Escuela
Politécnica
Superior

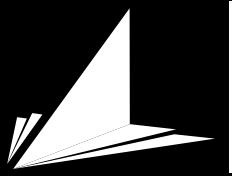
Título del Trabajo Fin de Grado/Máster

Máster Universitario en Ciencia de Datos

Trabajo Fin de Máster

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

Máster Universitario en Ciencia de Datos

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), su regresión precisa y el rendimiento de clasificación han atraído la atención de los investigadores, provocando 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) permite identificar las relaciones ocultas entre los factores de accidentes, el principal inconveniente de estos modelos es que pocos estudios 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 en el de 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. Debido a las características de los datos y para preparar la entrada al modelo de aprendizaje profundo, se ha utilizado una técnica para transformar números en matrices bidimensionales [1]. Esta técnica consiste en convertir un vector de características del accidente en una matriz de características, teniendo en cuenta que los vectores de características son las variables de los datos del accidente. Los resultados obtenidos se han comparado con otros resultados de modelos de aprendizaje automático con la finalidad de extender las conclusiones.

1.2 Objetivos

1.2.1 Objetivos Generales

El objetivo principal de este trabajo final de Máster es desarrollar un sistema predictivo de la gravedad 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

Para alcanzar este objetivo principal, se plantean los siguientes objetivos específicos:

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

Recientes investigaciones (MIT) proponen arquitecturas más complejas técnicas para crear mapas de riesgos de accidentes, tomando como input imágenes tomadas por satélite, segmentación de carreteras, información GPS de los vehículos y datos de accidentes con la finalidad de

que pueden ser monitorizados en tiempo real para ayudar a mitigarlos.

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.

Cabe resaltar que el código implementado en este proyecto es completamente escalable y reusable. Esto permite que para adaptar este proyecto a otro problema distinto únicamente sea necesario realizar la limpieza correspondiente del conjunto de datos específico y definir las jerarquías de categorías padre y características hijas correspondientes, el resto del proceso se realiza de forma autónoma sin necesidad de modificar el código. **NOTA: Realmente el lector aún no sabe nada de las jerarquías no?**

3.1 Modelos

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

3.1.1 Algoritmos Genéticos

Un *Algoritmo Genético* [2], 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.

Hay que remarcar que 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:

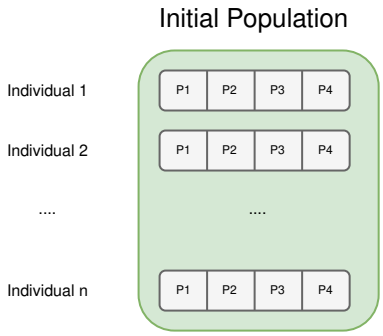


Figura 3.1: Inicialización de la población. Cada individuo está constituido por n parámetros del XGBoost a optimizar.

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. 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 poblÁción 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.

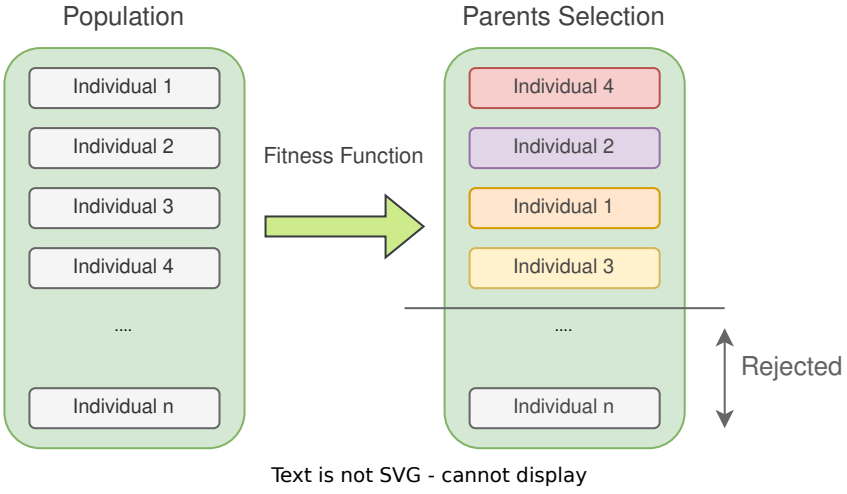


Figura 3.2: Selección de padres candidatos.

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].

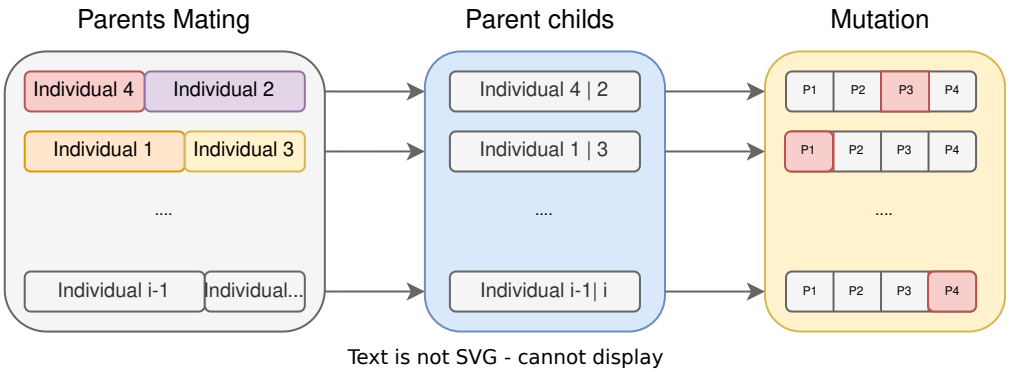


Figura 3.3: Mutación de hijos.

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 unos algoritmos muy utilizado en

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* [3].

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*) [4].

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.

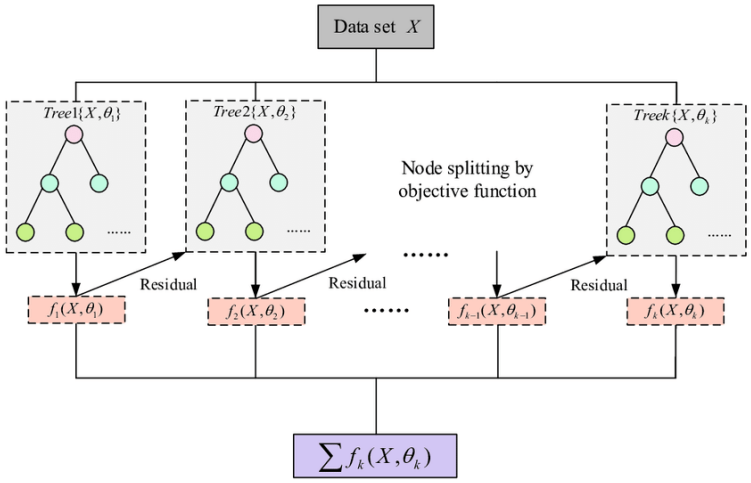


Figura 3.4: Proceso de optimización de XGBoost.

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 Gaussian Naive Bayes

El clasificador probabilístico Naive Bayes (NB) es un modelo que se basa en el *Teorema de Bayes*, este clasificador asume ciertas suposiciones de independencia sobre los predictores para realizar las clasificaciones. Se asume que la presencia o ausencia de cualquier predictor no está condicionada por la inclusión o exclusión de cualquier otro en el conjunto de datos, es decir, cualquier característica de las observaciones es independiente del resto. Asumiendo esta independencia entre los predictores la complejidad computacional del modelo se reduce considerablemente, haciendo de este modelo uno de los más eficientes en aprendizaje estadístico.

Para utilizar Naive Bayes con valores reales se asume la distribución Gaussiana de las características, a esta adaptación se le denomina como Gaussian Naive Bayes (GNB). Este enfoque es más simple que el anterior ya que únicamente es necesario calcular la media y la desviación típica de las variables de entrada para resumir dichas distribuciones, que se utilizarán para calcular la probabilidad de pertenencia a una clase en función de estas distribuciones normales gracias a la media y a la desviación típica de cada variable.

El primer paso será calcular la denominada probabilidad a priori, que es la probabilidad de que una nueva muestra pertenezca a alguna de las clases del conjunto de entrenamiento, este cálculo servirá para determinar la predicción final de las observaciones.

Posteriormente se calculará, para cada una de las variables de la muestra, la probabilidad de pertenecer a cada clase del conjunto de entrenamiento en función del grado de pertenencia a la distribución Gaussiana correspondiente a la variable. Después se aplicará esta multiplicación de probabilidades a la probabilidad a priori inicial definida a dicha clase.

En la siguiente fórmula se define la probabilidad de pertenencia a una clase c del conjunto Y en base a la característica X.

$$P(X|Y=c) = \frac{1}{\sqrt{2\pi\sigma_c^2}} e^{\frac{-(x-\mu_c)^2}{2\sigma_c^2}}$$

Donde σ es la varianza y μ es la media de la variable X perteneciente a la clase c del conjunto de clases posibles Y.

3.1.4 SVC

Los Support-Vector Classifier (SVC), también denominados como *Soft Margin Classifier*, son modelos orientados a clasificación múltiple [5]. Se basan en proyectar los datos de entrada en un espacio multidimensional buscando la separabilidad las estas muestras en este espacio determinando hiperplanos.

Los SVC construyen estos hiperplanos en base a la máxima separabilidad de las clases utilizando únicamente aquellas observaciones que se encuentren en la fronteras que maximizan la distancia en el plano entre las clases. La ventaja de esta técnica es que es necesario un conjunto muy pequeño de muestras de entrenamiento para sostener los hiperplanos que

forman las fronteras de decisión, estos puntos se denominan *vectores de soporte*.

La distancia entre los vectores de soporte y el hiperplano definido es denominado *margen*, y es en este punto donde entra en juego la flexibilidad de los SVC. Estos modelos aplican el denominado *margen blando*, que se basa en permitir que ciertas observaciones se encuentren en un lado erróneo del margen o incluso del hiperplano, haciendo que el modelo sea más robusto permitiendo mayor generalización a costa de clasificar erróneamente alguna de las observaciones mediante un parámetro de tolerancia C. A un mayor valor de C se permite que un mayor número de muestras violen el margen [6].

En la figura [3.5] se muestra un ejemplo aplicado de SVC, donde se observan los hiperplanos definidos que maximiza la separabilidad de ambas clases y los vectores de soporte definidos en función de distintos valores de tolerancia C.

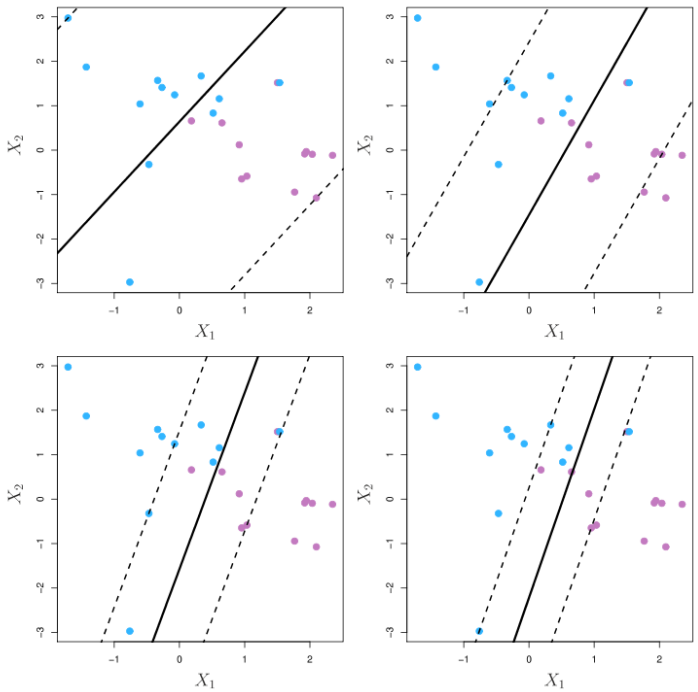


Figura 3.5: ISLR2. Ejemplo de SVC con distintos valores de C.

3.1.5 Redes Neuronales

Las Redes Neuronales [7] 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 3.6. 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*.

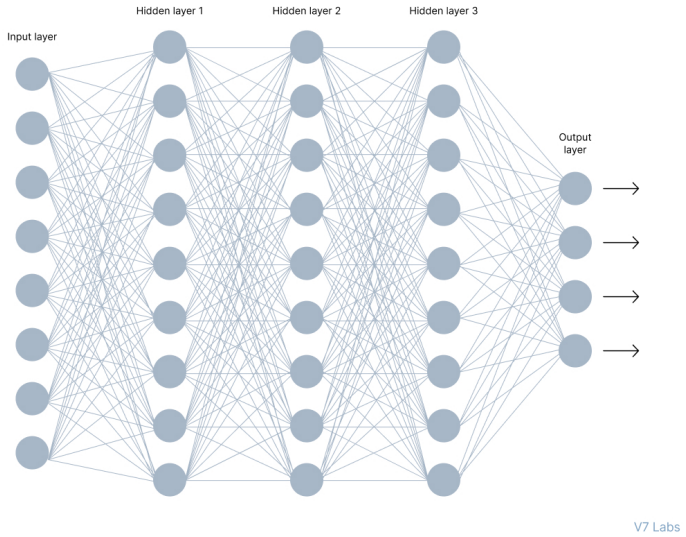


Figura 3.6: https://www.v7labs.com/blog/neural-networks-activation-functions. Arquitectura de una Red Neuronal.

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 [8].

  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 [9].

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

  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.

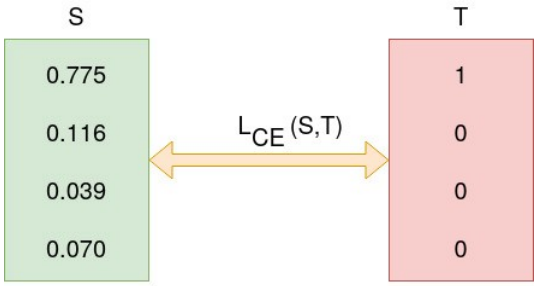


Figura 3.7: https://towardsdatascience.com/cross-entropy-loss-function-f38c4ec8643e Cross-Entropy entre la función Softmax (izquierda) y el valor de la clase verdadera (derecha).

