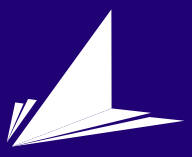


Escuela
Politécnica
Superior

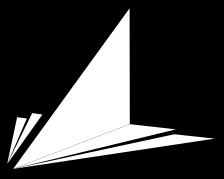
Título del Trabajo Fin de Grado/Máster

Grado en Ingeniería en Sonido e Imagen en Telecomunicación

Trabajo Fin de Grado

Autor:
Luis Pérez-Sala García-Plata
Tutores:
José Francisco Vicent Frances
Nombre Apellido1 Apellido2 (tutor2)

Mayo 2022



Escuela
Politécnica
Superior

Título del Trabajo Fin de Grado/Máster

Subtítulo del proyecto

Autor
Luis Pérez-Sala García-Plata

Tutores
José Francisco Vicent Frances
Ciencia de la Computacion e Inteligencia Artificial
Nombre Apellido1 Apellido2 (tutor2)
Departamento del cotutor

Grado en Ingeniería en Sonido e Imagen en Telecomunicación

ALICANTE, Mayo 2022

Preámbulo

Poner aquí un texto breve que debe incluir entre otras:

> "las razones que han llevado a la realización del estudio, el tema, la finalidad y el alcance y también los agradecimientos por las ayudas, por ejemplo apoyo económico (becas y subvenciones) y las consultas y discusiones con los tutores y colegas de trabajo. (**?**)"

Agradecimientos[1]

Este trabajo no habría sido posible sin el apoyo y el estímulo de mi colega y amigo, Doctor Rudolf Fliesning, bajo cuya supervisión escogí este tema y comencé la tesis. Sr. Quentin Travers, mi consejero en las etapas finales del trabajo, también ha sido generosamente servicial, y me ha ayudado de numerosos modos, incluyendo el resumen del contenido de los documentos que no estaban disponibles para mi examen, y en particular por permitirme leer, en cuanto estuvieron disponibles, las copias de los recientes extractos de los diarios de campaña del Vigilante Rupert Giles y la actual Cazadora la señorita Buffy Summers, que se encontraron con William the Bloody en 1998, y por facilitarme el pleno acceso a los diarios de anteriores Vigilantes relevantes a la carrera de William the Bloody.

También me gustaría agradecerle al Consejo la concesión de Wyndham-Pryce como Compañero, el cual me ha apoyado durante mis dos años de investigación, y la concesión de dos subvenciones de viajes, una para estudiar documentos en los Archivos de Vigilantes sellados en Munich, y otra para la investigación en campaña en Praga. Me gustaría agradecer a Sr. Travers, otra vez, por facilitarme la acreditación de seguridad para el trabajo en los Archivos de Munich, y al Doctor Fliesning por su apoyo colegial y ayuda en ambos viajes de investigación.

No puedo terminar sin agradecer a mi familia, en cuyo estímulo constante y amor he confiado a lo largo de mis años en la Academia. Estoy agradecida también a los ejemplos de mis difuntos hermano, Desmond Chalmers, Vigilante en Entrenamiento, y padre, Albert Chalmers, Vigilante. Su coraje resuelto y convicción siempre me inspirarán, y espero seguir, a mi propio y pequeño modo, la noble misión por la que dieron sus vidas.

Es a ellos a quien dedico este trabajo.

[1]Por si alguien tiene curiosidad, este "simpático" agradecimiento está tomado de la "Tesis de Lydia Chalmers" basada en el universo del programa de televisión Buffy, la Cazadora de Vampiros.http://www.buffy-cazavampiros.com/Spiketesis/tesis.inicio.htm

A mi esposa Marganit, y a mis hijos Ella Rose y Daniel Adams,
sin los cuales habría podido acabar este libro dos años antes [2]

[2]Dedicatoria de Joseph J. Roman en "An Introduction to Algebraic Topology"

Si consigo ver más lejos
es porque he conseguido auparme
a hombros de gigantes

Isaac Newton.

Índice general

Índice de figuras

Índice de tablas

Índice de Códigos

1 Introducción

1.1 Motivación

Los accidentes de tráfico son uno de los principales problemas de Salud Pública en nuestros días. La multicausalidad, la variedad de las fuentes de datos y la poca cantidad de análisis específicos, apuntan a una gran complejidad en su tratamiento. Mas de 3.500 personas mueren, en las carreteras, diariamente en todo el mundo. Esto, significa casi 1,3 millones de muertes y aproximadamente 50 millones de lesiones cada año. Por lo tanto, la predicción de la gravedad de los accidentes de tráfico es un componente muy importante que se debe tener en cuenta, adoptando cualquier mejora en la predicción de la gravedad de los mismos. Además, el impacto económico y social asociado con los accidentes de tráfico lleva a las administraciones a buscar de manera activa mejoras. En el campo de la investigación sobre seguridad vial, el desarrollo de metodologías fiables para predecir y clasificar el nivel de gravedad, de los accidentes de tráfico, en función de diversas variables es un componente clave. La creciente disponibilidad de datos a gran escala de varias fuentes, incluidos los vehículos conectados y autónomos exige una comprensión profunda de las relaciones causales entre la seguridad y factores asociados con la ayuda de nuevas metodologías.

En las últimas dos décadas, los rápidos desarrollos en los métodos de aprendizaje automático (ML) y su regresión precisa y rendimiento de clasificación han atraído mucho la atención de los investigadores. Ha habido un número creciente de aplicaciones de métodos ML en la investigación de la gravedad de los accidentes. Aunque los métodos estadísticos tradicionales tienen formas funcionales rigurosas y claramente definidas, los métodos ML tienen una gran flexibilidad,requieren poca o ninguna suposición previa con respecto a los datos de gravedad de choques y son capaces de manejar valores perdidos, ruidos y valores atípicos.

A pesar de que la utilización de técnicas de inteligencia artificial (ML) en lugar de otras técnicas simples permite identificar las relaciones ocultas entre los factores de accidentes debido a la heterogeneidad de los datos de accidentes de tráfico, el principal inconveniente de estos modelos es que muy pocos estudios utilizan modelos de aprendizaje profundo para mejorar el rendimiento lo que se traduce en un bajo rendimiento en accidentes graves. Además, como los conjuntos de datos de accidentes de tráfico existentes, están muy desequilibrados, es difícil mejorar el rendimiento de las clases menos comunes como los accidentes graves. Basándose en lo expuesto anteriormente, en este trabajo se plantea estudiar y formalizar un modelo de aprendizaje profundo para predicción de la gravedad de los accidentes de tráfico. Con la finalidad de las características de los datos y preparar la entrada para el modelo de aprendizaje profundo, se ha utilizado una técnica para transformar números en matrices bidimensionales (Sharma et al., 2019) consistente en convertir un vector de características del accidente en una matriz de características, teniendo en cuenta que los vectores de características eran las variables de los datos del accidente y la matriz de características tenía las posiciones de las variables. Para fines de comparación, los resultados se han comparado con otros resultados

de modelos de aprendizaje automático.

1.2 Objetivos

1.2.1 Objetivos Generales

El objetivo principal de este trabajo final de Máster es desarrollar un sistema predictivo de la avedad de accidentes de tráfico utilizando características que se pueden identificar en los lugares del accidente, como el sexo del conductor, el tipo de vehículo, el entorno o los atributos de la carretera.

1.2.2 Objetivos Específicos

Con la finalidad de realizar este objetivo principal, se plantean los siguientes objetivos secundarios:

1. Utilización de técnicas para la transformación de características cualitativas de los accidentes de tráfico en matrices numéricas.
2. Utilización de algoritmos de aprendizaje automático basado en árboles de decisiones (XGBoost) para inferir pesos de las características de accidentes de tráfico.
3. Utilizar técnicas de algoritmia evolutiva para optimización de parámetros de entrada del algoritmo XGBoost.
4. Desarrollar un enfoque basado en el aprendizaje profundo, redes convolucionales, para predecir la gravedad de los accidentes de tráfico.
5. Comparar los resultados de los modelos de aprendizaje profundo con otros modelos utilizando indicadores de rendimiento.

2 Marco Teórico

3 Tecnologías

Una vez definido el problema a tratar, los objetivos generales y específios del proyecto, la solución propuesta y los requisitos a implementar hay que analizar las herramientas y tecnologías a utilizar. Para esto habrá que plantearse diferentes modelos y tecnologías disponibles para el desarrollo y optar por aquellos que aporten estabilidad y buen rendimiento.

3.1 Modelos

En primer lugar analizaremos los modelos utilizados durante la realización del proyecto.

3.1.1 Algoritmos Genéticos

Un *Algoritmo Genético* Lingaraj (2016), es un modelo que imita la evolución de las especies en el ámbito de la biología, con el objetivo de encontrar una solución potencialmente óptima a un problema. El planteamiento de estos modelos consiste en evaluar los individuos pertenecientes a la población de la generación correspondiente para tomar el subconjunto de aquellas soluciones que mejor calidad ofrezcan mediante una función heurística denominada *función fitness* que aproxima soluciones ideales a un problema.

Las funciones heurísticas son funciones aproximadas a una solución ideal del problema, debido a que todas las posibles combinaciones de los parámetros a optimizar genera un espacio de búsqueda exponencial (problema NP-hard), es necesario crear una aproximación mediante la función heurística para acotar éste espacio.

Para comprender el funcionamiento de los algoritmos genéticos es necesario analizar cada una de las fases que lo constituyen:

1. Inicialización de la Población:

   En primer lugar es necesario generar los n individuos de la población aleatoriamente. Cada uno de estos individuos está formado por el conjunto de parámetros que son objeto de optimización 3.1.

2. Evaluación de Población:

   Es la primera fase del proceso iterativo del algoritmo genético y, en él, cada uno de los individuos pertenecientes a la población son evaluados mediante la función *fitness*.

3. Selección de Padres:

   Una vez evaluados los individuos de la población, se procede a seleccionar aquellos que mejor valor han obtenido para crear nuevas soluciones a partir de éstos padres.

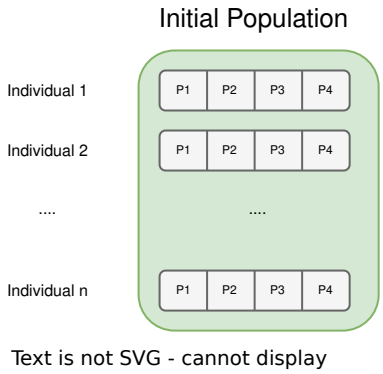


Figura 3.1: Inicialización de la población. Cada individuo está constituido por n parámetros del XGBoost a optimizar.

Existen distintas técnicas de selección, como por ejemplo escoger aquellos n mejores padres de la población o técnicas basadas en métodos probabilísticos, para que aquellos individuos que menos puntuación logren, tengan alguna probabilidad de ser elegidos para la generación de nuevas soluciones. Este tipo de técnicas se utilizan para aumentar la diversidad en la poblάción y evitar caer en soluciones acotadas a mínimos locales del problema. Se puede ver el proceso de cómo son seleccionados los padres en función de su *fitness* en la figura 3.2.

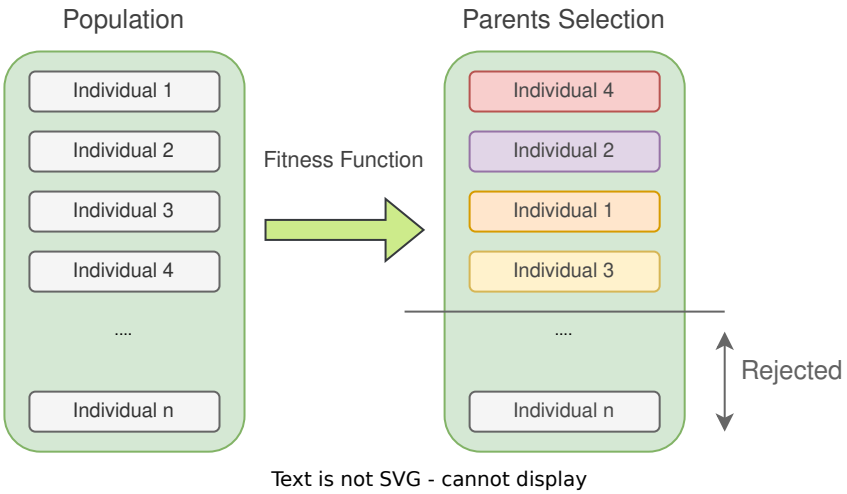


Figura 3.2: Selección de padres candidatos.

4. Cruce:

   Una vez seleccionados los padres es necesario que intercambien información entre ellos para dar lugar a nuevos individuos, este proceso de herencia de información es posible realizarlo aplicando distintas filosofías; seleccionar un punto o posición de cruce a lo largo del vector de los padres y combinar ambos padres, o escoger aleatoriamente las posiciones de los parámetros de ambos padres y combinarlos para dar lugar a la solución

generada figura [3.3].

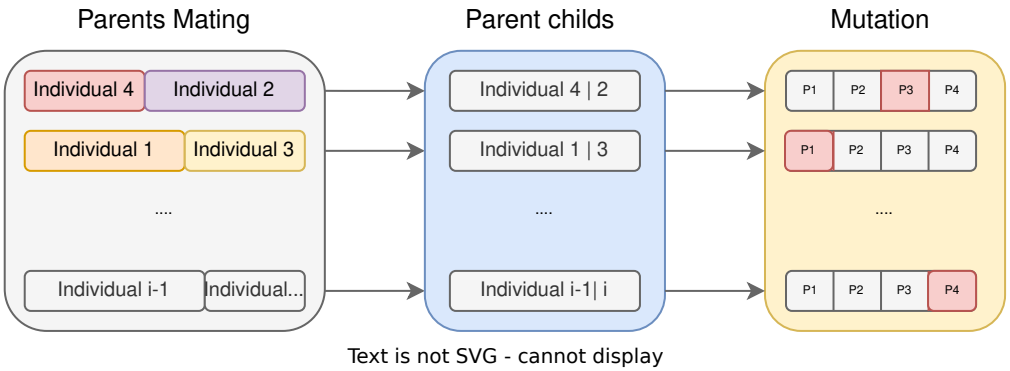


Figura 3.3: Mutación de hijos.

5. Mutación:

   Cuando los padres seleccionados son combinados para dar lugar a los candidatos de la nueva generación, es importante aplicar un proceso de mutación entre éstos debido a la necesidad de generar diversidad en la población. Si se combina la misma información entre generaciones se corre el riesgo de converger prematuramente a un mínimo local del espacio de búsqueda. Por este motivo se introduce el concepto de mutación, que consiste en aplicar una modificación aleatoria sobre alguno de los parámetros de los individuos generados para poder ampliar el espacio de soluciones.

Cada una de las fases enumeradas anteriormente se repite de forma iterativa entre generaciones hasta que se cumpla alguna condición de parada.

A continuación enumeraremos los parámetros con los que configurar un algoritmo genético. Dependiendo del tipo de algoritmo utilizado es posible encontrar distintos parámetros con los que inicializar el algoritmo, sin embargo estos son los más extendidos:

- Probabilidad de cruce:

  Probabilidad con la que dos padres intercambian su información entre sí para dar lugar a un nuevo individuo hijo.

- Probabilidad de mutación:

  Indica la probabilidad con la que cada hiperparámetro puede variar su valor.

