capa de neuronas de entrada y una capa de neuronas de salida, esta dispone de un conjunto de capas intermedias (capas ocultas) entre la capa de entrada y la de salida.
- **Redes neuronales Convolucionales**: en este tipo de red, cada neurona no se une con todas y cada una de las capas siguientes sino que solo con un subgrupo de ellas (se especializa). Con esto se consigue reducir el número de neuronas necesarias y la complejidad computacional necesaria para su ejecución.
- **Redes neuronales Recurrentes**: estas no tienen una estructura de capas, sino que permiten conexiones arbitrarias entre las neuronas. Integran bucles de realimentación para hacer posible que la información persista durante algunos ciclos ó épocas de entrenamiento mediante la integración de los resultados de las capas de salida en las capas de entrada. Básicamente son como una red formada por copias de sí misma, que se mandan información a las copias sucesoras.

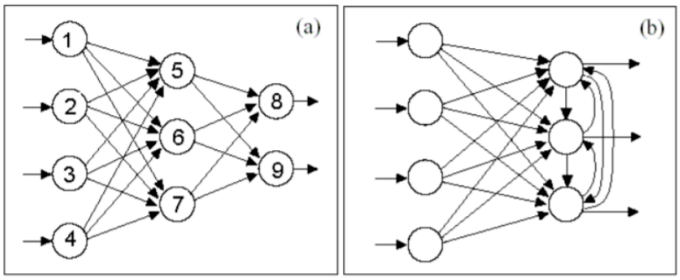

Figura 8.37: Conexiones hacia delante (a) frente a conexiones recurrentes (b).

Fuente: researchgate.net

Las principales ventajas de las RNN frente a los otros tipos de redes neuronales son:

- Se tratan los datos secuenciales de una forma eficiente.
- No mezclan información entre ejecuciones sino que registran las salidas de la copia anterior como entradas de la actual.
- Pueden tratar secuencias de datos muy largas elemento a elemento.

En este tipo de redes (RNN), si la secuencia es lo bastante larga, presentan problemas en cuanto a trasladar la información de etapas anteriores a posteriores. Suele ocurrir en las capas más tempranas [31].

Para solventar este problema de memoria a corto plazo se crearon dos tipos de redes neuronales recurrentes de las cuales vamos a hablar a continuación [32]:

- **LSTM**: también conocidas como Long Short Term Memory, utilizan un mecanismo para decidir que información se va a almacenar y cual no. Está compuesta por tres celdas de memoria capaces de mantener dicha información por un lapso de tiempo largo o corto, por lo que es capaz de recordar las cosas importantes. La primera es la puerta de entrada que decide cuándo meter información nueva en memoria. La segunda es la puerta de olvido que decide qué información existente en memoria se debe eliminar. La tercera es la puerta de salida, encargada de determinar cuándo se utiliza la información contenida en la propia celda.
- **GRU**: también conocidas como Gated Recurrent Unit, constituyen una simplificación de las LSTM, ya que se compone de solo dos puertas. La puerta de actualización que decide qué información nueva entra y cuál existente se elimina, y la puerta de reinicio que decide qué información del estado anterior se reinicia. Todo esto provoca que se

tengan que realizar menos operaciones, por lo que su eficiencia es mucho mayor con respecto al entrenamiento y ejecución.

Analizamos las capas por las que está formada la red neuronal de RouteNet, siendo estas las mostradas en la siguiente imagen:

```
class ComnetModel(tf.keras.Model):
    def __init__(self,hparams, output_units=1, final_activation=None):
        super(ComnetModel, self).__init__()
        self.hparams = hparams

        self.edge_update = tf.keras.layers.GRUCell(hparams.link_state_dim)
        self.path_update = tf.keras.layers.GRUCell(hparams.path_state_dim)


        self.readout = tf.keras.models.Sequential()

        self.readout.add(keras.layers.Dense(hparams.readout_units,
                activation=tf.nn.selu,
                kernel_regularizer=tf.contrib.layers.l2_regularizer(hparams.l2)))
        self.readout.add(keras.layers.Dropout(rate=hparams.dropout_rate))
        self.readout.add(keras.layers.Dense(hparams.readout_units,
                activation=tf.nn.selu,
                kernel_regularizer=tf.contrib.layers.l2_regularizer(hparams.l2)))
        self.readout.add(keras.layers.Dropout(rate=hparams.dropout_rate))

        self.readout.add(keras.layers.Dense(output_units,
                kernel_regularizer=tf.contrib.layers.l2_regularizer(hparams.l2_2),
                activation = final_activation ) )
```

Figura 8.38: Capas que forman RouteNet.