- 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*.

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* [10], 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* [11] 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.

3.1.5.1 Convolucionales

A contiuación definiremos los parámetros específicos aplicados a las Convolutional Neural Network 2D (CNN-2D):

- 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* [12].

- 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 [13]

- 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.

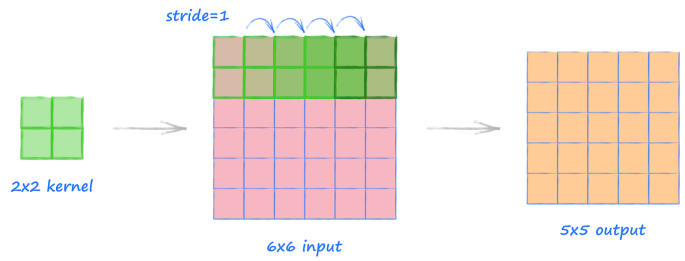


Figura 3.8: http://makeyourownneuralnetwork.blogspot.com/2020/02/calculating-output-size-of-convolutions.html

- 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 Íos 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.

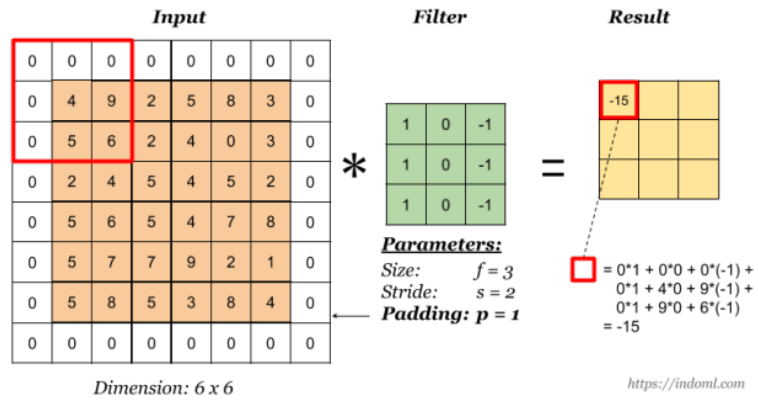


Figura 3.9: https://indoml.com/2018/03/07/student-notes-convolutional-neural-networks-cnn-introduction/

- 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 3.10 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*.

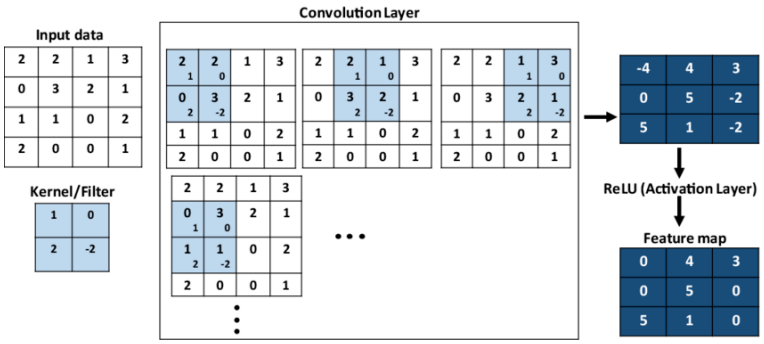


Figura 3.10: https://www.researchgate.net/journal/Transport-in-Porous-Media-1573-1634. Ejemplo de ReLU aplicado a una convolución de dos dimensiones.

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.

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 características en ellas.

Las complejidad computacional de las *CNN-1D* puede ser mayor que la de las *CNN-2D* debido a que necsitan desplazarse con un kernel generalmente más pequeño por la entrada de la red, 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 estas arquitectura las hacen atractivas para aplicarlas en tiempo real en dispositivos con pocos recursos como teléfonos móviles [14].

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 [13] o de la imagen de entrada si se trata de la primera capa de la red.

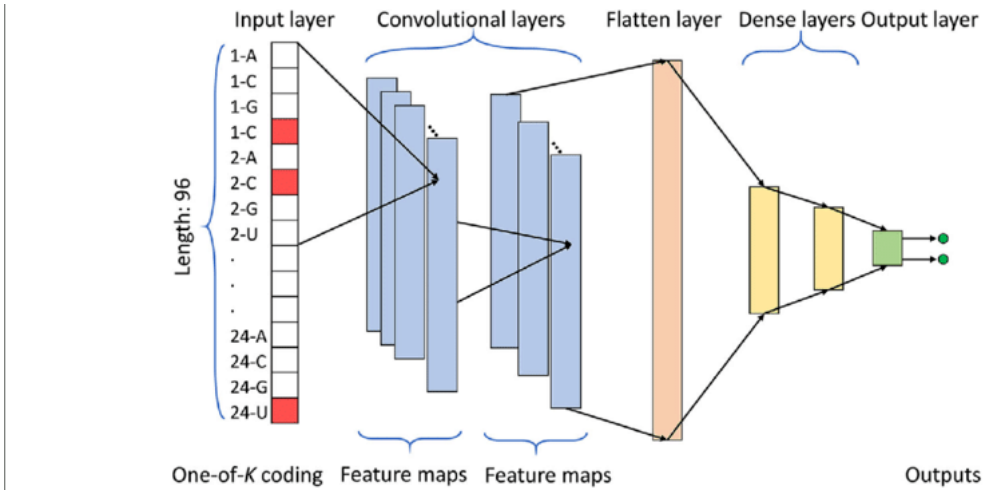


Figura 3.11: https://www.researchgate.net/publication/329201018/figure/fig2/AS:697400008138753@1543284527161/T architecture-of-our-1D-CNN-model-This-model-consists-of-two-convolutional-layers.png

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

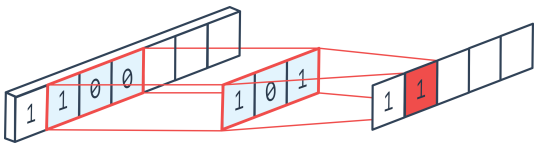


Figura 3.12: https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/blocks/1d-convolution Observamos la entrada de la red (izquierda) a la que se le aplica un kernel (en el centro) que resulta en la convolución de un mapa de características (derecha).

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.

3.1.6 KNN

El algoritmo *K vecinos más próximos*, o K-Nearest Neighbors (KNN) [15], 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 3.13 se muestra un ejemplo de un problema de clasificacíón 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.

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. [16]
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 [17].

Tensorflow: libería utilizada para la implementación de redes neuronales permitiendo su ejecución. Este proyecto se basa en la versión 2.8.0 [18].

Sklearn: librería que contiene múltiples modelos predictivos implementados, basado en NumPy, SciPy y Matplotlib. La versión configurada para este proyecto es 1.0.2 [19].

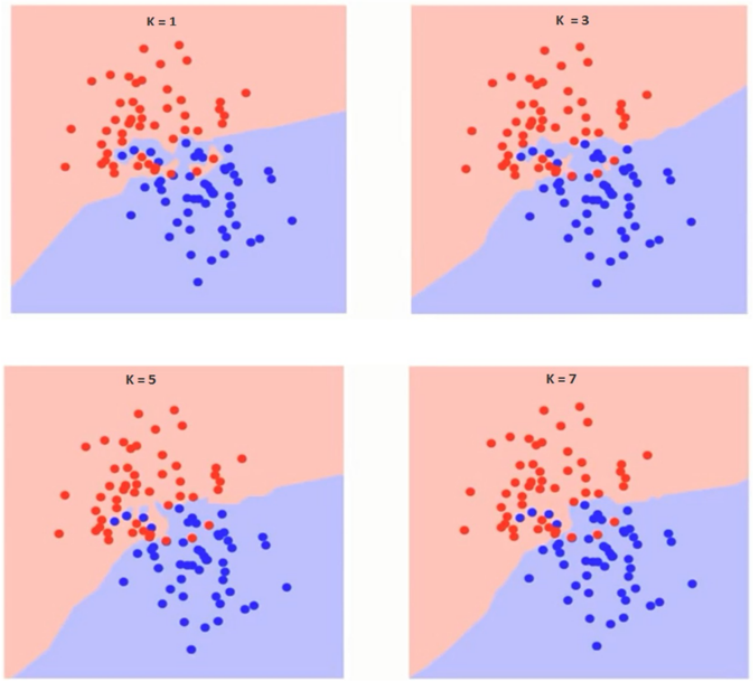


Figura 3.13: https://forum.huawei.com/enterprise/es/data/attachment/forum/202108/20/150033de3l4nyvmcbozh14.pr
KNN aplicado con distintos valores de k

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 [20].

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 [21].

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 [22].

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 [23].

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 [24].

DiagramsNet: plataforma utilizada para la confección de figuras mostradas en este documento [25].

Google Meets: plataforma utilizada para realizar reuniones semanales con el tutor [26].

Como sistema de control de versiones se ha utilizado [27], repositorio donde es posible subir versiones evolutivas del proyecto imprescincible en los proyectos de desarrollo *Software*.

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*:

Este diagrama proporciona una visión de las fases de un proyecto y su estimación temporal. En el caso que nos ocupa, el diagrama se muestra en al figura 4.1.

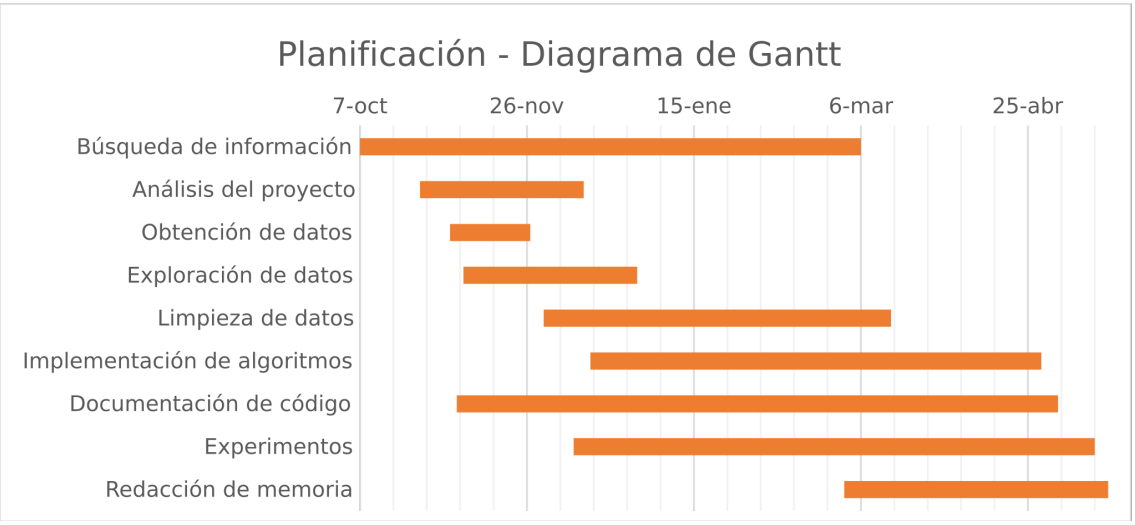


Figura 4.1: Diagrama de Gantt de la planificación del proyecto.

La búsqueda de información incluye la investigación de todo lo referente al proyecto, y por este motivo, esta fase, es la primera en el desarrollo siendo un proceso que se ha prolongado durante casi toda la evolución del trabajo.

El análisis del proyecto consiste en estudiar la viabilidad del mismo, buscando fuentes de información desde las que poder obtener los datos además de explorarlos. Es una de las fases más críticas en un proyecto *Data Science* ya que si las conclusiones del estudio de viabilidad no son sólidas, se podría cometer el riesgo de empezar un proyecto en el que no es factible cubrir los objetivos.

La obtención de datos consiste en la búsqueda de repositorios donde encontrar datos que contengan la información requerida para abarcar los objetivos del proyecto. Se ha realizado visitando distintos portales de datos abiertos nacionales hasta encontrar un conjunto de datos que encajase con los objetivos, en esta fase también se involucra la exploración inicial de los datos.

La exploración de datos abarca la fase de análisis de datos, esto se realiza utilizando métodos visuales para entender las características propias del dataset, además de hacer una valoración

de la calidad de los mismos. Las primeras etapas de esta fase se complementan con el análisis del proyecto.

El siguiente paso ha sido la limpieza de datos, gracias a la exploración se comprenden la naturaleza de los datos y se decide qué tecnicas usar para el tratamiento de valores atípicos del conjunto de datos para su posterior transfomación.

Una de las tareas más importantes en el ciclo de vida de desarrollo Software que garantizan la calidad del producto es la documentación de código. Tiene como fin clarificar conceptos y explicar el funcionamiento de los métodos que componen el desarrollo. Es una fase que se prolonga desde el inicio de la codificación en la limpieza de datos hasta poco tiempo después de terminar de implementar los modelos.

La última fase de este proyecto a nivel funcional de este proyecto ha sido la realización de experimentos, comienza poco antes que la implementación de los modelos y consiste en testar las distintas implementaciones, tanto de limpieza de datos como el funcionamiento de los modelos.

Finalmente hay que redactar la memoria, plasmando toda la información obtenida durante la evolución del proyecto.

4.1 Fases

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.

Un aspecto muy importante en un TFM es el diagrama de flujo de la metodología. En este diagrama se define el flujo de ejecución del proyecto y, como se puede observar en la figura 4.2, 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.

