Aplicación de sistemas de consenso sobre modelos de Machine Learning

Por
Raúl Dorado Pulido, Javier Mendoza García

UNIVERSIDAD COMPLUTENSE
MADRID

Grado en Ingeniería de Computadores
Facultad de Informática

Jorge Jesús Gómez Sanz
Aplicación de sistemas de consenso sobre modelos de Machine Learning

Madrid, 2018–2019

Resumen

En este trabajo se investigan y comparan distintos algoritmos de consenso aplicados entre modelos de Machine Learning. Más concretamente, se aplican algoritmos de consenso basados en la metodología Delphi sobre cuatro datasets con diferentes características. Primero se seleccionan los datasets y se aplica una limpieza sobre los mismos, y a continuación se generan diversos modelos de clasificación mediante algoritmos de Machine Learning (K-Nearest Neighbors, Máquinas de Soporte Vectorial, Bosques Aleatorios, Árboles de decisión, Redes Neuronales y Regresión Logística). Junto a estos modelos se aplican los algoritmos de consenso ideados en base al método Delphi, analizando los resultados obtenidos con los mismos.

Además, se incorpora una interfaz gráfica para observar los resultados de forma mas sencilla y poder aplicar distintos ejemplos individuales de forma manual sobre los algoritmos de consenso y los modelos de clasificación deseados.

Palabras clave: método Delphi, consenso, predicción, jupyter notebook, clasificadores, entrenamiento, Python, limpieza de datos, interfaz gráfica

Abstract

In this project various consensus algorithms are investigated, compared and applied to real world problems. The consensus algorithms are based on the Delphi method and applied with four different datasets. Firstly, the datasets are selected and cleaned, and then Machine Learning algorithms are used to create different classification models (K-Nearest Neighbor, Support Vector Machines, Random Forest, Decision Trees, Neural Networks and Logistic Regression).Along these models, the mentioned consensus algorithms created with our own approach are applied. Finally, the results are analyzed, reaching a conclusion about the use of meta-classifiers instead of the Machine Learning models alone.

Also and user interface is implemented so the results and the process can be followed by the users, making it easier for anyone to apply the developed methods. The accuracy of the application of those algorithms can be compared to the users' models, so the user can decide wheter to use or not the algorithms with their particular problems.

Keywords: Delphi method, consensus, prediction, jupyter notebook, classificators, training, Python, data cleaning, GUI

Sobre TEFLON

Teflon(cc0 1.0(documentación) MIT(código))es una plantilla de LaTeX creada por David Pacios Izquierdo con fecha de Enero de 2018. Con atribuciones de uso CC0.

Esta plantilla fue desarrollada para facilitar la creación de documentación profesional para Trabajos de Fin de Grado o Trabajos de Fin de Máster. La versión usada es la 1.3.

V:1.3 Overleaf V2 with pdfLaTeX, margin 1in, NO-bib

Índice general

Capítulo 1

Introducción

En el mundo actual, la cantidad de datos que existen y se pueden obtener es cada vez mayor. En este contexto, muchas empresas se han decantado por utilizarlos junto a algoritmos de Machine Learning, generando con ellos modelos que pueden mejorar el negocio en muchos aspectos. Son muchas las utilidades que se le pueden encontrar a estos modelos, desde crear detectores de caídas, recomendadores de productos, clasificación de especies en plantas, o herramientas de apoyo en diagnóstico al personal sanitario.

En cuanto a los datos, pueden ser obtenidos de forma directa mediante encuestas, sensores, datos históricos de un cliente e incluso de su interacción. Por ejemplo, con un sitio web,se puede recopilar información de qué productos han permanecido más tiempo a la vista del usuario o cuáles le han dirigido hacia una compra.Por otra parte, la limpieza de datos y el análisis de los mismos tiene una relevancia también muy significativa. Ante una gran cantidad de datos, es necesario filtrar los más relevantes, ya que una utilización masiva de estos puede generar una carga enorme.

Por otra parte, existen múltiples algoritmos que utilizan dichos datos, y pueden ser etiquetados de distintas formas: orientados a la clasificación u orientados a la regresión, parametrizados o sin parametrizar, supervisados o no supervisados. Cada algoritmo tiene unas características determinadas, y esto hace que sean más adecuados a distintos tipos de trabajos, por norma general. Mediante la aplicación de estos algoritmos de clasificación, podemos obtener modelos de clasificación; estos son modelos matemáticos que permiten detectar patrones y generar predicciones.

Los algoritmos de clasificación se diferencian de los de regresión en que los primeros tratan de predecir la clase de pertenencia de una instancia, mientras que aquellos de regresión predicen valores continuos.

Aunque por norma general, en la práctica siempre se ha utilizado un único modelo como clasificador (aquel que muestre un rendimiento más sobresaliente), también existen métodos para intentar mejorar los resultados obtenidos mediante la utilización conjunta de varios modelos, intentando tener otras cuestiones en cuenta:

- Pueden generarse distintos modelos, siguiendo un esquema parecido al Random Forest, con distintas partes del mismo dataset. De esta forma, se generan modelos con una cierta parcialidad que diferencia a cada uno; después, estas respuestas son tomadas en cuenta y generan una única salida.

- Pueden también utilizarse meta-modelos, es decir, entrenar a un modelo con las salidas de varios modelos generados con el mismo dataset.
- Pueden utilizarse métodos de votación y consenso, como el voto por mayoría, para obtener la respuesta más frecuente de los distintos modelos.

Hoy en día es viable utilizar algoritmos de votación y consenso, ya que a pesar de su posible carga computacional, es interesante plantearse esta alternativa. En muchas ocasiones, obtener unas respuestas más acertadas puede permitirnos hacer mejores recomendaciones de productos, y por ende aumentar nuestras ventas. En otras ocasiones, es una diferencia que puede resultar vital a la hora de hacer un diagnóstico al paciente mucho más preciso. Es por ello que, especialmente en los últimos años, se ha intentado investigar y experimentar con la fusión de modelos y la utilización conjunta de los mismos, con el objeto de obtener unos resultados mejores.

1.1. Objetivos

- Evaluar la mejora del rendimiento que pueda conseguirse en los algoritmos de clasificación utilizando un algoritmo básico de voto de mayoría y un algoritmo de consenso basado en el método Delphi, estando este último explicado en la sección 3.2.2. También se valoran los métodos encontrados en diversos papers en la sección 3.2
- Desarrollar una interfaz gráfica sencilla 6, que permita al usuario cargar una lista de modelos y sus datasets y utilizar los algoritmos desarrollados. Se busca probar así el rendimiento de los mecanismos de consenso para un problema en particular, de forma que el usuario pueda decidir si le compensa su uso. También se podrán aplicar sobre casos individuales. En ambas opciones, se mostrará al usuario un feedback del proceso que ha llevado a la respuesta final, indicando los resultados de los modelos particulares y cómo cambian si estos lo hicieran.

1.2. Plan de trabajo

- Investigación del estado actual del ámbito del Machine Learning respecto al uso de sistemas de consenso, buscando información sobre dichos métodos y la fusión de modelos ya utilizados de forma individual y unos datasets adecuados para probar este método.
- Limpieza de los dataset tomados en cuenta, mediante el uso de la librería Pandas. Esta limpieza se hace sobre los casos con variables desconocidas y eliminando variables que no sean muy relevantes o redundantes. Se utilizan dataset con salidas binarias y multiclase, debido al interés en analizar si hay una diferencia significativa en utilizar el algoritmo en un tipo u otro de clasificadores. Después de esto, se crean los modelos de Machine Learning, probando distintas configuraciones para que estos tengan una precisión aceptable para continuar en el proceso.
- Implementación de un voto por mayoría entre n modelos, donde se alimentan ciertos casos de ejemplo y se comprueba si esta práctica mejora los resultados gracias a la

combinación de distintos tipos de clasificadores. A continuación, se implementa un mecanismo de consenso basado en el método Delphi, en el cual los modelos tienen cierta comunicación entre ellos y ofrecen un feedback al usuario, de forma que se podrá facilitar una visualización del proceso.

- Creación de una interfaz que permite al usuario cargar los modelos y los dataset para ejecutar dichos métodos. Esta interfaz, a su vez, muestra el proceso de consenso de forma inteligible para el usuario.

1.3. Estructura del documento

- **Capítulo 3**: en este capítulo queda reflejado el estado del arte. En él se enumeran los distintos mecanismos de consensos relevantes encontrados en diversas investigaciones y los resultados de los mismos. También se habla de la interpretabilidad de modelos y los algoritmos utilizados para generarlos y del método Delphi.
- **Capítulo 4**: en este capítulo se explica el origen de los distintos datasets utilizados durante el proyecto, el contenido de los mismos y la limpieza de datos y el mapeo realizado sobre ellos.
- **Capítulo 5**: en este capítulo se exponen y explican los mecanismos de consenso implementados para el proyecto, además de hablarse de la propia implementación de los mismos.
- **Capítulo 6**: durante este capítulo se muestra la interacción con la interfaz gráfica implementada con casos de uso de la misma.
- **Capítulo 7**: en este capítulo se reflejan los resultados obtenidos al aplicar los mecanismos desarrollados sobre los datasets, además de las conclusiones y análisis de estos.
- **Capítulo 8 y 9**: se exponen las conclusiones del trabajo realizado y el posible trabajo futuro a seguir.
- **Capítulo 10**: por último, se indican las contribuciones personales de los colaboradores del proyecto.

1.4. Lenguaje y entornos utilizados

El lenguaje utilizado para la realización del proyecto es Python versión 3.7.0 y el entorno para editar y ejecutar Jupyter Notebook versión 5.6.0

1.4.1. Python

Python es un lenguaje de Programación que contiene gran cantidad de librerías orientadas al Machine Learning.

Para alcanzar los resultados deseados por los algoritmos y la obtención de gráficas y tablas, que realizan representaciones de los resultados más visuales, se necesita el uso de

librerías como Pandas, NumPy, MatplotLib, o Seaborn. Los modelos se guardan mediante la librería Pickle.

1.4.2. Jupyter notebook

Jupyter Notebook es una aplicación web de software libre que nos permite crear y compartir documentos, este incluye limpieza y transformación de datos, simulaciones numéricas, modelos estadísticos, visualización de datos, aprendizaje automático...

El uso de este entorno proporciona mayor facilidad y rapidez en el trabajo realizado que con el único uso de la consola de comandos, además de ser más interactivo y visual.

Capítulo 2

Introduction

Nowadays, the amount of data that exists and that can be obtained is increasing. In this context, many businesses have decided to use them along Machine Learning algorithms, thus creating models that could improve the business in many ways. There are a lot of utilities for the data, as fall detection, product recommendation, plant classification, or support tools for health professionals.

Data can be obtained directly through surveys, sensors, historical data of clients and their interaction. For example, a website can get information about what products have remained on the user's screen more time than others or which of them directed the user to the purchase.

On the other hand, there are plenty of algorithms that can use data, we can classify them in many ways: classification or regression, parameterized or non parameterized, supervised or non supervised. Each of them has its particular characteristics, making them more suitable for certain problems. Applying these classification algorithms, we can generate classification models, this is, mathematical models that are can detect patterns in the data and obtain predictions with new instances.

Even though in practice only one single model has always been used as classificator (the one with the best performance), there are other methods we can use in order to improve the performance, involving all those models and there are many approaches:

- Many models could be generated, just as in Random Forest's approach, with different subsets of the same dataset. This way, the models have certain bias that make them different; the answers of every model are all taken in count, generating one single answer.
- Meta-classifiers could also be used, that is, a model can be trained with the answers of models generated with the same dataset.
- Finally, voting and consensus methods could also be used, just as majority vote, you obtain the most repeated value along the models.

Nowadays, it is worth using voting methods, despite the computational cost. In most of cases, obtaining more accurate results let us make better recommendations, and so the sales can improve. There are also scenarios where is crucial to make more accurate

predictions, such as those of medical field. These are some of the reasons why the fusion of models is being an interesting option.

2.1. Objetivos

- Evaluate the actual improvement in the performance that can be obtained with classification algorithms using some consensus' algorithms such as majority vote or more elaborated algorithms based on the Delphi method, which is explained at section 3.2.2.
- Develop a straightforward GUI that helps the user in the process of loading models list and datasets and using the developed algorithms. It allows testing the performance of consensus mechanisms for a specific problem, so that the user can decide wheter its use is worth or not. The algorithms could also be applied on single cases. In both options, the user will see the feedback of the process that led to the final answer, including results from all the classificators and how their answers change if they do.

2.2. Working plan

- Research of the current state of the use of voting and consensus methods in the Machine Learning field, looking for information about consensus methods and fusion of models already used and suitable datasets to test the developed algorithms.
- Data cleaning, by using Pandas library. This cleaning is done in instances with unknown parameters and removing columns that are not that relevant or which do not provide enough information. Binary and multiclass datasets are used, due to the interest in analyze if there are significant differences between using the algorithms in any specific type of datasets. After this, Machine Learning models are created, testing different configurations looking for an acceptable goodness of them.
- Implementation of majority vote between n models, where some instances from test data are tested and it is checked if that practice improves the results thanks to the combination of different types of classifiers. Next, a consensus mechanism based on Delphi method is implemented, in which models have some communication between them and offers a feedback so the user can take a look at the process.
- Creation of a GUI which allows the user to load models and their datasets to execute the developed methods. This GUI, shows the consensus output in an intelligible way for the user.

2.3. Document structure

- **Chapter 3**: the state of the art is explained in this chapter. It enumerates the different consensus mechanisms found in various researches and their results. There is also a explanation about models' interpretability and the algorithms used to generate them.

- **Chapter 4**: this chapter explains the origin of the different datasets used during the project, their content and the data cleaning applied on them.
- **Chapter 5**: consensus mechanisms implemented for this project and their implementations are exposed and explained in this chapter.
- **Chapter 6**: during this chapter, the GUI is shown along some cases of use.
- **Chapter 7**: in this chapter the results obtained by applying the mechanisms developed on the datasets are shown and analyzed.
- **Chapter 8 & 9**: the conclusions about the project are and the possible future work.
- **Chapter 10**: finally, personal contributions of the project collaborators are indicated here.

2.4. Language and enviroments used

The language used in the project is Python, 3.7.0 version, and the environment used to edit and run the application is Jupyter Notebook, 5.6.0 version.

2.4.1. Python

Python is a programming language that brings a lot of libraries oriented to Machine Learning.

To achieve the expected results by the algorithms and obtain graphics and tables, which make visual representations of the results, the use of libraries such as Pandas, NumPy, MatplotLib, or Seaborn is needed. Models are saved using the Pickle library.

2.4.2. Jupyter notebook

Jupyter notebook is an open-code web application that allows to create and share documents, including data cleaning and data mapping, numerical simulations, statistics models, data visualization, machine learning...

The use of this environment provides an easy and fast way to work that the command prompt, being also more visual and interactive.

Capítulo 3

Estado del arte

En este capítulo se hablará del estado del arte. Aquí se enumeran los distintos tipos de algoritmos encontrados, además de un breve resumen de los papers correspondientes de donde se ha extraído dicha información. Además, también se explican los distintos tipos de algoritmos de aprendizaje utilizados durante el proyecto, en qué consiste la metodología Delphi en la que está basado el algoritmo de consenso y la interpretabilidad de modelos (también llamados clasificadores) de aprendizaje automático.

Estos clasificadores se evalúan atendiendo a determinadas métricas, y se dice que la bondad de un modelo (lo bueno que es dicho modelo prediciendo) está relacionado con los valores de las mismas. Las métricas más utilizadas son la precisión, aunque también es común acudir a otras medidas como el recall o el f1 score. Estos últimos se obtienen mediante la aplicación de fórmulas cuyas variables son los falsos positivos y negativos y los verdaderos positivos y negativos, explicados a continuación con valoraciones médicas:

- **Falso positivo:** un ejemplo sería la evaluación de presencia de enfermedad cuando realmente hay una ausencia de la misma.
- **Falso negativo:** un ejemplo sería la evaluación de ausencia de enfermedad cuando esta está presente.
- **Verdadero positivo y verdadero negativo:** estos son casos de instancias correctamente clasificadas, es decir, se determina que hay enfermedad cuando esta está presente y la ausencia de la misma cuando esta está ausente.

El planteamiento inicial es el de desarrollar una aplicación de consenso entre modelos con el fin de que sus predicciones sean más acertadas que de forma individual. También se quiere que los usuarios puedan usar estos algoritmos junto a sus propios modelos y que observen si mejoran las predicciones para su problema concreto. La búsqueda de información se centra en localizar métodos de consenso y sistemas de fusión de modelos previos. Puesto que también se quiere poder utilizar de forma general, debe atenderse también a la cuestión de la interpretabilidad.

3.1. Algoritmos de clasificación

En Machine Learning existen una amplia variedad de algoritmos y es posible clasificarlos atendiendo a distintos factores. Se puede así, por ejemplo, distinguir entre **algoritmos**

de regresión o de clasificación; un algoritmo de regresión devuelve un valor continuo, como puede ser el valor estimado de una vivienda de un barrio concreto (conociendo de antemano otros casos de viviendas de dicho barrio), mientras que un algoritmo de clasificación devuelve un valor concreto dentro del posible conjunto de valores etiquetados en los datos de entrenamiento (las instancias de los datos utilizadas para el entrenamiento del modelo de clasificación), como puede ser la especie de planta a la que pertenece una determinada hoja.

Otro de los principales criterios atiende al **tipo de aprendizaje**, que puede ser supervisado, semisupervisado o no supervisado. La supervisión como concepto indica si realmente a la hora de aplicar el algoritmo se conocen o no las etiquetas de caso de ejemplo, si son todas, o en qué medida. Por tomar un ejemplo, normalmente un algoritmo K-Means se utiliza de forma no supervisada, aunque en ciertas ocasiones sí se conocen las etiquetas del problema y puede evaluarse su rendimiento.

También hay que tener en cuenta que ciertos algoritmos admiten parametrización y otros no, y esta también es una característica clave. Un ejemplo de parametrización sería la profundidad de un árbol de decisión, es decir, al algoritmo se le puede indicar cómo de profundo queremos que llegue en la construcción del árbol, y esto tiene una repercusión en el resultado final.

También es de especial importancia la interpretación de los resultados. Es fácil distinguir por qué un Àrbol de decisión ha dado una respuesta final, aunque esto se complica significativamente si aplicamos un Random forest en su lugar. Por otra parte, hay algoritmos que son prácticamente una caja negra, donde no se puede distinguir por qué ha dado una respuesta, como por ejemplo en una red neuronal.

Los tipos de algoritmos se pueden separar bajo muchos otros criterios similares a los anteriores, ya que, por ejemplo, no es el mismo tipo de clasificación aquella realizada mediante clusters que mediante árboles de decisión o redes neuronales. En esta sección se explican, a grandes rasgos, algunos de los algoritmos más comunes, y más en particular aquellos que se han planteado para utilizar durante el proyecto para generar nuestros modelos.

3.1.1. K-Nearest Neighbour

El algoritmo KNN (del inglés K Nearest Neighbour, K vecinos más cercanos) [1][2] es un algoritmo de aprendizaje supervisado utilizado tanto para clasificación como para regresión.

El entrenamiento de este algoritmo funciona mediante la entrada de N casos de ejemplo, en forma de vector, que ubicará en un hiper espacio y cuyas dimensiones coinciden con la longitud de los vectores de los datos.

Este algoritmo utiliza una constante K, definida por el usuario, que es utilizada a la hora de realizar la clasificación de un caso de prueba: cuando al modelo le preguntamos por un caso sin clasificar, para él la clase de dicha instancia será inferida teniendo en

cuenta cuáles son las clases de los K casos más cercanos, según la métrica de similitud utilizada, y se asignará la clase más frecuente.

Figura 3.1 : El nuevo ejemplo para clasificar se encuentra en un punto bastante difuso, a simple vista, no se puede clasificar de forma clara debido a que se encuentra rodeado de casos de la clase A y de la clase B. Puede apreciarse que, asignando distintos valores de K para obtener los vecinos cercanos, se obtienen más vecinos de la clase B cuando el valor K = 3, y más de la case A cuando el número de vecinos es K = 7.

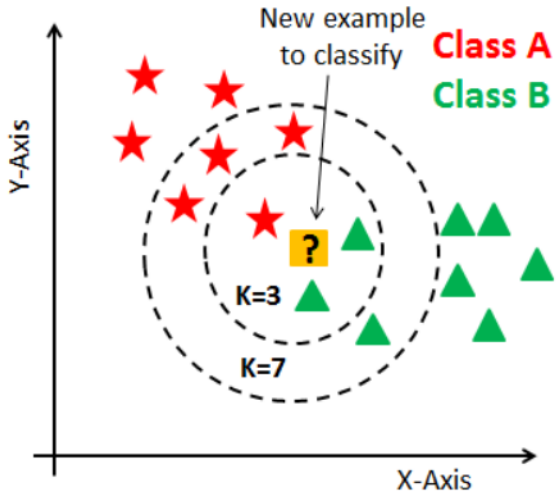

Figura 3.1: Influencia de K al determinar la clase de una nueva instancia [2]

Puesto que el valor de K debe ser elegido a la hora de crear el modelo y no existe un valor K adecuado para todos los tipos de datos, es importante analizar cuál es el valor óptimo para un determinado caso: un número bajo de vecinos puede provocar problemas por el ruido, y un valor muy alto es, computacionalmente hablando, costoso. De la misma forma, un número de vecinos bajo evita la parcialidad y mantiene un valor alto de varianza, mientras que un número mayor de vecinos tiende a producir cierto overfitting y a tener un valor de varianza más bajo.

Para analizar un caso en particular, se puede utilizar el método del codo (Figura 3.2), que consiste en ver la relación entre el valor de error de cada modelo en función del número de vecinos que le hemos dado. Mediante una gráfica, se puede ver cuál es el valor más adecuado teniendo en cuenta lo anteriormente mencionado.

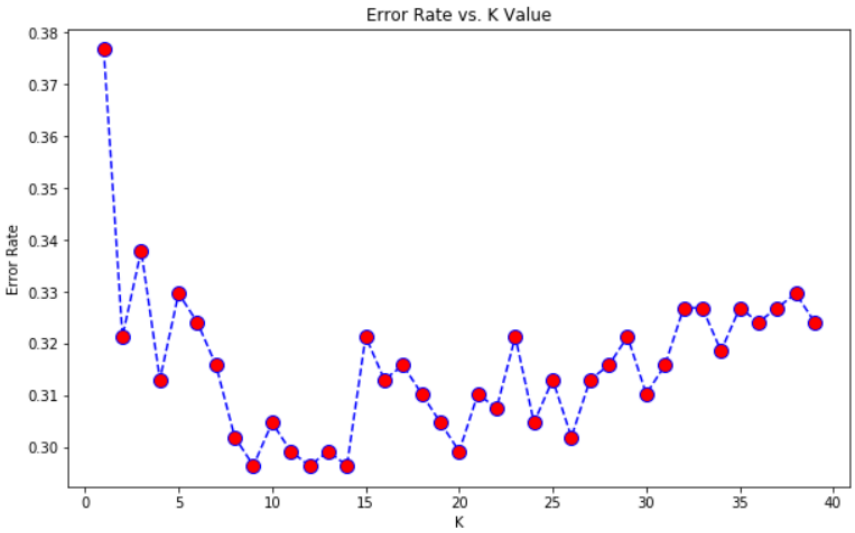

Figura 3.2: Gráfica generada con el método del codo

3.1.2. K-Means Clustering

El algoritmo de K-Means [3][4] es un algoritmo no supervisado enfocado exclusivamente al clustering.

Este trata de agrupar los casos de entrada similares, es decir, aquellos que son cercanos en el hiperespacio, de tal forma que se generan una serie de clusters en función del número K de centroides (el punto representante del centro del cluster) que se le haya indicado. Esta asignación de cada elemento a un cluster se realiza teniendo en cuenta cuál es el cluster con el valor medio más cercano, es decir, dónde están los elementos más similares. Existen distintas métricas utilizadas para determinar la distancia, como pueden ser la Manhattan o la Euclidea.

Estos centroides son generados de forma arbitraria, y con cada iteración (cada vez que incluimos un valor) el valor medio de cada cluster cambiará. Esto es debido a que el valor medio se usa como medida para elegir a qué cluster pertenece un elemento, pero dicho elemento también va a definir el valor medio de ese cluster una vez sea añadido. El algoritmo seguirá su curso hasta cierto número de iteraciones o hasta que los centroides queden estabilizados.

Figura 3.3 : se encuentran los k-centros de los grupos y en este caso se dividen en 3, de forma que el cuadrado de las distancias del grupo al centro están minimizadas. Suelen ser grupos del mismo tamaño, debido a que se asignará el tipo según el centroide más cercano.

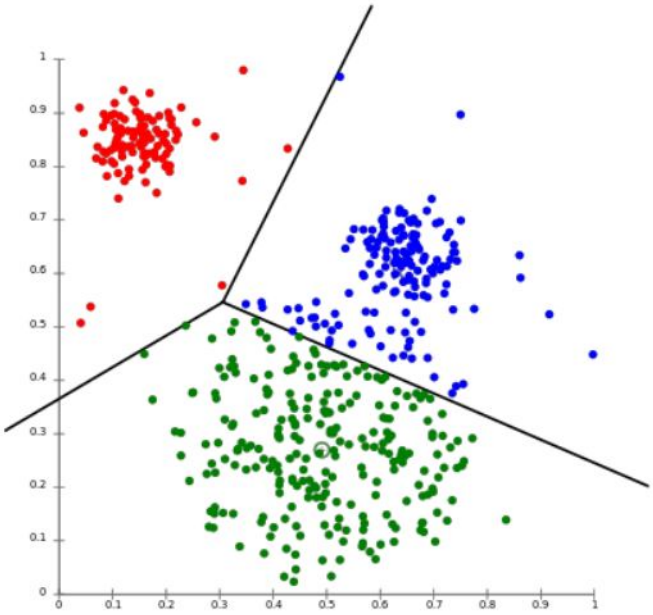

Figura 3.3: Representación gráfica de K-Means de tres clases [5]

3.1.3. Support Vector Machine

Las SVM (Máquinas de Soporte Vectorial) [6] son un conjunto de algoritmos de aprendizaje supervisado utilizados tanto para clasificación como para regresión. Concretamente durante este proyecto, se utilizan las SVC, es decir, aquellas que están orientadas a la clasificación, tanto binaria como multiclase.

