Universidad de Castilla-La Mancha

Escuela Superior de Ingeniería Informática

Trabajo Fin de Grado

Grado en Ingeniería Informática

Tecnología Específica de

Computación

Pasarela y Programación en Python de las APIs de Aprendizaje Automático Weka + Scikit-Learn + SHAPValues: Caso de Estudio de predicciones de lesiones en jugadores profesionales de futbol-sala

Autor

Pablo Moreira García

Abril, 2022

Trabajo Fin de Grado

Grado en Ingeniería Informática

Tecnología Específica de

Computación

Pasarela y Programación en Python de las APIs de Aprendizaje Automático Weka + ScikitLearn + SHAPValues: Caso de Estudio de predicciones de lesiones en jugadores profesionales de futbol-sala

Autor: Pablo Moreira García

Tutor: José Miguel Puerta Callejón

Co-Tutor: José Antonio Gámez Martín

Julio, 2022

Declaración de Autoría

Yo, Pablo Moreira García con DNI 48150755W, declaro que soy el único autor del trabajo fin de grado titulado “Pasarela y Programación en Python de las APIs de Aprendizaje Automático Weka + ScikitLearn + SHAPValues: Caso de Estudio de predicciones de lesiones en jugadores profesionales de futbol-sala” y que el citado trabajo no infringe las leyes en vigor sobre propiedad intelectual y que todo el material no original contenido en dicho trabajo está apropiadamente atribuido a sus legítimos autores.

Albacete, a …….. 2022

Fdo: .....

Resumen

Hoy en día el uso de métodos de aprendizaje automático está a la orden del día para la resolución de problemas de predicción, clasificación y/o agrupamiento. En estos últimos años, se ha impuesto el uso del lenguaje de programación *Python* como lenguaje paradigmático en su uso en temas relacionados con el aprendizaje automático y análisis de datos. Sin embargo, existen un gran esfuerzo previo en otros tipos de lenguajes, como por ejemplo Java, en la programación de paquetes muy poderosos en procesos de minería de datos, como por ejemplo *WEKA*.

En el caso de *Python* el paquete que se está imponiendo para su uso en este tipo de problemas es *Scikit-Learn*. Sin embargo, hay algoritmos y técnicas que no están programadas en este paquete y sí que están programadas y más que testadas en *WEKA* (*Java*).

Una de las tendencias en aprendizaje automático muy reciente es la denominada modelos explicables, o explicabilidad en inteligencia artificial. Básicamente consistirá en descubrir cómo se hacen los procesos de razonamiento en los modelos utilizados y hacerlos entendibles por usuarios no expertos. Existen varios paquetes programados en *Python* para generar este tipo de explicaciones, entre ellos *SHAP* el cual utilizaremos en nuestro proyecto, haciendo accesibles estos paquetes a modelos programados en *WEKA* con *Java*.

Para finalizar, utilizaremos como caso de estudio bases de datos reales con las características de jugadores profesionales de futbol sala recogidas, además de conocer las lesiones que han sufrido en la temporada correspondiente. Esta base de datos sería muy interesante de analizar para ver cuando un jugador sufre o no una lesión muscular junto con su explicación correspondiente.

El objetivo principal de esta propuesta consistirá en construir un módulo en lenguaje *Python* para integrar todos los algoritmos necesarios bajo el enfoque de *Scikit-Learn* de la librería *WEKA* y su uso en el análisis de la base de datos de jugadores profesionales de futbol sala.

Agradecimientos

En primer lugar, me gustaría agradecer a mis tutores José Miguel Puerta Callejón y José Antonio Gámez Martín, por su paciencia, confianza y la gran cantidad de conocimiento que han compartido conmigo, para la finalización de este proyecto.

También me gustaría agradecer a mis compañeros de la Delegación de Alumnos por su gran apoyo que me han dado durante el proceso del proyecto.

Por último, pero no menos importante, también me gustaría agradecer a mi familia, mis padres y mis amigos.

Índice general

Índice de Diagramas

Índice de tablas

Índice de graficas

Índice de figuras

Capítulo 1

Introducción

1.1 Introducción

Esta memoria presenta el desarrollo de un proyecto en el cual se pretenden hacer accesibles los algoritmos implementados con *WEKA* al mundo del aprendizaje automático desarrollado en *Python*. Gracias al uso de una clase como punto de unión entre *WEKA* y *Scikit-Learn*, la cual tiene desarrollada las funciones básicas de un modelo como son el aprendizaje, la capacidad de predecir tanto instancias únicas como bases de datos…, lograríamos que los algoritmos de *WEKA* fueran reconocidos por *Scikit-Learn* y muchas otras aplicaciones y librerías de *Python* logrando este objetivo.

Gracias a esto vamos a poder desarrollar distintos modelos utilizando indistintamente métodos de *WEKA* (*Java*) y de *Scikit-Learn* (*Python*) para poder comparar los resultados obtenidos con ellos, mostrando las posibles ventajas y desventajas de los distintos lenguajes y métodos utilizados.

También se quiere mostrar el funcionamiento de los distintos modelos y de qué forma trabajan de manera visual y fácilmente comprensible, usando para ello librerías como *SHAP*, donde se evalúa la importancia de los distintos datos utilizados, y cuales afectan en mayor o menor medida a la hora de predecir la clase objetivo en cuestión, en nuestro caso si un jugador puede llegar a lesionarse o no.

1.2 Objetivos

El objetivo principal de este proyecto es el desarrollo de código en *Python* para la construcción de un módulo que nos permita integrar los algoritmos disponibles y modelos desarrollados en *WEKA* bajo el enfoque de *Scikit-Learn* y su uso, en este caso, para el análisis de las bases de datos reales sobre jugadores de futbol sala y la predicción de una posible lesión. Esto se codificará en una o varias libretas de *Jupyter*, en los cuales se implementarán distintos métodos para la carga, procesamiento y visualización de los distintos tipos de datos (Diagrama 1). Los modelos desarrollados gracias a *Python-Weka-Wrapper3*, implementaran distintas técnicas utilizadas para el desbalanceo. Por último, se representará como han trabajado dichos modelos gracias a librerías destinadas a la explicabilidad en inteligencia artificial como es *SHAP*.

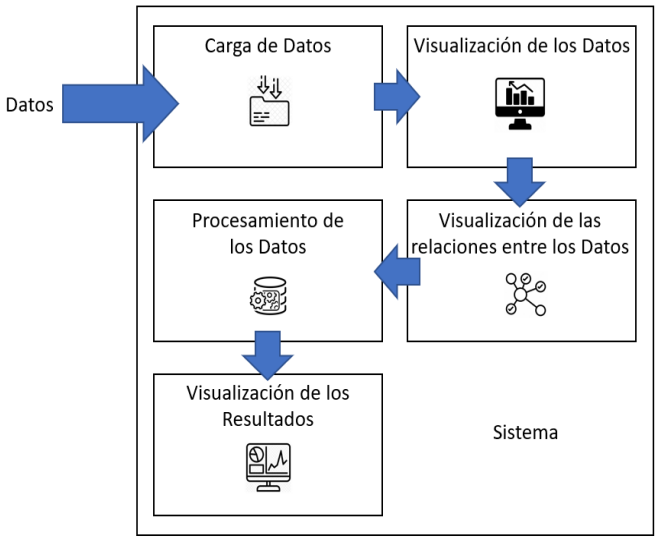


Diagrama 1: Flujo de los datos en nuestro sistema

La motivación para el desarrollo de este proyecto también parte de que los expertos que nos han facilitados las bases de datos utilizan algoritmos diseñados en *WEKA*, los cuales no están implementados en *Scikit-Learn*, y con este proyecto le daríamos solución a este problema de cara a futuros análisis, ya que cualquier persona puede utilizar fácilmente *WEKA*, debido a su interfaz cómoda y entendible para el usuario, y gracias a esto podríamos pasar esos modelos a *Scikit-Learn* y otros módulos dedicados a la explicabilidad de los modelos como *SHAP*, o módulos para la representación gráfica de datos y resultados como *Seaborn*.

1.3 Competencias

- Capacidad para evaluar la complejidad computacional de un problema, conocer estrategias algorítmicas que puedan conducir a su resolución y recomendar, desarrollar e implementar aquella que garantice el mejor rendimiento de acuerdo con los requisitos establecidos.

- Capacidad para conocer los fundamentos, paradigmas y técnicas propias de los sistemas inteligentes y analizar, diseñar y construir sistemas, servicios y aplicaciones informáticas que utilicen dichas técnicas en cualquier ámbito de aplicación.

- Capacidad para adquirir, obtener, formalizar y representar el conocimiento humano en una forma computable para la resolución de problemas mediante un sistema informático en cualquier ámbito de aplicación, particularmente los relacionados con aspectos de computación, percepción y actuación en ambientes entornos inteligentes.

- Capacidad para conocer y desarrollar técnicas de aprendizaje computacional y diseñar e implementar aplicaciones y sistemas que las utilicen, incluyendo las dedicadas a extracción automática de información y conocimiento a partir de grandes volúmenes de datos.

1.4 Estructura del proyecto

La memoria consta de los siguientes capítulos:

- Introducción: Las razones que han motivado la realización de este proyecto y los principales objetivos.
- Estado del Arte: Introducción a los distintos conceptos clave para el desarrollo del proyecto, con especial atención a la ciencia de datos, el aprendizaje automático y sus distintas técnicas, una descripción del funcionamiento y utilidad de las librerías utilizadas (*Scikit-Learn*, *SHAP*, *Python-Weka-Wrapper3*, *Seaborn* entre otras) y descripción de la metodología agile *SCRUM* utilizada para el desarrollo de nuestro proyecto.
- Metodología y Desarrollo: En este capítulo se describirá el proyecto realizado separado en los distintos *Sprints* (metodología *SCRUM*) que ha necesitado, el tiempo que han necesitado, trabajo realizado en cada uno de ellos, problemas que se han dado y unas conclusiones sobre cada *Sprint*.
- Experimentos y Resultados: Tras el capítulo de Metodología y Desarrollo, en este vamos a detallar las bases de datos utilizadas, y los resultados obtenidos por los modelos desarrollados.
- Conclusiones y futuro trabajo: Por último, el proyecto concluirá con una reflexión sobre el trabajo realizado y sobre el análisis basado en aprendizaje automático para resolver problemas de minería de datos, además de proponer posibles líneas de trabajo que podrían ser utilizadas en un futuro.

Capítulo 2

Estado del Arte

2.1 Introducción

En este capítulo vamos a tratar lo que es la minería de datos y el aprendizaje automático, como funcionan explicando sus procesos y las técnicas más utilizadas. También vamos a explicar la librarías que hemos utilizado, explicando su utilidad. Acabando este capítulo trataremos los modelos que hemos seleccionado para el proyecto, lo que es el desbalanceo y técnicas para solucionarlo y métricas comúnmente usadas para ponderar lo correctos que son nuestros modelos. Y por último explicaremos lo que es la metodología *SCRUM* utilizada para el desarrollo del proyecto, la aplicación de esta metodología en nuestro proyecto se desarrollará más adelante.

2.2 Minería de Datos

2.2.1 Introducción

La minería de datos es el proceso de encontrar información útil dentro de grandes conjuntos de datos. Utiliza análisis matemático y estadístico para encontrar patrones, tendencias o reglas que expliquen el comportamiento de los datos en un determinado contexto.

Estos patrones y tendencias se pueden definir como un modelo de minería de datos que podrá utilizarse posteriormente para generar predicciones y deducir relaciones al aplicarlos a nuevos datos.

2.2.2 Proceso de Minería de Datos.

Conocemos como proceso de minería de datos al proceso iterativo realmente denominado *KDD* (*Knowledge Discovery in Databases*), minería de datos sería únicamente el paso de la creación de modelos.

Este proceso se puede desarrollar en 6 pasos (Diagrama 2):

1. Definición del Problema: En el primer paso se desarrolla el entendimiento sobre el contexto que rodea al problema, se descubre el conocimiento y la información que es realmente relevante y se definen los objetivos del proceso del *KDD*.
2. Preparación de los datos: Durante este paso se almacenan y se integran todos los datos relevantes obtenidos del paso 1, homogeneizando toda la información para facilitar el procesamiento y en análisis de estos en los próximos pasos.
3. Exploración de los datos: La exploración de los datos se basa en la “limpieza y transformación” de los datos recogidos gracias a los pasos anteriores. Es decir, eliminación de ruido, datos aislados y datos con una elevada correlación, uso de estrategias para manejar información nula o faltante, debido a que estos datos “sucios” pueden provocar un mal funcionamiento de nuestros modelos obteniendo unos resultados poco fiables o incluso no válidos.

En este paso también se realiza el análisis exploratorio de datos, el cual es una exploración descriptiva de los datos, permitiendo entender mejor la información que contiene cada variable, detectar posibles errores e incluso dar información sobre que variables son más adecuadas como predictoras que otras. También incluiríamos el preprocesado, es decir las transformaciones de la base de datos con el objetivo de que sean interpretadas por los algoritmos de aprendizaje automático lo más eficientemente posible.

4. Creación de los modelos: Este paso es realmente lo que denominamos proceso de minería de datos, donde utilizamos diferentes algoritmos para obtener patrones en los datos anteriores. Este proceso se basa en saber la tarea que queremos realizar (predicciones, asociaciones, identificar secuencias de datos…), seleccionar el algoritmo o algoritmos a utilizar según los parámetros a utilizar, los criterios de evaluación, que tipo de modelo seria óptimo para encontrar patrones en mis datos, y finalmente si coincide el algoritmo seleccionado con el objetivo final del proceso de *KDD*.

   En este paso se suelen realizar el entrenamiento de los modelos gracias al cual el modelo aprende a partir de los datos de entrenamiento para que luego sea capaz de hacer buenas predicciones.

5. Selección y validación de los modelos: En esta etapa encontramos los modelos que más se ajustan a nuestros problemas sean de clasificación o de regresión, haciendo pruebas con el conjunto de entrenamientos y viendo los que dan mejores resultados en sus predicciones.

   También se suele realizar la búsqueda de la combinación optima de hiperparametros para el modelo, esto se realiza gracias a funciones como "*GridSearch()*" o "*RandomizedSearch()*", estas funciones realizan una búsqueda exhaustiva evaluando todas las combinaciones de hiperparametros, aunque es una buena estrategia pare encontrar un bueno modelo, tiene el inconveniente del tiempo de ejecución que necesita, normalmente muy elevado, con "*RandomizedSearch()*" se reduce en gran medida el tiempo necesario y explora el espacio de búsqueda de una forma más distribuida.

En este paso se suele realizar la validación de los modelos, es decir, ver cómo funciona el modelo antes de utilizar el conjunto de datos de test. Esta validación se puede realizar con el conjunto de entrenamiento, pero utilizando estrategias de validación cruzada, como puede ser *KFold*, *RepeatedKFold*, *LeaveOneOut*. Cada estrategia funciona internamente de formas distintas, pero todos se basan en la idea de ajustar y evaluar el modelo de forma repetida, utilizando cada vez distintos subconjuntos como entrenamiento y test (todos creados a partir del conjunto de entrenamiento principal) y obteniendo en cada iteración una estimación del error.

6. Implementar y actualizar los modelos: Finalmente utilizamos los patrones y modelos obtenidos de la fase anterior, los cuales son analizados y evaluados para convertirse en conocimiento y ser transformado a acciones dentro del proceso de negocio. Algunas de las tareas que se efectúan son planear la implementación, monitorizar y mantener los modelos, realización de un informe final del proyecto y una revisión final de lo correcto y lo incorrecto del modelo. En este paso solemos realizar predicciones con datos del conjunto test o datos que no han tenido los modelos para su entrenamiento y validación, gracias a esto podemos comprobar el error que tiene nuestro modelo, dependiendo del problema son más interesantes utilizar unas métricas u otras que veremos más delante de una forma más detallada.

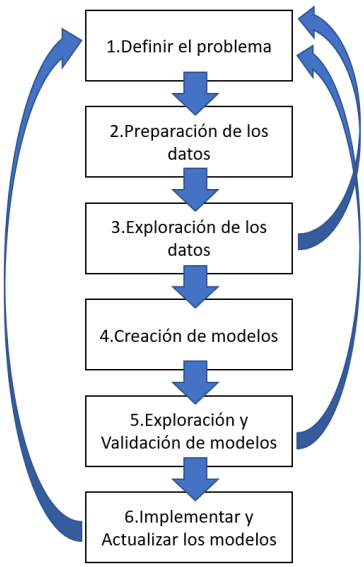


Diagrama 2: Proceso iterativo de minería de datos.

2.3 Aprendizaje Automático (*Machine Learning*)

2.3.1 Introducción

El aprendizaje automático en un subconjunto especifico de la Inteligencia Artificial, el cual se basa en darle a la maquina un ciclo de retroalimentación que le permite aprender de la experiencia, este campo existe desde la década de 1980, aunque recientemente hemos tenido un gran avance en poder de procesamiento y almacenamiento de datos, pudiendo implementar y desarrollar el aprendizaje automático cada vez más.

Este puede implementarse en muchos casos de uso, mientras que existan datos que analizar, el aprendizaje automático podrá darles un sentido.

Mediante métodos matemáticos y estadísticos, las técnicas y los algoritmos de aprendizaje automático se entrenan y mejoran, tanto para hacer clasificaciones y predicciones de los datos, como para encontrar información fundamental en conjuntos de datos para procesos de minería de datos.

2.3.2 Cómo funciona el aprendizaje automático

Un algoritmo de aprendizaje automático se puede dividir en tres partes fundamentales (Diagrama 3):

- Un proceso de decisión, se encarga de la predicción o clasificación a partir de unos datos de entrada etiquetados o no, normalmente cuando se entrena un algoritmo de aprendizaje automático, los datos de entrada ya están etiquetados, cuando se quiere realizar una predicción real los datos de entrada están sin etiquetar.
- Una función de error, se encarga de evaluar la precisión del modelo.
- Un proceso de optimización de modelos, se encarga de analizar los fallos y luego actualizar el proceso de decisión, de modo que la próxima vez el fallo no sea tan grave.

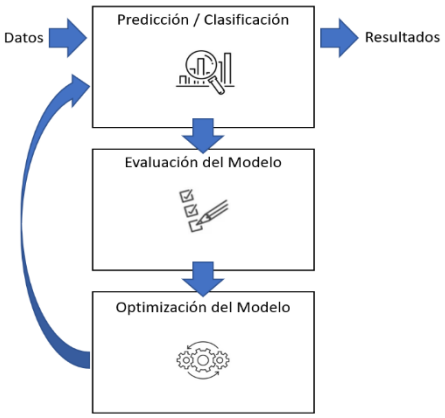


Diagrama 3: Funcionamiento de los algoritmos de aprendizaje automático.