3.1.2 XGBoost

Extreme Gradient Boosting Algortihm (*XGBoost*) es una libería open-source que implementa algoritmos *ML Ensembles de tipo Boosting*. Se trata de uno de los algoritmos más importantes de los últimos años, logrando un rendimiento diez veces mayor a otras soluciones populares en una misma máquina y llegando a las primeras posiciones en numerosos retos propuestos en *Kaggle* Chen y Guestrin (2016).

XGBoost se basa en los modelos *Ensemble Boosting* de los algoritmos de *ML*, de ahí su nombre *Gradient Boosting*. La técnica *Boosting* se basa en construir un modelo robusto (*strong learner*) combinando una serie de modelos débiles (*weak learners*) Nvidia (s.f.-b).

Concretamente *XGBoost* implementa *Gradient Boosting Decision Trees (GBDT)*, que es un algoritmo similar a los *Random Forest*, y es utilizado tanto para clasificación como para regresión.

La diferencia principal entre los algoritmos *Random Forest* y *XGBoost* radica en el hecho de que los primeras utilizan algoritmos *Ensembles* tipo *Bagging* y construyen un modelo final predictivo mediante en el que la salida será la combinación de las predicciones de cada uno de los modelos individuales. En el caso de los modelos *Ensembles Boosting* (*XGBoost*) se combinan secuencialmente de forma que su objetivo es minimizar el error producido los árboles generales utilizando para ello el método de descenso por gradiente.

En la figura [3.4] se puede observar cómo se van construyendo los árboles clasificadores y cómo cada árbol minimiza el residuo producido por el árbol anterior.

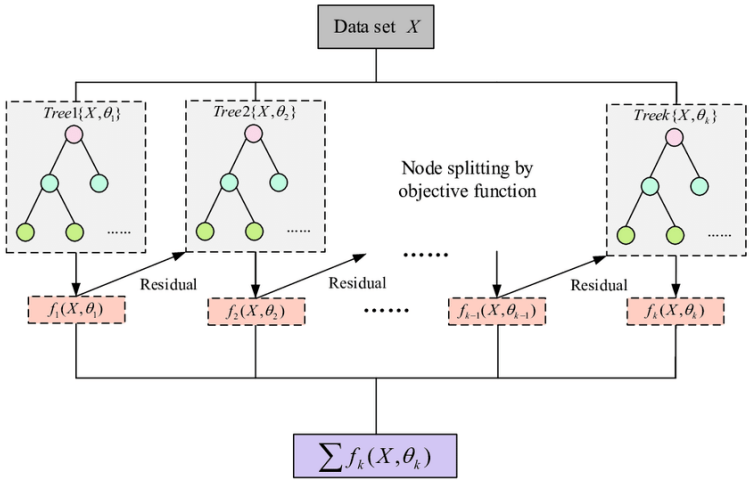


Figura 3.4: Proceso de optimización de XGBoost.

Además, *XGBoost* aplica técnicas de regularización en su función de pérdida, de tal forma que reduce la influencia individual de cada árbol generado y de sus hojas para poder dar lugar a posteriores árboles que consigan mejorar el modelo, evitando de esta manera el sobreajuste u *overfitting*.

3.1.3 Redes Neuronales

Las Redes Neuronales Emmert-Streib y cols. (2020) son modelos que emulan comportamiento del cerebro a la hora de procesar la información, entrenándose sobre un conjunto de datos de entrenamiento para identificar patrones entre ellos.

A modo de comprensión, se muestra la arquitectura de una *NN*, donde se pueden observar las conexiones entre cada capa de la red y sus salidas en la figura 3.5. Existe una capa de entrada inicial que corresponde a cada una de las características del ejemplo de entrada (en el caso de imágenes se trata de cada uno de los píxeles). Cada una de las características de la capa de entrada se conecta con cada una de las neuronas de la primera capa oculta *hidden layer* a través de conexiones denominadas pesos (*weights*) que son los que serán aprendidos en la red con el objetivo de encontrar patrones entre las características mediante el método de *Descenso por Gradiente* o *Gradient Descent (GD)*.

Esta metodología se aplica para cada una de las *hidden layers* presentes en la arquitectura hasta la última capa, llegando hasta la última función *SoftMax* que dictará la clase predicha en base a la salida de la capa final, u *output layer*.

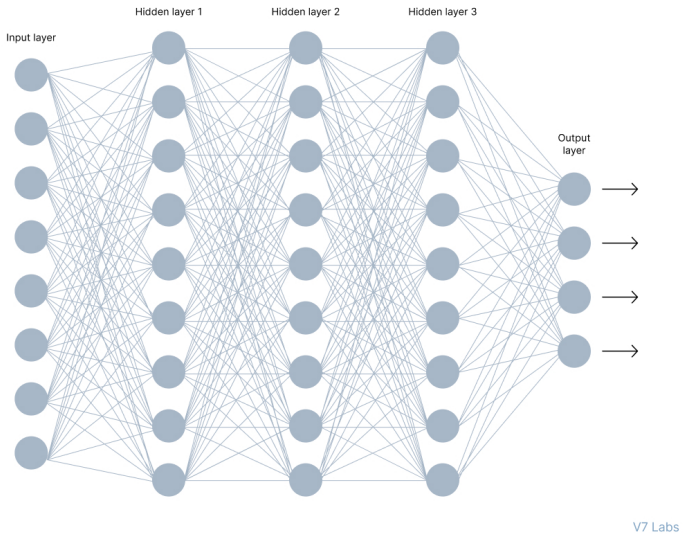


Figura 3.5: https://www.v7labs.com/blog/neural-networks-activation-functions. Arquitectura de una Red Neuronal.

Existen una serie de conceptos comunes a todas las *NN* que es conveniente nombrar para su comprensión:

- Función de activación:

  Las funciones de activación son las encargadas de decidir si una neurona es activada o no en función de la entrada a esta función. Esta función suele situarse en la salida de cada capa de la red para decidir si una determinada neurona de la siguiente capa se activa o no.

  Existen distintos enfoques en lo que respecta a las funciones de activación, como la función logística, tangente hiperbólica y *softmax* entre otras.

- Entropía:

La entropía sobre una varible aleatoria X es el grado de incertidumbre producido en base a los posibles valores que puede tomar como respuesta. Se calcula en base a la siguiente fórmula:

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log_2 p(x_i)$$

Donde x_i es cada uno de los posibles valores que puede tomar la variable como respuesta y $p(x_i)$ es la probabilidad de obtener dicho valor.

- Softmax:

  Se trata de la generalización en forma multidimensional de la función logística o *logistic function*. *Softmax* es una función que permite la transformación de un vector de números reales a una distribución de probabilidad. Esta función se suele aplicar como capa de activación en el último nivel en las *NN*, representando la probabilidad de que la salida de la red pertenezca a cada una de las K clases posibles en el modelo Wikipedia (s.f.).

  Esto es necesario debido a que normalmente los componentes de la salida de la última capa de una red neuronal (*logits*) son el resultado de la predicción en forma de números reales no normalizados, por lo tanto pueden ser mayores que 1 e incluso negativos. La función permite calcular la distribución de probabilidad en base a estos *logits* y calcular la probabilidad de pertenencia de la muestra a cada una de las K clases que componen el modelo mediante la siguiente fórmula:

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad for\ i = 1, 2, \ldots, K$$

  Esta función es continua y diferenciable, por lo que es posible aplicar el método *GD* para actualizar los valores de los pesos de la red neuronal en las capas anteriores gracias a estas propiedades que permiten la derivabilidad de la función.

- Cross-Entropy:

  El objetivo de cross-entropy es minimizar la función de pérdida (*log loss*) comparando la probabilidad de la clase predicha con respecto a su valor verdadero. Al aplicar una penalización logarítmica, las probabilidades de las clases que más disten respecto a su valor verdadero se verán más acentuadas Koech (s.f.).

$$L_{CE} = -\sum_{i=1}^{n} y_i \log_2(p_i)$$

  Donde n es el número de clases a predecir, el valor y_i es la etiqueta de la clase verdadera, y p_i es la probabilidad de que la muestra actual pertenezca a la clase i.

- Back Propagation:

  En el proceso de entrenamiento de la *NN* es necesario actualizar los pesos de las capas ocultas con el objetivo de minimizar la cross-entropy total del modelo. Para minimizar este valor se utiliza un algoritmo de optimización que maximiza el valor de una función *GD*.

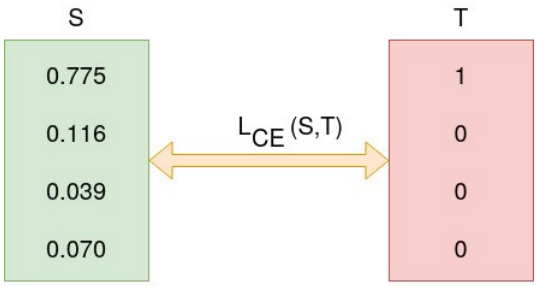


Figura 3.6: https://towardsdatascience.com/cross-entropy-loss-function-f38c4ec8643e Cross-Entropy entre la función Softmax (izquierda) y el valor de la clase verdadera (derecha).

Gracias a las propiedades de la función *Softmax* es posible utilizar descenso por gradiente para calcular la derivada de la función de pérdida respecto de cada uno de los pesos de las capas intermedias de la red y actualizar cada uno de ellos con el objetivo de minimizar la función de pérdida.

- Learning Rate:

  La *tasa de aprendizaje*, o *learning rate* define el tamaño de paso que se aplicará a a la hora de optimizar los pesos de la red. El valor del *learning rate* se comprende en el intervalo $[0, 1]$, de tal forma que a menor valor de la tasa de aprendizaje menor será el cambio que se produce en los pesos de las neuronas del modelo, el caso contrario ocurre cuando adopta un valor alto.

  Este parámetro no es sencillo de especificar, el *learning rate* es un regularizador que con valores altos permite que la red no caiga en *overfitting*, caso que ocurriría si el valor fuera demasiado pequeño y la minimización de la función de pérdida cayese en un mínimo local debido a aprender demasiado de los datos de entrenamiento. Si el valor es muy alto los pesos de los parámetros de la red variarán excesivamente impidiendo la convergencia en un mínimo de la función.

  Esta disyuntiva ha sido ampliamente estudiada en los últimos tiempos, acerca de cuál es el valor ideal del *learning rate* en las *NN* Smith (2018), aunque comunmente se configura con el valor 0.1.

- Batch Normalization:

  El proceso de entrenamiento de una red neuronal puede ser costoso, ya que existen una gran cantidad de parámetros para cada capa que son optimizados durante el proceso de entrenamiento. Esto hace que este proceso sea demasiado sensible a los parámetros inicializados en el inicio del entrenamiento de la red, como es el ejemplo de establecer una tasa de aprendizaje *learning rate* muy baja para poder converger. Esto hace que el entrenamiento sea muy lento, por lo que es necesario normalizar las entradas para cada mini-batch (subconjuntos de datos de entrenamiento con el que se entrena la red en cada época.

El método *Batch Normalization* Ioffe y Szegedy (2015) normaliza los datos de tal forma que permite utilizar tasas de aprendizaje mayores además de actuar como regularizador, reduciendo así el número de capas *Dropout* que pudieran existir en la arquitectura.

3.1.3.1 Convolucionales

A contiuación definiremos los parámetros específicos aplicados a las *CNNs*:

- Kernels:

  Son las estructuras que se encargan de realizar la convolución. Los kernels son matrices de pesos unidimensionales $(1 \times n)$ o bidimensionales $(n \times m)$ dependiendo del tipo de convolución que se esté aplicando (Convolución 1-D o Convolución 2-D), y se encargan de inferir patrones en la matriz de entrada gracias a la actualización de sus pesos. En función de la profundidad de la capa, los kernels podrán encontrar patrones más simples o más complejos debido a que la entrada a cada capa es una convolución (aplicación del filtro) de las capas anteriores.

  Un kernel realiza el producto escalar de sus pesos con respecto a las posiciones de la matriz de entrada (imagen o *feature map*) que colisionen en ese momento con el kernel, generando un nuevo *feature map* de menor dimensión, a menos que se utilice la técnica de *Padding* Ganesh (s.f.).

- Filtros:

  Un filtro no es más que una agrupación de kernels, siempre tendrán una dimensión mayor que la definida para el kernel. Por ejemplo, si disponemos de un kernel unidimensional $(1 \times n)$, el filtro contendrá k kernels de $(1 \times n)$ en la capa convolucional definida Khandelwal (s.f.)

- Strides:

  Son los saltos producidos por un kernel en cada etapa de la convolución, dependiendo si la convolución es dimensional o bidimensional, el stride constará de una dimensión (i), es decir, los pasos hacia la derecha en la imagen, o de dos dimensiones (i, j), los pasos hacia la derecha y hacia abajo en la imagen.

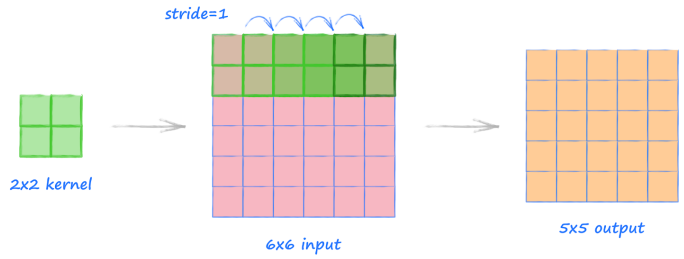


Figura 3.7: http://makeyourownneuralnetwork.blogspot.com/2020/02/calculating-output-size-of-convolutions.html

- Padding:

  El padding es un método utilizado para incrementar la dimensionalidad de la entrada de la capa debido el desvanecimiento de los valores posicionados en zonas comprometedoras una vez aplicada la convolución.

  El *padding* es el proceso de generar celdas artificiales inicializadas con valor *0* que permiten que el filtro convolucional sea aplicado en su totalidad en la posición de estas zonas comprometedoras, manteniendo la información de los límites de la imagen. De otra forma los valores de estos extremos se desvanecerían a medida se incrementa la profundidad de la red (número de capas) con respecto al resto de valores de la matriz.

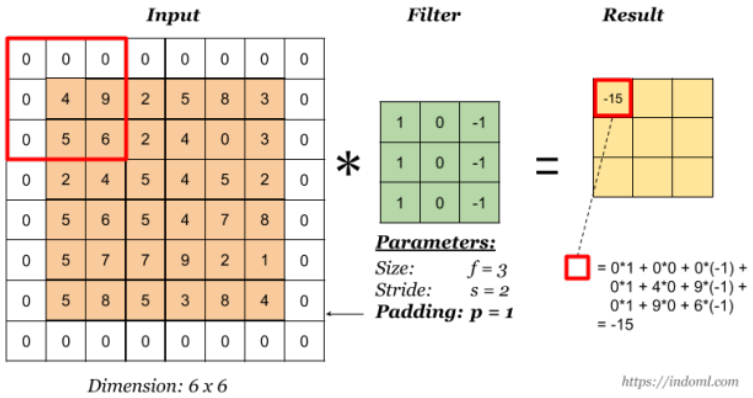


Figura 3.8: https://indoml.com/2018/03/07/student-notes-convolutional-neural-networks-cnn-introduction/

- Función de activación:

  En el caso de las redes neuronales convolucionales, la función de activación se aplica a cada casilla proyectada por cada kernel sobre la matriz de entrada, de tal forma que cambia el valor a 0 sobre aquellos valores que resulten negativos de la convolución.

