



UNIVERSIDAD DE MÁLAGA



Graduado en Ingeniería de la Salud (Bioinformática)

Machine Learning para detección temprana de patologías en
podología

Machine Learning for early detection of pathologies in
podiatry

Realizado por
Carlos Beltrán López

Tutorizado por
José Manuel Jerez Aragónés

Departamento
Lenguajes y Ciencias de la Computación
UNIVERSIDAD DE MÁLAGA

MÁLAGA, septiembre 2023



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADUADA/O EN INGENIERÍA DE LA SALUD

**Machine Learning para detección temprana de
patologías en podología.**

**Machine Learning for early detection of pathologies in
podiatry**

Realizado por
Carlos Beltrán López

Tutorizado por
José Manuel Jerez Aragonés

Departamento
Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, SEPTIEMBRE DE 2023

Fecha defensa: septiembre de 2023

Agradecimientos

Me gustaría agradecer a mi tutor José Manuel Jerez Aragonés por su ayuda y sus valiosos consejos.

A Joaquín Páez por aceptar el trabajo de investigación, facilitar los datos de su clínica de podología y ayudar en cualquier cuestión planteada.

Al personal del laboratorio 3.3.10A que me han ahorrado muchas horas de búsqueda con explicaciones suyas.

A mi tío Manuel por su acompañamiento durante estos años de carrera y por supuesto, por acogerme.

Y con cariño a mis padres, patrocinadores oficiales de mis estudios y mantenimiento.

Resumen:

El auge de la inteligencia artificial que estamos viviendo invita a plantear nuevas aplicaciones de este campo en diferentes ámbitos aún no explorados. Uno de ellos es el de la podología. En el que no hay apenas estudios, investigaciones o proyectos de machine learning aplicado al diagnóstico/tratamiento de patologías relacionadas con el tren inferior.

En este proyecto se ha trabajado junto a la clínica de Podología ”Dr. Páez.” en Málaga con el objetivo de poder crear un modelo entrenado a partir de los datos obtenidos de pacientes sometidos a una serie de pruebas realizadas por un instrumento común entre los profesionales de este campo: *Optogait*. Este adquiere información tal como la longitud de paso, tiempo de los pies en el aire, balanceo del pie... Se ha buscado el modelo óptimo entre todos los entrenados con distintos algoritmos de aprendizaje computacional, usando R y Python y sus respectivas librerías.

Una vez encontrado, además se ha desarrollado una aplicación web con R, RShiny, JS, HTML y CSS para poder predecir la patología de los nuevos pacientes adjuntando los datos tomados con *Optogait*.

Palabras claves: Desarrollo Web, Machine Learning, R, Python, RShiny, Aprendizaje Computacional

Abstract:

The rise of artificial intelligence that we are experiencing invites us to propose new applications of this field in different areas that have not yet been explored. One of them is podiatry. With hardly any studies, research or projects of machine learning applied to the diagnosis/treatment of pathologies related to the lower extremities.

In this project, we worked with the Podiatry Clinic ”Dr. Páez” in Malaga with the objective of creating a model trained from data obtained from patients subjected to a series of tests performed by a common instrument among professionals in this field: *Optogait*. This acquires information such as step length, time of feet in the air, foot balance... The optimal model has been searched among all those trained with different computational learning algorithms, using R and Python and their respective libraries.

Once found, a web application has also been developed with R, RShiny, JS, HTML, and CSS to predict the pathology of new patients by attaching the data taken with *Optogait*.

Keywords: Web Application Developing, Machine Learning, R, Python, RShiny

Índice de contenidos

1. Introducción	1
1.1. Estructura del Documento	1
1.2. Contexto	2
1.3. Motivación	2
1.4. Objetivos	2
1.5. Tecnologías usadas	3
2. Metodología	5
2.1. Limpieza y preprocesamiento de datos	5
2.2. Distribución de variables	7
2.2.1. Análisis Univariante	7
2.2.2. Análisis Multivariante	11
2.3. Flujo de trabajo	15
Dataset con ocho patologías	16
Dataset con cuatro patologías	20
Dataset con tres patologías	23
Dataset con dos patologías	26
3. Resultados	29
4. RShiny app	33
4.1. Desarrollo de la aplicación Web	33
4.1.1. Interfaz gráfica (UI.R)	33
4.2. Servidor (Server.R)	35
4.2.1. Funcionamiento de la aplicación	35
5. Conclusiones y líneas futuras	41
5.1. Conclusiones	41
5.2. Futuras líneas de trabajo	42
Bibliografía	43
Apéndice A. Guía de uso de la aplicación OptoPredict	47

CAPÍTULO 1

Introducción

1.1. Estructura del Documento

El documento será estructurado de la siguiente manera:

- **Introducción:** en la que está incluida este mismo apartado con intención de contextualizar e informar al lector de los objetivos, cómo se ha hecho y por qué motivo.
- **Metodología:** detallada explicación de cada paso tomado desde la obtención de los datos hasta qué algoritmos se han empleado y cómo.
- **Resultados:** análisis de tales resultados obtenidos con los diferentes algoritmos, selección del mejor y justificación.
- **RShiny app:** explicación y muestra de una interfaz creada para facilitar el uso de este modelo obtenido por parte del profesional clínico.
- **Conclusión final y líneas futuras:** reflexión sobre la utilidad de los resultados obtenidos y el camino que abre esta investigación para continuar con otras.

1.2. Contexto

Partiendo de una época en el que la inteligencia artificial está experimentando un drástico alzamiento, muchos estudios y aplicaciones aparecen a diario, investigaciones como la IA aplicada a numerosos campos de todas clases, tanto de ciencias como de letras, orientada hasta a la más mínima actividad que realizamos a diario, como la cocina o encender la luz.

