UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

TFG del Grado en Ingeniería Informática

Clasificación de grado de Parkinson mediante aprendizaje supervisado

Presentado por Jorge Martínez Martín
en Universidad de Burgos — 1 de julio de 2024
Tutor: Álvar Arnaiz González

UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

D. Álvar Arnaiz González, profesor del departamento de Ingeniería Informática, área de Lenguajes y Sistemas Informáticos.

Expone:

Que el alumno D. Jorge Martínez Martín, con DNI 71482657Z, ha realizado el Trabajo final de Grado en Ingeniería Informática titulado «Clasificación de grado de Parkinson mediante aprendizaje supervisado».

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección del que suscribe, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, 1 de julio de 2024

Vº. Bº. del Tutor:

D. Álvar Arnaiz González

Resumen

La enfermedad de Parkinson afecta a un número creciente de personas en todo el mundo. Aunque sigue siendo incurable, la detección temprana y el monitoreo continuo de su progreso son cruciales para adaptar los tratamientos a cada paciente de manera personalizada. Este proyecto presenta una página web que permite la clasificación multiclase de vídeos de personas con Parkinson, cuyos resultados se muestran en una gráfica temporal. Para lograrlo, se han empleado técnicas avanzadas de remuestreo y análisis de datos.

El objetivo de esta herramienta es proporcionar a pacientes y médicos una plataforma que, a través del uso de visión artificial, minería de datos e inteligencia artificial, facilite la visualización de la evolución de la enfermedad. Esta herramienta es una ampliación del Trabajo de Fin de Grado (TFG) PADDEL [4].

Descriptores

enfermedad de Parkinson, bradicine, clasificación multiclase, aprendizaje supervisado, desarrollo *web*, despliegue, Docker

Abstract

A **brief** presentation of the topic addressed in the project.

Keywords

keywords separated by commas.

Índice general

Índice general iii

Índice de figuras v

Índice de tablas vi

1. Introducción 1

2. Objetivos del proyecto 3
2.1. Objetivos técnicos 3
2.2. Objetivos de *software* 4

3. Conceptos teóricos 5
3.1. Enfermedad de Parkinson 5
3.2. Aprendizaje automático 6
3.3. Minería de datos 7

4. Técnicas y herramientas 17
4.1. Técnicas 17
4.2. Herramientas 17

5. Aspectos relevantes del desarrollo del proyecto 21
5.1. Fase de experimentación 21
5.2. Resultados 26
5.3. Fase de desarrollo de la aplicación 30

6. Trabajos relacionados 35
6.1. A computer vision framework for finger-tapping evaluation . 35

6.2. The discerning eye of computer vision 36
6.3. Supervised classification of bradykinesia 37
6.4. Paddel 37

7. Conclusiones y Líneas de trabajo futuras **39**

Bibliografía **41**

Índice de figuras

1.1. Rapid finger tapping test 1
3.1. Diagrama K-fold cross-validation 15
5.1. Número de instancias por clase antes de realizar el tratamiento de datos. 23
5.2. Número de instancias por clase tras realizar el tratamiento de datos. 24
5.3. Resultados de modelos con todas las características y SMOTE . 25
5.4. Resultados de modelos con 320 características y SMOTE. 27
5.5. Resultados de modelos con 320 características y ROS. 27
5.6. Resultados de modelos con 320 características y ROS. 29
5.7. Resultados de modelos con 320 características y SMOTE. 29
5.8. Resumen del reporte de *SonarCloud* sobre PDAIvolution. 31

Índice de tablas

3.1. Matriz de confusión. 13

1. Introducción

El parkinson es una enfermedad crónica e irreversible que afecta a más de 150 000 personas en toda España, siendo esta la segunda enfermedad neurodegenerativa más común después del Alzheimer [16]. Esta enfermedad se caracteriza por la perdida de neuronas dopaminérgicas lo que conlleva un desequilibrio en los circuitos neuronales que controlan el movimiento, lo cual puede provocar temblores, rigidez muscular, lentitud de movimiento (bradicinesia) y problemas de equilibrio y coordinación [14]. Pese a que no exista una cura, saber la evolución del paciente y su respuesta a los medicamentos y tratamientos puede ayudar enormemente tanto a pacientes como a médicos a adecuarse mejor a las circunstancias del paciente y con ello mejorar su calidad de vida.

La bradicinesia es uno de los síntomas más importantes a la hora de clasificar el grado de la enfermedad. Este síntoma se hace muy visible en un ejercicio conocido como test de golpeteo rápido de los dedos (o «rapid finger tapping test» en inglés). Este ejercicio consiste en separar y juntar repetidas veces los dedos índice y pulgar de forma rápida y constante.

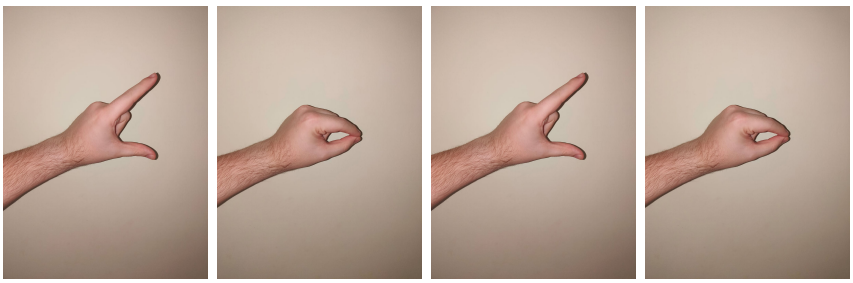

Figura 1.1: Rapid finger tapping test

En las últimas décadas, el campo de la inteligencia artificial ha experimentado un gran avance, con aplicaciones prometedoras en el ámbito de la salud. Específicamente, las técnicas de aprendizaje automático han demostrado su potencial para el diagnóstico y clasificación de enfermedades neurodegenerativas a partir de datos clínicos, genéticos y de neuroimagen [15].

En este proyecto se ha hecho uso de la inteligencia artificial para la creación de modelos capaces de clasificar el grado de bradicinesia en vídeos del «rapid finger tapping test». Ver de manera individual la clasificación de cada vídeo puede no ser del todo útil para poder comprobar la evolución del paciente. Es por ello que se ha decidido crear, usando vídeos tomados de distintos días, una gráfica en la que se pueden ver las dos características de la bradicinesia: la amplitud y la lentitud.

Este proyecto es la continuación de un Trabajo de Fin de Grado anterior llamado PADDEL [4] desarrollado por Catalín Andrei Cacuci. La idea de este proyecto es pasar de un clasificador binario, que indicaba si el paciente presentaba o no la enfermedad de Parkinson, a un clasificador multiclase, el cual evalúa el grado de amplitud y lentitud del video. Además, como se indica anteriormente, se muestra una gráfica con la evolución del paciente.

2. Objetivos del proyecto

El principal objetivo de este proyecto es la creación de una aplicación web que permita, mediante la evaluación del grado de bradicinesia utilizando vídeos de la prueba del «rapid finger tapping test» y el uso de inteligencia artificial realizar dicha evaluación. Además, esta aplicación debe poder ser usada por el usuario promedio.

2.1. Objetivos técnicos

Por ello, para poder alcanzar estos objetivos generales, se plantean los siguientes objetivos técnicos:

- **Revisión de trabajos relacionados**: para comprender mejor el campo que se esta investigando y obtener conocimientos relacionados con el tema.
- **Desarrollar el modelo de aprendizaje supervisado**: entrenar, evaluar y seleccionar el modelo que mejor resultados ofrezca.
- **Evaluar la calidad de la solución**: comprobando si los resultados que nos ofrece son coherentes.
- **Creación de una aplicación *web***: que sea fácil de usar e intuitiva para el usuario.
- **Familiarizarse con nuevas tecnologías**: como programación *web* o Docker.

- **Documentar el proyecto**: de manera clara y concisa, condensando toda la información relevante en un formato accesible y fácil de entender.

2.2. Objetivos de *software*

Se han propuesto los siguiente objetivos de desarrollo de *software*.

- **Crear una interfaz de usuario accesible**: para que el usuario medio pueda usarla sin dificultad.
- **Crear un sistema de gestión de datos persistente**: como bases de datos SQL.
- **Asegurar la calidad del código**: mediante herramientas de análisis de código como SonarCloud.
- **Deplegar en plataformas**: mediante el uso de herramientas como Docker, permitiendo además la familiarización con estas tecnologías.
- **Implementar trabajos anteriores**: para realizar la extracción de características y obtener pautas para el correcto desarrollo.

3. Conceptos teóricos

En este capítulo se definirán algunos conceptos teóricos para facilitar la comprensión de este proyecto.

3.1. Enfermedad de Parkinson

La enfermedad de Parkinson (EP) es una enfermedad neurodegenerativa multisistémica progresiva que cada afecta principalmente a a gente de avanzada edad [19]. Entre los síntomas principales de esta enfermedad podemos encontrar: Pérdida significativa de parte de las células productoras de dopamina lo que produce que aparezcan mucho antes los síntomas relacionados con el movimiento que presenta la EP. Otro de los síntomas que podemos encontrar en la EP es la bradicinesia, el cual se caracteriza por la lentitud al realizar movimientos voluntarios así como la ralentización y decremento de amplitud a la hora de realizar movimientos repetitivos. El diagnóstico de la EP, se basa en criterios específicos. Los síntomas iniciales incluyen lentitud de movimientos, junto con rigidez muscular, temblores o problemas de equilibrio. Los controles posteriores descartan otras posibles causas y confirman la presencia de factores que sugieren claramente la EP. La EP suele empezar en un lado del cuerpo y extenderse a lo largo de unos años. Los síntomas incluyen postura encorvada, rigidez, letra más pequeña y marcha arrastrando los pies. Los temblores son frecuentes. Las alteraciones de la marcha (como pasos indecisos o congelación repentina) se vuelven habituales. La pérdida de equilibrio es un problema importante, que aumenta el riesgo de caídas y lesiones.