  En la figura 3.9 se puede apreciar cómo se aplica la función de activación *ReLU* para cada casilla de la proyección de un *kernel* bidimensinoal sobre una *red convolucional de dos dimensiones*.

En el proceso de entrenamiento de las *CNNs* cada uno de los valores de los kernels son optimizados mediante el método *GD*, de tal forma que los pesos de cada capa son actualizados gracias a la derivabilidad de la función de pérdida.

3.1.3.1.1 Unidimensionales Una *Red Neuronal Convolucional de 1 Dimensión*, o *Convolutional Neural Network 1-Dimensional (CNN-1D)*, es un tipo de red convolucional que utiliza kernels *(filters)* que son aplicados a las imágenes de entrada con el objetivo encontrar ciertas

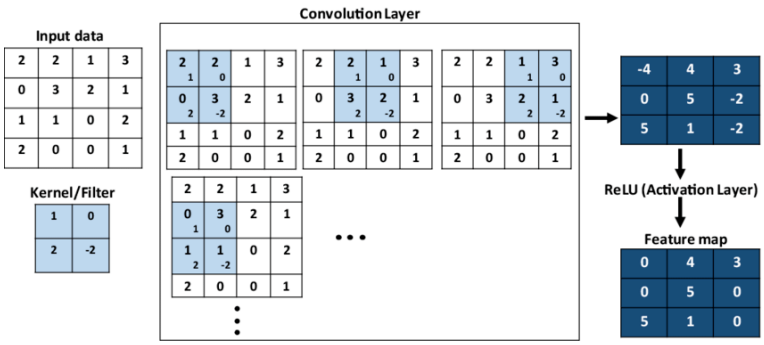


Figura 3.9: https://www.researchgate.net/journal/Transport-in-Porous-Media-1573-1634. Ejemplo de ReLU aplicado a una convolución de dos dimensiones.

características en ellas.

Las complejidad computacional de las *CNN-1D* es mucho más simple que las *CNN-2D*, además, estas redes se han demostrado ser más efectivas en distintos campos con respecto a las *CNN-2D*, como en técnicas de análisis de señales. La baja carga computacional de esta arquitectura la hace atractiva para aplicarla en tiempo real en dispositivos con pocos recursos como teléfonos móviles Kiranyaz y cols. (2019).

El resultado de este proceso es la proyección del filtro sobre un espacio dimensional denominado mapa de caracerísticas *(feature maps)*. Este filtro se utiliza para convolucionar los feature maps de la capa anterior Khandelwal (s.f.) o de la imagen de entrada si se trata de la primera capa de la red.

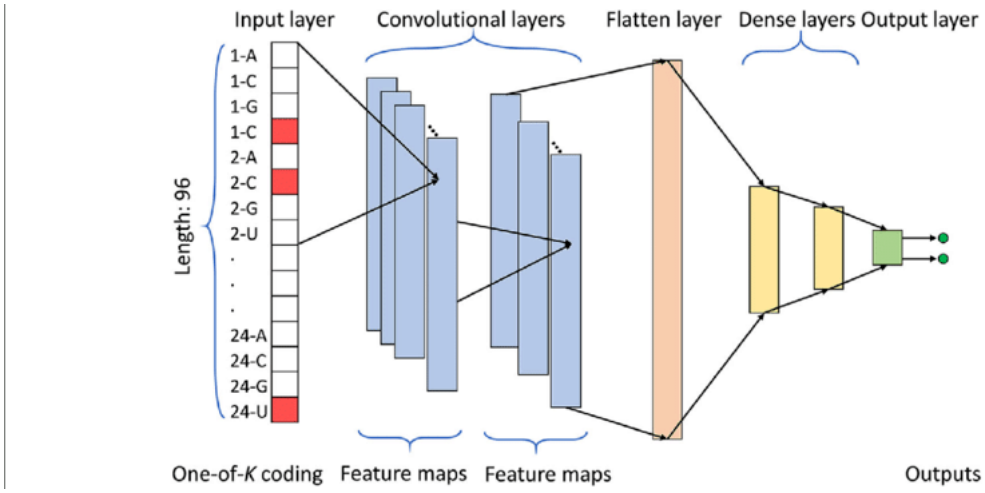


Figura 3.10: https://www.researchgate.net/publication/329201018/figure/fig2/AS:697400008138753@1543284527161/T
architecture-of-our-1D-CNN-model-This-model-consists-of-two-convolutional-
layers.png

Al ser una arquitectura unidimensional, el *kernel* tendrá que tener un tamaño $(1 \times n)$.

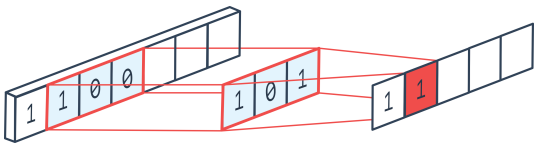


Figura 3.11: https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/blocks/1d-convolution Observamos la entrada de la red (izquierda) a la que se le aplica un kernel (en el centro) que resulta en la convolución de un mapa de características (derecha).

3.1.3.1.2 Bidimensionales Otra de las técnicas convolucionales implementadas en este proyecto son las *Redes Neuronales Convolucionales de 2 Dimensiones*, o *Convolutional Neural Networks 2-Dimensional (CNN-2D)*. Al igual que en el caso de las *CNN-1D*, el filtro se aplica sobre el input de la capa, resultando en *feature maps*, sin embargo, en este caso *kernel* será de bidimensional al ser una convolución de dos dimensiones, por lo que es necesario especificar el tamaño de la matriz que recorrerá la matriz pasada como input a la capa correspondiente.

3.1.4 KNN

El algoritmo *K vecinos más próximos*, o K-Nearest Neighbors (KNN) Altman (1992), es una técnica de aprendizaje supervisado ampliamente utilizada para tareas de clasificación y regresión, es uno de los algoritmos más básicos de *ML* y tiene una gran importancia en el campo de la *Ciencia de Datos* debido a su simplicidad de implementación y a su interpretabilidad.

El algoritmo *KNN* asume que muestras con características parecidas deben pertenecer a la misma clase. O lo que es lo mismo, muestras parecidas deben estar cercanas en el espacio dimensional.

Esta técnica se basa en representar las muestras de un conjunto de datos en base a sus características en un espacio n-dimensional y clasificar la muestra correspondiente como la clase mayoritaria al calcular la distancia de dicha muestra a los k vecinos más próximos. Normalmente la distancia empleada es la *Euclídea*, aunque es posible calcular otras distancias como *Manhattan* o *Chebyshev*.

A medida que se incrementan las dimensiones del espacio resulta más costoso computacionalmente calcular distancias entre los puntos.

Por lo tanto, el único parámetro a configurar es el número de vecinos más cercanos que se calculará en base al punto que queramos clasificar, en la figura 3.12 se muestra un ejemplo de un problema de clasificación aplicando distintos valores al parámetros k.

3.2 Especificaciones técnicas

En esta sección detallaremos las herramientas que han sido utilizadas para poder realizar este proyecto.

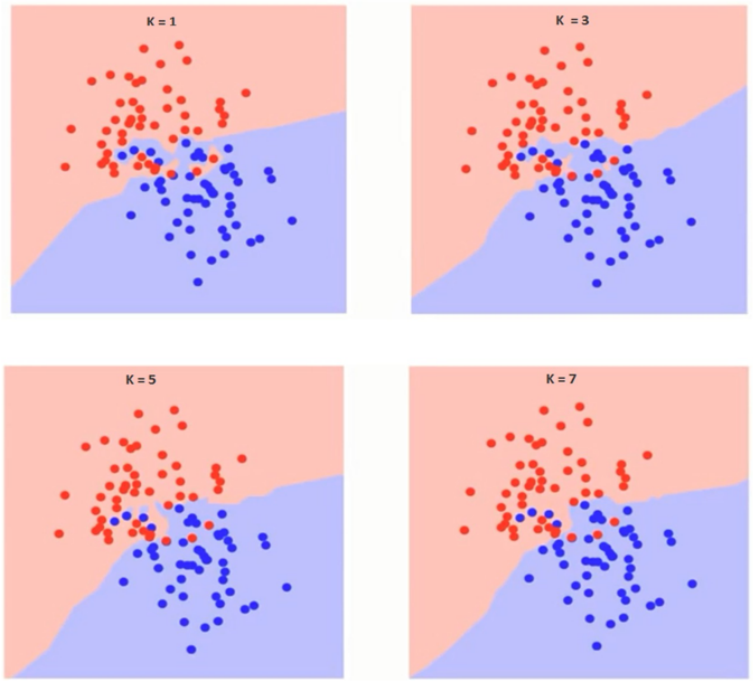


Figura 3.12: https://forum.huawei.com/enterprise/es/data/attachment/forum/202108/20/150033de3l4nyvmcbozh14.pr
KNN aplicado con distintos valores de k

3.2.1 Herramientas utilizadas

En primer lugar se ha utilizado de forma intensiva *Python*, que es un lenguaje de programación interpretado de nivel alto multiparadigma. La versión empleada para el desarrollo del proyecto ha sido 3.9.11. Python (s.f.)

Las liberías más utilizadas han sido:

Pandas: libería que provee de herramientas que permiten el análisis y la manipulación de datos. La versión utilizada para este proyecto es la 1.3.5 pandas dev (s.f.).

Tensorflow: libería utilizada para la implementación de redes neuronales permitiendo su ejecución. Este proyecto se basa en la versión 2.8.0 tensorflow (s.f.).

Sklearn: librería que contiene múltiples modelos predictivos implementados, basado en NumPy, SciPy y Matplotlib. La versión configurada para este proyecto es 1.0.2 scikit learn (s.f.).

XGboost: paquete que contiene la implementación del algoritmo XGBoost, además de múltiples configuraciones como la ejecución en *CPU, GPU y GPU paralelizada*. La importación de esta librería ha sido en base a la versión 1.5.0 dmlc (s.f.).

Respecto a las herramientas *Software* utilizadas destacan:

CUDA: plataforma de computación paralelizada que permite la ejecución de código en *GPU*, esto permite que las redes neuronales puedan entrenarse con mayor rapidez que en *CPU*

debido a la velocidad con la que se realizan las operaciones orientadas a datos en las tarjetas gráficas. Se ha utilizado la versión 11.6 Nvidia (s.f.-a).

Anaconda: distribución open-source que ofrece la flexibilidad de mantener varios entornos con distintas configuraciones y versiones de distinas librerías, facilitando además la migración entre equipos. La versión utilizada ha sido la 4.12.0 Anaconda (s.f.).

Jupyter Notebook: entorno interactivo que permite la creación, edición y ejecución de notebooks de forma local y remota. La versión utilizada para el desarrollo de este proyecto ha sido la 6.4.10 Jupyter (s.f.-b).

Jupyter Lab: interfaz de nueva generación que convive con el entorno Jupyter Notebook, ofrece numerosas funcionalidades como es la navegación entre distintos repositorios dentro de la interfaz. La versión instalada para la realización del proyecto ha sido la 3.3.2 Jupyter (s.f.-a).

DiagramsNet: plataforma utilizada para la confección de figuras mostradas en este documento jGraph (s.f.).

Google Meets: plataforma utilizada para realizar reuniones semanales con el tutor Google (s.f.).

Como sistema de control de versiones se ha utilizado Github Github (s.f.), repositorio donde es posible subir versiones evolutivas del proyecto imprescincible en los proyectos de desarrollo **?**.

3.2.2 Especificaciones del servidor

La mayor parte de los experimentos realizados en este trabajo se han ejecutado bajo un servidor con CPU *Dual AMD Rome 7742* (128 cores) y contando con una GPU *DGX NVIDIA A100* de 40 GigaBytes (*GB*).

4 Metodología

El desarrollo de este proyecto respeta tanto el proceso de desarrollo *Software* como el ciclo de vida de proyectos de *Ciencia de Datos*, por lo que se ha dividido en varias fases acontecidas que definiremos a continuación en base a la planificación mediante un *Diagrama de Gantt*:

4.1 Diagrama de Gantt

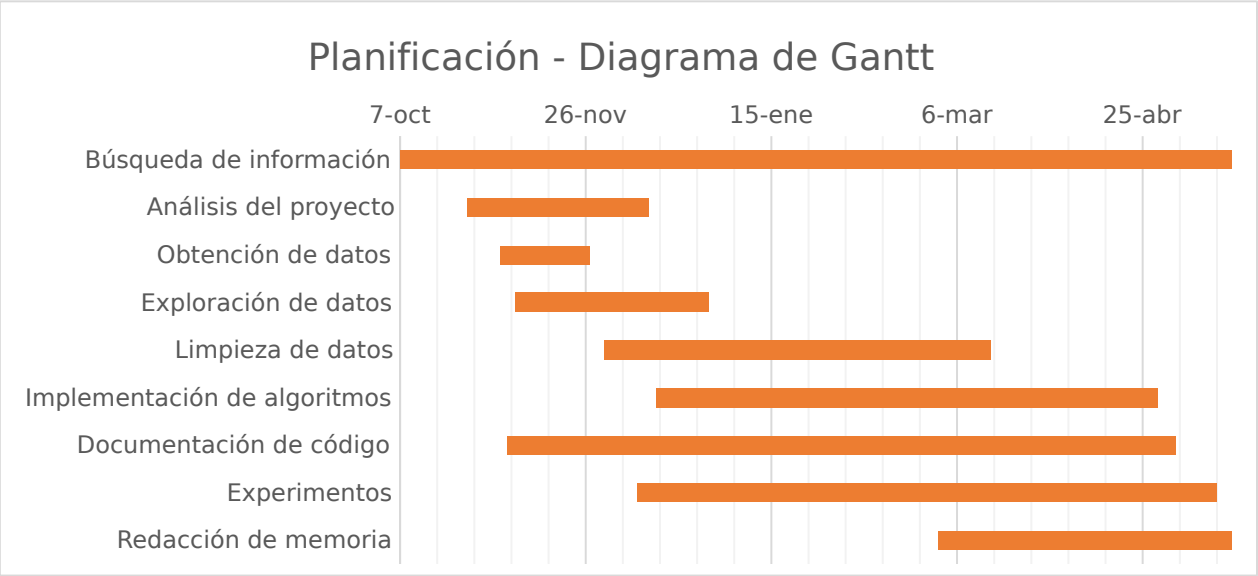


Figura 4.1: Diagrama de Gantt de la planificación del proyecto.

4.2 Diagrama de flujo

En primer lugar se define el flujo de ejecución del proyecto como se muestra en la figura 4.2, donde se han dividido las etapas del proyecto en seis bloques claramente diferenciados. Cada uno de estos está orientado a una disciplina distinta en proyectos de *Ciencia de Datos* y por lo tanto serán detallados en cada una de las subsecciones posteriores.