Sin embargo, desde este punto de vista, la podología no ha sido experimentada, hay todavía un gran abanico de posibilidades por descubrir mezclando estos dos ámbitos, lo cual hace su estudio más intrigante aún. La podología enfrenta desafíos en el diagnóstico de patologías del tren inferior, ya que a menudo los síntomas son subjetivos y difíciles de cuantificar. La aplicación de algoritmos de machine learning puede mejorar significativamente la precisión del diagnóstico, al permitir la identificación de patrones en grandes conjuntos de datos.

1.3. Motivación

El interés reside en desarrollar un caso real de aplicación de Machine Learning a una serie de datos con el fin de crear un modelo que pudiese obtener resultados útiles, más allá de la habitual teoría y casos prácticos previamente preparados que hacen parecer la resolución de tales problemas muy sencillos. Además, busca contribuir al avance de la aplicación del machine learning en la podología y proporcionar una herramienta útil, que no substituta, para los profesionales en el diagnóstico y tratamiento de patologías del tren inferior.

Se ha trabajado junto al doctor Joaquín Páez, podólogo y emprendedor de la Clínica Dr. Páez, en Puerta Blanca (Málaga). Joaquín, interesado en aplicar este creciente campo de la inteligencia artificial en podología, nos explicó su idea de utilizar todos los datos recogidos de pacientes suyos durante años con Optogait, un aparato muy utilizado entre los profesionales de la fisioterapia y podología.

1.4. Objetivos

El trabajo de fin de grado consiste en la creación de un modelo de machine learning basado en un algoritmo de clasificación, a partir de datos obtenidos de pacientes sometidos a una serie de pruebas realizadas por un instrumento común entre los profesionales de la podología, *Optogait*. El objetivo es encontrar patrones en los datos que permitan predecir la presencia de patologías del tren inferior en nuevos pacientes.

Se utilizarán técnicas de procesamiento de datos y algoritmos de aprendizaje supervisado en lenguajes como Python y R, para entrenar el modelo. Se explorarán diferentes algoritmos de clasificación y se compararán los resultados obtenidos. Además, se desarrollará una interfaz de usuario para la aplicación del modelo en la práctica clínica.

1.5. Tecnologías usadas

Optogait

Optogait está compuesto de una barra óptica transmisora y una receptora (ver Figura 1.1a). Cada una contiene 96 leds Infrarrojos. Estos leds están ubicados sobre la barra transmisora y se comunican continuamente con los leds ubicados en la barra receptora. El sistema detecta eventuales interrupciones y su duración.



Figura 1.1: Optogait

Esto permite la medición de los tiempos de vuelo y de contacto durante la ejecución de una serie de saltos, con una precisión de 1/1000 de segundo. Partiendo de esta base de datos fundamentales, el software específicamente diseñado, montado en una cinta de correr, permite la obtención, con la máxima precisión y en tiempo real, de una serie de parámetros ligados al rendimiento del atleta.

En la clínica Dr. Páez trabajan con tres principales programas donde guardan diferentes datos de cada paciente en cada uno de ellos.

- **OptoGait:** software propio del dispositivo que toma las medidas previamente explicado. Almacena todos estos datos sobre la marcha de un paciente, con los cuales se va a trabajar en el futuro modelo.
- **S-Plate:** software de otro dispositivo que consiste en una placa de presión que mide el área de pisada plantar de un paciente. En este programa, los profesionales de esta clínica almacenan también la altura, peso y número de pie de la persona. Estos tres datos son los que nos interesan de la base de datos de este programa.
- **Clinni:** una página web de gestión de pacientes contratada por la clínica Dr. Páez. En la base de datos de esta web se incluye la patología que le ha sido diagnosticada al paciente por parte del podólogo profesional.

RStudio y R

RStudio es el entorno por excelencia del lenguaje de programación R. Será usado para el preprocesamiento de los datos en paralelo con Python junto a librerías útiles de machine learning que facilitan todo el proceso.

- **MLR3**: una librería de R que incluye funciones de muy alto nivel para aplicar machine learning a los datos, se puede asemejar a SKlearn de Python.
- **RShiny**: librería de R para desarrollo web que tiene un estilo CSS propio basado en BOOTSTRAP.

Python

Python para machine learning, junto a la librería SKlearn.

HTML

Lenguaje de programación para desarrollo web.

CSS

Lenguaje de programación para estilizar el contenido web creado.

Overleaf

Web online para crear el documento de la memoria en Latex.

Jupyter

Entorno de programación para Python, usado desde Anaconda.

CAPÍTULO 2

Metodología

2.1. Limpieza y preprocesamiento de datos

Se parte de tres bases de datos correspondientes a cada software usado en la clínica (Optogait, S-Plate y Clinni). Estas bases de datos deberán ser unidas desechando toda la información no útil para finalmente obtener un único archivo con todas las variables de interés.

- S-Plate: todos los pacientes testeados con la placa de presión S-Plate en la clínica desde 2017 hasta marzo de 2023. Tras eliminar las variables innecesarias como 'Fecha Creación' y procesar los pacientes de prueba, pacientes duplicados, fallos en sintaxis... nos queda una base de datos con 1664 registros y de formato el siguiente:

ID	Nombre	Apellidos	Peso	Altura	Nº Pie
1	Eduardo	García	80	183	43
2	María	Paredes	65	172	39
3	Alberto	Fernández	93	186	45
4	Manuel	García	71	170	42
⋮	⋮	⋮	⋮	⋮	⋮
1664	Fulanito	Mengano	80	170	43

Tabla 2.1: S-plate

- Clinni: todos los pacientes diagnosticados en la clínica desde el inicio de su uso (alrededor de 2019) hasta marzo de 2023. Una base de datos con 2325 registros y de formato el siguiente:

ID	Nombre	Apellidos	Historial
1	Eduardo	García	quiropodia. Uñas encarnadas, callo...
2	María	Paredes	Onicocriptosis primer dedo lado medial PI
3	Alberto	Fernández	no tiene fasciosis, baxter. bupi en sural y baxter
4	Manuel	García	Sever PD. Rugfoot 39 con taloneras
:	:	:	:
1664	Fulanito	Mengano	Paciente que acude por papiloma en 2o meta

Tabla 2.2: Clinni

- Optogait: el programa que, conectado a toda la maquinaria instalada en la cinta de correr, recoge los datos de las pruebas realizadas a los pacientes, marcha descalzo, marcha con calzado, correr descalzo... Los datos recopilados son mediciones de cada paso (longitud, tiempo de pisada, separación entre pies...), tras eliminar algunas variables innecesarias que se incluyen en este programa como fecha de nacimiento, edad o dirección, la base de datos resultante tiene el siguiente formato :

Nombre	Apellidos	TContacto.media	TContacto.desviacion	...
Eduardo	García	0.75	0.15	...
María	Paredes	0.78	0.13	...
Alberto	Fernández	0.85	0.23	...
Manuel	García	0.71	0.17	...