Rapid finger tapping test

El test de golpeteo rápido de los dedos, también conocido como «rapid finger tapping test» o RFTT es un procedimiento por el cual el paciente realiza un golpeteo de manera repetitiva durante un periodo de entre 10 y 15 segundos en los cuales ha de intentar generar la mayor amplitud posible entre el dedo indice y el pulgar sin bajar la frecuencia a la que lo realiza. Este test es usado comúnmente para el diagnostico de personas con la EP ya que como se explicó anteriormente uno de los síntomas que presenta esta enfermedad el la bradicisnesia el cual produce que una persona con EP al realizar este test muestre deterioro ya sea en la amplitud o en la frecuencia según avanza la prueba. Este test será el que se utilizará en este proyecto para determinar el grado de EP en el que se encuentra el paciente.

Unified parkinson disease rating scale

La escala de evaluación unificada de la EP o UPDRS por sus siglas en inglés es una herramienta creada por la Movement Disorder Society que permite, mediante la evaluación de diversos parámetros del la EP, medir la gravedad de la enfermedad. En ella se miden diversos aspectos de las experiencias tanto motoras como no motoras de la vida diaria. Para el proyecto utilizaremos solo algunas de las medidas que se utilizan para la clasificación del estado de la persona ya que son los datos que se extraerán de los vídeos proporcionados por los pacientes.

3.2. Aprendizaje automático

El aprendiza automático es una rama de la inteligencia artificial centrada en el desarrollo de métodos y algoritmos para que el computador, de manera autónoma sea capaz de, mediante la experiencia y el procesamiento de datos, mejorar en esa tarea. De esta manera los modelos realizaran predicciones cada vez más precisas.

Dentro de un «dataset» podemos encontrar lo que se conocen como instancias. Una instancia es cada fila del «dataset» caracterizadas por atributos que pueden ser tanto categóricos (nombres, colores, categorías,etc) como valores numéricos (números).

Dentro del aprendizaje automático podemos distinguir tres tipos principales:

- Aprendizaje supervisado: a este aprendizaje se le proporciona un con junto de datos completo con ejemplos etiquetados, y es mediante estos ejemplos con los que aprende a clasificar nuevos datos o realizar predicciones. Se puede dividir en regresión (predicción de datos numéricos) o clasificación (predicción de datos categóricos)

- Aprendizaje no supervisado: este aprendizaje, a diferencia del supervisado, se entrena con datos sin clasificar de manera que tiene que ser él el que encuentre las relaciones. Se puede dividir en: clustering (agrupación de datos por su similitud) y reducción dimensional (disminuye el número de variables en un *dataset* sin perder información importante).

- Aprendizaje semi-supervisado: este aprendizaje es una combinación de los dos aprendizajes anteriores ya que se le proporcionan dos conjuntos de datos. En el primer conjuntos los datos están etiquetados, este conjunto es el conocido como entrenamiento (*training*) que es con el cual el modelo se entrena para encontrar la relación entre los datos. En el segundo conjunto los datos no están etiquetados, este conjunto es conocido como test que es el que permite al modelo comparar como de precisas son las predicciones que se realizan ya que, pese a que al modelo se le introduzcan los datos sin etiquetar, el «dataset» si contiene las etiquetas de estas instancias. Al igual que el aprendizaje supervisado su objetivo principal es la creación de clasificadores o regresores.

3.3. Minería de datos

La minería de datos es un proceso automatizado que permite realizar un análisis de grandes cantidades de datos para extraer patrones, relaciones y conocimientos sobre los datos. Este proceso se compone de varias fases: recopilación, preprocesamiento y análisis de los datos, creación del modelo e interpretación de resultados.

Recopilación de datos

El objetivo principal de esta fase es recopilar y organizar la información necesaria para realizar un correcto análisis. Esto puede incluir desde recopilación de datos mediante bases de datos, archivos de texto hasta extracción de características de archivos de vídeo y audio. En este proyecto, por ejemplo, esta fase se ha compuesto de archivos de vídeo de personas realizando el RFTT y la evaluación a ciegas de dos médicos de estos vídeos.

Preprocesamiento de los datos

En esta fase se limpian y transforman los datos para prepararlos para el análisis. Dentro de esta fase podemos encontrar distintas subfases para cada preprocesado.

Extracción de características

En muchos casos, los datos que se recopilan para realizar un estudio no se pueden analizar de forma directa. Es por ello que es necesario transformar estos datos para que sean numéricos. En este proyecto al utilizar archivos de vídeo como fuente de información de los datos, se ha tenido que extraer las características en forma de series temporales para luego poder ser analizadas ya que los modelos no son capaces de procesar un vídeo de forma automática y saber cómo ha de analizarlo.

Dependiendo del *dataset* con el que se este tratando, la extracción de características puede variar bien sea por el tipo de datos, los modelos que se quieran aplicar o los recursos que dispongan.

Tratamiento de valores nulos

Durante el análisis de los datos puede darse el caso de que ciertos valores son nulos, esto se puede deber a diversas razones: que se desconozcan completamente, que al realizar la extracción de características no haya sido capaz de extraer esa característica, que haya habido perdida de información ... Para realizar un correcto análisis de datos, es fundamental saber cómo tratar los valores nulos. Tenemos tres opciones principales para ello:

- Valores continuos: podemos asignar la media bien de las instancias con la misma clase o bien de todas las instancias.
- Atributos categóricos: se puede sustituir por el valor que más se repita.
- Eliminar instancias: podemos optar por ignorar esas instancias y eliminarlas de nuestro *dataset*.

En conjuntos de datos muy grandes, es recomendable eliminar las instancias con valores nulos, ya que un tratamiento inadecuado puede causar errores en el modelo. Además, eliminar una instancia entre miles no afecta significativamente el análisis.

Desbalanceo de las clases

En el análisis de los datos se puede dar la situación en el que una clase este desbalanceada. Esto quiere decir que el número de instancias de ese subgrupo dentro del atributo o clase sea considerablemente menor que el resto de subgrupos. Estas situaciones pueden dar problemas puesto que los algoritmos tienden a centrarse en las clases mayoritarias e ignorar las minoriatiras.

Este problema se puede resolver de distintas maneras:

- ***Random Over-sampling***: el sobremuestro también conocido como *oversampling* consiste en duplicar instancias de las clases minoritarias para que exista el mismo número de instancias de cada clase. Esta solución trae consigo dos problemas principales que son el sobreajuste y la duplicación de instancias.
- ***Undersampling***: el submuestro o *undersampling* consiste en reducir el número de instancias de cada clase hasta lograr que todas tengan el mismo número de instancias. El gran problema que acarrea este método es la perdida de información debido a que al reducir el número total de instancias podríamos estar eliminando algunas con información relevante para el modelo.
- ***Synthetic Minority Over-sampling Technique* (*SMOTE*)**: este método de sobremuestreo crea instancias sintéticas de las clases minoritarias. Para ello primero se selecciona al azar una instancia de la clase minoritaria, se encuentran sus vecinos k más cercanos (siendo k un parámetro del método) mediante una métrica de distancia. Para cada ejemplo seleccionado, se elige uno de sus vecinos más cercanos al azar y se crea un nuevo ejemplo sintético. Este nuevo ejemplo es generado interpolando las características del ejemplo original y el vecino cercano.
- Eliminar la clase: en el caso en el que tengamos un reducidísimo número de instancias de una clase, por ejemplo una o dos, no es viable realizar ninguna de las técnicas anteriormente nombradas ya que el sobremuestro va a llevar consigo un sobreajuste muy elevado debido a que el número de instancias es insuficiente para crear nuevas instancias sintéticas de la clase, creando instancias duplicadas. Asimismo utilizar el submuestreo solo hará que perdamos información de del resto de las clases lo cual dificultará a la hora de entrenar al modelo. Es por ello que hay situaciones en las que la mejor opción es ignorar esta clase

minoritaria, bien sea mediante la reevaluación de las instancias de esa clase o su eliminación del dataset.

Creación del modelo

Esta fase, al igual que la anterior, se compone de varias subfases.

Categoría de problema

Una vez se han analizado los datos y se ha identificado la clase objetivo se ha identificar el categoría de problema. Podemos distinguir tres grandes bloques:

- Regresión: la salida de estos modelos es continua y se utiliza para predecir datos numéricos como temperatura, precios de casas, etc.
- Clasificación: los modelos de clasificación devuelven una etiqueta asociada a una clase. Dentro de este tipos de modelo podemos distinguir:
  - Clasificación binaria: solo existen dos clases, suelen ser clases como si y no, verdadero y falso o apto y no apto. Se puede usar, por ejemplo, para clasificación de correos de *spam*, detección de fraude o diagnostico médico (enfermo/no enfermo).
  - Clasificación multiclase: en este modelo existen más de dos clases. Se puede usar para clasificación de tipos de flores, clasificación del nivel de gravedad de una enfermedad, etc.
  - Clasificación multietiqueta: una sola instancia puede pertenecer a múltiples clases simultáneamente. Por ejemplo, una imagen puede contener múltiples objetos, y cada objeto pertenece a una clase diferente.
- *Clustering*: el *clustering* o agrupamiento se utiliza para agrupar un conjunto de instancias de manera que los objetos en el mismo grupo (o *cluster*) sean más similares entre sí que aquellos en otros grupos. Se puede usar para clasificar imágenes, agrupar documentos, detectar anomalías...

Elección del modelo

Una vez se ha identificado la categoría del problema, se debe seleccionar el modelo adecuado. Para cada categoría de problema, existen modelos concretos que pueden ser útiles. Debido a la extensa variedad de modelos

que existen, solo se explicarán aquellos que se han utilizado durante el desarrollo del proyecto.