2.3.3 Métodos de Aprendizaje Automático

Existen cuatro categorías principales de algoritmos de machine learning (Diagrama 4):

- Aprendizaje Supervisado, se produce cuando se entrena a los algoritmos con datos etiquetados para realizar predicciones o clasificaciones.
- Aprendizaje No Supervisado, los algoritmos buscan similitudes para agrupar conjuntos de datos sin etiquetar.
- Aprendizaje Semisupervisado, durante el entrenamiento, utiliza un conjunto de datos etiquetados más pequeño para guiar la clasificación y la extracción de características de un conjunto de datos sin etiquetar de mayor tamaño.
- Aprendizaje Por Refuerzo, se produce cuando una máquina aprende por medio de prueba y error.

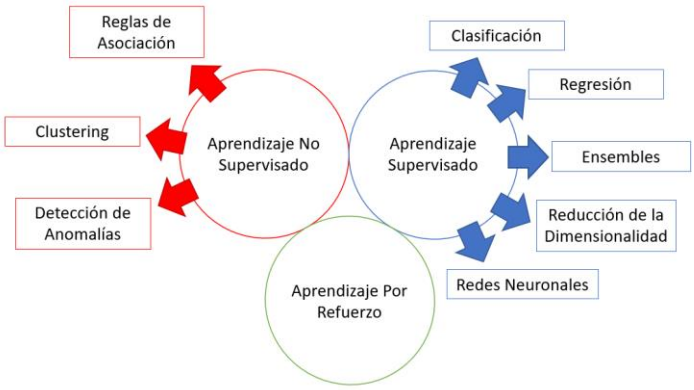


Diagrama 4: Métodos y Algoritmos de Aprendizaje Automático

2.4 Desbalanceo de Clases

2.4.1 Qué es y por qué es un problema

En los últimos años a medida que avanza la tecnología para clasificar información, la cantidad de datos clasificables también lo hace. A causa de esto también se vuelve más complicada la clasificación de dichos datos, debido al gran tamaño y a su desbalanceo. El desbalanceo se da a causa de la desproporción del número de elementos de cada clase dentro de un conjunto de datos, donde nos encontraríamos con una clase minoritaria, la cual tiene menos muestras, y una mayoritaria, la que tiene el mayor número de muestras.

El desbalance puede darse en mayor o menor medida dependiendo del número de muestras que tiene la clase mayoritaria por cada una que tiene la clase minoritaria.

Este desbalanceo es un problema debido a que muchos algoritmos de clasificación se ven afectados por la distribución desigual de las clases, ya que no manejan bien este tipo de bases de datos. Muchos asumen que las bases de datos con las que se entrenan están balanceadas y su objetivo es minimizar el error total, es decir están basados en la precisión, por esto la clase minoritaria puede llegar a ser tratada como ruido o valores anómalos debido a su baja influencia sobre el modelo. También en muchas ocasiones, asumen que todos los errores tienen el mismo coste. Por lo general en estos casos la clase minoritaria es la más perjudicada y normalmente estos datos son los que más necesitan ser bien clasificados.

Estos problemas se ven sobre todo en áreas como la detección de enfermedades, donde hay bases de datos con miles de instancias negativas de ciertas enfermedades frente a muy pocas positivas.

2.4.2 Métodos para solucionarlo

En la actualidad son muchos los investigadores que se centran en solucionar el problema del desbalanceo de clases, sobre todo en clases binarias, como puede ser la identificación de enfermedades. Esta comunidad principalmente se centra en estas dos técnicas:

- Preprocesado: Se basa en hacer un preprocesamiento de los datos antes del entrenamiento de los modelos para minimizar el desajuste entre las clases.
- Refinamiento de los algoritmos: Se basa en ajustar los parámetros de los clasificadores para que tengan en cuenta el desbalanceo de las clases y/o modificar la penalización de las malas predicciones.
- Uso de ensembles, teniendo en cuenta la ratio de desbalanceo en la generación de la muestra

2.4.3 Técnicas de Preprocesado

Para enfrentar el desbalanceo de clases se aplican distintos métodos de remuestreo de los datos para cambiar la distribución de las clases, transformando la base de datos en una más equilibrada, estas técnicas las podemos separar en *Oversampling* y *Undersampling*:

- *Oversampling*:

  El *Oversampling* soluciona el desbalanceo replicando datos de la clase minoritaria (Diagrama 5), existen distintas formas de aplicar el *Oversampling* entre los que cabe destacar:

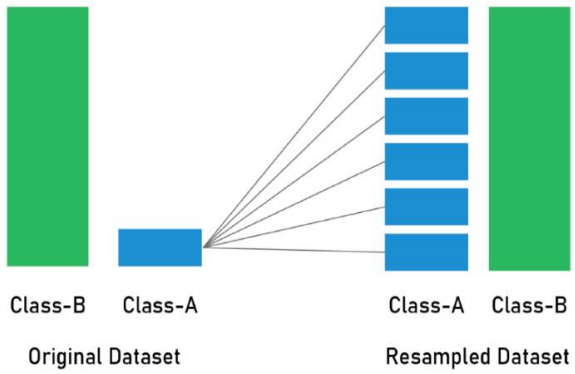


Diagrama 5: Oversampling de la clase minoritaria

  - ***Random Oversampling***: Se basa en duplicar aleatoriamente instancias de la clase minoritaria y añadirlas al conjunto de datos. Las instancias son seleccionadas del conjunto de entrenamiento con reemplazo, es decir, una misma instancia puede ser seleccionada varias veces. Con *Oversampling* podemos indicar la proporción de la clase mayoritaria que queremos mantener.

  - ***SMOTE (Synthetic Minority Oversampling TEchnique):*** Se puede considerar una técnica de *data-augmentation*, es decir, una técnica que se utiliza para aumentar el tamaño del conjunto de entrenamiento generando nuevos datos modificados a partir de los datos existentes, es una buena práctica para prevenir el sobreajuste o si la base de datos inicial es muy pequeña para entrenar con ella (Diagrama 6). Esta técnica se aplica sobre la clase minoritaria, y se generan nuevas instancias mediante interpolación a partir de sus vecinos más cercanos de la misma clase.

Normalmente el procedimiento se basa en 4 pasos:

- Elegir una instancia de la clase minoritaria e
- Calcular sus k-vecinos más cercanos (si k = 5): e_1, e_2, …, e_k.
- Seleccionar uno de los vecinos e_i.
- Construir una nueva instancia interpolando entre e y e_i usando aleatoriedad.

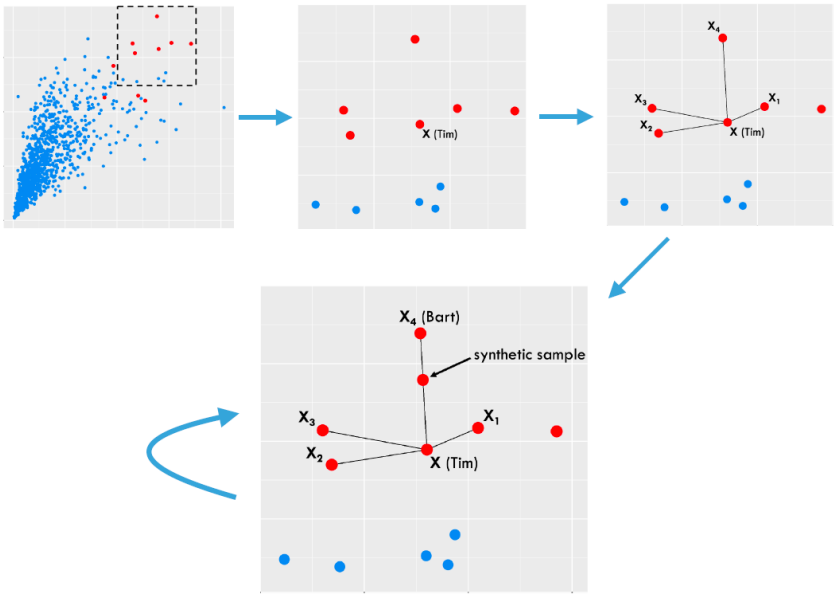


Diagrama 6: Técnica SMOTE gráficamente

- *Undersampling*:

Se basa en reducir el número de observaciones de la clase mayoritaria para conseguir una base de datos balanceada (Diagrama 7). Es recomendable utilizar este método solo cuando nuestra base de datos es muy grande y cuenta con muchas muestras ya que reduciendo instancias también mejoramos el tiempo de entrenamiento del modelo. En el proyecto hemos utilizado:

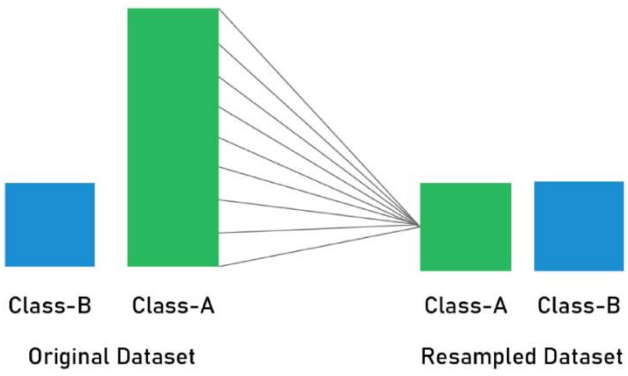


Diagrama 7: Undersampling de la clase mayoritaria

- *Random Undersampling*: Se basa en seleccionar de forma aleatoria instancias de la clase mayoritaria y eliminarlas del conjunto de entrenamiento, consiguiendo reducir el número de instancias de la clase mayoritaria en la nueva versión del conjunto de entrenamiento, esta es una técnica de muestreo sin reemplazo. Donde el objetivo del muestreo es crear un *dataset* de entrenamiento modificando la distribución de la variable objetivo, obteniendo un conjunto de datos más balanceado. Con *Undersampling* podemos indicar la proporción de la clase mayoritaria que queremos mantener.

- *Random Undersampling y Random Oversampling*: Utilizar *Undersampling* seguido de *Oversampling*, nos ayuda a buscar una ratio que nos permita no perder algunos conceptos de la clase mayoritaria, y no tener que generar tantas copias de la clase minoritaria, reduciendo así el sobreajuste.

2.4.4 Refinamiento de Algoritmos

Otro método de solucionar el desbalanceo de clases es mediante el aprendizaje sensible al coste. Este método tiene en cuenta los costes de los errores que se pueden llegar a cometer en una clasificación durante la construcción de un modelo. La diferencia es que el modelo se basaría en minimizar el coste en vez de minimizar la tasa de error del clasificador, prestando más atención a la clase minoritaria la cual suele ser la de mayor importancia. Esto se suele lograr asignando pesos a las clases modificando el conjunto de entrenamiento en función de los costes. Otra forma seria modificando el funcionamiento interno de los algoritmos para que tenga en cuenta los costes.

2.5 Explicabilidad

Actualmente cuando realizamos modelos de aprendizaje automático, se suelen ver como una caja negra, cuyo funcionamiento es únicamente entendido por un grupo pequeño de personas y tienen una baja interpretabilidad. A la gente que utiliza estos modelos también les beneficiaria una explicación de como el modelo ha llegado a cierta predicción y que valores ha tomado en consideración para ello en mayor o menor medida.

Para solucionar esto cada vez se van implementando más herramientas dedicadas a la explicabilidad de los modelos y/o de los procesos de decisión basados en ellos.

Estas herramientas deben mostrar los datos que utilizan los modelos para sus predicciones, estimar bajo qué información o estimulo podría cambiar la predicción de un modelo, o simplificar los modelos hasta el punto de que los haga comprensibles para la audiencia que lo necesite.

Este campo de la explicabilidad de los modelos actualmente se le conoce como *XAI* (*Explainable Artificial Intelligence*), donde se tienen en cuenta tres factores fundamentales:

- La naturaleza del modelo a explicar, que puede hacerlo desde transparente como un árbol de decisión hasta totalmente opaco e ininteligible como un modelo de aprendizaje profundo.
- La audiencia objetivo, ya que dependiendo de esta se buscará su explicabilidad con finalidades diferentes, desde entender mejor los modelos para extender sus capacidades a simplemente información en las implicaciones que tiene el uso de esta tecnología en cuestiones sensibles como la privacidad de los datos.
- La forma en la que dicha explicación se va a generar y va a ser presentada, ya que dependerá del grado de conocimiento de la audiencia y las posibilidades que brinde el modelo de ser explicado de una u otra forma, es decir, mostrar las explicaciones de una forma que una persona pueda entenderlo fácilmente sin perderse con tecnicismos.

2.5.1 SHAP

Con el objetivo de la explicabilidad de los modelos de una forma clara, existe la librería *SHAP* (*SHapley Additive exPlanations*), utilizada en nuestro proyecto para explicar las predicciones de cualquier modelo de aprendizaje automático. *SHAP* es un enfoque de teoría de juegos para explicar el resultado de cualquier modelo.

El valor clásico de *Shapley* (*Shapley values*) es la contribución marginal promedio de una característica a través de todos los subconjuntos de variables posibles. *SHAP* está basado en estos valores *Shapley*, donde su objetivo es interpretar y explicar las predicciones de un modelo mediante la contribución de cada parámetro de entrada.

Para un modelo donde la función predictiva es f(x) y F es el conjunto de todos los parámetros de entrada del modelo, los valores *Shapley* se pueden calcular con la siguiente formula:

$$\varphi_i = \sum_{S \subseteq F \setminus \{i\}} \frac{|S|!(|F| - |S| - 1)!}{|F|!} [f_{S \cup \{i\}}(x_{S \cup \{i\}}) - f_S(x_S)]$$

En esta fórmula, |F| es el número de parámetros de entrada del modelo, S es un conjunto de características que no incluyen la i-ésima característica, |S| es la cardinalidad de este subconjunto y fs() representa la función de predicción probabilística del modelo.

2.5.2 Métodos de SHAP

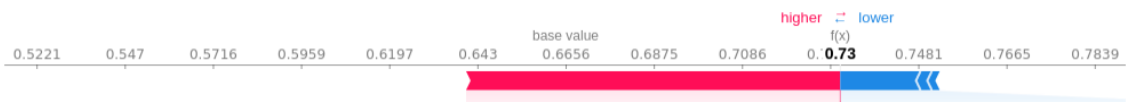
SHAP nos proporciona distintos métodos para los distintos algoritmos de aprendizaje automático, entre los que se encuentran los siguientes:

- ***DeepExplainer***: Método utilizado para modelos de aprendizaje profundo.
- ***TreeExplainer***: Método para modelos basados en arboles de decisión como *DecisionTree, XGBoost, RandomForest*…
-
- ***LinearExplainer***: Método útil para modelos lineales como *LinearRegression*.
- ***GradientExplainer***: Método utilizado para modelos de aprendizaje profundo pero que aproxima los valores *SHAP* (*SHAP values*) con gradientes.
- ***KernelExplainer***: Este método puede ser utilizado para cualquier modelo, ya sea basado en árboles, de aprendizaje profundo, modelos lineales…

2.5.3 Visualizaciones

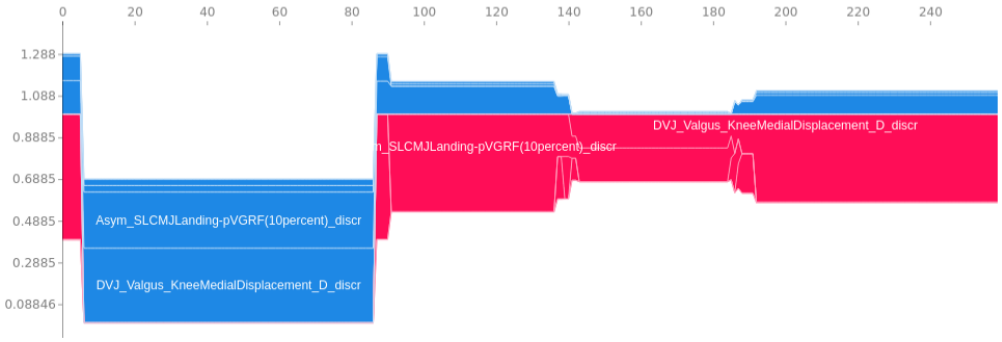
Esta librería incluye una gran variedad de métodos de representación gráfica de las explicaciones, los que hemos utilizado en nuestro proyecto son:

- *Force Plots*: Para visualizar la importancia de cada parámetro de entrada en la predicción del modelo (Gráfica 1).



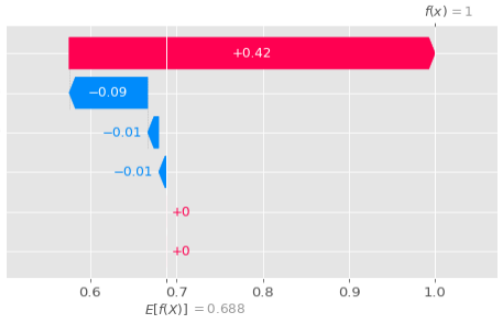
Gráfica 1: Representación de la importancia que tiene cada parámetro en una predicción

También se puede ver la importancia de estas características en las predicciones de todo el conjunto de datos (Gráfica 2).



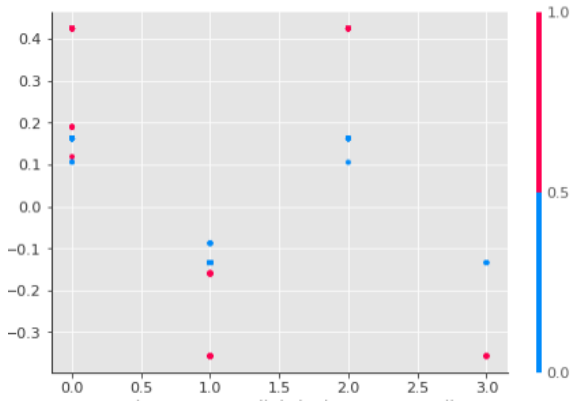
Gráfica 2: Representación de la importancia que tiene cada parámetro en las predicciones de todo el conjunto de datos

Si queremos ver de una forma más desglosada el impacto de cada característica a la predicción se pueden utilizar los *Waterfall Plot (Gráfica 3)*.