Podemos observar como se compone de dos capas GRU y una capa Sequential. Esta última consta de capas Dense, que son básicamente capas ocultas que utilizan el mecanismo de Dropout [33], una técnica para evitar el sobreajuste. Esta consiste en llevar a cero de forma aleatoria diversos valores del vector de entrada en cada iteración del entrenamiento. De esta manera se consigue que las neuronas no le den demasiada importancia a un conjunto pequeño de la salida de una capa anterior dejando de lado otros valores que pueden servir para realizar una mejor predicción. En la imagen que vemos a continuación se ve claramente el funcionamiento de dicho mecanismo:

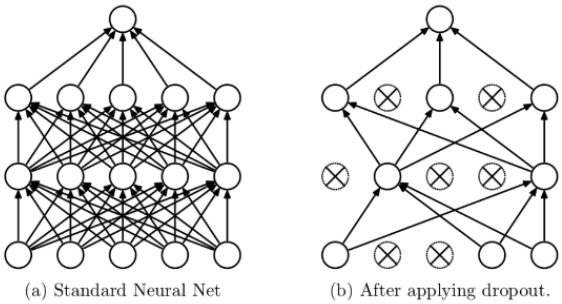

Figura 8.39: Mecanismo de Dropout.

Fuente: mc.ai

8.4.1 Hiperparametrización

Los hiperparámetros que han sido utilizados para la creación del modelo de RouteNet se explican a continuación:

- **Learning rate:** es un parámetro de ajuste que determina el tamaño del paso en cada iteración mientras se mueve hacia una función de pérdida mínima. Dado que influye en la medida en que la información recién adquirida anula la información antigua, se podría considerar como la velocidad a la que un modelo aprende. De manera general, el mejor learning rate es el que va disminuyendo a medida que pasan los ciclos (epochs), ya que así permite un aprendizaje rápido al inicio y a medida que avanza el proceso se disminuye para conseguir que la función de perdida sea la menor posible [34]. En RouteNet se utiliza la función que podemos ver a continuación para reducir progresivamente dicho ratio de aprendizaje:

```
1                    tf.train.exponential_decay(params.learning_rate,
2                              tf.train.get_global_step(), 82000,
3                              0.8, staircase=True)
4
```

Listing 8.1: Función utilizada para el Learning Rate.

- **Batch size:** define el número de muestras que se propagarán a través de la red. Por ejemplo, si tenemos 1000 muestras de entrenamiento y establecemos un tamaño de lote igual a 100, el algoritmo toma las primeras 100 muestras del conjunto de datos de entrenamiento y entrena la red. Luego, toma las segundas 100 muestras y así hasta que

las propague todas. En general se suelen utilizar tamaños de lote pequeños ya que requieren menos memoria, aunque los resultados arrojados son menos precisos que para lotes mayores [35]. En nuestro caso se utiliza un tamaño de lote de 32.

- **Epochs:** hace referencia a un ciclo, es decir, cuándo el dataset pasa por completo por la red de neuronas. Se ha decidido emplear diversos valores para las épocas para así poder realizar el entrenamiento de RouteNet y observar las diferencias en los resultados obtenidos. A continuación se contrastan algunas métricas con el número de ciclos utilizados:

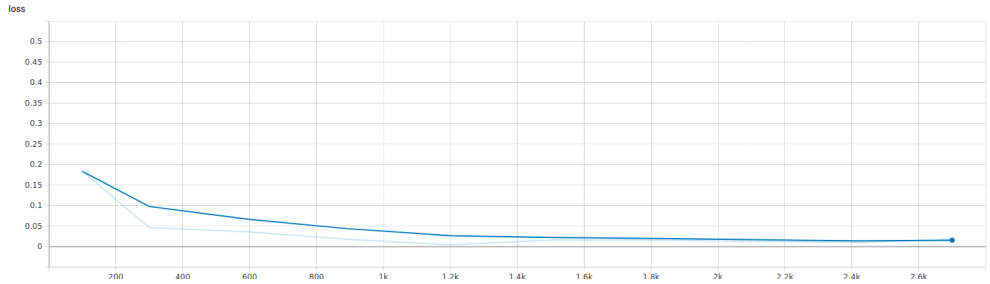

Figura 8.40: Pérdida en función del número de épocas utilizado.

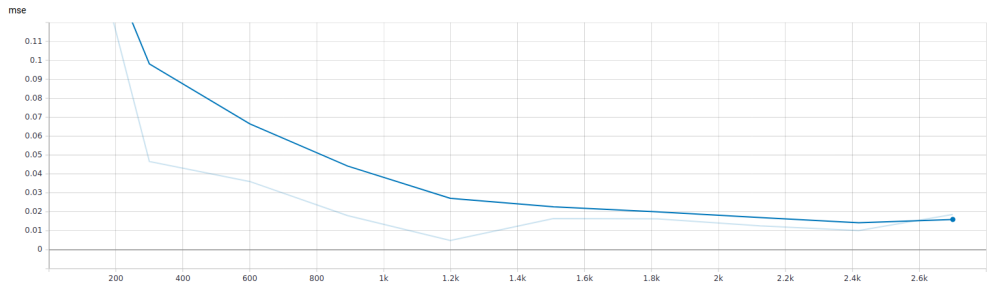

Figura 8.41: MSE en función del número de épocas utilizado.