- **Vecinos más cercanos**: también conocido como KNN (*K-Nearest Neighbors*) es un algoritmo de aprendizaje básico que sirve tanto para regresión como para clasificación. Este algoritmo trata de clasificar una instancia por medio de obtener la clase más frecuente entre los k vecinos más cercanos. En el caso de la regresión KNN predice el valor objetivo a partir de la media de los k vecinos más cercanos. A diferencia del resto de algoritmos KNN, no necesita una fase de entrenamiento, por ello se ubica en la categoría de los método denominado *Lazy Learning*, ya que usa directamente el *dataset* entero para realizar las predicciones, es decir, no tiene que optimizar pesos. La forma en la que este algoritmo calcula las distancias dependerá del parámetro que se le indique.
- **Máquina de vectores de soporte**: una máquina de vectores de estado o SVM (*Support Vector Machine*), es un algoritmo que, al igual que vecinos más cercanos, se puede usar para problemas tanto de regresión como de clasificación. El objetivo de este algoritmo es encontrar el hiperplano que separa las instancias en cada clase. El hiperplano se puede interpretar como un borde entre una o varias clases.
- ***Random forest***: este algoritmo combina varios árboles de decisión para mejorar la precisión y robustez de las predicciones. Su funcionamiento se basa en crear múltiples grupos de datos, por cada set se creará un árbol de decisiones en el que se escogerán de manera aleatoria un subgrupo de características. Una vez creado cada árbol se hace lo que se conoce como una votación, en la que cada uno de los arboles predice una clase para la instancia. En el caso de los problemas de regresión el resultado de la votación es la media de los resultados de los distintos árboles. Por otro lado, en el caso de los problemas de clasificación, la clase ganadora es la más votada.
- ***Boosting***: es una técnica de aprendizaje automático que, mediante la combinación de modelos débiles (aquellos modelos que son un poco mejores que un modelo completamente aleatorio), mejora la precisión predictiva creando a su vez un modelo más robusto. En cada iteración, el algoritmo de *boosting* ajusta los pesos de los datos de entrenamiento, dando más importancia a los ejemplos que fueron mal clasificados por los modelos anteriores.

- ***Extreme Gradient Boosting*** (***XGBoost***) [1]: este algoritmo es una implementación mejorada y optimizada del algoritmo de *boosting*. Este algoritmo sigue la técnica de potenciación de gradiente o *gradient boosting* donde los modelos se entrenan para corregir el error residual (diferencia entre la predicción y el valor real) de los modelos anteriores mediante la optimización de una función de perdida diferenciable. Utiliza árboles de decisión como modelos débiles.

Selección de hiperparámetros

Una vez elegido el modelo, se han de encontrar los hiperparámetros que más se adecúen al problema. Para ello existen varias técnicas, entre ellas se encuentra el *GridSearch* (la utilizada en el proyecto). Esta técnica consiste en proveer al modelo una lista de hiperparámetros y que, con todas las posibles combinaciones entre los diferentes hiperparámetros, devuelva tanto el mejor resultado obtenido para la métrica que se ha solicitado como los hiperparámetros óptimos que dan ese resultado.

Evaluación del modelo

A la hora de hacer la evaluación del modelo hay distintos métodos y métricas que permiten hacerse una idea de la eficacia del mismo. Uno de los métodos más usados es la matriz de confusión (ver tabla 3.1). Esta matriz se utiliza principalmente en clasificación binaria. Este método se trata de una matriz de dimensiones 2×2 en la cual se reflejan los siguientes datos:

- True positives (TP): los *true positives* o verdaderos positivos son aquellas instancias en las que el valor predicho para su clase y el valor real de su clase coinciden, siendo esta la clase positiva.
- False positives (FP): los *false positives* o falsos positivos son aquellas instancias que han sido predichos como una clase positiva pero realmente pertenecen a la clase negativa.
- False negatives (FN): los *false negatives* o falsos negativos son aquellas instancias que han sido predichos como una clase negativa pero realmente pertenecen a la clase positiva.
- True negatives (TN): los *true negatives* o verdaderos negativos son aquellas instancias en las que el valor predicho para su clase y el valor real de su clase coinciden,siendo esta la clase negativa.

[1] Fuente: https://xgboost.readthedocs.io/en/stable/

		Valor predicho		
		p	**n**	**total**
Valor real	**p′**	True Positive	False Negative	P′
	n′	False Positive	True Negative	N′
	total	P	N	

Tabla 3.1: Matriz de confusión.

Métricas de evaluación

Una vez facilitados los conceptos anteriores se exponen y explican algunas de las métricas más comunes y las utilizadas durante el proyecto.

- ***Accuracy* o Exactitud**: esta medida es la más común en la clasificación binaria ya que proporciona una buena visión general de qué tan eficaz es el modelo. No obstante, esta medida no siempre es útil ya que, en caso de que exista desbalanceo de clases o el modelo no sea capaz de predecir correctamente una clase, no se puede distinguir como de eficaz es para cada clase.

$$\text{Exactitud} = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.1}$$

- ***Precision***: mide cuantas predicciones son correctas frente al número total de predicciones de la clase.

$$\text{Precisión} = \frac{TP}{TP + FP} \tag{3.2}$$

- ***Recall* o Sensibilidad**: mide el número de positivos predichos correctamente frente al número total de positivos reales.

$$\text{Recall} = \frac{TP}{TP + FN} \tag{3.3}$$

- ***Specificity* o Especificidad**: mide el número de negativos predichos correctamente frente al número total de negativos reales.

$$\text{Especificidad} = \frac{TN}{TN + FP} \tag{3.4}$$

- ***F-Measure***: combina la *precision* y el *recall*. Esta medida es especialmente útil en casos donde existe desequilibrio entre clases o cuando los falsos positivos y falsos negativos tienen distinto coste.

$$\text{F-Measure} = 2 \cdot \frac{\text{Precision} \cdot \text{Recuperación}}{\text{Precision} + \text{Recuperación}} \tag{3.5}$$

- **G-mean**: combina la *specificity* y el *recall*. Es muy útil en problemas de clasificación desequilibrados ya que la precisión general del modelo puede ser engañosa al solo tener en cuenta si clasifica bien los casos positivos.

$$\text{G-mean} = \sqrt{\left(\frac{TP}{TP + FN}\right) \times \left(\frac{TN}{TN + FP}\right)} \tag{3.6}$$

Técnicas de evaluación

La validación cruzada es un proceso mediante el que se evalúa un modelo comprobando que no depende de la partición de datos asignada al entrenamiento. Para realizar esta evaluación existen varias técnicas pero la más común (y la utilizada en este proyecto) es la conocida como *K-folds* en la que se divide el *dataset* en k grupos o *folds*. Cada uno de estos grupos se compone del mismo número de instancias. La forma en la que se seleccionan las instancias depende de los parámetros que se indiquen. Por ejemplo, existe una separación balanceada que procura que en cada grupo exista el mismo número de instancias de cada clase conocido como muestreo estratificado. Otra forma de seleccionar estas instancias es de manera aleatoria.

Una vez divido el *dataset* se realiza un entrenamiento y prueba por cada grupo de manera que en cada iteración se selecciona uno de los grupos como validación y el resto sirven como entrenamiento. Tras realizar cada una de las iteraciones se extraen las métricas necesarias y se realiza una media aritmética siendo esta el resultado deseado.

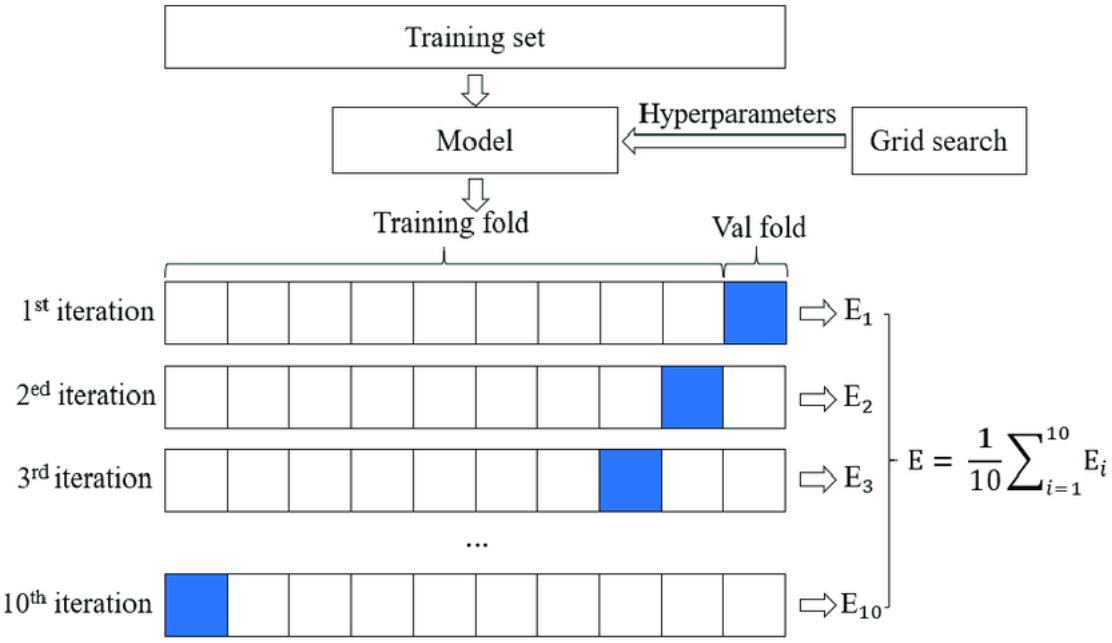

Figura 3.1: Diagrama K-fold cross-validation extraído de «Predicting Mechanical Properties of High-Performance Fiber-Reinforced Cementitious Composites by Integrating Micromechanics and Machine Learning» [10]

4. Técnicas y herramientas

En este capítulo se muestran las diferentes técnicas y herramientas que se han utilizado para el desarrollo del proyecto.

4.1. Técnicas

En esta sección se muestran las técnicas principales empleadas para el desarrollo del proyecto.

Scrum

Para el desarrollo de este proyecto ha sido utilizado marco de gestión de proyectos de metodología ágil conocido como Scrum.