Entre sus puntos fuertes es posible destacar que son efectivas en espacios dimensionales grandes, incluso en aquellos en los que el número de dimensiones es mayor que el de casos de ejemplo, donde habrá que prestar especial atención al kernel utilizado; es una función mediante la cual se generan los hiperplanos que separarán los casos de ejemplo. Algunos de estos kernel serían el lineal, el RBF o los polinomiales. Por otra parte, el coste de sus cálculos también puede ser mayor que el de otros algoritmos.

Figura 3.4 : puede apreciarse como aplicando distintos tipos de kernel, se obtienen diferentes tipos de áreas para la predicción de clases.

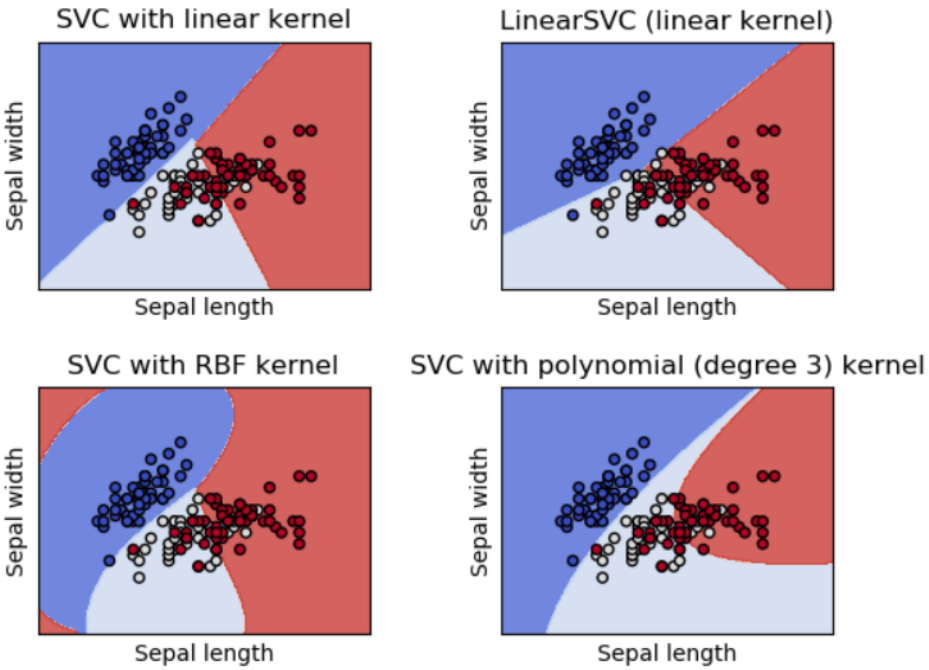

Figura 3.4: Distintos tipos de kernel aplicados sobre el iris dataset con SVC [7]

Este algoritmo necesita además que se le especifiquen dos parámetros de entrada: el coste y el gamma. El coste, C, indica la rigidez con la que el algoritmo busca separar los puntos mediante el hiperplano. Gamma, por su parte, es un parámetro libre de los kernel y su valor indica hasta qué distancia un caso de ejemplo influencia a la hora de construir el modelo: un valor alto implicaría una mayor parcialidad y menor varianza y un valor bajo lo contrario.

Un aspecto a destacar es que para clasificaciones multiclase se utilizan dos estrategias diferentes según el algoritmo concreto utilizado:

- One vs One: se construyen NCLASS * (NCLASS - 1) / 2 clasificadores, donde nclass es el número de clases, ya que se entrena un clasificador para cada par de de las mismas. Una función de decisión determina después de haber aplicado todos los clasificadores cuál es el resultado.
- One vs All: se entrenan tantos modelos como clases haya y nos quedamos con la clase cuyo clasificador asociado ha dado el valor más alto.

3.1.4. Decision Trees

El algoritmo árbol de decisiones es un modelo analítico, llamado así por su representación esquemática de las alternativas disponibles. [8]

Este algoritmo facilita la toma de mejores decisiones para la resolución de un problema, por lo general está formado por un **único nodo raíz**, el cual se ramifica dando lugar a nuevos nodos adicionales, los cuales solo se pueden alcanzar por un único camino desde el nodo raíz. A partir de estos nuevos nodos, se llega a los nodos hoja, que representan

un dato de predicción.

Para avanzar por las distintas ramificaciones que conectan a los nodos, se atiende a su conjunto de reglas de clasificación, asociada a su etiqueta que se halla al final de la ramificación.

Figura 3.5 : En el ejemplo, puede verse como se avanza entre ramas dependiendo del número de unidades que se hayan solicitado, y si es necesario, dependiendo del lugar de envío. Para aplicar un 20 % de descuento, sería necesario solicitar entre 10 y 50 unidades del producto y enviar a España. Si el número de unidades no llega a 10, no se aplicaría descuento para España ni para Europa.

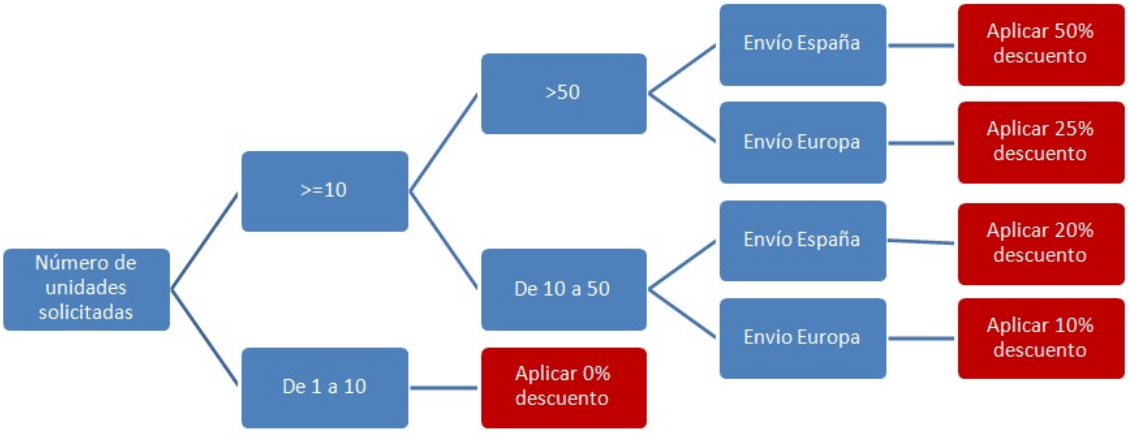

Figura 3.5: Ejemplo de un árbol de decisión [9]

3.1.5. Random Forest

Random Forest (o Bosques Aleatorios) [10] es el nombre que recibe un algoritmo de aprendizaje supervisado que se basa en unir distintos árboles de decisión. Es utilizado tanto para clasificación como para regresión.

Se caracteriza por ser un algoritmo muy flexible y sencillo de utilizar: un bosque está formado por distintos árboles de decisión creados mediante sets de datos seleccionados de forma aleatoria. Después se dará mediante votación la respuesta final, siendo esta más o menos precisa dependiendo de la cantidad de árboles que lo conformen.

Otros aspectos a destacar también son que permite ver la importancia relativa de cada variable de los datos y que por norma general suelen evitar el overfitting típico de árboles de decisión de cierta profundidad, ya que al formar su respuesta con aquellas dadas por muchos árboles generados de forma distinta, se evita la parcialidad. Esto a su vez genera una mayor carga computacional, ya que estamos hablando de varios árboles de decisión, y su interpretabilidad se complica considerablemente.

Figura 3.6 : el algoritmo funciona con 4 pasos, primero selecciona muestras aleatorias del dataset, segundo construye un árbol de decisiones para cada muestra y obtiene la predicción para cada árbol, después realiza una votación para cada resultado obtenido y para finalizar elige como predicción el valor más votado anteriormente.

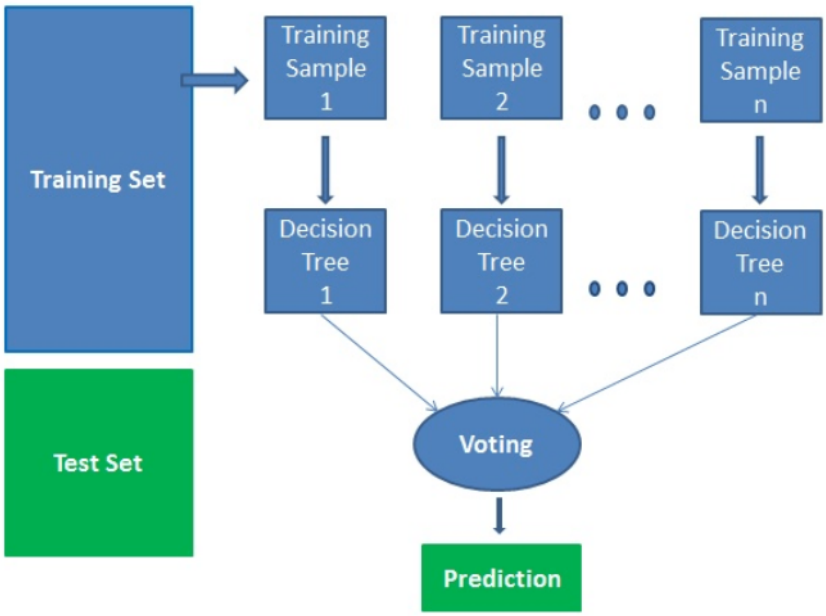

Figura 3.6: Esquema del funcionamiento de un Random Forest [11]

3.1.6. Logistic Regression

El algoritmo de Regresión Logística es un tipo de análisis estadístico utilizado para predecir el resultado de una variable binaria según los datos que se proporcionan como entrada. [12]

La Regresión Logística es un caso especial de regresión lineal, donde la variable de salida es del tipo categórica o cualitativa, mientras que en una regresión lineal al uso es de tipo cuantitativa. (Figura 3.7) de modo que la función que se genera es lineal para la regresión lineal y no lineal para la regresión logística.

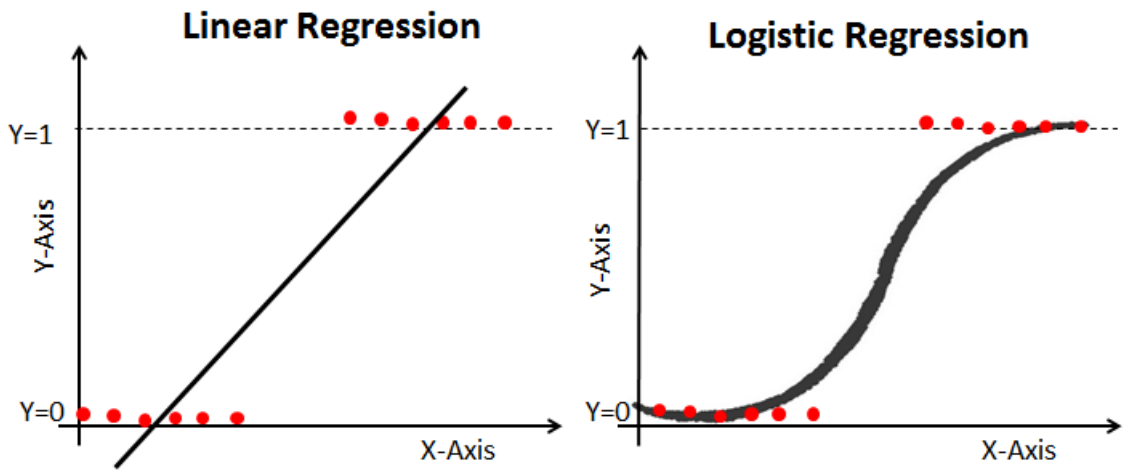

Figura 3.7: Ejemplo de regresión lineal VS. regresión logística [13]

Este algoritmo se aproxima a los valores 0 (no ocurre) y 1 (ocurre) según los datos de entrada, utilizando como función de enlace la función **logit**, la cual, se modela como una función lineal de los casos, representados con los nombresX_i.

$$\operatorname{logit}(p_i) = \ln\left(\frac{p_i}{1-p_i}\right) = \beta_0 + \beta_1 x_{1,i} + \cdots + \beta_k x_{k,i}.$$

Tipos de regresión logística:

- Regresión logística binaria: la variable objetivo tiene solo 2 posibles salidas.
- Regresión logística multinomial: la variable objetivo tiene 3 o más posibles salidas nominales.
- Regresión logística ordinal: la variable objetivo tiene 3 o más posibles salidas ordinales.

Este algoritmo genera modelos eficientes respecto a los recursos computacionales y a su implementación, pero en cambio no es bueno para un alto número de ejemplos/variables, ya que es propenso al overfitting.

3.1.7. Redes Neuronales

El algoritmo de Redes Neuronales es un modelo computacional que simula el comportamiento de las neuronas humanas, siendo válido tanto para clasificación como para regresión. [14][15]

Las neuronas estás conectadas entre sí a través de enlaces, y su ventaja es que no necesitan de una programación explícita, sino que aprenden por ellas mismas. Para realizar dicho aprendizaje, se busca minimizar la función de pérdida, lo cual se logra actualizando los valores de los pesos de las neuronas mediante propagación hacia atrás. El objetivo del

algoritmo es resolver el problema, de la misma manera que lo haría un humano.

En la práctica, se utiliza frecuentemente el **perceptrón multicapa**, el cual contiene la **capa de entrada**, formada por las entradas de valores a la red, la **capa oculta**, que contiene las neuronas entre las capas de entrada y de salida, y la **capa de salida**, formada por las neuronas que constituye la salida de la red. Diferentes pesos se establecen en las conexiones de dichos nodos y se van ajustando a medida que se entrena a la propia red con casos de ejemplo.

Figura 3.8 : cada nodo circular representa una neurona artificial y cada flecha representa una conexión desde la salida de una neurona a la entrada de otra, dependiendo del lugar en que se encuentren pueden pertenecer a la capa de entrada, capa oculta o capa de salida.

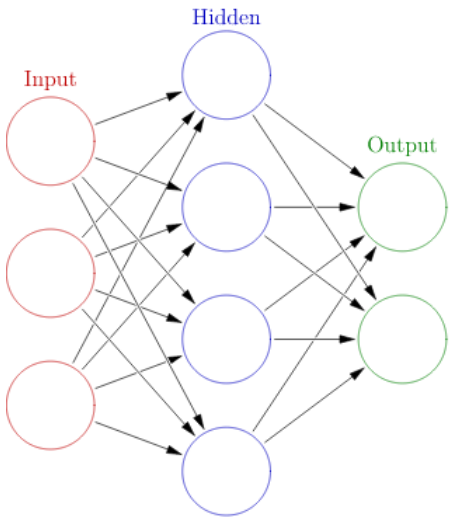

Figura 3.8: Ejemplo de Red Neuronal con 3 nodos de entrada. una capa oculta con 4 nodos y 2 nodos de salida [16]

3.2. Algoritmos y tecnologías de consenso

Tras revisar diversos trabajos previos [17] [18] [19], se pueden encontrar una gran diversidad de algoritmos de consenso en la actualidad. Aunque todos tienen una línea común, que sería la utilización de los modelos y alguna característica de los mismos, en esta sección se resumen aquellos considerados de especial relevancia de entre los trabajos encontrados y los trabajos donde aparecen o se ha obtenido mayor información sobre estas tecnologías.

Podemos enumerar las siguientes metodologías relevantes:

- **Voto mayoría**: para un caso concreto, la respuesta final es aquella que recibe más votos entre todos los modelos listados. Es la metodología más simple de entre todas las encontradas, aunque a pesar de ello, proporciona unos resultados satisfactorios. Más detalles y la explicación se pueden ver en 5.1.1 [17]
- **Probabilidad Media**: para esta metodología se precisa la utilización de varios vectores, uno por cada clase, del tamaño del número de modelos a utilizar. En estos se guardan las probabilidades de pertenencia a cada clase según cada modelo. Para

dar una respuesta, se necesita calcular la media final de cada vector, y el resultado proporcionado al usuario será aquella clase de valor más alto. [18]

- **Media Ponderada AUC**: en esta metodología, se necesita calcular la curva de ROC de cada modelo. Esta curva se genera mediante la matriz de confusión, ya que para construirla se precisa conocer el número de falsos y verdaderos positivos que tienen los modelos. Si se calcula el AUC (Area Under Curve) de cada modelo, se puede asumir que aquellos con un mayor valor, y por tanto con un mayor área, son modelos cuyo recall es mejor, y por tanto su rendimiento es mayor. De esta forma, se puede llegar a justificar que un modelo tenga más peso en la votación final que otro con un área menor. Hay que tener en cuenta que las curvas ROC se obtienen de forma directa únicamente para problemas binarios, y que utilizarlo en problemas multiclase implica la utilización de un esquema One vs All para obtener las distintas curvas a considerar. [18]

- **Stacking con GLM**: en este método, se crean una serie de clasificadores, cada uno con cierta parcialidad particular mediante, por ejemplo, un cross validation. Las predicciones de estos clasificadores se utilizarán para entrenar un metaclasificador, que será el que dé una respuesta final. Este método puede implementarse por clasificación o regresión, siendo este último posible de implementar teniendo en cuenta la probabilidad de pertenencia a cada clase. Como metaclasificador se puede utilizar para este caso una regresión logística. [18]

- **Stacking con Random Forest**: este método sigue el mismo esquema que el anterior, pero utilizando un Random Forest como metaclasificador. [18]

- **Media Ponderada del Error de Predicción**: este algoritmo se basa en el error de predicción para dar a los modelos un peso. Por tanto, se necesita calcular dicho error y asignar a cada modelo una relevancia ponderada a sus respuestas. Una vez implantada esta base, se entrena un clasificador con las salidas de esos modelos, teniendo en cuenta su peso. [18]

- **Locally Weighted Fusion**: este método tiene como base crear también varios modelos entrenados con distintas porciones del dataset. A la hora de dar una respuesta final, se evalúa el rendimiento local de cada modelo, utilizando casos similares al que se está evaluando. Estos casos son obtenidos mediante una fórmula que ubica este caso a evaluar en el hiperespacio, y selecciona los casos de tal forma que, para cada modelo, se calculan una serie de parámetros, los cuales permiten estimar cuál de ellos es el más adecuado para dar la respuesta final. [19]

3.2.1. Voto por mayoría

Aunque existen diversos mecanismos para la fusión de modelos, una de las primeras y más simple y común de las aproximaciones sería un voto por mayoría. El funcionamiento es sencillo (Figura 3.9): dada una lista de modelos y un caso de prueba, cada modelo, al ser preguntado por ese caso, genera una respuesta. Entre modelos generados con el mismo algoritmo y distinta configuración es relativamente fácil que esta respuesta varíe, aunque dependerá de lo límite que pueda ser el caso, es decir, si por ejemplo se utilizan unos modelos KNN con distinto número de vecinos, es posible que un caso concreto sea cercano a dos etiquetas, y el número de vecinos que tenemos en cuenta al preguntar es de

suma importancia. Esta diferencia en las posibles respuestas es más variable si se habla de algoritmos diferentes, especialmente teniendo en cuenta que algunos son más apropiados para algunos trabajos y su rendimiento puede variar.

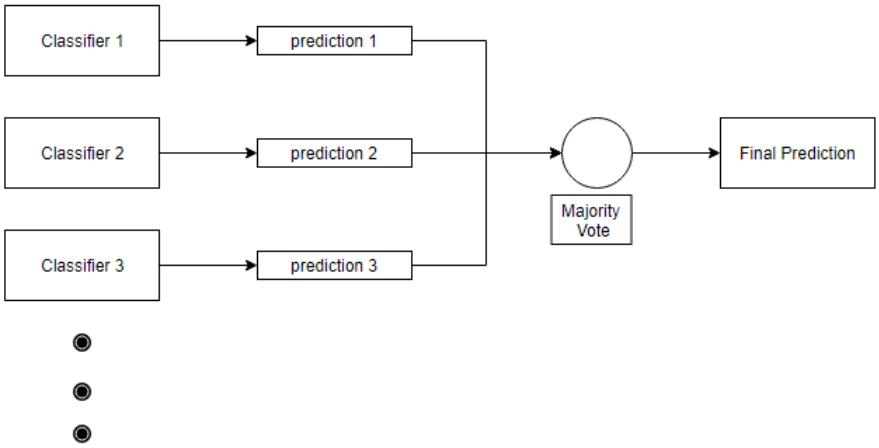

Figura 3.9: Consenso de modelos utilizando voto por mayoría

Es fácil encontrar referencias académicas respecto al uso de este sistema en la actualidad [17]. En este paper se documentan los resultados de aplicar esta técnica entre diversos modelos: una red neuronal, un árbol de decisión, una SVM y un KNN aplicados a un problema de reconocimiento de emociones en voz, basados en las variables más interesantes que se pueden extraer de las muestras; en cuanto a los dos set de datos empleados, se han dividido de forma proporcional y todas las clases (angry, happy, neutral y sad) están equitativamente representadas, aunque en el segundo tienen una representación mayoritaria de la primera de las clases.

Aunque el proceso de extracción de las features y la implementación de los modelos no sea realmente relevante, el paper llega a unas conclusiones que sí son interesantes. El uso de forma paralela de los cuatro modelos y el voto de la mayoría proporciona un incremento medio del 2-3 % en la precisión y llega a la conclusión de que, aunque en concreto el modelo de SVM tenía unos resultados satisfactorios de por sí, el uso del voto de la mayoría generaba un aumento de la precisión suficientemente interesante como para usarlo. Sin embargo, también se señala la importancia del tiempo: no todos los modelos tardan lo mismo en ejecutarse, y debido a esto el tiempo de ejecución es un poco más largo, aunque si lo que buscamos es una precisión más alta, es definitivamente la mejor opción.

3.2.2. El método Delphi

El método Delphi es un método de comunicación estructurada entre diferentes expertos, planteado para abordar un problema entre ellos e intentar llegar a un consenso.

Su objetivo principal es llegar a dicho consenso, basándose en la discusión entre la comunidad de expertos mediante un proceso de interacción entre los mismos. El resultado dependerá en gran medida de la calidad del cuestionario, cuyo objetivo debería ser

guiar hacia una respuesta unánime a unos expertos, cuya calidad también es importante. Para lograr esto, se realizan preguntas consecutivas en diferentes rondas, de forma que las opiniones de los expertos no solamente pueden variar, sino que además aportan a la forma de la pregunta a realizar en la siguiente ronda. De esta forma se tienen una serie de expertos, coordinados por un moderador, cuyas respuestas se ven variadas por la influencia de las respuestas propias y del resto de expertos.

Como se verá en el paper resumido a continuación **??**, este método aplica a situaciones donde las respuestas pueden variar ante la misma pregunta. En el proyecto en particular también se parte de esta base, ya que se lanza una pregunta y esta puede tener diferentes respuestas variando de un modelo a otro. Además, el hecho de que no exista una respuesta única, hace que los modelos puedan estar en desacuerdo, y las respuestas del resto de modelos influencian en dicha situación, ya que se pedirá al moderador que proporcione unos casos nuevos a evaluar.

3.2.2.1. Revisando el Método Delphi para Agentes

Este documento [20] trata sobre la interacción de agentes que discuten y proporcionan una conclusión después de algunas rondas de intercambio de información mutua. Para modelar, implementar e ilustrar los ejemplos, utilizan la metodología INGENIAS.

Tras coger un trabajo previo, el cual investigó sobre la argumentación automática real entre los expertos para producir discusiones entre agentes que conduzcan a conclusiones, se realiza una mejora a la solución añadiendo el método Delphi.

Se utilizó dicho método para que los expertos se vean obligados a considerar las razones y conclusiones de otros expertos, y que lleguen a reconsiderar sus opiniones para poder llegar a un consenso. Es necesario definir unos límites para evitar que el proceso se extienda más de lo deseado, y para ellos se ejecutan un número finito de rondas contestando el cuestionario, y después de cada ronda se decide si las respuestas constituyen un acuerdo. Si no existe dicho acuerdo, se vuelve a reformular la pregunta utilizando las respuestas anteriores gracias a un moderador, que se encarga de distribuir las preguntas y recoger las respuestas.

La hipótesis es que al reformular la pregunta con las respuestas recopiladas, los expertos considerarán los argumentos del resto y modificarán sus respuestas también, buscando como resultado un acuerdo común. Estas rondas con las preguntas reformuladas acaban mediante un timeout o cuando todos los expertos han dado su respuesta.

En este paper se explora así el uso de este método para sistemas multi-agente (MAS) basándose en la posibilidad de que estos se beneficien de su uso. Esta posibilidad se debe a que el método se adapta bien a las situaciones en las que las preguntas pueden tener respuestas distintas y a que se cumplen los requisitos necesarios:

- Preguntas que se puedan expresar de tal forma que los expertos puedan analizarlas y responderlas.
- Debe haber múltiples posibles respuestas.

- Dada una respuesta válida, esta debe poder dar una información, extraída de forma automática, que genere una nueva pregunta.

3.2.3. Fusión de modelos predictivos mediante método de ponderación local

En este paper [19] se habla de una investigación que se realizó utilizando redes neuronales y el método "Locally Weighted Models", método que, según los autores de este documento, consigue reducir la varianza, aunque no la parcialidad. Esto consigue resultados más adaptados al problema en concreto, aunque no mejora su capacidad frente a posibles nuevos casos.

El foco principal es crear un sistema de fusión, en el que primero se atiende a la creación de modelos y su selección y después al diseño del mecanismo de la fusión, basándose en crear modelos diversos que usen distintos datos de entrenamiento y con diferentes configuraciones.