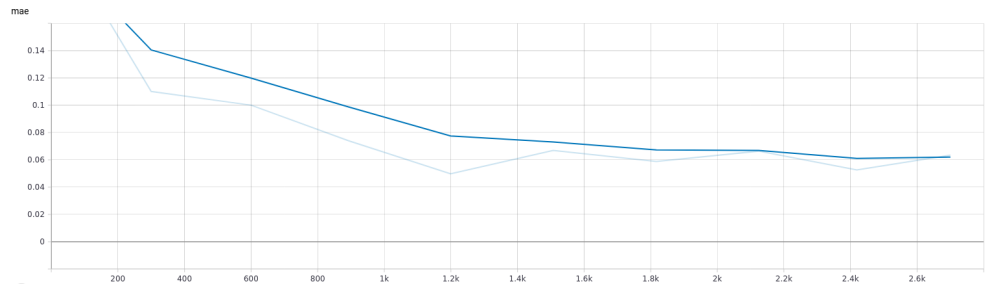

Figura 8.42: MAE en función del número de épocas utilizado.

El número óptimo consideramos que está alrededor de unas 2700 épocas, ya que es ahí donde se consiguen los mejores resultados. Por encima de las 5000 no se consiguieron mejoras además de que la carga computacional aumenta considerablemente.

Finalmente comentar que la pérdida inicial es bastante elevada y, a medida que aumentamos las épocas se reduce considerablemente, por lo que podemos considerar que el modelo está entrenado.

8.5 Calidad del Modelo

Una vez ajustados los hiperparámetros vamos a ver como de buenas son las predicciones realizadas por el modelo óptimo. En las siguiente figuras podemos ver los resultados obtenidos por este:

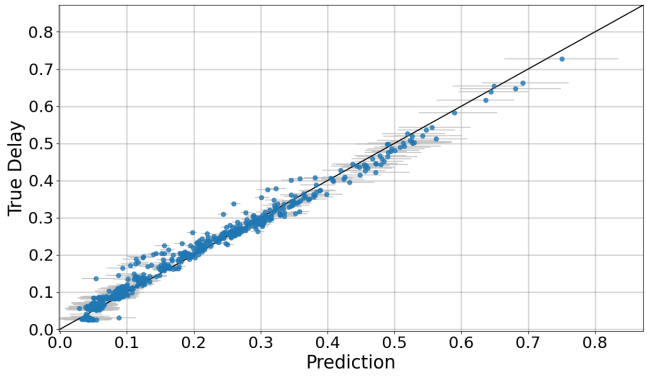

Figura 8.43: Predicción obtenida por el modelo óptimo.

```
Número de iteraciones -> 2700:

- global_step = 2700,
- label/mean = -0.04878619,
- loss = 0.018586375,
- mae = 0.06349329,
- mre = 0.02064778,
- mse = 0.018586373,
- prediction/mean = -0.06764504,
- rho = 0.9926284S
```

Figura 8.44: Resultados de las métricas más relevantes para el modelo óptimo.

8.5.1 Conclusiones

Se han obtenido unos resultados bastante sólidos para todos los mecanismos de ML utilizados. A partir de esto podemos considerar que el modelo generado por RouteNet podría ser útil en cuanto a la creación de modelos con capacidad de aprendizaje para los operadores de red, aunque su coste computacional es bastante grande. También hemos visto algoritmos como K-NN y Random Forest que son capaces de obtener unos resultados superiores frente a los arrojados por RouteNet. Además destaca que su coste computacional es mucho menor, por lo que a priori se seleccionan como primera opción en cuanto a la construcción de modelos de red livianos para los operadores de estas.

Capítulo 9

Evaluación

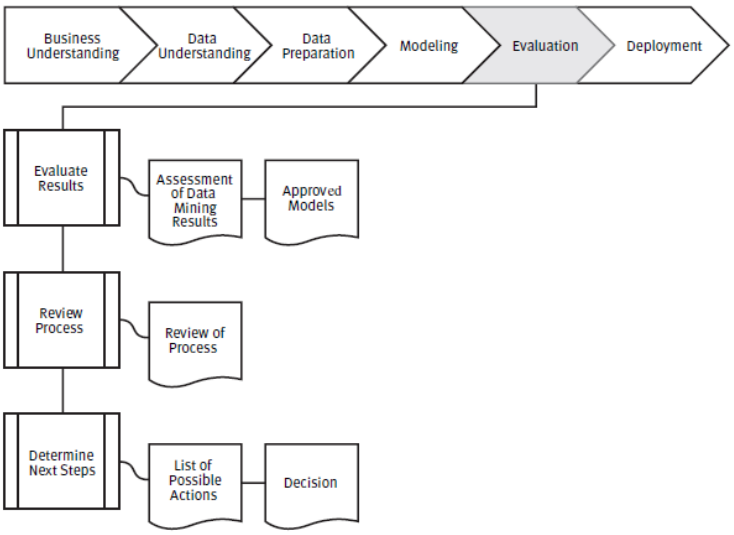

Figura 9.1: CRISP-DM (5)

Una vez que hemos analizado la calidad de los modelos construidos para cada una de las técnicas de machine learning que se utilizaron, vamos a proceder a probar con qué calidad predicen dichos modelos sobre un conjunto de datos independiente a los de entrenamiento. Nos basaremos en los resultados que arrojen las métricas utilizadas anteriormente, así como en las gráficas de las predicciones.

En lo que respecta a los conjuntos de evaluación utilizados para los modelos coseno y jaccard, debemos decir que se han construido los mismo atributos que en el conjunto de datos de entrenamiento. Estos son valores estadísticos obtenidos a partir de los resultados dados por la distancia coseno y el índice jaccard.