La metodología Scrum trata de fraccionar la duración de un proyecto en lo que se conoce como «sprints», cuya duración varía de una a dos semanas. El objetivo de los «sprints» es decidir que parte del proyecto se va a desarrollar durante ese periodo de tiempo realizando revisiones diarias para ver como avanza así como una revisión de «sprint» en el que hace una valoración general del «sprint» y se decide en que va a consistir el siguiente.

4.2. Herramientas

En esta sección se describen las herramientas que se han utilizado durante la realización del proyecto.

Portales

Usados principalmente para el seguimiento de la metodología Scrum, creación de diagramas y bocetos y control de versiones:

- **Zube**: portal dedicado a la gestión de proyectos «software» [24]. Permite visualizar proyectos, sprints, gráficos propios de Scrum y crear tableros.
- **GitHub**: es una plataforma web que proporciona una interfaz gráfica para el control de versiones utilizando Git [9]. Además, GitHub permite a los usuarios revisar versiones anteriores de proyectos y realizar un seguimiento de los cambios a lo largo del tiempo.
- **DrawIO**: es una herramienta que permite la creación de diagramas entidad-relación, diagramas de flujo, casos de uso, etc [5]. Esta herramienta dispone además de los símbolos UML necesarios para todas las funcionalidades.

Librerías

Principalmente de Python:

- **Scikit-Learn**: esta librería ofrece una amplia variedad de algoritmos de aprendizaje supervisados, semisupervisados y no supervisados [17]. Además posee una gran compatibilidad con otras librerías como así como una extensa documentación que facilitan su implementación.
- **TSFresh**: es una biblioteca de Python, para la extracción y representación de características de series temporales [20]. Esta herramienta agiliza el proceso de extracción de características a partir de datos de series temporales para apoyar las tareas de aprendizaje automático. Ofrece una selección de características definidas que abarcan diferentes medidas estadísticas, como análisis de tendencias, patrones estacionales, métricas de correlación y evaluaciones de complejidad.

Entorno de desarrollo

Incluye el entorno integrado de desarrollo (IDE), lenguaje y otros programas usados.

- **Visual Studio Code**: editor y depurador de código, así como algunas de sus extensiones más importantes.

- **Python**: lenguaje de programación de alto nivel ampliamente usado. Ha sido escogido para realizar este proyecto debido a que posee una gran variedad de paquetes y librerías de Machine Learning [13].
- **Git**: es un software de control de versiones distribuido, utilizado para la control de versiones en proyectos software.
- **Github Copilot**: es un asistente de programación con inteligencia artificial que te permite realizar código de forma más rápida mediante la realización de consultas y las sugerencias que provee.

Desarrollo web

A continuación se exponen tanto librerías como recursos para el desarrollo del apartado web:

- **Boostrap**: es un «framework» utilizado para la creación de páginas web [3]. En esencia es un librería de código que simplifica el proceso de desarrollo de una página web. Unas de sus principales características son:
  - **Diseño «responsive»**: Boostrap permite de una forma muy sencilla implementar la posibilidad de que la página web con todos sus componentes se adapten automáticamente en tiempo real a cualquier pantalla incluso si se modifica el tamaño de la ventana.
  - **Documentación y comunidad**: dado que Boostrap es un framework muy conocido, tanto la documentación del framework como la comunidad que ayuda a desarrollarlo facilitan mucho la posibilidad de aprenderlo y poder encontrar soluciones especificas a nuestro diseño en poco tiempo.
  - **Enfoque móvil**: Boostrap esta diseñado teniendo en cuenta también el diseño en dispositivos móviles, lo que ayuda al desarrollo de la web para multiplataformas.
- **Flask**: es un «framework» escrito en Python que simplifica y facilita la creación de aplicaciones web mediante el uso del patrón Modelo-Vista-Controlador [6]. Dado que es considerado un micro «framework» no viene con todas las funcionalidades por defecto; sin embargo, existen librerías compatibles con Flask que permiten añadir todas las funcionalidades que se necesiten. Entre ellas se han usado:

  - **Werkzeug**: es un conjunto de librerías de Python usados para el desarrollo web [21]. Se ha utilizado Flask junto a Werkzeug ya que este contiene una librería de cifrado que se ha utilizado durante el desarrollo del proyecto.
  - **SQLAlchemy**: dado que Flask no puede hacer uso de forma directa de las bases de datos hace uso de la librería SQLAlchemy [18]. Esta librería permite la creación de bases de datos mediante la creación de clases en Python, definir el esquema y la relación entre las tablas y hacer consultas mediante métodos y objetos en vez de consultas SQL sin formato.
  - **Flask-Login**: es una extensión de Flask que simplifica añadir la funcionalidad de iniciar sesión, administrar sesión y cerrar sesión [8]. Permite además otorgar y restringir accesos de distintas secciones de la aplicación así como cargar datos relacionados con la sesión actual.

- **Jinja**: es un motor de plantillas rápido, seguro y fácil de usar para Python [11]. Permite la modificación de partes de la plantilla de la web mediante el uso de código como la agregación de contenido dependiendo del valor de variables mediante bucles o condicionales.

- **Font Awesome**: es una herramienta usada para el diseño y desarrollo de páginas web [2]. En vez de usar archivos de imágenes para iconos utiliza fuentes. Esto ofrece diversas ventajas:

  - Escalabilidad: los iconos se escalan perfectamente a cualquier tamaño sin perder calidad.
  - Colores personalizables: permite modificar el color de los iconos usando solo CSS. Para poder utilizar Font Awesome con Flask se ha utilizado la librería Flask-FontAwesome que permite su implementación [7].

5. Aspectos relevantes del desarrollo del proyecto

En capítulo de la memoria pretende mostrar y resumir los aspectos relevantes durante el desarrollo del proyecto. En el se incluyen decisiones tomadas durante el desarrollo, modelado del problema y resultados de los experimentos.

En la subsección 5.1 se explica y desarrolla la fase de experimentación con modelos. Para ello se expone como se ha realizado la extracción de características, que algoritmos se han utilizado durante la experimentación, como se ha realizado el tratamiento de los datos y como se ha realizado el entrenamiento y la evaluación de los modelos.

En la subsección 5.2 se exponen los resultados mediante gráficas y se expone el análisis realizado para seleccionar el modelo que mejor se adecúa al problema.

Por último en la subsección 5.3 se exponen detalles de la fase de desarrollo entre los que se incluyen el motivo de las tecnologías utilizadas y problemas durante el desarrollo.

5.1. Fase de experimentación

Extracción de características mediante un Trabajo de Fin de Grado previo

Para la extracción de características se hizo uso de un apartado del Trabajo de Fin de Grado de Catalín llamado PADDEL [4]. Este TFG

consiste en detectar mediante la bradicinesia si una persona tiene o no Parkinson. Para ello extrae las características de vídeos haciendo el RFTT mediante visión artificial y, aplicando técnicas de minería de datos, es capaz de predecir si una persona tiene o no la enfermedad de Parkinson.

Realizando un estudio del TFG de Catalín se pudo extraer y utilizar correctamente el código que permitía la extracción de características. Este código, tras algunas modificaciones, ha sido implementado en el proyecto.

Las características que se extraen de los vídeos, se pueden dividir en tres grupos:

- Atributos iniciales: son los atributos que de los que se dispone mediante el titulo del vídeo como son la edad, el sexo o la mano dominante.
- Atributos extraídos del vídeo: son los atributos que extrae mediante series temporales como son la velocidad, la amplitud o el número de golpes.
- Atributos extraídos del vídeo con TSFresh: Son los 794 atributos del vídeo extraídos mediante TSFresh.

Análisis y tratamiento de los datos

Una vez se extrajeron las características de los datos se pudo llevar a cabo un análisis de los datos. En el *dataset* utilizado durante la fase de experimentación se disponía de 222 vídeos. Cada video disponía de los siguientes atributos:

- Sexo de la persona.
- Edad de la persona.
- Mano dominante de la persona.
- Mano que aparece en el vídeo.
- Fecha en la que se ha tomado.
- Valoración del grado de lentitud del vídeo. Primera variable objetivo.
- Valoración del grado de amplitud del vídeo. Segunda variable objetivo.

Las valoraciones fueron facilitadas por dos neurólogas del Hospital Universitario de Burgos. Estas valoraciones diferían entre sí de manera considerable y es por ello se decidió realizar el experimento utilizando solo las clasificaciones de una de las dos.

Una vez analizados los datos y extraídas las características se hicieron varias comprobaciones. Primero se comprobó que todos los vídeos estuviesen correctamente valorados. Este no fue el caso ya que 2 de los vídeos de los que se disponía no estaban valorados es por ello que, debido a que el tipo de aprendizaje utilizado es supervisado, se decidió que no se incluirían en la fase de experimentación.

Tras eliminar los vídeos que no fueron valorados, se comprobó si las clases estaban desbalanceadas. Al realizar esta comprobación se vio que había un desbalanceado muy elevado, habiendo clases que solo tenían menos de 8 instancias. Dado que se pretendía utilizar algoritmos como KNN o técnicas de remuestreo como *Random Oversampling* se tomo la decisión de eliminar aquellas clases que tuviesen menos de 8 instancias. Las clases que se eliminaron fueron lentitud 3 y 4 así como amplitud 4.