El entrenamiento se llevó a cabo seleccionando una porción de forma aleatoria para crear un pequeño dataset de verificación variado, y el resto se separa de tal forma que queden distintos subsets, tantos como modelos se quieren entrenar.

Para la fusión, se tiene en cuenta el rendimiento local de los modelos creados para un caso en concreto sobre el que se quiere hacer la predicción. Para esto, hay que fijarse en cuáles de los modelos entrenados lo fueron con casos similares al de prueba; estos casos son escogidos mediante una fórmula que define un hiper-rectángulo del cual extraer dichos casos.

Después se evaluaría el rendimiento local de cada modelo, calculando entonces el error de predicción de cada uno de forma individual, tanto el error medio como el error medio absoluto; el primero indica la parcialidad del modelo y el segundo indica la precisión global del mismo.

Finalmente, se utilizan dos métodos de evaluación para fusionar los modelos: uno atendiendo únicamente a la precisión y otro considerando, además, la parcialidad. Los resultados indicaron una mejora notable en la media del error absoluto, de entre 20-40 %, concluyendo así que existe una mejora de rendimiento de los modelos. Esto indica que esta fusión de modelos tienen un efecto relevante, obteniendo así un metamodelo que mejora el rendimiento obtenido anteriormente de un único modelo.

3.2.4. Métodos de consenso basados en técnicas de Machine Learning para la detección de fitoplancton

En este paper [18] se refleja la investigación realizada sobre un dataset de detección de plancton marino, utilizando métodos de consenso variados sobre modelos de Machine Learning.

Se consideraron varias aproximaciones para fusionar modelos generados con algoritmos variados, desde SVMs y Random Forest a algoritmos lineales y técnicas de boosting. Fi-

nalmente, se consideraron seis distintos métodos de consenso, evaluados con diferentes datasets del mismo ámbito (detección de presencia o ausencia de plancton en muestras marinas) y varios de otros ámbitos de acceso libre.

Se encontró que para el 72 % de los datasets probados, los métodos de consenso ofrecían una mejora en el error respecto al uso de modelos individuales. En concreto, uno de los métodos mostró unos mejores resultados: este tenía en cuenta la precisión individual de los modelos y les asignaba una relevancia acorde en la votación.

El estudio se centra en hacer una aproximación en particular hacia la mejora de clasificadores binarios para estudiar la presencia de ciertas especies de plancton. En el paper [18], se ilustran los resultados obtenidos en conjunto, reflejando el número de victorias (para ese dataset era quien obtenía un rendimiento mayor) que obtuvo cada algoritmo y método de consenso.

3.3. Interpretabilidad de modelos

La intepretabilidad de los modelos generados con algoritmos de Machine Learning empeora cuanto más complejo sea el algoritmo, aunque también suelen tener unos mejores resultados. Así, algunos algoritmos, como el del Árbol de decisión, son especialmente fáciles de interpretar, pero otros, como los SVM, son especialmente complicados, más cuanto mayor sea la dimensionalidad del problema en cuestión.

Tras realizar una búsqueda de diversos papers [21] [22] al respecto de este problema, no se ha encontrado ningún método concreto mediante el que se pueda explicar de forma simple y directa el resultado proporcionado por un modelo o incluso el propio modelo, aunque sí se hacen unas aproximaciones relevantes.

En el paper [23], se explica cómo el ámbito de la salud se ve reforzado por el uso de modelos, pero sin embargo resultan difícilmente explicables los resultados, y es un área en el que encontrar la contribución concreta de cada variable y entender el por qué de un resultado es importante. También se expone brevemente que la forma de hacer interpretable el modelo es proporcionar al usuario cierta información, como podrían ser las contribuciones de cada clase al resultado o la probabilidad de cada clase. En este caso de investigación en concreto, se la posible mortalidad, ilustrado en la siguiente imagen:

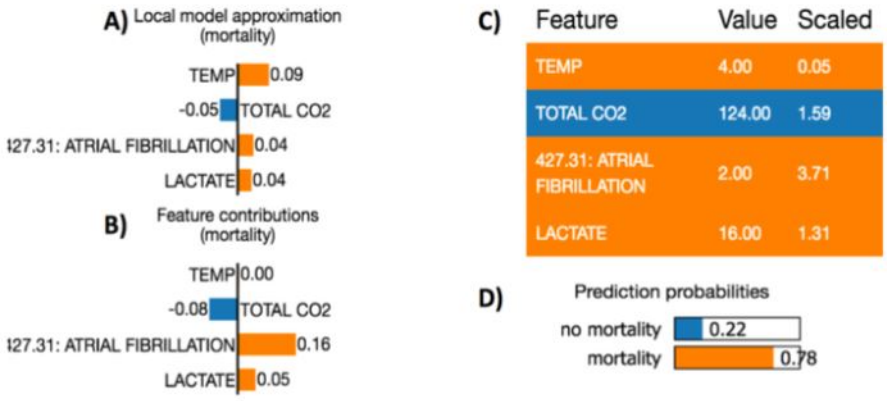

Figura 3.10: Datos proporcionados al usuario al utilizar un modelo complejo [21]

En este paper [22], se discute sobre los dos principales problemas a la hora de medir la interpretabilidad de los modelos: el uso, en la literatura al respecto, de distintos términos (algunos de ellos referidos a prácticamente el mismo concepto, como puede ser 'comprensible' y 'entendible' y otros a un concepto más concreto y distinto, como podría ser la usabilidad) y la poca distinción entre la interpretabilidad de los modelos en sí y de sus representaciones, siendo ambas medidas válidas para medir la interpretabilidad.

Las comparaciones para la interpretabilidad de modelos o representaciones dependen así de las heurísticas y a encuestas de usuario. Sin embargo, según el autor, no hay suficiente información al respecto de esto último como para inferir una relación entre ambas cosas y concluir que, efectivamente, las heurísticas modelan la comprensión del usuario.

3.4. Entorno de Jupyter Notebook

Jupyter notebook es un entorno de trabajo que, entre otros, permite desarrollar código en python, integrando junto al mismo otros recursos, tales como imágenes, gráficas o texto, lo cual es muy útil en análisis y machine learning, entre otros campos de la informática y las matemáticas.

Para empezar a utilizar este entorno, hay que tener instalado Python: este se puede descargar desde la web [24] o podemos realizar un *pip install* desde nuestro cmd. Una vez instalado, se trata de actualizar al paquete más reciente con *pip install –upgrade pip*. El siguiente paso es instalarse jupyter notebook con *pip install jupyter* y al terminar la instalación, estaría listo para ejecutar con el comando *jupyter notebook*, lo cual nos abriría el workspace en la ubicación desde la que lo ejecutemos en la consola.

Se crea un nuevo Python 3 notebook desde la interfaz de jupyter, tal y como se puede ver en la Figura 3.11.

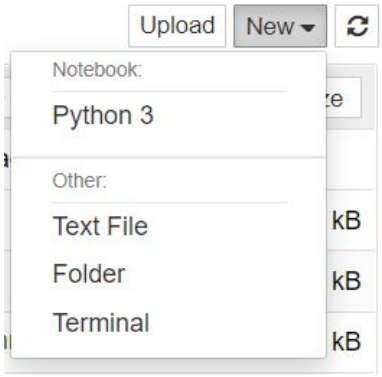

Figura 3.11: Crear un nuevo notebook

Este abre una nueva pestaña con la barra de menú y una celda vacía, donde se pueden escribir una o varias líneas de código y comentarios, tal y como puede verse en la Figura 3.12. También se puede escribir texto, de modo que al ejecutar la celda no se ejecuta como código, sino que hace que el texto funcione como markdown. Una celda puede marcarse como markdown desde el desplegable de la barra de herramientas.

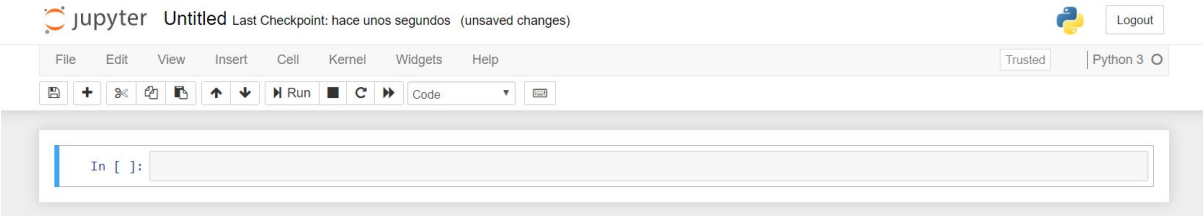

Figura 3.12: Imagen de un nuevo notebook con una celda

Para ejecutar se puede seleccionar la celda y pulsar el botón *Run* de la barra de herramientas, o utilizar los atajos *Shift + Enter* para simplemente ejecutar o *Alt + Enter* para ejecutar la celda y crear una nueva debajo para seguir codificando. Para saber si una celda se ha ejecutado hay que mirar su cabecera, si aparece **In**[] aún no se ha ejecutado, si aparece **In**[*] la ejecución está en proceso, y si aparece un número entre los corchetes ya ha terminado de ejecutar y el número indica el orden de ejecución de las celdas.

Se muestra con el ejemplo de la figura 3.13, como se aplica código en lenguaje Python, y se realiza un bucle que cuenta y escribe por pantalla en que vuelta del propio bucle se encuentra.

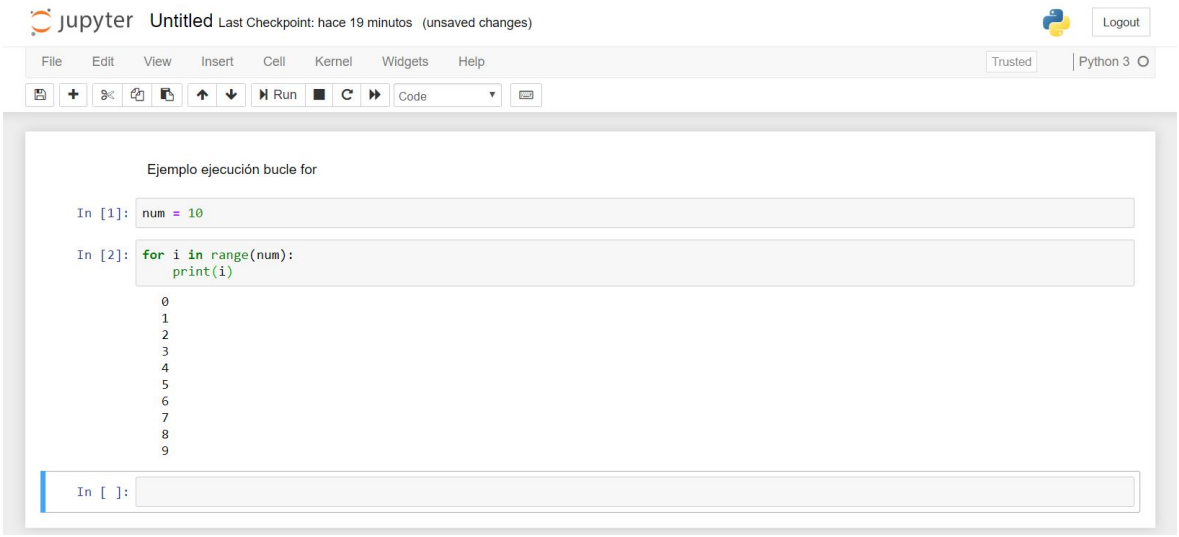

Figura 3.13: Codificando en notebook

3.4.1. Librerías relevantes

Ciertas librerías pueden importarse para realizar de forma más simple funciones que ya están definidas por dicha librería, las más importantes para este trabajo son pandas, numpy y sklearn.

3.4.1.1. Librería Pandas

La librería Pandas es ampliamente utilizada en análisis de datos, y aporta algunas estructuras de datos tales como los DataFrame (estructuras similares a las tablas de las bases de datos relacionales) o las Series (arrays unidimensionales con indexación etiquetada, similares a los diccionarios).

Proporciona además funciones para cargar los datos desde formatos como hojas Excel o CSV a estructuras como los DataFrame, lo que permite hacer todo tipo de operaciones sobre ellos, facilitando la fase de limpieza de datos.

3.4.1.2. Librería NumPy

Esta librería proporciona un mayor soporte para vectores y matrices, a parte de funciones matemáticas para operar con este tipo de elementos, son una buena alternativa a las listas, ya que son rápidos, fáciles de usar y permiten realizar cálculos a través de arrays completos.

3.4.1.3. Librería Sklearn

Gracias a la librería de Sklearn, podemos generar modelos de machine learning de forma rápida e intuitiva. Además, dichos modelos tienen un amplio abanico de posibilidades de configuración mediante los parámetros que crean los objetos modelo.

3.4.1.4. Librería Pickle

Esta librería es de gran utilidad para la persistencia de modelos. Con ella, podemos guardar los modelos generados en el formato *.pkl*, y posteriormente cargarlos con otra función de esta misma librería.

3.4.1.5. Tkinter

La librería de Tkinter es una librería ampliamente utilizada para la creación de interfaces en el lenguaje Python. Con ella se ha implementado el apartado visual de la aplicación. (Capítulo 6).

3.5. Conclusiones

Puesto que este método del voto por mayoría es habitual y ampliamente usado, incorporarlo parecía un paso evidente, especialmente teniendo en cuenta que es probable que se requiera un voto de la mayoría y, de ser posible, sobre el que se tenga cierto control. Es por ello que se ha implementado este método con una función en la que se recogen las respuestas de todos los modelos listados y se efectúa un recuento de votos. Respecto a la creación de modelos, se decide utilizar aquellos de aprendizaje supervisado y siempre y cuando los resultados tengan una precisión superior al 70 %.

Algunos de los posibles problemas que se manifiestan con este método son los casos en los que dos o más de los posibles resultados coinciden en número de votos. En este punto hay que tomar una decisión a la hora de implementar una solución que tenga en cuenta dicha posibilidad, y finalmente, en esta implementación, como se explicará en el capítulo correspondiente 5.1.1, se planteó recoger en un vector auxiliar las posibles modas.

Puesto que se han podido encontrar antecedentes de mejora al utilizar sistemas de consenso, se decide implementar uno propio basado en el método Delphi. Más en concreto, se plantea utilizar información de los propios modelos, aunque esto limita las posibilidades a utilizar aquellos algoritmos que no sean de caja negra y que además puedan proporcionar algún tipo de información una vez creados. Con ello se cumplen los requisitos para poder aplicar Delphi.

Respecto a la interpretabilidad de los modelos se opta por mostrar trazas del proceso de los votos, de forma que el usuario pueda tener acceso a la información de cómo se desarrolla el consenso y las respuestas individuales de cada modelo y cómo estas cambian en caso de hacerlo.

Capítulo 4

Preparación de los datos

En el Machine Learning, uno de los procedimientos iniciales y básicos es preprocesar y limpiar los datasets para crear los distintos modelos [25], consiguiendo además así reducir su complejidad. Antes de empezar a modelar, es importante que los datos tengan la misma forma, de modo que hay que analizar los casos que contienen campos vacíos, entre otras cuestiones, a lo largo de esta fase. Este proceso ayuda también a entender el dataset y tratar mejor con el mismo.

El procedimiento a seguir puede ser diferente en función de la fuente y formato de los datos: dependen del tipo de fichero que se utiliza para guardar los datos y qué tipos de datos son los que contiene, pero las tareas principales de preprocesamiento de datos no se ve afectada por ello. Las tareas principales son [25]:

- **Limpieza de datos**: eliminar o sustituir datos sin valor, eliminar datos con ruido y outliers.
- **Transformación de datos**: normalizar los datos para reducir la dimensionalidad y el posible ruido.
- **Reducción de datos**: realizar sampling de los datos relevantes o reducir el número de atributos a únicamente los más relevantes.
- **Discretización de datos**: convertir datos continuos a discretos o categóricos.

En este procedimiento, donde se busca construir modelos optimizados para que su rendimiento sea el máximo posible, el dataset debe facilitar dicho cometido. En muchas ocasiones, se mapean los datos para que sea posible utilizarlos independientemente del algoritmo. Es posible, a su vez, que sea interesante normalizar o estandarizar los datos en determinadas circunstancias para entrenar con ciertos algoritmos. Como el proceso de mapeado y normalización se hace extremadamente costoso cuanto mayor sea el número de variables localizadas en los datos, un paso importante es tratar de restar complejidad, eliminando variables desde el principio, pero solo aquellas cuyo aporte a las predicciones sea bajo.

Para la realización de este trabajo se han utilizado distintos datasets con dimensiones y características diferentes. El objetivo era encontrar una serie datasets, comprender los datos que estos representan y crear unos modelos utilizando distintos tipos de algoritmos

de aprendizaje automático con ellos. Más adelante se encuentra la explicación de los datasets. Para llevar a cabo la limpieza, se utilizarán distintos notebooks de Jupyter para cada dataset.

Los datasets son de distintos ámbitos, pertenecen a diversas fuentes y contienen tipos de datos diferentes. El objetivo de esto es ver cómo se adapta el método de consenso planteado a dichos ámbitos y cómo responde ante clasificaciones binarias y multiclase. Este último es el principal motivo de utilizar varios dataset en lugar de únicamente uno, ya que es probable que el comportamiento en clasificaciones binarias o multiclase sea distinto, y que por tanto los resultados fueran mejores o peores para unos datos en concreto, haciendo que se pudiera llegar a conclusiones erróneas. Si bien el uso de diferentes datasets no garantiza poder dar una respuesta definitiva, sí permite ver una cierta tendencia.

La limpieza de datos se ha realizado sobre estos dataset atendiendo a cada uno de forma diferente y siempre que se ha podido. Esta parte es una práctica común, ya que muchos dataset contienen casos con valores indeterminados que acaban produciendo ruido en los modelos.

Algunos de los dataset utilizados venían divididos en dos directamente de las fuentes: uno con datos para entrenar y otro con datos para testear los modelos creados. Es común que los datos se dividan en entrenamiento para un 70-80 % de los casos disponibles y el resto se utilicen para validación y testing. Otros en cambio no venían divididos, y se indica en su sección propia si es así y cómo se ha realizado la división.

En este caso, era de vital importancia probar el consenso sobre los mismos, con lo cual en algunos no se ha prestado tanta atención a cómo de precisos eran los modelos o se han seleccionado por su sencillez en este aspecto.

Finalmente se han utilizado dos datasets multiclase y dos binarios:

- **Masas anómalas en mamas:** dataset que recoge datos sobre pacientes con masas sospechosas en las mamas, siendo casos de mamas malignas y benignas.
- **Reconocimiento de dígitos escritos a mano:** dataset multiclase donde se recogen datos sobre dígitos escritos a mano de diversas personas.
- **Datos de Telescopio MAGIC Gamma:** dataset binario con instancias para detección de partículas gamma en la atmósfera.
- **Tipos de vino:** dataset multiclase que representa diversas muestras de vino de tres tipos diferentes.

A continuación se detallan aspectos importantes sobre los diferentes dataset de forma más concreta y se especifican las transformaciones, si proceden, ejecutadas sobre los mismos para crear los modelos.

4.1. Dataset de masas anómalas en mamas

Estos dataset tratan sobre masas y calcificaciones cancerígenas en mamas. [26] [27] [28].

Su estudio se basa en imágenes reales de mamografías, de las que se obtienen los distintos datos de entrada analizando dichas imágenes y se generan tanto el dataset de entrenamiento como el de pruebas con esta mecánica.

Las filas son los pacientes y las columnas los datos recopilados, los cuales son:

- **patient_id:** el id asignado a cada paciente, que puede repetirse ya que quizás se detecten varias masas sospechosas, o se listen desde distinta perspectiva (indicado en image view).
- **breast_density:** indica la densidad de la mama.
- **left or right breast:** variable para indicar si la anomalía se localiza izquierdo o derecho.
- **image view:** Craniocaudal (CC) o Medio-lateral oblicuo (MLO).
- **abnormality_id:** número de anomalía para ese paciente. Esta variable aparece porque es necesario en casos de pacientes con múltiples anomalías, necesitándose etiquetar cada una de forma individual.
- **abnormality_type:** indica si la anomalía es debido a una masa o calcificación. En este caso, masas y calcificaciones vienen en datasets separados, y se decidió trabajar sobre el primero, por lo que siempre tiene como valor 'mass'.
- **mass_shape:** forma de la masa.
- **mass_margins:** estado del borde de la masa.
- **assessment:** evaluación BI-RADS de los radiólogos.
- **pathology:** indica si la masa es benigna o maligna.
- **subtlety:** clasificación de los radiólogos de que tan complicado era identificar y analizar la anomalía en la imagen.
- **image file path:** url a la imagen completa.
- **cropped image file path:** url a la imagen recortada a la zona de la masa.
- **ROI mask file path:** url a la imagen con filtro de píxels.

Para estos datos, primero se realiza una limpieza de aquellos que no son necesarios o no aportan suficiente información para poder realizar las predicciones (Figura 4.1). Una vez que se obtienen los datos necesarios, estos se mapean para poder trabajar con ellos fácilmente.

- Para comenzar, se eliminan las columnas con las URLs que indican los enlaces de las imágenes, ya que previamente se habían extraído los datos relevantes en distintas columnas del dataset y son los que vamos a utilizar; los path a esas imágenes ya no son necesarias para nuestro proceso.

- Por otra parte, todos los pacientes con uno o más datos desconocidos (NaN) también son retirados, ya que los datos son abundantes de por sí y aunque podrían sustituirse esos valores por otros utilizando métodos tales como la media, el valor más frecuente u otros ampliamente utilizados en la práctica, se ha optado por eliminarlos.

- Para cada paciente, pueden darse distintos puntos de vista (CC y MLO) de las masas, y varias masas sospechosas, y la intención es hacer predicciones respecto a una masa en concreto y no respecto al paciente, hemos eliminado la columna de patient_id. Al mismo tiempo, como el abnormality_id está relacionado directamente con el anterior, en este punto es una variable que no aporta al resultado final desde este planteamiento.

- Al utilizar este dataset separado del dataset de calcificaciones, la columna de abnormality_type tiene siempre el mismo contenido, por lo que no aporta nada a la predicción y puede prescindirse de este valor.

- Por último, es indiferente que las masas pertenezca a una mama izquierda o derecha, y en las pruebas no cambiaba los resultados de forma significativa, por tanto también queda eliminada esta columna también para reducir la dimensionalidad del problema.

Estos pasos se realizan utilizando las funciones drop y dropna de la librería Pandas, y pueden verse reflejados en la Figura 4.1 para los datos de entrenamiento y test:

```
In [ ]: dataTrain.drop(columns = ["image file path", "cropped image file path", "ROI mask file path"], inplace = True)
        dataTest.drop(columns = ["image file path", "cropped image file path", "ROI mask file path"], inplace = True)
        dataTrain.dropna(inplace = True)
        dataTest.dropna(inplace = True)
        dataTrain.drop(columns = ["patient_id", "abnormality type", "left or right breast"], inplace = True)
        dataTest.drop(columns = ["patient_id", "abnormality type", "left or right breast"], inplace = True)
```

Figura 4.1: Step Data Cleaning

En la Figura 4.2 se aprecian los 5 primeros casos del Test y los valores que contienen para cada columna antes de realizar el mapeo. Tras los cambios anteriores, el resultado obtenido es el de la misma:

	breast_density	image view	mass shape	mass margins	assessment	pathology	subtlety
0	4	CC	IRREGULAR	SPICULATED	5	MALIGNANT	5
1	4	MLO	IRREGULAR	SPICULATED	5	MALIGNANT	5
2	2	CC	ROUND	CIRCUMSCRIBED	4	MALIGNANT	4
3	2	MLO	ROUND	ILL_DEFINED	4	MALIGNANT	4
4	3	CC	ROUND	OBSCURED	0	BENIGN	2

Figura 4.2: Mass Test Dataset

Para poder trabajar de manera más eficaz con dichos datos con todos los algoritmos, se mapean sus valores al tipo numérico, de tal forma que estos datos son modificados como sigue a continuación:

- "image view": CC : 0,
  MLO: 1
- "mass shape": IRREGULAR : 0,
  OVAL : 1,
  LOBULATED : 2,
  ROUND : 3,
  ARCHITECTURAL_DISTORTION : 4,
  IRREGULAR-ARCHITECTURAL_DISTORTION : 5,
  ASYMMETRIC_BREAST_TISSUE : 6,
  FOCAL_ASYMMETRIC_DENSITY : 7,
  LYMPH_NODE : 8,
  LOBULATED-LYMPH_NODE : 9,
  LOBULATED-IRREGULAR : 10,
  OVAL-LOBULATED : 11,
  IRREGULAR-ASYMMETRIC_BREAST_TISSUE : 12,
  OVAL-LYMPH_NOD": 13,
  ROUND-OVAL : 14,
  IRREGULAR-FOCAL_ASYMMETRIC_DENSITY : 15,
  LOBULATED-ARCHITECTURAL_DISTORTION : 16,
  LOBULATED-OVAL : 17,
  ROUND-IRREGULAR-ARCHITECTURAL_DISTORTION : 18,
  ROUND-LOBULATED : 19

- "mass margins": CIRCUMSCRIBED : 0,
  SPICULATED : 1,
  ILL_DEFINED : 2,
  OBSCURED : 3,
  MICROLOBULATED : 4,
  CIRCUMSCRIBED-ILL_DEFINED : 5,
  ILL_DEFINED-SPICULATED : 6,
  OBSCURED-ILL_DEFINED : 7,
  CIRCUMSCRIBED-OBSCURED : 8,
  OBSCURED-SPICULATED : 9,
  OBSCURED-ILL_DEFINED-SPICULATED : 10,
  MICROLOBULATED-ILL_DEFINED : 11,
  MICROLOBULATED-ILL_DEFINED-SPICULATED : 12,
  MICROLOBULATED-SPICULATED : 13,
  CIRCUMSCRIBED-MICROLOBULATED : 14,
  CIRCUMSCRIBED-OBSCURED-ILL_DEFINED : 15,
  CIRCUMSCRIBED-MICROLOBULATED-ILL_DEFINED : 16,
  OBSCURED-CIRCUMSCRIBED : 17,
  CIRCUMSCRIBED-SPICULATED : 18
- "pathology": BENIGN : 0,
  MALIGN : 1

Tras todo el proceso, los datos obtenidos están listos para modelar, y el resultado es el reflejado en la Figura 4.3:

	breast_density	image view	mass shape	mass margins	assessment	pathology	subtlety
0	4	0	0	1	5	1	5
1	4	1	0	1	5	1	5
2	2	0	3	0	4	1	4
3	2	1	3	2	4	1	4
4	3	0	3	3	0	0	2

Figura 4.3: Mapped Mass Test Dataset

4.2. Dataset para el reconocimiento de dígitos escritos a manos

El segundo dataset con el que se ha trabajado contiene datos sobre la escritura de dígitos del intervalo 0-9 a mano. Este dataset se ha elegido por su sencillez a la hora de conseguir entrenar modelos, ya que todos los datos están transformados y no requieren ningún tipo de mapeo, sin que haya además valores desconocidos. Estos se tomaron utilizando una tablet y un stylus, y se guardaron las coordenadas y presión con la que se tomaron los casos, aunque posteriormente solo se utilizarían las coordenadas.