9.1 Random Forest

En las siguientes imágenes vamos a ver como han sido las predicciones obtenidas sobre el conjunto de datos independiente al de entrenamiento, así como los resultados de las métricas más relevantes:

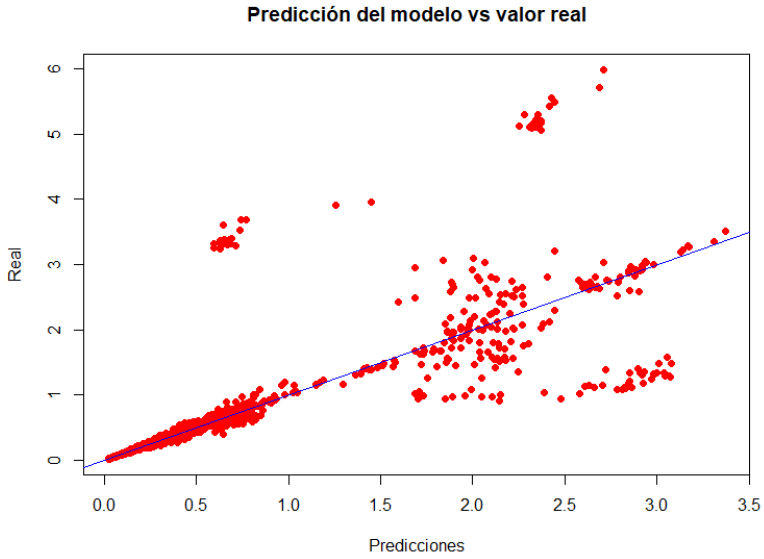

Figura 9.2: Modelo construido a partir del dataset original.

```
MSE modelo original: 0.0833219147866516
RMSE modelo original: 0.288655356414274
MAE modelo original: 0.0542695647721788
R² modelo original: 0.754338833324593
```

Figura 9.3: Resultados del modelo original.

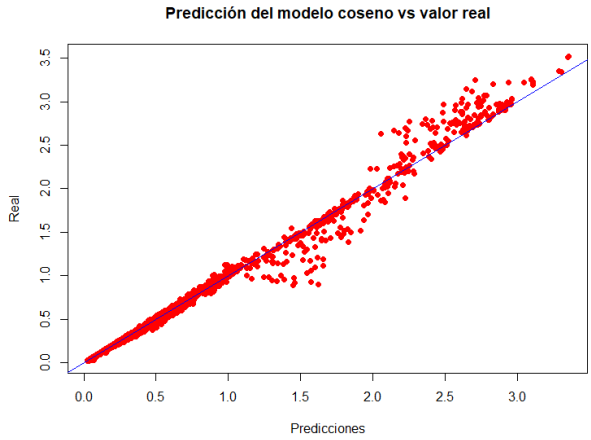

Figura 9.4: Modelo construido a partir del dataset coseno.

```
MSE modelo coseno: 0.00166753547124327
RMSE modelo jaccard: 0.0408354682995465
MAE modelo jaccard: 0.0184416130560806
R² modelo jaccard: 0.96654953783693
```

Figura 9.5: Resultados del modelo coseno.

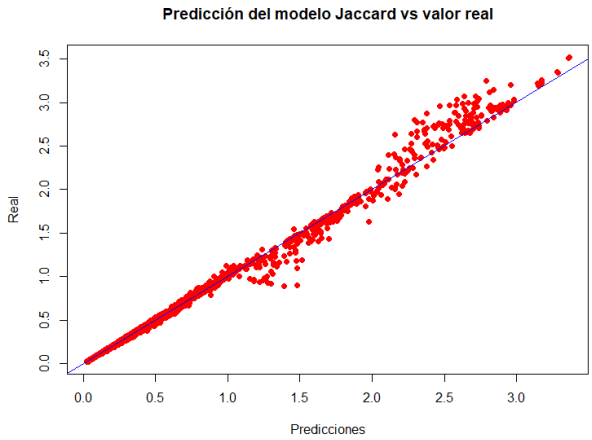

Figura 9.6: Modelo construido a partir del dataset jaccard.

```
MSE modelo jaccard: 0.00262758511234087
RMSE modelo jaccard: 0.0512599757348837
MAE modelo jaccard: 0.029571227543295
R² modelo jaccard: 0.940418276692909
```

Figura 9.7: Resultados del modelo jaccard.

Como conclusión hemos visto que el modelo que peor predice sobre un conjunto de datos independiente es el modelo original, seguido por el modelo jaccard y siendo el modelo coseno el que mejores predicciones realizó. La diferencia existente entre modelo coseno y el jaccard no es demasiado amplia pero con respecto al dataset original si que observamos un cambio muy drástico gracias a los valores estadísticos aportados por la distancia coseno y el índice jaccard.