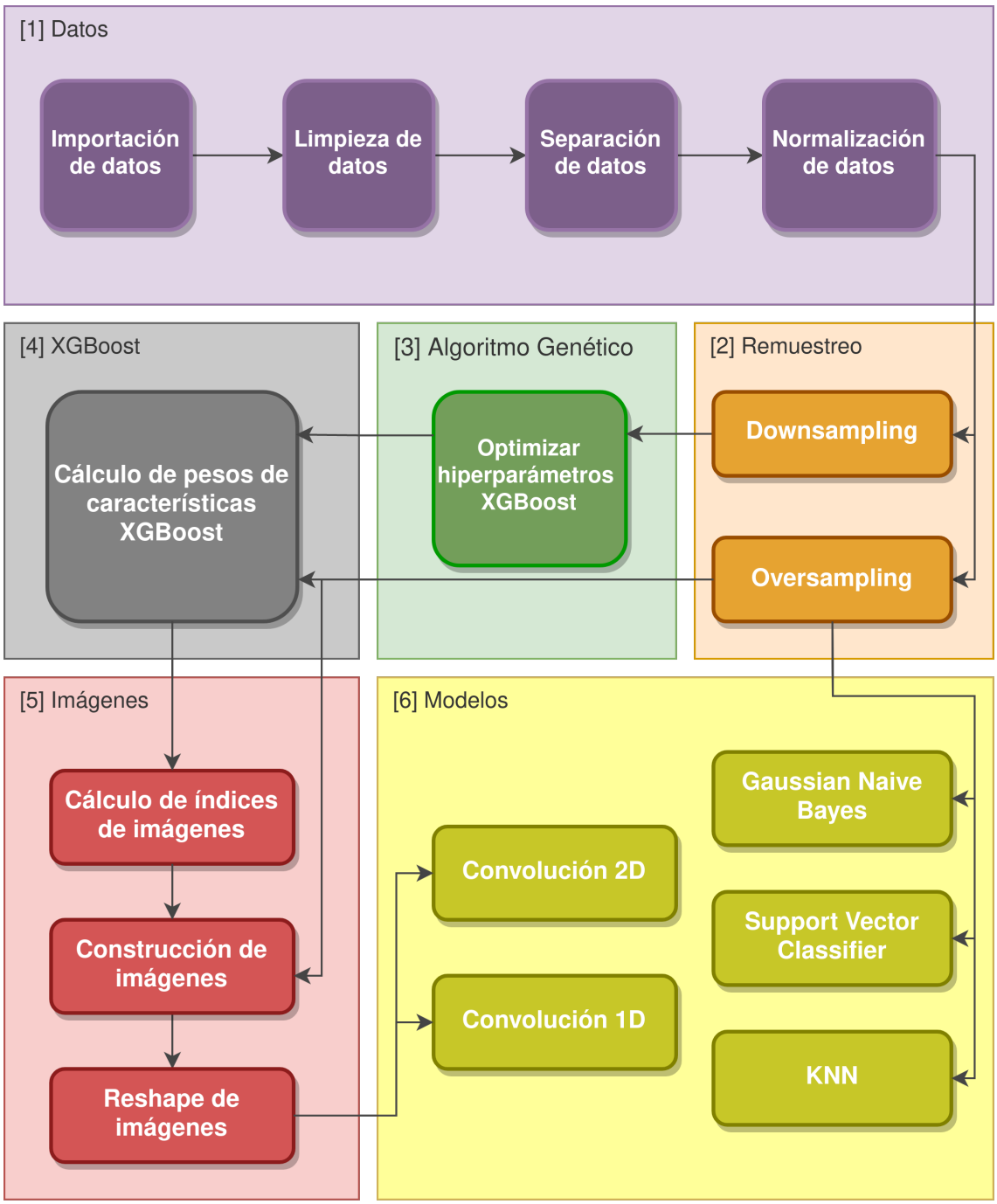


Figura 4.2: Flujo de datos del proceso del proyecto.

4.1.1 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 [28], 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 la web [29], 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 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.

Hay que resaltar la importancia del atributo lesividad, puesto que es la variable respuesta de este trabajo. Existen una serie de valores asignados a esta categoría, así cada cdato de lesividad 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. El tipado numérico de estas casuísticas es el siguiente:
  - 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.
- Consecuencias severas: implicados que han requerido un ingreso hospitalario superior a 24 horas. En este caso el tipado numérico del conjunto de datos corresponde a:
  - 3: Ingreso superior a 24 horas.
- Consecuencias fatales: víctimas mortales dentro del margen de las 24 horas posteriores al accidente. La asignación numérica a este campo es:
  - 4: Fallecido 24 horas.

Se pueden consultar más detalles de las características del dataset descripción del conjunto de datos del portal *Open Data* de Madrid [30].

1. **Limpieza de datos**

   Una vez importados los datos, es indisepensable 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 la aplicación de distintas fases de análisis.

   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 las variables (también denominados predictores) con los que trabajarán los modelos, se han eliminado aquellos registros duplicados y aquellos que tuvieran algún valor. 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 60.966 filas originales.

2. **Transformaciones de datos**

   A continuación se requiere la realización de transformaciones sobre los datos para que los modelos utilizados trabajen con un conjunto de datos bien definido y consistente. Gran parte de los modelos de ML necesitan de datos de entrada numéricos para poder comprenderlos, además existe la necesidad de normalizar estos valores para que estén en la misma escala y el entrenamiento sea exitoso, por lo tanto, lo primero es la realización

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, ya que estas variables eran inicialmente de tipo *String*. Además, se da el caso de que el rango de estas variables es del orden de 7 a 10 dígitos, evitando cualquier formato de decimales estandarizado (utilizando puntos y comas indistintamente en distintas posiciones de los mismos), 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.

La variable *localización* ha sido transformada a *tipo de carretera*, con el objetivo de definir un criterio para clasificar el tipo de vía donde se ha producido el incidente, como por ejemplo una carretera, una autovía, glorieta, etc. Al no existir un servicio que ofrezca la tipología de la calle dado el nombre de la vía o las coordenadas, ha sido necesario tratar esta problema desde otro punto de vista mediante expresiones regulares. El término *expresión regular* debe su origen a la teoría de las matemáticas y ciencias de computación, reflejan la propiedad de *regularidad* que caracterizan a las expresiones matemáticas [31]. Una expresión regular es un tipo de patrón utilizado para encontrar una combinación de caracteres en una cadena de texto. Para clasificar el tipo de carreteras en este proyecto se han diseñado una serie de expresiones regulares que coinciden con ciertos criterios respecto a los valores de la variable *localizacion* con el fin de distinguir distintos tipos de vía.

Las tipologías de clases de carreteras son *Parking, Aeropuerto, Cuesta, Paseo, Parque, Túnel, Polígono, Camino, Glorieta, Puerta, Puente, Plaza, Bulevar, Travesía, Calzada, Carretera, Avenida, Autovía* y *Calle*.

No obstante, la aplicación de las expresiones regulares generalmente no son una solución empírica a un problema, ya que hay casos que pueden ser no contemplarse al diseñar estos patrones manualmente. En el caso de este proyecto su aplicación conlleva una serie de inconvenientes. Aśí, debido a las necesidades del problema, estas expresiones regulares están diseñadas jerárquicamente, de forma que aquellos valores de la variable *localizacion* que coincidan con el primer patrón definido de la expresión regular serán asignados a esta categoría. Esto supone un impedimento a tener en cuenta, ya que muchos de los accidentes se producen en intersecciones, carriles de aceleración que conectan dos vías, vías cuyo nombre contiene el tipo de otra vía, etc. Por ejemplo si se da el caso de que exista algún patrón no definido como expresión regular, esto provocará que estas localizaciones se asignen con respecto al primer patrón definido encontrado, llegando a clasificar erróneamente el tipo de vía. Como consecuencia, la jerarquía de la definición de las expresiones regulares es un punto crítico ya que se busca maximizar la clasificación de aquellos tipos de vía con menor número de apariciones, como es el caso de parking o aeropuerto, mientras se minimiza el error en la clasificación.

Para solucionar esto, existen herramientas como Open Refine [32] que utilizan técnicas de clustering nominal para detectar semejanzas entre caracteres. Debido a las limitacio-

nes de tiempo, esta técnica no se ha podido aplicar, aunque se propone como trabajos para futuro con el objetivo de aumentar el rendimiento de los modelos aplicados.

Por otro lado hay que tratar aquellos valores que no hayan coincidido con los patrones definidos, sin embargo, al ser una cantidad de registros se han podido revisar manualmente coincidiendo todas con nombres de calles al que no se les ha asignado la palabra *"calle"*.

En la variable *hora*, al ser un campo continuo, ha sido necesario discretizarla según intervalos, distinguiendo entre noche y día en función de la hora en la que se ha producido el incidente.

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.

La transformación numérica de los datos es un proceso crítico, ya esta codificación será la entrada de los modelos utilizados. Una representación que no refleje correctamente el tipo de cada campo original influirá negativamente en la precisión de los modelos implementados.

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.

   Si se analizan los datos que correspondan a cada una de las tres posibles clases de lesividad (leves, severos y fatales) realizando un histograma 4.3, se puede observar que el dataset se encuentra 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 [33]. 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 clasificación 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 solventarlo mediante distintas técnicas de

muestreo [34], por lo que será necesario remuestrear estos datos mediante la técnica Borderline Synthetic Minority Over-sampling Technique 2 (SMOTE-II).

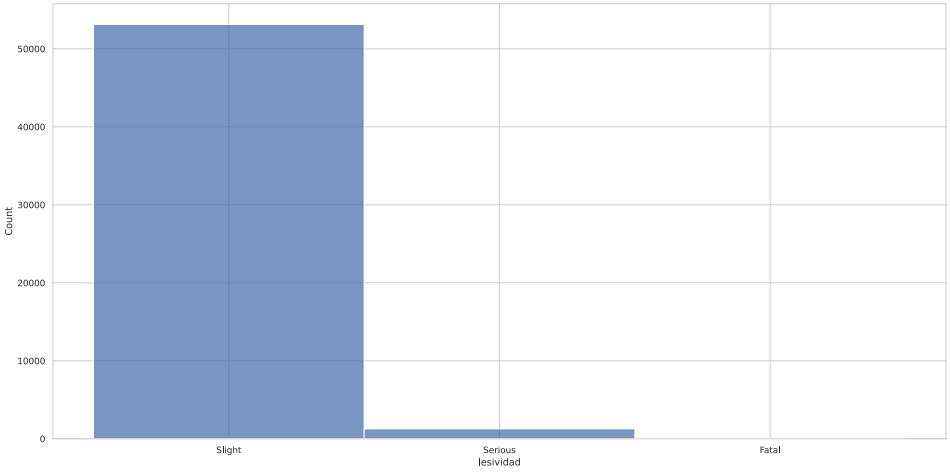


Figura 4.3: Histograma de clases de accidente en el conjunto de datos original.

Es interesante observar el grado de correlación entre las variables de forma que si se encuentran campos muy correlacionados se podría eliminar alguno de ellos. Para realizar este análisis se utiliza el coeficiente de correlación de *Pearson* [35], que toma un rango de valores entre -1 y 1. Como resultado la correlación será más pronunciada cuanto más se aproxime a -1 o a 1 mientras que será más débil al aproximarse a 0.

En la figura 4.4 podemos observar la matriz de correlación de las variables explicativas del conjunto de datos del proyecto. Analizando dicha matriz se puede comprobar que las únicas variables que están medianamente correlacionadas son *coordenada x utm* y *distrito*, con una correlación positiva de 0.44, sin embargo no es un valor tan alto como para eliminar alguna de las dos las características. Así pues, el estudio de la correlación muestra que las variables de datos elegidas no están correlacionadas y se puede continuar con la siguiente fase del proyecto.

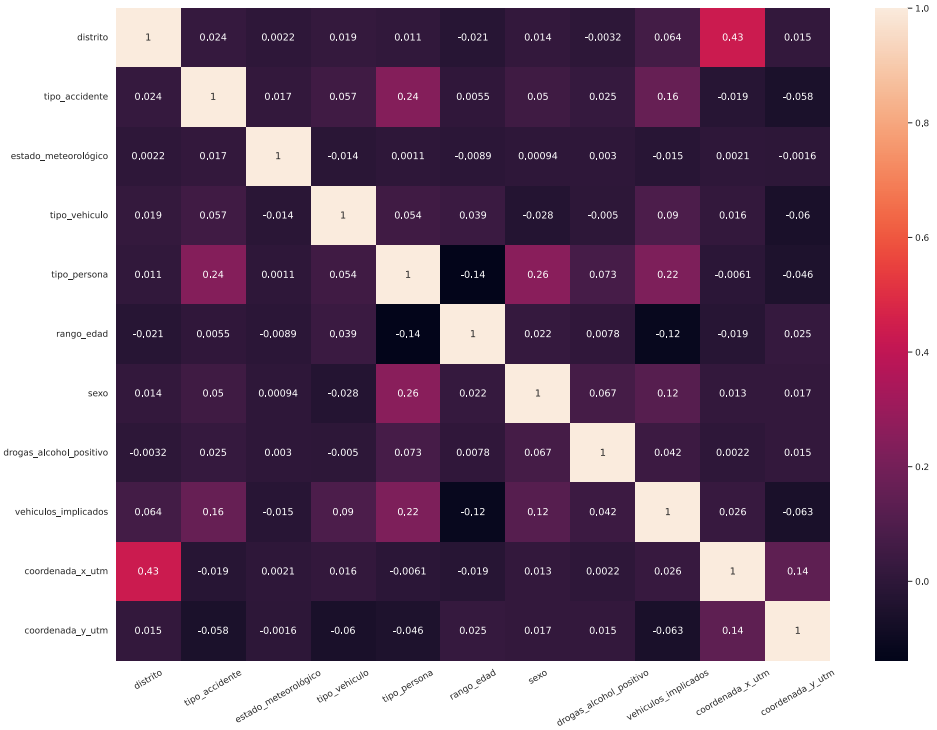


Figura 4.4: Matriz de correlación para los predictores escogidos del dataset.