Tabla 2.3: Optogait

Se formatearon los datos con el uso de diferentes librerías de R y Python para poder procesarlos o Excel para cambiar el tipo de datos o de extensión entre más cosas. Cada base de datos venía en una diferente extensión de archivo (.JSON, .XLS, .WORD...) además de la falta de estructuración de tales datos, que dificultó aún más la labor. Finalmente, se guardó cada base de datos en un archivo .csv distinto.

Usando el Nombre y Apellidos del paciente como clave primaria (Primary Key) se unieron las tres tablas para juntar todas las variables de cada programa en una base de datos común con la que trabajar (ver Figura 2.1).

Tras esta fusión de bases de datos, se desecharon muchos registros de pacientes que no coincidían en las tres bases de datos, pacientes cuya historia era nula o simplemente no era útil para usarla como la patología de salida, otros con muchas variables vacías o creaciones de prueba por parte de los profesionales en la clínica.

A pesar de parecer, en un principio, un problema con muchos datos disponibles para trabajar, tras la limpieza y la unión, la cantidad de pacientes útiles con los que poder entrenar el modelo disminuyó cinco veces lo que parecía haber.

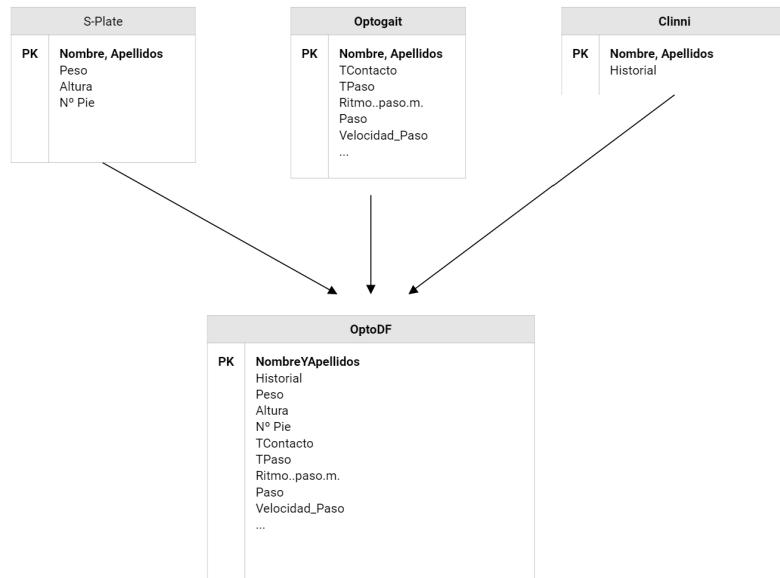


Figura 2.1: Bases de Datos

2.2. Distribución de variables

2.2.1. Análisis Univariante

Trabajamos inicialmente con 61 variables (sin contar la clase etiqueta, es decir, la patología) en nuestro data frame con las que empezaremos a realizar el preprocesamiento de los datos e iremos reduciendo el número de ellas dependiendo de la importancia que tengan.

De cada test de cada paciente se obtiene una medida por paso. A cada una de estas medidas se les ha aplicado la **media** y la **desviación típica**. Al ser medidas de distancia y tiempo, las trabajamos como tipo float.

A continuación se muestran todas las variables (obviadas la mitad de ellas que solo muestran la desviación típica) acompañadas de dos imágenes para facilitar su entendimiento:

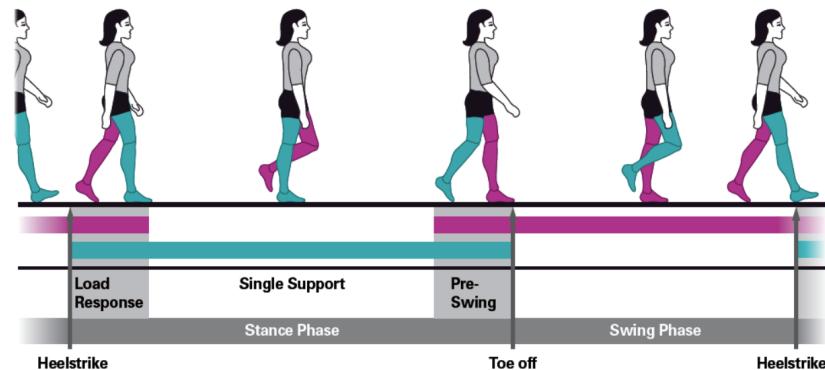


Figura 2.2: Visualización de la Marcha

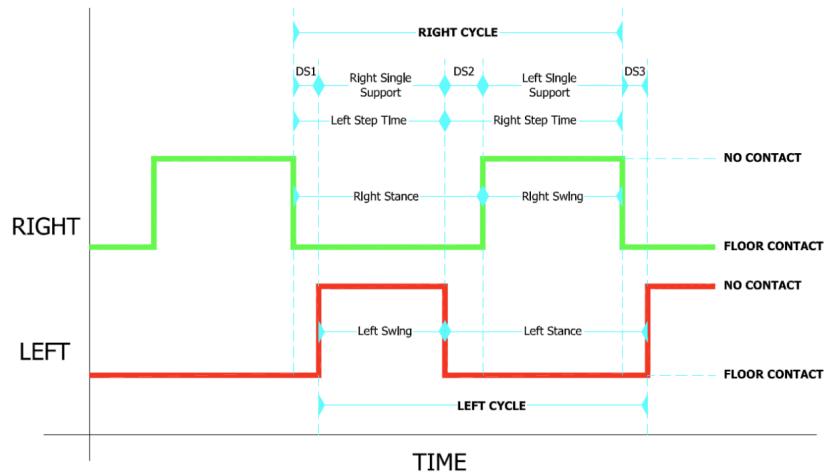


Figura 2.3: Estado de cada pie en la Marcha

Patología: La clase salida, con un valor dependiendo de la patología. En un principio se intentó predecir distintas patologías pero debido a malos resultados se redujo a:

1 → Fascitis

0 → Otro o nada

Peso.Kg: Masa del paciente en KG.