La recopilación [29] se llevó a cabo con 44 personas diferentes escribiendo cada uno un total de 250 dígitos. Los primeros diez intentos de cada persona fueron borrados, ya que se quería que los sujetos se adaptasen primero a escribir sobre la tablet para no tomar

casos que produjesen ruido.

A la entrada original se le aplicó una normalización para evitar errores de escalas, ya que unos números pueden haber sido escritos más grandes que otros. Además, se representarán finalmente los dígitos como vectores generados con un resampling espacial que representa, mediante un único número, un punto de coordenadas (x,y). Por tanto, los datos de entrada son puntos del espacio representados como un entero en el intervalo 0-100, además del número real al que representan, con un total de 17 variables de entrada.

Los datos estaban ya divididos de forma homogénea, tanto en representación (no varían los casos de ejemplo de unos dígitos a otros por mucho) como en cuanto a entrenamiento y testing, por lo que este dataset ha podido ser utilizado de forma directa sin tener que aplicar ninguna transformación, análisis de las features o separación de los datos para generar los modelos.

En la figura 4.4 a continuación se refleja en la tabla los datos, siendo las filas casos particulares y los columnas las distintas variables sin nombre:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	47	100	27	81	57	37	26	0	0	23	56	53	100	90	40	98	8
1	0	89	27	100	42	75	29	45	15	15	37	0	69	2	100	6	2
2	0	57	31	68	72	90	100	100	76	75	50	51	28	25	16	0	1
3	0	100	7	92	5	68	19	45	86	34	100	45	74	23	67	0	4
4	0	67	49	83	100	100	81	80	60	60	40	40	33	20	47	0	1

Figura 4.4: Pen Digits Test Dataset

Este dataset, se utiliza en el proyecto por su sencillez para mostrar ejemplos, ya que el consenso se puede interpretar de manera muy sencilla mediante los cambios en las opiniones de los modelos. Cabe destacar, además, que el porcentaje de acierto de los modelos para este dataset son elevados.

Las confusiones en las predicciones suelen darse entre números cuyos trazos son parecidos, y a lo largo de las predicciones podemos observar cómo los modelos cambian de opinión entre ciertos conjuntos de números que, también a simple vista y debido a que la escritura es manual y de diversas personas, puedan parecerse, (1, 7), (3, 5), (3, 8), (8, 0). Es fácil encontrar ejemplos donde los modelos no encuentran en primera instancia una unanimidad, pero finalmente acaban encontrándola cuando son casos de números similares en la escritura.

Los altos porcentajes, se deben en parte al buen contenido en casos de entrenamiento, estos tienen más de 700 ejemplos de entrenamiento para cada número a predecir

Distribución de los datos Figura 7.1.
Como ya se mencionó, es un dataset homogéneo, y los resultados son bastante buenos gracias a ello.

Clase	Instancias
0	780
1	779
2	780
3	719
4	780
5	720
6	720
7	778
8	719
9	719

Cuadro 4.1: Número de ejemplos en el dataset de entrenamiento de pendigits

4.3. Dataset del telescopio MAGIC Gamma

Este dataset hace una predicción binaria sobre el registro de partículas gamma o hadron.

Los datos se basan en el método Monte Carlo, para simular el registro de partículas gamma de alta energía en un telescopio atmosférico terrestre utilizando la técnica de la imagen, el telescopio observa los rayos gamma aprovechando la radiación emitida por la partículas cargadas producidas dentro del haz de luz electromagnético, iniciado por los gamma y desarrollándose en la atmósfera.

Esta radiación de onda, visible a niveles de Ultra-violeta, se filtra en la atmósfera y se registra, lo que permite la construcción de los parámetros del dataset. Dependiendo de la energía del gamma primario, se permite discriminar estadísticamente estas señales causadas por rayos gamma, y las hadronic iniciadas por rayos cósmicos en la atmósfera superior.

Este dasaset [29] se incluye para poder estudiar y realizar conclusiones sobre los modelos que se han creado con distintos ejemplos de datos. Sus variables son número reales, a excepción de la columna a predecir, la cual está constituida por los caracteres **g (gamma signal)**, y **h (hadron background)**. Como el objetivo es estudiar el algoritmo aplicado sobre los modelos, este dataset es adecuado por no necesitar de un gran esfuerzo para su limpieza y mapeo de datos. Además, no contiene ejemplos con atributos vacíos que necesiten atención. Sin embargo, sí ha sido necesaria una división en datos de entrenamiento y datos de testeo.

El punto negativo es que no es un dataset homogéneo, debido a que los casos de señal gamma, duplican a lo de hadrón:

- g = gamma (signal): 12332 examples
- h = hadron (background): 6688 examples

Respecto a la información de los atributos del dataset, se explican a continuación:

- **fLength**: eje mayor de la elipse [mm]

- **fWidth**: eje menor de la elipse [mm]
- **fSize**: log_{10} de la suma del contenido de los píxeles
- **fConc**: ratio de la suma de los 2 píxeles más altos sobre fSize [ratio]
- **fConc1**: ratio de la suma del píxel más altos sobre fSize [ratio]
- **fAsym**: distancia del píxel más alto al centro [mm]
- **fM3Long**: tercera raíz, del tercer momento sobre el eje mayor [mm]
- **fM3Trans**: tercera raíz, del tercer momento sobre el eje menor [mm]
- **fAlpha**: ángulo del eje mayor con vector al origen [deg]
- **fDist**: distancia del origen al centro de la elipse [mm]
- **class**: gamma (signal), hadron (background)

Para tener listo el dataset y que se puedan realizar pruebas con él, se necesita dividir entre casos de test y de entrenamiento, para ello, se aplica la técnica de Cross Validation para ver la precisión estimada que podemos esperar de una división aleatoria, dividiendo el dataset en un 70/30 %.

```
X_train, X_test, y_train, y_test = train_test_split(data, target, test_size=0.3, random_state=1)
```

Figura 4.5: Cross validation para crear los datasets de Test y Entrenamiento

Una vez dividido el dataset, se realiza el mappeo de los atributos de 'class', para transformarlos de caracteres a enteros:

```
dTrain["class"] = dTrain["class"].map({"g" : 0, "h" : 1})
dTest["class"] = dTest["class"].map({"g" : 0, "h" : 1})
```

Figura 4.6: Mapeo del atributo 'class' en datasets de Test y Entrenamiento

Tras estos pasos, el dataset está listo para su uso. En la Figura4.7 se pueden observar algunos de los casos del dataset, cuyas variables se explicaron anteriormente:

	fLength	fWidth	fSize	fConc	fConc1	fAsym	fM3Long	fM3Trans	fAlpha	fDist	class
0	24.3356	12.2325	2.5105	0.4660	0.2731	8.4297	16.1253	5.8151	81.6645	142.9840	1
1	21.1241	9.6714	2.2227	0.5928	0.3623	-3.9600	15.5160	-10.1431	11.1686	176.8790	0
2	59.9960	19.7337	2.9248	0.2580	0.1397	22.8397	55.9957	10.4966	7.8247	257.0960	0
3	43.2557	12.0269	2.5205	0.3831	0.1976	-63.8415	16.1869	-6.1091	9.6970	299.4060	0
4	98.9198	15.6593	2.7513	0.2258	0.1335	-17.7075	-79.4405	-13.3692	67.3458	202.8971	1

Figura 4.7: Ejemplo final de los primeros casos de los datos de test

4.4. Dataset de clasificación de vinos

Este dataset [29] contiene muestras de tres vinos diferentes de la misma región de Italia, pero procedentes de distintas cultivares. Concretamente, las variables de las que constan los casos de ejemplo son características (como puede ser la intensidad del color) o el tanto por ciento de los componentes químicos presentes en cada una.

Todos los atributos son continuos, y no existe ningún caso de valor desconocido, por lo cual este dataset no requiere ninguna transformación previa necesaria en estos aspectos para trabajar con el mismo, siendo este el principal motivo de selección del mismo.

El dataset es pequeño, pero no hay un gran desbalance en la representación de ninguna de las clases, la cual consta de la siguiente forma Figura 4.2:

Tipo	Instancias
1	59
2	71
3	48

Cuadro 4.2: Número de ejemplos en wine dataset

No consta ninguna información más al respecto del proceso de obtención de los atributos ni se justifica la diferencia en representación de los tres tipos de vino, pero sí consta que ofrece unos resultados satisfactorios, siendo este otro motivo por el cual se selecciona el dataset.

Las variables que conforman el dataset, aunque no son todos los resultados del análisis químico, son las siguientes:

- **Alcohol:** cantidad de alcohol en tanto por ciento que contiene la muestra.
- **Malic acid:** cantidad de ácido málico presente.
- **Ash:** cantidad de ceniza.
- **Alcalinity of ash**: alcalinidad de dicha ceniza.
- **Total phenols:** número total de ácidos fenoles en la muestra.
- **Flavanoids:** porcentaje de compuestos flavanoides.
- **Nonflavanoid phenols:** porcentaje de fenoles no flavanoides.
- **Proanthocyanins:** porcentaje de este tipo de fenoles.
- **Color intensity:** intensidad del color de la muestra.
- **Hue:** otro indicador que matiza el color por el tono.
- **OD280/OD315 of diluted wines:** mediciones sobre contenidos proteicos.
- **Proline:** aminoácidos de la muestra.

4.5. Conclusiones

Como conclusiones, cabe destacar que el proceso ha sido muy diferente para el primer dataset, puesto que sus datos estaban en bruto al haberse obtenido un dataset con datos extraídos directamente de las imágenes. Es por ello que el trabajo para el mismo ha sido mayor, ya que requería la limpieza que se requiere en un caso real.

Para el resto de datasets, al haberse obtenido de forma que este proceso ya había sido aplicado anteriormente por los autores, y al encontrarse así en la fuente [29], la limpieza de datos necesaria no es tan pesada como en el primero. Esto explica que sus secciones y la cantidad de pasos realizados en su limpieza son menores. La razón de esto es que se requerían datasets que permitiesen crear modelos de forma más ágil que el primero, ya que era fundamental tener disponibles modelos de distintos problemas.

En cuanto a los tipos de dataset, para la búsqueda se tiene en cuenta si son binarios o multiclase, para aplicar pruebas en ambos tipos y poder obtener un análisis de resultados más elaborado y fundamentado.

Capítulo 5

Diseño e implementación de los algoritmos de consenso

En este capítulo se explican los diferentes **tipos de consenso** que se han implementado durante el proyecto. Los métodos se han implementado de cero y han sido ejecutados en distintas configuraciones y con diferentes problemas para conocer hasta qué punto resulta de interés aplicar este tipo de algoritmos junto a una serie de modelos.

5.1. Implementación de métodos

En esta sección se explican los métodos implementados, es decir, el método de voto por mayoría y los consensos de una y dos vueltas, exponiendo desde su planteamiento hasta la lógica detrás de los mismos. Además se ilustra, de forma esquemática, el funcionamiento paso a paso.

5.1.1. Voto mayoría

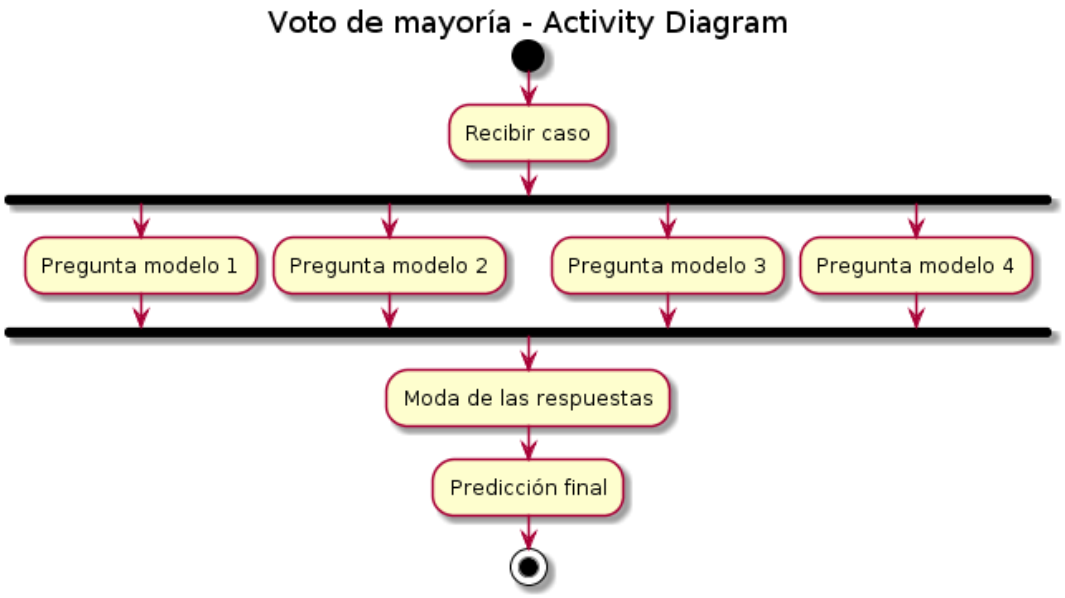

Figura 5.1: Diagrama de actividad de la función del voto por mayoría

El voto por mayoría, tal y como se puede observar en el diagrama de actividad de la Figura 5.1, funciona de forma sencilla: se recibe un caso a probar, y este es pasado para su evaluación a una lista de modelos. Estos modelos generarán una salida cada uno, y una vez obtenidos todos, se calcula la moda (valor más repetido), que será la respuesta final.

La implementación de este algoritmo se ha llevado a cabo mediante el uso de funciones de algunas librerías, las cuales son Counter, de la librería *Collections* y la función Groupby de la librería *itertools*.
El procedimiento implementado es el siguiente: el algoritmo realiza un conteo mediante mediante la creación de una lista de tuplas, donde aparecen los *valores encontrados* y su *frecuencia*. A continuación, estos valores se agrupan en un vector de listas mediante la mencionada función de Groupby, agrupando así los valores de las tuplas que tienen la misma frecuencia. Por tanto, los valores de la primera lista del vector de listas son los valores (o único valor) más repetidos. Por decisión de implementación, el valor escogido es siempre el primer valor de la primera lista del vector. El código puede verse en la Figura 5.2:

```
for j in self.models:
    aux.append(j.results[i])
    data += "    Response from " + str(j) + " is " + str(j.results[i]) + "\n"

possibleModes = []

freqs = groupby(Counter(aux).most_common(), lambda x:x[1]) # group most_common output by frequency
possibleModes.append([val for val,count in next(freqs)[1]])  # pick off the first group (highest frequency)
possibleModesAsArray = np.array(possibleModes) #as array we can check if there are more than one mode

mode = possibleModes[0][0]

data += "\n    Majority vote result: " + str(mode) + '\n\n\n'
```

Figura 5.2: Código principal del voto por mayoría

Esta implementación se ha llevado a cabo en lugar de utilizar funciones de bibliotecas por tres motivos principales:

- La librería *statistics* es la librería que tiene una función mode, la cual calcula la moda de un vector. Un problema que se encontró es que en caso de haber más de una posible moda, se lanzaba una excepción.
- El segundo motivo es consecuencia directa de la primera, y es que a pesar de poder tratar dichas excepciones, resultaba más interesante poder tener más control sobre la función en sí y que pudiese adaptarse a posibles futuras necesidades. Además, esta implementación se ha llevado a cabo de forma que se evita el problema mencionado anteriormente.
- El tercer motivo es que quería asegurarse el hecho de que se mantuviese una prioridad en el orden, ya que los modelos están ordenados por sus métricas de precisión (de forma que primero se encuentran los modelos más precisos) dentro de la lista de modelos. Debido a esto, nos interesa que se mantenga el orden para dar prioridad de forma implícita a las respuestas los modelos de mayor precisión. Durante la implementación se tiene especial atención a este detalle,y se ha corroborado que los métodos mencionados mantienen dicho orden.

Tal y como se comentó en 3.2.1, es un método ampliamente utilizado y de suma simpleza, que además de actuar como un pseudo metaclasificador en sí mismo, era necesario para realizar la implementación de los algoritmos de consenso que se habían planteado. Por tanto, su inclusión no solo favorece la variedad de algoritmos proporcionados, sino que sirve de base para los consensos implementados.

También se debe mencionar que el hecho de que los modelos estén ordenados de forma decreciente (respecto a las métricas de precisión) es una decisión tomada en base a la práctica, en cuyas pruebas se comprobó que ofrecía mejores resultados que un orden aleatorio o creciente: esto se debe a la mencionada prioridad implícita a los modelos más precisos.

5.1.2. Consenso con modelo KNN y el método Delphi

La implementación de los modelos se ha implementado teniendo como base el método Delphi, cuya explicación podemos encontrar en 3.2.2, realizando por tanto unas rondas de votaciones y discusiones. Es necesario definir un moderador y unas nuevas preguntas para generar dichas discusiones. Esto se ha conseguido definiendo un **modelo Máster**, cuya finalidad es actuar como moderador y proporcionar nuevas preguntas.

Se ha optado porque dicho modelo sea del tipo KNN, pudiendo configurar a su vez el número de vecinos (nuevas preguntas) que se devuelven por el mismo, lo cual proporciona una cierta posibilidad de adaptabilidad a los diferentes datasets según la distribución de los datos que presenten.

Para entender por qué se ha utilizado un modelo como moderador y por qué concretamente un KNN, hay que analizar el planteamiento hecho durante el desarrollo. La primera necesidad es una realimentación de algún tipo a los modelos una vez han dado ya una respuesta ante un caso de prueba: puesto que los modelos son deterministas, de nada serviría probar dándoles de nuevo el mismo caso, ya que proporcionarían la misma respuesta.

Para solucionar las situaciones de no unanimidad, había que introducir un mecanismo de votación (apoyado en el de la mayoría, para aquellos casos donde no se alcance unanimidad poder dar una respuesta válida) que introdujese ciertas variaciones que permitiesen que la salida pueda ser diferente. Ante esto se pueden plantear varias alternativas:

- Una de las opciones sería variar ligeramente el caso de prueba. Si se hiciesen pequeñas variaciones en los datos, es posible que la respuesta sea distinta para algún modelo y que el resultado final cambie. Esto tiene más riesgos, puesto que sin ser expertos en la materia que trate el dataset en concreto, no se puede saber si se introduce un caso real ni que realmente sea similar al proporcionado. Tampoco existe la posibilidad de indicar a un modelo que atienda con mayor relevancia a unas u otras variables, aunque bien es cierto que pueden abstraerse con ciertas técnicas las variables más relevantes, y no sería lógico hacerlo según un criterio impreciso y esperar un resultado satisfactorio de ello.
- La segunda opción pasa por encontrar casos similares al de prueba. Para ello, sería necesario algún mecanismo para conseguir dichos casos, y no sería muy lógico

intentar concebir un criterio para discernir si un caso es similar a otro dado y buscarlo dentro de nuestros propios datos, ya que esto último sería ineficiente. Ante esta situación, las opciones pasaban por acudir a los propios modelos, quienes ya estaban entrenados con dichos datos. Aunque la mayoría de algoritmos generan un modelo de caja negra, como las redes neuronales, sí hay algunos algoritmos que son capaces de devolver un feedback una vez entrenados mediante alguna función propia, como lo son por ejemplo los clusters de K-Means, capaces de devolver el centroide de cada cluster, o los modelos KNN, capaces de devolver sus vecinos.

En este caso, la opción considerada más segura y viable era la segunda: utilizar los modelos entrenados para conseguir un feedback que permitiese introducir nuevos casos como realimentación. Además es un planteamiento más general, y no atiende al tipo de dataset o de datos, por lo que es más sencillo que puedan ser probados frente a estos métodos.

En primera instancia, tanto K-Means como KNN eran capaces de ello, ya que en ambas posibilidades las instancias de los datasets están ubicadas en el hiperespacio de forma cercana a un caso de prueba concreto. También esas instancias son aquellas que tienen unas características más similares al mismo, siendo los candidatos más válidos para este planteamiento. Sin embargo, había que tener un par de consideraciones en cuenta:

- La primera era que, al menos en los experimentos realizados, no resultaban mejores los modelos de K-Means que aquellos generados con KNN.
- La segunda, que K-Means solamente podía devolver un único caso por cada cluster con el API actual, el centroide, mientras que KNN es capaz de devolver hasta K vecinos en función de cómo haya sido creado el modelo.

Finalmente se optó por la vía de utilizar KNN por estos dos motivos y porque el hecho de que haya, potencialmente, más de un caso vecino (y no únicamente uno, el correspondiente al clúster más cercano).

Hay que destacar que no es directo obtener vecinos, ya que la función encargada devuelve tanto la distancia desde el caso de prueba inicial a cada caso vecino como el índice que ocupa cada uno de dichos vecino en el set de entrenamiento. Por tanto, **para obtener los datos de dichos vecinos, es necesario acudir a dicho set, utilizado originalmente para entrenar el modelo**.

En el siguiente fragmento de código, se puede ver el funcionamiento del algoritmo para obtener casos vecinos. Los parámetros de entrada corresponden a la instancia del caso del que se quieren obtener los vecinos y al número de los mismos deseados. Después se devuelve una lista con los casos vecinos devueltos en la lista indices, la cual contiene un número K de vecinos, siendo K el número de vecinos con el que se entrenó a ese clasificador KNN en concreto. El código en cuestión puede verse en la Figura 5.3

```
def giveNeighbors(self, case, neig):
    distances, indices = self.knn[0].model.kneighbors(case.reshape(1, -1))
    sampleList = []
    for i in range(int(neig)):
        sampleList.append(self.df[indices[0][i]])
    return sampleList
```

Figura 5.3: Código para la obtención de casos vecinos

Una vez hecho esto, se puede generar un nuevo subset de datos de tamaño **0 <n <= K**, donde K sería el número de vecinos con los que el modelo KNN se entrenó.

También hay que tener en cuenta que la métrica que se indique como parámetro al KNN será vital a la hora de generar unos resultados satisfactorios con ambos, para que las distancias entre casos semejantes se calcule con mayor precisión; un modelo generado de forma incorrecta o con un dataset con ruido podría provocar que los vecinos no sean necesariamente casos similares al de prueba y por tanto generaría más errores.

5.1.2.1. Consenso de una vuelta

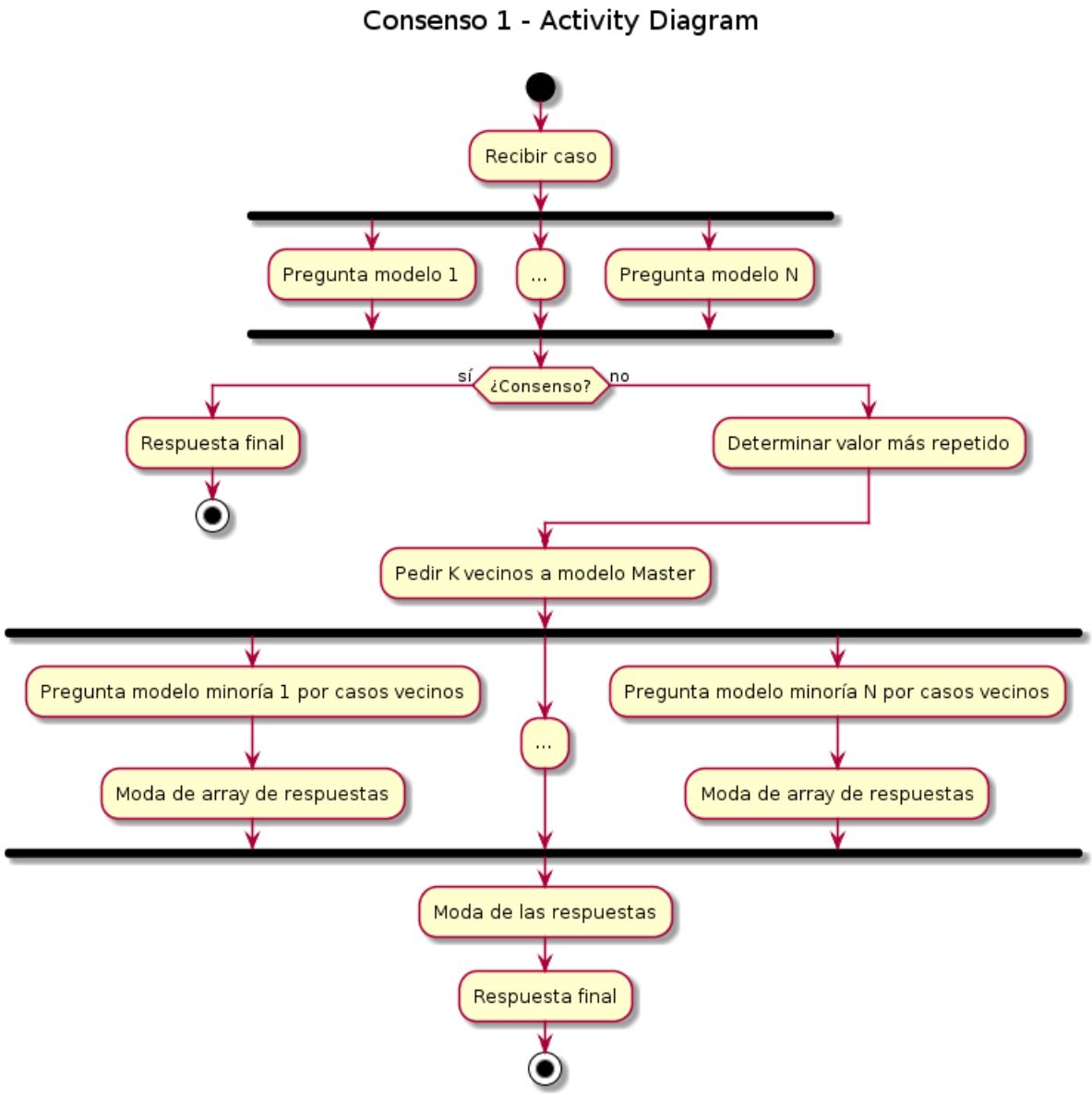

Figura 5.4: Consenso de modelos utilizando Consenso de una vuelta

Tal y como se puede observar en la Figura 5.4, el procedimiento del consenso de una vuelta tiene varias fases diferenciadas, utilizando en las últimas de ellas el algoritmo del voto de la mayoría implementado, explicado en la sección 5.1.1.