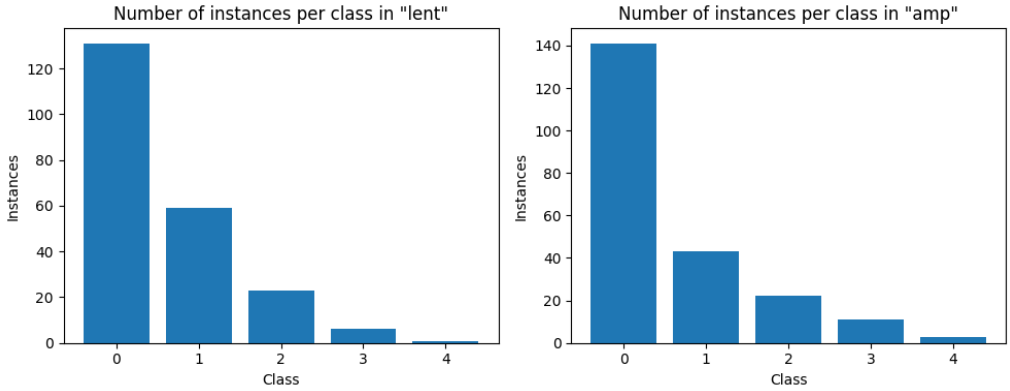

Figura 5.1: Número de instancias por clase antes de realizar el tratamiento de datos.

Además de lo anteriormente mencionado se comprobó si existía alguna característica de los vídeos que fuese nula. Al comprobarlo se observó que en algunas todas había una característica que era nula por lo que se decidió eliminar esa característica nula de todas las instancias ya que no aportaba ninguna información adicional. Una de esas características era la edad, que solo fue nula en vídeos de control, es decir vídeos de personas sin la enfermedad de Parkinson y dado al elevado número de vídeos de control,

no supuso una gran perdida de información. Posteriormente se eliminaron aquellos vídeos cuya duración no era superior a 15 segundos ya que, según aparece en la documentación de PADDEL, no se realizaba correctamente la extracción de características si el vídeo no cumplía ese requisito.

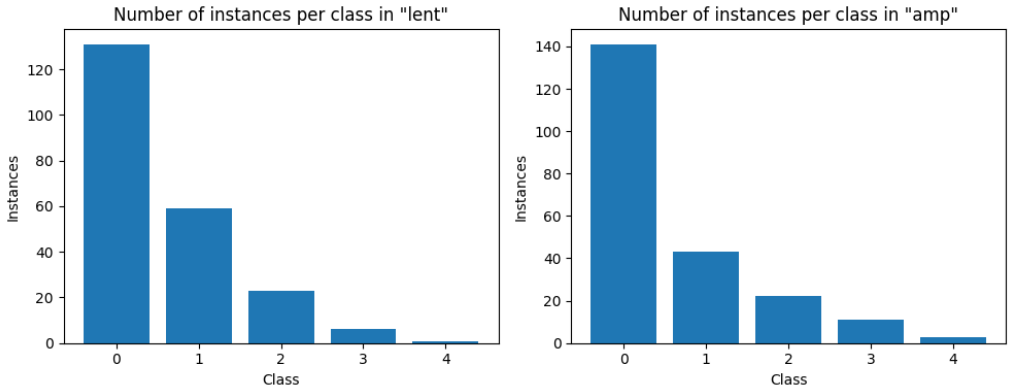

Figura 5.2: Número de instancias por clase tras realizar el tratamiento de datos.

Tras eliminar las instancias anteriormente mencionadas, se realizó un tratamiento de datos en el que las característica categóricas se transformaron en numéricas para poder entrenar correctamente los modelos.

Elección del modelo

Una vez se trataron los datos correctamente, se realizó una selección de los modelos a evaluar. Para ello se eligieron aquellos modelos que mejor funcionaron en las pruebas de PADDEL así como los que obtuvieron mejores resultados en el artículo «Computer Vision for Parkinson's Disease Evaluation: A Survey on Finger Tapping» [1]. Los algoritmos seleccionados fueron KNN, *Random Forest*, XGBoost y SVM.

Para la comparación de modelos, se utilizo *GridSearchCV*, una técnica de validación cruzada que nos permite evaluar los mejores hiperparámetros para un modelo. Para ello el modelo se entrena utilizando una parte del *dataset* como entrenamiento y la restante como prueba. Para estas pruebas se decidió utilizar 320 características de las 807 resultantes tras el tratamiento de los datos. Este dato fue extraído de la documentación del proyecto de PADDEL [4] y probado durante los experimentos viendo que era el que mejor resultados reportaba (ver comparativa en las figuras 5.3 y 5.4). Es

importante destacar que para entrenar cada modelos, las estadísticas que se seleccionan difieren ya que no tienen las mismas correlaciones.

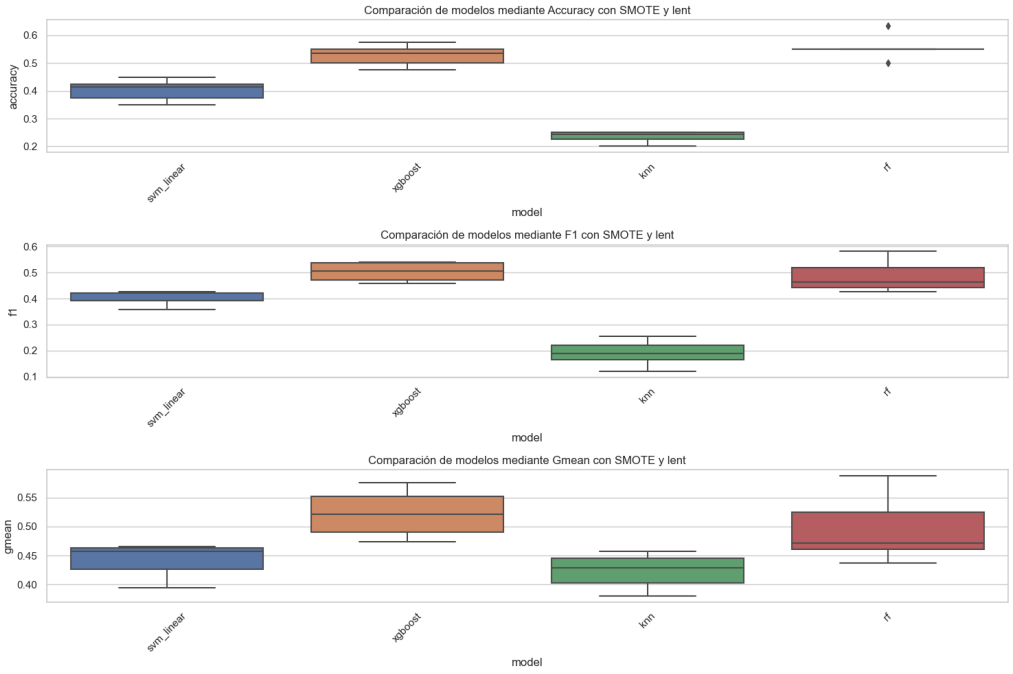

Figura 5.3: Resultados de modelos con todas las características y SMOTE

Durante la fase de entrenamiento se hizo uso tanto de *Random Over-Sampling* (ROS) como de *Synthetic Minority Over-sampling Technique* (SMOTE). Estas técnicas son esenciales en problemas de desequilibrio de clases, ya que permiten entrenar el modelo haciendo que todas las clases tengan el mismo número de instancias y evitar que el modelo ignore las clases minoritarias y pueda predecirlas adecuadamente. Al utilizar este tipo de técnicas hay que tener en cuenta que solo se ha de hacer remuestreo sobre los datos de entrenamiento ya que los datos de test no se deben modificar.

Esto supuso un problema ya que internamente *GridSearchCV* no te permite separar la parte de entrenamiento y test y hacer el *Over-Sampling* correctamente. Es por ello que se decidió hacer de manera externa al entrenamiento del modelo una validación cruzada estratificada dividida en 5 grupos. En ella 4 de los 5 grupos eran utilizados para entrenar el modelo y el restante se utilizaba para evaluar el rendimiento del modelo con los datos no modificados.

Dado que este problema se componía de dos variables objetivo, se opto por realizar modelos independientes para cada clase de manera que cada modelo predijese una variable. Es por ello que, durante el entrenamiento de cada una de las variable objetivo, la variable que no se predecía se omitía y no se introducía durante los entrenamientos.

5.2. Resultados

Tras realizar el tratamiento de los datos y el entrenamiento de los modelos se obtuvieron 20 grupos de evaluaciones para cada modelo. Estos grupos están compuestos de las siguientes evaluaciones:

- *Accuracy*
- *F-score*
- *G-mean*

Cada modelo fue evaluado con cinco veces, una por cada grupo dentro de las divisiones de la validación cruzada y dentro de cada grupo fue evaluado utilizando ROS y SMOTE.

Para realizar acabo estas evaluaciones, se utilizó tanto en la medida F-score como en la *G-mean* el parámetro de *average* en *weighted* debido al gran desbalanceo entre clases ya que, si se hubiese indicado *macro*, debido a que internamente realiza una media aritmética entre cada *F-score* de cada clase, no se habría tenido en cuenta el desbalanceo.

Para obtener una información más general sobre los modelos y sus resultados se decidió que las gráficas que representaran los valores fuesen diagramas de cajas y bigotes, lo cual permitió un mejor visión general del cada modelo.

Lentitud

Para el caso de la lentitud, se hicieron de forma paralela la evaluación del modelo para la lentitud utilizando SMOTE y ROS (ver figuras 5.4 y 5.5 respectivamente).