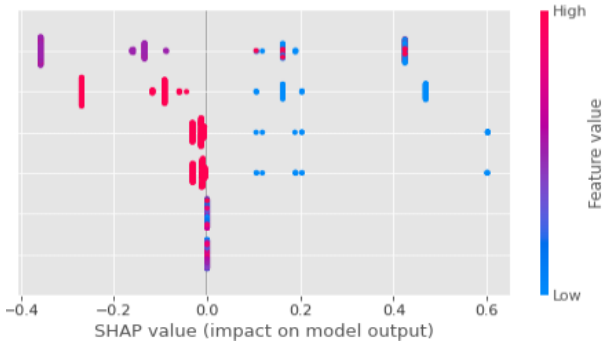
Gráfica 3: Representación de la importancia que tiene cada parámetro en una predicción de forma desglosada

- *Dependence Plots*: Se puede utilizar para visualizar la relación de un parámetro de entrada con respecto a las predicciones del modelo, este gráfico mostrara la variable seleccionada con la que esté más relacionada cuando se realiza la predicción (Gráfica 4).



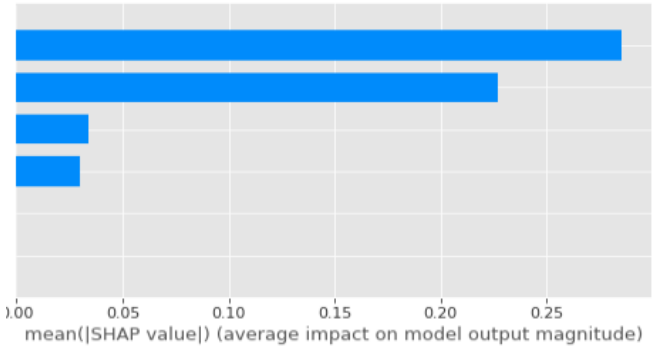
Gráfica 4: Representación de una variable con la variable que más relación tenga en una predicción

- *Summary Plots*: Se utilizan para ver la relación de todas las variables con el modelo y el impacto de estas. También podemos ver el valor SHAP de cada característica del conjunto (Gráfica 5).



Gráfica 5: Representación de los valores SHAP de cada característica

También se puede ver la importancia de las variables con un diagrama de barras (Gráfica 6).



Gráfica 6: Importancia de las características en el modelo

2.6 WEKA

WEKA es un software de código abierto bajo la *GNU* (*General Public License*) y desarrollado en la universidad de *Waikato*, que proporciona herramientas para el preprocesamiento de datos, implementación de muchas técnicas y algoritmos de aprendizaje automático y herramientas de visualización, y poder aplicarlas a problemas reales. Las bases de datos que utiliza esta aplicación tienen la extensión *ARFF* (*Attribute-Relation File Format*) los cuales son ficheros de texto *ASCII* que describen una serie de instancias con atributos comunes.

2.7 Scikit-Learn

Scikit-Learn es una de las principales librerías para *Python* utilizadas en el mundo de la ciencia de datos gracias a la gran variedad de algoritmos que contiene, clasificación, regresión, técnicas de agrupamiento/segmentación, reducción de dimensionalidad… Aparte permite un uso compatible con otras librerías utilizadas en el sector como *NumPy*, *Matplotlib*, *Pandas* o *Seaborn* que las explicaremos más adelante.

Todo esto la convierte en una de las herramientas básicas y principales para programar y estructurar sistemas de análisis de datos y modelado.

2.8 NumPy

NumPy es una de las principales librerías utilizadas para el cálculo numérico y el análisis de datos, especialmente para grandes volúmenes de datos. Gracias a esta librería apareció la clase de objetos llamada *array*, estructura que permite representar datos del mismo tipo organizada en forma de tabla o cuadrícula de distintas dimensiones, y la aparición de funciones muy eficientes para su manipulación.

Antes de la aparición de *NumPy* se utilizaban las listas genéricas ofrecidas por *Python*, pero estos *arrays* tienen muchas mejoras, son más compactos y utilizar menos memoria, tienen una mejor velocidad de procesamiento, también tienen muchas más funcionalidades como la posibilidad de utilizar operaciones vectorizadas como la suma o la multiplicación de elementos y la posibilidad de almacenar objetos de diferentes tipos, en *Python* para lograr esto debería almacenar la información de tipo para cada elemento y ejecutar un código de distribución de tipos al operar en cada elemento, cosa que es muy poco eficiente y que limita la cantidad de operaciones con listas disponibles.

2.9 Pandas

Pandas es una librería de *Python* de código abierto que se usa más ampliamente para problemas de ciencia de datos y aprendizaje automático. Esta construida sobre la librería *NumPy*, la cual nos facilita el uso de *arrays* multidimensionales.

Pandas como una de las librerías de gestión de datos más populares, funciona bien con otros muchos módulos, por ejemplo, para trazar funciones de *Matplotlib*, entrenar

algoritmos de aprendizaje automático en *Scikit-Learn*... Normalmente se incluye en las distribuciones de *Python*.

Pandas facilita la realización de las tareas más repetitivas asociadas a trabajar con datos, limpieza de datos, normalización, visualización, análisis estadístico, carga y almacenamiento de datos y mucho más. Todo esto gracias a la estructura de datos *DataFrame*, esta es una estructura con dos dimensiones en la cual se pueden almacenar datos de distintos tipos en columnas, es similar a una hoja de Excel, una tabla de SQL o un *data.frame* de *R*.

2.10 Matplotlib

Matplotlib fue una de las primeras librerías que se desarrollaron para la visualización de datos y trazado de gráficos para el lenguaje *Python*, siendo clave para trabajar con librerías más avanzadas como *Seaborn*.

Pyplot como parte de *Matplotlib* le permite ser una gran alternativa de código abierto a *MATLAB*, esta *API* es una colección de funciones y objetos que se encargan de hacer cambios en las figuras que se están dibujando, desde crear un área de trazado, a dibujar líneas en un área o decorar la gráfica con texto…

Para organizar y estructurar la visualización de los gráficos, *Matplotlib* suele crear una figura que representa al gráfico, indicando el número de ejes y después mostrando la figura, esta técnica se conoce como *Object-Oriented Style*.

Utiliza también *Pandas*, para la manipulación y el análisis de datos, y a diferencia de *NumPy*, *Pandas* no tiene una dependencia obligatoria con *Matplotlib*.

2.11 Seaborn

Seaborn es una librería de *Python* de código abierto realizada sobre *Matplotlib*, proporcionándole una interfaz de alto nivel. Esta como muchas otras librerías se utiliza para la visualización y el análisis exploratorio de datos, integrándose estrechamente con *Pandas* y estructuras de datos como los *DataFrames*.

Utilizando un conjunto de datos y un gráfico en específico como parámetros de entrada, *Seaborn* es capaz de realizar automáticamente un mapeado de los valores de los datos y asignarlos a atributos visuales, como color, tamaño, estilo. Aunque se generen automáticamente, los gráficos generados son fácilmente personalizables.

Esta librería puede ser utilizada durante todo el ciclo de vida de un proyecto, ya que produce gráficos completos a partir de una única llamada de función con un mínimo de argumentos, facilitando la creación de prototipos y el análisis exploratorio de los datos.

2.12 Python-Weka-Wrapper3

Python-weka-wrapper3 es una herramienta que permite utilizar *WEKA* desde dentro de *Python*, desarrollada por Peter Reutemann (@fracpete) trabajador en la universidad de *Waikato*.

Esta librería, utiliza la librería *Python-Javabridge* para iniciar, comunicarse y apagar una máquina virtual de *Java* en la cual se ejecutan los procesos de *WEKA* (Diagrama 8).

Python-weka-wrapper3 nos proporciona la funcionalidad básica fuera de la interfaz gráfica del usuario de *WEKA*, permitiendo agregar también los paquetes de *WEKA* al *classpath*. Algunos gráficos están disponibles gracias a funcionalidades de *Python*.

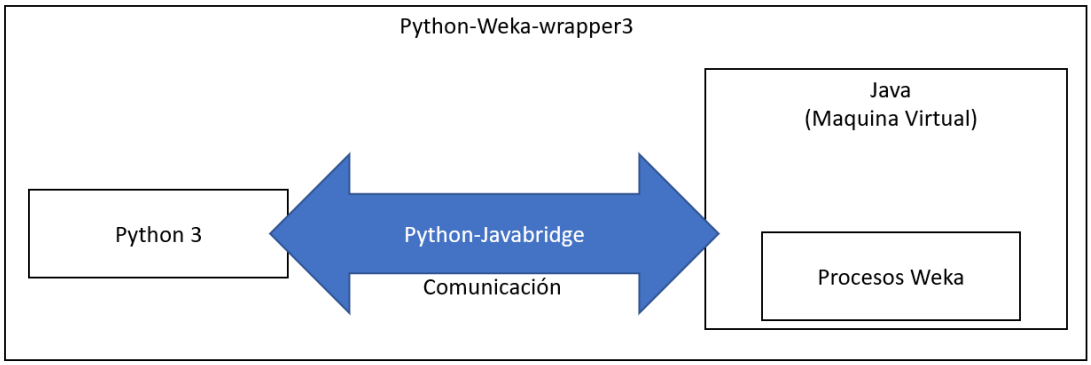


Diagrama 8: Comunicación establecida por Python-Javabridge

2.13 Problema de Clasificación y Modelos Seleccionados

En los problemas de clasificación lo que se desea es predecir el valor de una variable objetivo mediante la clasificación de la información obtenida de otras variables, estos valores a predecir suelen estar predefinidos en un conjunto de posibles valores.

Los siguientes modelos son las técnicas de aprendizaje supervisado que hemos utilizado en nuestro proyecto:

- **Árboles de Decisión:** Los árboles de decisión son uno de los métodos más usuales, siendo utilizados en distintos campos como el aprendizaje automático, el procesamiento de imágenes o la identificación de patrones. Cada nodo de un árbol de decisión representa características en una categoría a clasificar, y cada rama representa un valor o conjunto de valores que puede tomar la categoría del nodo del que nacen (Diagrama 9).

  La técnica de los árboles de decisión se basa en que cualquier camino a partir de la raíz se describe mediante la división de datos hasta llegar a un nodo hoja (solución) donde se muestra el resultado de la predicción.

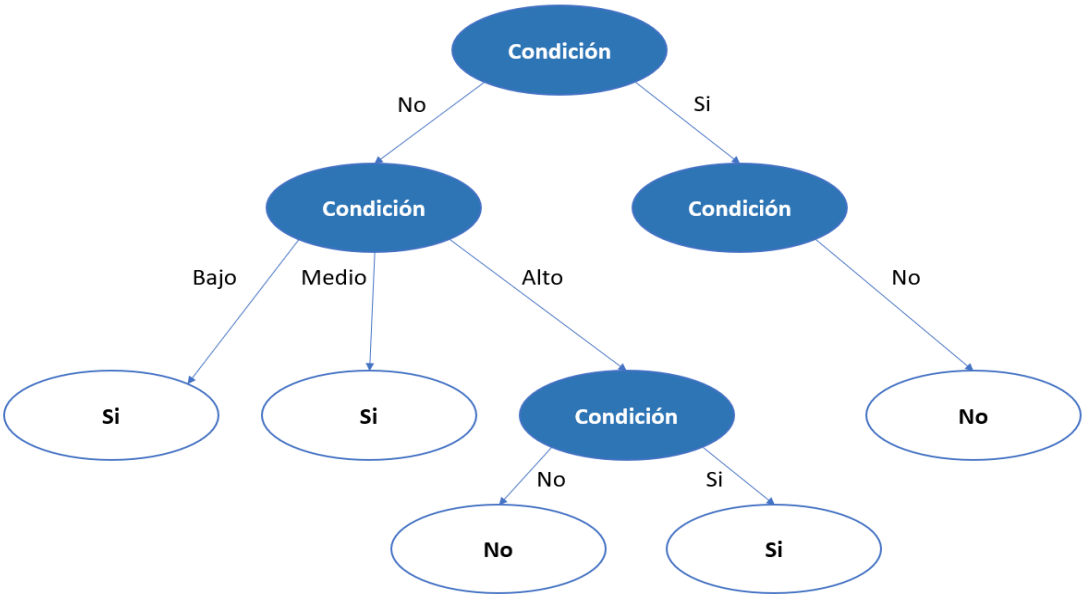


Diagrama 9: Árbol de Decisión

- **Ensembles:** Son algoritmos de aprendizaje automático que mejoran la generalización, utilizando diferentes estrategias de combinación, es decir, la combinación de algoritmos sencillos para formar uno más complejo y potente.

  En nuestro proyecto utilizamos las técnicas de *Bagging*, *Boosting*, *Random Forest*, *Gradient Boosting* y *AdaBoost*:

  - ***Bagging***: Se basa en la combinación de varios modelos de aprendizaje automático (Diagrama 10), compensando posibles errores entre sí gracias a que cada modelo se entrena con distintos subconjuntos del conjunto global de entrenamiento. El proceso de agregación consigue reducir la varianza de los modelos individuales. Los resultados de los modelos utilizados se combinan (voto por la mayoría, promedio, …) para obtener el resultado final.

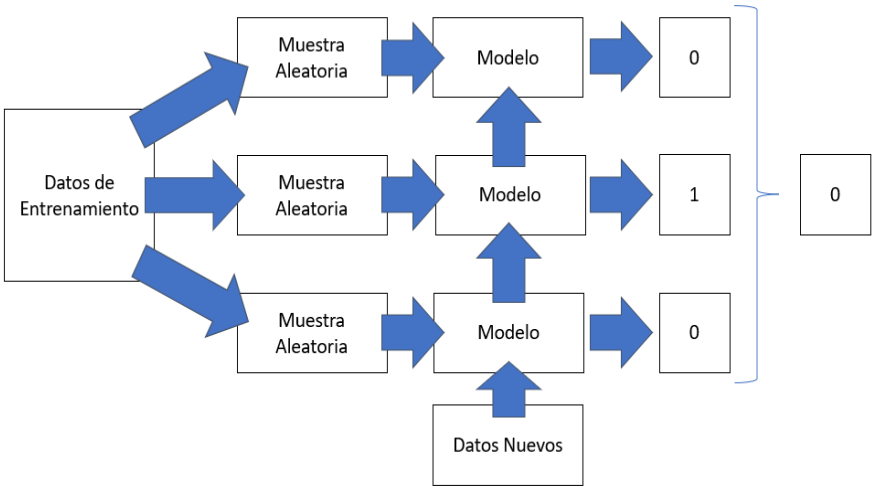


Diagrama 10: Funcionamiento de Bagging

- ***Boosting***: Es un mecanismo secuencial que se basa en que cada modelo va a ir mejorando los errores del modelo anterior (Diagrama 11), es decir, el primer modelo aprenderá las relaciones entre los datos de entrada y los de salida, aunque seguramente cometerá fallos, el siguiente modelo tratará de reducir estos errores, aunque a cambio puede cometer otros diferentes. Esto se puede lograr dándole más peso a las muestras mal clasificadas y menos peso a las que han sido clasificadas correctamente. En nuestro proyecto utilizaremos específicamente la técnica *AdaBoost*, donde la única diferencia con la técnica de Boosting original es que en *AdaBoost* se utiliza el subconjunto completo de entrenamiento para cada clasificador.

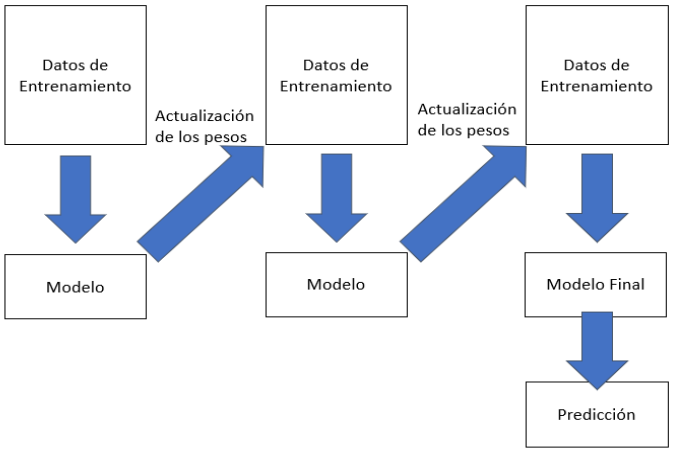


Diagrama 11: Funcionamiento de Boosting

- ***Random Forest***: Es un clasificador que se basa en el uso de un gran número de árboles de decisión que actúan como un ensemble (Diagrama 12). Cada uno de los árboles de decisión realiza una predicción y el resultado que más veces ha sido predicho sería la solución predicha por el modelo global.

  La diferencia entre *Random Forest* y *Bagging* reside en que los árboles se construyen de forma pseudoaleatoria. En particular, cada vez que se va a particionar el conjunto de datos recibido en un nodo, no se consideran todas las variables, si no un subconjunto de ellas. La selección de las posibles variables candidatas no es fija, si no que se realiza para cada partición. Como consecuencia, la correlación entre los árboles obtenidos es menor que en *Bagging*, lo que suele redundar en un mejor rendimiento.

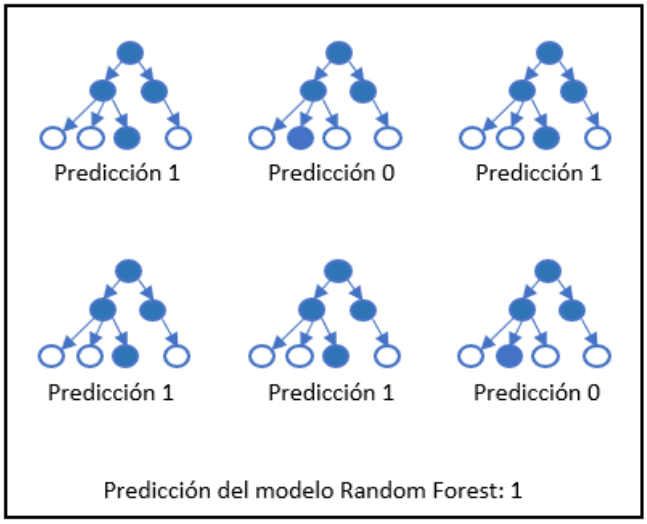


Diagrama 12: Representación gráfica Random Forest

- ***GradientBoosting***: Es la combinación de la técnica de *Boosting* y la optimización por gradiente descendiente (Diagrama 13). En este algoritmo, en lugar de aprender los modelos para que acierten donde fallaban los anteriores como se hace *Boosting*, lo que se busca es reducir el error (función de coste) cometido, usando la técnica del gradiente descendiente, para ello se parte de un modelo base y se van construyendo modelos secuencialmente para minimizar este error.