9.2 K-NN

Vamos a realizar el mismo análisis que para Random Forest y así poder ver como de buenas son las predicciones para los tres modelos construidos. A continuación podemos ver las imágenes que contienen los resultados:

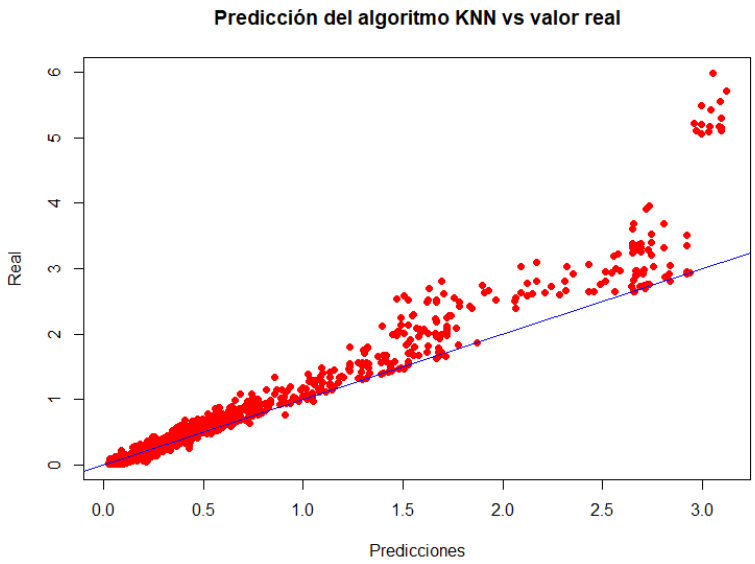

Figura 9.8: Predicciones realizas por el modelo original.

```
MSE modelo original: 0.0298101023389347
RMSE modelo original: 0.172656023175951
MAE modelo original: 0.0525104597946244
R² modelo original: 0.578590776401288
```

Figura 9.9: Resultados de las métricas para el modelo original.

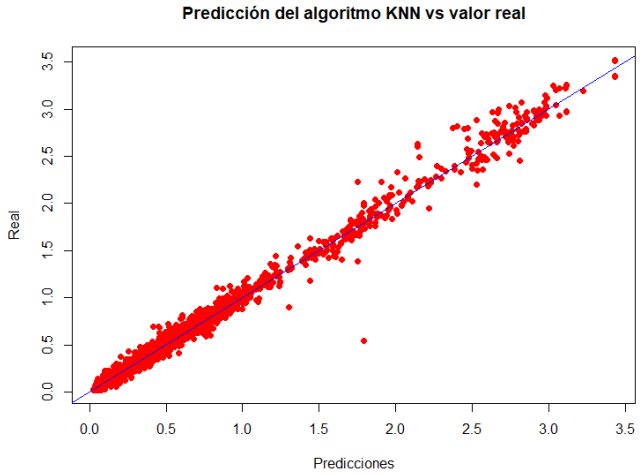

Figura 9.10: Predicciones realizas por el modelo coseno.

```
MSE modelo coseno: 0.00101090594725316
RMSE modelo coseno: 0.0317947471644792
MAE modelo coseno: 0.0127642449707323
R² modelo coseno: 0.977806189764824
```

Figura 9.11: Resultados de las métricas para el modelo coseno.

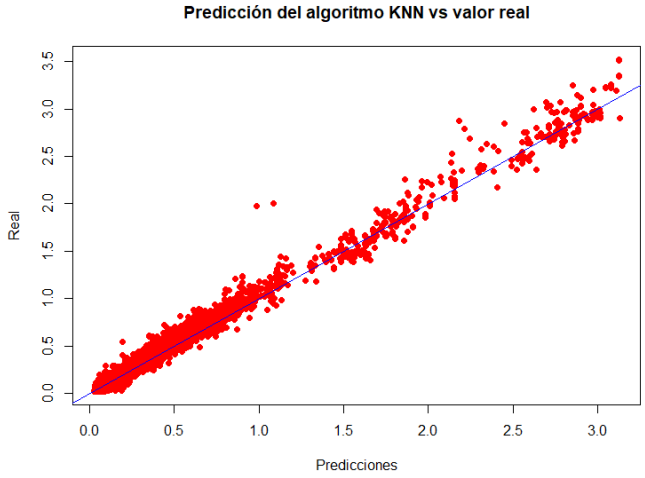

Figura 9.12: Predicciones realizas por el modelo jaccard.

```
MSE modelo jaccard: 0.00200759577664182
RMSE modelo jaccard: 0.0448062024349511
MAE modelo jaccard: 0.0251298044391335
R² modelo jaccard: 0.948622281005469
```

Figura 9.13: Resultados de las métricas para el modelo jaccard.

De nuevo nos encontramos con que el modelo coseno arroja los mejores resultados así como realiza las mejores predicciones. Es seguido por el modelo jaccard con el cuál anda muy parejo, pero esta vez si que tenemos que el modelo original no dista tanto de ambos en cuanto a resultados nos referimos.