Altura.cm.: Altura del paciente en cm.

N.Pie: Talla del pie (EU).

TContacto.media: Media del tiempo de contacto de cada pie en cada paso.

TPaso.media: Tiempo entre el primer contacto de un pie y el primer contacto del pie contrario.

Ritmo.paso.media: Ritmo expresado en pasos por segundo.

Paso.media: Longitud del paso (cm).

Velocidad.media: Velocidad media referida a un paso (m/s).

Aceleracion.media: Variación de la velocidad en los dos pasos anteriores.

Desequilibrio.media: Índices de desequilibrio entre paso derecho e izquierdo.

Distancia.media: Distancia progresiva recorrida por el paciente (cm).

Zancada.media: Amplitud de la zancada (cm).

StrideTime.Cycle.media: Tiempo entre el primer contacto de dos pasos consecutivos del mismo pie.

SingleSupport.media: Apoyo monopodal(single support) es el tiempo desde el último contacto del apoyo actual hasta el primer contacto del próximo apoyo del mismo pie, es decir, el tiempo que transcurre con el pie apoyado en el suelo. El apoyo individual de un pie equivale al tiempo de oscilación del pie contrario (s).

SingleSupport..media: SingleSupport, pero se expresa como porcentaje del tiempo total del Ciclo de la marcha.

TotalDoubleSupport.media: Es la suma de dos dobles apoyos parciales(s).

TotalDoubleSupport..media: Igual pero representado como porcentaje del tiempo total del Ciclo de la Marcha.

TStance.media: La fase de apoyo (Stance Phase) es la parte de soporte del peso en cada Ciclo de la marcha. Comienza con el contacto del talón y termina con la separación de la punta de los dedos del mismo pie. Por lo tanto, es el tiempo entre el primer y el último contacto de dos apoyos consecutivos en el mismo pie.

TStance..media: También se presenta como un porcentaje del Ciclo de la marcha.

TSwing.media: Es el tiempo entre el último contacto del apoyo del pie y el primer contacto del apoyo sucesivo (s).

TSwing..media: Del mismo modo, pero mostrado en un porcentaje con respecto al tiempo de Contacto Total.

ContactPhase.media: Tiempo desde el primer contacto del talón hasta el apoyo total del pie.

ContactPhase..media: Porcentaje indica el valor con respecto al tiempo de contacto total.

FootFlat.media: Tiempo que toda la planta del pie está en contacto con el suelo.

FootFlat..media: Porcentaje respecto al tiempo contacto total.

PropulsivePhase.media: Tiempo desde el ascenso del talón hasta el levantamiento de la punta del pie.

PropulsivePhase..media: Porcentaje con respecto al tiempo de contacto total.

LoadResponse.media: Es el primer Tiempo de doble apoyo (s).

LoadResponse..media: Valor con respecto al tiempo de contacto total.

PreSwing.media: Es el segundo Tiempo de doble apoyo (s).

PreSwing..media: Valor con respecto al tiempo de contacto total.

Algunas de estas variables se muestran a continuación descritas con datos estadísticos como la media, desviación, percentiles o máximo entre otros. Debajo, el histograma de cuatro de ellas.

	Peso.Kg.	Altura.cm.	N.Pie	TContacto.media	TPaso.media	Zancada.media	TPaso.desviacion	TSwing.desviacion
count	298.000000	298.000000	298.000000	298.000000	298.000000	298.000000	298.000000	298.000000
mean	67.484899	164.201342	40.038591	0.848197	0.603592	108.522747	0.089978	0.052616
std	20.778934	17.108283	3.353377	0.181393	0.120916	18.113814	0.075582	0.069581
min	13.000000	39.000000	27.000000	0.338337	0.169388	75.264151	0.013600	0.010259
25%	55.000000	159.000000	38.000000	0.763942	0.550554	96.135665	0.026441	0.019328
50%	67.100000	167.000000	40.000000	0.879480	0.604601	104.997312	0.070417	0.025780
75%	80.750000	175.000000	43.000000	0.960274	0.651878	115.634512	0.136484	0.059394
max	134.000000	198.000000	47.000000	1.441130	1.312833	179.527211	0.410483	0.611347