En el primer compás de este algoritmo se recibe un caso de prueba y se pregunta a la lista de los modelos por el mismo. A continuación, se comprueba si existe un consenso total en las respuestas, es decir, si hay una **unanimidad**. En caso de que este sea el caso, el grueso del algoritmo no se ejecuta, puesto que ya se ha llegado a un estado que no puede mejorarse. En cambio, en caso de que no haya consenso, se calcula el **valor más repetido**, es decir, la moda entre las respuestas de los modelos y también se realiza una petición al Máster para que proporcione K casos vecinos, tal y como puede verse en el

siguiente fragmento de código de la Figura 5.5:

```
if(not self.consensus(array)):
    neighbors = self.giveNeighbors(case, neig)
    mode = self.modesArray(array)
    resp += "\nThere is no consensus. We will ask the models that didn't answer " + str(mode) +
                " what they think about the " + str(neig) + " neighbors of this case\n"
```

Figura 5.5: Obtención de casos vecinos en situación de no consenso y realimentación a la minoría

Tras esto, se pregunta a los modelos cuya respuesta no fue el valor más repetido por sus resultados para los K vecinos. Con esto cada modelo genera un array de K respuestas, calculando acto seguido la moda del mismo para obtener una nueva respuesta de cada modelo. Una vez obtenidas esas nuevas respuestas de los modelos, estas se cambian en el array original de respuestas, calculando de nuevo la moda del mismo y tomando dicha respuesta como final. En la imagen a continuación de la Figura 5.6 puede verse el código de dicha ronda de consenso:

```
if(not self.consensus(array)):
    neighbors = self.giveNeighbors(np.array(casoPrueba), neig)
    mode = self.modesArray(array)
    data += "\n\n    There is no consensus. We will ask the models that didn't answer "
    + str(mode) + " what they think about the " + str(neig) + " neighbors of this case\n\n"
    response = []
    for i in self.models:
        if(array[self.models.index(i)] != mode):
            for j in neighbors:
                response.append(i.execute(j)[0])
            data += "        Answer from " + str(i) + " for case's neighbors is "
            + str(response) + "\n"
            array[self.models.index(i)] = self.modesArray(response)
            response.clear()
    data += "\n        Responses after:" + str(array) + "\n"
    mode = self.modesArray(array)
    data += "\n    Final answer: " + str(mode) + "\n\n"
else:
    data += "\n    Final answer: "+ str(array[0]) + "\n\n"

data += "-----------------------------------------------\n\n"
self.emitter.do_some_stuff(data)
```

Figura 5.6: Código de una ronda de consenso

Puede darse el caso de que no haya unanimidad e incluso de que haya varias modas, en cuyo caso se opta por devolver, también en este algoritmo como ya se hacía en el voto de la mayoría (explicado en la sección 5.1.1), la primera de las modas.

5.1.2.2. Consenso de doble vuelta

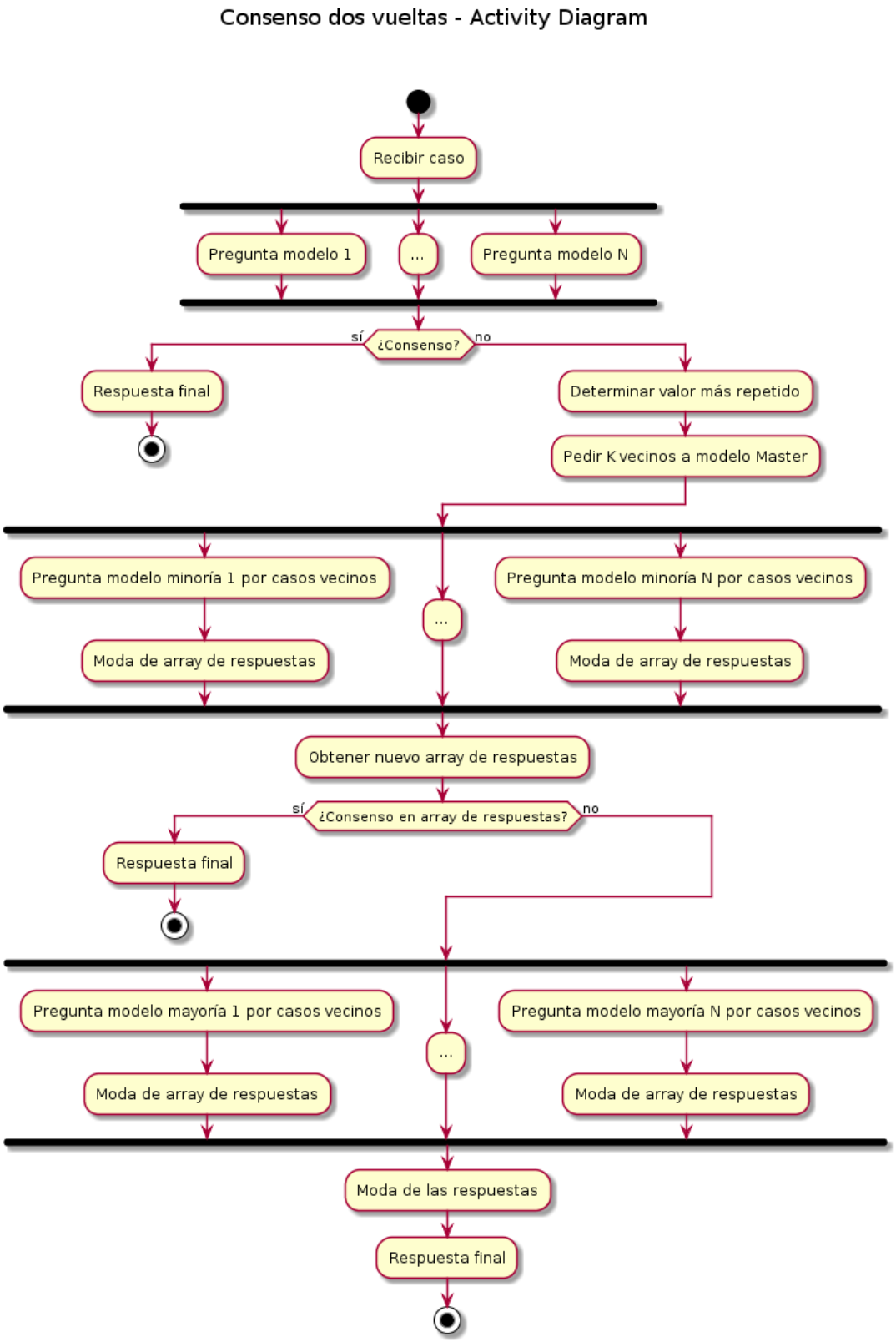

Figura 5.7: Consenso de modelos utilizando Consenso de doble vuelta

Tal y como puede observarse en la Figura 5.7, la complejidad de este algoritmo es la mayor de entre los implementados y tiene un mayor número de pasos, lo cual hace que potencialmente este algoritmo tenga una carga computacional elevada.

Es una ampliación lógica del anterior consenso de una sola vuelta, siendo todo el procedimiento común entre ambos, pero continuando con una nueva ronda de preguntas donde el anterior algoritmo terminaba. Esto puede observarse comparando ambos diagramas de actividad.

Continuando donde acababa la primera ronda de preguntas, una nueva ronda se lleva a cabo en caso de que no exista unanimidad. Puesto que anteriormente se preguntó a aquellos modelos cuya respuesta no había sido el valor más repetido, en este caso preguntamos a aquellos que sí respondieron el valor más repetido para ver si sus respuestas cambiarían. Sería el equivalente a preguntar a los pertenecientes a la mayoría si realmente están seguros sobre su respuesta, instándolos a analizar los mismos casos analizados en la ronda anterior por el resto de modelos. En la siguiente imagen de la Figura 5.8 se puede ver el código correspondiente:

```
if(not self.consensus(array)):
    resp += "\nThere is no consensus after the first round. We will ask the models that answered the MRV now..." + "\n"
    for i in self.models:
        if(i.results[0] == mode):
            for j in neighbors:
                response.append(i.execute(j)[0])
            resp += "Answer from "+ str(i) +" for case's neighbors is " + str(response) + "\n"
            array[self.models.index(i)] = self.modesArray(response)
            response.clear()
    resp += "The answers from the models (2 rounds) are: " + str(array) +
            " , so the final answer is " + str(self.modesArray(array)) + "\n"
    resp += "-------------------------------------------" + "\n"
```

Figura 5.8: Realimentación a los modelos de acuerdo con la mayoría en una segunda vuelta

En este punto, se habrá acabado preguntando a todos los modelos (ya que anteriormente se pregunta a aquellos que estaban en la minoría) por los casos proporcionados por el modelo KNN que actúa como Máster. Al no haber más posibles rondas, el algoritmo acaba en este punto de la misma forma que lo hacía el de una vuelta: si hay unanimidad, se indicará esta situación, y en caso de no haberla, se devolverá el valor más repetido, devolviendo el primero de ellos en caso de haber varios candidatos en empate.

Capítulo 6

Interfaz y caso de uso

Puesto que la programación se ha realizado en Jupyter Notebook, todo el uso e implementación de los algoritmos se había llevado a cabo mediante un Notebook, desde el cuál se utilizaban los algoritmos. Si bien no es un entorno particularmente complicado mediante el que utilizarlo, es ciertamente incómodo cuando la utilización es reiterada y requiere una configuración previa, tal como la selección de datos, modelos y algoritmo a ejecutar. Para que el usuario pueda aplicar fácilmente los algoritmos de consenso sobre los modelos creados, cargar estos últimos y obtener las predicciones, se implementa una interfaz capaz de realizar una comunicación entre el usuario y la aplicación más amigable. Además, esto permite que usuarios sin conocimiento suficiente para utilizar un Notebook de Jupyter tengan accesible el uso de la herramienta.

Los resultados obtenidos y los mensajes que informan del estado de las acciones realizadas se muestran por pantalla en un área de texto, para que el usuario esté informado en todo momento de si sus acciones se están ejecutando y del resultado de las mismas. Además, una vez se han cargado los modelos, simplemente con seleccionar el algoritmo, el número de vecinos(en caso de consenso) y pulsar el botón de Run Datasets-Test cases, se obtiene la predicción, tanto para casos simples (necesario rellenar el área correspondiente y pulsar Run Input Case), como para evaluar el rendimiento de los algoritmos sobre un tipo de dataset con una lista de casos (necesario haber cargado el dataset de testeo).

El aspecto de la pantalla principal de la interfaz es el siguiente:

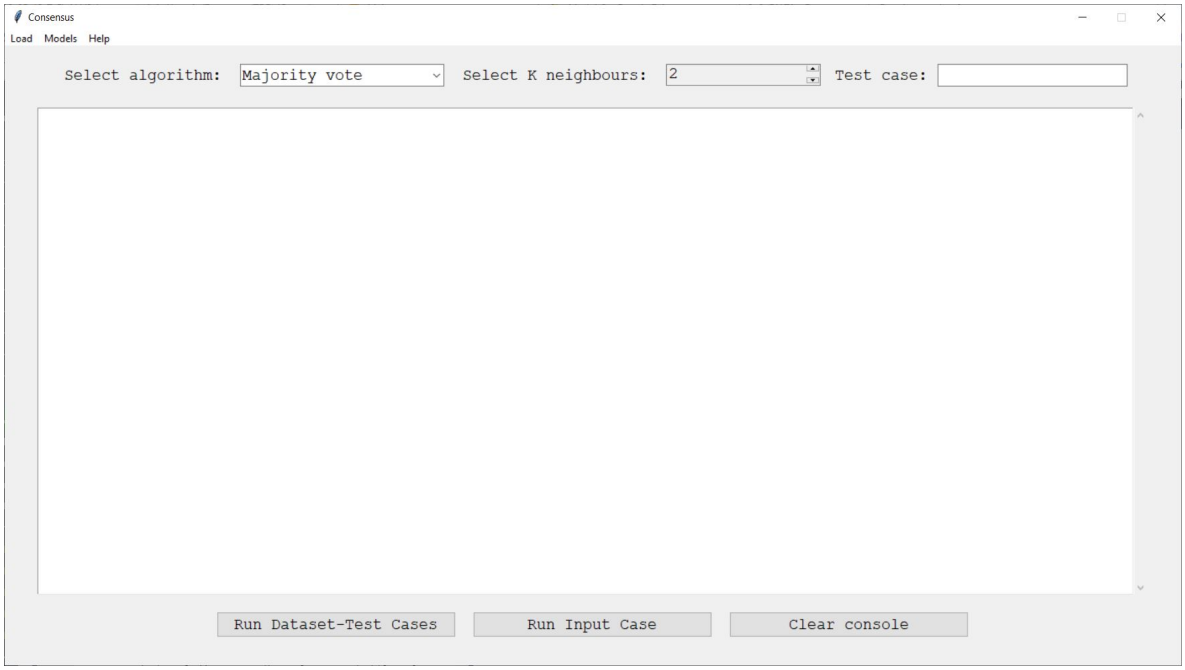

Figura 6.1: Imagen del estado inicial de la GUI

A simple vista, en la ventana principal se pueden apreciar la barra de menú, los elementos en la parte superior relativos a la configuración de la ejecución. En la zona central, se encuentra el campo de salida que informa al realizar las distintas acciones, y en la parte inferior, se encuentran los botones que ejecutan los tests o el caso simple del campo superior, además de un botón que hace una limpieza del texto de salida.

Desde este punto, podemos seguir los siguientes casos de uso identificados:

- Caso 1: carga de datos y modelos.
- Caso 2: ejecución de casos mediante los modelos y los algoritmos.

6.1. Caso de uso de carga de datos y modelos

A continuación, se muestra un ejemplo de ejecución de los algoritmos sobre unos datos de testeo. En caso de ejecutar algún tipo de consenso sin realizar correctamente la carga de los datos, se mostrará un mensaje de error como el siguiente mostrado en 6.2, indicando la falta de elementos necesarios para su correcto funcionamiento e invitando al usuario a visitar el menú de ayuda donde pueda encontrar unas instrucciones:

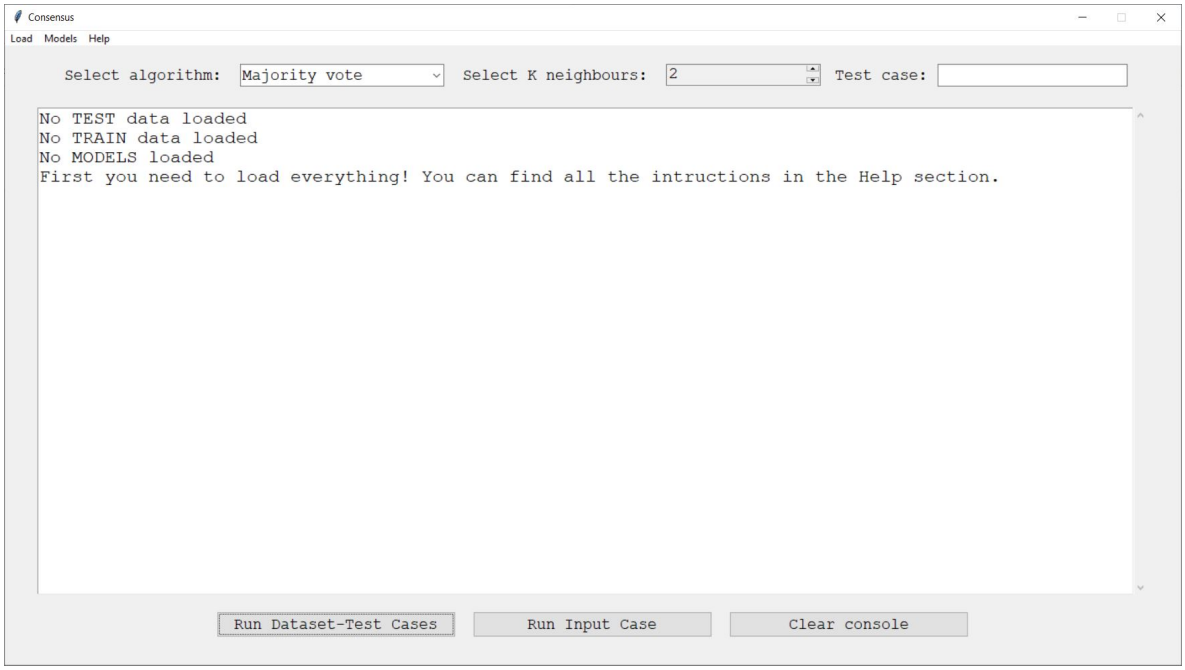

Figura 6.2: Indicación de falta de elementos por cargar

Para empezar, se carga el archivo de datos de tests desde la barra de menú (se pueden cargar los elementos en cualquier orden), se selecciona Load Test Dataset, como se ve en la Figura 6.3 y se realiza la carga de lo datos para los tests tal y como se ve en la Figura 6.4:

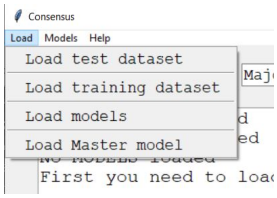

Figura 6.3: Carga de ficheros desde la interfaz

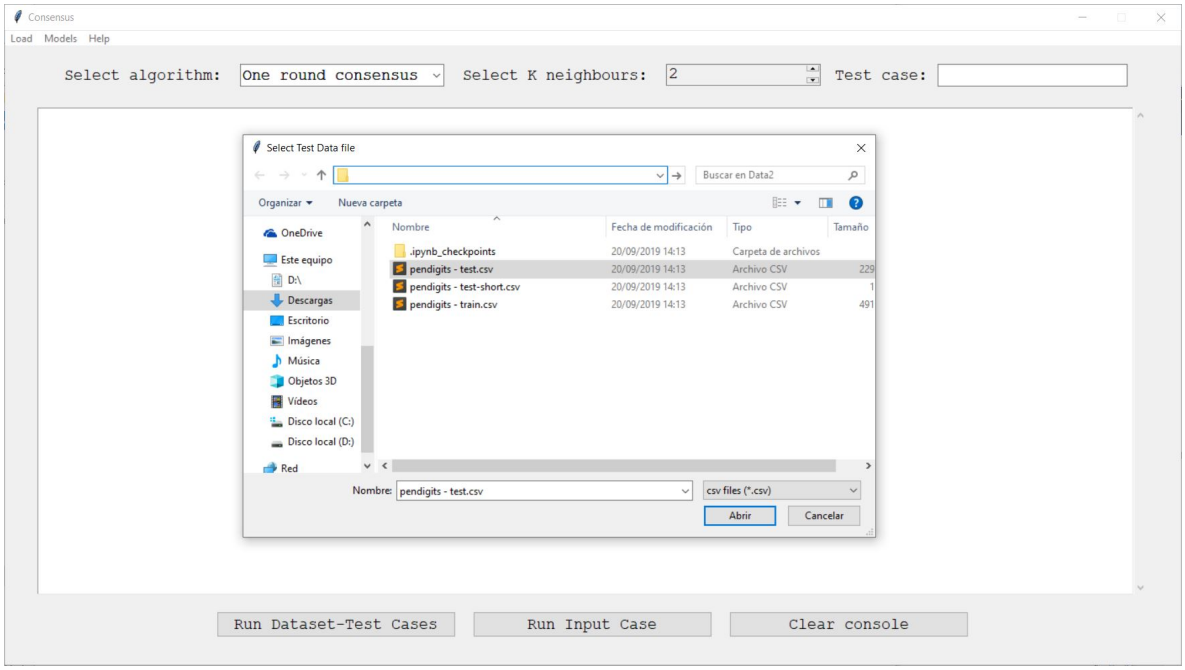

Figura 6.4: Carga de fichero de prueba

Utilizando el mismo proceso pero con Load Training Dataset, como se indica en la Figura 6.5, se carga el fichero con los datos de entrenamiento. En ambos casos solo se admiten ficheros de extensión csv.

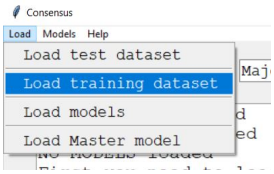

Figura 6.5: Carga de fichero de entrenamiento

Se debe indicar para ambos a continuación si los datasets tienen o no cabecera:

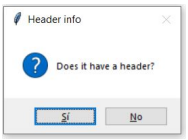

Figura 6.6: Diálogo para indicar si tienen cabecera los datasets

Una vez se han cargado los datasets correctamente, se procede a cargar los modelos de clasificación en la sección de load, se selecciona Load models como puede verse en la Figura 6.7, y se indica el directorio del que se desean obtener los modelos, tal y como se indica en la Figura 6.8.

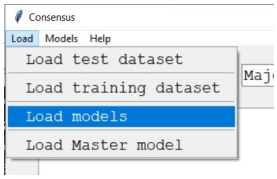

Figura 6.7: Carga de los modelos de clasificación

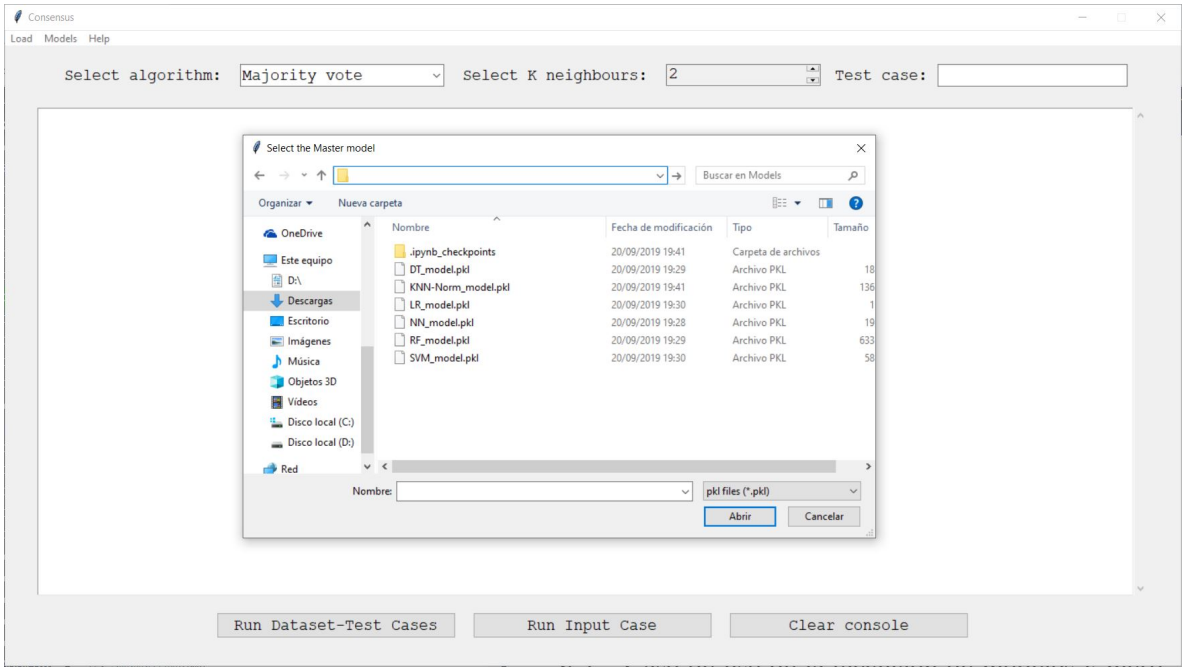

Figura 6.8: Selección del directorio con los modelos

6.2. Caso de uso de la ejecución de modelos y métodos de consenso

No es necesario un modelo Máster para ejecutar el consenso por mayoría (para más información al respecto, se puede leer en 5.1.1), por ello pulsando el botón de Run Datasets-Test cases con este algoritmo, se obtienen los resultados de los tests de prueba:

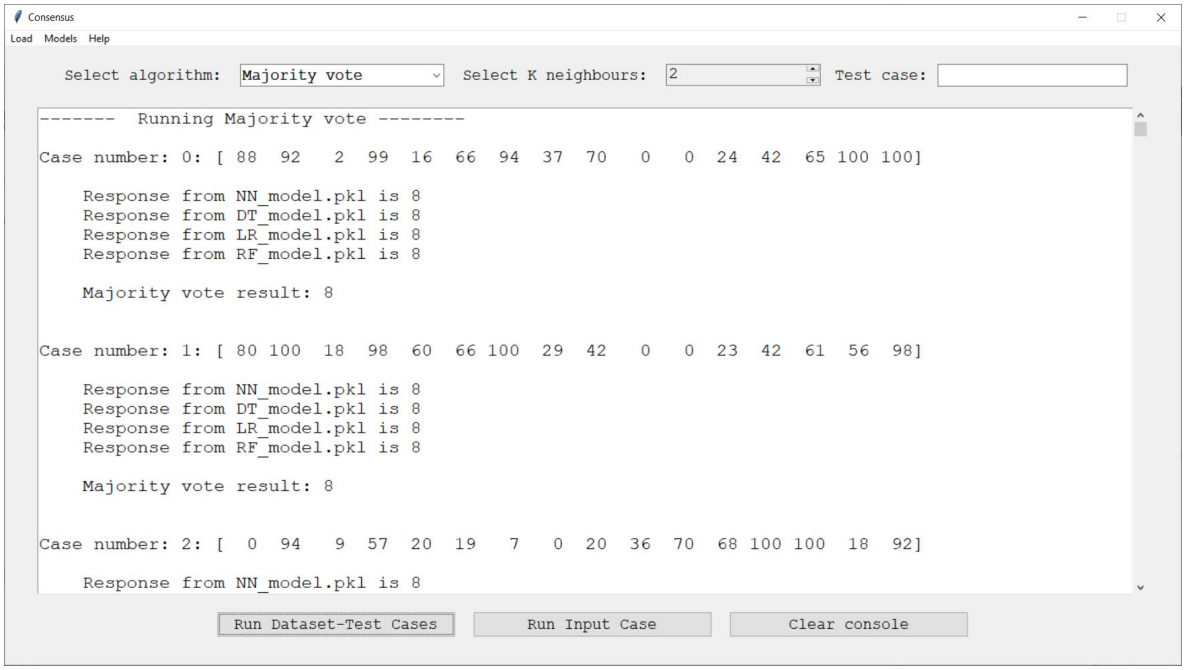

Figura 6.9: Voto por mayoría

En la Figura 6.9 podemos ver la salida correspondiente a la ejecución del voto de la mayoría para los datos de test. En la misma se indica el número correspondiente al caso, las variables del mismo y las respuestas para cada modelo, además del resultado final del algoritmo.