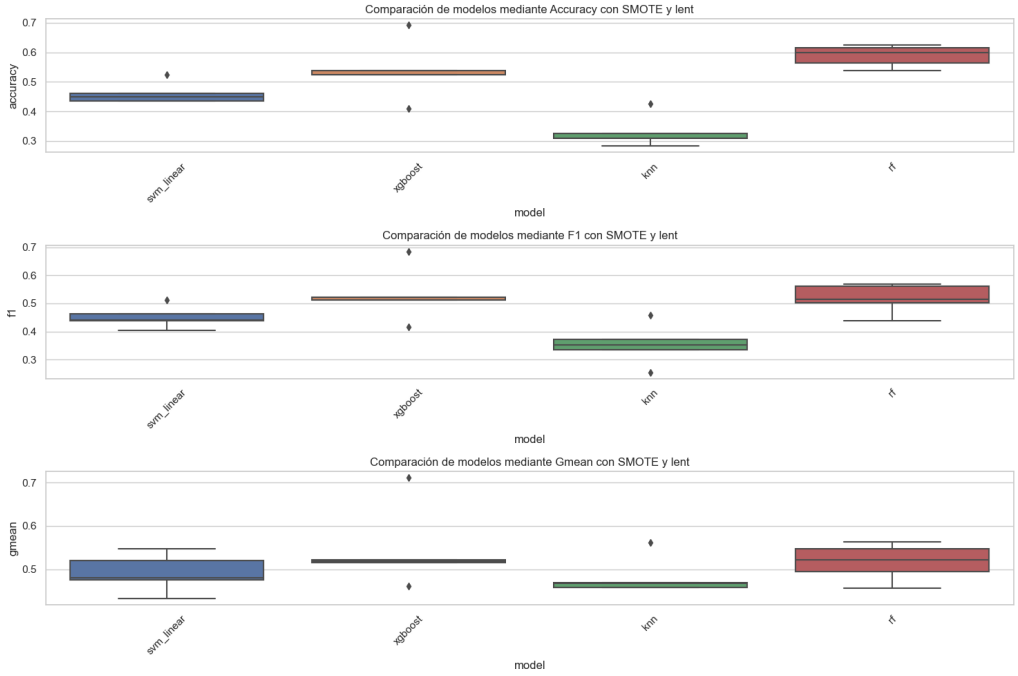

Figura 5.4: Resultados de modelos con 320 características y SMOTE.

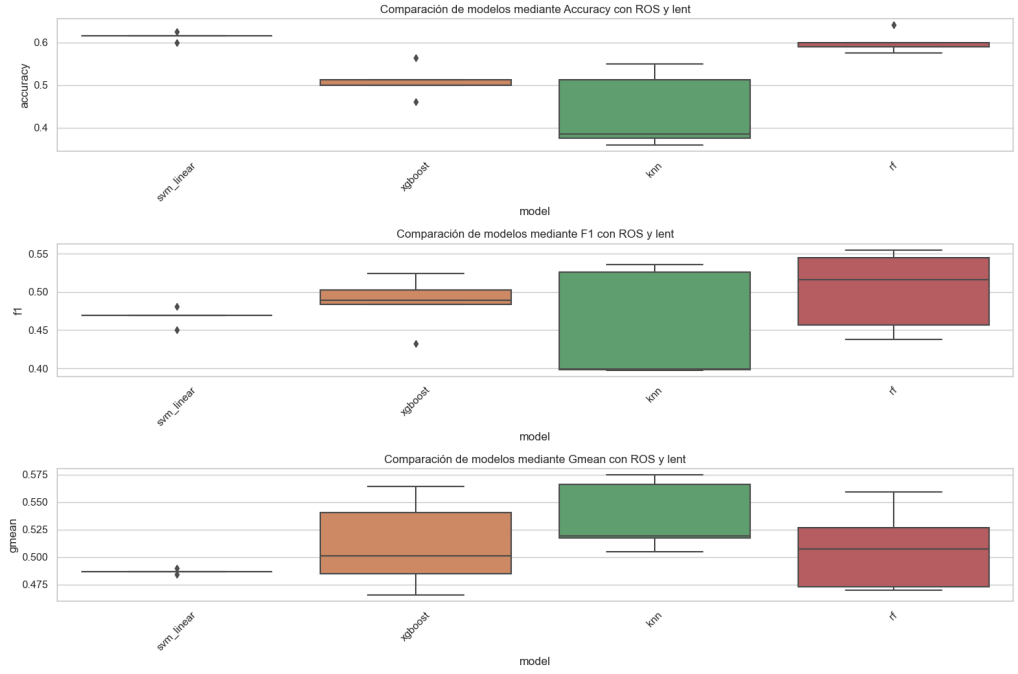

Figura 5.5: Resultados de modelos con 320 características y ROS.

Para realizar las comparativas entre los modelos, se omitió la *Accuracy* ya que, en una clasificación multiclase, no es una buena medida para indicar como de bueno es el modelo. Es por esto que las medidas utilizadas para la comparación entre los modelos fueron el *F-score* y *G-mean*. No obstante, no se descartó la medida de *Accuracy* durante la experimentación ya que proveía una vista general del rendimiento de los modelos frente a un problema desconocido.

En términos generales se puede observar que SMOTE tiene mejor desempeño que ROS. Tras descartar ROS, la comparación entre los modelos muestra que el algoritmo de KNN tiene un desempeño peor comparado con respecto al resto de modelos. Es por ello que queda descartado como modelo candidato. Entre los modelos que quedan *XGBoost* y *Random Forest* destacan sobre el SVM, descartandolo así como modelo candidato. Por último, entre los dos modelos restantes, *XGBoost* pese a tener una mediana peor, posee un valor máximo que destaca en gran medida con respecto a *Random Forest*. Es por ello que se decidió utilizar *XGBoost* como modelo para la lentitud.

Dado que este diagrama es la representación de cinco iteraciones, los mejores hiperparámetros son aquellos que proporcionen el valor máximo. Pese a tener dos valores máximos para dos medidas distintas, los valores pertenecen a la misma instancia por lo que solo tenemos un grupo de hiperparámetros.

Para el caso de la lentitud los hiperparámetros fueron:

- *grow_policy*: *depthwise*.
- *learning_rate*: 0,1.
- *Number of estimators*: 200.

Este modelo con los hiperparámetros indicados obtuvo un *F-score* de 0,69 y un *G-mean* de 0,71 siendo estos los mejores valores reportados entre todos los modelos.

Amplitud

Al igual que en el caso de la lentitud, se hicieron de forma paralela la evaluación del modelo para la lentitud utilizando ROS y SMOTE (ver figuras 5.6 y 5.7 respectivamente).

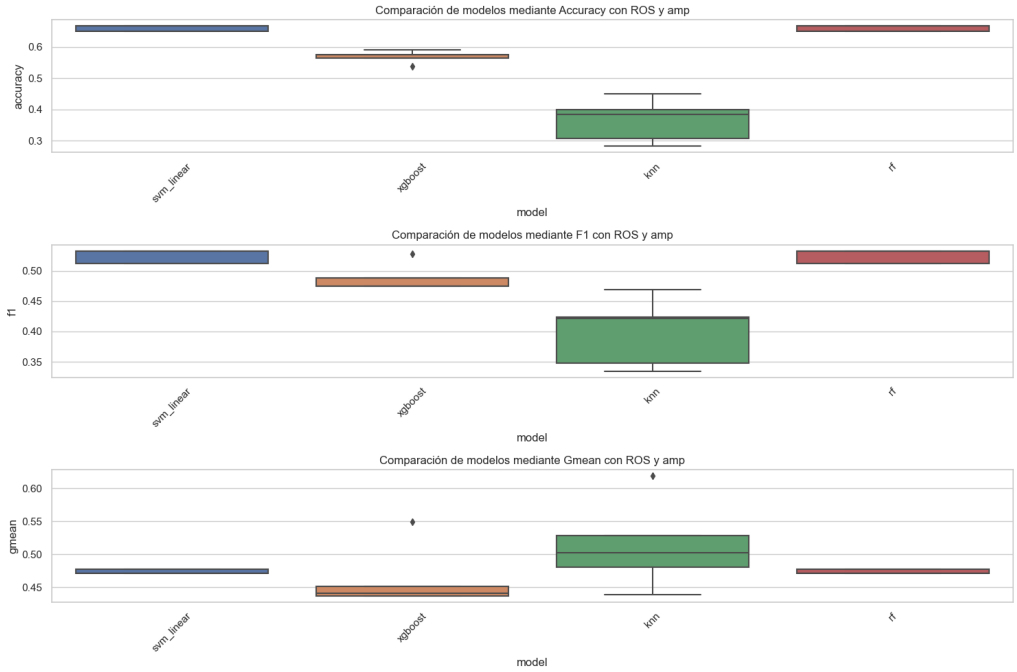

Figura 5.6: Resultados de modelos con 320 características y ROS.

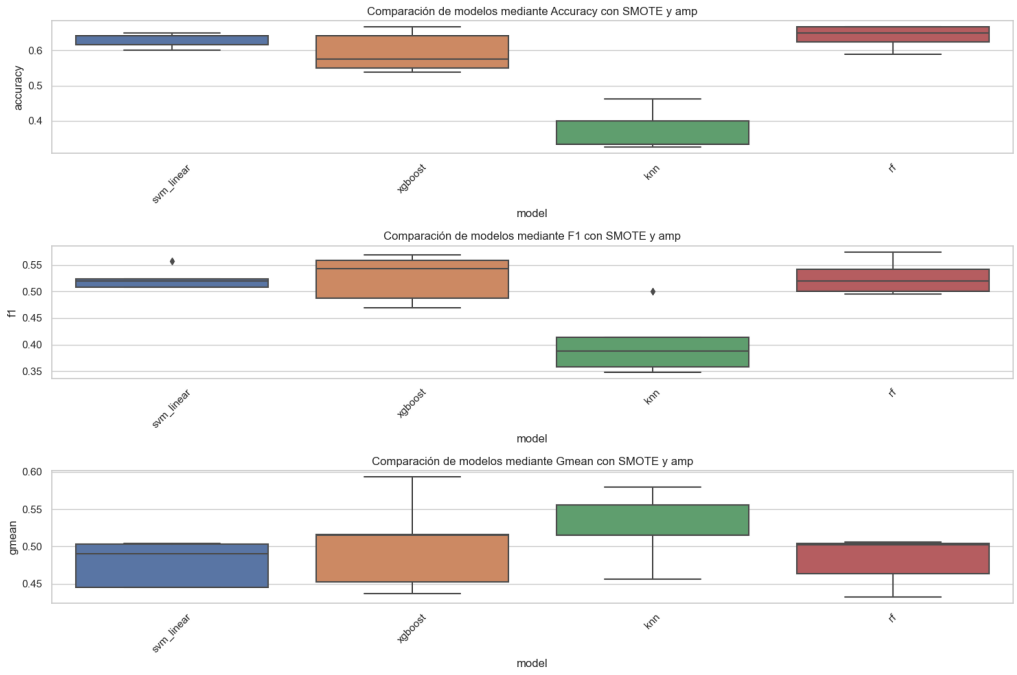

Figura 5.7: Resultados de modelos con 320 características y SMOTE.