Figura 2.4: Descripción de algunas variables

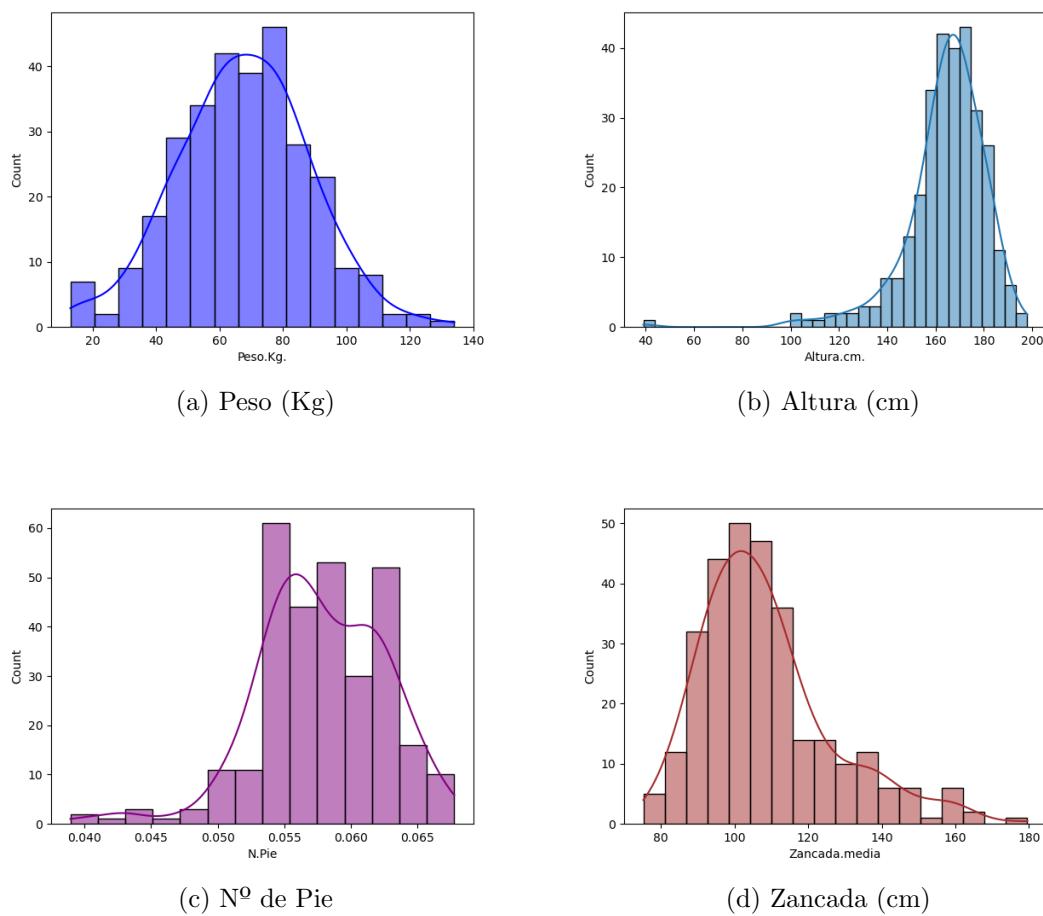


Figura 2.5: Histogramas de cuatro variables

Así como hemos podido observar en las anteriores gráficas, muchas de las variables siguen una distribución normal. De igual modo, se procederá a estandarizar todas las variables como parte del procesamiento para evitar que el modelo otorgue mayor peso a unas variables que a otras por problemas de la escala.

2.2.2. Análisis Multivariante

Empecemos estudiando la correlación entre cada una de las variables y la clase salida (Patología).

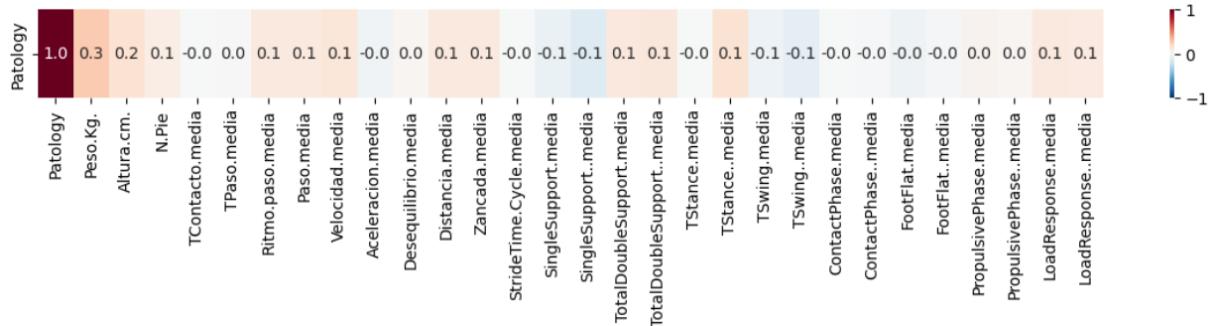


Figura 2.6: Correlación 30 primeras variables

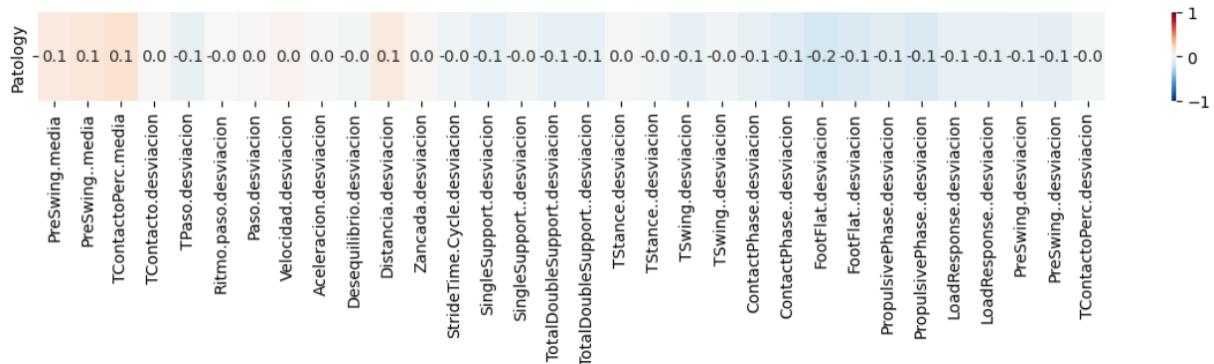


Figura 2.7: Correlación 32 últimas variables

Por desgracia, ninguna de las variables tiene un alto índice de correlación con la clase Patología, siendo la que más el peso con un 0.3, una cantidad apenas significativa. No podemos usar la variación para explicar el comportamiento de las variables, por lo que usaremos otros métodos.

Umbral de Varianza

Aunque la varianza no haya funcionado con respecto a la salida, podemos estudiarla entre las diferentes variables.