  Esta técnica requiere que la función de coste sea diferenciable, pero actualmente puede considerarse como el estado del arte en clasificación automática.

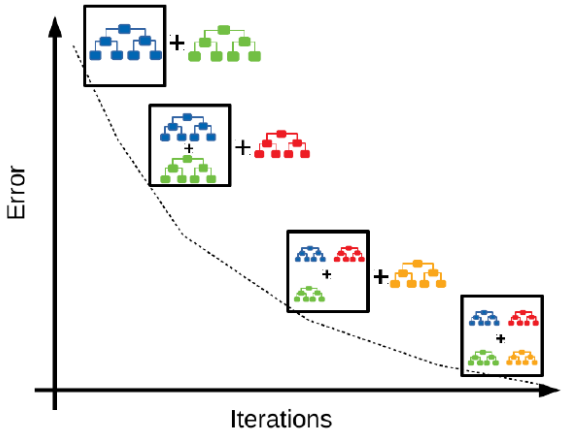


Diagrama 13: Representación gráfica de Gradient Boosting donde se puede observar cómo cuantas más iteraciones hace el modelo, más clasificadores se agregan y mejor es el resultado.

En nuestro proyecto también utilizaremos el caso particular *Histogram Based Gradient Boosting* en el que se utiliza *binning* (histogramas) para reducir la carga computacional (Diagrama 14), y *XGBoost*, el cual es una implementación de *Gradient Boosting* diseñado para obtener una gran velocidad y rendimiento. *XGBoost* es una librería centrada en *Gradient Boosting* optimizada para ser más eficiente, flexible y portátil. Incorpora algoritmos de aprendizaje automático bajo el marco de *Gradient Boosting*.

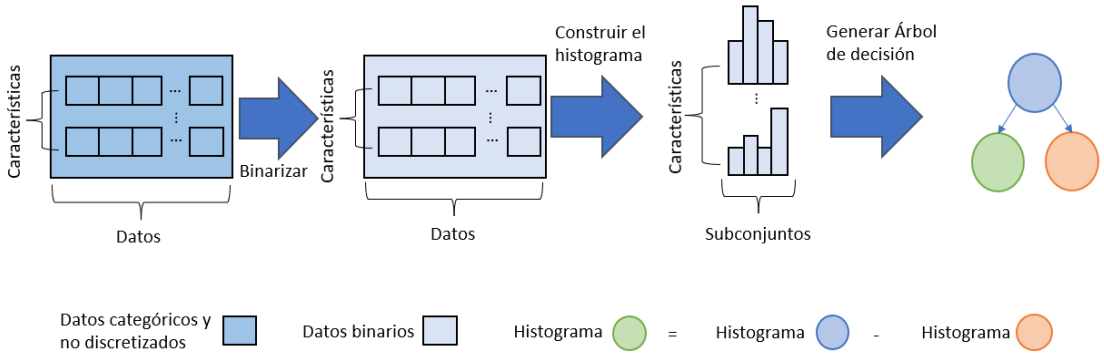


Diagrama 14: Descripción gráfica del funcionamiento del Histogram Gradient Boosting

- Arboles de Decisión Alternativos: En nuestro proyecto vamos a utilizar los *Alternating Decision Trees* (*ADTrees*) o Arboles de Decisión Alternativos (Diagrama 15), técnica utilizada por los expertos que nos facilitaron la base de datos y algunos modelos de prueba.

  Los árboles de decisión alternativos proporcionan el poder predictivo de conjuntos de árboles de decisión en un único árbol.

  Estos árboles son una variante de los árboles de decisión de opciones, es decir arboles de decisión aumentados con nodos de predicción y crecidos utilizando *Boosting*.

  Estos árboles son similares a los árboles de decisión tradicionales excepto porque pueden contener nodos de predicción además de nodos de decisión y nodos hoja, donde se representa el resultado de la predicción. Cada nodo de decisión implica una división, mientras que cada nodo de predicción implica un numero de valor real. Un nodo de decisión divide el conjunto de instancias de entrenamiento en dos partes, donde cada una pertenece a un nodo de predicción. Una instancia recorre el árbol definiendo un conjunto de caminos desde la raíz hasta algún nodo hoja, la clasificación de dicha instancia es el signo de la suma de los valores de predicción a lo largo de la ruta definida por dicha instancia, esta suma puede ser interpretada como una medida de confianza. Por ejemplo, para el árbol del diagrama, la instancia (Edad, Ganancia) = (25, 1350) la clasificación sería signo(0.5 + 0.3 – 0.5 – 0.2) = signo(0.1) = +1. Los nodos predicción de esta instancia son los sombreados.

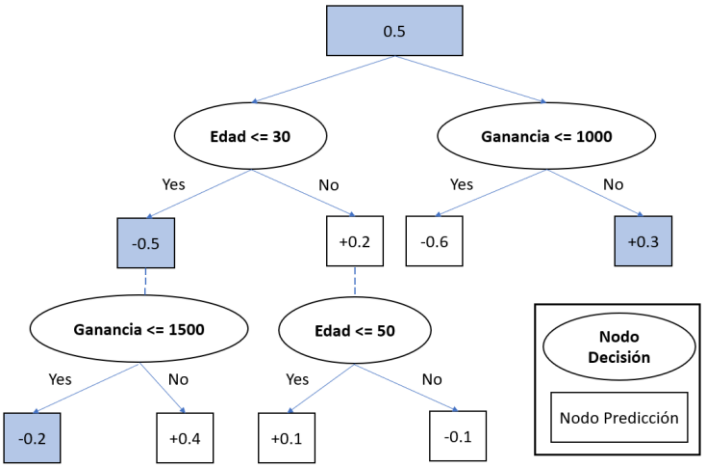


Diagrama 15: Ejemplo de Alternating Decision Tree para la instancia (Edad, Ganancia) = (25, 1350)

2.14 Métricas de Evaluación

Las métricas de evaluación se utilizan para cuantificar el rendimiento de los modelos entrenados y así poder evaluarlos. Esto es importante para poder comparar distintos modelos y poder seleccionar los mejores según lo que estemos buscando.

La forma más intuitiva de evaluar un modelo es comprobando su porcentaje de acierto en la clasificación de los datos del subconjunto de test, aunque este valor sea alto no implica que este modelo sea mejor que otros necesariamente.

Las matrices de confusión (Tabla 1), muy utilizadas en nuestro proyecto, son una herramienta que muestra gráficamente el desempeño de un clasificador explicando la distribución de las predicciones.

		Valores de Predicciones	
		0	**1**
Valores Reales	**0**	Verdaderos Negativos (VN)	Falsos Positivos (FP)
	1	Falsos Negativos (FN)	Verdaderos Positivos (VP)

Tabla 1: Representación de una matriz de confusión

- Los Verdaderos Negativos (VN) son los elementos pertenecientes a la clase negativa que han sido correctamente clasificados.
- Los Falsos Negativos (FN) son los valores pertenecientes a la clase negativa que han sido clasificados como positivos.
- Los Verdaderos Positivos (VP) son valores pertenecientes a la clase positiva que han sido clasificados como positivos.
- Los Falsos Positivos (FP) son los elementos de la clase positiva que han sido mal clasificados.

La diagonal principal muestra las predicciones realizadas correctamente, mientras que la otra diagonal muestra los errores cometidos por el clasificador. No debemos tener en cuenta únicamente estos valores para valorar positivamente un modelo, ya que, en problemas de clasificación con clases desbalanceadas, tener un alto porcentaje de acierto no implica el buen funcionamiento del modelo. La matriz de confusión no es una medida de rendimiento por si sola pero las siguientes métricas que veremos se basan en los datos que contiene.

- Exactitud o Accuracy: Es la relación entre predicciones correctas y el número total de predicciones, es decir la tasa de acierto del clasificador.

  Esta métrica tiene la ventaja de que es fácilmente interpretable, sin embargo, solo es buena para modelos cuyas clases están relativamente balanceadas ya que tiene limitaciones en clases desbalanceadas.

$$AC = \frac{VP + VN}{VP + VN + FN + FP}$$

- Precisión o Precision: La precisión mide el nivel de calidad del modelo, representando el porcentaje de verdaderos positivos.

  Esta métrica es muy interesante cuando los falsos positivos tienen un coste elevado.

$$P = \frac{VP}{VP + FP}$$

- Sensibilidad o Recall: Es la proporción de positivos que son correctamente clasificados, es decir, la capacidad del modelo de detectar una condición.

$$R = \frac{VP}{VP + FN}$$

- $\boldsymbol{F_\beta - score}$: Esta es una métrica muy utilizada ya que une la precisión y la sensibilidad en una única métrica. Esta se suele utilizar en casos de clases desbalanceadas y donde el coste de los falsos positivos y los falsos negativos es diferente.

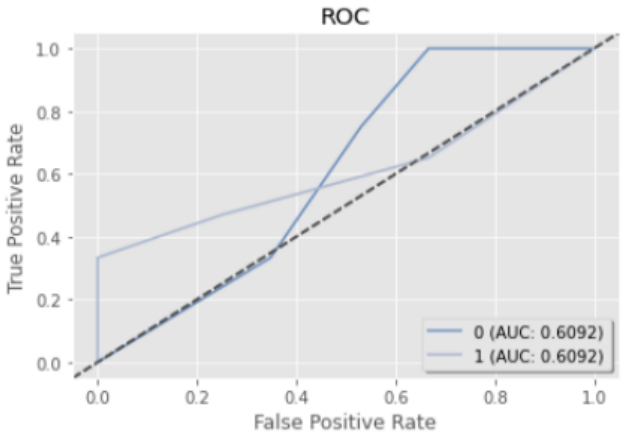
  El valor de β depende de a qué queremos dar más importancia, si es mayor a 1 nos interesa más la sensibilidad, si es menor significa que nos interesa más la precisión, si es uno entonces es que nos interesan por igual. En nuestro caso utilizamos F1-score, es decir, nos interesan por igual.

$$F_\beta = (1 - \beta^2) \frac{precision * recall}{(\beta^2 * precision) + recall}$$

- Curva *ROC*: Una curva *ROC* es un gráfico que muestra el rendimiento de un modelo de clasificación, donde se relaciona la sensibilidad con el porcentaje de falsos positivos. Esta es una buena métrica ya que, si con nuestro modelo aumentamos la sensibilidad, este clasificará más datos como positivos y por esto aumentará también el número de falsos positivos. El porcentaje de falsos positivos se calcula como:

$$PFP = \frac{FP}{FP + VN}$$

  La diagonal que va del eje [0,0] al [1,1] representa un mal clasificador, es decir, aquel que no hace una mejor clasificación a la que podría hacerse de manera aleatoria o constante. La curva *ROC* de un buen clasificador debe alejarse de esta diagonal acercándose lo máximo posible al eje [0,1] aumentando así en gran medida el área bajo la curva (*AUC*). Sus valores suelen tomar desde el 0.5 al 1 por lo que cuanto mayor sea el *AUC*, mejor será el modelo.



Gráfica 7: Ejemplo de curva ROC

Todas las métricas utilizadas emplean *'macro'* para el parámetro *average*, ya que esta métrica es insensible al desequilibrio de las clases y trata a todas por igual, razón por la cual se obtienen unas puntuaciones relativamente bajas, aproximadamente un 50% de acierto.

En muchos casos esto es lo preferible, ya que si tratamos con enfermedades (en este caso lesiones de jugadores de fútbol sala), y éstas aparecen en un número muy reducido de ocasiones, no resulta útil usar un clasificador que siempre predice que el jugador está sano pese a obtener una precisión del 99%. Lo que se pretende, por tanto, es dar mayor importancia a la clase "enfermo" y poder hacer una predicción más exacta para la clase minoritaria, aunque deba bajar la precisión del clasificador.

2.15 Metodología SCRUM

2.15.1 Definición

SCRUM es de las metodologías del paradigma *agile* más conocidas. Se basa en aspectos como la flexibilidad, el factor humano, la colaboración y el desarrollo iterativo. La metodología *SCRUM* permite a los equipos autoorganizarse mientras aborda un problema, esta metodología es principalmente utilizada por equipos de desarrollo software, aunque puede ser extrapolable a otros ámbitos de trabajo en equipo. *SCRUM* incluye un conjunto de funcionalidades, herramienta que ayudan de forma coordinada a los equipos a organizar sus proyectos y trabajo. El trabajo se divide en etapas llamadas Sprints con una duración aproximada de un mes, donde el equipo desarrolla unas funcionalidades o mejora errores, con reuniones diarias y una reunión final para mostrar los resultados del *Sprint*.

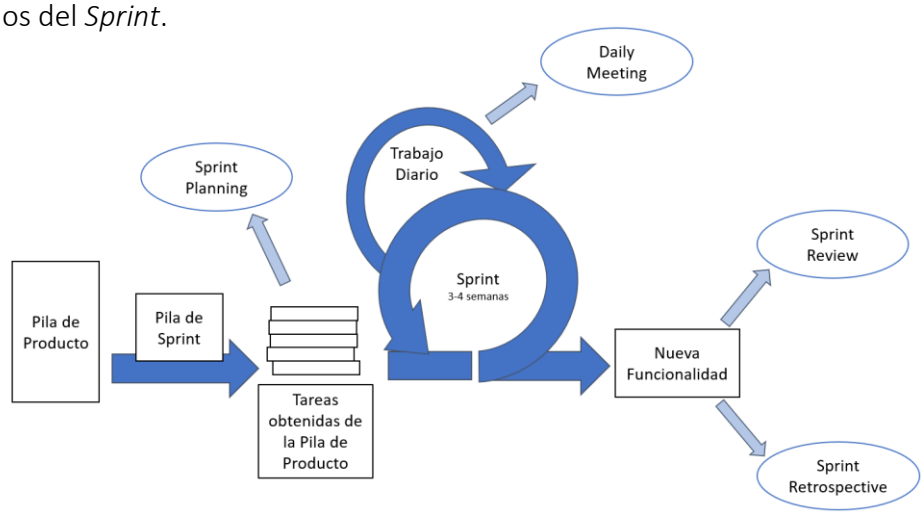


Diagrama 16: Ciclo de trabajo y eventos principales de la

2.15.2 Ciclo de Trabajo

El ciclo de trabajo de la metodología *SCRUM* se basa en 4 pasos:

1. Toma de requisitos al cliente, para cada requisito principal se crea un bloque de trabajo, llamado historia de usuario.
2. El cliente ordena los bloques de trabajo en una pila de producto según la prioridad.
3. El equipo de trabajo toma un conjunto de historias de usuario con el que trabajan durante una etapa o *Sprint*.
4. Una vez terminado el Sprint, se entrega al cliente los resultados del trabajo y se vuelve al punto 2 hasta terminar la pila de producto.

2.15.3 Eventos Principales

La metodología SCRUM tiene cuatro eventos principales (Diagrama 16):

- Sprint Planning: Reunión de todo el equipo donde se especifican los problemas y tareas que se van a abordar durante el Sprint, la reunión tiene una duración máxima de 4 horas.
- Daily Meeting: Reuniones diarias durante todo el Sprint con una duración aproximada de 15 minutos donde se hablan las tareas que se hicieron el día anterior y lo que se quiere hacer durante el día de la reunión, también se puede comentar si hay algún inconveniente para realizar algo y solicitar ayuda del equipo.
- Sprint Review: Reunión que se hace al final de cada Sprint con una duración máxima de 4 horas, y es a la única a la que puede ir el cliente. En esta reunión se presenta lo desarrollado durante el Sprint al cliente, este valida los cambios realizados y da su feedback.
- Sprint Retrospective: Es el ultimo evento de SCRUM y es una reunión en la que se hace una evaluación de como se ha trabajado durante el Sprint, permitiendo hacer mejoras de cara al próximo Sprint.

Capítulo 3

Metodología y Desarrollo

3.1 Introducción

Para el desarrollo del proyecto se ha utilizado la metodología *SCRUM*. Los requisitos y necesidades para completar este proyecto son:

- Estudio de librerías y datos a utilizar en el proyecto
- Preparación y representación de los datos
- Preprocesamiento de los datos
- Modelado
- Evaluación
- Explicación de los modelos con *SHAP*

Una vez obtenidos los requisitos necesarios, organizamos estos por prioridades y realizamos una planificación base para cada uno de ellos con las tareas a desarrollar (Tabla 2).

Una vez se han planificado todos los *Sprints* se comienza con el desarrollo y ejecución de todas las tareas, tras cada tarea se revisaban los resultados buscando posibles mejoras hasta poder dar por finalizado el sprint. Al terminar un sprint hacemos una retrospectiva del proyecto para ver las cosas realizadas correctamente y las que no han salido bien, tras esto se comienza el próximo *Sprint* inmediatamente.

Fase	Duración	Dificultad
Requisito 1: Comprensión de los datos y las librerías a utilizar.	1 Sprint	Ninguna
Requisito 2: Preparación y análisis exploratorio de los datos.	1 Sprint	Función para transformar archivos y tipos de datos
Requisito 3: Preprocesamiento de los datos	2 Sprint	Encontrar los mecanismos necesarios de preprocesamiento para nuestros datos.
Requisito 4: Modelado.	2 Sprints	Problemas de implementación.
Requisito 5: Evaluación.	1 Sprint	Aparición de resultados inesperados
Requisito 6: Explicación de Modelos con *SHAP*.	2 Sprints	Problemas de implementación.

Tabla 2: Desglose de los requisitos del proyecto con su duración estimada y dificultades a la hora de su elaboración

Es importante destacar que, las gráficas y datos mostrados durante el desarrollo de este capítulo corresponden a la "*Base de datos modelos 1 y 2.arff*", facilitada por los expertos, al ser de un tamaño más reducido.

Esta y las demás bases de datos serán detalladas junto a los resultados de los modelos realizados en las cuatro libretas del proyecto en el siguiente capítulo.

3.2 Requisito 1: Comprensión de los datos y librerías a utilizar

El objetivo de este requisito es estudiar en profundidad los datos y librerías que se van a utilizar, con el fin de familiarizarnos con ellos y tener un contacto con el problema real.
También para completar este requisito creamos la máquina virtual y el entorno sobre el que trabajamos durante todo el proyecto, el cual se detallara en el siguiente capitulo.