En el caso de la amplitud, la diferencia entre ROS y SMOTE no es tan pronunciada; sin embargo, podemos observar una diferencia notoria en el caso de *XGBoost* el cual se ve claramente beneficiado por SMOTE. El resto de modelos, aunque no de forma tan notoria también se ven beneficiados por SMOTE. Es por ello que se decide tomar SMOTE como método de remuestreo como se hizo para la lentitud.

Parecido a lo que ocurría con la lentitud KNN tiene peor desempeño en el caso de *F-sore* y SVM, *Random Forest* y *XGBoost* tienen resultados muy parecidos. En el caso de *G-mean* tanto SVM como *Random Forest* presentan un peor rendimiento, mientras que los dos modelos restantes obtienen mejores resultados.

Dentro de las comparativas de SMOTE, vuelve a destacar *XGBoost* que, en esta ocasión, posee un mejor rendimiento general mejor que el resto de modelos a diferencia de lo que ocurría con la lentitud que se eligió por el caso máximo.

Como ya se explicó anteriormente al ser diagramas de cajas y bigotes, se elegirá el máximo ya que reporta los mejores rendimientos.

Para el caso de la amplitud los hiperparámetros fueron:

- *grow_policy*: *depthwise*.
- *learning_rate*: 0,1.
- *Number of estimators*: 800.

Este modelo con los hiperparámetros indicados obtuvo un *F-score* de 0,69 y un *G-mean* de 0,71 siendo estos los mejores valores reportados entre todos los modelos.

5.3. Fase de desarrollo de la aplicación

Calidad del código

Para evaluar y garantizar la calidad del código se hizo uso de la herramienta *SonarCloud*. *SonarCloud* es una plataforma de análisis de código y gestión de la calidad del software en la nube. Ofrece herramientas para detectar problemas en el código, tales como errores, vulnerabilidades, y código duplicado, y promueve prácticas de desarrollo de software de alta calidad.

Algunas de sus características principales son:

- **Análisis de Código**: Examina el código fuente para identificar errores y vulnerabilidades, asegurando que el software sea robusto y seguro.
- **Integración Continua**: Se integra con herramientas de integración continua (CI) como Jenkins, GitHub Actions, Azure DevOps, y más, permitiendo análisis automáticos con cada commit o push al repositorio.
- **Soporte Multilenguaje**: Soporta una amplia variedad de lenguajes de programación, incluyendo Java, JavaScript, TypeScript, C#, C++, Python, PHP, y muchos más.
- **Informes y Métricas**: Proporciona informes detallados y métricas sobre la calidad del código, facilitando el seguimiento del progreso y la identificación de áreas que necesitan mejora.

El uso de esta herramienta ha sido clave ya que ciertas vulnerabilidades, que no se habían tenido en cuenta por desconocimiento de su solución o de las propias vulnerabilidades, fueron solucionadas de forma ágil ya que la propia herramienta muestra las soluciones y la ubicación de los errores.

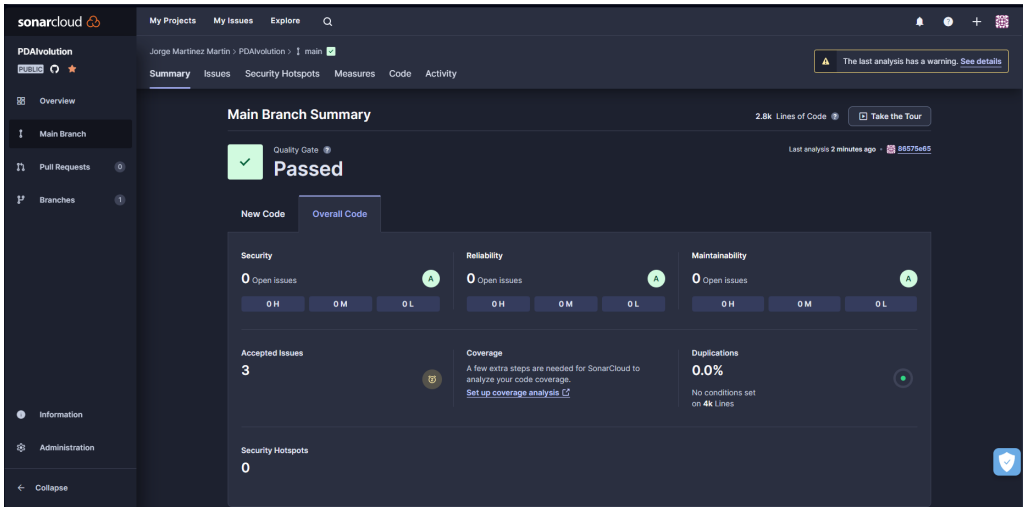

Figura 5.8: Resumen del reporte de *SonarCloud* sobre PDAIvolution.

Como se observa en el reporte de *SonarCloud* (ver figura 5.8), existen varias categorías principales:

- ***Security***: muestra el número de problemas de seguridad que tiene la aplicación. Suelen tratarse de problemas como dejar visibles claves secretas, no tener en cuenta vulnerabilidades . . . En el caso del proyecto, no existe ningún problema de seguridad.

- ***Reliability***: muestra el número de problemas de fiabilidad que tiene la aplicación. Se tratan de problemas como el uso de una variable antes de su declaración. En el caso del proyecto, no existe ningún problema de fiabilidad.

- ***Maintainability***: muestra el número de problemas de mantenimiento que tiene la aplicación. En el caso del proyecto, no existe ningún problema de mantenimiento.

- ***Duplications***: muestra el porcentaje de líneas duplicadas que tiene la aplicación. En el caso del proyecto, el porcentaje de líneas duplicadas es 0 %.

- ***Accepted Issues***: muestra el número de problemas marcados como aceptados, es decir, problemas que se tiene conocimiento de ellos pero no se van a solucionar. En el caso del proyecto tiene marcados tres problemas como aceptados, dos de los cuales vienen dados por el proyecto PADDEL y el restante se debe a que la solución para el problema resulta más compleja que el problema en sí.

- ***Security Hotspots***: son secciones del código que pueden no ser vulnerabilidades de seguridad por sí mismas, pero que merecen una revisión más detallada por parte de un desarrollador experimentado.

Por lo visto en el resumen, el proyecto, a excepción de los problemas previamente comentados, cumple los criterios de calidad de código.

Problemas durante el desarrollo

Planificación temporal

Durante el desarrollo de la aplicación, hubo dos problemas principales a la hora de realizar la parte de planificación temporal. El primero de ellos fue no conseguir conectar la aplicación de Zube con el repositorio de GitHub haciendo que este crease los *issues* de forma automática, por lo que se optó por indicar las tareas que se realizaban en cada uno de los *commits*. El segundo problema se dio cuando el proyecto ya estaba avanzado. Este problema fue la pérdida total del proyecto en Zube. Debido a un fallo, que

todavía se desconoce, el proyecto se eliminó por completo de la plataforma, eliminando a su vez todas las tareas de cada *Sprint* así como la duración de los mismos. Este problema no se logró solucionar, por lo que no se pudo recuperar ninguna información del proyecto.

Compatibilidad

Tanto durante la implementación del proyecto PADDEL como durante la fase de experimentación y despliegue, ha habido problemas de compatibilidad.

Este problema de compatibilidad viene dado por la versión de CUDA [2]y numba [3]. CUDA (Compute Unified Device Architecture) es una plataforma de computación paralela y un modelo de programación creada por NVIDIA y numba es un compilador justo a tiempo (JIT) para Python que traduce funciones numéricas escritas en Python y NumPy a código máquina altamente optimizado utilizando el compilador LLVM. El problema se da cuando la versión de CUDA y numba no son compatibles entre sí. Cuando se da este problema, se ha de modificar las versiones de uno de los dos para que sean compatibles; sin embargo, al hacer la experimentación y despliegue en un servidor donde no se puede modificar la versión de CUDA, se termino resolviendo mediante la desactualización de la versión de numba provocando a su vez una desactualización de la versión de Python de la 3.12 a la 3.10. Es por este motivo por el que el proyecto no se encuentra en la versión más actualizada.

Despliegue

Por último, se destaca el despliegue en un servidor Linux real. Primero se planteo el despliegue en un servicio gratuito pero, debido al gran tamaño de la aplicación y de los datos con los que se iba a trabajar, se descartó esta opción tras comprobar el reducido tamaño que ofrecen los servicios gratuitos. Tras esto, se opto por el despliegue en un servidor proporcionado por la Universidad de Burgos asegurando que la aplicación estuviera operativa y accesible de manera continua y confiable, pudiendo satisfacer las necesidades de uso. Durante la fase de desarrollo la aplicación fue probada en un entorno local haciendo uso de la herramienta integrada de Flask que permite lanzar un servidor de prueba y modificarlo en tiempo real. No obstante, a la hora del

[2]Fuente: https://docs.nvidia.com/cuda/cuda-toolkit-release-notes/index.html

[3]Fuente: https://numba.pydata.org/

paso a producción, la forma en la que se despliega es mediante la herramienta de Gunicorn [4]. Gunicorn *Green Unicorn* es un servidor HTTP Python WSGI para UNIX.

A su vez se creo una imagen de *DockerHub* [5] (disponible en `https://hub.docker.com/repository/docker/jornez/pdaivolution/general` que permite hacer un despliegue muy sencillo sin necesidad de instalar nada a excepción de *Docker* o *Docker Desktop*.

[4]Fuente: `https://gunicorn.org/`
[5]Fuente: `https://hub.docker.com/`

6. Trabajos relacionados

Durante los últimos años se han utilizado diferentes métodos para evaluar la enfermedad de Parkinson mediante la prueba del *Rapid Finger-tapping*. En este capítulo se recogen algunos de los trabajos de manera resumida.