Usando la función VarianceThreshold de la librería Sklearn de Python.

```

1 THR = VarianceThreshold(threshold=12)
2 THR.fit_transform(x)

```

Con ese umbral se eligen treinta y un variables:

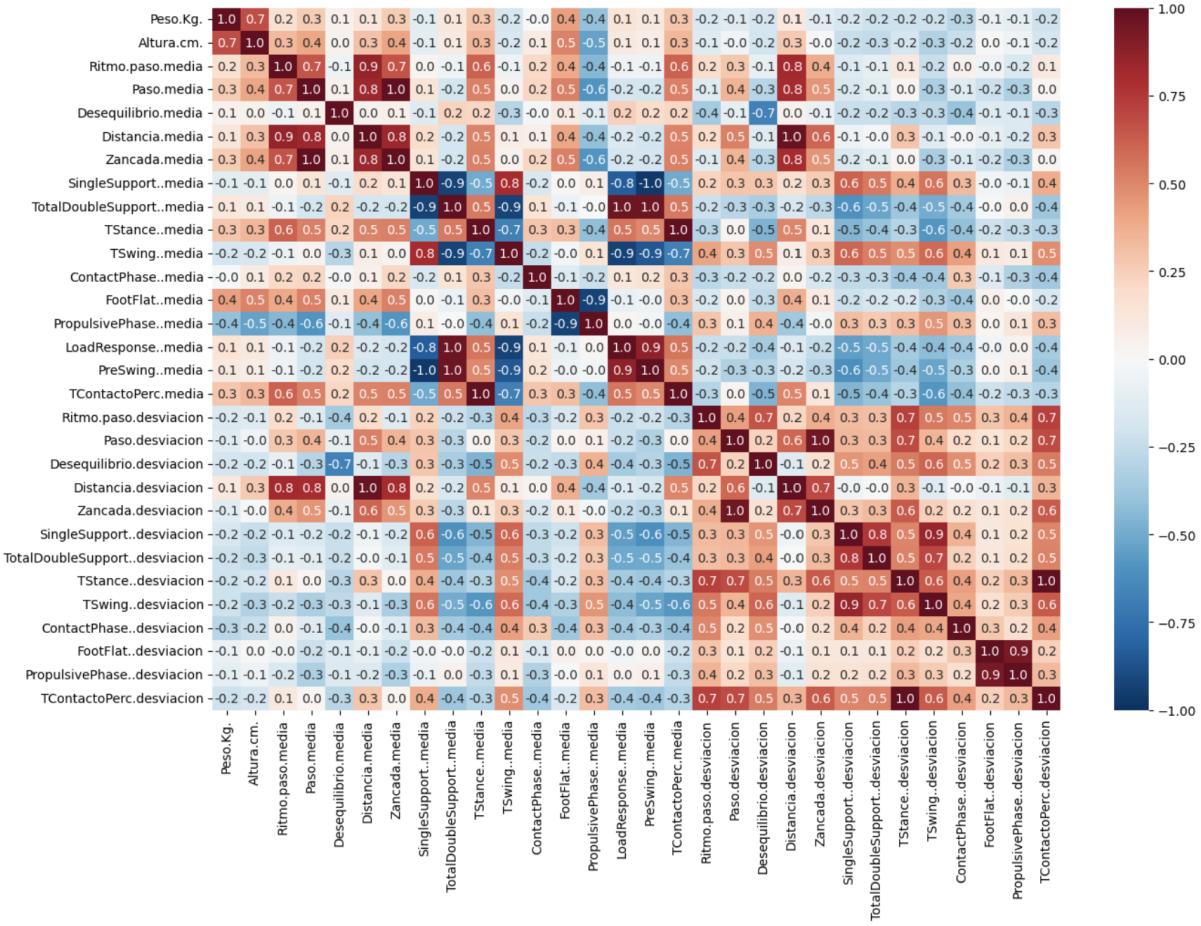


Figura 2.8: Matriz de correlación

A simple vista se distinguen cuatro recuadros, si los enumeramos como los cuadrantes de un gráfico bidimensional, el segundo y el cuarto tienen mayores índices de correlación en general que el primero y el tercero. Esto se debe a la división de las variables en media y desviación al no estar muy influenciadas entre sí estas dos medidas estadísticas.

- En el segundo cuadrante, destacan relaciones previsibles. El peso y la altura, longitud de paso y zancada, tiempo de apoyo de un pie y tiempo de apoyo de ambos...
 - El cuarto cuadrante es el más llamativo porque la mayoría de las desviaciones están relacionadas entre sí, algo lógico si pensamos que cuando hay una alta desviación en el Paso (cm) la va a haber en la Distancia total recorrida, así como en el tiempo de contacto, el tiempo de balanceo...
 - Cabe destacar aquellos valores próximos a 1 o -1 que se encuentran en el primer y tercer cuadrante, los cuales indican una relación entre variables de media y desviación interesantes para su estudio.
- Cuando un paciente tiene alguna dolencia en el pie, tiende a cambiar su manera de andar o a fallar en algún movimiento lo cual influye todos los demás valores. Por eso es más alta la correlación en el cuarto cuadrante.
- En general son variables muy relacionadas entre sí.

Eliminación de variables recursiva (RFE)

A este método (Recursive Feature Elimination) se le indica un algoritmo que asigne pesos a las variables. Empezará entrenando con todas y después seleccionando conjuntos de variables más pequeños hasta tener el número indicado.