3.2.1 Planificación

- Creación de la máquina virtual con la distribución de *Linux Ubuntu*.
- Creación del entorno virtual para la instalación de *Python* y todas las librerías necesarias, *Java*, *Jupyter* para el desarrollo del proyecto y demás librerías y funcionalidades necesarias.
- Estudio de las librerías y aplicaciones utilizadas leyendo documentos dedicados a estas, entrando en foros, y viendo explicaciones sobre sus funcionalidades.
- Por último, para representar todo lo aprendido finalizamos este requisito con el desarrollo de una libreta de *Jupyter* explicativa donde utilizamos muchas de las funcionalidades de las librerías que hemos analizado y estudiado.

3.2.2 Creación de la máquina virtual y del entorno para el desarrollo del proyecto

Esta tarea inicial se realizó sin problemas ya que *Linux* al ser un sistema operativo de código abierto tiene una gran variedad de distribuciones libres y gratuitas para descargar, nos decantamos por *Ubuntu* 20.04, dentro de esta realizamos un entorno virtual donde poder encapsular todo el proyecto y poder controlar las librerías de *Python* instaladas y todas sus versiones para que no hubiera incompatibilidades.
A continuación, se instala la aplicación *Jupyter*, para el desarrollo del proyecto utilizando las libretas, entorno muy cómodo para programar pudiendo combinar código y texto.

3.2.3 Estudio de las librerías utilizadas

Antes de empezar con el proyecto, se realizó un estudio de las librerías *SHAP* y *Python-Weka-Wrapper3* para entender su funcionamiento en profundidad, desde la lectura de la documentación que ofrecen estas librerías en sus páginas web, a la visualización de videos o ver foros sobre estas como *Stack Overflow* o grupos específicos en *Google Groups*

3.2.4 Realización de una libreta conjunta de las librerías con datos de prueba.

Como tarea final de este requisito se realiza una libreta de *Jupyter* para explicar y desarrollar las librerías que vamos a utilizar para el conjunto de datos seleccionado. En esta libreta utilizamos la base de datos de demostración *Iris*, ya que es una base de datos para clasificación, aunque este hecha para principiantes, pudimos desarrollar con ella una gran variedad de funcionalidades que nos brindaban las distintas librerías que estudiamos, funciones de carga, visualización, análisis de los datos, preprocesamiento y transformación de estos, creación de modelos, predicción y visualización de los resultados.

También nos centramos mucho en la librería *SHAP* utilizada para la explicabilidad de los modelos ya que es la primera vez que trabajo con algo relacionado con este tema.

3.2.5 Retrospectiva

Uno de los problemas que se puede comentar en este requisito pese a que no apareciera al inicio del proyecto es que antes de encapsular el entorno del proyecto en una carpeta llamada "*TFG_env*" estaba todo sin un control de cambios y hubo errores de versiones en los que la solución fue la eliminación de la máquina virtual y tener que crear una nueva, para solucionar esto creamos un entorno encapsulado dentro de la máquina virtual para tener todo controlado y que no se volvieran a generar problemas de versiones o compatibilidades.

3.3 Requisito 2: Preparación y Análisis Exploratorio de los Datos

Para este requisito nos centraremos sobre todo en analizar las características de los datos, estudiando las distintas variables, tanto numéricas como cualitativas, viendo los valores que toman y las relaciones que pueden tener entre ellas. Para facilitar la comprensión de la información utilizaremos librerías como *Seaborn*, *Matplotlib*, *Pandas* y *NumPy* y así representarla gráficamente y observar todo de una forma más sencilla y detallada.

3.3.1 Planificación

- División de las bases de datos y algoritmos con los que vamos a trabajar en diferentes libretas.
- Carga de los datos y transformación de estos a un *DataFrame* para poder ser utilizados por mecanismos de *Python*.
- Comprobación de que no hay errores en la información almacenada.
- Análisis de la distribución de las variables numéricas, cualitativas y de la variable objetivo.

3.3.2 División del trabajo a realizar en distintas libretas

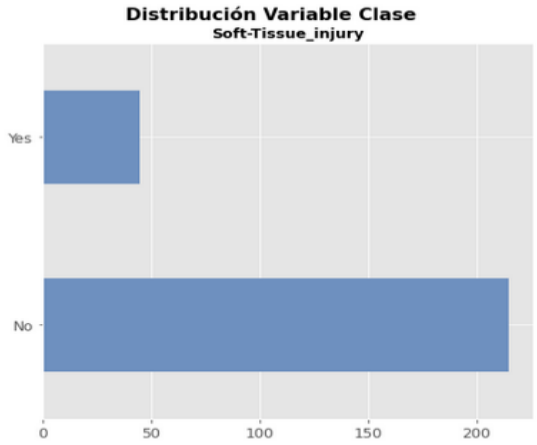
Antes de comenzar con el código se decidió dividirlo en cuatro libretas de *Jupyter* diferentes, una para cada base de datos las cuales se detallarán en el capítulo de experimentos y resultados, siendo las tres primeras donde utilizamos los distintos modelos entregados por los expertos y la cuarta libreta donde aplicamos los modelos de *Scikit-Learn* y todas las técnicas explicadas.

3.3.3 Carga transformación de los archivos donde son almacenados los datos

Inicialmente cargamos la base de datos a partir de un fichero *ARFF*, ya que son el tipo de fichero normalmente utilizado por *WEKA*, tras esto lo transformamos en un fichero *CSV*, para así poder cargarlo fácilmente en un *DataFrame* de *Pandas* y poder trabajar con los datos de una forma más sencilla, rápida y adecuada.

3.3.4 Análisis de la variable objetivo

Empezamos analizando la distribución de nuestra variable objetivo con un gráfico de barras (Gráfica 8) con el cual podemos observar un claro desbalanceo.



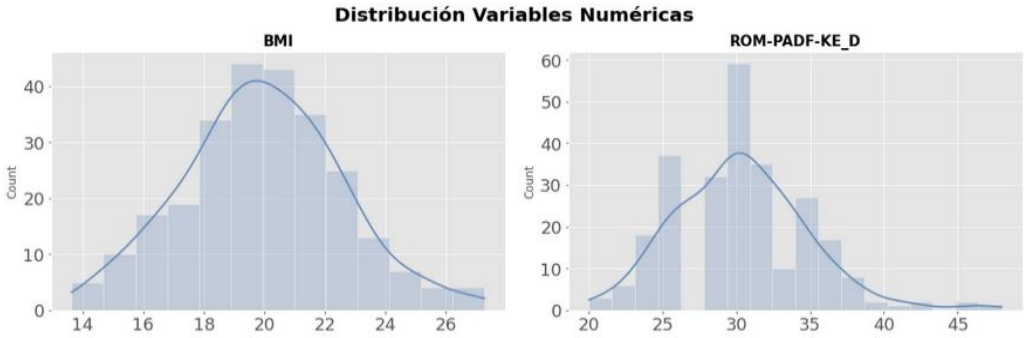
Gráfica 8: Distribución de la variable clase, como se puede observar claramente desbalanceada.

Tendremos por tanto que utilizar técnicas para tratar el desbalanceo.

3.3.5 Análisis de las variables numéricas

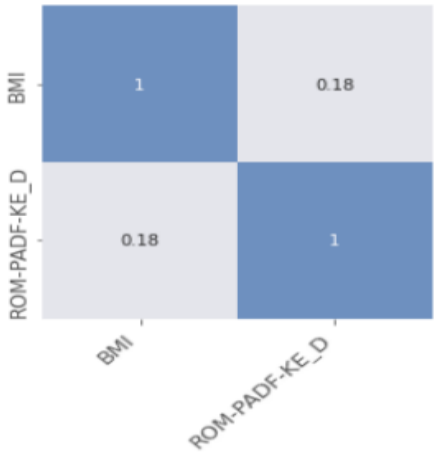
Comenzando por comprobar si todas las variables numéricas deben serlo necesariamente, ya que, si una variable toma pocos valores, utilizando un límite de 4 posibilidades, esta debería ser tratada como categórica y realizaríamos su transformación.

Tras esto observamos la distribución de nuestras variables numéricas (Gráfica 9) y vemos que casi ninguna sigue una distribución normal.



Gráfica 9: Distribución de las variables numéricas

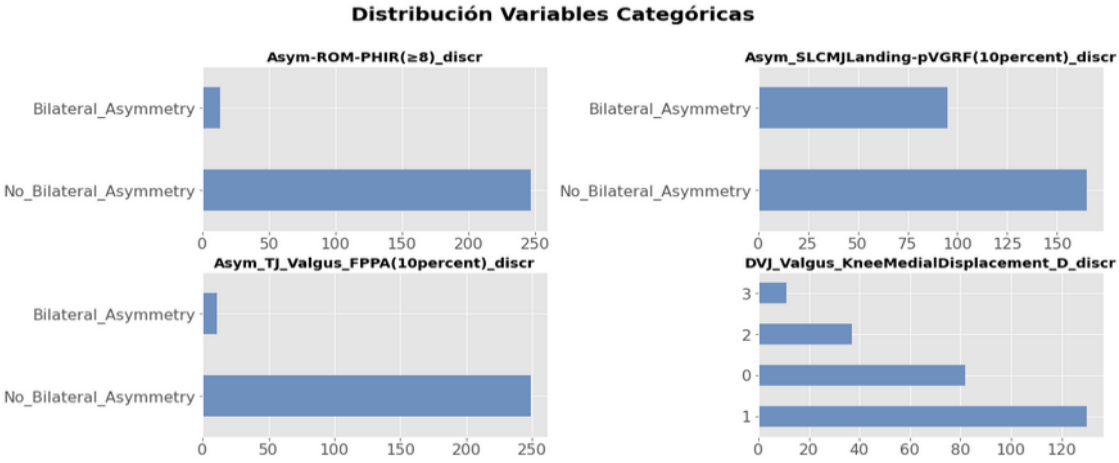
Tras esto comprobamos la correlación de las variables numéricas, ya que existen modelos de aprendizaje automático que se ven perjudicados si incorporan variables predictoras altamente correlacionadas, por esto es importante estudiar el grado de correlación de las variables disponibles (Gráfica 10).



Gráfica 10: Mapa de calor explicativo sobre la correlación de las variables numéricas

3.3.6 Análisis de las variables cualitativas

También realizamos un análisis sobre la distribución de las variables cualitativas con gráficos de barras (Gráfica 11).

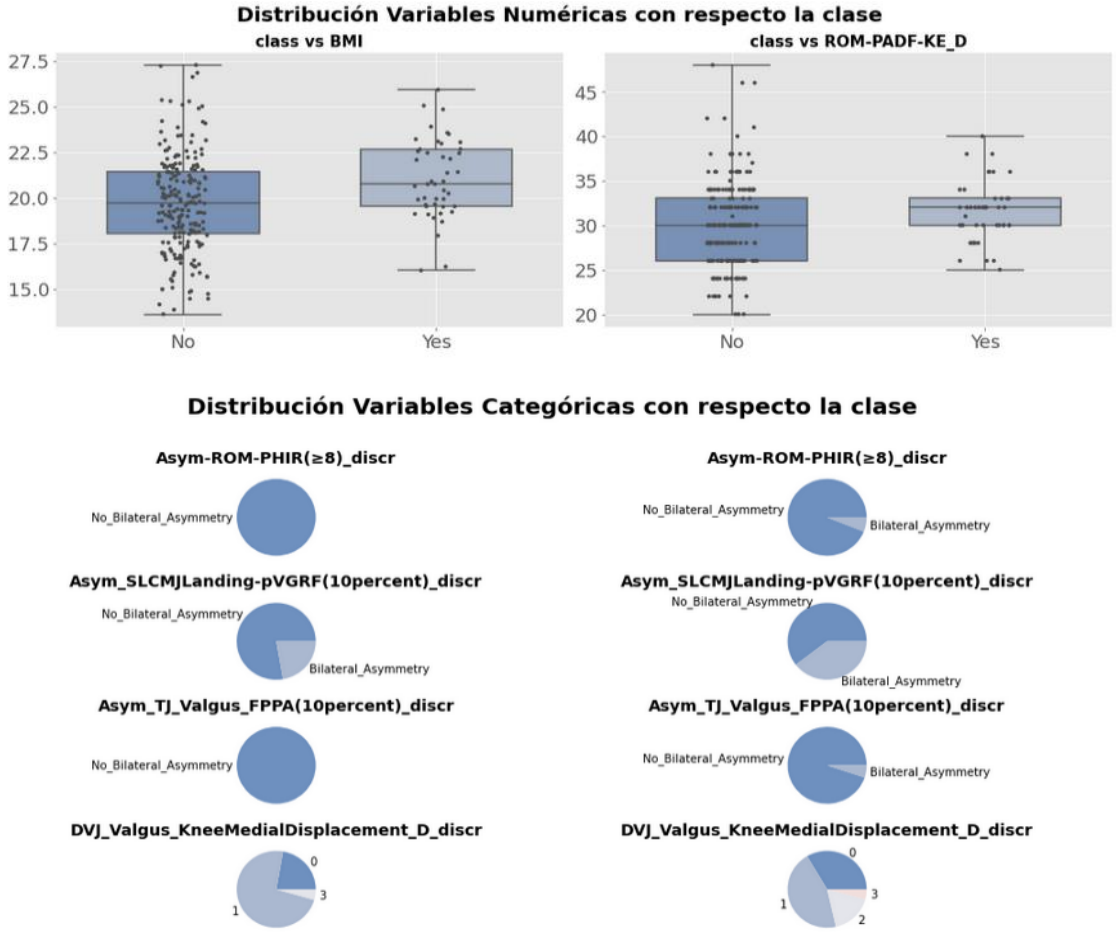


Gráfica 11: Distribución de las variables cualitativas con gráficos de barras

Podemos observar que estas variables predictoras también están desbalanceadas, esto puede provocar que, durante la validación cruzada, algunas particiones no contengan ninguna observación de la opción minoritaria (provocando que la varianza sea cero), lo que puede dar lugar a errores.

3.3.7 Análisis de las distribuciones de las variables numéricas y categóricas con respecto a la clase

También representamos la distribución de las variables con respecto a la clase (Gráfica 12), es decir cómo están distribuidas las variables tanto numéricas como categóricas para cada valor de la variable clase, en este caso *Yes* o *No*.



Gráfica 12: Grafica de franjas junto un diagrama de cajas para ver la distribución de las variables numéricas con respecto a la clase, y un gráfico circular para ver la distribución de las variables categóricas con respecto a la clase

3.3.8 Retrospectiva

Este requisito se cumplió sin mucho problema, lo que necesitó más tiempo para su correcto funcionamiento fue la función para la transformación de archivos *ARFF* a *CSV*. Las funciones para la representación gráfica de las variables fueron fácilmente implementadas gracias a la gran cantidad de documentación que hay sobre librerías de representación de datos. Como conclusiones de este requisito beneficiosas para los próximos, se puede destacar la observación del desbalanceo de las variables con el que tendremos que trabajar, también se ha comprobado que no existen valores desconocidos así que no va a ser necesaria la imputación de valores en nuestro proceso de preprocesamiento.

3.4 Requisito 3: Preprocesamiento de los datos

Tras realizar el análisis exploratorio de los datos, realizamos una etapa de preprocesado y limpieza de estos antes de utilizarlos como entrada para nuestros modelos.

La etapa de preprocesado engloba todas las transformaciones que se pueden realizar sobre una base de datos para lograr que sean interpretados por los distintos algoritmos utilizados de la forma más eficiente posible. Todos estos mecanismos deben ser entrenados y aprendidos con las observaciones de entrenamiento para luego aplicarse a los conjuntos de entrenamiento y de test, ya que ninguna de las observaciones del conjunto de datos de test debe formar parte en el ajuste de los modelos.

3.4.1 Planificación

Con la planificación de este requisito se decidió las técnicas de preprocesado que se iban a utilizar. Estas técnicas son las siguientes:

- Binarización de las variables cualitativas
- Estandarización y escalado de las variables numéricas
- Imputación de valores desconocidos
- Exclusión de variables con varianza cercana a cero.

3.4.2 Binarización de las variables cualitativas

La Binarización de las variables cualitativas lo hemos realizado gracias a la función "*OneHotEncoder()*" de la librería *Scikit-Learn*. Esta función recibe como entrada las columnas categóricas que se quieren binarizar, las cuales hemos seleccionado de nuestra base de datos con la función "*select_dtypes(include = ['object', 'category'])*".

La binarización se basa en crear nuevas variables dummy o indicadoras con cada una de las posibilidades de las variables cualitativas (Diagrama 17), es decir, si tenemos la variable "*EC*" (Estado Civil) con las posibilidades "*Casado*", "*Soltero*", "*Viudo*", esta se convertirá en tres nuevas variables, "*EC_Casado*", "*EC_Soltero*", "*EC_Viudo*", las cuales tomaran valores 0 o 1, siendo 1 la que coincide con la observación original.

Con la función "*OneHotEncoder()*" podemos pasarle como entrada también el parámetro "*drop='first'*", gracias a esto se elimina la primera categoría para evitar redundancias. Por ejemplo, con el caso anterior no es necesario tener las tres variables, si "*EC_Casado*" y "*EC_Soltero*" son 0 entonces "*EC_Viudo*" es necesariamente 1. Si "*EC_Casado*" o "*EC_Soltero*" toman el valor 1, entonces "*EC_Viudo*" tomará el valor 0. Esto es importante para modelos que son afectados si las variables predictoras están perfectamente correlacionadas.

En ciertas ocasiones puede ocurrir que, en los datos de test aparezcan nuevos niveles que no estaban en los datos de entrenamiento. Esto se puede solucionar utilizando "*handle_unknown = 'ignore'* " como parámetro de entrada de la función "*OneHotEncoder()*".

id	EC
0	Casado
1	Soltero
2	Viudo
3	Soltero

One Hot Encoding

Id	EC_Casado	EC_Soltero	EC_Viudo
0	1	0	0
1	0	1	0
2	0	0	1
3	0	1	0

Diagrama 17: Transformación con OneHotEncoder()

3.4.3 Estandarización y escalado de variables numéricas

Cuando las variables predictoras son numéricas, la magnitud de su varianza y la escala en la que se miden pueden influir en gran medida en el modelo. Muchos modelos de aprendizaje automático son sensibles a esto, por lo que, si no se regula, las variables predictoras que tengan mayor varianza y/o que se midan en una mayor escala, dominarán el modelo, aunque no sean las variables que tengan mayor relación con la variable objetivo. Para corregir esto se pueden realizar dos estrategias:

- Centrado: Consiste en restar a cada valor la media de la variable predictora a la que pertenece, es decir si la información esta almacenada en un *DataFrame*, a cada valor se le resta la media de la columna a la que pertenece. Gracias a esto, todos los valores tendrán una media de cero, es decir, que los valores se centran en torno al origen.
  Esta técnica se puede implementar con la función "*StandardScaler(with_std=False)*".
- Normalización: La normalización o estandarización se basa en transformar los datos de forma que todas las variables predictoras estén aproximadamente en el mismo intervalo de valores. Esto se puede hacer por ejemplo con los dos siguientes procedimientos:
  - *Z-score*: dividir cada predictor entre su desviación típica después de haber sido centrado, de esta forma los datos tendrán una distribución normal de media 0.

$$\boldsymbol{z = \frac{x - \mu}{\sigma}}$$

  - *Max-min*: transformar los datos de forma que estén dentro del rango [*X_min, X_max*], aunque normalmente se utiliza el rango [0, 1]

$$\boldsymbol{X = \frac{X - X_{min}}{X_{max} - X_{min}}}$$

3.4.4 Exclusión de variables con varianza próxima a cero

En el modelo no se deben incluir variables predictoras que contengan un único valor es decir que sean constantes (cero-varianza) ya que estas no aportan información. Tampoco es conveniente incluir variables predictoras con una varianza próxima a cero, es decir variables que toman pocos valores, de los cuales algunos no tienen mucha frecuencia de aparición, ya que estos se pueden convertir en variables predictoras con varianza cero cuando de dividen las muestras al utilizar técnicas como la validación cruzada.