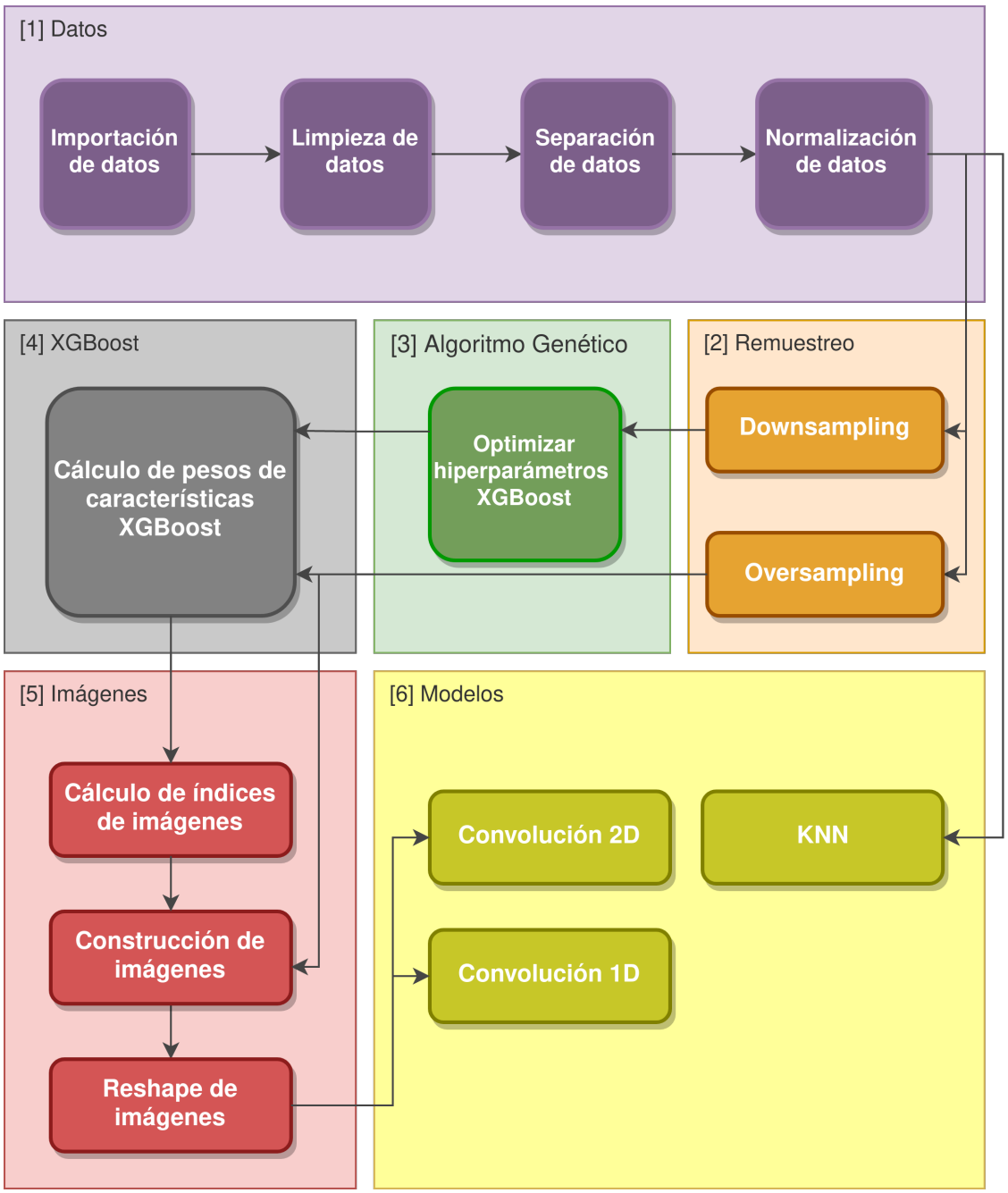


Figura 4.2: Flujo de datos del proceso del proyecto.

4.3 Proceso

En esta sección describiremos cada uno de los pasos que ha seguido el desarrollo del proyecto, desde la búsqueda inicial de datos hasta la implementación final de los algoritmos:

4.3.1 Datos

Como en cualquier proyecto de *Ciencia de Datos* el primer paso contempla la obtención, preparación y estudio de los datos. Este es el proceso en el que más tiempo se invierte en el ciclo de vida de un proyecto, algunos estudios apuntan que alrededor del 80 por ciento del tiempo en un proyecto se destina a esta tarea Projectpro.io (s.f.), por lo que es extremadamente importante aplicar buenas prácticas a la hora de llevar a cabo esta fase.

El conjunto de datos con el que se ha trabajado en este proyecto describe accidentes de tráfico de la ciudad de Madrid en un periodo concreto, desde el año 2019 hasta el 2022. Este dataset ha sido obtenido desde de Datos Abiertos del Ayuntamiento de Madrid (s.f.-a), repositorio donde es posible encontrar distintos datasets relativos a la Comunidad de Madrid.

El número total de registros en este periodo de tiempo es de 60.966 instancias, cada una de las cuales consta de 17 características que se describirán en la tabla 4.1:

Atributo	Descripción
Número de expediente	Identificador del incidente, si varios registros tienen el mismo un número de expediente se consideran como un mismo accidente y cada registro representa cada una de las distintas personas involucradas en él (Conductor, Pasajero o Peatón).
Fecha	Día mes y año en el que se ha producido el incidente.
Hora	Hora y minuto del día.
Localización	Nombre de la calle (si procede).
Número	Número de la calle donde ha ocurrido el incidiente (si procede).
Distrito	Nombre del distrito de Madrid donde ha ocurrido el incidente.
Tipo de accidente	Tipología de accidente, puede ser de diversos tipos (Colisión doble, Colisión múltiple, Alcance, Choque contra obstáculo, Atropello, Vuelco, Caída, Otras causas).
Estado meteorológico	Condiciones climatológicas en el momento del incidente (Despejado, Nublado, Lluvia débil, Lluvia intensa, Granizado o Nevando)
Tipo de vehículo	Clasificación en función del tipo de vehículos, p.e. motocicleta, turismo, cuadriciclo, etc.
Tipo de persona	Rol de la persona involucrada (Conductor, Pasajero o Peatón)
Rango de edad	Intervalo de edad la persona implicada.
Sexo	Sexo de la persona implicada (Hombre o Mujer).
Lesividad	Consecuencias físicas de la persona implicada, si ha necesitado asistencia sanitaria, si ha sido ingresada o si ha sido mortal.
Coordenada X	Coordenada X del accidente en formato *UTM*.
Coordenada Y	Coordenada Y del accidente en formato *UTM*.
Positivo en alcohol	Si la persona implicada ha dado positivo en el control de alcoholemia (S o N).
Positivo en drogas	Si la persona implicada ha dado positivo en el control de estupefacientes (S o N).

Tabla 4.1: Descripción de los datos.

La variable respuesta de este proyecto es la lesividad, siguiendo la descripción del conjunto de datos, existen una serie de valores asignados a ésta categoría. Cada uno de ellos pertenece al menos a uno de los siguientes casos:

- Consecuencias leves: comprende desde aquellas personas que no han resultado heridas hasta aquellas que han necesitado ingresar en un centro hospitalario no más de 24 horas.
- Consecuencias severas: implicados que han requerido un ingreso hospitalario superior a 24 horas.
- Consecuencias fatales: víctimas mortales dentro del margen de las 24 horas posteriores al accidente.

A continuación se muestran los códigos de numeración de la variable respuesta de este proyecto:

- Leves:
  - 1: Atención en urgencias sin posterior ingreso.
  - 2: Ingreso inferior o igual a 24 horas.
  - 5: Asistencia sanitaria ambulatoria con posterioridad.
  - 7: Asistencia sanitaria sólo en el lugar del accidente.
  - 14: Sin asistencia sanitaria.
- Severos:
  - 3: Ingreso superior a 24 horas.
- Fatales:
  - 4: Fallecido 24 horas.

Se pueden consultar los más detalles de las características del dataset descripción del conjunto de datos del portal *Open Data* de Madrid de Datos Abiertos del Ayuntamiento de Madrid (s.f.-b).

1. Limpieza de datos

   Una vez importados los datos en el proyecto, es requisito hacer una limpieza de éstos (*Data Cleaning*), eligiendo las características que serán utilizadas como variables explicativas en las predicciones atendiendo además a los valores de las instancias, ya que pueden contener valores nulos, valores atípicos (*outliers*) o contener valores erróneos. Esta problemática requiere de distintas estrategias a la hora de tratar estos valores.

   En el caso del dataset de accidentes de tráfico de este proyecto se han escogido las siguientes características como variables explicativas *hora, distrito, tipo accidente, estado meteorológico, tipo vehiculo, tipo persona, rango edad, sexo, positivo alcohol, positivo drogas, vehiculos implicados, coordenada x utm, coordenada y utm.*

   Una vez que se han obtenido los predictores con los que trabajarán los modelos, se han eliminado aquellos registros duplicados y aquellos que tuvieran algún valor nulo en alguno de sus predictores. Por lo tanto, las dimensiones finales del conjunto de datos pasarán a ser 12 variables y 54.364 registros con respecto a las 17 características y 60966 filas originales.

2. Transformaciones de datos

   En esta sección detallaremos las transformaciones sobre los datos que han sido necesarias para que los modelos trabajen con un conjunto de datos bien definido y consistente. Gran parte de los modelos de ML necesitan de datos numéricos para poder ser entrenados debido a la necesidad de normalizar estos valores para una correcta optimización del modelo, por lo tanto se requiere de una serie de transformacíones que conviertan las variables categóricas a variables numéricas.

   En primer lugar ha sido necesario transformar las columnas *coordenada x utm* y *coordenada y utm* a números enteros para que los modelos puedan interpretarlas y normalizarlas, ya que el rango de estas variables es del orden de 7 a 10 dígitos. Inicialmente estos valores estaban establecidos como *String*, evitando cualquier formato de decimales estandarizado (utilizando puntos y comas indistintamente en distintas posiciones de los dígitos), debido a esto ha sido necesario crear un proceso que analizase cada casuístca y los tradujese a un formato estandarizado.

   Las columnas *positiva alcohol* y *positiva drogas* se han unido en una nueva columna debido a la cantidad de valores nulos que existía en la segunda variable, por lo que se crea una nueva columna que hace referencia a la intoxicacion etílica o de estupefacientes.

   En la tabla 4.2 se define el resto de codificaciones aplicadas sobre el resto de variables del modelo para transformar estos campos a formato numérico.

Característica	**Tipado**
Lesividad	0: Accidentes leves (*1, 2, 5, 6, 7, 14*). 1: Accidentes severos (*3*). 2: Accidentes fatales (*4*).
Hora	1: Noche (*6 PM - 6 AM*). 2: Día (*6 AM - 6 PM*).
Distrito	Numeración en función de orden de aparición.
Tipo Accidente	1: Colisión fronto-lateral. 2: Alcance. 3: Colisión lateral. 4: Choque contra obstáculo fijo. 5: Colisión múltiple. 6: Caída. 7: Atropello a persona. 8: Colisión frontal. 9: Otro. 10: Solo salida de la vía. 11: Vuelco. 12: Atropello a animal.

	13: Despeñamiento.
Estado Meteorológico	1: Despejado. 2: Nublado. 3: Lluvia débil. 4: Lluvia intensa. 5: Granizando. 6: Nevando. 7: Se desconoce.
Tipo Vehículo	Numeración en función de orden de aparición.
Tipo Persona	1: Conductor. 2: Pasajero. 3: Peatón.
Rango Edad	1: Menores de 18 años. 2: De 18 a 25 años. 3: De 25 a 65 años. 4: Mayores de 65 años. 5: Edad desconocida.
Sexo	1: Hombre. 2: Mujer. 3: Desconocido.
Positivo	1: Sí. 2: No.

Tabla 4.2: Transformaciones aplicadas a los datos.

3. Análisis de datos

   Una de las fases más importantes antes de comenzar el modelado en cualquier proyecto *Data Science* es el análisis de datos. Este proceso tiene como objetivo la descripción de los datos, identificación de *outliers*, valores erróneos y tendencias que se puedan dar en ellos.

   Comenzaremos analizando los datos que correspondan a cada una de las tres posibles clases (leves, severos y fatales) realizando un histograma 4.3. Como podemos observar, nos encontramos ante un dataset claramente desbalanceado con respecto a la variable respuesta, contando con 53.009 accidentes leves, 1.271 severos y 84 fatales.

   Un conjunto de datos desbalanceado se define como aquel que contiene un número de instancias mucho mayor de determinadas clases con respecto al resto Sun y cols. (2009). Esto se convierte en un problema para los modelos de clasificación ya que estos modelos tenderán a predecir las muestras como aquellas que pertenecen a las mayoritarias sobre

el conjunto de test debido a que las reglas de classificación de las clases menos numerosas tienden a ser ignoradas.

El desbalanceo de datos es un problema ampliamente estudiado a lo largo de los años y existen numerosos métodos orientados a tratar este problema mediante distintas técnicas de muestreo Haixiang y cols. (2017), por lo que será necesario aplicar distintas técnicas sobre el conjunto de datos.

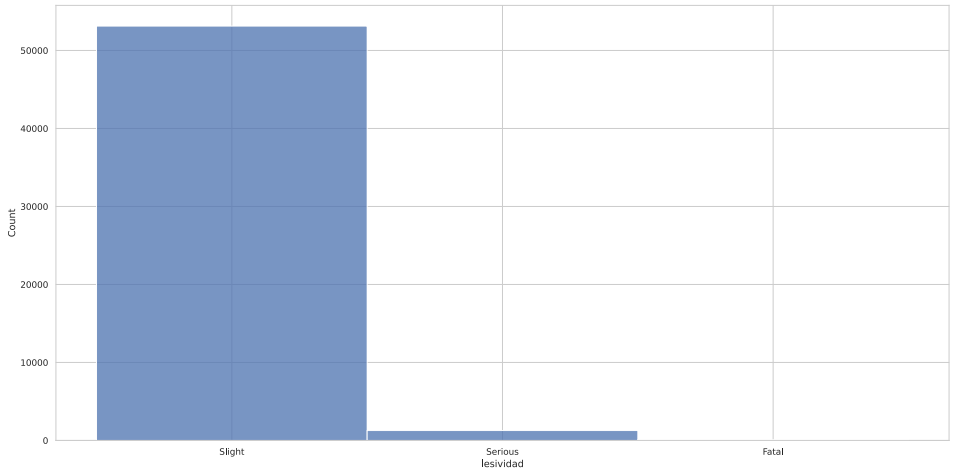


Figura 4.3: Histograma de tipo de accidente.