9.3 Modelo RouteNet

Finalmente vamos a comprobar que tal se desenvuelve RouteNet sobre un conjunto de datos independiente. En las siguientes figuras se pueden ver los resultados y predicciones arrojados por este:

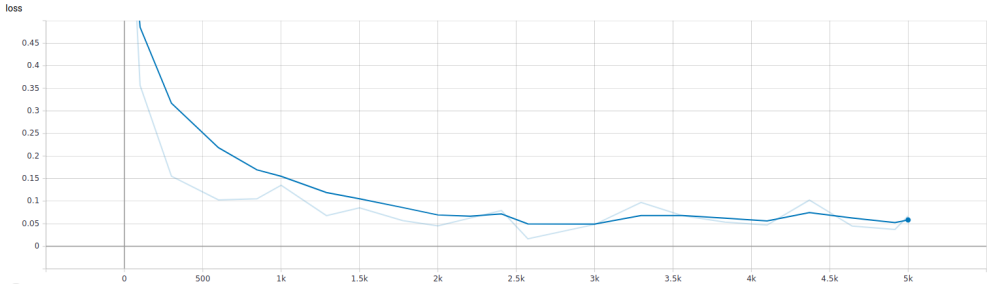

Figura 9.14: Pérdida en función del número de épocas utilizado.

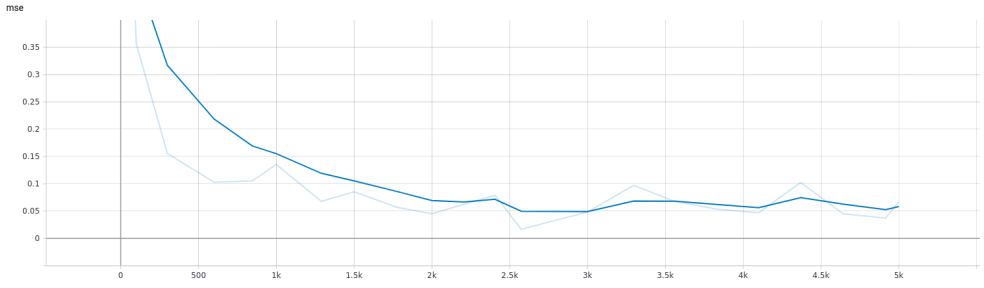

Figura 9.15: MSE en función del número de épocas utilizado.

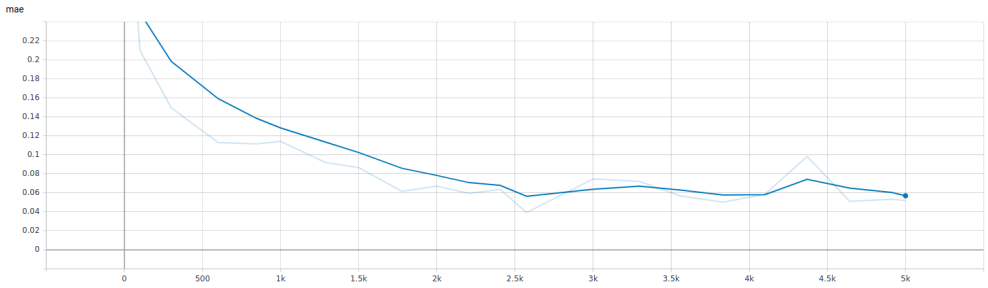

Figura 9.16: MAE en función del número de épocas utilizado.

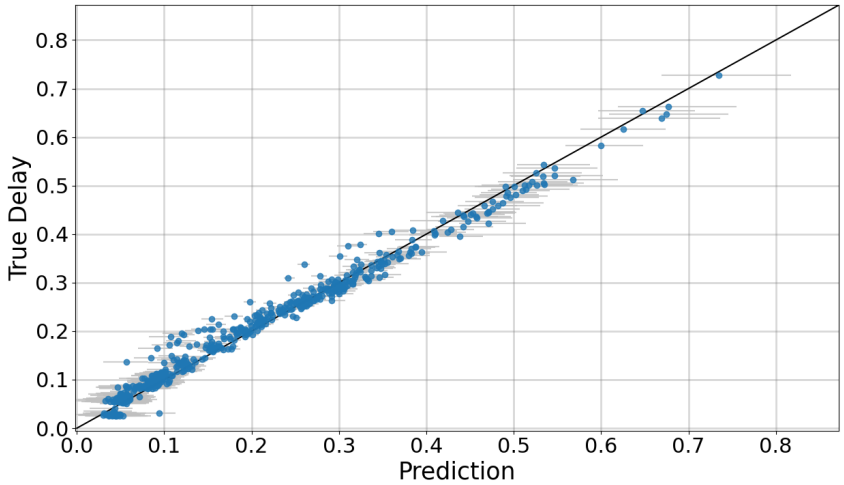

Figura 9.17: Predicción obtenida por el modelo óptimo.

```
Número de iteraciones -> 2700:
------------------------------

        - global_step = 2700,
        - label/mean = -0.055551894,
        - loss = 0.0421699,
        - mae = 0.05728327,
        - mre = 0.01855097,
        - mse = 0.042169888,
        - prediction/mean = -0.07513544,
        - rho = 0.9827648
```

Figura 9.18: Resultados de las métricas más relevantes para el modelo óptimo.

Observamos que los resultados devueltos por RouteNet no son malos pero no están a la altura de los que aportaron los mejores modelos para Random Forest (modelo coseno) y K-NN (modelo coseno).