La clase *VarianceThreshold* del módulo *sklearn.feature_selection* identifica y excluye aquellas variables predictoras cuya varianza no supera un límite.

3.4.5 Retrospectiva

El proceso que más se demoró para este requisito fue la selección de los métodos de preprocesamiento que debíamos utilizar, pero una vez se seleccionaron su implementación fue un proceso directo y sin complicaciones.

3.5 Requisito 4: Modelado

3.5.1 Planificación

- Recrear los modelos dados con *Python-Weka-Wrapper3*
- Utilizar estos modelos con diferentes técnicas para el desbalanceo.
- Mostrar y analizar los resultados

3.5.2 Modelos de los expertos (Weka)

Para las primeras libretas los modelos realizados han sido los proporcionados por los expertos. En ellas se han recreado con *Python-Weka-Wrapper3* los modelos generados con *WEKA*, para comprobar si existen mejoras de rendimiento o diferencias respecto a los originales.

Estos modelos se basan principalmente en Ensembles (*Bagging*) de Arboles de Decisión Alternativos.

Adicionalmente para la optimización de parámetros utilizan *MultiSearch Parameter Optimization*, similar a "*GridSearch()",* aunque este puede ser utilizado para la optimización de un número arbitrario de parámetros, y "*GridSearch()"* siempre necesitara un mínimo de dos parámetros a optimizar.

Su modelo también dispone de Selección de Atributos, *Attribute Selection* en forma de clasificador ("*weka.classifiers.AttributeSelectedClassifier"*), aunque también se encuentra disponible como una función de filtrado ("*weka.filters.AttributeSelection"*), para la búsqueda de las mejores variables predictoras para el modelo. El uso de Selección de Atributos como un clasificador se basa en dos pasos, primero reducción de la dimensionalidad a través de la selección de los mejores atributos y segundo la clasificación, permitiendo personalizar entre una gran variedad de métodos y clasificadores diferentes. En cambio, si se utiliza la Selección de Atributos como función de filtrado, es simplemente un filtro supervisado que se utiliza para seleccionar los mejores atributos.

Para tratar el desbalanceo utilizan técnicas como *SMOTE*, *Oversampling* o *Undersampling* en los distintos modelos con las distintas bases de datos.

Aparte de recrear los modelos proporcionados, en este trabajo también se utilizan las distintas técnicas para tratar el desbalanceo (apartado 2.4.3), con cada modelo y con cada base de datos en las tres primeras libretas.

3.5.3 Modelos de Scikit-Learn

En la cuarta libreta utilizamos una base de datos intacta, donde utilizamos las técnicas para la carga, análisis exploratorio, preprocesamiento y limpieza de los datos, en cuanto a los modelos utilizados nos decantamos por utilizar ensembles como *Bagging*, *RandomForest*, *AdaBoost*, *GradientBoosting*, *HistGradientBoosting* y *XGBoost*, todos ellos con árboles de decisión como clasificadores base, y previo a estos, también hemos analizado el comportamiento de los árboles de decisión como modelo único sin formar parte de un ensemble.

Con todos estos modelos se han seguido los siguientes pasos.

- Entrenamiento, Validación Cruzada y Evaluación de las predicciones con el modelo básico utilizando como métricas de error *ROC_AUC*, *Recall*, *Precision* y *F1_Score*. Tras esto el modelo básico es explicado con *SHAP* para ver su funcionamiento.
- Selección de los mejores parámetros para cada modelo mediante "*RandomizedSearch()"* el cual nos da una gran mejora respecto al tiempo que necesita para seleccionar los mejores parámetros frente a "*GridSearch()"*. Tras esto volvemos a entrenar los modelos con sus mejores parámetros y a evaluar las predicciones con las mismas puntuaciones anteriores, también volvemos a explicar con *SHAP* este modelo con sus mejores parámetros.
- Por último, realizamos una selección de atributos para cada modelo con la función "*SelectFromModel()*" del módulo de Scikit-Learn "*feature_selection*", con los cuales podemos conseguir en la mayoría de las ocasiones puntuaciones similares e incluso mejores a las obtenidas anteriormente con toda la base de datos, pero ahora con una selección de entre 5 y 25 atributos. Tras estos también realizamos una explicación del modelo con *SHAP* para ver el impacto de las variables predictoras seleccionadas en el modelo.

3.5.4 Retrospectiva

Este ha sido el proceso más duradero debido a problemas con la implementación de los modelos de *WEKA* entregados por los expertos con *Python-Weka-Wrapper3* debido a que algunas funcionalidades aun no estaban del todo implementadas como la capacidad de hacer la división de la base de datos en entrenamiento y test con estratificación para asegurarnos de que no haya casos en los que la clase sea univariable y pueda producir problemas. La sección de *Scikit-Learn* fue algo tediosa debido a que era repetir el mismo procedimiento con todos los modelos, pero al mismo tiempo también fue más ágil y sencilla de realizar.

3.6 Requisito 5: Evaluación

La sección de evaluación de los modelos se realizó en varias secciones para cada modelo, con el modelo genérico después de entrenarlo y validarlo con validación cruzada. Luego tras la selección de los mejores parámetros con "*RandomizedSearch()"* se evalúan otra vez. Y por último tras la selección de atributos con la función *SelectFromModel*, hacemos otra evaluación del modelo final. Hacemos estas tres evaluaciones pare comprobar una posible mejora de nuestros modelos conforme lo vamos ajustando más.

3.6.1 Métricas de Evaluación

Las métricas de evaluación seleccionadas para nuestros modelos han sido *ROC_AUC*, *Recall*, *Precision* y *F1_Score*, debido a que son las que pueden ser de mayor interés para nuestros modelos que utilizan una base de datos desbalanceada (apartado 2.14).

3.6.2 Retrospectiva

Este requisito tomó alrededor de un sprint completo a causa de la aparición de resultados inesperados, como el caso de que todas las predicciones son de la misma clase. Este caso se da en los modelos de *WEKA* cuando utilizamos "*Random Undersampling*" o "*SMOTE*" para tratar con el desbalanceo en las tres primeras libretas.

3.7 Requisito 6: Explicabilidad de los Modelos con SHAP

El último requisito para completar los objetivos de este proyecto se refiere a la explicabilidad de los modelos desarrollados en el Requisito 4, para lo cual se utiliza la librería SHAP.

3.7.1 Planificación

- Creación de un puente de unión entre los algoritmos de Weka / Python-Weka-Wrapper3 y la librería SHAP.
- Seleccionar un método de explicación óptimo para nuestros modelos.
- Selección de los diagramas más apropiados para la explicación de nuestros modelos.

3.7.2 Puente Weka – SHAP

Para el puente de unión se ha creado una clase llamada "*weka_classifier*", la cual extiende de "*BaseEstimator*" y "*ClassifierMixin*", y en ella se han definido las funcionalidades básicas de un clasificador como "*__init__*", para poder inicializarlo, "*fit*", para construir el clasificador, "*predict*" para realizar las predicciones… Todas estas funciones llaman a funciones de Python-weka-wrapper3, permitiendo así obtener la información necesaria, y por consiguiente, que las librerías de Scikit-Learn como SHAP reconozcan estos modelos.

3.7.3 Explainers seleccionados

Los métodos de explicación seleccionados (sección 2.5.2) han sido *KernelExplainer*, para los algoritmos de Weka, y *TreeExplainer*, para los algoritmos utilizados de Scikit-Learn, basados estos últimos en árboles de decisión.

3.7.4 Diagramas utilizados

- *Force Plots*
- *Waterfall Plot*
- *Dependence Plot*
- *Summary Plots*

(sección 2.5.3)

3.7.5 Retrospectiva

En el desarrollo de este requisito se han encontrado varios problemas. En primer lugar, los valores Shapley para los algoritmos de Weka debían ser calculados con el *KernelExplainer*. Para la "*Base de datos modelos 1 y 2.arff*" no hubo problemas, pero en el resto se generaba un error con el mensaje "*Kernel has died*". Para solucionar esto se trató de buscar información en diversos foros, como *Stack Overflow*, y se encontraron varias causas, así como algunas posibles soluciones:

1. La máquina virtual que ejecuta java (*jvm*) se queda sin memoria. Para solucionar esto se aumentó la memoria *heap*, es decir el espacio que se utiliza para almacenar de forma aleatoria o desorganizada la información, ya sean valores, funciones, etc. Para lograr esto se le asignó un tamaño a la memoria al inicializar la máquina virtual de java "*jvm.start(system_cp=True, packages=True, max_heap_size="10g")*". Tras hacer varias pruebas asignando distintos tamaños a la memoria no se llegó a ningún resultado válido.
2. La cantidad de instancias que se utilizan para calcular los valores *Shapley* y generar el explicador es demasiado grande. Inicialmente se realizaba con la base de datos completa, pero al no ser viable, se realizó con el subconjunto de entrenamiento, luego con el de test y finalmente con instancias individuales. Sin embargo, ninguna de estas opciones solucionó este problema.

A causa de esto solo se ha podido realizar la explicación de los modelos de *WEKA* en la primera libreta para la base de datos más reducida.

En la cuarta y última libreta, donde se desarrollaron los modelos de *Scikit-Learn*, también aparecieron varios inconvenientes. Estos eran la no implementación de los modelos "*BaggingClassifier()*" y "*AdaBoostClassifier()*" en el marco de *SHAP*, incapacitando su explicación con *TreeExplainer*. Para solucionar esto se intentó realizar su explicación con *KernelExplainer,* pero al igual que en las otras libretas, también se generaba el error "*Kernel has died*". Para solucionar este problema se trató de implementar estos clasificadores en *SHAP*, modificando el archivo "*_tree.py*" pero tras realizar un estudio exhaustivo de las posibles implementaciones que podría haber para esta situación, tampoco se dio con ninguna solución válida.

A causa de estos problemas no se ha podido dar explicabilidad a los modelos realizados en las libretas dos y tres, ni a los modelos de Scikit-Learn "*BaggingClassifier()*" y "*AdaBoostClassifier()*".

Capítulo 4

Experimentos y Resultados

4.1 Introducción

Tras haber implementado las cuatro libretas en Jupyter para las distintas bases de datos se van a probar los modelos con distintas variantes para ver los resultados y las posibles mejoras o deficiencias entre las variantes de los modelos dados.
Todas las libretas de Jupyter son ejecutadas en el mismo entorno en VirtualBox.

4.2 Características Entorno Virtual Box

La realización de todas las pruebas se ha hecho en el mismo entorno virtual, la configuración de este entorno se encuentra en la Tabla 3.

General
Nombre: Ubuntu 20.04.3 LTS Sistema operativo: Ubuntu (64-bit)
Sistema
Memoria base: 10684 MB Procesadores: 6 Orden de arranque: Disquete, Óptica, Disco duro Aceleración: VT-x/AMD-V, Paginación anidada, Paravirtualización KVM
Pantalla
Memoria de vídeo: 16 MB Controlador gráfico: VMSVGA Servidor de escritorio remoto: Inhabilitado Grabación: Inhabilitado
Almacenamiento
Controlador: IDE IDE secundario maestro: [Unidad óptica] Vacío Controlador: SATA Puerto SATA 0: Ubuntu 20.04.3 LTS.vdi (Normal, 40,00 GB)
Audio
Controlador de anfitrión: Windows DirectSound Controlador: ICH AC97
Red
Adaptador 1: Intel PRO/1000 MT Desktop (NAT)
USB
Controlador USB: OHCI Filtros de dispositivos: 0 (0 activo)
Carpetas Compartidas
Ninguno
Descripción
Ninguno

Tabla 3: Configuración de la máquina virtual

4.3 Libreta 1

4.3.1 Bases de Datos utilizadas

La base de datos utilizada, inicialmente consta de 260 muestras, con cuatro atributos categóricos, dos numéricos y la variable objetivo.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 260 entries, 0 to 259
Data columns (total 7 columns):
 #   Column                                      Non-Null Count  Dtype
---  ------                                      --------------  -----
 0   BMI                                         260 non-null    float64
 1   ROM-PADF-KE_D                               260 non-null    int64
 2   Asym-ROM-PHIR(≥8)_discr                     260 non-null    object
 3   Asym_SLCMJLanding-pVGRF(10percent)_discr    260 non-null    object
 4   Asym_TJ_Valgus_FPPA(10percent)_discr        260 non-null    object
 5   DVJ_Valgus_KneeMedialDisplacement_D_discr   260 non-null    object
 6   Soft-Tissue_injury_≥4days                   260 non-null    category
dtypes: category(1), float64(1), int64(1), object(4)
```

Figura 1: Imagen explicativa de la base de datos utilizada en la primera libreta previa al preprocesado de los datos

Tras el preprocesamiento de los datos, previamente a ser utilizados en los modelos, la base de datos resultante contiene ocho atributos numéricos y la variable objetivo. También ha sido dividida en dos subconjuntos, entrenamiento y test con 182 y 78 muestras respectivamente.

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 182 entries, 127 to 194
Data columns (total 9 columns):
 #   Column                                                  Non-Null Count  Dtype
---  ------                                                  --------------  -----
 0   Asym-ROM-PHIR(≥8)_discr_No_Bilateral_Asymmetry          182 non-null    float64
 1   Asym_SLCMJLanding-pVGRF(10percent)_discr_No_Bilateral_Asymmetry  182 non-null  float64
 2   Asym_TJ_Valgus_FPPA(10percent)_discr_No_Bilateral_Asymmetry      182 non-null  float64
 3   DVJ_Valgus_KneeMedialDisplacement_D_discr_1             182 non-null    float64
 4   DVJ_Valgus_KneeMedialDisplacement_D_discr_2             182 non-null    float64
 5   DVJ_Valgus_KneeMedialDisplacement_D_discr_3             182 non-null    float64
 6   BMI                                                     182 non-null    float64
 7   ROM-PADF-KE_D                                           182 non-null    float64
 8   Soft-Tissue_injury_≥4days                               182 non-null    category
dtypes: category(1), float64(8)
```

Figura 2: Imagen del subconjunto de entrenamiento posterior al preprocesado de la base de datos original

4.3.2 Resultados

Resultados WEKA

Método	Precision_Score	Recall_Score	F1_Score	ROC_AUC_Score	
Modelo Libreta 1 SMOTE	0.846	0.859	0.853	0.670	Clase 0
	0.308	0.286	0.296	0.670	Clase 1
	0.750	0.756	0.753	0.670	Media Ponderada
Modelo Libreta 1 RUS	0.821	1.00	0.901	0.691	Clase 0
	?	0.00	?	0.691	Clase 1
	?	0.821	?	0.691	Media Ponderada
Modelo Libreta 1 ROS	0.838	0.891	0.864	0.699	Clase 0
	0.300	0.214	0.250	0.699	Clase 1
	0.742	0.769	0.753	0.699	Media Ponderada
Modelo Libreta 1 RUS & ROS	0.836	0.875	0.855	0.751	Clase 0
	0.273	0.214	0.240	0.751	Clase 1
	0.735	0.756	0.745	0.751	Media Ponderada

Tabla 4: Tabla resumen de los resultados obtenidos gracias a Python-Weka-Wrapper3 en la primera libreta.

Resultados Scikit-Learn

Método	Precision_Score	Recall_Score	F1_Score	ROC_AUC_Score	Media
Modelo Libreta 1 SMOTE	0.577	0.572	0.574	0.572	0.5741
Modelo Libreta 1 RUS	0.5	0.5	0.410	0.450	0.4652
Modelo Libreta 1 ROS	0.552	0.552	0.569	0.556	0.5577
Modelo Libreta 1 RUS & ROS	0.545	0.545	0.554	0.547	0.5477

Tabla 5:Tabla resumen de los resultados obtenidos gracias a Scikit-Learn en la primera libreta.

4.3.3 Conclusiones

En esta libreta y viendo los resultados se puede comprobar que con *Scikit-Learn* pese a obtener unas peores puntuaciones que *Python-Weka-Wrapper3* se han podido observar las buenas funcionalidades que tiene, incluso con modelos externos a Scikit-Learn. Las puntuaciones son bajas debido al uso de *'macro'* para el parámetro *average* (sección 2.14).