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, existen características representadas en distintas escalas, y las que contienen un rango de valores numérico más alto, ya sea por la naturaleza de la variable o porque se encuentren en otro rango, dominan a las que se encuentren en un rango menor, influenciando negativamente en los modelos ML, como por ejemplo las CNN o KNN [36].

   Así pues, el proceso de normalización tiene como objetivo minimizar el sesgo 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 [37].

   En este proyecto se ha utilizado la normalización Z-Score Normalization (ZSN) debido a que consigue representaciones de acuerdo a una distribución normal. Para ello, se 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.

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

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.

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

5. Separación de datos

   La siguiente fase en cualquier modelo de ML es la separación de datos (*split*). Esta fase consiste en dividir el conjunto total de registros en al menos dos subconjuntos, uno de entrenamiento y otro de test. El modelo se entrenará en base al conjunto de entrenamiento y se evaluará con el 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, y en nuestro caso hemos elegido un valor de *80* por ciento para el entrenamiento y *20* por ciento restante para el conjunto de test, por lo que tendremos un total de *43.603* muestras para el conjunto de entrenamiento y *10.901* para el de test.

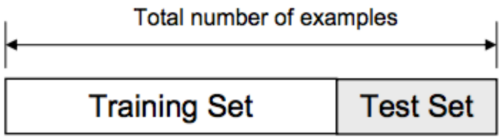


Figura 4.5: https://towardsdatascience.com/train-test-split-and-cross-validation-in-python-80b61beca4b6. División de un conjunto de datos en datos de entrenamiento y test.

4.1.2 Remuestreo

Con el objetivo de que el modelo no se vea muy afectado debido a la diferencia de las clases mayoritarias respecto a las minoritarias se hace necesaria la utilización de técnicas de remuestreo. Estas técnicas permiten operar sobre un conjunto de datos para balancearlos, y se aplican únicamente al conjunto de entrenamiento, ya que la intención es influir sobre el aprendizaje del modelo para aumentar el rendimiento de las predicciones sobre el conjunto de test, si aplicásemos remuestreo sobre este último conjunto estaríamos falseando los resultados de las predicciones.

En este proyecto se ha hecho uso de la generación de datos sintéticos como método de remuestreo. Estos procedimientos permiten generar datos artificiales en base a los límites que separan unas clases de otras. Para esto se ha hecho uso de la teécnica Borderline Synthetic Minority Over-sampling Technique 2 (SMOTE-II) [38]. Este método 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*). Este nuevo conjunto de datos servirá como entrenamiento para los modelos, evitando que tiendan al sobreajuste. Una vez aplicado SMOTE-II se obtienen 42508 muestras de cada una de las clases de accidentes.

Una vez aplicada la técnica de remuestreo, es útil observar cuál es la representación espacial de los datos sintéticos generados por SMOTE-II en deferencia a los datos originales mediante el método t-Distributed Stochastic Neighbor Embedding (T-SNE) [39]. T-SNE es un algoritmo que permite visualizar las proyecciones de datos multidimensionales en espacios bidimensionales o tridimensionales. Este modelo debe su origen a su antecesor Stochastic Neighbor Embedding (SNE), siendo una versión optmizada de éste.

Mientras que SNE proyecta los datos convirtiendo las distancias Euclídeas entre las muestras a probabilidades condicionales, el T-SNE modifica la función de coste para proyectarlos mediante una distribución t-Student, permitiendo además trabajar con gradientes simplificados.

En la figura 4.6 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.

El T-SNE de 2D aplicado sobre los datos generados por SMOTE-II 4.6b nos permite observar cómo se han generado las muestras con respecto a los datos originales 4.6a en una proyección bidimensional. Se aprecia cómo se han producido las nuevas muestras de los accidentes graves (azul) calculando la cercanía de los puntos en base a las fronteras de división.

4.1.3 Algoritmo Genético

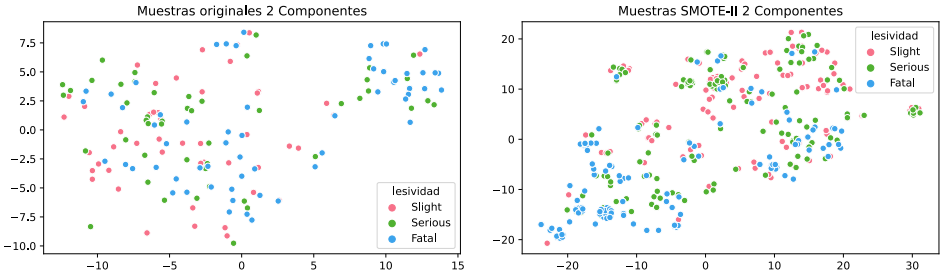
Para entrenar el modelo haremos uso del algoritmo XGBoost. Y para que este entrenamiento sea adecuado es importante optimizar los hiperparámetros del mismo.

Para ello se ha hecho uso de algoritmos genéticos, 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 [40].

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 ellos, se ha seleccionado un subcojunto de aquellos que más influencia tienen en el entrenamiento del modelo. Estos hiperparámetros de los que consta cada individuo de la población son:

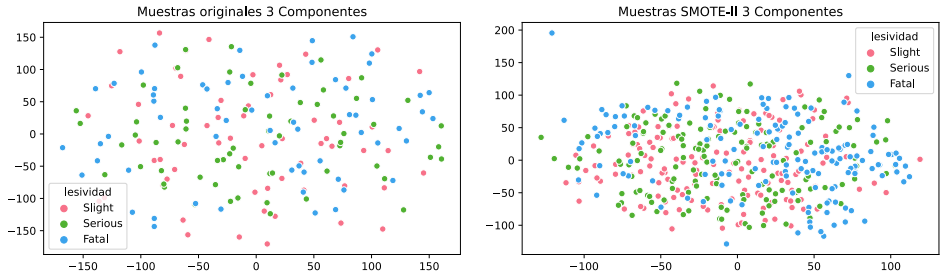
1. Profundidad máxima: es la máxima altura que puede tomar el árbol. Si el árbol de decisión alcanza demasiada profundidad tenderá al *overfitting* ya que aprenderá relaciones complejas entre los datos que pueden deberse a ruido en los datos de entrenamiento.

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



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

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. Si esta restricción no se contemplase, los hiperparámetros podrían tomar valores extremos, reduciendo así el rendimiento de entrenamiento y el de las predicciones. Por lo tanto los individuos variarán sus parámetros tomando un valor aleatorio dentro de rangos definidos. Los parámetros de las nuevas soluciones mutarán dentro de unos límites específicos para cada parámetro establecidos como máximos y mínimos.

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 *50* individuos especificados en la población, éstos se evaluarán instanciando los *50* modelos XGBoost correspondientes. El conjunto de datos sobre el que entrenará cada instancia XGBoost será el conjunto de entrenamiento remuestreado por SMOTE-II. La función *fitness* que evaluará cada individuo será la métrica Micro F1-score aplicada sobre el conjunto de test obtenido en el proceso de separación de datos, de tal forma que en cada una de las *50* generaciones definidas se comprobará si existe algún individuo de la población en ese momento que logre superar el Micro F1-score del mejor individuo hasta el momento.

Una vez evaluados los individuos de una generación se ha especificado que los *15* mejores de ellos sean los que se crucen para dar luegar a nuevos hijos, que tendrán una probabilidad de mutación de *0.4* para cada una de sus *3* características. En el momento de que existan individuos repetidos en la población, estos serán eliminados para dar lugar a nuevas inicializaciones aleatorias favoreciendo así la diversidad de la población. Los parámetros del mejor individuo al cabo de las *50* generaciones serán con los que se entrenará el algoritmo XGBoost.

4.1.4 XGBoost

Una vez se han optimizados los hiperparámetros del XGBoost, se entrena un nuevo modelo con los pesos calculados sobre el conjunto de entrenamiento remuestreado. Con este nuevo modelo XGBoost entrenado se obtiene la importancia de cada clase mediante la ponderación de sus pesos [41]. 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, por lo

tanto, 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 y saber qué predictores son más importantes.

4.1.5 Matrices

Las CNN aprenden patrones sobre la entrada de datos como matrices, esto implica que estén construidas de tal forma que se maximice la representación de la información, es decir, es necesario aplicar técnicas que posicionen cada característica en un elemento de la matriz maximizando la información de los accidentes.

Por lo tanto, una vez se tienen los datos normalizados y muestreados, el siguiente paso será transferir cada una de las muestras tabulares de los accidentes a una matriz (*5 x 5*), donde a cada característica se le asignará un elemento de la matriz, de tal forma que éstas sean la entrada a las *CNN*.

Para este objetivo se va a aplicar el algoritmo *FV2GI* propuesto en el artículo [1], 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 [42] 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*. Basándonos en esto se realizará una asignación de cada característica del dataset en una de estas categorías. La tabla 4.4 muestra 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. Tipo de accidente.
Carretera	Distrito. Tipo Carretera.
Ambiente	Estado meteorológico.
Vehículo	Tipo de vehículo.
Conductor	Tipo Persona. Sexo. Rango Edad. Positivo.

Tabla 4.4: 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 matrices de acuerdo a los siguientes pasos:

1. Generación de n matrices 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 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 construir las matrices:

1. La importancia de un padre viene dada por Íla suma de los pesos de las características hijas, en la tabla 5.2 se observa el cálculo de cada característica para las categorías principales.

2. La asignación de las filas de la matriz a las características 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 las categorías padre a las filas se realiza el mismo procedimiento con las características hijas a nivel de columna. Estas características se asignarán en una posición dentro de la fila de su categoría padre, donde aquella que más importancia tenga irá posicionada en el centro, la segunda irá posicionada a su izquierda, la tercera a la derecha y así sucesivamente.

La finalidad de posicionar las características más influyentes en las zonas centrales de la matriz se debe a que las CNN utilizan los kernels para recorrer la imagen en base a un desplazamiento, por lo tanto, estos kernels convolucionarán más veces estas posiciones donde se encuentran las caracerísticas más influyentes respecto a si se sitúan en los extremos de la matriz.

Un ejemplo de la construcción de estas matrices se ve reflejado en la figura 4.7.

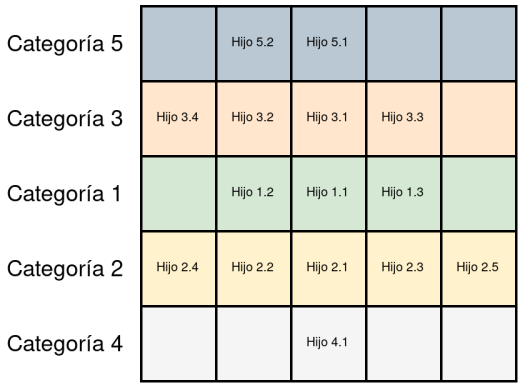


Figura 4.7: Ejemplo de posicionamiento de los elementos en una matriz. A las características se les asignan las filas en función de su peso y a las categorías hijas una columna dentro de la categoría correspondiente en función de su peso.

Una vez obtenidos los índices de las matrices se posicionará cada característica de los datos en la posición asignada dentro cada matriz. Cabe recordar que estas características están normalizadas bajo ZSN, por lo que la matriz contendrá los valores de cada característica normalizados. Como resultado de este proceso se obtienen las matrices que serán la entrada de las CNN.

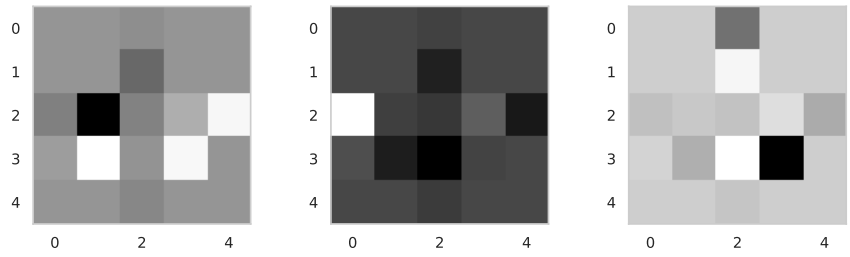


Figura 4.8: Representación de las muestras de accidentes en forma de matriz.

4.1.6 Modelos

En esta sección analizaremos los modelos utilizados en este proyecto para la realización de un estudio comparativo entre las arquitecturas. En primer lugar definiremos aquellos modelos que trabajan con observaciones tabulares normalizadas, es decir, aquellos que tienen como entrada un conjunto de datos de tipo real, como son GNB, KNN y SVC. Posteriormente detallaremos las implementaciones de las CNN, que son las que trabajarán con los datos en forma de matrices mediante la transformación definida anteriormente.

4.1.6.1 Gaussian Naive Bayes

El modelo GNB asumirá que cada una de las 12 variables pertenecientes al dataset son independientes. En la instanciación de este modelo no especificaremos ninguna probabilidad a priori, por lo que el molelo las calculará en función del número de muestras pertenecientes a cada clase en función del resto.

4.1.6.2 SVC

Para implementar el modelo SVC ha sido necesario fijar el valor de regularización C, que representa la tolerancia a que ciertas observaciones puedan sobrepasar los márgenes del hiperplano separador con la finalidad de generalizar el modelo, este parámetro se ha establecido a 1. Por otro lado es necesario especificar el tipo de kernel con el que entrenará el modelo, en nuestro caso se ha utilizado un kernel radial, que medirá la distancia entre los puntos en función de la similaridad que presenten.

Las distancias que mide el kernel radial se definen en la siguiente fórmula:

$$K(X_1, X_2) = exp(-\frac{||X_1 - X_2||^2}{2\sigma^2})$$

Donde X_1 y X_2 son los puntos que se medirán en base al kernel K, $||X_1 - X_2||$ es la distancia euclídea (norma L_2) y σ es la varianza.

SVC Ha sido entrenado en base a los datos de entrenamiento tabulares remuestrados por SMOTE-II.