Para ejecutar los algoritmos de consenso, sí es necesario cargar el modelo Máster, por ello hay que seleccionar en la sección de Load el apartado de Load Master Model, como puede verse en la Figura 6.10, y una vez se despliega la ventana de ayuda, seleccionar un modelo del tipo KNN, como se indica en la Figura 6.11, para que la carga se realice correctamente.

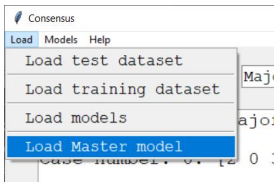

Figura 6.10: Carga del modelo Máster

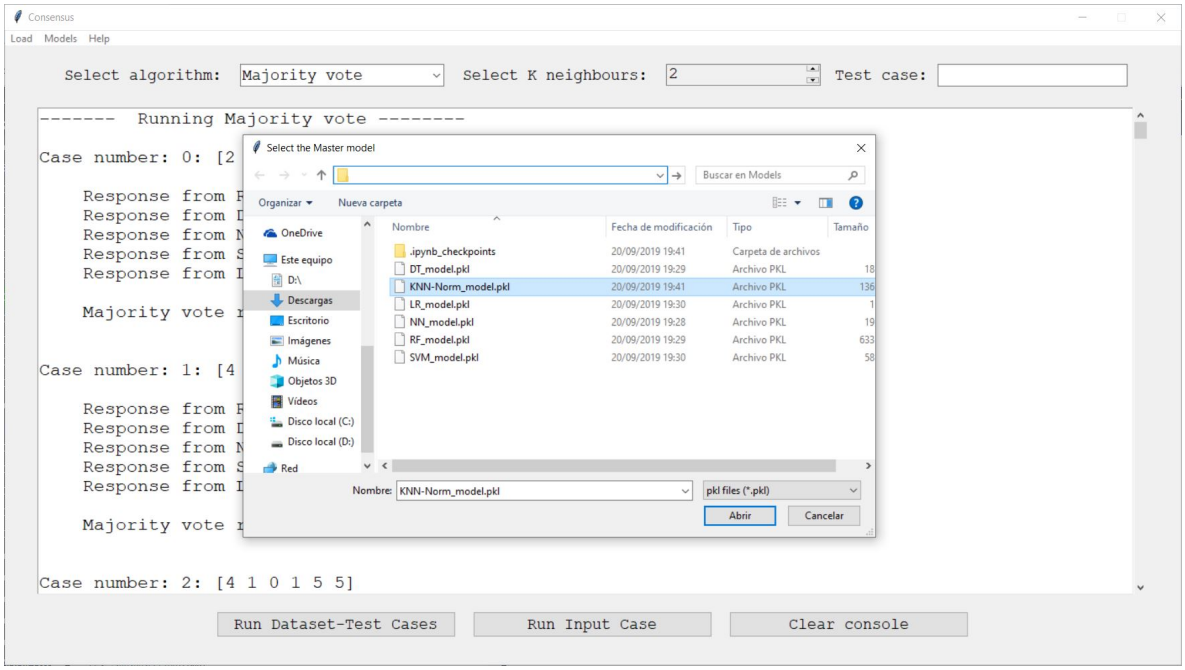

Figura 6.11: Selección del modelo Máster

Tras la correcta carga de datos, se ejecuta el algoritmo de consenso de una ronda (más información al respecto en la sección 5.1.2.1), seleccionándolo en el combobox de la parte superior izquierda de la interfaz. En este caso se selecciona One Round Consensus,y el valor de la K en el spinbox, y así indicar cuántos vecinos se utilizan para convencer a los modelos en sus nuevas respuestas. Primero se utiliza k = 2, cuya ejecución se muestra en la Figura 6.12, y luego para k = 4, cuya ejecución se muestra en la Figura 6.13 para comparar los distintos resultados del consenso.

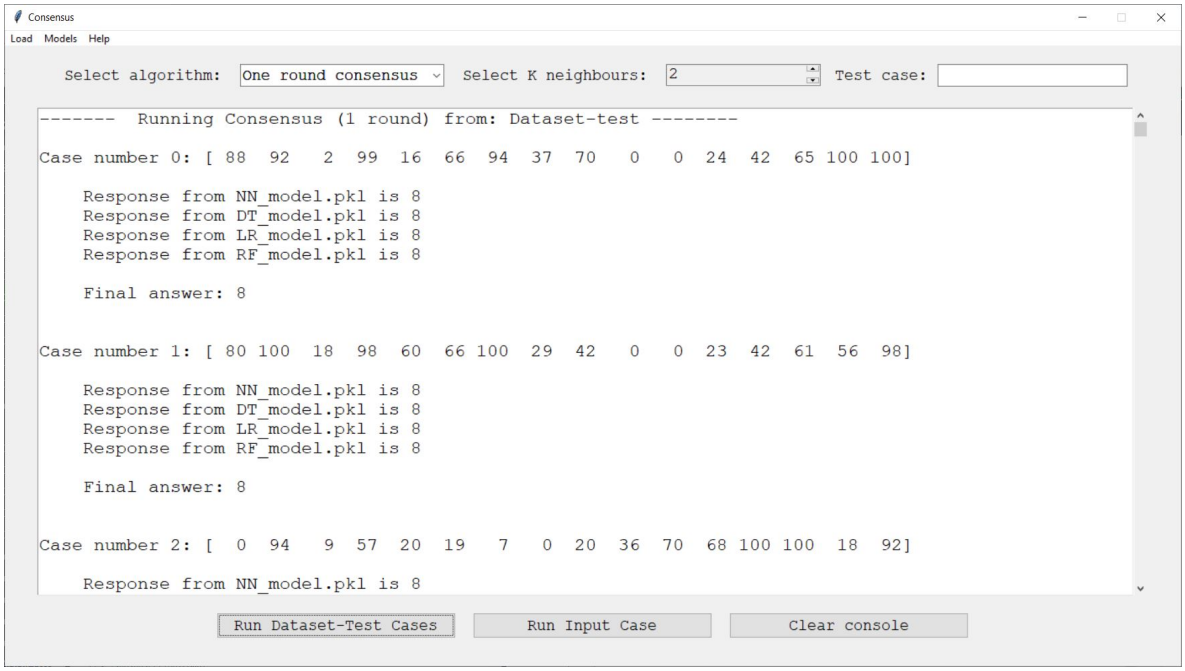

Figura 6.12: Resultados del consenso de una vuelta K = 2

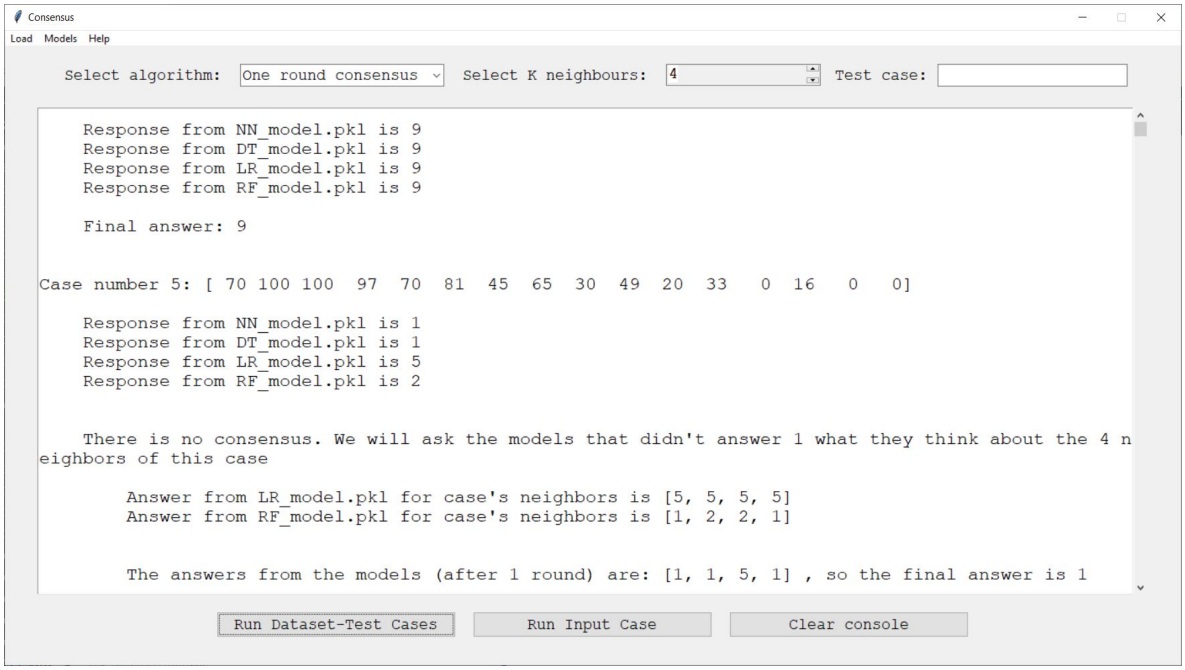

Figura 6.13: Resultados del consenso de una vuelta K = 4

En las anteriores imágenes se puede ver el feedback que ofrece la aplicación, mostrando al igual que en el voto de la mayoría el número de caso, los datos del mismo y las respuestas de cada modelo. Dependiendo de si hay consenso o no, se muestra la respuesta final, como puede verse en el Caso 4 de la imagen, o se pasa a la ronda de consenso, como ocurre en el Caso 5, donde se pregunta a los modelos cuya respuesta no ha sido el valor de moda por el número K de vecinos del spinbox de la parte superior. Puede a su vez observarse que el array de respuestas en el mencionado caso cambia tras la ronda de consenso.

También puede seleccionarse el algoritmo Two Rounds Consensus (más información puede encontrarse en la sección 5.1.2.2) en el combobox y probar distintos valores de K, en los ejemplos siguientes de la Figura 6.14 y la Figura 6.15 se muestran los resultados utilizando K = 2 y K = 4:

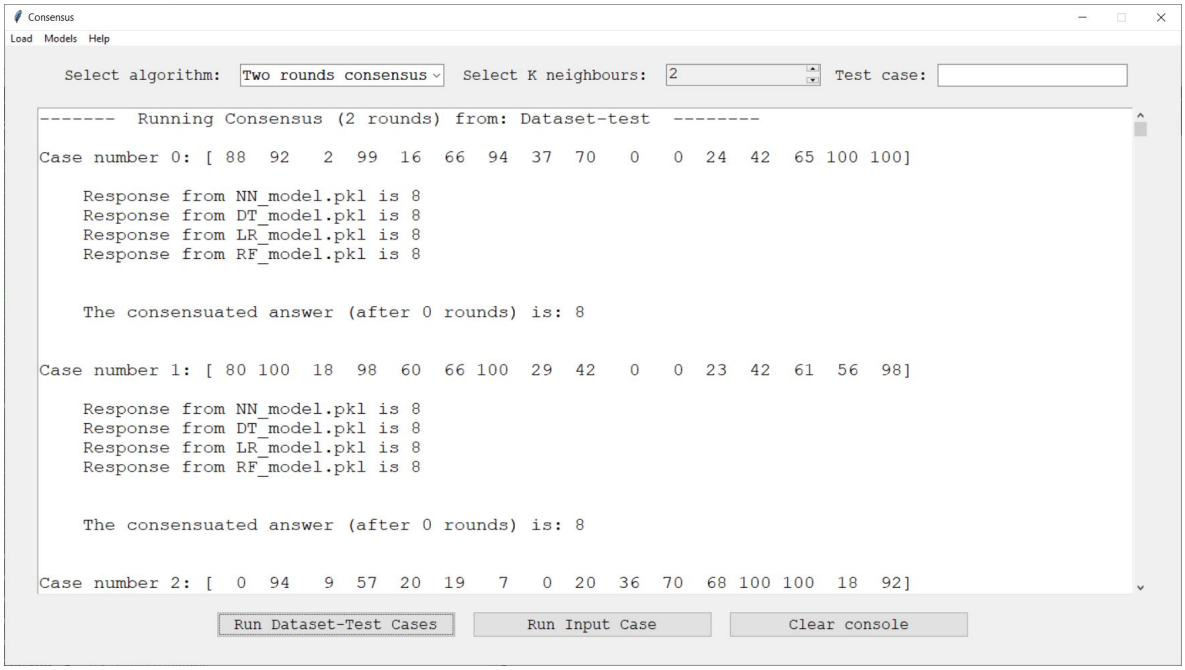

Figura 6.14: Resultados del consenso de doble vuelta K = 2

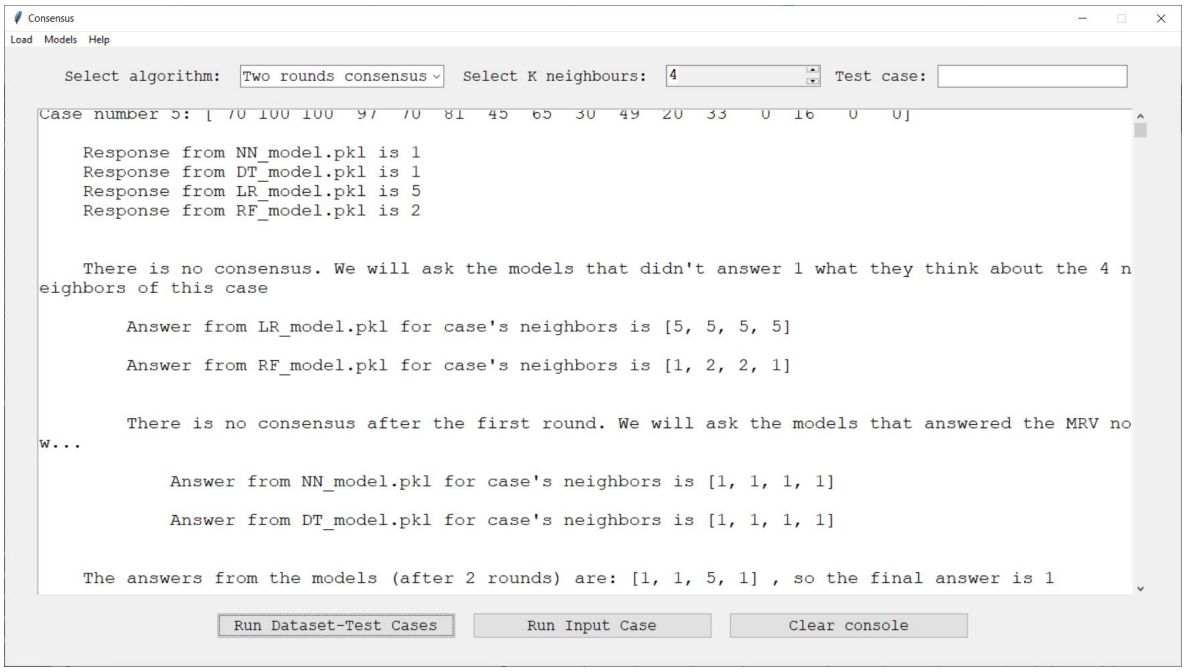

Figura 6.15: Resultados del consenso de doble vuelta K = 4

Aquí, al igual que anteriormente, podemos ver que se ejecutan rondas de consenso en caso de no tener unanimidad. El feedback es prácticamente el mismo que en el consenso de una ronda, salvo por el hecho de que puede haber hasta dos rondas diferentes. También podemos observar los cambios que se dan en el array de respuestas de los modelos.

Una vez se han obtenido los resultados en las predicciones con las ejecuciones de los distintos algoritmos, se puede comparar estos resultados con los clasificadores tradicionales. La precisión de estos últimos puede compararse con la de los algoritmos que se

hayan ejecutado, como se muestra en la Figura 6.16, y valorando así si el uso de los algoritmos de consenso es positivo con respecto a los resultados entre ellos y los modelos de clasificación tradicional.

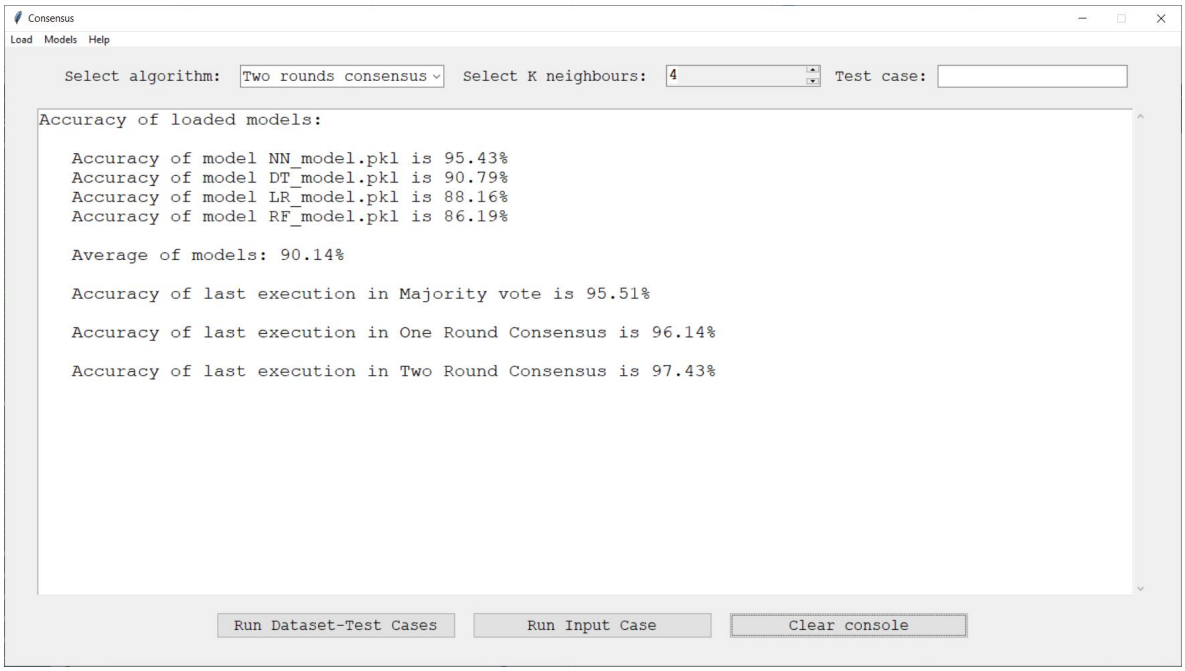

Figura 6.16: Precisiones en los resultados de los clasificadores y los algoritmos de consenso

Puesto que el consenso de dos vueltas ha sido el que ha demostrado una mejor métrica, se utilizaría para predecir casos simples escritos a mano en el input superior derecho de la interfaz, como puede verse en la Figura 6.17, y para ejecutarlo y obtener la predicción a este caso simple, habría que pulsar el botón de Run Input Case.

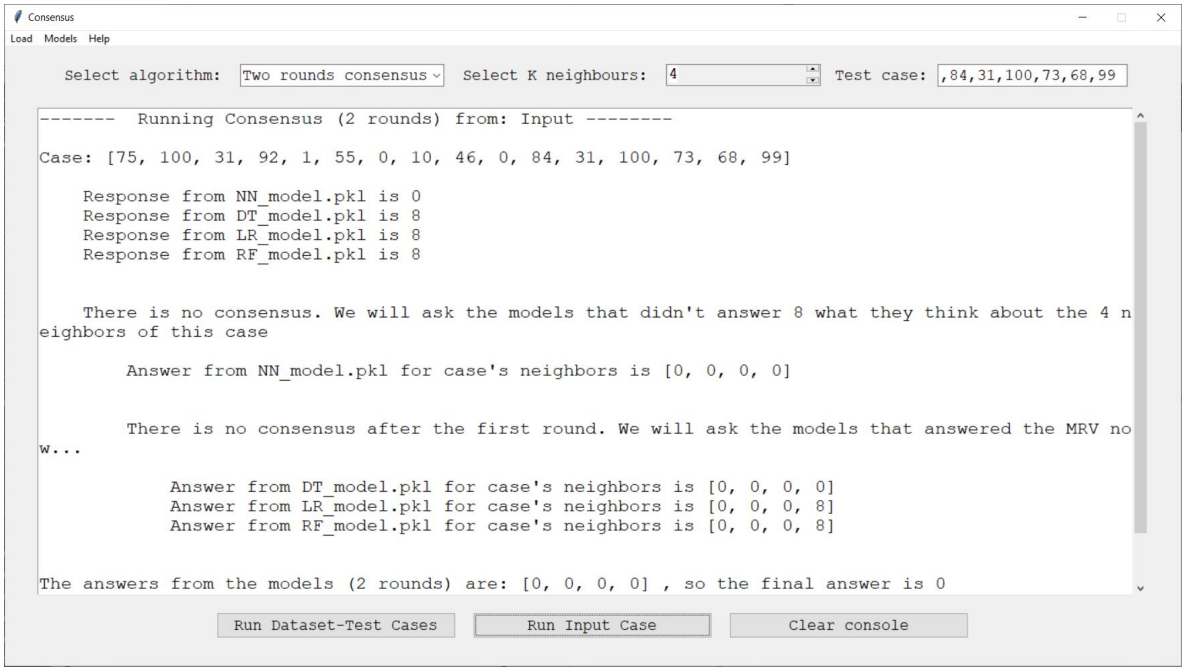

Figura 6.17: Predicción para un Input Case

Además, desplegando la sección de Models, como puede verse en la Figura 6.18, puede seleccionarse la opción de añadir o eliminar clasificadores de la lista actual. Para ello, seleccionamos el modelo que quiera añadirse o eliminarse de la lista, como se indica en la Figura 6.19

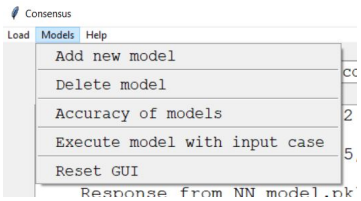

Figura 6.18: Opciones en la sección de Models

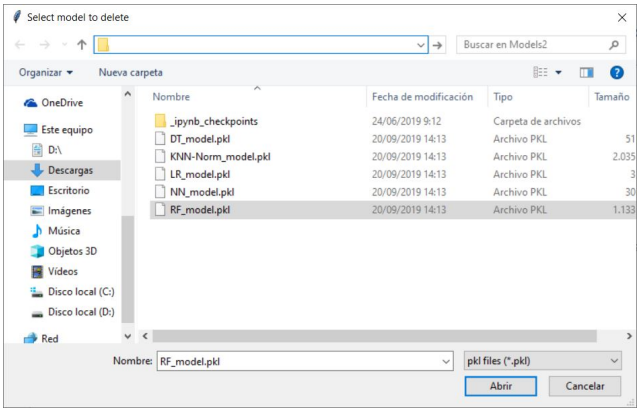

Figura 6.19: Selección del modelo a borrar

También pueden ejecutarse modelos tradicionales con formato .pkl desde la opción de la sección Models de la barra de menú. En la Figura 6.20 puede verse el feedback en dicho caso.

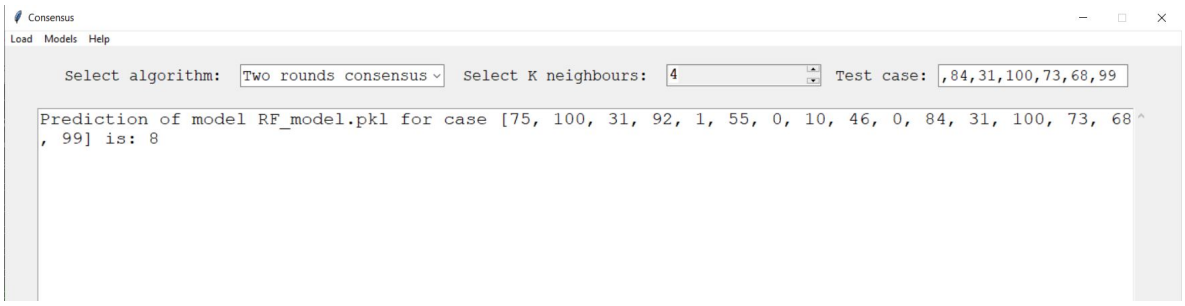

Figura 6.20: Utilizar un clasificador tradicional para predecir el caso del input

Los resultados obtenidos por los algoritmos de consenso, predicción para cada caso y porcentaje final de acierto, se han dejado completados en el notebook Main-data en el directorio del proyecto.