4.4 Libretas 2 y 3

4.4.1 Bases de Datos utilizadas

Para la segunda libreta la base de datos consta de 260 muestras con 32 atributos numéricos 29 categóricos y la variable objetivo

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 260 entries, 0 to 259
Data columns (total 62 columns):
 #   Column                                          Non-Null Count  Dtype
---  ------                                          --------------  -----
 0   DVJ_Valgus_KneeMedialDisplacement_D_discr       260 non-null    int64
 1   BMI                                             260 non-null    float64
 2   AgeGroup                                        260 non-null    object
 3   ROM-PADF-KE_D                                   260 non-null    int64
 4   DVJ_Valgus_FPPA_D_discr                         260 non-null    int64
 5   TrainFrequency                                  260 non-null    int64
 6   DVJ_Valgus_FPPA_ND_discr                        260 non-null    int64
 7   Asym_SLCMJLanding-pVGRF(10percent)_discr        260 non-null    object
 8   Asym-ROM-PHIR(≥8)_discr                         260 non-null    object
 9   Asym_TJ_Valgus_FPPA(10percent)_discr            260 non-null    object
 10  TJ_Valgus_FPPA_ND_discr                         260 non-null    int64
 11  Asym-ROM-PHF-KE(≥8)_discr                       260 non-null    object
 12  TJ_Valgus_FPPA_D_discr                          260 non-null    int64
 13  Asym_SLCMJ-Height(10percent)_discr              260 non-null    object
 14  Asym_YBTpl(10percent)_discr                     260 non-null    object
 15  Position                                        260 non-null    object
 16  Asym-ROM-PADF-KE(≥8°)_discr                     260 non-null    object
 17  DVJ_Valgus_KneeMedialDisplacement_ND_discr      260 non-null    int64
 18  DVJ_Valgus_Knee-to-ankle-ratio_discr            260 non-null    object
 19  Asym-ROM-PKF(≥8)_discr                          260 non-null    object
 20  Asym-ROM-PHABD(≥8)_discr                        260 non-null    object
 21  Asym-ROM-PHF-KF(≥8)_discr                       260 non-null    object
 22  Asym-ROM-PHER(≥8)_discr                         260 non-null    object
 23  AsymYBTanterior10percentdiscr                   260 non-null    object
 24  Asym-ROM-PHABD-HF(≥8)_discr                     260 non-null    object
 25  Asym-ROM-PHE(≥8)_discr                          260 non-null    object
 26  Asym(>4cm)-DVJ_Valgus_Knee;edialDisplacement_discr  260 non-null  object
 27  Asym_SLCMJTakeOff-pVGRF(10percent)_discr        260 non-null    object
 28  Asym-ROM-PHADD(≥8)_discr                        260 non-null    object
 29  Asym-YBTcomposite(10percent)_discr              260 non-null    object
 30  Asym_SingleHop(10percent)_discr                 260 non-null    object
 31  Asym_YBTpm(10percent)_discr                     260 non-null    object
 32  Asym_DVJ_Valgus_FPPA(10percent)_discr           260 non-null    object
 33  Asym_SLCMJ-pLFT(10percent)_discr                260 non-null    object
 34  DominantLeg                                     260 non-null    object
 35  Asym-ROM-PADF-KF(≥8)_discr                      260 non-null    object
 36  ROM-PHER_ND                                     260 non-null    int64
 37  CPRDmentalskills                                260 non-null    object
 38  POMStension                                     260 non-null    int64
 39  STAI-R                                          260 non-null    float64
 40  ROM-PHER_D                                      260 non-null    int64
 41  ROM-PHIR_D                                      260 non-null    int64
 42  ROM-PADF-KF_ND                                  260 non-null    int64
 43  ROM-PADF-KF_D                                   260 non-null    int64
 44  Age_at_PHV                                      260 non-null    float64
 45  ROM-PHIR_ND                                     260 non-null    int64
 46  CPRDtcohesion                                   260 non-null    object
 47  Eperience                                       260 non-null    float64
 48  ROM-PHABD-HF_D                                  260 non-null    int64
 49  MaturityOffset                                  260 non-null    float64
 50  Weight                                          260 non-null    float64
 51  ROM-PHADD_ND                                    260 non-null    int64
 52  Height                                          260 non-null    float64
 53  ROM-PHADD_D                                     260 non-null    int64
 54  Age                                             260 non-null    float64
 55  POMSdepressio                                   260 non-null    int64
 56  ROM-PADF-KE_ND                                  260 non-null    int64
 57  POMSanger                                       260 non-null    int64
 58  YBTanterior_Dnorm                               260 non-null    float64
 59  YBTanterior_NDnorm                              260 non-null    float64
 60  POMSvigour                                      260 non-null    int64
 61  Soft-Tissue_injury_≥4days                       260 non-null    category
dtypes: category(1), float64(10), int64(22), object(29)
```

Figura 3: Base de datos original de la segunda libreta

Tras el preprocesado la base de datos resultante queda dividida en los subconjuntos de entrenamiento y test, al igual que anteriormente, con 182 y 78 muestras correspondientemente. Cada muestra de la base de datos resultante contará con 103 columnas numéricas y la variable objetivo.

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 182 entries, 127 to 194
Columns: 103 entries, DVJ_Valgus_KneeMedialDisplacement_D_discr_1 to POMSvigour
dtypes: float64(103)
```

Figura 4: Imagen del subconjunto de entrenamiento posterior al preprocesado de la base de datos original

De manera homologa, la tercera libreta la base de datos original consta de 260 muestras con 14 atributos numéricos, 28 categóricos y la variable objetivo

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 260 entries, 0 to 259
Data columns (total 43 columns):
 #   Column                                          Non-Null Count  Dtype
---  ------                                          --------------  -----
 0   DVJ_Valgus_KneeMedialDisplacement_D_discr       260 non-null    int64
 1   BMI                                             260 non-null    float64
 2   AgeGroup                                        260 non-null    object
 3   ROM-PADF-KE_D                                   260 non-null    int64
 4   DVJ_Valgus_FPPA_D_discr                         260 non-null    int64
 5   TrainFrequency                                  260 non-null    int64
 6   DVJ_Valgus_FPPA_ND_discr                        260 non-null    int64
 7   Asym_SLCMJLanding-pVGRF(10percent)_discr        260 non-null    object
 8   Asym-ROM-PHIR(≥8)_discr                         260 non-null    object
 9   Asym_TJ_Valgus_FPPA(10percent)_discr            260 non-null    object
 10  TJ_Valgus_FPPA_ND_discr                         260 non-null    int64
 11  Asym-ROM-PHF-KE(≥8)_discr                       260 non-null    object
 12  TJ_Valgus_FPPA_D_discr                          260 non-null    int64
 13  Asym_SLCMJ-Height(10percent)_discr              260 non-null    object
 14  Asym_YBTpl(10percent)_discr                     260 non-null    object
 15  Position                                        260 non-null    object
 16  Asym-ROM-PADF-KE(≥8°)_discr                     260 non-null    object
 17  DVJ_Valgus_KneeMedialDisplacement_ND_discr      260 non-null    int64
 18  DVJ_Valgus_Knee-to-ankle-ratio_discr            260 non-null    object
 19  Asym-ROM-PKF(≥8)_discr                          260 non-null    object
 20  Asym-ROM-PHABD(≥8)_discr                        260 non-null    object
 21  Asym-ROM-PHF-KF(≥8)_discr                       260 non-null    object
 22  Asym-ROM-PHER(≥8)_discr                         260 non-null    object
 23  AsymYBTanterior10percentdiscr                   260 non-null    object
 24  Asym-ROM-PHABD-HF(≥8)_discr                     260 non-null    object
 25  Asym-ROM-PHE(≥8)_discr                          260 non-null    object
 26  Asym(>4cm)-DVJ_Valgus_Knee;edialDisplacement_discr  260 non-null  object
 27  Asym_SLCMJTakeOff-pVGRF(10percent)_discr        260 non-null    object
 28  Asym-ROM-PHADD(≥8)_discr                        260 non-null    object
 29  Asym-YBTcomposite(10percent)_discr              260 non-null    object
 30  Asym_SingleHop(10percent)_discr                 260 non-null    object
 31  Asym_YBTpm(10percent)_discr                     260 non-null    object
 32  Asym_DVJ_Valgus_FPPA(10percent)_discr           260 non-null    object
 33  Asym_SLCMJ-pLFT(10percent)_discr                260 non-null    object
 34  DominantLeg                                     260 non-null    object
 35  Asym-ROM-PADF-KF(≥8)_discr                      260 non-null    object
 36  ROM-PHER_ND                                     260 non-null    int64
 37  CPRDmentalskills                                260 non-null    object
 38  POMStension                                     260 non-null    int64
 39  STAI-R                                          260 non-null    float64
 40  ROM-PHER_D                                      260 non-null    int64
 41  ROM-PHIR_D                                      260 non-null    int64
 42  Soft-Tissue_injury_≥4days                       260 non-null    category
dtypes: category(1), float64(2), int64(12), object(28)
```

Figura 5: Base de datos original de la tercera libreta

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 182 entries, 127 to 194
Data columns (total 73 columns):
 #   Column                                                                Non-Null Count  Dtype  
---  ------                                                                --------------  -----  
 0   DVJ_Valgus_KneeMedialDisplacement_D_discr_1                           182 non-null    float64
 1   DVJ_Valgus_KneeMedialDisplacement_D_discr_2                           182 non-null    float64
 2   DVJ_Valgus_KneeMedialDisplacement_D_discr_3                           182 non-null    float64
 3   AgeGroup_U13U14                                                       182 non-null    float64
 4   AgeGroup_U15U16                                                       182 non-null    float64
 5   AgeGroup_U17U19                                                       182 non-null    float64
 6   DVJ_Valgus_FPPA_D_discr_1                                             182 non-null    float64
 7   DVJ_Valgus_FPPA_D_discr_2                                             182 non-null    float64
 8   DVJ_Valgus_FPPA_D_discr_3                                             182 non-null    float64
 9   TrainFrequency_3                                                      182 non-null    float64
 10  TrainFrequency_4                                                      182 non-null    float64
 11  DVJ_Valgus_FPPA_ND_discr_1                                            182 non-null    float64
 12  DVJ_Valgus_FPPA_ND_discr_2                                            182 non-null    float64
 13  DVJ_Valgus_FPPA_ND_discr_3                                            182 non-null    float64
 14  Asym_SLCMJLanding-pVGRF(10percent)_discr_No_Bilateral_Asymmetry       182 non-null    float64
 15  Asym-ROM-PHIR(≥8)_discr_No_Bilateral_Asymmetry                        182 non-null    float64
 16  Asym_TJ_Valgus_FPPA(10percent)_discr_No_Bilateral_Asymmetry           182 non-null    float64
 17  TJ_Valgus_FPPA_ND_discr_1                                             182 non-null    float64
 18  TJ_Valgus_FPPA_ND_discr_2                                             182 non-null    float64
 19  TJ_Valgus_FPPA_ND_discr_3                                             182 non-null    float64
 20  Asym-ROM-PHF-KE(≥8)_discr_No_Bilateral_Asymmetry                      182 non-null    float64
 21  TJ_Valgus_FPPA_D_discr_1                                              182 non-null    float64
 22  TJ_Valgus_FPPA_D_discr_2                                              182 non-null    float64
 23  TJ_Valgus_FPPA_D_discr_3                                              182 non-null    float64
 24  Asym_SLCMJ-Height(10percent)_discr_No_Bilateral_Asymmetry             182 non-null    float64
 25  Asym_YBTpl(10percent)_discr_No_Bilateral_Asymmetry                    182 non-null    float64
 26  Position_Forward                                                      182 non-null    float64
 27  Position_Goalkeeper                                                   182 non-null    float64
 28  Position_Midfielder                                                   182 non-null    float64
 29  Asym-ROM-PADF-KE(≥8°)_discr_No_Bilateral_Asymmetry                    182 non-null    float64
 30  DVJ_Valgus_KneeMedialDisplacement_ND_discr_1                          182 non-null    float64
 31  DVJ_Valgus_KneeMedialDisplacement_ND_discr_2                          182 non-null    float64
 32  DVJ_Valgus_KneeMedialDisplacement_ND_discr_3                          182 non-null    float64
 33  DVJ_Valgus_Knee-to-ankle-ratio_discr_Varo                             182 non-null    float64
 34  Asym-ROM-PKF(≥8)_discr_No_Bilateral_Asymmetry                         182 non-null    float64
 35  Asym-ROM-PHABD(≥8)_discr_No_Bilateral_Asymmetry                       182 non-null    float64
 36  Asym-ROM-PHF-KF(≥8)_discr_No_Bilateral_Asymmetry                      182 non-null    float64
 37  Asym-ROM-PHER(≥8)_discr_No_Bilateral_Asymmetry                        182 non-null    float64
 38  AsymYBTanterior10percentdiscr_No_Bilateral_Asymmetry                  182 non-null    float64
 39  Asym-ROM-PHABD-HF(≥8)_discr_No_Bilateral_Asymmetry                    182 non-null    float64
 40  Asym-ROM-PHE(≥8)_discr_No_Bilateral_Asymmetry                         182 non-null    float64
 41  Asym(>4cm)-DVJ_Valgus_Knee;edialDisplacement_discr_No_Bilateral_Asymmetry  182 non-null  float64
 42  Asym_SLCMJTakeOff-pVGRF(10percent)_discr_No_Bilateral_Asymmetry       182 non-null    float64
 43  Asym-ROM-PHADD(≥8)_discr_No_Bilateral_Asymmetry                       182 non-null    float64
 44  Asym-YBTcomposite(10percent)_discr_No_Bilateral_Asymmetry             182 non-null    float64
 45  Asym_SingleHop(10percent)_discr_No_Bilateral_Asymmetry                182 non-null    float64
 46  Asym_YBTpm(10percent)_discr_No_Bilateral_Asymmetry                    182 non-null    float64
 47  Asym_DVJ_Valgus_FPPA(10percent)_discr_No_Bilateral_Asymmetry          182 non-null    float64
 48  Asym_SLCMJ-pLFT(10percent)_discr_No_Bilateral_Asymmetry               182 non-null    float64
 49  DominantLeg_Right                                                     182 non-null    float64
 50  Asym-ROM-PADF-KF(≥8)_discr_No_Bilateral_Asymmetry                     182 non-null    float64
 51  CPRDmentalskills_15                                                   182 non-null    float64
 52  CPRDmentalskills_16                                                   182 non-null    float64
 53  CPRDmentalskills_17                                                   182 non-null    float64
 54  CPRDmentalskills_18                                                   182 non-null    float64
 55  CPRDmentalskills_19                                                   182 non-null    float64
 56  CPRDmentalskills_20                                                   182 non-null    float64
 57  CPRDmentalskills_21                                                   182 non-null    float64
 58  CPRDmentalskills_22                                                   182 non-null    float64
 59  CPRDmentalskills_23                                                   182 non-null    float64
 60  CPRDmentalskills_24                                                   182 non-null    float64
 61  CPRDmentalskills_25                                                   182 non-null    float64
 62  CPRDmentalskills_26                                                   182 non-null    float64
 63  CPRDmentalskills_27                                                   182 non-null    float64
 64  CPRDmentalskills_28                                                   182 non-null    float64
 65  CPRDmentalskills_30                                                   182 non-null    float64
 66  BMI                                                                   182 non-null    float64
 67  ROM-PADF-KE_D                                                         182 non-null    float64
 68  ROM-PHER_ND                                                           182 non-null    float64
 69  POMStension                                                           182 non-null    float64
 70  STAI-R                                                                182 non-null    float64
 71  ROM-PHER_D                                                            182 non-null    float64
 72  ROM-PHIR_D                                                            182 non-null    float64
dtypes: float64(73)
```

Figura 6: Subconjunto de entrenamiento tras el preprocesado de la tercera libreta

Tras el preprocesado la base de datos resultante queda dividida en los subconjuntos de entrenamiento y test, igual que antes, con 182 y 78 muestras correspondientemente. La base de datos resultante constara de 260 muestras con 73 columnas numéricas y la variable objetivo.

4.4.2 Resultados

Resultados WEKA Libreta 4

Método	Precision_Score	Recall_Score	F1_Score	ROC_AUC_Score	
Modelo Libreta 3 SMOTE	0.831	1.000	0.908	0.626	Clase 0
	1.000	0.071	0.133	0.626	Clase 1
	0.861	0.833	0.769	0.626	Media Ponderada
Modelo Libreta 3 RUS	0.821	1.000	0.901	0.645	Clase 0
	?	0.000	?	0.645	Clase 1
	?	0.821	?	0.645	Media Ponderada
Modelo Libreta 3 ROS	0.847	0.953	0.897	0.644	Clase 0
	0.500	0.214	0.300	0.644	Clase 1
	0.785	0.821	0.790	0.644	Media Ponderada
Modelo Libreta 3 RUS & ROS	0.847	0.953	0.897	0.644	Clase 0
	0.500	0.214	0.300	0.644	Clase 1
	0.785	0.821	0.790	0.644	Media Ponderada

Tabla 6: Tabla representativa de los resultados dados por Python-Weka-Wrapper3 para la segunda libreta

4.4.3 Conclusiones

Para concluir se debe expresar que a causa de un fallo de ultima hora, solo se han podido tener los resultados de la tercera libreta dados por Python-Weka-Wrapper3 y no se vas a poder mostrar los obtenidos por esta misma librería en la segunda libreta, ni los obtenidos gracias a mecanismos de Scikit-Learn de ambas pese a que el código está realizado.

4.5 Libreta 4

4.5.1 Bases de Datos utilizadas

En la cuarta libreta, inicialmente la base de datos consta al igual que en las anteriores de 260 muestras, esta vez con 135 atributos cada una, 101 atributos numéricos, 33 atributos categóricos y la variable clase.

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 260 entries, 0 to 259
Columns: 133 entries, AgeGroup to Soft-Tissue_injury_≥4days
dtypes: category(1), float64(48), int64(53), object(33)
```

Figura 7: Imagen explicativa de la base de datos utilizada en la cuarta libreta previa al preprocesado de los datos

Tras el preprocesamiento de esta, la base de datos resultante de todas las transformaciones consta al igual que la inicial de 260 muestras, pero esta vez con

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 182 entries, 127 to 194
Columns: 298 entries, AgeGroup_U13U14 to Tibia_Length
dtypes: float(298)
```

Figura 8: Imágenes del subconjunto de entrenamiento posterior al preprocesado de la base de datos original

Al utilizar las técnicas para tratar con el desbalanceo el tamaño de nuestro subconjunto de entrenamiento se altera para así reducir el desbalanceo, esto se puede observar en las figuras 5 y 6.

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 302 entries, 0 to 301
Columns: 298 entries, AgeGroup_U13U14 to Tibia_Length
dtypes: float(298)
```

Figura 9: Subconjunto de entrenamiento alterado por la técnica Random Oversampling o por la técnica SMOTE

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 62 entries, 0 to 61
Columns: 298 entries, AgeGroup_U13U14 to Tibia_Length
dtypes: float(298)
```

Figura 10:Subconjunto de entrenamiento alterado por la técnica Random Undersampling

4.5.2 Resultados

Método	Precision_Score	Recall_Score	F1_Score	ROC_AUC_Score	Media
Decision Tree	0.553968	0.556920	0.555254	0.556920	0.555765
Decision Tree RUS	0.502674	0.504464	0.423645	0.504464	0.483812
Decision Tree ROS	0.471429	0.469866	0.470540	0.469866	0.470425
Decision Tree SMOTE	0.460463	0.454241	0.456516	0.454241	0.456365
Decision Tree con hiperparametros	0.489583	0.483259	0.461851	0.483259	0.479488
Decision Tree con hiperparametros RUS	0.455556	0.446429	0.449597	0.446429	0.449502
Decision Tree con hiperparametros ROS	0.465726	0.462054	0.463492	0.462054	0.463331
Decision Tree con hiperparametros SMOTE	0.535680	0.541295	0.537032	0.541295	0.538825
Decision Tree con hiperparametros y selección de atributos	0.45623	0.43527	0.43590	0.43527	0.440667
Decision Tree con hiperparametros y selección de atributos RUS	0.45556	0.44643	0.44960	0.44643	0.449505
Decision Tree con hiperparametros y selección de atributos ROS	0.50752	0.51004	0.50272	0.51004	0.50758
Decision Tree con hiperparametros y selección de atributos SMOTE	0.53568	0.54129	0.53703	0.54129	0.538822

Tabla 7:Tabla resumen de los resultados obtenidos gracias a los modelos de árboles de decisión en la cuarta libreta.