4.1.6.3 KNN

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

Es necesario optimizar los parámetros de este algoritmo para conseguir un buen rendimiento, por lo que aplicaremos la técnica *Grid Search* [43]. 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.

KNN construirá el espacio de proyecciones mediante los datos de entrenamineto remuestreados por XGBoost, y clasificará las muestras del conjunto de test en base a la cercanía de los vecinos proyectados del conjunto de entrenamiento.

4.1.6.4 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 aplicarán añadiendo ceros en los límites de las matrices, 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 4.9. 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 [44].

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

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 CNN propuestas se muestra el caso de la CNN-2D en la figura 4.10

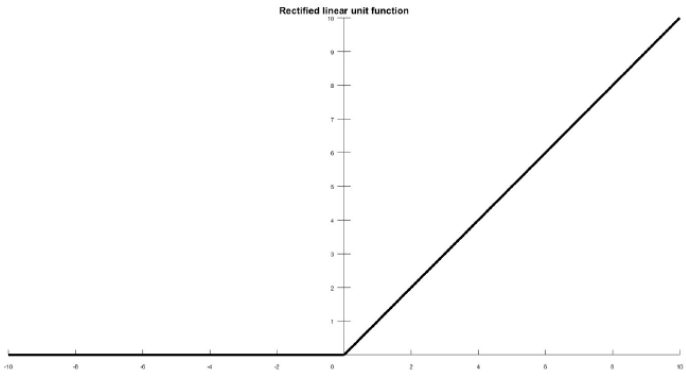


Figura 4.9: https://www.researchgate.net/journal/Transport-in-Porous-Media-1573-1634. Función ReLU

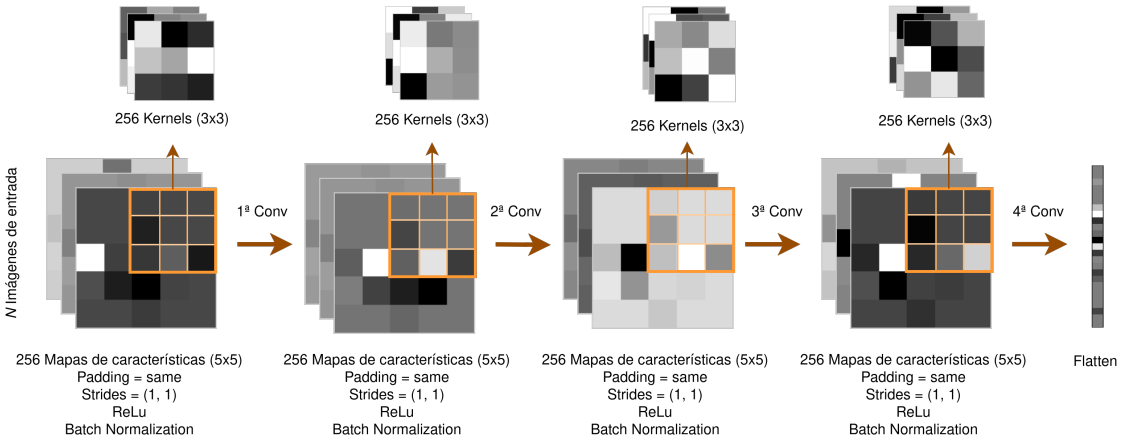


Figura 4.10: Arquitectura de la CNN-2D mostrando una kernels aprendidos durante el entrenamiento.

4.2 Métricas

A continuación definiremos las métricas comúnmente utilizadas para evaluar la calidad de las clasificaciones de los modelos.

4.2.1 Reporte de clasificación

1. Precisión: se trata el porcentaje de acierto al clasificar una clase. En el caso de este proyecto representará el porcentaje de accidentes correctos de una clase con respecto al total de accidentes predichos como dicha clase. Por ejemplo, de todos los accidentes identificados como leves, el porcentaje de los que eran leves realmente. Esta métrica se calcula dividiendo el total de verdaderos positivos respecto a la suma de verdaderos positvos y los falsos positivos.

$$Precision = \frac{TP}{TP+FP}$$

2. Exhaustividad: se trata el porcentaje de identificaciones totales del modelo para una clase. Para este proyecto la exhaustividad representa el porcentaje identificaciones correctas para una clase de accidente. Por ejemplo, del total de accidentes leves, qué porcentaje de identificaciones correctas han sido predecidas por el modelo. El cálculo de esta métrica se basa en la división de verdaderos positivos con respecto a la suma de verdaderos positivos más los falsos negativos.

$$Exhaustividad = \frac{TP}{TP+FN}$$

3. F1-Score: es una métrica que resume la precisión y exhaustividad en un solo valor para mostrar cómo de bien se realiza la clasificación de verdaderos positivos con respecto al coste de falsos negativos que esto conlleva.

$$F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Exhaustividad}}{\text{Precisión} + \text{Exhaustividad}}$$

Estas métricas se aplican para cada una de las clases a predecir, por lo que existirán tantos valores como clases se encuentren en el problema.

4.2.2 Métrica de optimización

Debido al desbalanceo de datos sobre el conjunto de test es necesario aplicar una función de pérdida que optimice el modelo sin tener en cuenta dicho desbalanceo. Las funciones de pérdida más comunes como la precisión total del modelo están orientadas a minimizar el error de clasificación del número total de clases, por lo que en este proyecto se usará la métrica Micro F1-score, que mide el rendimiento del modelo en función del total de clases en el conjunto de datos. En las siguientes ecuaciones mostraremos cómo se calcula esta métrica en base a la micro precisión y la micro exhaustividad:

$$\text{Micro Precision} = \frac{\sum_{n=1}^{clases} TP_i}{\sum_{n=1}^{clases} TP_i + \sum_{n=1}^{clases} FP_i}$$

$$\text{Micro Exhaustividad} = \frac{\sum_{n=1}^{clases} TP_i}{\sum_{n=1}^{clases} TP_i + \sum_{n=1}^{clases} FN_i}$$

$$\text{Micro F1-score} = 2 \cdot \frac{\text{Micro Precision} \cdot \text{Micro Exhaustividad}}{\text{Micro Precisión} + \text{Micro Exhaustividad}}$$

4.2.3 Matriz de confusión

Una matriz de confusión es una técnica que permite evaluar la eficacia de un modelo respecto a la predicción de cada una de las clases objetivo. En el eje de las X se muestran las etiquetas predecidas por el modelo y en el de las Y las etiquetas de las clases verdaderas. De esta forma la matriz de confusión permite visualizar cómo clasifica el modelo cada una de las clases y observar a qué etiquetas asigna cada muestra sobre cualquier conjunto de datos.

En la figura 4.11 se puede observar el esquema que sigue una matriz de confusión en base a dos posibles valores a predecir (verdadero y negativo). Las muestras clasificadas erróneamente se considerarán falsos positivos o falsos negativos en función de cómo el modelo clasifica las observaciones.

		Predicción	
		Positivos	**Negativos**
Observación	**Positivos**	Verdaderos Positivos (VP)	Falsos Negativos (FN)
	Negativos	Falsos Positivos (FP)	Verdaderos Negativos (VN)

Figura 4.11: https://rpubs.com/chzelada/275494. Ejemplo de matriz de confusión.

5 Resultados

En esta sección se detallarán los resultados de cada una de las etapas del proyecto y los experimentos finales para realizar un análisis comparativo entre ellos.

5.1 Algoritmo Genético

En primer lugar analizaremos la evolución de los hiperparámetros del XGBoost gracias al algoritmo genético. En la figura 5.1 se pueden visualizar la evolución de los hiperparámetros del mejor individuo en cada generación a través del transcurso de interaciones. Se puede observar cómo el mejor individuo de cada generación va variando los tres hiperparámetros probando distintos valores hasta converger aproximadamente en la iteración *21*.

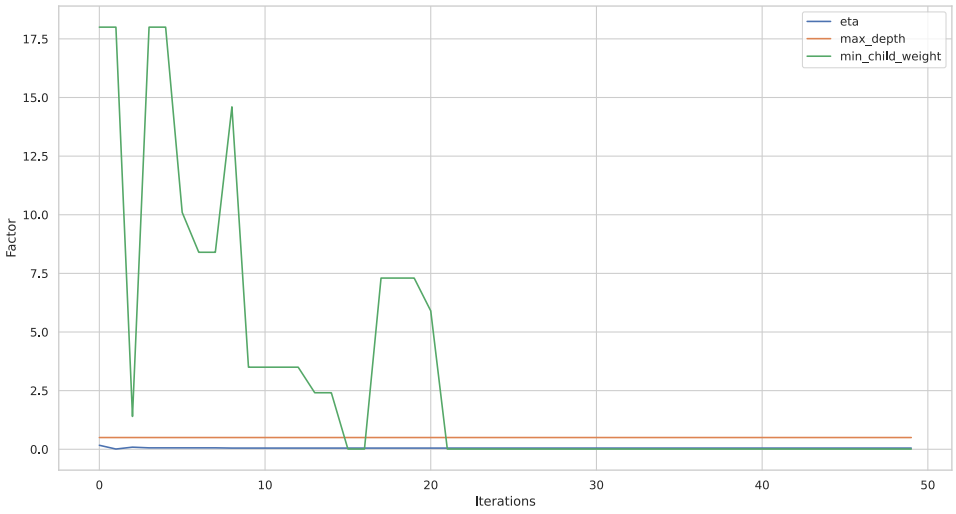


Figura 5.1: Evolución de hiperparámetros a lo largo de las iteraciones.

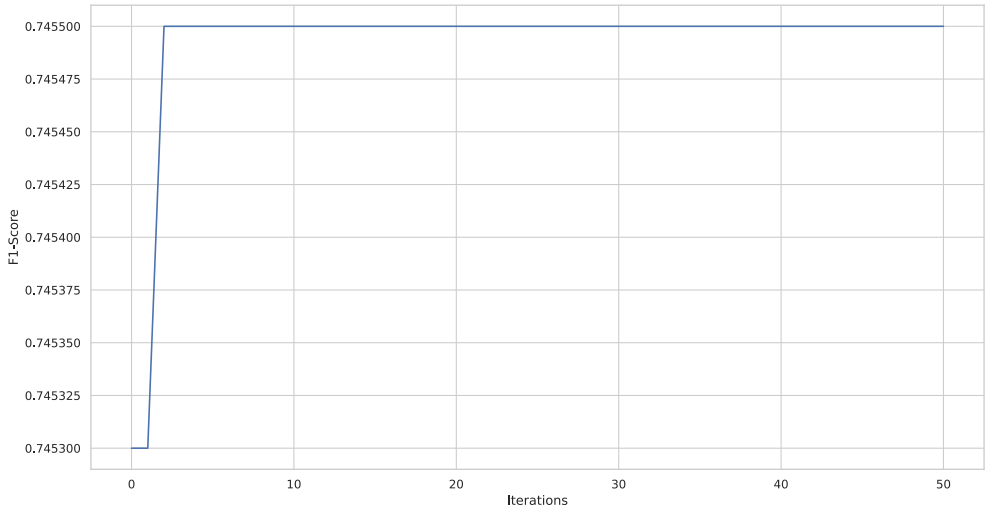


Figura 5.2: Evolución del macro F1 score a lo largo de las iteraciones.

En la tabla 5.1 se puede observar el mejor individuo obtenido después de *50* generaciones.

Hiperparámetro	**Valor**
Profundidad Máxima	1
Peso mínimo de los hijos	0.01
ETA	0.049

Tabla 5.1: Mejores parámetros de XGBoost tras aplicar el algoritmo genético.

5.2 XGBoost

A continuación se mostrará la importancia de las clases calculada por XGBoost mediante los hiperparámetros obtenidos en la ejecución del algoritmo genético. En la figura [**?**] se puede observar un diagrama de barras en el que las características *tipo de persona*, *tipo de carretera* y *sexo* son las que más han influido a la hora de entrenar el modelo XGBoost, obteniendo un peso de *0.177*, *0.127* y *0.111* respectivamente. Se pueden consultar los pesos calculados de todas las características en la tabla 5.2, en la que los pesos de las categorías padres es la suma de cada una de las características hijas.

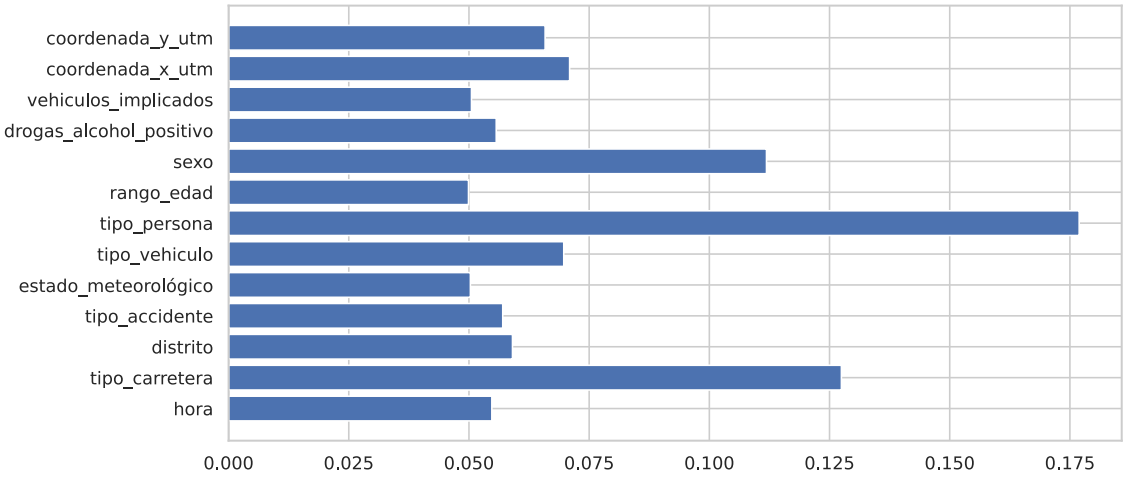


Figura 5.3: Pesos asignados por XGboost a las características.