Se ha elegido Decision Tree tras probar con varios y no haber mucha diferencia en los resultados de los algoritmos.

```

1 varRFE = DecisionTreeClassifier() # Seleccion de estimador
2 varRFE = RFE(varRFE, n_features_to_select=30, step=1) #
    Parametros
3 varRFE = RFE.fit(x, y.values.ravel()) # Entrenamiento
4 varRFE.get_feature_names_out() # Mostrar las variables
    seleccionadas

```

Seleccionador de variables secuencial (SFS)

El seleccionador de variables secuencial (Sequential Feature Selector) empieza entrenando el modelo con el algoritmo indicado. Usa todas o una variable, dependiendo del parámetro que se le haya indicado 'backward' o 'forward' respectivamente.

```

1 NeuralN = MLPClassifier(max_iter=1000, early_stopping=True,
    activation = 'tanh', solver='sgd', alpha=1, hidden_layer_sizes
    =(10, 5), random_state=1) # Modelo elegido, redes neuronales y
    sus parametros
2 sfs = SequentialFeatureSelector(NeuralN, n_features_to_select=30,
    direction='forward') # SFS
3 sfs.fit(x, y.values.ravel()) # Ejecucion
4 sfs.get_feature_names_out()

```

Tras ejecutar cada uno de estos métodos, obtenemos tres conjuntos de variables seleccionadas como más importantes:

Threshold	REF	SFS
Peso.Kg	Peso.Kg	Peso.Kg
Altura.cm.	Altura.cm.	Altura.cm.
Ritmo.paso.media	Ritmo.paso.media	Ritmo.paso.media
Paso.media	Paso.media	Paso.media
Distancia.media	Distancia.media	Distancia.media
TotalDoubleSupport..media	TotalDoubleSupport..media	TotalDoubleSupport..media
ContactPhase..media	ContactPhase..media	ContactPhase..media
PropulsivePhase..media	PropulsivePhase..media	PropulsivePhase..media
Zancada.desviacion	Zancada.desviacion	Zancada.desviacion
TSwing..media	—	TSwing..media
—	TContacto.media	TContacto.media
Ritmo.paso.desviacion	Ritmo.paso.desviacion	—
PreSwing..media	—	PreSwing..media
—	Velocidad.media	Velocidad.media
—	Aceleracion.media	Aceleracion.media
FootFlat..desviacion	FootFlat..desviacion	—
TStance..media	—	TStance..media
:	:	:

Tabla 2.4: Métodos de reducción de variables

Estos tres métodos de reducción de variables coinciden en aquellas con fondo verde en la tabla. Podemos asumir que estas son las más significativas para este problema.

Análisis del Componente Principal (PCA)

También se ha utilizado PCA (Principal Component Analysis) como método de reducción de dimensiones, para disminuir la cantidad de variables necesarias para entrenar el modelo, que sea menos costoso computacionalmente y más entendible. Para utilizarlo primero hace falta ver cuál es el número de componentes principales mínimo necesario para que los datos no pierdan significancia.

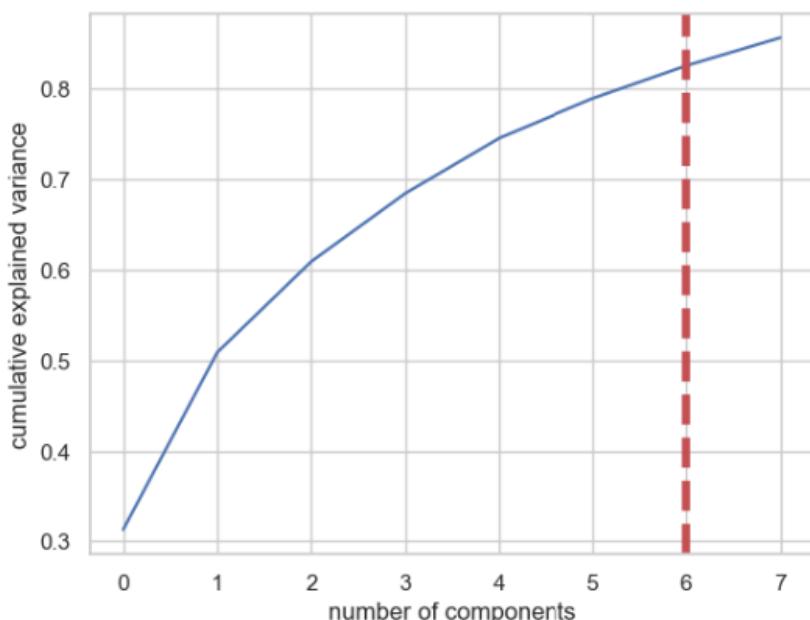


Figura 2.9: Varianza explicada acumulada

Aplico un PCA con siete componentes, porque suma un 85 % de varianza explicada, un valor aceptable y así reducimos tantas variables.

Como ya veremos más adelante, el PCA realiza un buen trabajo como simplificador de variables y podemos usarlo en el modelo final. El único problema es que este método no nos da información sobre la importancia de las variables, solo las transforma en nuevas totalmente numéricas e ininterpretables a simple vista. Para la realización de la conclusión y resultados, es un dato importante cuáles son las variables que más han influido para seguir trabajando en líneas futuras, por esta razón se priorizará otras formas de reducción de dimensionalidad.

La selección de variables hay que hacerla antes de hacer 'resampling' o estandarizar los datos porque si no, se ve influenciada por las nuevas muestras que tienen datos en sus variables muy parecidos a las muestras que ya existían entonces. Como el resampling se hace sobre una clase, le da más importancia a algunas variables que a otras. En teoría, si a esas variables se les da más importancia porque el método de resampling ha creado nuevas muestras viendo el patrón en esos datos de las variables es porque ha detectado que tienen más peso, sin embargo al no ser los métodos de 'resampling' perfectos, estamos influenciando a la selección de variables de manera artificial (por nuestra actuación) es decir, está seleccionando sobre datos que no son como naturalmente venían. En nuestro caso, si hacemos la selección de variables después del oversampling, obtenemos mejores valores de rendimiento, pero esa mejoría podría comprometer al modelo en un futuro en el que tenga que predecir realmente, porque aprenda demasiado como son los datos artificiales creados y no sepa como clasificar los nuevos (un caso de overfitting).