6.3. Modelo-Vista-Controlador

Para diseñar una interfaz que tenga el comportamiento anterior, se propone la arquitectura que puede verse en la Figura 6.21.

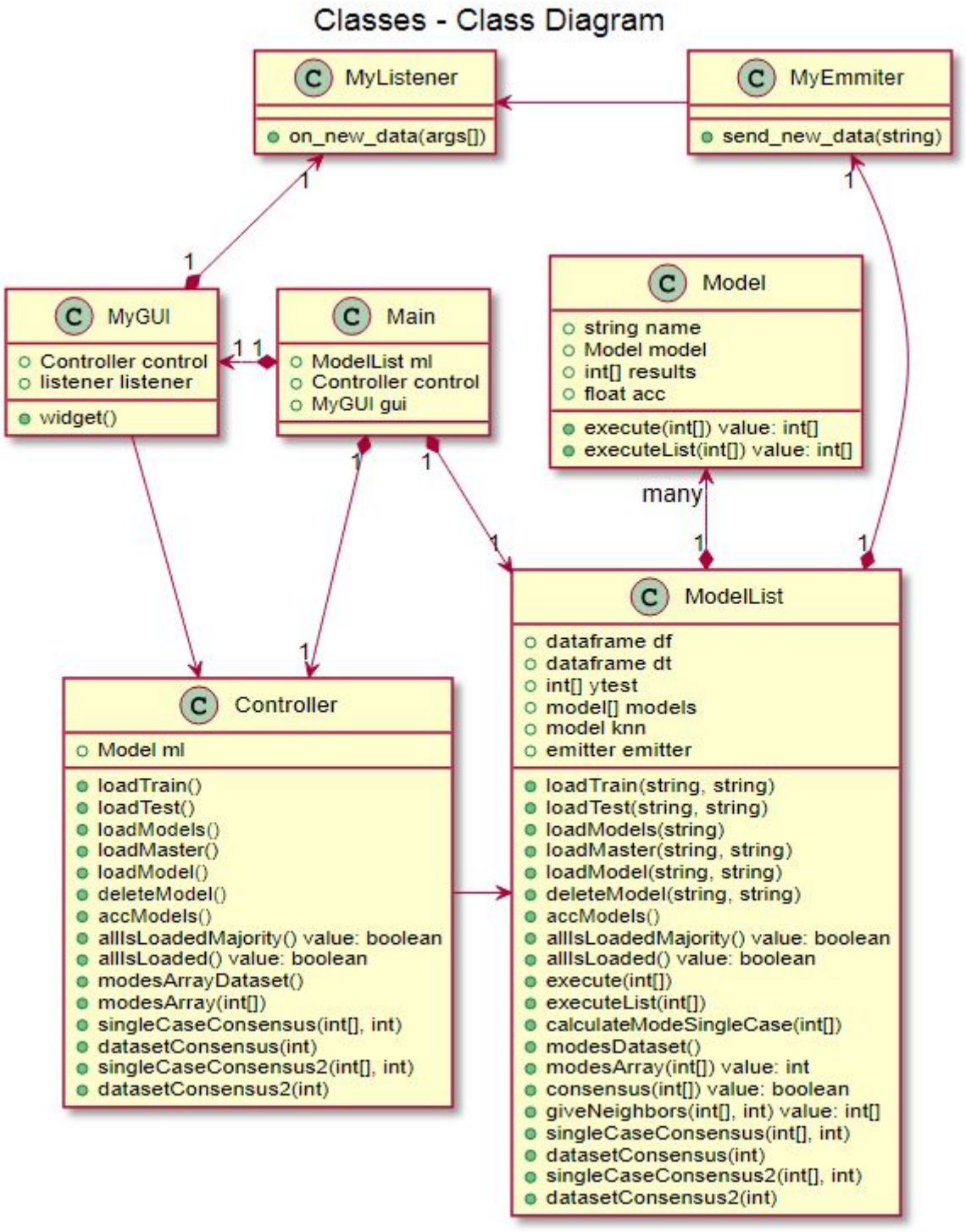

Figura 6.21: Diagrama de clases

La aplicación sigue la estructura del patrón modelo-vista-controlador. La comunicación entre estas partes se realiza de forma que si el usuario interactúa con los botones de la GUI, esta se lo transmite al controlador, y el controlador crea una instancia del comando

adecuado e indica que se ejecute la función correspondiente en el modelo. La información que debe mostrarse al usuario es devuelta al ejecutarse un evento durante la ejecución de la función del modelo, escribiendo así en la vista la información relevante para el usuario. En la Figura 6.22 puede verse un diagrama de secuencia de una interacción con la aplicación.

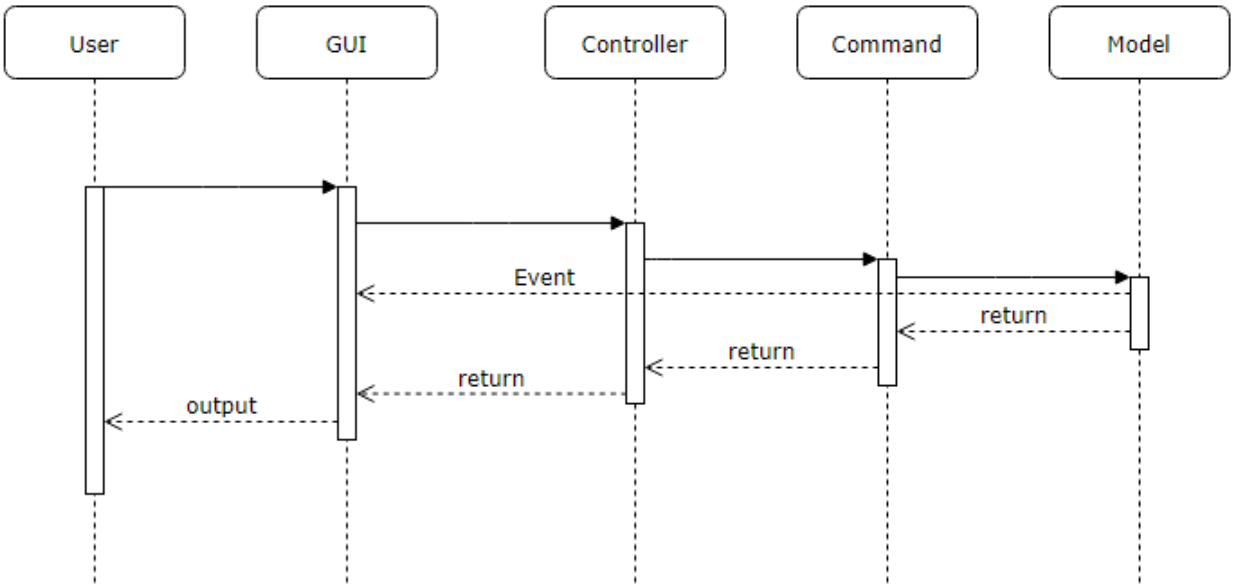

Figura 6.22: Diagrama de secuencia

- **Ventana:** parte gráfica que representa el modelo de forma adecuada, donde el usuario puede interactuar fácilmente con la aplicación.

  Actúa como receptor (listener) durante la ejecución de eventos que se producen en el modelo para visualizar lo que va ocurriendo durante la ejecución en la consola de salida y comunica las acciones realizadas por el usuario directamente al controlador.

- **Controlador:** actúa de intermediario entre el modelo y la vista, realizando las peticiones de información y ejecución al modelo, permitiendo separar así la lógica de la aplicación de la parte visual.

- **Comandos:** con el patrón de comandos se consigue solicitar acciones al modelo, creándose una instancia de un objeto Comando adecuada a la acción a realizar. Esto permite, además, poder incluir con facilidad otros comandos nuevos posteriormente para aumentar la funcionalidad.

- **Lista de modelos:** es la representación de la información relativa a los modelos y datasets, gestionando el acceso a dicha información y las operaciones que se realizan sobre las mismas. Actúa como emisor (emitter) de información para que la vista reciba información sobre la ejecución.

- **Eventos:** son señales que la GUI recibe desde el modelo para ejecutar ciertas acciones en la salida de la interfaz.

Cada una de las clases mencionadas han sido implementadas en un archivo de extensión .py, aunque cabe mencionar que todas las clases de comandos se han implementado en el

mismo archivo, ya que Python permite esto. Por otra parte, también la implementación de la clase Emitter y la clase Listener quedan declaradas en la clase Main.

Las instancias de emisor y receptor se enlazan en la función principal de la aplicación. Los datos enviados por el emisor (asociado al modelo), generalmente de texto, son recogidos por el receptor correspondiente (asociado a la vista), actualizando la información disponible para el usuario.

Capítulo 7

Experimentación con los algoritmos de consenso

En este capítulo se analizan los resultados obtenidos de forma individual para cada dataset y de forma global. Puesto que la precisión de un modelo no es una medida que realmente represente de forma adecuada su rendimiento, se acude a la matriz de confusión que se genera en el testeo de los modelos y algoritmos. De esta forma, se pueden conocer el número de falsos positivos y negativos, además de los verdaderos positivos y negativos, y otras métricas como el recall y el f1-score.

7.1. Análisis de los resultados

Para analizar los resultados se tienen en cuenta varios factores, siendo de especial relevancia aquellas conclusiones que pueden extraerse de las matrices de confusión.

Para todos los dataset se ha realizado el mismo análisis, teniendo en cuenta las precisiones de los modelos individuales, la precisión media de estos y la obtenida con la aplicación de los distintos algoritmos de consenso, utilizando configuraciones diferentes para el número K de vecinos a implicar en estos últimos. A continuación se exponen por secciones las conclusiones para cada dataset utilizado en el proyecto.

Utilizando Jupyter Notebook pueden consultarse unos notebooks específicos para cada dataset, en los que se pueden ver las distintas salidas de varias configuraciones utilizadas durante el proyecto. Toda la información de cada caso de estudio, desde los notebooks utilizados para la creación de los modelos, los datasets, los propios modelos y los archivos necesarios para la ejecución de los mismos están separados en carpetas específicas dentro de la carpeta Resultados del proyecto.

7.1.1. Masas anómalas en mamas

Para este dataset en concreto se han aplicado cinco algoritmos: SVM, Redes Neuronales, Árboles de Decisión y Random Forest para los modelos a consensuar y un modelo KNN para actuar como Máster. En cuanto a las métricas, los modelos tienen una media de 77.58 %, siendo el mejor entre ellos el generado mediante random forest con un 79.65 %.

Los resultados para los algoritmos de la mayoría y el consenso de una vuelta son semejantes al resultado del modelo Random Forest; se mejora la media de precisiones obtenidas de los modelos, variando únicamente el número de falsos positivos y negativos respecto al modelo Random Forest y siendo mayor los falsos positivos en el clasificador del consenso. En casos como este de ámbito médico esto puede ser interesante, ya que es mejor obtener un falso positivo y seguir realizando pruebas a un paciente que diagnosticar un falso negativo y que no se atienda la enfermedad.

Uno de los primeros factores a tener en cuenta al respecto de los consensos sería que se trabaja con un par de valores para la K: K = 2 y K = 3. La variación de estos valores a la hora de evaluar el consenso no ha mostrado ninguna diferencia en los resultados para el algoritmo de consenso de una vuelta. Sin embargo, sí se pueden observar dos falsos positivos más al utilizar una K = 3 respecto a K = 2, mostrando un resultado peor. Las precisiones mencionadas puede observarse en la Figura 7.1

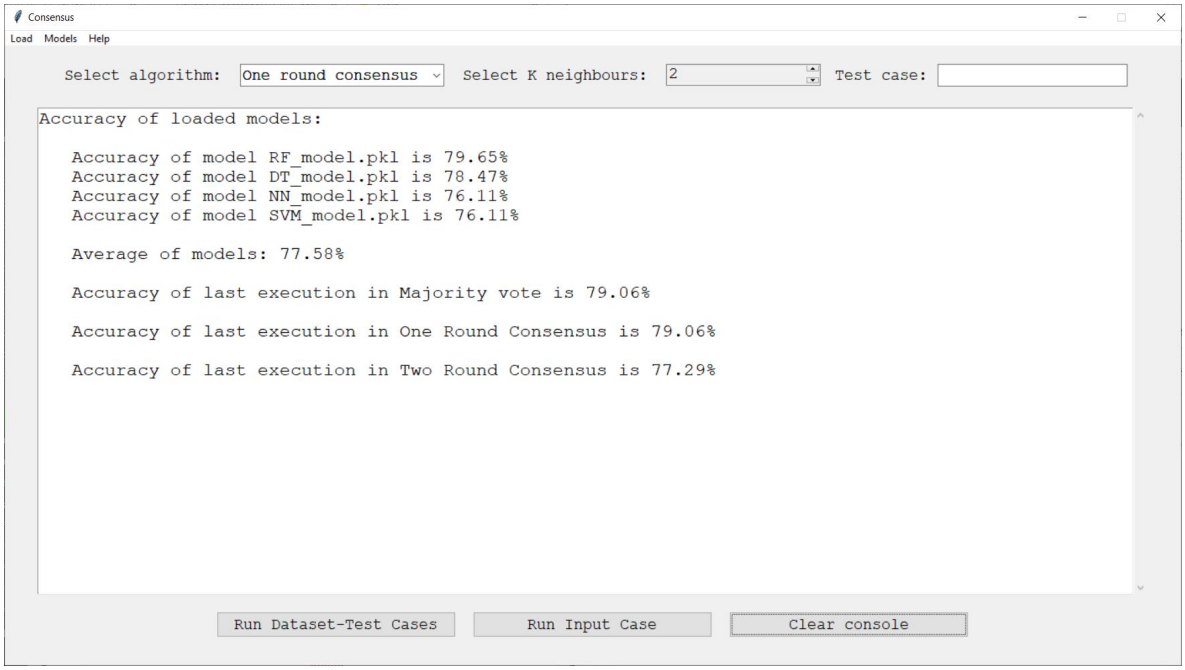

Figura 7.1: Precisiones relativas al dataset de masas en mamas

La ejecución de los casos de test para estas configuraciones y todos los modelos pueden encontrarse en la carpeta Resultados del proyecto, con el nombre correspondiente al dataset.

7.1.2. Reconocimiento de dígitos escritos a mano

Para este dataset en concreto se han aplicado cinco algoritmos: Regresión Logística, Redes Neuronales, Árboles de Decisión y Random Forest para los modelos a consensuar y KNN para actuar como Máster.

El modelo KNN utilizado como máster tiene una precisión de 95.65 % (mejor que el resto de los modelos), y la media de los modelos es un 90.14 %, donde el mejor de entre

ellos es la Red Neuronal, alcanzando un 95.43 %. En este caso, mayoría y consenso de una vuelta dan resultados distintos debido a que la clase a predecir es multiclase. Una explicación anexa al respecto se puede encontrar en 7.2.

En este dataset multiclase los resultados son bastantes satisfactorios, y además las matrices de confusión son fácilmente interpretables (Cuadro 7.1), porque se aprecia qué números han fallado al confundirse con otros, resultando en que realmente esos números son los más parecidos entre ellos, como pueden ser 1 y 7 o 0 y 8.

Tras ejecutar el algoritmo de la mayoría se obtiene una precisión del 95.51 %, superando eficazmente la media de los modelos y alcanzando a la red neuronal, que era el mejor de ellos, pero aún sin superar al KNN. Utilizando K = 2, 3, 4 en las pruebas se obtiene en K = 3 los mejores resultados para los consensos, donde el de 1 vuelta alcanza 96.14 % mejorando al de la mayoría. Ejecutando el algoritmo de consenso de doble vuelta, se obtiene 97.57 %, siendo este el mayor de los porcentajes de acierto en las predicciones Las precisiones mencionadas pueden verse en la Figura 7.2.

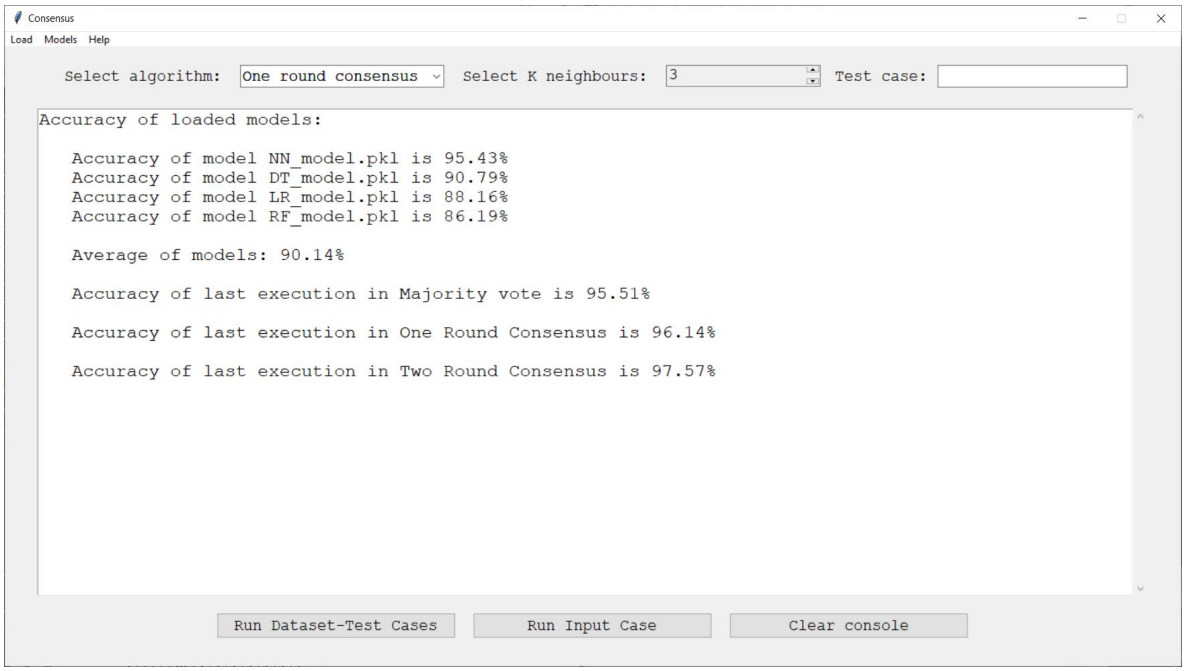

Figura 7.2: Precisiones relativas al dataset de dígitos

A continuación, en el Cuadro 7.1 se muestra una matriz de confusión correspondiente al consenso de doble vuelta, donde se pueden observar qué números se confunden con cuales generalmente.

La ejecución de los casos de test para estas configuraciones y todos los modelos pueden encontrarse en la carpeta Resultados del proyecto, con el nombre correspondiente al dataset.

	0	1	2	3	4	5	6	7	8	9
0	353	0	0	0	0	0	2	0	7	1
1	0	346	15	1	1	0	0	1	0	0
2	0	1	363	0	0	0	0	0	0	0
3	0	2	0	332	0	0	0	0	0	2
4	0	0	0	0	355	9	0	0	0	0
5	0	0	0	7	0	321	0	0	0	7
6	0	0	0	0	0	0	336	0	0	0
7	0	14	0	0	0	0	0	349	1	0
8	1	0	0	0	0	1	0	0	334	0
9	0	1	0	5	0	2	0	3	1	324

Cuadro 7.1: Matriz de confusión para el consenso de doble vuelta con K = 3

7.1.3. MAGIC Gamma

Para este dataset en concreto se han aplicado seis algoritmos: Regresión Logística, Redes Neuronales, Árboles de Decisión, Random Forest y Máquinas de Soporte Vectorial para los modelos a consensuar y un modelo KNN para actuar como Máster.

A diferencia de los casos anteriores, este dataset tiene el modelo Máster con una precisión por debajo del resto de modelos, con un 77.5 %, y la media de los modelos se sitúa en un 82.77 %, siendo el mejor de estos el modelo de Árbol de Decisión, con un 87.89 %.

Un problema que se ha podido apreciar en este dataset, es que la cantidad de casos de entrenamiento era del doble de 0 que de 1 y afectaba a la hora de consensuar, ya que otorgando los casos vecinos era más probable que fueran de una clase que de otra por estadística para determinados casos.

Los resultados para los algoritmos de mayoría y consenso de 1 vuelta son iguales ya que este dataset es binario, y ambos alcanzan un 84.49 % superando al resto de modelos y siendo el mejor algoritmo para este dataset.

Respecto al consenso de doble vuelta, se ha podido probar con varios valores de K al haber entrenado KNN con 13 vecinos, utilizando K = 2, 5, 8, 11, se obtienen los mejores resultados con K = 5, alcanzando un resultado de 79.57 %, siendo este menor que el obtenido con el algoritmo de la mayoría y el del modelo del Árbol de Decisión. Las precisiones mencionadas pueden verse en la Figura 7.3 a continuación:

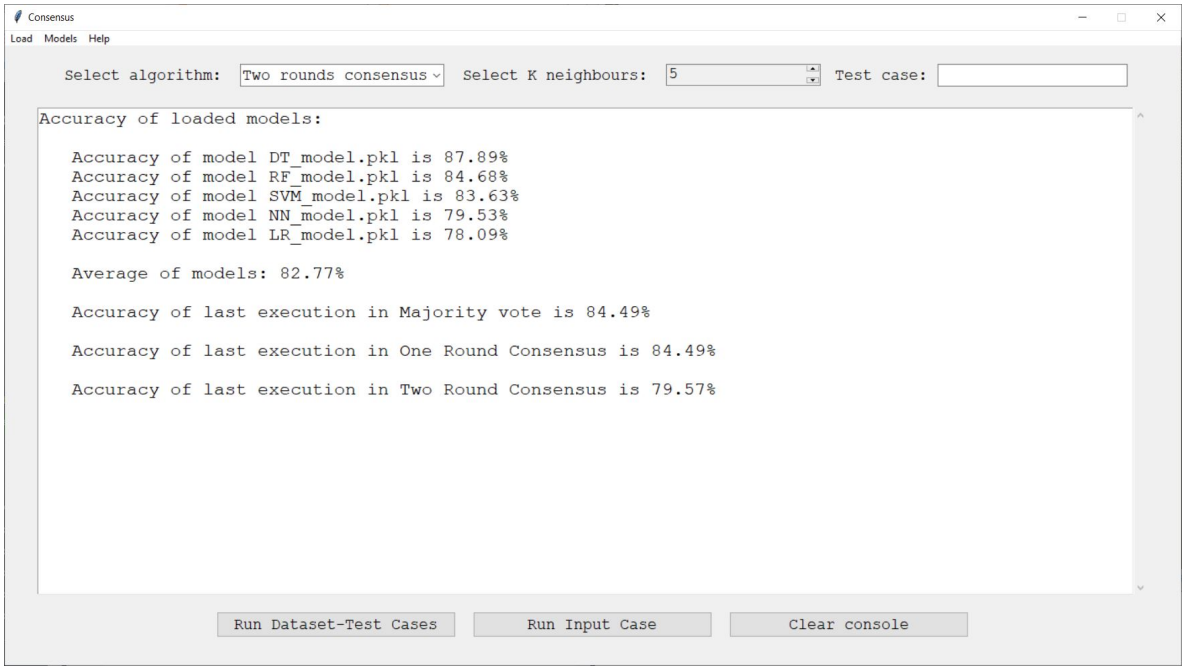

Figura 7.3: Precisiones relativas al dataset de partículas

La ejecución de los casos de test para estas configuraciones y todos los modelos pueden encontrarse en la carpeta Resultados del proyecto, con el nombre correspondiente al dataset.

7.1.4. Tipos de vino

Para este dataset en concreto se han aplicado cinco algoritmos: Regresión Logística, Árboles de Decisión, Random Forest y Máquinas de Soporte Vectorial para los modelos a consensuar y KNN para el modelo que va a actuar como Máster.

De entre la lista de modelos para consenso, el mejor es el generado mediante Random Forest, con un 98.15 %, seguido de cerca por el Decision Tree, con un 96.30 %. La media de los mismos un 89.35 %.

Aunque es un dataset multiclase, al tener un número de instancias tan reducido no aparece ningún caso que diferencie las predicciones entre el algoritmo de la mayoría y el consenso de una vuelta, a pesar de que esto es potencialmente posible, como se explicó anteriormente en este capítulo. Además, tras probar con los valores K = 2, 3, 4, se obtiene el mismo resultado para todas las K, con un porcentaje de acierto del 96.30 %, superando así a todos los modelos excepto al modelo de Random Forest e igualando al Decision Tree.