Categoría	Peso Categoría	Característica	Peso Característica
Accidente	0.299	Coordenada X	0.071
		Coordenada Y	0.066
		Hora	0.055
		Vehículos implicados	0.051
		Tipo de accidente	0.057
Carretera	0.187	Distrito	0.059
		Tipo carretera	0.127
Ambiente	0.050	Estado meteorológico	0.050
Vehículo	0.070	Tipo de vehículo	0.070
Conductor	0.394	Tipo Persona	0.177
		Sexo	0.111
		Rango Edad	0.050
		Positivo	0.056

Tabla 5.2: Cálculo de pesos de características y categorías mediante XGBoost (los valores se han redondeado en base a tres decimales).

Característica	Valor
hora	2
tipo carretera	19
distrito	14
tipo accidente	1
estado meteorológico	1
tipo vehiculo	4
tipo persona	1
rango edad	3
sexo	1
drogas alcohol positivo	2
vehiculos implicados	1
coordenada x utm	438950266
coordenada y utm	4473953232

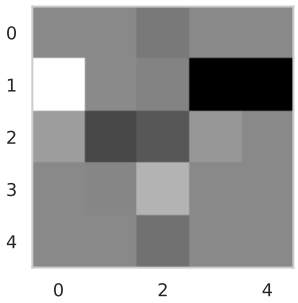
(a) Muestra de accidente tipificada.

Característica	Valor
hora	1.2548
tipo carretera	0.4597
distrito	-0.0297
tipo accidente	-1.4528
estado meteorológico	-0.2508
tipo vehiculo	-0.1621
tipo persona	-0.5316
rango edad	0.2129
sexo	-0.7004
drogas alcohol positivo	0.1488
vehiculos implicados	-1.4591
coordenada x utm	-0.0524
coordenada y utm	0.0081

(b) Muestra de accidente normalizada.

	0	1	2	3	4
0	0.0	0.0	-0.1621	0.0	0.0
1	1.2548	0.0081	-0.0524	-1.4528	-1.4591
2	0.2129	-0.7004	-0.5316	0.1488	0.0
3	0.0	-0.0297	0.4597	0.0	0.0
4	0.0	0.0	-0.2508	0.0	0.0

(c) Muestra transformada en matriz de valores.



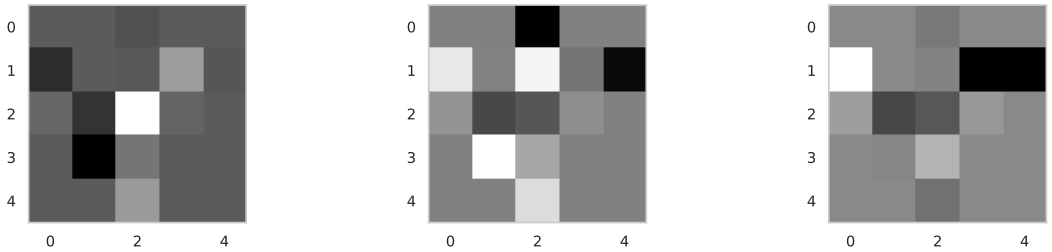
(d) Muestra transformada en matriz de grises.

Figura 5.4: Proceso que sigue una muestra del conjunto de datos tipificada hasta llegar a una matriz.

5.3 Matrices

En esta sección se mostrará el proceso que sigue una observación del dataset original tipificada hasta llegar a una matriz de características 5.4. En primer lugar, los valores de la muestra original 5.4a se normalizan según el criterio ZSN para pasar a ser una observación cuyos valores estan acotados en un rango en función de la distribución normal de cada característica en el dataset 5.4b. El siguiente paso será posicionar cada uno de los valores de la muestra en la posición correspondiente de la matriz de acuerdo a los índices calculados en el paso anterior, de tal forma que servirán como entradas a las redes neuronales convolucionales 5.4c.

A modo de ejemplo se muestra en la figura 5.5 un accidente de cada clase transformados a su matriz correspondiente.



(a) Accidente leve. **(b)** Accidente severo. **(c)** Accidente fatal.

Figura 5.5: Tres clases de accidentes convertidos a matrices.

5.4 KNN

Se puede observar en la tabla 5.3 los mejores parámetros para *KNN* después de haber ejecutado *Grid Search* sobre el conjunto de entrenamiento.

Atributo	Valor
Número de vecinos	91
Tipo de distancia	Minkowski

Tabla 5.3: Mejores parámetros de KNN tras aplicar GridSearch.

El resultado de aplicar ambas técnicas da lugar a la representación en forma matricial de los accidentes, como se puede observar en la figura 4.8.

5.5 Entrenamiento de modelos

A continuación se detallará una comparativa desde distintos puntos de vista de los resultados de los modelos. A nivel de rendimiento computacional se mostrarán los tiempos de entrenamiento. En lo que respecta a los resultados de predicción se analizarán las métricas de clasificación y matrices de confusión para los conjuntos de dato de entrenamiento y test. Para las redes neuronales además se mostrará la gráfica de evolución de la función de pérdida en base al Micro-F1Score para los datos de entrenamiento y test.

5.5.1 Tiempos de entrenamiento

En la figura 5.6 se muestran los tiempos de entrenamiento empleados por los modelos. Hay que recalcar que el método KNN, al proyectar las muestras de entrenamiento en un espacio n-dimensional, no conlleva un entrenamiento como tal. Observando la gráfica se puede apreciar que el modelo que más tiempo de entrenamiento emplea es el SVC con 593 segundos, seguido de las red convolucionales de una y dos dimensiones con 287 y 259 segundos respectivamente

y por último el clasificador GNB, que debido a su naturaleza es el que menos tiempo emplea en entrenar con 0.02 segundos.

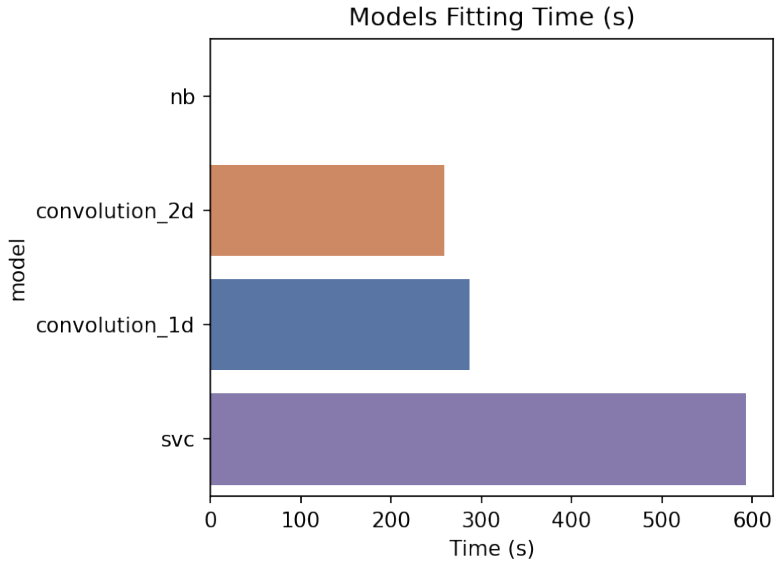


Figura 5.6: Comparación entre los tiempos de entrenamiento de los modelos.

5.5.2 Gráficas de entrenamiento

En las figuras 5.7 y 5.8 podemos observar la evolución de la métrica definida a minimizar (Micro F1-score) en función de las 100 épocas que se ejecutarán para las redes convolucionales.

Visualizando la gráfica de la convolucional de una dimensión 5.7 se puede comprobar que el Micro F1-score de entrenamiento va aumentando ligeramente a lo largo de las épocas sufriendo altibajos a medida que el modelo es entrenado, partiendo inicialmente desde **TODO: valor incial del Micro F1-score** hasta **TODO: valor final del Micro F1-score**. Observamos que los datos de validación **TODO: Describir en función de si entrenamos con los datos de validación smote-ii o con los de test**.

En la siguiente figura 5.8 se puede apreciar que la tendencia del Micro F1-score **TODO: es más pronunciada....** con respecto a la convolución de una dimensión. Esto es debido a que .....

5.5.3 Reportes de clasificación

A continuación se mostrarán los reportes de clasificación resultantes de predecir los datos de entrenamiento con los que han aprendido los modelos. Este proceso es útil para entender qué han aprendido los modelos tras su ejecución, con la finalidad de observar cómo se clasifican cada una de las clases para cada uno. Como se puede observar, el modelo que peor resultados nos da visualizando el F1 score **TODO: el modelo**

	precision	recall	f1-score	support
Slight	0.701	0.724	0.712	42509.0
Serious	0.696	0.523	0.597	42508.0
Fatal	0.754	0.917	0.828	42509.0
accuracy	0.721	0.721	0.721	0.721
macro avg	0.717	0.721	0.712	127526.0
weighted avg	0.717	0.721	0.712	127526.0

(a) Métricas de clasificación del conjunto de entrenamiento CNN-1D.

	precision	recall	f1-score	support
Slight	0.634	0.178	0.278	42509.0
Serious	0.648	0.14	0.231	42508.0
Fatal	0.39	0.977	0.558	42509.0
accuracy	0.432	0.432	0.432	0.432
macro avg	0.558	0.432	0.356	127526.0
weighted avg	0.558	0.432	0.356	127526.0

(b) Métricas clasificación del conjunto de entrenamiento Naive Bayes.

	precision	recall	f1-score	support
Slight	0.865	0.778	0.819	42509.0
Serious	0.796	0.776	0.786	42508.0
Fatal	0.861	0.969	0.912	42509.0
accuracy	0.841	0.841	0.841	0.841
macro avg	0.841	0.841	0.839	127526.0
weighted avg	0.841	0.841	0.839	127526.0

(c) Métricas clasificación del conjunto de entrenamiento SVC.

	precision	recall	f1-score	support
Slight	0.583	0.865	0.697	42509.0
Serious	0.845	0.568	0.679	42508.0
Fatal	0.873	0.737	0.799	42509.0
accuracy	0.724	0.724	0.724	0.724
macro avg	0.767	0.724	0.725	127526.0
weighted avg	0.767	0.724	0.725	127526.0

(d) Métricas clasificación del conjunto de entrenamiento KNN.

	precision	recall	f1-score	support
Slight	0.488	0.974	0.65	42509.0
Serious	0.646	0.299	0.408	42508.0
Fatal	0.966	0.524	0.679	42509.0
accuracy	0.599	0.599	0.599	0.599
macro avg	0.7	0.599	0.579	127526.0
weighted avg	0.7	0.599	0.579	127526.0

(e) Métricas clasificación del conjunto de entrenamiento CNN-2D.

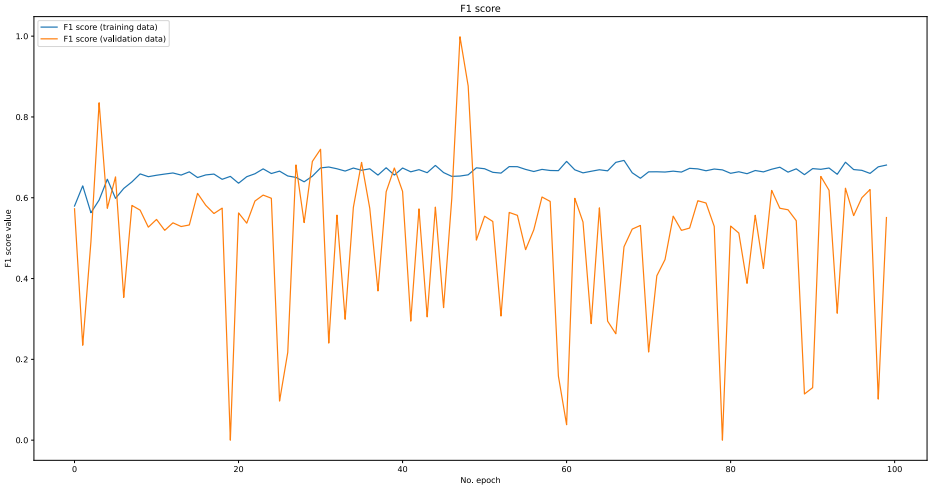


Figura 5.7: Evolución de Micro F1-score sobre el conjunto de entrenamiento y validación CNN-1D.

5.5.4 Matrices de confusión

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

5.5.5 Comparativa entre modelos

En la figura 5.10 se muestra gráficamente el rendimiento de cada modelo sobre el conjunto de entrenamiento, las métricas utilizadas son la precisión, sensitividad y el F1-score.

Como podemos observar, los modelos predicen bien los datos de con los que han sido entrenados, esta primera clasificación nos da una idea de cómo han evaluado los modelos las muestras. Sin embargo, unos resultados prometedores sobre este conjunto no implican un buen rendimiento, el objetivo es la predicción de datos futuros que puedan llegar al modelo, por lo tanto, comprobaremos la calidad mediante el conjunto de test.