Analizando la matriz de correlación 4.4 se puede observar que la variable *coordenada x utm* y la variable *distrito* están relativamente correlacionadas. Por lo que

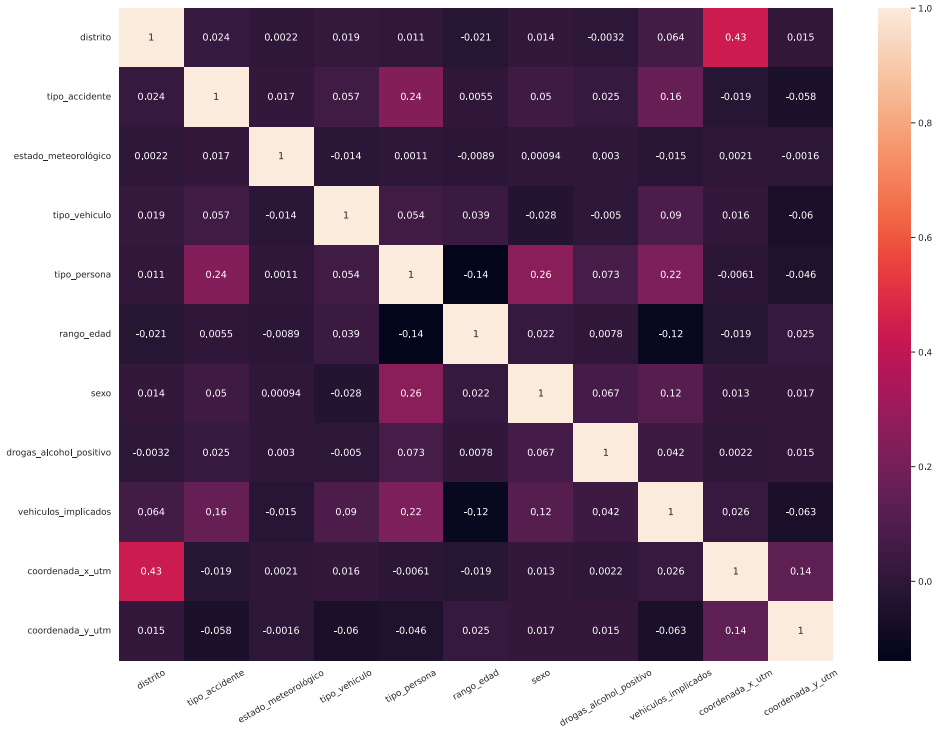


Figura 4.4: Matriz de correlación para los predictores escogidos del dataset.

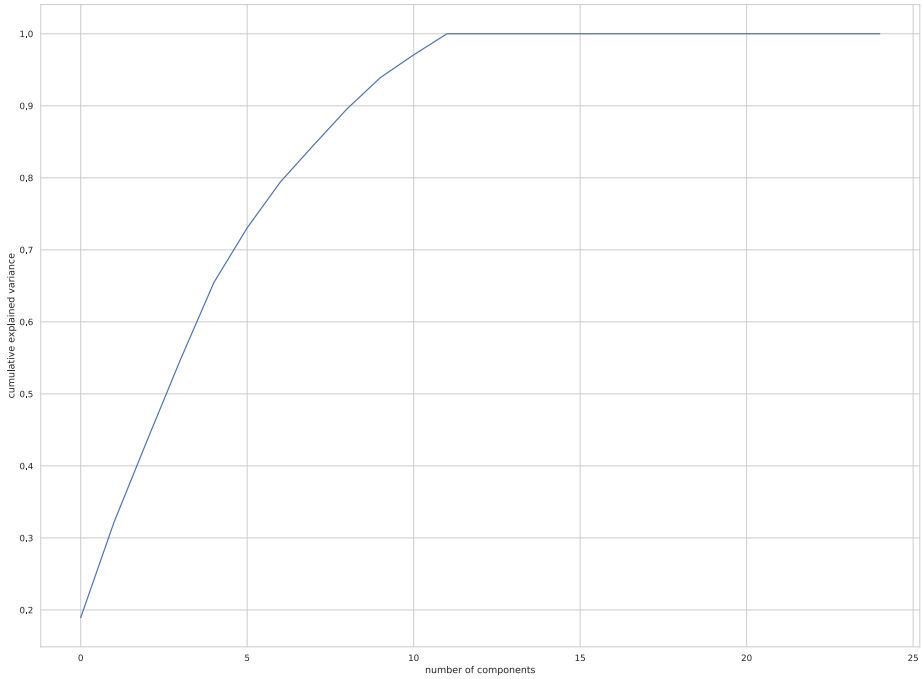


Figura 4.5: PCA aplicado a los datos.

4. Normalización de datos

   La normalización de datos es un proceso necesario a la hora de obtener buenos resultados en los modelos predictivos de ML. Cuando un modelo es entrenado, es bastante probable existan características en distintas escalas. En consecuencia, las características que contengan un rango de valores numérico más alto, ya sea por la naturaleza de la variable o porque se encuentren en otra escala, dominarán a aquellas características que contengan un rango menor a la hora de aplicar modelos ML que sean sensibles a la desnormalización de datos, como por ejemplo ML o KNN School (s.f.).

   El proceso de normalización tiene como objetivo minimizar el *bias* de aquellas características cuya contribución sea mayor a la hora de encontrar patrones entre los datos. Existen distintas técnicas de normalización como por ejemplo Mean Centered (MC), Variable Stability Scaling (VSS) o Min-Max Normalization (MMN) entre otras Singh y Singh (2020).

   En este proyecto se ha utilizado la normalización Z-Score Normalization (ZSN) debido a las propiedades que ofrece, ZSN que utiliza la media y la desviación típica para reescalar los datos de tal forma que la distribución de ellos esté definida por una media de

cero y una desviación típica unitaria, consiguiendo representaciones de acuerdo a una distribución normal.

Los resultados obtenidos después de aplicar esta técnica se pueden interpretar como la distancia de cada valor con respecto a la media.

$$x^* = \frac{x-u}{\sigma}$$

Donde x^* es una muestra de una característica de los datos, u es la media total de dicha característica y σ es la desviación típica total de los valores de la característica.

5. Separación de datos

   La siguiente fase en cualquier modelo de es la separación de datos (*split*). Esta fase consiste en dividir el conjunto total de datos en al menos dos subconjuntos, uno de entrenamiento y otro de test. El modelo se entrenará en base a los datos de entrenamiento y se evaluará con los datos de test, los resultados sobre este conjunto permitirán comparar los modelos en base a las predicciones sobre las muestras que nunca han visto.

   Comúnmente la proporción de datos de entrenamiento y test está establecida en *0.8* y *0.2* respectivamente, en la figura 4.6 se muestra una representación visual de esta división.

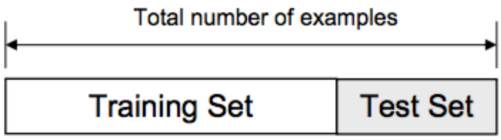


Figura 4.6: https://towardsdatascience.com/train-test-split-and-cross-validation-in-python-80b61beca4b6. División de un conjunto de datos en datos de entrenamiento y test.

4.3.2 Remuestreo

Estas técnicas permiten operar sobre el conjunto de datos para balancearlo, con el objetivo de que el modelo no se vea muy afectado debido a la diferencia de las clases mayoritarias respecto a las minoritarias. En este proyecto se han hecho uso de las siguientes técnicas de remuestreo:

1. Downsampling: tiene como objetivo balancear el conjunto de datos para que todas las clases tengan el mismo número de muestras igualando el número de muestras de las clases mayoritarias a aquellas clases que contienen menos muestras, descartando aquellas que sobrepasen este límite.

   Liu y cols. (2009)

No se redactará más hasta que se compruebe en los experimentos que esa técnica nos sirve

2. Generación de datos sintéticos: técnica que permite generar datos artificiales en base a los límites que separan unas clases de otras. Se ha hecho uso de la teécnica Borderline Synthetic Minority Over-sampling Technique 2 (SMOTE-II) Chawla y cols. (2002). Esta técnica selecciona los vecinos más cercanos de la misma clase y genera nuevas muestras en base al espacio entre la clase minoritaria y sus vecinos más cercanos.

   SMOTE-II ha sido utilizado para generar más muestras artificiales de los accidentes pertenecientes a las clases minoritarias (*severos* y *graves*).

Una vez aplicadas las técnicas de remuestreo al conjunto de datos para balancearlo, es útil observar cuál es la representación espacial de los datos sintéticos generados por SMOTE-II con respecto a los datos originales mediante t-Distributed Stochastic Neighbor Embedding (t-SNE) Van der Maaten y Hinton (2008). t-SNE es una técnica que permite visualizar datos multidimensionales proyectando cada muetra en un espacio bidimensional o tridimensional. Este método es una variante simplificada y optimizada del algoritmo Stochastic Neighbor Embedding (SNE).

Mientras que SNE convierte las distancias Euclídeas entre las muestras a probabilidades condicionales que presenten similaridades entre sí mediante probabilidades Gaussianas, t-SNE modifica su función de coste de tal forma que utiliza una distribución t-Student, permitiendo además trabajar con gradientes simplificados.

En la figura 4.7 se muestran las proyecciones de t-SNE (tanto la representación bidimensional como la tridimensional) sobre el conjunto de datos de entrenamiento original y aquellos que han sido generados artificialmente por SMOTE-II.

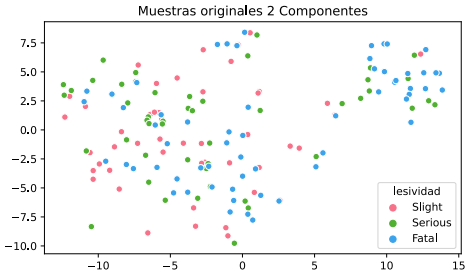
4.3.3 Algoritmo Genético

Para una correcto entrenamiento de un modelo predictivo es necesario optimizar los hiperparámetros con los que el modelo será entrenado. En este proyecto es necesario optmizar los hiperparámetros del algoritmo XGBoost.

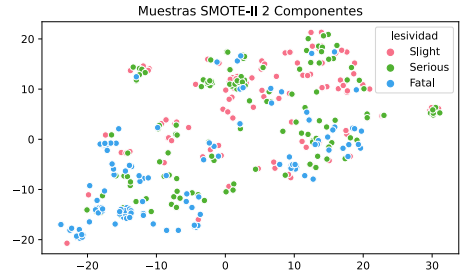
Para ello se ha hecho uso de algoritmos genéticos Jain (s.f.), donde cada individuo perteneciente a la población de una generación está formado por una configuración de hiperparámetros específica, de tal forma que a lo largo de las iteraciones los individuos evolucionarán mediante el cruce y la mutación para dar lugar a nuevas configuraciones de hiperparámetros optimizadas Jiang y cols. (2019).

Debido al coste computacional que tendría optimizar todos los parámetros por la magnitud del espacio de búsqueda y al no ser necesario tener en cuenta todos, se han seleccionado aquellos que más influencia tienen en el entrenamiento del modelo. Los hiperparámetros de los que consta cada individuo de la población son:

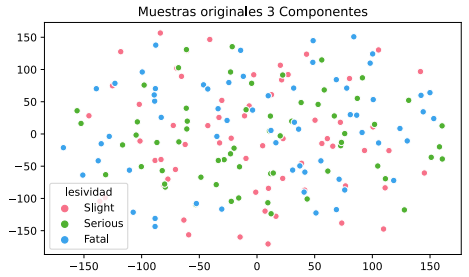
1. Profundidad máxima: es la máxima altura que puede tomar el árbol. Si el árbol de decisión alcanza demasiada profundidad tenderá al *overfitting* ya que aprenderá relaciones complejas entre los datos que pueden deberse a ruido en los datos de entrenamiento.



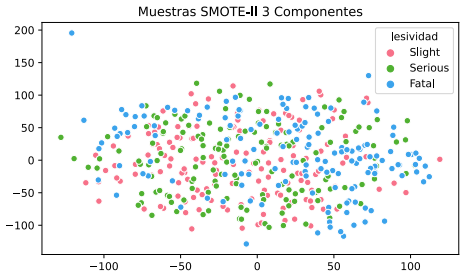
(a) TSNE de 2 componentes aplicado a los datos originales.



(b) TSNE de 2 componentes aplicado a los datos generados por SMOTE-II.



(c) TSNE de 3 componentes aplicado a los datos originales.



(d) TSNE de 3 componentes aplicado a los datos generados por SMOTE-II.

Figura 4.7: TSNE de 2 y 3 componentes aplicado a los datos originales y a los generados sintéticamente (SMOTE-II).

2. Peso mínimo de los hijos: es el mínimo peso que se establece a la hora de crear un nuevo nodo en el árbol. Cuando se entrena un árbol de decisión éste genera nuevos nodos en base a máxima separabilidad de los datos de entrenamiento en cada nivel. Con el límite de peso de los hijos establecemos un umbral mínimo de muestras que deben pertenecer a un nodo para realizar la separación. Un valor bajo en este parámetro permitirá crear nodos con menos muestras y por lo tanto el modelo tenderá al *overfitting*.

3. ETA: tamaño de paso utilizado para aplicar descenso por gradiente para minimizar la pérdida de los árboles anteriores.

La inicialización y mutación de los valores de los individuos viene dada por una limitación mínima y máxima que pueden, ya que, si no se contemplase, los valores podrían tomar valores extremos reduciendo el rendimiento de entrenamiento y resultados. Por lo tanto los individuos variarán sus parámetros tomando un valor aleatorio de estos rangos. Los parámetros de las nuevas soluciones mutarán dentro de unos límites específicos para cada parámetro establecidos como máximos y mínimos para que no tomen valores extremos. La tabla **?** muestra los límites de los hiperparámetros.

Hiperparámetro	**Inicialización**	**Mutación**
Profundidad Máxima	[1, 25]	[-6, 6]
Peso mínimo de los hijos	[0.01, 20.0]	[-7, 7]
ETA	[0.01, 1]	[-0.3, 0.3]

Tabla 4.3: Límites de inicialización y mutación de los hiperparámetros de los individuos.

Una vez inicializados aleatoriamente los TODO:X individuos especificados en la población, éstos se evaluarán instanciando TODO:X modelos XGBoostcon los valores de cada individuo en la población. El conjunto de datos sobre el que entrenará cada instancia XGBoost será el conjunto de entrenamiento TODO:XX. La función *fitness* que evaluará cada individuo será la métrica TODO:F1-X aplicada sobre el conjunto de test obtenido en el proceso de separación de datos.

TODO: No se redactará más hasta tener claro cuál es la mejor forma de entrenar el XGBoost (función de validación y datos de entrenamiento)

TODO: Las siguientes dos figuras irían mejor en Resultados? En la figura [5.1] se pueden visualizar la evolución de los parámetros del mejor individuo en cada generación.

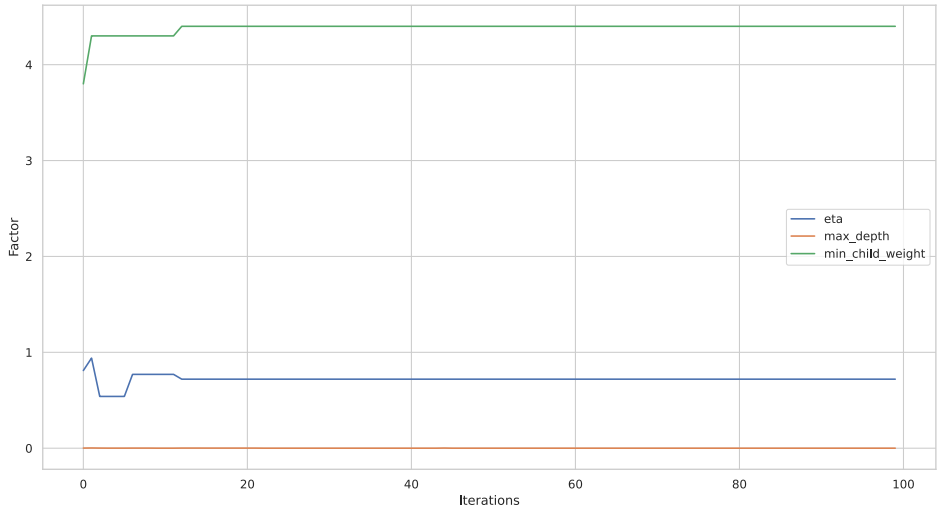


Figura 4.8: Evolución de hiperparámetros a lo largo de las iteraciones.