Respecto al consenso de doble vuelta, se obtiene una tendencia a clasificar los vinos como el tipo 2 debido a que existen más instancias de dicha clase. Esto es lo que hace que los casos límite de 1 y 3 se clasifiquen incorrectamente con frecuencia, por lo que los resultados no alcanzan los resultados esperados e incluso no se supera la media entre los modelos simples, alcanzando un acierto del 85.19 %. Las precisiones mencionadas pueden verse en la Figura 7.4

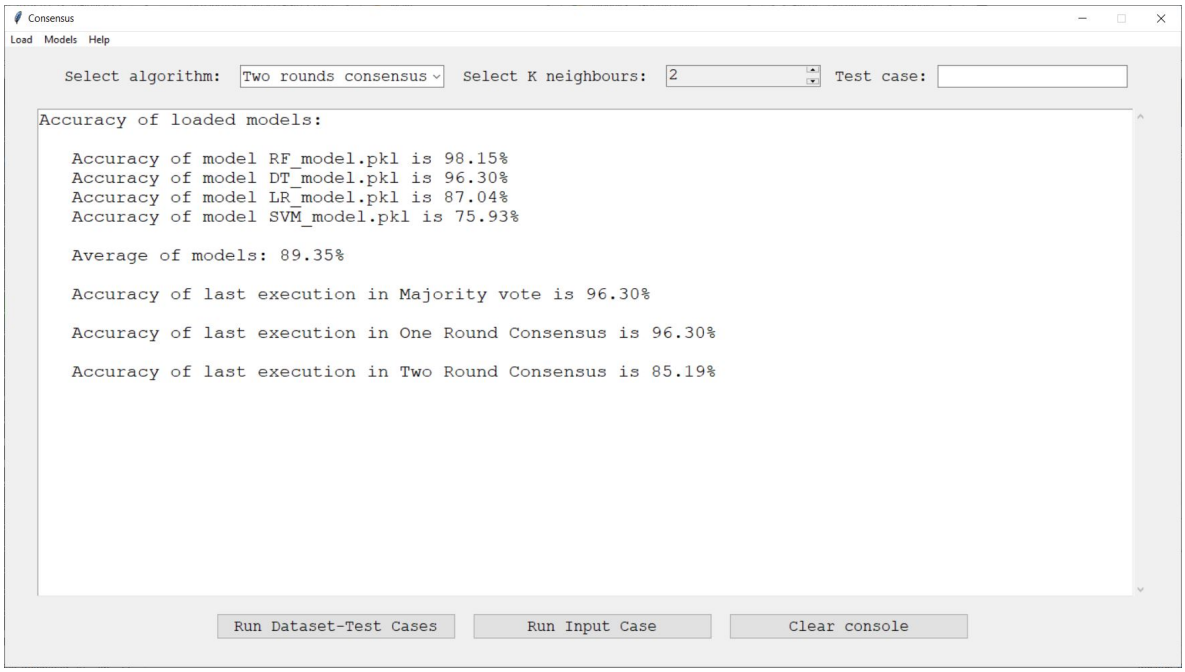

Figura 7.4: Precisiones relativas al dataset de tipos de vino

La ejecución de los casos de test para estas configuraciones y todos los modelos pueden encontrarse en la carpeta Resultados del proyecto, con el nombre correspondiente al dataset.

7.2. Conclusiones generales

Tras analizar todos los resultados, se puede concluir que en general es especialmente importante que el modelo KNN que actúe de Máster utilice una métrica adecuada al problema, es decir, que consiga proporcionar unos vecinos que efectivamente sean casos similares. Dependiendo de la métrica utilizada y de la distribución de los datos en un dataset concreto, es posible que, a pesar de que un modelo KNN sea superior si atendemos a su precisión, la métrica del entrenamiento no se ajuste adecuadamente a la función que lleva a cabo como Máster. Por tanto, podemos decir que **la precisión no afecta en última instancia a los resultados**, puesto que actúa como una fuente de la que obtener nuevos casos. La precisión solo afecta en el caso que esté situado junto a la lista de modelos que van a llevar a cabo el consenso

Para corroborar esto, se han llevado a cabo distintas pruebas en todos los datasets, evitando únicamente no variar la métrica durante el entrenamiento, y los resultados han sido exactamente los mismos en todos los casos para el mismo valor de K. Podemos decir, por tanto, que **a la hora de utilizar un modelo KNN como Máster debemos atender únicamente a la constante K**, que determinará el número de vecinos que podemos pedirle al modelo **y a la métrica utilizada**. Esto implicaría que quizá sea adecuado generar un modelo exclusivamente para realizar esta función.

Así, los requisitos para utilizar de forma adecuada los algoritmos de consenso serían:

- Una lista de modelos donde existan al menos dos modelos
- Un modelo KNN que actúe como Máster y cuya métrica esté adecuada al problema.
- Proporcionar los datos con los que se entrenaron los modelos, con el fin de poder devolver casos vecinos.

Una vez proporcionada esa base, se puede decir que los algoritmos de consenso pueden ejecutarse con ciertas garantías, si bien esto no significa que los resultados vayan a mejorar, como ocurre en 7.1.4.

Más que hacer de apoyo sobre el modelo con mejor porcentaje y ayudar a acertar los casos que no son correctos, también puede darse que casos correctos pasen a ser incorrectos, ya que todos los modelos ponderan por igual, a excepción de una situación de empate, donde se pondera según el orden de aparición en la lista. Es por ello que si la diferencia en el % entre modelos es de un intervalo grande, al ejecutar con todos los modelos es muy difícil superar al mejor de estos, aunque sobrepasar la media sí es factible.

Puesto que el porcentaje del KNN no afecta en las predicciones, sino en los vecinos generados para la búsqueda del consenso, si su % de acierto es mucho mayor que el resto de modelos, es muy difícil alcanzarlo, ya que las predicciones las harían el resto de modelos y los resultados variarían entorno a ellos.

La situación ideal es que el intervalo entre la mejor de las precisiones y la peor sea lo menor posible y que el modelo KNN tenga la métrica más apropiada.

Por último, merece la pena explicar un par de casos hipotéticos donde se visualice el por qué existe una diferencia entre el algoritmo del voto de la mayoría y el consenso de una ronda en datasets multiclase y no en binarios, podemos ilustrarlo con los siguientes:

- Ejemplo binario: si se dispone de la siguiente predicción [1 1 1 0 0 0] (caso límite donde ambos valores son moda), la mayoría es uno, porque hay tantos unos como ceros, pero este aparece antes (orden por precisión), y si tratamos de convencer a los ceros con el consenso de 1 vuelta, en ningún caso el resultado será distinto de 1. Esto se debe a que estos ceros seguirán con su valor o cambiarán a uno, mientras que el resto no variará, no cambiando así el resultado final. El tamaño de la lista de modelos a convencer es menor o igual al tamaño de la lista de modelos que cumplen la mayoría, por lo que nunca será otro valor final tras ejecutar el consenso.
- Ejemplo multiclase: si se dispone de la siguiente predicción [5 5 5 3 3 8 8], la mayoría es cinco, pero en el consenso de 1 vuelta, donde se van a intentar convencer a los modelos que respondieron 3 y 8, estos pueden ponerse de acuerdo y alcanzar una mayoría distinta de cinco, como podría ser [5 5 5 8 8 8 8], puesto que los modelos que respondieron con 3 pueden cambiar de opinión y resultar en otro número distinto al de la mayoría. Si el tamaño de la lista de modelos a convencer es mayor que el tamaño de la lista modelos que cumplen la mayoría, entonces puede darse un caso como el del ejemplo y que el resultado final cambie.

Capítulo 8

Conclusiones y trabajo futuro

Vistos los resultados y análisis de los mismos expuestos en el capítulo 7, puede afirmarse que la adecuación del algoritmo de consenso al problema es de especial importancia.

Se han implementado estos algoritmos de consenso con una función en la que se recogen las respuestas de todos los modelos listados y se efectúa un recuento de votos. Respecto a la creación de modelos, se decide utilizar aquellos de aprendizaje supervisado y siempre y cuando los resultados tengan una precisión superior al 70 %.

Los casos en los que dos o más de los posibles resultados coinciden en número de votos se etiquetan de peligrosos, ya que la función mode() de la librería maths no es capaz de resolver estos casos y se producen excepciones que detienen la ejecución. Se implementa una moda que sea capaz de resolver estos casos donde varias modas tienen el mismo grado de aparición (Apartado 5.1.1).

En los inicios del proyecto, se planteó la utilización de un único dataset, aunque durante el análisis de los resultados del mismo, se observó que no existía una diferencia entre los resultados obtenidos con el algoritmo de voto de la mayoría y los obtenidos con el consenso de una única ronda. 7.2. Esto se debía a que el dataset es del tipo binario, es decir, las etiquetas posibles de las instancias son o bien 0 o bien 1. Ante esto, se llegó a la conclusión de que era necesario plantear dichos algoritmos con diferentes tipos de dataset si quería verse con una perspectiva más objetiva el rendimiento de todos los algoritmos de consenso. Por ello, se optó a la utilización de datasets multiclase, es decir, uno en el que las posibles etiquetas de cada instancia sean más que 0 o 1. Como se comentó en 4.5, la búsqueda de dichos dataset se hizo atendiendo también a su predisposición a la limpieza, ya que se quería priorizar la utilización de más datasets.

En cuanto a los resultados obtenidos, queda patente que en nuestros algoritmos en particular es muy relevante la métrica (usada como parámetro) para el entrenamiento del modelo KNN utilizado como Máster, como ya se comentó en la sección 7.2, ya que esta métrica utilizada es la que determinará los vecinos devueltos para su análisis en las rondas de discusión de los algoritmos. Se puede decir que la dependencia de la bondad de dichos casos vecinos es muy alta, y que es de vital importancia tener este factor en cuenta.

Sería importante que el modelo utilizando como Máster utilice una métrica que se adapte al problema concreto, es decir, sería recomendable generar modelos KNN cuya

finalidad sea exclusivamente actuar como Máster con los distintos tipos de métricas que ofrecen en la librería de sklearn, dejando de lado las métricas de precisión, y utilizar el modelo que ofrezca mejores resultados junto a los algoritmos. Esto se debe a que la ubicación en el hiperespacio de los datos es un factor que cambia de forma drástica de un problema a otro, y para cada uno es conveniente la utilización de un sistema de métricas distinto.

También cabe destacar que, aunque los resultados esperados eran que las métricas de bondad de los algoritmos de consenso fueran mejores que aquellas alcanzadas por los modelos, no es así en todos los casos. También se esperaba que el consenso de dos rondas mejorase los resultados del consenso de una sola ronda, quien a su vez mejoraría aquellos del voto de la mayoría. Tal y como se comenta en 7.2, esto no es siempre así, y depende del problema en concreto y las características del dataset, más concretamente de la distribución de los datos. Esto queda reflejado en los resultados obtenidos en 7.1.3, 7.1.4.

En definitiva, podemos concluir que los algoritmos de consenso y la fusión de modelos puede potencialmente aumentar el rendimiento de los modelos de Machine Learning generados con los algoritmos tradicionales, a pesar del aumento de la carga computacional que eso conlleva. Es importante prestar atención especialmente a cuál usar en cada problema y tener una visión panorámica de los resultados que nos ofrecen todas las posibilidades antes de usar cualquier metaclasificador, tal y como se hace con cualquier algoritmo más tradicional.

8.1. Trabajo futuro

Respecto al trabajo futuro, podría decirse que hay varias posibilidades al respecto:

- Existen algoritmos de votación y consenso, como los encontrados durante la fase de investigación, que podrían incluirse entre las posibilidades que facilite la aplicación. Esto haría que la misma tuviera un mayor espectro de posibles aplicaciones, ya que podría valorarse el uso de una cantidad mayor de metaclasificadores. Cabe destacar al respecto que el proyecto se ha llevado a cabo de forma que nuevos métodos de consenso puedan incluirse fácilmente.
- Algunos aspectos de la interfaz podrían ampliarse, añadiendo más funcionalidades o secciones que pudiesen considerarse convenientes. Quizá también, con una inclusión de algoritmos de consenso o fusión diferentes, sean necesarias secciones no consideradas durante este proyecto.
- Los algoritmos desarrollados podrían investigarse más en profundidad, usando un mayor número de problemas y analizando los resultados obtenidos.
- Una variante de los mismos de especial interés sería aquella realizada con predicciones estadísticas, donde el resultado sea la probabilidad de pertenencia a las clases en lugar de la propia clase. Esto diferenciaría los resultados del voto de la mayoría y del consenso de una vuelta para los datasets binarios.

- Actualmente los datos a utilizar deben pertenecer a tipos numéricos, ya que todos nuestros datasets utilizan únicamente dichos tipos, y podría ampliarse para aceptar cualquier tipo de dato de entrada.
- Actualmente la columna de clase se supone siempre por defecto que es la última, ya que puede haber problemas en detectar esto dependiendo de si el dataset tiene o no cabeceras. Una posible mejora sería preguntar al usuario el nombre en caso de que tenga header o preguntar por el número de columna correspondiente a la clase, haya header o no.
- Existe un inconveniente en el envío de eventos, ya que aunque programáticamente los eventos están implementados de forma que deberían enviarse de forma paulatina, se produce una espera y se envían todos juntos. El problema podría posiblemente evitarse mediante la inclusión de los eventos de forma asíncrona, aunque no ha sido posible comprobar esto por falta de tiempo.

Capítulo 9

Conclusions and future work

After analyzing the results as shown in chapter 7, the adecuation of the consensus algorithms is specially important.

These consensus algorithms have been implemented as a function in which the responses of all listed models are collected and then a vote count is made. About the models' creation, it is decided to use those of supervised learning and as long as the results have an accuracy greater than 70 %.

The cases were two or more possible outputs are even in number of votes are labeled as cases that need attention, since the mode function of the Math library is not capable of resolve this kind of situation, so a exception is raised. The implementation of a majority vote function was considered the best way to go, were we could manage this and some other possible problems in the future, having more control over that.

When the project started, it was only considered to use only one dataset, but during the analysis of the results it was observed that there was no difference between the results of the majority vote and the one round consensus7.2. This is due to the dataset being binary, this is, the labels of the instances are either 0 or 1. The conclusion so was that the use of more datasets was mandatory if the analysis of the results was expected to be fair and objetive. This is why the multiclass dataset (datasets were the possible labels are more than 0 or 1) is used. As commented at 4.5, the research of the datasets was atending to this fact, looking for those who were more prone to be cleaned and mapped, priorizing those dataset to be used.

About the obtained results, it is clear that in our algorithms the metric used training the KNN model that will be used as Master model is the most relevant thing about it, as already mentioned in section 7.2. This is because this metric will determine which neighbors will be returned for its analysis in the discussion rounds of the algorithms. We can say that the depence of the goodness ot these neighbors is very high, and that is very important to take this factor into account.

It is important that the model used as Master implements a metric adapted to the problem, this is, it is recommendable to generate as much models as possible metrics are in sklearn library and which its only purpose is being used as a candidate Master. Metrics like accuracy are irrelevant for this purpose, so the metric that shows better accuracy

when used along the algorithms is the one which should be used. The reason why is because the data distribution is different in each problem, so the best metrics are also different from one to another.

Even if the results and the goodness of the consensus algorithms were expected to be better that those of the models, that is not always the case. It was also expected a better performance of the two rounds consensus algorithm over the performance of the one round consensus, which was also expected to have better results than those of the majority vote. As is commented at 7.2, this is not always the case neither, and it depends on the particular problem and the dataset characteristics, specially the data distribution. This is also commented at 7.1.3, 7.1.4.

All in all, we can say that the consensus algorithms and the fusion of models can potentially boost the performance of the Machine Learning models, despite the computional cost. It is important to take a deep look at the performance in every particular problem and have an overview of the results of all the possibilities before using any of the meta classifier algorithms befode we use them, just as we do with the tradional ones.

9.1. Future work

About future work, there are a few possibilities:

- There are voting and consensus algorithms, like those found during the investigation, that could be included in the application. This would made it more varied and more problems would fit in the possibilities, which would also improve the amount of possible metaclassifiers to use. The project has been done with this in mind, so new algorithms could be included with ease.
- Some matters about the interface could be better, adding some new funcionalities if needed or new sections that could be considered of relevance. Maybe the inclusion of new algorithms would make this mandatory as not all the possibilities have been considered.
- The developed algorithms could be investigated more deeply, using more datasets and analysing the obtained results.
- A variant of interest would be one in which the results are probabilities of class instead of the class itself. This would difference the results for binary datasets between the majority vote and the one round consensus.
- Currently the only accepted input data are numeric types, as all of our datasets are type numbered, and this could be extended to any type of data.
- Currently the label column is supposed to be the last one, as there are problems with headers and without them, so the last one is expected to be the label by default. A possible improvement would be to ask the user the name of the column if there is a header or the number in any case.

- There is a problem regarding the events. Even though the events are supposed to be sent gradually, there is an implicit wait and all of them are sent all togheter. This may be fixed by the use of asynchronous events, but it could not be checked due to lack of time at the end of the project.

Capítulo 10

Contribuciones personales

10.1. Javier Mendoza García

10.1.1. Investigación

- Realización de cursos online de Python, orientados a Machine Learning, para crear modelos de predicción en este lenguaje.
- Estudio de tecnologías y librerías relacionadas con el ámbito, además del entorno de Jupyter.
- Búsqueda de trabajos relativos a la fusión de modelos de predicción y de la metodología Delphi.
- Búsqueda y filtración de distintos tipos de datasets, tanto binarios como multiclase, con el objetivo de realizar un análisis más variado.

10.1.2. Desarrollo

- Planificación del proyecto y de la organización del desarrollo.
- Apoyo y desarrollo en la creación de la estructura MVC, para que las llamadas a las funciones sigan un flujo correcto y se utilicen comandos y eventos al interactuar el usuario con la interfaz gráfica.
- Desarrollo y diseño de los algoritmos de consenso.
- Diseño y creación en los elementos de la interfaz utilizada y búsqueda de las librerías adecuadas para su correcta funcionalidad.
- Limpieza y mapeo de datos del dataset de masas anómalas en mamas. Limpieza, mapeo y cross validation de los datos en dataset del telescopio MAGIC Gamma y apoyo en el resto de datasets.
- Obtención de mayores porcentajes en los clasificadores probando entre los diferentes tipos de métricas en los KNN y distintos vecinos en los algoritmos de consenso.
- Testeo de la funcionalidad de la interfaz y corrección de errores.

- Revisión final del código del proyecto.
- Reorganización de los directorios relativos a los notebooks del proyecto.
- Inclusión de la función para ejecutar los modelos tradicionales desde la GUI.
- Adaptación de la clase ModelList para poder ejecutarse nuevamente desde notebooks.

10.1.3. Memoria

- Introducción creada conjuntamente, tanto en español como en inglés.
- Clasificadores del capítulo 3.1: Bosques Aleatorios, Redes Neuronales y Regresión Logística.
- Sobre la sección 3.2: lo referente a los algoritmos, el método Delphi, precedentes en la fusión de clasificadores, entorno Jupyter Notebook y conclusiones.
- Sobre el capítulo 4: parte inicial, dataset de masas anómalas en mamas, dataset del telescopio magic gamma, conclusiones y apoyo en el resto de secciones del capítulo.
- Esquematización y revisión del capítulo 6
- Desarrollo del capítulo 7.
- Participación conjunta en el capítulo 8.
- Referencias y bibliografías conjuntamente.
- Redacción en inglés de Introducción y Conclusiones.
- Diagrama de secuencia y diagrama de actividad del consenso de una vuelta.

10.2. Raúl Dorado Pulido

10.2.1. Investigación

- Planificación del proyecto y de la organización del desarrollo.
- Realización de cursos online de Python, orientados a Machine Learning, para crear modelos de predicción en este lenguaje.
- Estudio de tecnologías y librerías relacionadas con el ámbito, además del entorno de Jupyter.
- Búsqueda de trabajos relacionados al resultado alcanzado entre modelos mediante algoritmos de consenso y la interpretabilidad de los modelos.
- Búsqueda y filtración de datasets referentes al ámbito de la salud y búsqueda de posibles datasets proporcionados con limpieza y mapeo de datos para poder realizar más pruebas con los algoritmos desarrollados.

10.2.2. Desarrollo

- Diseño y desarrollo en la creación de la estructura MVC, para que las llamadas a las funciones sigan un flujo correcto y se utilicen comandos y eventos al interactuar el usuario con la interfaz gráfica.
- Desarrollo y diseño de los algoritmos de consenso.
- Diseño y apoyo en los elementos de la interfaz con potencial mejora, diseño del feedback devuelto al usuario.
- Limpieza y mapeo de datos de los dataset de masas anómalas en mamas, dígitos escritos a mano y tipos de vinos.
- Mejora del rendimiento de los clasificadores individuales, analizándolos mediante la matriz de confusión y la medida F1-Score.
- Testeo de la funcionalidad de la interfaz y corrección de errores.
- Reorganización de los directorios relativos a los notebooks del proyecto.
- Inclusión del botón con la funcionalidad de reset.
- Inclusión de la funcionalidad para imprimir en consola los porcentajes de precisión de los modelos y los algoritmos de consenso.

10.2.3. Memoria

- Introducción creada por ambos, tanto en español como en inglés.
- Clasificadores del capítulo 3.1: K-Nearest Neighbors, K-Medias Clustering, Máquinas de Soporte Vectorial y Árboles de decisión.
- Sobre la sección 3.2: lo referente a los algoritmos, la interpretabilidad de modelos, precedentes en la fusión de clasificadores, entorno Jupyter Notebook y las conclusiones.
- Sobre el capítulo 4: parte introductoria, dataset para el reconocimiento de dígitos escritos a mano, dataset de clasificación de vinos y conclusiones y apoyo en el resto del capítulo.
- Referencias y bibliografías conjuntamente.
- Desarrollo del capítulo 6
- Esquematización y revisión del capítulo 7.
- Participación conjunta en el capítulo 8.
- Repaso general de la memoria, unificando el estilo impersonal y eliminando errores de los apartados en inglés.
- Desarrollo en inglés de Introducción, resumen y conclusiones, además de las pertinentes correcciones.

- Diagrama de actividad del voto por mayoría y diagrama de actividad del consenso de doble vuelta.
- Revisión final de la memoria tras la correción del borrador.

Índice de figuras

Índice de cuadros

Bibliografía

[1] Wikipedia contributors. K-nearest neighbors algorithm. 9 2019. https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm.

[2] Avinash Navlani. Knn classification using scikit-learn. 08 2018. https://www.datacamp.com/community/tutorials/k-nearest-neighbor-classification-scikit-learn.

[3] Zohar Karnin Gitansh Chadh, Piali Das. K-means clustering. 11 2018. https://aws.amazon.com/es/blogs/machine-learning/k-means-clustering-with-amazon-sagemaker/.

[4] Scikit learn developers. Clustering. 2007 - 2019. https://aws.amazon.com/es/blogs/machine-learning/k-means-clustering-with-amazon-sagemaker/.

[5] Wikimedia Commons. Kmeans-gaussian-data.svg, 2017. https://commons.wikimedia.org/w/index.php?title=File:KMeans-Gaussian-data.svg&oldid=271403304.

[6] Wikipedia contributors. Support-vector machine. 9 2019. https://en.wikipedia.org/wiki/Support-vector_machine.

[7] scikit-learn developers. Plot different svm classifiers in the iris dataset, 2010 - 2016. https://scikit-learn.org/0.18/auto_examples/svm/plot_iris.html.

[8] Wikipedia. Árbol de decisión, 2019. https://es.wikipedia.org/w/index.php?title=%C3%81rbol_de_decisi%C3%B3n&oldid=117562795.

[9] Wikimedia Commons. Arbol decision, 2013. https://commons.wikimedia.org/w/index.php?title=File:Arbol_decision.jpg&oldid=103073462.

[10] Scikit learn developers. Ensemble methods. 2007 - 2019. https://scikit-learn.org/stable/modules/ensemble.html#forest/.

[11] Avinash Navlani. Understanding random forests classifiers in python. 05 2018. https://www.datacamp.com/community/tutorials/random-forests-classifier-python.

[12] Wikipedia. Regresión logística, 2019. https://es.wikipedia.org/w/index.php?title=Regresi%C3%B3n_log%C3%ADstica&oldid=116126298.

[13] ODSC Open Data Science. Logistic regression with python. 04 2019. https://medium.com/@ODSC/logistic-regression-with-python-ede39f8573c7.

[14] Wikipedia. Red neuronal artificial, 2018. https://es.wikipedia.org/w/index.php?title=Red_neuronal_artificial&oldid=117638543.

[15] Fernando Sancho Caparrini. Redes neuronales: una visión superficial, 2018. http://www.cs.us.es/~fsancho/?e=72.

[16] Wikimedia Commons. Colored neural network, 2018. https://commons.wikimedia.org/w/index.php?title=File:Colored_neural_network.svg&oldid=279111871.

[17] Md Kamruzzaman Sarker, Kazi Alam, and Md Arifuzzaman. Emotion recognition from speech based on relevant feature and majority voting. pages 1–5, 05 2014.

[18] Mathias Bourel, Carolina Crisci, and Ana Martínez. Consensus methods based on machine learning techniques for marine phytoplankton presence-absence prediction. *Ecological Informatics*, 42, 09 2017.

[19] Fred Xue, R Subbu, and P Bonissone. Locally weighted fusion of multiple predictive models. pages 2137 – 2143, 01 2006.

[20] Jorge Gómez-Sanz and Rubén Fuentes-Fernández. Revisiting the delphi method for agents. volume 524, pages 367–376, 06 2015.

[21] Gajendra Katuwal and Robert Chen. Machine learning model interpretability for precision medicine. 10 2016.

[22] Adrien Bibal and Benoît Frénay. Interpretability of machine learning models and representations: an introduction. 04 2016.

[23] Gajendra J. Katuwal and Robert Chen. Machine learning model interpretability for precision medicine. 2016. https://pdfs.semanticscholar.org/7070/95416e8d814c08a2f33e7533ddaefdf4f338.pdf.

[24] Web oficial de python. https://www.python.org/downloads/release/python-364.

[25] azure-docs.es es. Tareas para preparar los datos para el aprendizaje automático mejorado. 11 2017. https://docs.microsoft.com/es-es/azure/machine-learning/team-data-science-process/prepare-data.

[26] Assaf Hoogi Daniel Rubin Rebecca Sawyer Lee, Francisco Gimenez. Curated breast imaging subset of ddsm. the cancer imaging archive. 2016.

[27] Assaf Hoogi Kanae Kawai Miyake Mia Gorovoy Daniel L. Rubin. Rebecca Sawyer Lee, Francisco Gimenez. A curated mammography data set for use in computer-aided detection and diagnosis research. scientific data volume 4. 2017. https://www.nature.com/articles/sdata2017177.

[28] Smith K Freymann J Kirby J Koppel P Moore S Phillips S Maffitt D Pringle M Tarbox L Prior F. Clark K, Vendt B. The cancer imaging archive (tcia): Maintaining and operating a public information repository, journal of digital imaging, volume 26, number 6. pages 1045 – 1057, 12 2013. https://link.springer.com/article/10.1007%2Fs10278-013-9622-7.

[29] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. http://archive.ics.uci.edu/ml.

PASCAL
ENERO 2018
Ult. actualización 20 de septiembre de 2019
LaTeX lic. LPPL & powered by TEFLON CC-ZERO