5.6 Predicciones de modelos

5.6.1 Reportes de clasificación

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

5.6.2 Matrices de confusión

5.11

5.6.3 Comparativa entre modelos

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

	precision	recall	f1-score	support
Slight	0.984	0.492	0.656	10640.0
Serious	0.031	0.394	0.058	246.0
Fatal	0.002	0.333	0.004	15.0
accuracy	0.49	0.49	0.49	0.49
macro avg	0.339	0.407	0.239	10901.0
weighted avg	0.961	0.49	0.642	10901.0

(a) Métricas de clasificación de test CNN-1D.

	precision	recall	f1-score	support
Slight	0.98	0.369	0.536	10640.0
Serious	0.025	0.699	0.048	246.0
Fatal	0.0	0.0	0.0	15.0
accuracy	0.376	0.376	0.376	0.376
macro avg	0.335	0.356	0.195	10901.0
weighted avg	0.957	0.376	0.525	10901.0

(b) Métricas clasificación de test Naive Bayes.

	precision	recall	f1-score	support
Slight	0.979	0.644	0.777	10640.0
Serious	0.029	0.411	0.053	246.0
Fatal	0.0	0.0	0.0	15.0
accuracy	0.638	0.638	0.638	0.638
macro avg	0.336	0.352	0.277	10901.0
weighted avg	0.957	0.638	0.76	10901.0

(c) Métricas clasificación de test SVC.

	precision	recall	f1-score	support
Slight	0.982	0.689	0.81	10640.0
Serious	0.042	0.382	0.076	246.0
Fatal	0.001	0.067	0.002	15.0
accuracy	0.681	0.681	0.681	0.681
macro avg	0.342	0.379	0.296	10901.0
weighted avg	0.96	0.681	0.792	10901.0

(d) Métricas clasificación de test KNN.

	precision	recall	f1-score	support
Slight	0.982	0.919	0.949	10640.0
Serious	0.097	0.313	0.149	246.0
Fatal	0.0	0.0	0.0	15.0
accuracy	0.904	0.904	0.904	0.904
macro avg	0.36	0.411	0.366	10901.0
weighted avg	0.961	0.904	0.93	10901.0

(e) Métricas clasificación de test CNN-2D.

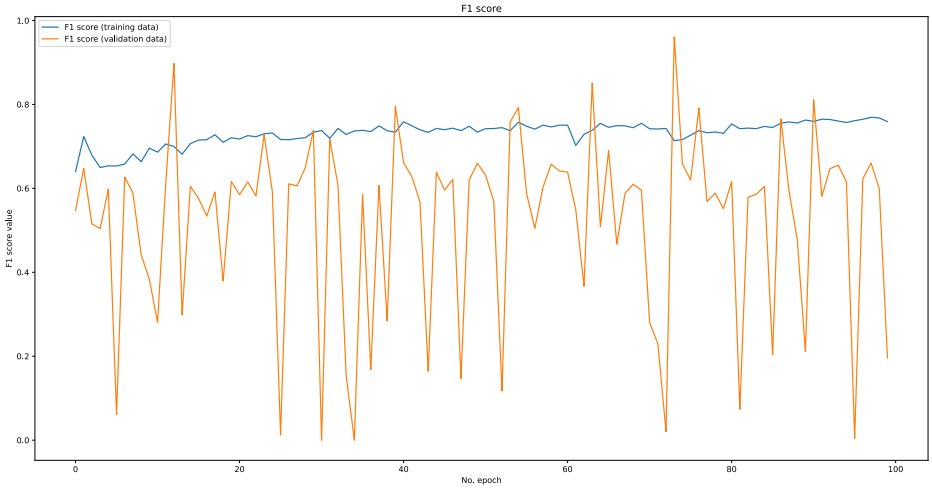


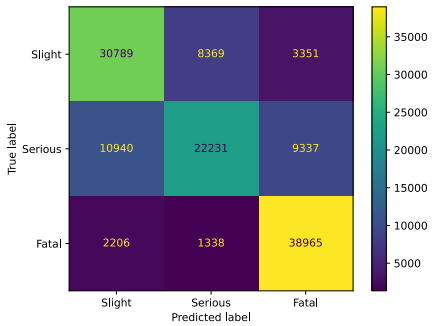
Figura 5.8: Evolución de Micro F1-score sobre el conjunto de entrenamiento y validación CNN-2D.

ninguno de los modelos aplicados es capaz de generalizar bien sore el conjunto de datos de test sin smote

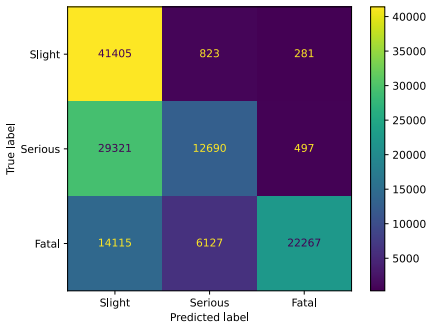
ya sea porque el modelo está sobreajustado o por

Es importante recalcar que los resultados de estas predicciones están muy condicionados al tratamiento y las transformaciones de los datos.

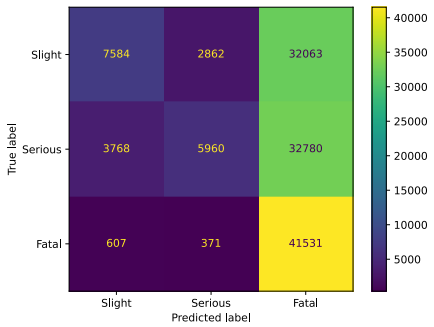
de la informacio que tenemos y de el tipo de limpieza que se ha hecho. Se ppodria hacer un estudio del valor numerico que se le asigna a cada variable en la limpieza, ya que seguramente al estar nosotros haciendolo aleatoriamente seguramente se esté creando algún patrón de jerarquía dentro de una columna o se le da mas importancia a un todoterreno que a un avion porque el todoterrento al tener un valor asignado más alto sera mas negro y por lo tanto la red predicira lo que sea



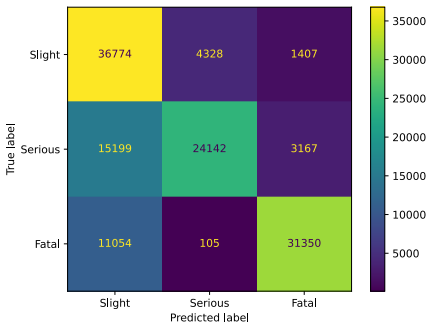
(a) Matriz de confusión de CNN-1D sobre entrenamiento.



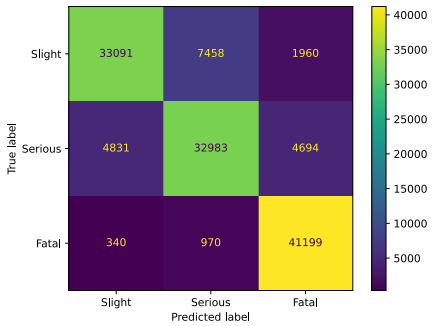
(b) Matriz de confusión de CNN-2D sobre entrenamiento.



(c) Matriz de confusión de NB sobre entrenamiento.



(d) Matriz de confusión de KNN sobre entrenamiento.



(e) Matriz de confusión de SVC sobre entrenamiento.

Figura 5.9: Matrices de confusión de los modelos sobre el conjunto de entrenamiento.

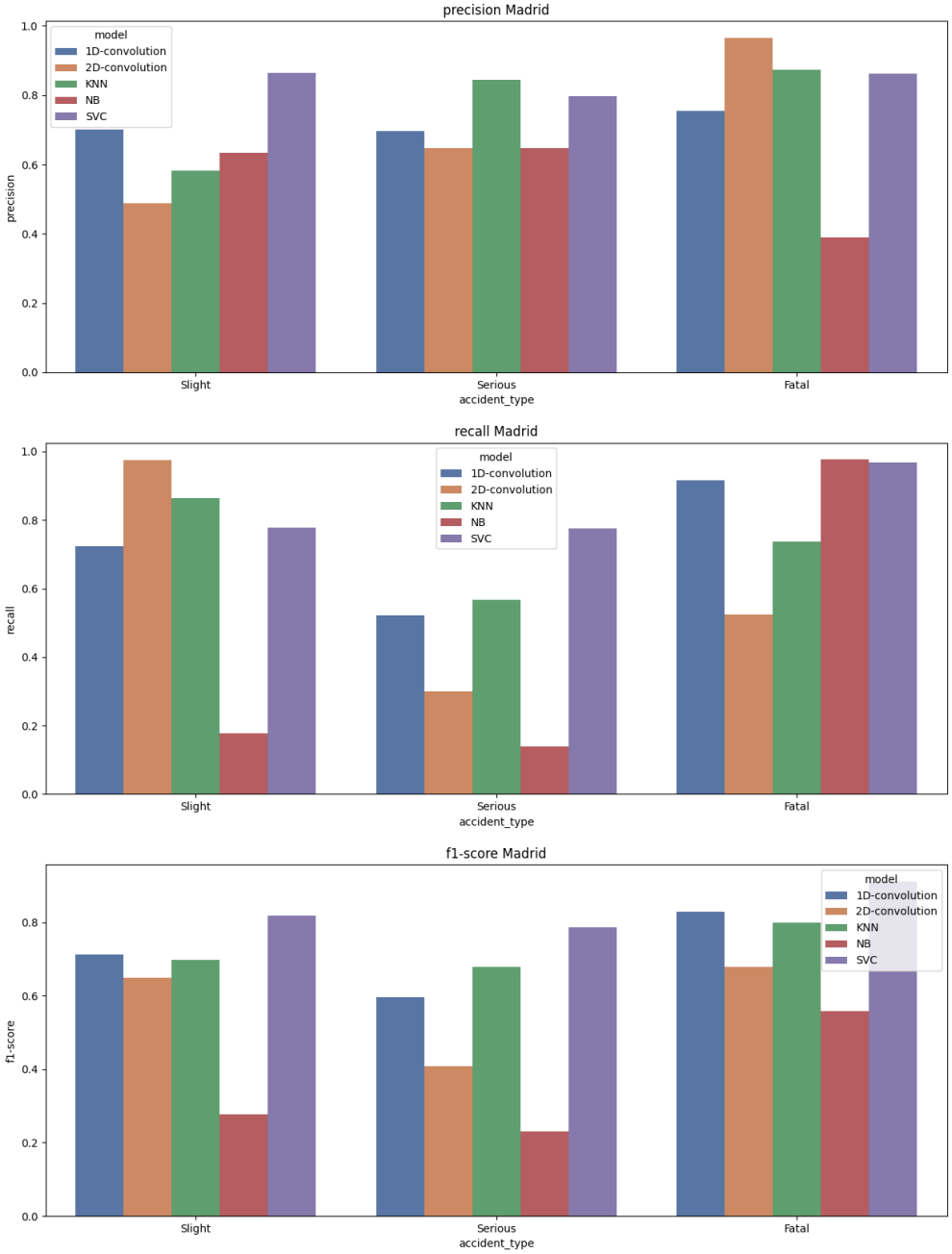
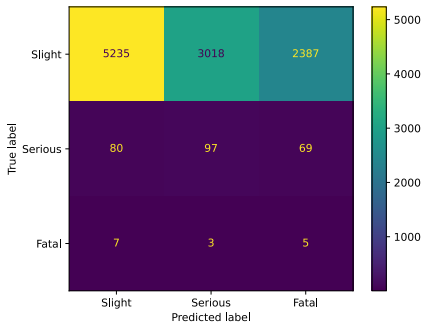
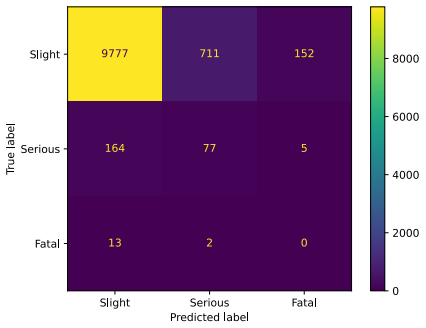


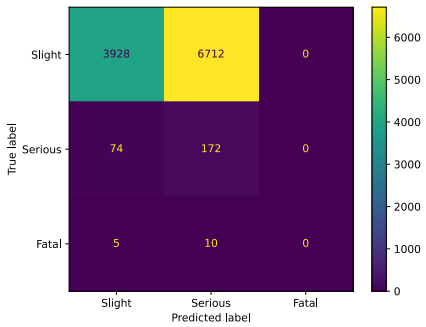
Figura 5.10: Comparativa de las métricas de las predicciones sobre el conjunto de entrenamiento de los modelos.



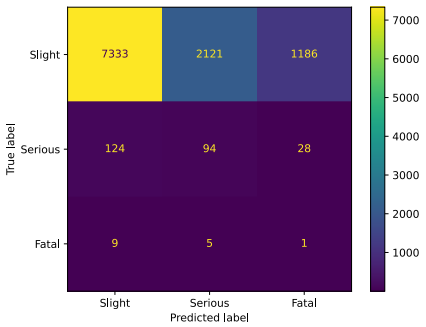
(a) Matriz de confusión de CNN-1D sobre test.



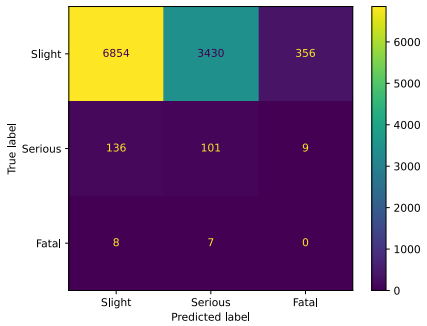
(b) Matriz de confusión de CNN-2D sobre test.



(c) Matriz de confusión de NB sobre test.



(d) Matriz de confusión de KNN sobre test.



(e) Matriz de confusión de SVC sobre test.

Figura 5.11: Matrices de confusión de los modelos sobre el conjunto de test.