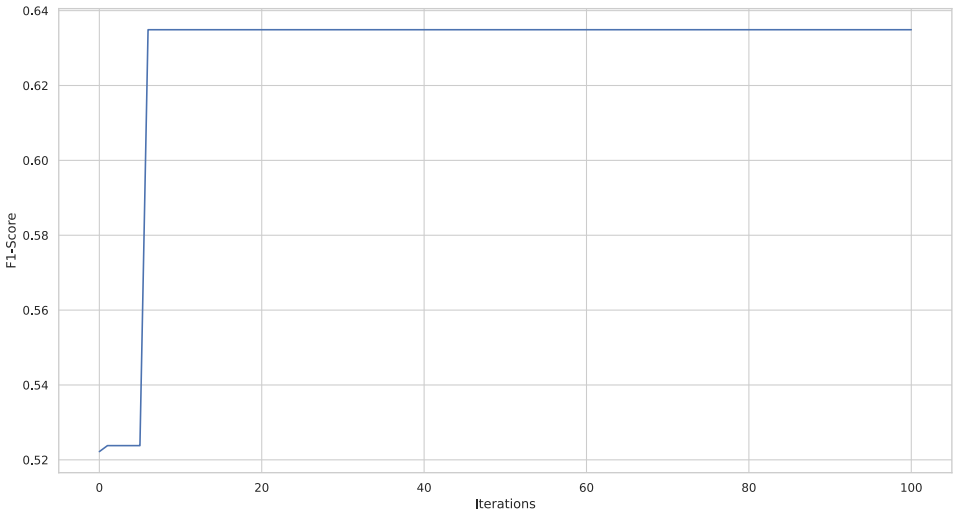


Figura 4.9: Evolución del macro F1 score a lo largo de las iteraciones.

En la tabla 4.4 se pueden observar el mejor individuo obtenido después de *TODO: X* generaciones.

Hiperparámetro	Valor
Profundidad Máxima	TODO
Peso mínimo de los hijos	TODO
ETA	TODO

Tabla 4.4: Mejores parámetros de XGBoost tras aplicar el algoritmo genético.

4.3.4 XGBoost

TODO: Decir que se entrena con los datos train con downsampling si es el mejor resultado al final en los experimentos

Una vez se han optimizados los hiperparámetros del XGBoost, se obtiene la importancia de cada clase mediante el cálculo de los pesos. Brownlee (s.f.). Estos pesos son calculados por XGBoost, e indican el grado de importancia que ha tenido cada característica del conjunto de datos a la hora de entrenar el árbol. Aquellas caracerísticas a las que se les asigne más peso habrán sido aquellas que han jugado un papel clave en las decisiones de los árboles.

Los pesos son asignados para cada característica del conjunto de datos con el que ha sido entrenado XGBoost de tal forma que permite realizar un análisis comparativo de los atributos.

Generally, importance provides a score that indicates how useful or valuable each feature was in the construction of the boosted decision trees within the model. The more an attribute is used to make key decisions with decision trees, the higher its relative importance.

This importance is calculated explicitly for each attribute in the dataset, allowing attributes to be ranked and compared to each other.

Importance is calculated for a single decision tree by the amount that each attribute split point improves the performance measure, weighted by the number of observations the node is responsible for. The performance measure may be the purity (Gini index) used to select the split points or another more specific error function.

The feature importances are then averaged across all of the the decision trees within the model.

4.3.5 Imágenes

Debido a que las CNN aprenden patrones sobre la entrada de datos como imágenes, es importante que éstas estén construidas de tal forma que maximice la representación de la información, es decir, es necesario aplicar técnicas que posicionen cada característica en un pixel de la matriz maximizando la representación de los accidentes (TODO: **relamente no entiendo muy bien la motivación que ha seguido la gente de TASPCNN a la hora de construir las imágenes, falta entender por qué lo hacen así, no lo he encontrado en el paper.**)

Por lo tanto, el siguiente paso será transferir cada una de las muestras tabulares de los accidentes a una imagen en escala de grises, donde a cada característica se le asignará una posición de la matriz (*5 x 5*), de tal forma que éstas sean la entrada a las *CNN*.

Para este objetivo se aplicará el algoritmo FV2GI propuesto en el artículo Zheng y cols. (2019), que asigna las características de una muestra en representación tabular en función de la importancia que éstas presenten en una jerarquía.

Tal y como se propone en Kopelias y cols. (2007) las características que provocan un accidente de tráfico pueden englobarse en una serie de elementos o categorías principales, concretamente: *características del conductor, el estado de la carretera, características propias del vehículo y condiciones del ambiente*. Se muestra en la tabla 4.5 las asignaciones de las variables explicativas del conjunto de datos en función de esta jerarquía.

Categoría	**Variable**
Accidente	Coordenada X.
	Coordenada Y.
	Hora.
	Vehículos implicados.
Ambiente	Estado meteorológico.
Carretera	TODO: ESTO NO DEBERIA ESTAR AQUI Tipo de accidente
	Distrito
Conductor	Tipo Persona.
	Sexo.
	Rango Edad.
	Positivo.

Tabla 4.5: Transformaciones aplicadas a los datos.

Una vez definida la jerarquía de características y los pesos asociados a cada una de ellas gracias al algoritmo XGBoost se construyen las imágenes de acuerdo al criterio *FV2GI*. Este proceso consta de los siguientes pasos:

1. Generación de n imágenes inicializadas a 0, donde n es el número de muestras tabulares en el dataset.
2. Asignación de una fila a cada padre en función de su peso.
3. Asignacíón de la la posición de cada característica hija en función de su peso dentro de la fila de su padre.

Se deben tomar las siguientes consideraciones a la hora de aplicar *FV2GI*:

1. La importancia de un padre viene dada por Íla suma de los pesos de las características hijas, en la tabla 4.6 se observa el cálculo de cada característica para las categorías principales.

2. La asignación de las filas de los padres se realiza de forma intercalada en función de su peso. Aquel padre que más importancia tenga irá posicionado en la fila central de la matriz, el segundo padre irá posicionado por encima de éste y el tercero por debajo y así sucesivamente, de tal forma que se irá creando una estructura en la que dichos padres se interpolan entre las filas en función de su peso.

3. Una vez se han asignado los padres a las filas se realiza el mismo procedimiento con las características hijas a nivel de columna. Donde la característica que más importancia tenga irá posicionada en el centro, la segunda irá posicionada a su izquierda, la tercera a la derecha y así sucesivamente.

Categoría	**Peso Categoría**	**Característica**	**Peso Característica**
Accidente	TODO: 1	Coordenada X	TODO: 1
		Coordenada Y	TODO: 1
		Hora	TODO: 1
		Vehículos implicados	TODO: 1
Ambiente	TODO: 1	Estado meteorológico	TODO: 1
Carretera	TODO: 1	Tipo de accidente	TODO: 1
		Distrito	TODO: 1
Conductor	TODO: 1	Tipo Persona	TODO: 1
		Sexo	TODO: 1
		Rango Edad	TODO: 1
		Positivo	TODO: 1

Tabla 4.6: Cálculo de pesos de características y categorías mediante XGBoost.

El resultado de aplicar ambas técnicas da lugar a la representación en forma matricial de los accidentes, como se puede observar en la figura 4.10.

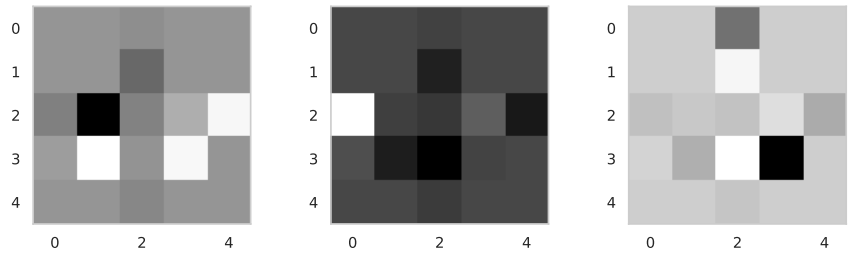


Figura 4.10: Representación de las muestras de accidentes en forma de matriz de grises.

4.3.6 Modelos

En esta sección analizaremos los modelos utilizados en este proyecto con el objetivo de realizar un estudio comparativo entre las arquitecturas.

1. KNN

   El método KNN servirá como referencia para testar el rendimiento del resto de modelos. Al ser un método que se aplica sobre un conjunto de datos tabular original, no hará uso de las imágenes generadas.

   Es necesario optimizar los parámetros de este algoritmo para conseguir un buen rendimiento, por lo que aplicaremos la técnica *Grid Search* Sklearn (s.f.). Esta técnica permite la optimización de los valores de los hiperparámetros mediante una búsqueda exhaustiva en un espacio de búsqueda definido por el usuario, probando distintas combinaciones hasta cubrirlo por completo.

   Se puede observar en la tabla 4.7 los mejores parámetros para *KNN* después de haber ejecutado *Grid Search* sobre el conjunto de entrenamiento.

Atributo	Valor
Número de vecinos	21
Tipo de distancia	Minkowski

Tabla 4.7: Mejores parámetros de KNN tras aplicar GridSearch.

2. CNN

   Una vez definidos los pasos que sigue el entrenamiento de una *NN* y las características propias de las *CNNs*, podemos analizar las arquitecturas de las *CNNs* implementadas que se han aplicado en este proyecto. Cabe mencionar que el funcionamiento de ambas *CNNs* únicamente difiere en el tamaño del *kernel* (unidimensionales o bidimensionales según sea el caso correspondiente) y la forma en la que éste se desplaza debido a su dimensionalidad, por lo tanto detallaremos ambas arquitecturas de la misma forma.

   La arquitectura consta de cuatro capas convolucionales con tamaños de kernel (*1 x 3*) en caso de las CNN-1D y de tamaño (*3 x 3*) para las CNN-2D. Estos kernels se proyectarán en *256* canales para formar el filtro convolucional asociado a cada capa. A la salida de cada uno de los mapas de características se aplica un proceso de BN.

   El *padding* del kernel se ha establecido en 1 para ambos tipos de redes, de tal forma que las convoluciones se aplícarán añadiendo ceros en los límites de las imágenes, y los *strides* en (*1*) para las CNN-1D y (*1, 1*) para las CNN-2D, por lo tanto el desplazamiento de los kernels se hará píxel a píxel tanto en las CNN-1D como en las CNN-2D.

   A la salida de cada capa convolucional se aplica la función de activación Rectified Linear Unit (ReLU), que se comporta devolviendo el valor 0 para aquellas entradas que sean negativas y el valor original para aquellas que sean positivas, se puede apreciar el

comportamiento en la figura 4.11. El rendimiento de esta función de activación en el campo de las imágenes está ampliamente extendido, además, evita la probabilidad de aparición del gradiente evanescente Glorot y cols. (2011).

$$f(x) = \begin{cases} 0 & \text{if } x <= 0 \\ x & \text{if } x > 0 \end{cases}$$

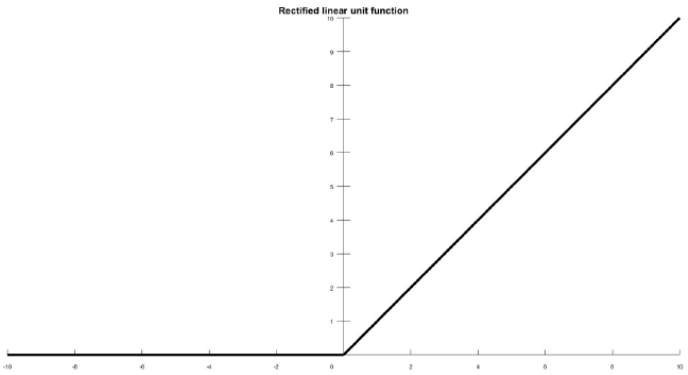


Figura 4.11: https://www.researchgate.net/journal/Transport-in-Porous-Media-1573-1634. Función ReLU

La salida de la última capa de la convolución transformará el mapa de características de tamaño (*5 x 5*) generada a una capa Flatten, que aplanará la matriz de tal forma que la convertirá en un vector unidimensional de (*1 x 25*). A continuación se aplicará una capa densa que conectará cada uno de los *25* nodos de la capa *Flatten* con los *128* nodos de la capa densa, que generará los logits antes de aplicar la última función de activación *Softmax* que devolverá la clase predicha.

Para ejemplificar la arquitectura de las NN propuestas se muestra el caso de la CNN-2D en la figura 4.12

Feurer y cols. (2020)

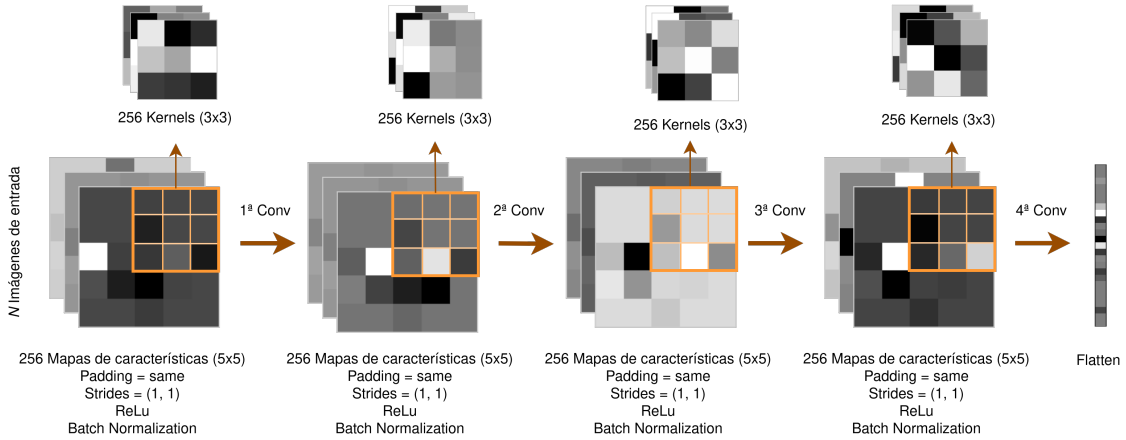


Figura 4.12: Arquitectura de la CNN-2D mostrando una kernels aprendidos durante el entrenamiento.

5 Resultados

En esta sección se detallarán los resultados de los experimentos y se realizará un análisis comparativo entre ellos.

En primer lugar....

5.1 Algoritmo Genético

NOTA Ponemos aquí las figuras 5.1 y 5.2 de evolución de hiperparámetros además de tabla de resultados 4.4 en lugar de en metodologías?

En la figura [5.1] se pueden visualizar la evolución de los parámetros del mejor individuo en cada generación.