Método	Precision_Score	Recall_Score	F1_Score	ROC_AUC_Score	Media
Bagging	0.664474	0.527902	0.512500	0.527902	0.558194
Bagging RUS	0.497326	0.495536	0.458333	0.495536	0.486683
Bagging ROS	0.464531	0.473214	0.464531	0.473214	0.468078
Bagging SMOTE	0.560387	0.555804	0.560386	0.555804	0.585228
Bagging con hiperparametros	0.526877	0.537946	0.521690	0.537946	0.531115
Bagging con hiperparametros RUS	0.450937	0.438616	0.442720	0.438616	0.442722
Bagging con hiperparametros ROS	0.460648	0.443080	0.443354	0.443080	0.447541
Bagging con hiperparametros SMOTE	0.583333	0.540179	0.541176	0.540179	0.551217
Bagging con hiperparametros y selección de atributos	0.51509	0.52232	0.50427	0.52232	0.516
Bagging con hiperparametros y selección de atributos RUS	0.49437	0.49107	0.47026	0.49107	0.486692
Bagging con hiperparametros y selección de atributos ROS	0.46065	0.44308	0.44335	0.44308	0.44754
Bagging con hiperparametros y selección de atributos SMOTE	0.55427	0.54464	0.54748	0.54464	0.5477575

Tabla 8:Tabla resumen de los resultados obtenidos gracias a los modelos de Bagging en la cuarta libreta.

Método	Precision_Score	Recall_Score	F1_Score	ROC_AUC_Score	Media
RandomForest	0.410256	0.500000	0.450704	0.500000	0.465240
RandomForest RUS	0.570833	0.613839	0.556818	0.613839	0.588833
RandomForest ROS	0.664474	0.527902	0.512500	0.527902	0.558194
RandomForest SMOTE	0.664474	0.527902	0.512500	0.527902	0.558194
RandomForest con hiperparametros	0.753333	0.563616	0.570884	0.563616	0.612862
RandomForest con hiperparametros RUS	0.533288	0.554688	0.506719	0.554688	0.537346
RandomForest con hiperparametros ROS	0.493056	0.496652	0.483824	0.496652	0.492546
RandomForest con hiperparametros SMOTE	0.580000	0.520089	0.504867	0.520089	0.531261
RandomForest con hiperparametros y selección de atributos	0.53716	0.51228	0.49758	0.51228	0.514825
RandomForest con hiperparametros y selección de atributos RUS	0.49269	0.48772	0.44955	0.48772	0.47952
RandomForest con hiperparametros y selección de atributos ROS	0.53716	0.51228	0.49758	0.51228	0.514825
RandomForest con hiperparametros y selección de atributos SMOTE	0.66447	0.52790	0.51250	0.52790	0.558192

Tabla 9:Tabla resumen de los resultados obtenidos gracias a los modelos de RandomForest en la cuarta libreta

Método	Precision_Score	Recall_Score	F1_Score	ROC_AUC_Score	Media
AdaBoost	0.492424	0.493304	0.492308	0.493304	0.492835
AdaBoost RUS	0.462302	0.436384	0.411321	0.436384	0.436598
AdaBoost ROS	0.555308	0.569196	0.558019	0.569196	0.562930
AdaBoost SMOTE	0.563889	0.577009	0.567540	0.577009	0.571362
AdaBoost con hiperparametros	0.507519	0.510045	0.502715	0.510045	0.507581
AdaBoost con hiperparametros RUS	0.555000	0.585938	0.543860	0.585938	0.567684
AdaBoost con hiperparametros ROS	0.915584	0.535714	0.520567	0.535714	0.626895
AdaBoost con hiperparametros SMOTE	0.915584	0.535614	0.520567	0.535614	0.626895
AdaBoost con hiperparametros y selección de atributos	0.54010	0.55357	0.53955	0.55357	0.546697
AdaBoost con hiperparametros y selección de atributos RUS	0.50202	0.50335	0.46711	0.50335	0.493957
AdaBoost con hiperparametros y selección de atributos ROS	0.91558	0.53571	0.52057	0.53571	0.626892
AdaBoost con hiperparametros y selección de atributos SMOTE	0.66447	0.52790	0.51250	0.52790	0.558192

Tabla 10:Tabla resumen de los resultados obtenidos gracias a los modelos de AdaBoost en la cuarta libreta.

Método	Precision_Score	Recall_Score	F1_Score	ROC_AUC_Score	Media
GradientBooting	0.539286	0.524554	0.523745	0.524554	0.528034
GradientBooting RUS	0.518605	0.531250	0.479306	0.531250	0.515103
GradientBooting ROS	0.454412	0.465402	0.458333	0.465402	0.460887
GradientBooting SMOTE	0.583333	0.540179	0.541176	0.540179	0.551217
GradientBooting con hiperparametros	0.608929	0.568080	0.576662	0.568080	0.580438
GradientBooting con hiperparametros RUS	0.500000	0.500000	0.442755	0.500000	0.485689
GradientBooting con hiperparametros ROS	0.664474	0.527902	0.512500	0.527902	0.558194
GradientBooting con hiperparametros SMOTE	0.537162	0.512277	0.497585	0.512277	0.514825
GradientBooting con hiperparametros y selección de atributos	0.53929	0.52455	0.52374	0.52455	0.5280325
GradientBooting con hiperparametros y selección de atributos RUS	0.48352	0.47210	0.43193	0.47210	0.4649125
GradientBooting con hiperparametros y selección de atributos ROS	0.53716	0.51228	0.49758	0.51228	0.514825
GradientBooting con hiperparametros y selección de atributos SMOTE	0.51096	0.50446	0.49059	0.50446	0.502617

Tabla 11:Tabla resumen de los resultados obtenidos gracias a los modelos de GradientBoosting en la cuarta libreta.

Método	Precision_Score	Recall_Score	F1_Score	ROC_AUC_Score	Media
HistGradientBoosting	0.753333	0.563616	0.570884	0.563616	0.612862
HistGradientBoosting RUS	0.458133	0.431920	0.422222	0.431920	0.436049
HistGradientBoosting ROS	0.617808	0.547991	0.550519	0.547991	0.566077
HistGradientBoosting SMOTE	0.668919	0.555804	0.560386	0.555804	0.585228
HistGradientBoosting con hiperparametros	0.537162	0.512277	0.497585	0.512277	0.514825
HistGradientBoosting con hiperparametros RUS	0.480299	0.467634	0.445059	0.467634	0.465157
HistGradientBoosting con hiperparametros ROS	0.493056	0.496652	0.483824	0.496652	0.492546
HistGradientBoosting con hiperparametros SMOTE	0.617808	0.547991	0.550519	0.547991	0.566077
HistGradientBoosting con hiperparametros y selección de atributos	0.46135	0.47321	0.46453	0.47321	0.468075
HistGradientBoosting con hiperparametros y selección de atributos RUS	0.45781	0.42857	0.40280	0.42857	0.429437
HistGradientBoosting con hiperparametros y selección de atributos ROS	0.49306	0.49665	0.48382	0.49665	0.492545
HistGradientBoosting con hiperparametros y selección de atributos SMOTE	0.43846	0.44196	0.44014	0.44196	0.44063

Tabla 12:Tabla resumen de los resultados obtenidos gracias a los modelos de HistGradientBoosting en la cuarta libreta.

Método	Precision_Score	Recall_Score	F1_Score	ROC_AUC_Score	Media
XGBoost	0.510959	0.504464	0.490588	0.504464	0.502619
XGBoost RUS	0.475758	0.459821	0.436658	0.459821	0.458015
XGBoost ROS	0.539286	0.524554	0.523745	0.524554	0.528034
XGBoost SMOTE	0.617808	0.547991	0.550519	0.547991	0.566077
XGBoost con hiperparametros	0.524155	0.516741	0.515528	0.516741	0.518291
XGBoost con hiperparametros RUS	0.455556	0.446429	0.449597	0.446429	0.449502
XGBoost con hiperparametros ROS	0.438462	0.441964	0.440138	0.441964	0.440632
XGBoost con hiperparametros SMOTE	0.583333	0.540179	0.541176	0.540179	0.551217
XGBoost con hiperparametros y selección de atributos	0.49306	0.49665	0.48382	0.49665	0.492545
XGBoost con hiperparametros y selección de atributos RUS	0.45556	0.44643	0.44960	0.44643	0.449505
XGBoost con hiperparametros y selección de atributos ROS	0.46964	0.48103	0.47083	0.48103	0.475632
XGBoost con hiperparametros y selección de atributos SMOTE	0.58333	0.54018	0.54118	0.54018	0.551217

Tabla 13:Tabla resumen de los resultados obtenidos gracias a los modelos de XGBoost en la cuarta libreta.

4.5.3 Conclusiones

Para concluir, se puede observar gracias a las tablas anteriores (mejores resultados sombreados) que de una manera generalizada la técnica de desbalanceo con la que consiguen mejores resultados nuestros modelos es Random Oversampling y SMOTE.

Como se puede comprobar y era de esperar que los ensembles obtienen mejores resultados que los árboles de decisión individuales, y entre todos los ensembles AdaBoost ha sido el mejor en esta libreta.

Capítulo 5

Conclusiones y Trabajo Futuro

5.1 Conclusiones

El objetivo de este trabajo era el desarrollo de código en Python para la construcción de un módulo que nos permita integrar los algoritmos disponibles y modelos desarrollados en WEKA bajo el enfoque de Scikit-Learn.

Tras la implementación del módulo y el desarrollo de las cuatro libretas, se puede concluir con el cumplimiento de los requisitos especificados. SHAP puede encargarse de la explicabilidad de modelos de Weka gracias a esto, con el punto negativo de que solo con bases de datos de un tamaño reducido, debido al problema comentado con el "KernelExplainer".

5.2 Trabajo futuro

Algunas de las ideas como trabajo a future serian:

- Implementación de los algoritmos "Bagging" y "AdaBoost" en SHAP para su buen funcionamiento.
- Realizar una interfaz gráfica, donde puedas trabajar con algoritmos de Weka o Scikit-Learn indistintamente y utilizar las funcionalidades desarrolladas en este Proyecto.
- Solucionar el problema de "Kernel has died" buscando una mejor implementación o un mejor entorno para su ejecución.

Bibliografía

Minería de Datos:

KDD: Knowledge Discovery in Databases (mnrva.io)

Cómo es el proceso de extraer conocimiento a partir de bases de datos | campusMVP.es

230 (ucc.edu.co)

Daniel Berrar: Cross-validation

Petro Liashchynskyi, Pavlo Liashchynskyi, Grid Search, Random Search, Genetic Algorithm: A Big Comparation for NAS

Aprendizaje Automático:

Aprendizaje Automático En Acción - Alan T. Norman - Google Libros

What Is Machine Learning (ML)? - I School Online (berkeley.edu)

Te contamos qué es el 'machine learning' y cómo funciona (bbva.com)

'Machine Learning': definición, tipos y aplicaciones prácticas - Iberdrola

Desbalanceo:

Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. 2002. SMOTE: synthetic minority over-sampling technique. J. Artif. Int. Res. 16, 1 (January 2002), 321–357.

L. Peng, H. Zhang, B. Yang, and Y. Chen, "A new approach for imbalanced data classifcation based on data gravitation," Information Sciences, vol. 288, p. 347–373, 12 2014.

"Problema del desbalanceo." https://machinelearningmastery.com/ imbalanced-classification-is-hard/. Accedido 03-2021.

Explicabilidad:

Machine Learning con SHAP: Cómo evitar blackbox - sitiobigdata.com

slundberg/shap: A game theoretic approach to explain the output of any machine learning model. (github.com)

Welcome to the SHAP documentation — SHAP latest documentation

A Unified Approach to Interpreting Model Predictions (neurips.cc)

Explainable machine-learning predictions for the prevention of hypoxaemia during surgery | Nature Biomedical Engineering

Explicabilidad de modelos de Machine Learning con SHAP (squeezethedata.com)

Métodos de explicabilidad de la Inteligencia Artificial en salud - IIC (uam.es)

Explicabilidad, transparencia, trazabilidad y equidad: no todo es precisión en el uso responsable de la inteligencia artificial - Inspiring Blog (tecnalia.com)

WEKA:

Attribute-Relation File Format (ARFF) (waikato.ac.nz)

What is Weka? (tutorialspoint.com)

Weka 3 - Data Mining with Open Source Machine Learning Software in Java (waikato.ac.nz)

Scikit-Learn:

Scikit-learn (itop.es)

Scikit-Learn, herramienta básica para el Data Science en Python (master-data-scientist.com)

NumPy:

NumPy user guide — NumPy v1.22 Manual

La librería Numpy | Aprende con Alf

arrays - What are the advantages of NumPy over regular Python lists? - Stack Overflow

Pandas:

What Is Pandas in Python? Everything You Need to Know - ActiveState

pandas documentation — pandas 1.4.3 documentation (pydata.org)

pandas: powerful Python data analysis toolkit (pydata.org)

Matplotlib:

Matplotlib — Visualization with Python

Matplotlib in Python - ActiveState

Data Visualization con pandas y matplotlib | by Patricia Carmona | Ironhack | Medium

Seaborn:

Journal of Open Source Software: seaborn: statistical data visualization (theoj.org)

seaborn: statistical data visualization — seaborn 0.11.2 documentation (pydata.org)

How to use Seaborn for Data Visualization | Engineering Education (EngEd) Program | Section

Python-Weka-Wrapper3:

Introduction — python-weka-wrapper3 0.2.9 documentation (fracpete.github.io)

Clasificación y Modelos seleccionados:

Bahzad Taha Jijo, Adnan Mohsin Abdulazeez: Classification Based on Decision Tree Algorithm for Machine Learning

Árboles de decisión: qué son y cuál es su uso en Big Data (unir.net)

download (psu.edu)

Métodos de bagging y de boosting: ¿Cuál es su diferencia? - Immune Technology Institute

Ensembles: voting, bagging, boosting, stacking - IArtificial.net

sklearn.ensemble.AdaBoostClassifier — scikit-learn 1.1.1 documentation

Microsoft Word - OptBoostJCD.doc (calvin-vision.net)

La publicación en 1995 del algoritmo *AdaBoost (Adaptative Boosting)* por parte de Yoav Freund y Robert Schapire

randomforest2001.pdf (berkeley.edu)

Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. Annals of Statistics, 29(5), October 2001.

A comparative analysis of gradient boosting algorithms | SpringerLink

A Gentle Introduction to the Gradient Boosting Algorithm for Machine Learning (machinelearningmastery.com)

Mathematics | Free Full-Text | Predicting Hard Rock Pillar Stability Using GBDT, XGBoost, and LightGBM Algorithms | HTML (mdpi.com)

Histogram Boosting Gradient Classifier - Analytics Vidhya

A LightGBM-Based EEG Analysis Method for Driver Mental States Classification (hindawi.com)

A comparative analysis of gradient boosting algorithms | SpringerLink

Jason Brownlee: XGBoost With Python: Gradient Boosted Trees with XGBoost and scikit-learn

Bishan Yang, Tengjiao Wang, Dongqing Yang, and Lei Chang : BOAI: Fast Alternating Decision Tree Induction based on Bottom-up Evaluation

Yoav Freund, Llew Mason: The alternating decisión tree learning algorithm

Ron Kohavi, Clayton Kunz: Option Decision Trees with Majority Votes

Métricas de Evaluación:

Goutte Cyril, Gaussier Eric (2005) A Probabilistic Interpretation of Precision, Recall and F-score, with Implication for Evaluation.

SCRUM:

Scrum: qué es, cómo funciona y por qué es excelente (atlassian.com)

Scrum: el marco de trabajo ágil para adaptarse a los cambios (wearemarketing.com)

Anexo I. Código Implementado para la adaptación de los algoritmos de *WEKA* a *Scikit-Learn*

```
class weka_classifier(BaseEstimator, ClassifierMixin):

    def __init__(self, classifier = None, dataset = None):
        if classifier is not None:
            self.classifier = classifier
        if dataset is not None:
            self.dataset = dataset
            self.dataset.class_is_last()
        if index is not None:
            self.index = index

    def fit(self, X, y):
        return self.fit2()

    def fit2(self):
        return self.classifier.build_classifier(self.dataset)

    def predict_instance(self,x):
        x.append(0.0)
        inst = Instance.create_instance(x,classname='weka.core.DenseInstance', weight=1.0)
        inst.dataset = self.dataset

        return self.classifier.classify_instance(inst)

    def predict_proba_instance(self,x):
        x.append(0.0)
        inst = Instance.create_instance(x,classname='weka.core.DenseInstance', weight=1.0)
        inst.dataset = self.dataset

        return self.classifier.distribution_for_instance(inst)

    def predict_proba(self,X):
        prediction = []

        for i in range(X.shape[0]):
            instance = []
            for j in range(X.shape[1]):
                instance.append(X[i][j])
            instance.append(0.0)
            instance = Instance.create_instance(instance,classname='weka.core.DenseInstance', weight=1.0)
            instance.dataset=self.dataset
            prediction.append(self.classifier.distribution_for_instance(instance))

        return np.asarray(prediction)

    def predict(self,X):
        prediction = []

        for i in range(X.shape[0]):
            instance = []
            for j in range(X.shape[1]):
                instance.append(X[i][j])
            instance.append(0.0)
            instance = Instance.create_instance(instance,classname='weka.core.DenseInstance', weight=1.0)
            instance.dataset=self.dataset
            prediction.append(self.classifier.classify_instance(instance))

        return np.asarray(prediction)


    def set_data(self,dataset):
        self.dataset = dataset
        self.dataset.class_is_last()
```