9.4 Comparativa

Como ya sabemos, para hacer esta comparativa entre los tres métodos de ML, hemos tenido en cuenta las métricas más importante desde nuestro punto de vista. El modelo que mejores resultados ha aportado es el modelo coseno para K-NN seguido muy de cerca por el mismo modelo pero utilizando Random Forest. Esto se debe a que sobre el conjunto de datos independiente seleccionado K-NN consigue mejores resultados. Para asegurarnos de que K-NN aporta mejores resultados que Random Forest tendríamos que seguir haciendo pruebas con otros conjuntos independientes y así saber con absoluta certeza cuál es el que mejores predicciones realiza. En cuanto a RouteNet creemos que aporta soluciones óptimas al problema pero hay una diferencia considerable con la calidad obtenida frente a los otros algoritmos de machine learning. Esto seguramente se deba a que no se utilizan los atributos construidos a partir de la distancia coseno y el índice jaccard.

Capítulo 10

Conclusiones finales

Este proyecto se ha centrado en resolver el reto que tienen los operadores de red para conseguir soluciones de optimización y operación de red, como es el tráfico extremo a extremo. Se ha llevado a cabo el estudio y comparación de diferentes técnicas de ML con el fin de aplicarlas para crear modelos de red livianos y conseguir una calidad óptima. Para conseguirlo se analizaron los resultados de aplicar dichas técnicas sobre el conjunto de datos de tráfico de red seleccionado, además de realizar una exploración y limpieza adecuada de los datos. También, mediante el uso de técnicas como la validación cruzada, se ajustaron los valores de los hiperparámetros más interesantes de cara a conseguir unas predicciones satisfactorias por parte de los modelos construidos.

Con respecto a las técnicas mencionadas anteriormente, destaca el hecho de que todas ellas han arrojado resultados más que satisfactorios, siendo las más efectivas Random Forest y K-NN para los modelos que cuentan con atributos creados a partir de los estadísticos calculados con la distancia coseno (K-NN fue el que mejor resultado obtuvo en la evaluación 9). Por otro lado, hemos comprobado que RouteNet es bastante efectivo a la hora de construir modelos de red pero obtuvimos diferencias considerables en la calidad frente a los mencionados anteriormente. También comentar que la carga computacional necesaria para ejecutar RouteNet y construir el modelo oportuno es mucho mayor que en las otras técnicas utilizadas.

Gracias a este trabajo he podido introducirme en un campo que me era desconocido pero que ahora, una vez acabado, me aportó una gran cantidad de conocimiento sobre estas técnicas de Inteligencia Artificial (IA).

Finalmente este trabajo se podría ampliar utilizando otros algoritmos de ML (SVM entre otros) para resolver este mismo problema y comparar los resultados frente a los ya obtenidos. Llevar a cabo pruebas con otros conjuntos de datos aportaría una buena visión en cuanto a

la eficiencia de estos así como a la calidad de los modelos resultantes para cada una de las técnicas.

Apéndices

Bibliografía

[1] "¿cuál es el papel del aprendizaje automático en las redes?" [Accedido 15-04-2020]. [En línea]. Disponible en: https://searchdatacenter.techtarget.com/es/respuesta/Cual-es-el-papel-del-aprendizaje-automatico-en-las-redes/

[2] "Los beneficios del aprendizaje automático en la administración de redes," [Accedido 15-04-2020]. [En línea]. Disponible en: https://searchdatacenter.techtarget.com/es/cronica/Los-beneficios-del-aprendizaje-automatico-en-la-administracion-de-redes

[3] "Crisp-dm (metodología de desarrollo de software)," [Accedido 17-04-2020]. [En línea]. Disponible en: ftp://ftp.software.ibm.com/software/analytics/spss/support/Modeler/Documentation/14/UserManual/CRISP-DM.pdf

[4] M. X. D. R. Claudia L. Hernández G., "Hacia una metodología de gestión del conocimiento basada en minería de datos," [Accedido 18-04-2020]. [En línea]. Disponible en: http://repositorio.uigv.edu.pe/bitstream/handle/20.500.11818/982/COMTEL-2009-80-96.pdf?sequence=1&isAllowed=y

[5] J. A. G. Arancibia, "Metodología para el desarrollo de proyectos en minería de datos crisp-dm," [Accedido 19-04-2020]. [En línea]. Disponible en: http://www.oldemarrodriguez.com/yahoo_site_admin/assets/docs/Documento_CRISP-DM.2385037

[6] F. R. S. H. y. J. T. Paloma Recuero, Carmen Torrano, *Machine Learning aplicado a Ciberseguridad: Técnicas y ejemplos en la detección de amenazas*, 1st ed. 0xWord, 2019, [Accedido 22-04-2020].

[7] A. T. . ANALYTICS, "El algoritmo k-nn y su importancia en el modelado de datos," [Accedido 24-04-2020]. [En línea]. Disponible en: https://www.analiticaweb.es/algoritmo-knn-modelado-datos/

[8] "Clasificar con k-nearest-neighbor ejemplo en python," [Accedido 24-04-2020]. [En línea]. Disponible en: https://www.aprendemachinelearning.com/clasificar-con-k-nearest-neighbor-ejemplo-en-python/

[9] J. M. Heras, "Random forest (bosque aleatorio): combinando árboles," [Accedido 26-04-2020]. [En línea]. Disponible en: https://iartificial.net/random-forest-bosque-aleatorio/#¿Que_es_un_Random_Forest

[10] J. O. Alvear, "¿cómo se contruye un modelo random forest?" [Accedido 27-04-2020]. [En línea]. Disponible en: https://bookdown.org/content/2031/ensambladores-random-forest-parte-i.html#random-forest

[11] J. F. Trevor Hastie, Robert Tibshirani, *The Elements of Statistical Learnin. Data Mining, Inference, and Prediction.* Springer, 2017, [Accedido 28-04-2020].