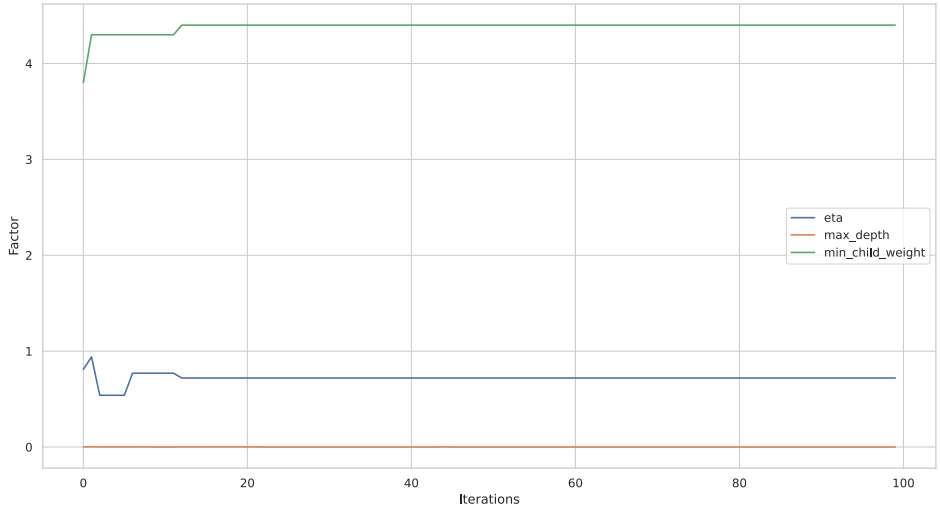


Figura 5.1: Evolución de hiperparámetros a lo largo de las iteraciones.

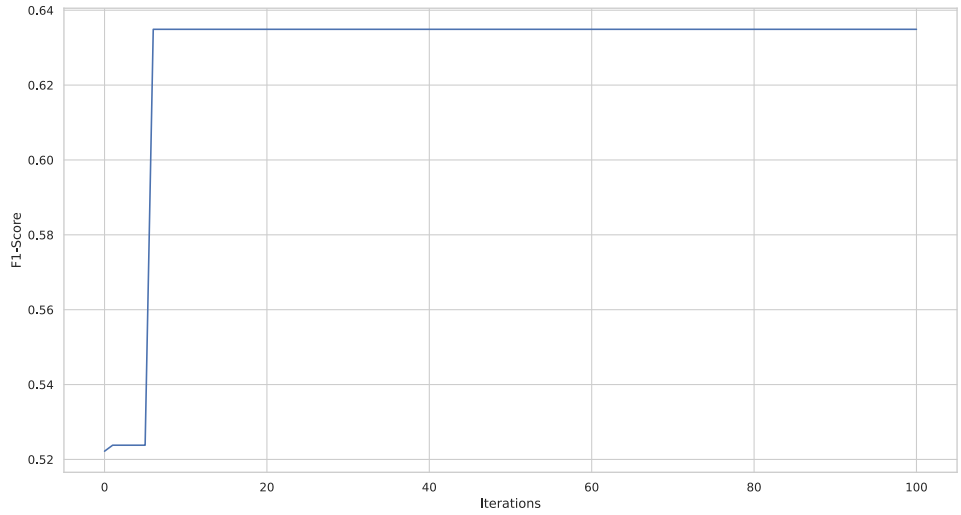


Figura 5.2: Evolución del macro F1 score a lo largo de las iteraciones.

En la tabla 4.4 se pueden observar el mejor individuo obtenido después de *TODO: X* generaciones.

5.2 XGBoost

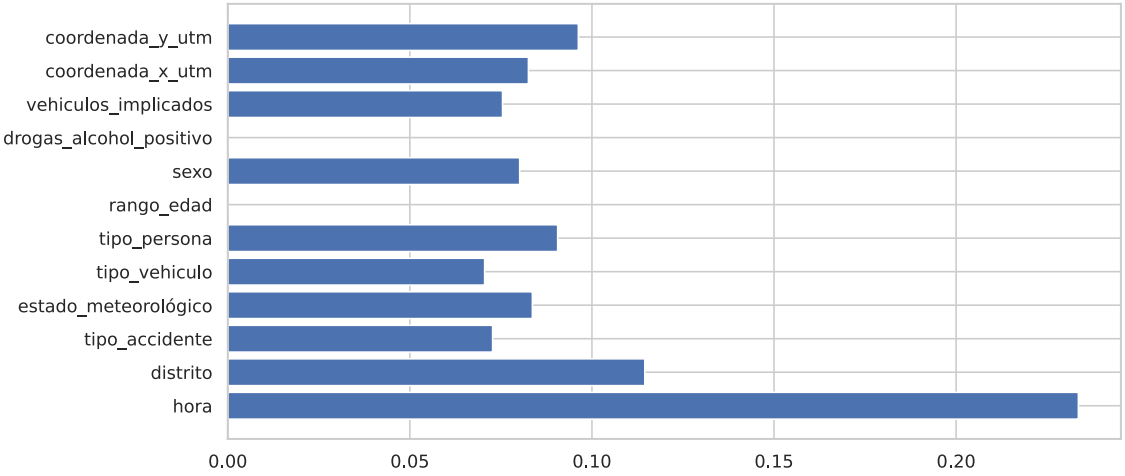


Figura 5.3: Pesos asignados por XGboost a las características.

NOTA Ponemos aquí las figuras de los pesos calculados 5.3 y la tabla de cáclulo de características 4.6?

5.3 Entrenamiento de modelos

5.3.1 Gráficas de entrenamiento

Explicar las gráficas de evolución F1-score de las redes neuronales.

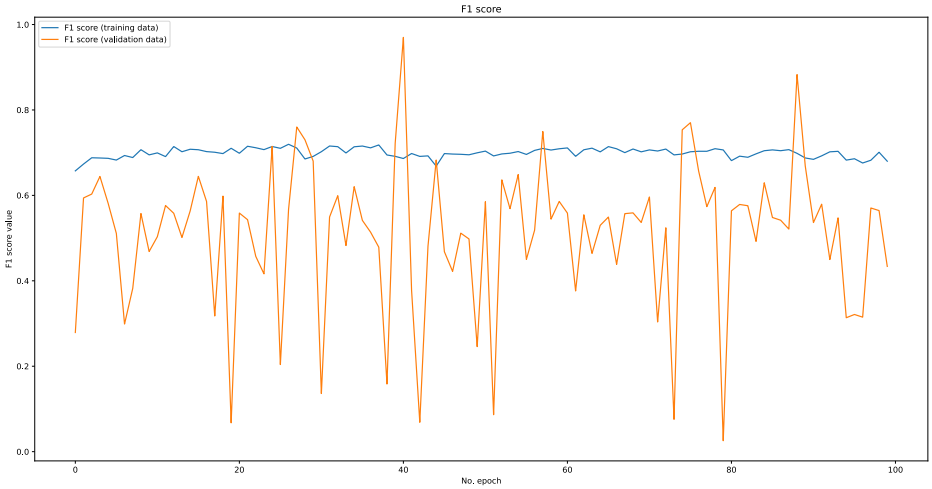


Figura 5.4: Comparación de TODO(tipo) F1 score en validación y test CNN-1D.

5.3.2 Reportes de clasificación

Explicar los reportes de clasificación con datos de **ENTRENAMIENTO.**

alo	precision	recall	f1score	support
Slight	0.9881019830028328	0.32859161563824774	0.49317780134323075	10615.0
Serious	0.03694921952183363	0.677536231884058	0.07007682218474799	276.0
Fatal	0.0012987012987012987	0.3	0.002586206896551724	10.0
accuracy	0.33740023851022843	0.33740023851022843	0.33740023851022843	0.337400238510
macro avg	0.3421166346077893	0.4353759491741019	0.18861361014151015	10901.0
weighted avg	0.9631147161889811	0.3374002385102284	0.48201535879739016	10901.0

Tabla 5.1: Métricas CNN-1D.

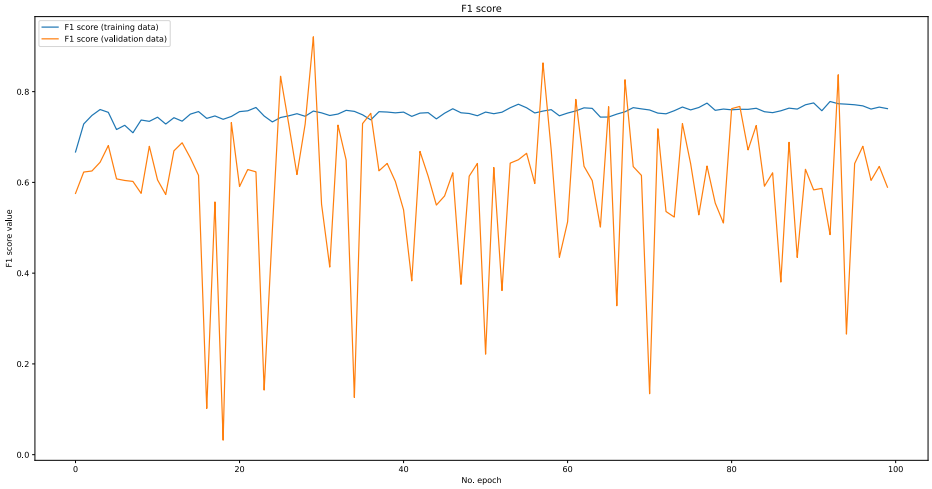


Figura 5.5: Comparación de TODO(tipo) F1 score en validación y test CNN-2D.

	precision	recall	f1-score	support
Slight	0.9881019830028328	0.32859161563824774	0.49317780134323075	10615.0
Serious	0.03694921952183363	0.677536231884058	0.07007682218474799	276.0
Fatal	0.0012987012987012987	0.3	0.002586206896551724	10.0
accuracy	0.33740023851022843	0.33740023851022843	0.33740023851022843	0.337400238510228
macro avg	0.3421166346077893	0.4353759491741019	0.18861361014151015	10901.0
weighted avg	0.9631147161889811	0.3374002385102284	0.48201535879739016	10901.0

Tabla 5.2: Métricas Naive Bayes.

	precision	recall	f1-score	support
Slight	0.9881019830028328	0.32859161563824774	0.49317780134323075	10615.0
Serious	0.03694921952183363	0.677536231884058	0.07007682218474799	276.0
Fatal	0.0012987012987012987	0.3	0.002586206896551724	10.0
accuracy	0.33740023851022843	0.33740023851022843	0.33740023851022843	0.337400238510
macro avg	0.3421166346077893	0.4353759491741019	0.18861361014151015	10901.0
weighted avg	0.9631147161889811	0.3374002385102284	0.48201535879739016	10901.0

Tabla 5.3: Métricas SVC.

	precision	recall	f1-score	support
Slight	0.9881019830028328	0.32859161563824774	0.49317780134323075	10615.0
Serious	0.03694921952183363	0.677536231884058	0.07007682218474799	276.0
Fatal	0.0012987012987012987	0.3	0.002586206896551724	10.0
accuracy	0.33740023851022843	0.33740023851022843	0.33740023851022843	0.337400238510
macro avg	0.3421166346077893	0.4353759491741019	0.18861361014151015	10901.0
weighted avg	0.9631147161889811	0.3374002385102284	0.48201535879739016	10901.0

Tabla 5.4: Métricas CNN-2D.

5.3.3 Matrices de confusión

Explicar las matrices de confusión con datos de **ENTRENAMIENTO.**

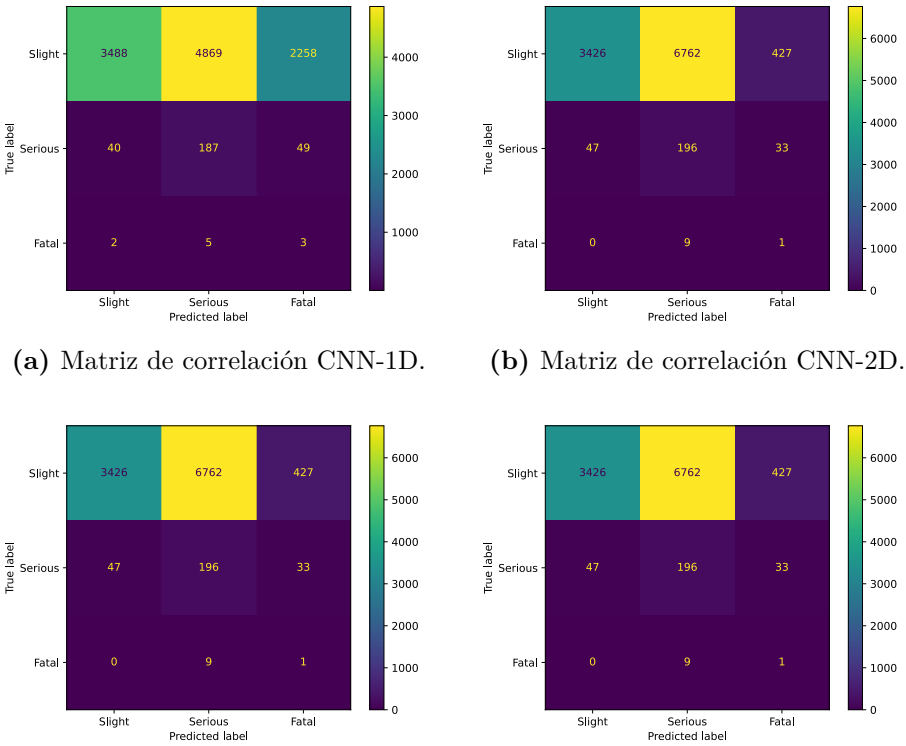
5.3.4 Comparativa entre modelos

Explicar la comparativa de modelos con datos de **ENTRENAMIENTO.**

5.4 Predicciones de modelos

5.4.1 Reportes de clasificación

Explicar los reportes de clasificación con datos de **TEST.**



(a) Matriz de correlación CNN-1D. **(b)** Matriz de correlación CNN-2D.

(c) Matriz de correlación NB. **(d)** Matriz de correlación SVC.

Figura 5.6: Matriz de correlación sobre el conjunto de test sobre los modelos aplicados.