6.1. A computer vision framework for finger-tapping evaluation

En este artículo [12] se documenta el uso de la visión artificial para la clasificación de individuos según el nivel de gravedad de la enfermedad de Parkinson.

Se emplea un método que utiliza la cara para la calibración de la amplitud de la prueba. Para ello el sujeto ha de elevar las manos a la altura de la cara y apuntar con las puntas de los dedos hacia la misma.

El estudio se realizó con 387 vídeos de *rapid finger-tapping test* (RFT) de 13 pacientes diagnosticados con Enfermedad del Parkinson en estado avanzado y 84 vídeos de *rapid finger-tapping test* de 6 personas de control sanas. En total 471 vídeos.

Metodología

- Se realiza un reconocimiento facial del individuo y se crea a partir de el 2 cuadrados en los costados de la cara para realizar la medición de la amplitud.
- Se genera una serie temporal que representa la distancia desde el dedo indice al pulgar siendo esta la amplitud del movimiento.

- Se extraen algunas características de la serie temporal, por ejemplo, la velocidad media de apertura y cierre de dedos, el número total de toques de los dedos, la amplitud máxima...
- Se selección las características no redundantes mediante el algoritmo chi-cuadrado
- Se entrena una máquina de vectores de soporte (SVM) mediante las características obtenidas para realizar la clasificación.

Resultados

Se encontró una nueva característica representativa del ritmo de golpeteo llamada «correlación cruzada entre los picos normalizados», la cual mostró una fuerte correlación de Guttman con las valoraciones clínicas. Al utilizar el clasificador de máquina de vectores de soporte y una validación cruzada de 10 grupos, se logró categorizar las muestras de pacientes en los niveles UPDRS-FT con una precisión del 88 %. Este mismo esquema de clasificación también permitió discriminar entre las muestras **RFT** de los controles sanos y los pacientes con **EP**, logrando una precisión del 95 %.

6.2. The discerning eye of computer vision

Este artículo [23] utilizan 133 vídeos de manos realizando el **RFT** procedentes de 39 pacientes de enfermedad de Parkinson y 30 pacientes de control.

A través de estos vídeos se ha logrado extraer características y comprobar la relación que existe entre estas y la gravedad de la enfermedad en un paciente siendo esta valorada desde 0 (normal) hasta 4 (enfermedad muy grave).

Metodología

- Se utiliza la librería de DeepCutLab la cual se trata de una librería de visión artificial para obtener la serie temporal de la amplitud.
- Se normaliza la serie con la amplitud máxima siendo igual a 1 y escalando los valores conforme a esta medida.
- Al igual que en el estudio anterior se extraen ciertas características de la serie siendo estas la velocidad, la amplitud y el ritmo.

Resultados

DeepLabCut rastreó y midió con fiabilidad el *finger-tap* en un vídeo estándar de smartphone. Las medidas del ordenador se relacionaron bien con las valoraciones clínicas de la bradicinesia.

6.3. Supervised classification of bradykinesia

Este artículo [22] utiliza 70 vídeos de evaluaciones de *finger-tap* en un entorno clínico (40 manos con **Parkinson**, 30 manos de control). Dos expertos clínicos en **Parkinson**, que desconocían los diagnósticos, evaluaron los vídeos para dar un grado de gravedad de la bradicinesia entre 0 y 4 utilizando la Escala Unificada de Calificación de la Enfermedad de Parkinson (UPDRS, por sus siglas en inglés)

Metodología

- Extraer la frecuencia: La frecuencia de intervención se estimó como la frecuencia correspondiente al pico de amplitud máxima en el espectro de la transformada rápida de Fourier (FFT).
- La densidad espectral de energía se calculó como la integral al cuadrado del espectro FFT, una medida que se espera que aumente con la amplitud del golpeteo.

Resultados

Una máquina de vectores de soporte con núcleos de función de base radial predijo la presencia de bradicinesia leve/moderada/grave con una precisión de prueba estimada del 0,8 %. Un modelo Naïve Bayes predijo la presencia de la enfermedad de Parkinson con una precisión de prueba estimada de 0,67.

6.4. Paddel

Este proyecto [4] utiliza 158 vídeos de evaluación etiquetados para determinar la presencia o ausencia de Parkinson. El objetivo es realizar una predicción binaria sobre la presencia de la enfermedad de Parkinson en una persona

Metodología

- Extracción de «landmarks»: Convertir el vídeo de la prueba de «finger-tapping» en una secuencia temporal de puntos de referencia de la mano.
- Lectura de imágenes del vídeo: haciendo uso de la librería OpenCV en Python para leer las imágenes del vídeo sin cargar todo el archivo en memoria.
- Extracción de «landmarks» de las imágenes: Utilizar Mediapipe Hands para obtener puntos de referencia de la mano en cada imagen del vídeo.
- Manejo de oclusión de «landmarks»: Descartar puntos de referencia no fiables debido a la perspectiva de los vídeos de «finger-tapping».
- Extracción precisa de «landmarks»: Elegir no paralelizar el proceso para obtener resultados más precisos en la aplicación web.
- Extracción de series temporales: Generar una secuencia de datos que represente la pose de la mano en cada fotograma del vídeo.
- Extracción de características: Obtener atributos útiles para entrenar el modelo de aprendizaje a partir de las series temporales.

Resultados

Una máquina de vectores de soporte con los hiperparámetros optimizados por medio de *GridSearch* da una precisión media del 78,85 %.

7. Conclusiones y Líneas de trabajo futuras

Bibliografía

[1] Javier Amo-Salas, Alicia Olivares-Gil, Álvaro García-Bustillo, David García-García, Álvar Arnaiz-González, and Esther Cubo. Computer vision for parkinson's disease evaluation: A survey on finger tapping. *Healthcare*, 12(4):439, February 2024.

[2] Font Awesome. Font awesome. https://fontawesome.com/, 2024.

[3] Bootstrap. Bootstrap · the most popular html, css, and js library in the world. https://getbootstrap.com/, 2024.

[4] Catalin Andrei Cacuci. Github:paddel. https://github.com/cataand/tfg-paddel, 2024.

[5] DrawIO. Drawio. security-first diagramming for teams. https://drawio.com/, 2024.

[6] Flask. Guía del usuario. https://flask-es.readthedocs.io/, 2024.

[7] Flask-FontAwesome. Flask-fontawesome. https://pypi.org/project/Flask-FontAwesome/, 2024.

[8] Flask-Login. Flask-login. https://flask-login.readthedocs.io/en/latest/, 2024.

[9] Github. Github. let's build from here. https://github.com/, 2024.

[10] Pengwei Guo, Weina Meng, Mingfeng Xu, Victor C. Li, and Yi Bao. Predicting mechanical properties of high-performance fiber-reinforced cementitious composites by integrating micromechanics and machine learning. *Materials*, 14(12):3143, June 2021.

[11] Jinja. Jinja templating engine. https://jinja.palletsprojects.com/en/3.1.x/, 2024.

[12] Taha Khan, Dag Nyholm, Jerker Westin, and Mark Dougherty. A computer vision framework for finger-tapping evaluation in parkinson's disease. *Artificial Intelligence in Medicine*, 60(1):27–40, January 2014.

[13] Phyton. 3.12.3 documentation. https://docs.python.org/3/, 2024.

[14] Werner Poewe, Klaus Seppi, Caroline M. Tanner, Glenda M. Halliday, Patrik Brundin, Jens Volkmann, Anette-Eleonore Schrag, and Anthony E. Lang. Parkinson disease. *Nature Reviews Disease Primers*, 3(1), March 2017.

[15] R. Bharat Rao, Sriram Krishnan, and Radu Stefan Niculescu. Data mining for improved cardiac care. *ACM SIGKDD Explorations Newsletter*, 8(1):3–10, June 2006.

[16] Diego Santos García, Marta Blázquez-Estrada, Matilde Calopa, Francisco Escamilla-Sevilla, Eric Freire, Pedro J. García Ruiz, Francisco Grandas, Jaime Kulisevsky, Lydia López-Manzanares, Juan Carlos Martínez Castrillo, Pablo Mir, Javier Pagonabarraga, Francisco Pérez-Errazquin, José María Salom, Beatriz Tijero, Francesc Valldeoriola, Rosa Yáñez, Arantxa Avilés, and María-Rosario Luquín. Present and future of parkinson's disease in spain: Parkinson-2030 delphi project. *Brain Sciences*, 11(8):1027, July 2021.

[17] Scikit-Learn. Scikit-learn. machine learning in python. https://scikit-learn.org/stable/, 2024.

[18] SQLAlchemy. Sqlalchemy. the python sql toolkit and object relational mapper. https://www.sqlalchemy.org/, 2024.

[19] Sigurlaug Sveinbjornsdottir. The clinical symptoms of parkinson's disease. *Journal of Neurochemistry*, 139(S1):318–324, 2016.

[20] TSfresh. Tsfresh.time series feature extraction based on scalable hypothesis tests. https://tsfresh.readthedocs.io/en/latest/, 2024.

[21] Werkzeug. Werkzeug documentation. https://werkzeug.palletsprojects.com/en/3.0.x/, 2024.

[22] Stefan Williams, Samuel D. Relton, Hui Fang, Jane Alty, Rami Qahwaji, Christopher D. Graham, and David C. Wong. Supervised classification

of bradykinesia in parkinson's disease from smartphone videos. *Artificial Intelligence in Medicine*, 110:101966, November 2020.

[23] Stefan Williams, Zhibin Zhao, Awais Hafeez, David C. Wong, Samuel D. Relton, Hui Fang, and Jane E. Alty. The discerning eye of computer vision: Can it measure parkinson's finger tap bradykinesia? *Journal of the Neurological Sciences*, 416:117003, September 2020.

[24] Zube. Zube. developer collaboration, solved. https://www.zube.io/, 2024.