[12] "Manual–setting up, using, and understanding random forests v4.0," [Accedido 28-04-2020]. [En línea]. Disponible en: https://www.stat.berkeley.edu/~breiman/Using_random_forests_v4.0.pdf

[13] T. H. Gareth James, Daniela Witten and R. Tibshirani, *An Introduction to Statistical Learning with Applications in R.* Springer, [Accedido 30-04-2020]. [En línea]. Disponible en: http://faculty.marshall.usc.edu/gareth-james/ISL/ISLR%20Seventh%20Printing.pdf

[14] "Qué son las redes neuronales y sus funciones," [Accedido 01-05-2020]. [En línea]. Disponible en: https://www.atriainnovation.com/que-son-las-redes-neuronales-y-sus-funciones/

[15] G. Julián, "Las redes neuronales: qué son y por qué están volviendo," [Accedido 02-05-2020]. [En línea]. Disponible en: https://www.xataka.com/robotica-e-ia/las-redes-neuronales-que-son-y-por-que-estan-volviendo

[16] "Las redes neuronales: qué son y por qué están volviendo," [Accedido 02-05-2020]. [En línea]. Disponible en: http://redes-neuronales.wikidot.com/definicion-ventajas-desventajas

[17] "Omnet++." [En línea]. Disponible en: https://omnetpp.org/

[18] "Tensorflow." [En línea]. Disponible en: https://www.tensorflow.org/

[19] "5 ways to detect outliers/anomalies that every data scientist should know," [Accedido 5-05-2020]. [En línea]. Disponible en: https://towardsdatascience.com/5-ways-to-detect-outliers-that-every-data-scientist-should-know-python-code-70a54335a623

[20] "Índice jaccard," [Accedido 06-05-2020]. [En línea]. Disponible en: https://es.wikipedia.org/wiki/Índice_Jaccard

[21] "Similitud coseno," [Accedido 06-05-2020]. [En línea]. Disponible en: https://es.wikipedia.org/wiki/Similitud_coseno

[22] "Information gain in decision trees," [Accedido 07-05-2020]. [En línea]. Disponible en: https://en.wikipedia.org/wiki/Information_gain_in_decision_trees

[23] "Error cuadrático medio," [Accedido 12-05-2020]. [En línea]. Disponible en: https://es.wikipedia.org/wiki/Error_cuadrático_medio

[24] "Raíz del error cuadrático medio," [Accedido 13-05-2020]. [En línea]. Disponible en: https://es.wikipedia.org/wiki/Raíz_del_error_cuadrático_medio

[25] "Error absoluto medio," [Accedido 12-05-2020]. [En línea]. Disponible en: https://es.wikipedia.org/wiki/Error_absoluto_medio

[26] "Análisis de regresión: ¿cómo puedo interpretar el r-cuadrado y evaluar la bondad de ajuste?" [Accedido 15-05-2020]. [En línea]. Disponible en: https://blog.minitab.com/es/analisis-de-regresion-como-puedo-interpretar-el-r-cuadrado-y-evaluar-la-bondad-de-ajuste#:~:text=El%20R%2Dcuadrado%20es%20una,se%20trata%20de%20regresi%C3%B3n%20m%C3%BAltiple.

[27] D. Dalpiaz, "R for statistical learning," [Accedido 23-05-2020]. [En línea]. Disponible en: https://daviddalpiaz.github.io/r4sl/knn-reg.html

[28] "Network modeling datasets," [Accedido 16-04-2020]. [En línea]. Disponible en: https://github.com/knowledgedefinednetworking/NetworkModelingDatasets

[29] "Routenet," [Accedido 27-05-2020]. [En línea]. Disponible en: https://dl.acm.org/doi/10.1145/3342280.3342327

[30]

[31] "Illustrated guide to lstm's and gru's: A step by step explanation," [Accedido 31-05-2020]. [En línea]. Disponible en: https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

[32] "Arquitecturas de aprendizaje profundo," [Accedido 02-06-2020]. [En línea]. Disponible en: https://www.ibm.com/developerworks/ssa/library/cc-machine-learning-deep-learning-architectures/index.html

[33] "Regularizando nuestra red:dropout," [Accedido 03-06-2020]. [En línea]. Disponible en: https://mc.ai/regularizando-nuestra-red-dropout/

[34] "Learning rate," [Accedido 04-06-2020]. [En línea]. Disponible en: https://en.wikipedia.org/wiki/Learning_rate

[35] "What is batch size in neural network?" [Accedido 04-06-2020]. [En línea]. Disponible en: https://stats.stackexchange.com/questions/153531/what-is-batch-size-in-neural-network