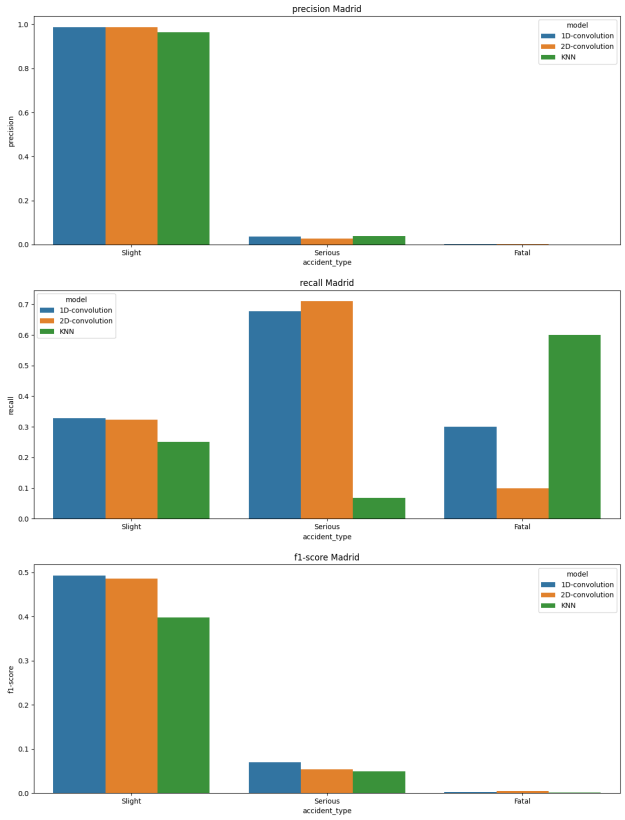


Figura 5.7: Comparativa de las métricas predicciones de los modelos.

alo	precision	recall	f1score	support
Slight	0.9881019830028328	0.32859161563824774	0.49317780134323075	10615.0
Serious	0.03694921952183363	0.677536231884058	0.07007682218474799	276.0
Fatal	0.0012987012987012987	0.3	0.002586206896551724	10.0
accuracy	0.33740023851022843	0.33740023851022843	0.33740023851022843	0.337400238510
macro avg	0.3421166346077893	0.4353759491741019	0.18861361014151015	10901.0
weighted avg	0.9631147161889811	0.3374002385102284	0.48201535879739016	10901.0

Tabla 5.5: Métricas CNN-1D.

	precision	recall	f1-score	support
Slight	0.9881019830028328	0.32859161563824774	0.49317780134323075	10615.0
Serious	0.03694921952183363	0.677536231884058	0.07007682218474799	276.0
Fatal	0.0012987012987012987	0.3	0.002586206896551724	10.0
accuracy	0.33740023851022843	0.33740023851022843	0.33740023851022843	0.3374002385102
macro avg	0.3421166346077893	0.4353759491741019	0.18861361014151015	10901.0
weighted avg	0.9631147161889811	0.3374002385102284	0.48201535879739016	10901.0

Tabla 5.6: Métricas Naive Bayes.

	precision	recall	f1-score	support
Slight	0.9881019830028328	0.32859161563824774	0.49317780134323075	10615.0
Serious	0.03694921952183363	0.677536231884058	0.07007682218474799	276.0
Fatal	0.0012987012987012987	0.3	0.002586206896551724	10.0
accuracy	0.33740023851022843	0.33740023851022843	0.33740023851022843	0.3374002385102
macro avg	0.3421166346077893	0.4353759491741019	0.18861361014151015	10901.0
weighted avg	0.9631147161889811	0.3374002385102284	0.48201535879739016	10901.0

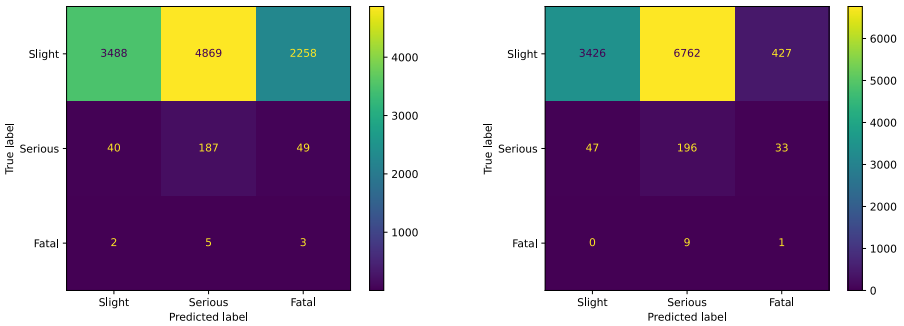
Tabla 5.7: Métricas SVC.

	precision	recall	f1-score	support
Slight	0.9881019830028328	0.32859161563824774	0.49317780134323075	10615.0
Serious	0.03694921952183363	0.677536231884058	0.07007682218474799	276.0
Fatal	0.0012987012987012987	0.3	0.002586206896551724	10.0
accuracy	0.33740023851022843	0.33740023851022843	0.33740023851022843	0.3374002385102
macro avg	0.3421166346077893	0.4353759491741019	0.18861361014151015	10901.0
weighted avg	0.9631147161889811	0.3374002385102284	0.48201535879739016	10901.0

Tabla 5.8: Métricas CNN-2D.

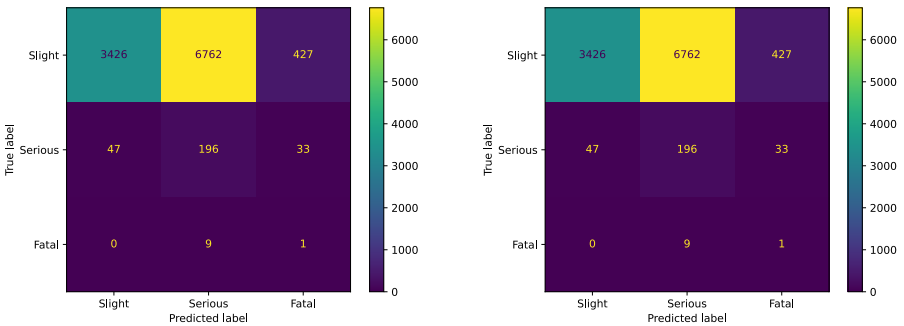
5.4.2 Matrices de confusión

Explicar las matrices de confusión con datos de **TEST.**



(a) Matriz de confusión CNN-1D.

(b) Matriz de confusión CNN-2D.



(c) Matriz de confusión NB.

(d) Matriz de confusión SVC.

Figura 5.8: Matrices de confusión de los modelos sobre el conjunto de entrenamiento.

5.4.3 Comparativa entre modelos

Explicar la comparativa de modelos con datos de **ENTRENAMIENTO.**

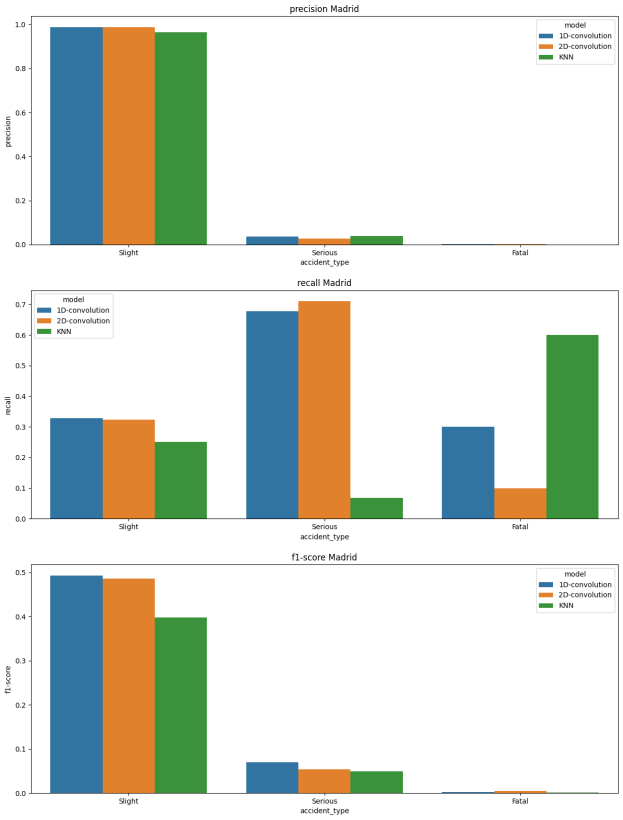


Figura 5.9: Comparativa de las métricas predicciones de los modelos.

6 Conclusiones

Bibliografía

Altman, N. S. (1992). An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, *46*(3), 175–185.

Anaconda, I. (s.f.). *Anaconda library.* Descargado de `https://www.anaconda.com/`

Brownlee, J. (s.f.). *Feature importance and feature selection with xgboost in python.* Descargado de `https://machinelearningmastery.com/feature-importance-and-feature-selection-with-xgboost-in-python/`

Chawla, N. V., Bowyer, K. W., Hall, L. O., y Kegelmeyer, W. P. (2002, jun). Smote: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, *16*, 321–357. Descargado de `https://doi.org/10.1613%2Fjair.953` doi: 10.1613/jair.953

Chen, T., y Guestrin, C. (2016, aug). XGBoost. En *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining.* ACM. Descargado de `https://doi.org/10.1145%2F2939672.2939785` doi: 10.1145/2939672.2939785

de Datos Abiertos del Ayuntamiento de Madrid, P. (s.f.-a). *Accidentes de tráfico madrid.* Descargado 2022-05-11, de `https://datos.madrid.es/portal/site/egob/menuitem.c05c1f754a33a9fbe4b2e4b284f1a5a0/?vgnextoid=7c2843010d9c3610VgnVCM2000001f4a900aRCRD&vgnextchannel=374512b9ace9f310VgnVCM100000171f5a0aRCRD&vgnextfmt=default`

de Datos Abiertos del Ayuntamiento de Madrid, P. (s.f.-b). *Estructura del conjunto de datos.* Descargado de `https://datos.madrid.es/FWProjects/egob/Catalogo/Seguridad/Ficheros/Estructura_DS_Accidentes_trafico_desde_2019.pdf`

dmlc. (s.f.). *Xgboost library.* Descargado de `https://xgboost.readthedocs.io/en/stable/`

Emmert-Streib, F., Yang, Z., Feng, H., Tripathi, S., y Dehmer, M. (2020). An introductory review of deep learning for prediction models with big data. *Frontiers in Artificial Intelligence*, *3*. Descargado de `https://www.frontiersin.org/article/10.3389/frai.2020.00004` doi: 10.3389/frai.2020.00004

Feurer, M., Eggensperger, K., Falkner, S., Lindauer, M., y Hutter, F. (2020). Auto-sklearn 2.0: Hands-free automl via meta-learning. *arXiv:2007.04074 [cs.LG]*.

Ganesh, P. (s.f.). *Types of convolution kernels : Simplified.* Descargado de `https://towardsdatascience.com/types-of-convolution-kernels-simplified-f040cb307c37`

Github. (s.f.). *Github svc.* Descargado de `https://github.com/`

Glorot, X., Bordes, A., y Bengio, Y. (2011). Deep sparse rectifier neural networks. En *Proceedings of the fourteenth international conference on artificial intelligence and statistics* (pp. 315–323).

Google. (s.f.). *Google meets application.* Descargado de `https://meet.google.com/`

Haixiang, G., Yijing, L., Shang, J., Mingyun, G., Yuanyue, H., y Bing, G. (2017). Learning from class-imbalanced data: Review of methods and applications. *Expert Systems with Applications*, *73*, 220-239. Descargado de `https://www.sciencedirect.com/science/article/pii/S0957417416307175` doi: https://doi.org/10.1016/j.eswa.2016.12.035

Hinton, G. E., y Roweis, S. (2002). Stochastic neighbor embedding. *Advances in neural information processing systems*, *15*.

Ioffe, S., y Szegedy, C. (2015). *Batch normalization: Accelerating deep network training by reducing internal covariate shift.* arXiv. Descargado de `https://arxiv.org/abs/1502.03167` doi: 10.48550/ARXIV.1502.03167

Jain, M. (s.f.). *Hyperparameter tuning in xgboost using genetic algorithm.* Descargado de `https://towardsdatascience.com/hyperparameter-tuning-in-xgboost-using-genetic-algorithm-17bd2e581b17`

jGraph. (s.f.). *Diagramsnet software.* Descargado de `https://www.diagrams.net/`

Jiang, Y., Tong, G., Yin, H., y Xiong, N. (2019). A pedestrian detection method based on genetic algorithm for optimize xgboost training parameters. *IEEE Access*, *7*, 118310-118321. doi: 10.1109/ACCESS.2019.2936454

Jupyter, P. (s.f.-a). *Jupyter lab.* Descargado de `https://jupyterlab.readthedocs.io/en/stable/`

Jupyter, P. (s.f.-b). *Jupyter notebook.* Descargado de `https://jupyter.org/`

Khandelwal, R. (s.f.). *Filters and feature maps.* Descargado de `https://towardsdatascience.com/convolutional-neural-network-feature-map-and-filter-visualization-f75012a5a49c`

Kiranyaz, S., Avci, O., Abdeljaber, O., Ince, T., Gabbouj, M., y Inman, D. J. (2019). *1d convolutional neural networks and applications: A survey.* arXiv. Descargado de `https://arxiv.org/abs/1905.03554` doi: 10.48550/ARXIV.1905.03554

Koech, K. E. (s.f.). *Cross-entropy loss function.* Descargado de `https://towardsdatascience.com/cross-entropy-loss-function-f38c4ec8643e`

Kopelias, P., Papadimitriou, F., Papandreou, K., y Prevedouros, P. D. (2007, 12). Urban freeway crash analysis: Geometric, operational, and weather effects on crash number and severity. *Transportation Research Record*, *2015*, 123-131. doi: 10.3141/2015-14

Kullback, S. (1997). *Information theory and statistics.* Courier Corporation.

Lingaraj, H. (2016, 10). A study on genetic algorithm and its applications. *International Journal of Computer Sciences and Engineering*, *4*, 139-143.

Liu, X.-Y., Wu, J., y Zhou, Z.-H. (2009). Exploratory undersampling for class-imbalance learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, *39*(2), 539-550. doi: 10.1109/TSMCB.2008.2007853

Nvidia. (s.f.-a). *Cuda library.* Descargado de `https://developer.nvidia.com/cuda-zone`

Nvidia. (s.f.-b). *Nvidia xgboost.* Descargado de `https://www.nvidia.com/en-us/glossary/data-science/xgboost/`

pandas dev. (s.f.). *Pandas library.* Descargado de `https://pandas.pydata.org/docs/getting_started/overview.html`

Projectpro.io. (s.f.). *Why data preparation is an important part of data science?* Descargado de `https://www.projectpro.io/article/why-data-preparation-is-an-important-part-of-data-science/242`

Python. (s.f.). *Python language.* Descargado de `https://www.python.org/about/quotes/`

School, D. (s.f.). *Comparing supervised learning algorithms.* Descargado de `https://www.dataschool.io/comparing-supervised-learning-algorithms/`

scikit learn. (s.f.). *Scikit-learn library.* Descargado de `https://scikit-learn.org/stable/`

Singh, D., y Singh, B. (2020). Investigating the impact of data normalization on classification performance. *Applied Soft Computing*, *97*, 105524. Descargado de `https://www.sciencedirect.com/science/article/pii/S1568494619302947` doi: https://doi.org/10.1016/j.asoc.2019.105524

Sklearn. (s.f.). *Gridsearchcv.* Descargado de `https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html`

Smith, L. N. (2018). *A disciplined approach to neural network hyper-parameters: Part 1 – learning rate, batch size, momentum, and weight decay.* arXiv. Descargado de `https://arxiv.org/abs/1803.09820` doi: 10.48550/ARXIV.1803.09820

Spark, C. (s.f.). *Hyperparameter tuning in xgboost.* Descargado de `https://blog.cambridgespark.com/hyperparameter-tuning-in-xgboost-4ff9100a3b2f`

Sun, Y., Wong, A. K., y Kamel, M. S. (2009). Classification of imbalanced data: A review. *International journal of pattern recognition and artificial intelligence*, *23*(04), 687–719.

tensorflow. (s.f.). *Tensorflow library.* Descargado de `https://www.tensorflow.org`

Van der Maaten, L., y Hinton, G. (2008). Visualizing data using t-sne. *Journal of machine learning research*, *9*(11).

Wikipedia. (s.f.). *Softmax function.* Descargado de `https://en.wikipedia.org/wiki/Softmax_function`

Zheng, M., Li, T., Zhu, R., Chen, J., Ma, Z., Tang, M., … Wang, Z. (2019). Traffic accident's severity prediction: A deep-learning approach-based cnn network. *IEEE Access*, *7*, 39897–39910.