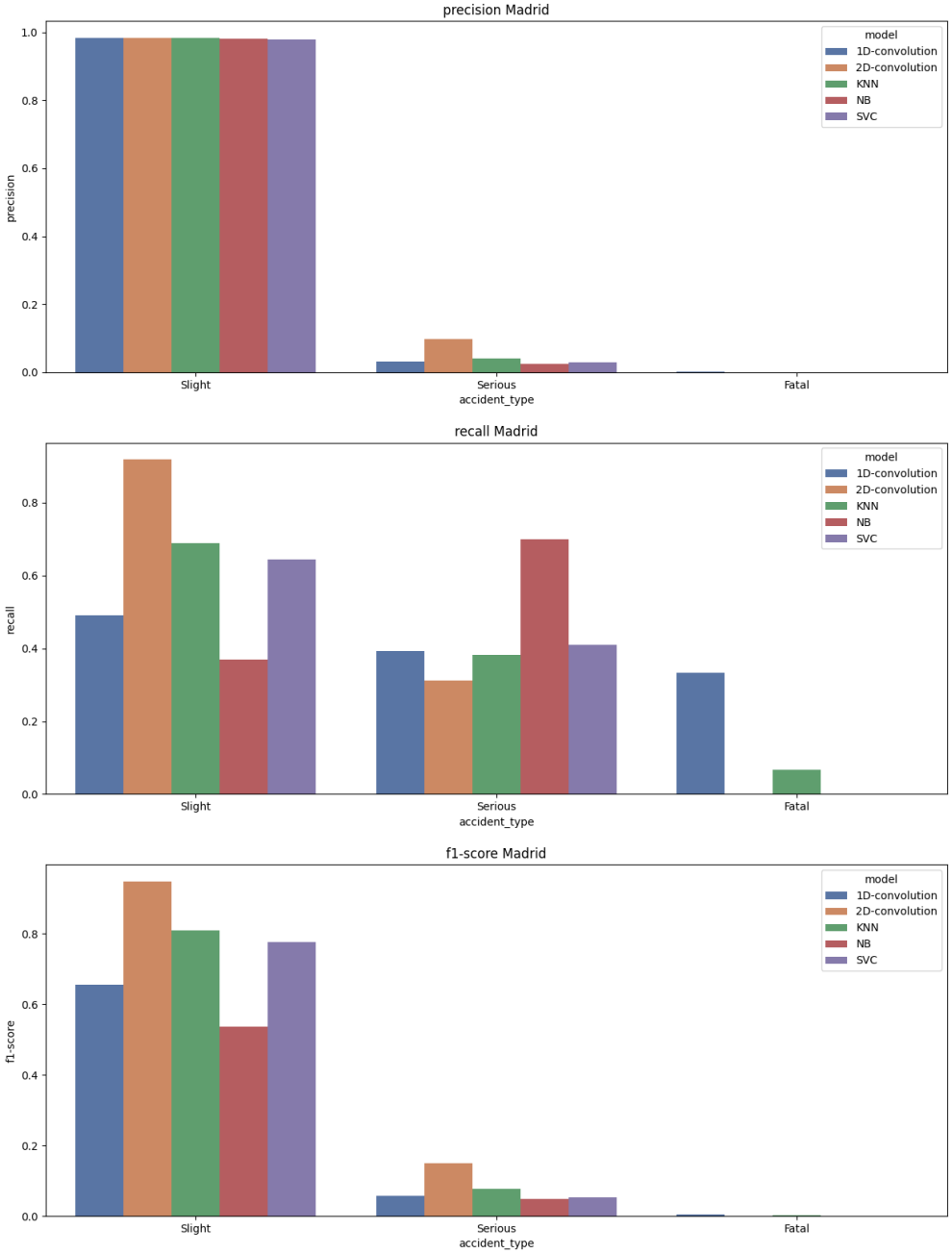


Figura 5.12: Comparativa de las métricas de las predicciones sobre el conjunto de test de los modelos.

6 Conclusiones

6.1 Líneas de investigación

Teniendo en cuenta los resultados de los experimentos, existen múltiples líneas de investigación en las que seguir desarrollando el proyecto de cara a obtener un mejor rendimiento:

1. Las transformaciones sobre las variables del dataset son un punto crítico, por lo que es necesario realizar un estudio de las mismas para probar si otras tipificaciones sobre las variables tienen un efecto positivo en el rendimiento de las clasificaciones. Un ejemplo de este caso es la variable *hora*, que puede ser transformada de acuerdo a otro rango de horas para declarar si un accidente se ha producido de día o de noche o modificar la tipificación del rango de edad. Estos cambios afectaría notablemente a los árboles de decisión producidos por XGBoost.

2. Se propone también usar herramientas como *Open Refine* [32] que permitan realizar clústers nominales sobre la localización de los accidentes para obtener una tipificación más sólida del tipo de carretera.

3. Otra de las líneas a investigar es el añadido de características provenientes de otros datasets localizados en el portal de datos abiertos de la ciudad de Madrid. Dichos datasets proveen de información tal como la velocidad media de los tramos, la densidad de tráfico o la temperatura ambiente. Estos posibles futuros predictores pueden ser claves para la clasificación, como por ejemplo las bajas temperaturas que afectan al agarre de los neumáticos.

4. Como siguiente opción se propone aumentar el número de características del dataset en base a las propias variables del conjunto de datos original, realizando más transformaciones que permitan obtener más predictores. Por ejemplo el mes, día y estación del año en el que se ha producido el accidente en base a la hora.

5. En lo que respecta a los algoritmos sería conveniente ampliar los hiperparámetros a utilizar por XGBoost, optimizándolos además mediante le algoritmo genético. En este proyecto se optimizan tres de ellos debido a las limitaciones computacionales, mientras que existen otros como *gamma* (parámetro de reducción mínima) *hojas máximas* (parámetro de regularización) o la *política de crecimiento* (controla la forma en la que los nuevos nodos se añaden al árbol) entre otros.

6. Otro enfoque consistiría estudiar alguna otra técnica para formar matrices en base a representaciones tabulares de datos, maximizando la información de los accidentes en base a la posición de las características.

7. Utilizar metaaprendizaje para entrenar modelos que averiguen los mejores hiperparametros para otros modelos

8. En lo que respecta al remuestreo de datos sería conveniente a analizar distintos métodos de remuestreo además de *Undersampling, Oversampling y SMOTE* ya estudiados. Existen numerosos parámetros de SMOTE-II que pueden ser configurados, entrenando los modelos con este nuevo remuestreo de datos sería posible aumentar el rendimiento de los mismos.

9. Utilizar técnicas de *Automated Machine Learning* como Neural architecture search (NAS) como AutoKeras [45] para encontrar la mejor estructura de la red para el problema. Las NAS buscan encontrar una arquitectura neuronal óptima basándose en ensayo y error. Este tipo de técnias requiere grandes recursos computacionales, por lo que para poder llevarla a cabo es necesario disponer de sistema con grandes capacidades de cómputo en *GPU*.

Bibliografía

M. Zheng, T. Li, R. Zhu, J. Chen, Z. Ma, M. Tang, Z. Cui, and Z. Wang, "Traffic accident's severity prediction: A deep-learning approach-based cnn network," *IEEE Access*, vol. 7, pp. 39 897–39 910, 2019.

H. Lingaraj, "A study on genetic algorithm and its applications," *International Journal of Computer Sciences and Engineering*, vol. 4, pp. 139–143, 10 2016.

T. Chen and C. Guestrin, "XGBoost," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, aug 2016. [Online]. Available: https://doi.org/10.1145%2F2939672.2939785

Nvidia, "Nvidia xgboost," 2022. [Online]. Available: https://www.nvidia.com/en-us/glossary/data-science/xgboost/

C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

G. James, D. Witten, T. Hastie, and R. Tibshirani, *An introduction to statistical learning*. Springer, 2013, vol. 112.

F. Emmert-Streib, Z. Yang, H. Feng, S. Tripathi, and M. Dehmer, "An introductory review of deep learning for prediction models with big data," *Frontiers in Artificial Intelligence*, vol. 3, 2020. [Online]. Available: https://www.frontiersin.org/article/10.3389/frai.2020.00004

Wikipedia, "Softmax function," 2022. [Online]. Available: https://en.wikipedia.org/wiki/Softmax_function

K. E. Koech, "Cross-entropy loss function," 2020. [Online]. Available: https://towardsdatascience.com/cross-entropy-loss-function-f38c4ec8643e

L. N. Smith, "A disciplined approach to neural network hyper-parameters: Part 1 – learning rate, batch size, momentum, and weight decay," 2018. [Online]. Available: https://arxiv.org/abs/1803.09820

S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," 2015. [Online]. Available: https://arxiv.org/abs/1502.03167

P. Ganesh, "Types of convolution kernels : Simplified," 2019. [Online]. Available: https://towardsdatascience.com/types-of-convolution-kernels-simplified-f040cb307c37

R. Khandelwal, "Filters and feature maps," 2020. [Online]. Available: https://towardsdatascience.com/convolutional-neural-network-feature-map-and-filter-visualization-f75012a5a49c

S. Kiranyaz, O. Avci, O. Abdeljaber, T. Ince, M. Gabbouj, and D. J. Inman, "1d convolutional neural networks and applications: A survey," 2019. [Online]. Available: https://arxiv.org/abs/1905.03554

N. S. Altman, "An introduction to kernel and nearest-neighbor nonparametric regression," *The American Statistician*, vol. 46, no. 3, pp. 175–185, 1992.

Python, "Python language," 2022. [Online]. Available: https://www.python.org/about/quotes/

pandas dev, "Pandas library," 2022. [Online]. Available: https://pandas.pydata.org/docs/getting_started/overview.html

tensorflow, "Tensorflow library," 2022. [Online]. Available: https://www.tensorflow.org

scikit learn, "Scikit-learn library," 2022. [Online]. Available: https://scikit-learn.org/stable/

dmlc, "Xgboost library," 2022. [Online]. Available: https://xgboost.readthedocs.io/en/stable/

Nvidia, "Cuda library," 2022. [Online]. Available: https://developer.nvidia.com/cuda-zone

I. Anaconda, "Anaconda library," 2022. [Online]. Available: https://www.anaconda.com/

P. Jupyter, "Jupyter notebook," 2022. [Online]. Available: https://jupyter.org/

——, "Jupyter lab," 2022. [Online]. Available: https://jupyterlab.readthedocs.io/en/stable/

jGraph, "Diagramsnet software," 2022. [Online]. Available: https://www.diagrams.net/

Google, "Google meets application," 2022. [Online]. Available: https://meet.google.com/

Github, "Github svc," 2022. [Online]. Available: https://github.com/

Projectpro.io, "Why data preparation is an important part of data science?" 2022. [Online]. Available: https://www.projectpro.io/article/why-data-preparation-is-an-important-part-of-data-science/242

P. de Datos Abiertos del Ayuntamiento de Madrid, "Accidentes de tráfico madrid," 2022. [Online]. Available: https://datos.madrid.es/portal/site/egob/menuitem.c05c1f754a33a9fbe4b2e4b284f1a5a0/?vgnextoid=7c2843010d9c3610VgnVCM2000001f4a900aRCRD&vgnextchannel=374512b9ace9f310VgnVCM100000171f5a0aRCRD&vgnextfmt=default

——, "Estructura del conjunto de datos," 2019. [Online]. Available: https://datos.madrid.es/FWProjects/egob/Catalogo/Seguridad/Ficheros/Estructura_DS_Accidentes_trafico_desde_2019.pdf

J. Goyvaerts and S. Levithan, *Regular Expressions Cookbook*, ser. Oreilly and Associate Series. O'Reilly Media, Incorporated, 2012. [Online]. Available: https://books.google.es/books?id=6k7IfACN_P8C

I. Metaweb Technologies, "Open refine project," 2022. [Online]. Available: https://openrefine.org/

Y. Sun, A. K. Wong, and M. S. Kamel, "Classification of imbalanced data: A review," *International journal of pattern recognition and artificial intelligence*, vol. 23, no. 04, pp. 687–719, 2009.

G. Haixiang, L. Yijing, J. Shang, G. Mingyun, H. Yuanyue, and G. Bing, "Learning from class-imbalanced data: Review of methods and applications," *Expert Systems with Applications*, vol. 73, pp. 220–239, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417416307175

P. Schober, C. Boer, and L. Schwarte, "Correlation coefficients: Appropriate use and interpretation," *Anesthesia and Analgesia*, vol. 126, p. 1, 02 2018.

D. School, "Comparing supervised learning algorithms," 2015. [Online]. Available: https://www.dataschool.io/comparing-supervised-learning-algorithms/

D. Singh and B. Singh, "Investigating the impact of data normalization on classification performance," *Applied Soft Computing*, vol. 97, p. 105524, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1568494619302947

N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: Synthetic minority over-sampling technique," *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, jun 2002. [Online]. Available: https://doi.org/10.1613%2Fjair.953

L. Van der Maaten and G. Hinton, "Visualizing data using t-sne." *Journal of machine learning research*, vol. 9, no. 11, 2008.

Y. Jiang, G. Tong, H. Yin, and N. Xiong, "A pedestrian detection method based on genetic algorithm for optimize xgboost training parameters," *IEEE Access*, vol. 7, pp. 118 310–118 321, 2019.

J. Brownlee, "Feature importance and feature selection with xgboost in python," 2016. [Online]. Available: https://machinelearningmastery.com/feature-importance-and-feature-selection-with-xgboost-in-python/

P. Kopelias, F. Papadimitriou, K. Papandreou, and P. D. Prevedouros, "Urban freeway crash analysis: Geometric, operational, and weather effects on crash number and severity," *Transportation Research Record*, vol. 2015, pp. 123–131, 12 2007.

Sklearn, "Gridsearchcv," 2021. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 2011, pp. 315–323.

H. Jin, Q. Song, and X. Hu, "Auto-keras: An efficient neural architecture search system," 2018. [Online]. Available: https://arxiv.org/abs/1806.10282

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

X.-Y. Liu, J. Wu, and Z.-H. Zhou, "Exploratory undersampling for class-imbalance learning," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 39, no. 2, pp. 539–550, 2009.

G. E. Hinton and S. Roweis, "Stochastic neighbor embedding," *Advances in neural information processing systems*, vol. 15, 2002.

M. Jain, "Hyperparameter tuning in xgboost using genetic algorithm," 2018. [Online]. Available: https://towardsdatascience.com/hyperparameter-tuning-in-xgboost-using-genetic-algorithm-17bd2e581b17

C. Spark, "Hyperparameter tuning in xgboost," 2017. [Online]. Available: https://blog.cambridgespark.com/hyperparameter-tuning-in-xgboost-4ff9100a3b2f

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