2.3. Flujo de trabajo

Tras toda la limpieza y preprocessamiento, encontrar y adaptar el modelo ha sido una tarea difícil. Como objetivo, se planteaba encontrar un modelo cuyas predicciones dieran buenos valores. La idea inicial ha sido seguir el siguiente diagrama:

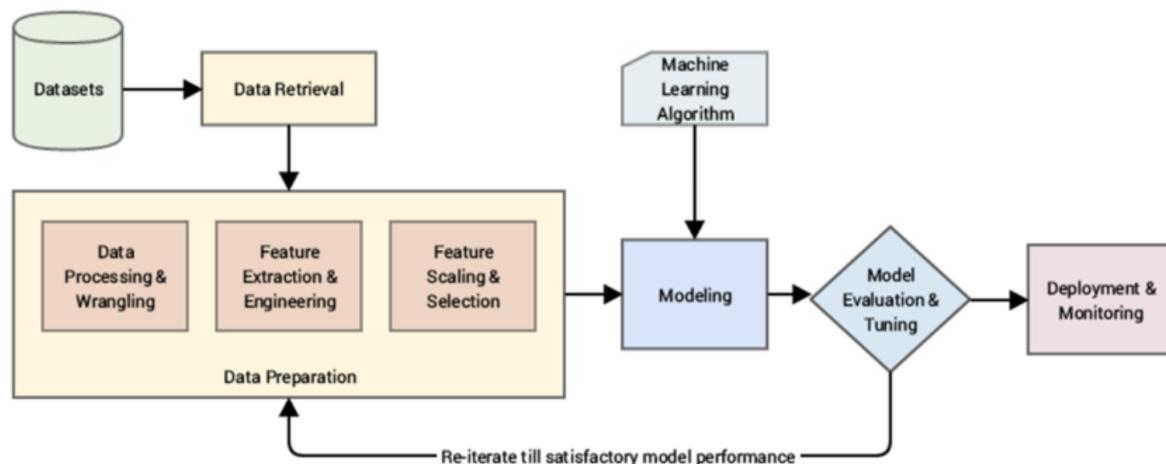


Figura 2.10: Flujo de trabajo

Sin embargo, por acuerdo con el podólogo de la clínica, en un principio el modelo estaba intencionado para que fuera capaz de predecir ocho diferentes patologías y una novena que fuera 'otro'.

Estas patologías eran:

Fascitis, Esguince, Pie plano, Pie cavo, Aquiles, Síndrome de Predislocación, Retropie Varo, Retropie Valgo, Otro

A los que les fueron asignados las etiquetas del 1 al 9 respectivamente (Ver figura 2.11).

Dataset con ocho patologías

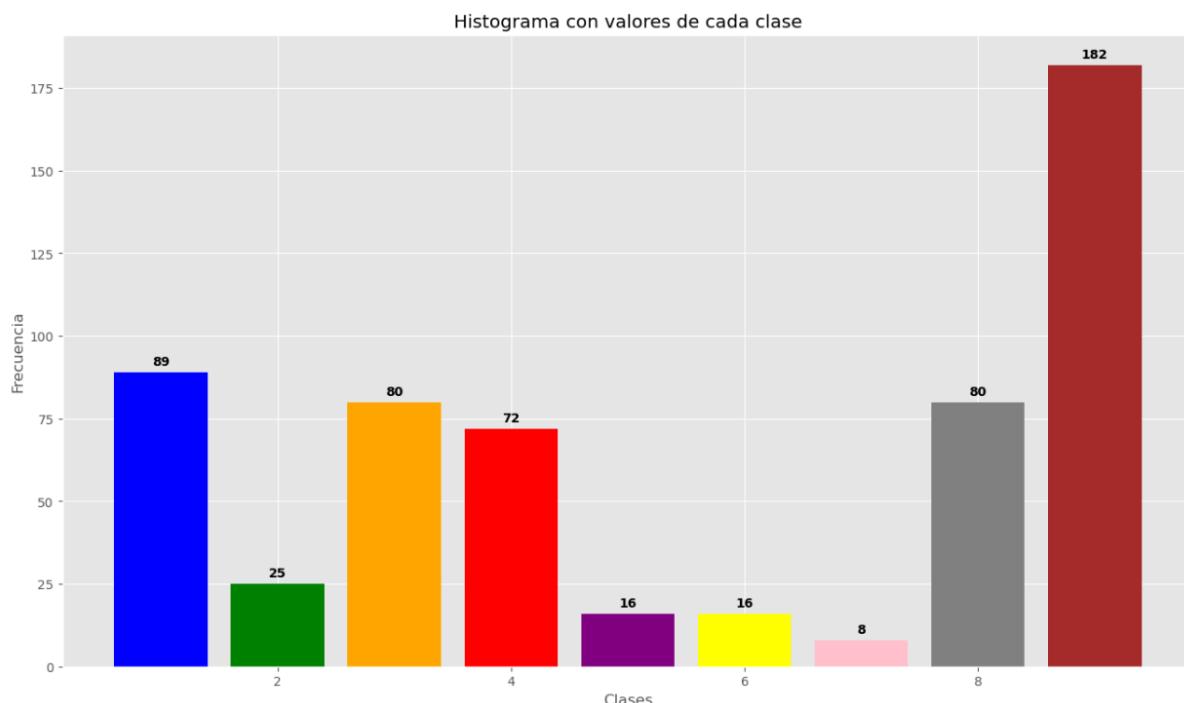


Figura 2.11: Histograma de las 9 clases

El primer problema aparece con el desbalanceo de clases, entrenar un modelo con esta diferencia de datos en cada etiqueta de salida lo hace muy complicado.

Primer intento de entrenamiento del modelo usando el paquete MLR3 de R y el algoritmo GLMNET.

```

1 #Normalizo los datos
2 normData <- normaliseData(data[-1], getLambda(data[-1], parallel = TRUE))
3 normData <- mutate(data[1], normData)
4
5 #creo un mlr3 task, que es el datafram pero en un formato mlr3
6 task_Opto = as_task_classif(normData, target = "Patology", id="pieses")
7 #Hago las particiones
8 splits = partition(task_Opto)
9 #Elijo el algoritmo que voy a usar (GLMNET, modelos lineales
  generalizados) y como quiero la predicción si en probabilidades ("prob")
  o en clases ("response")
10 learner_rpart = lrn("classif.cv_glmnet", predict_type="response")
11

```

```

12 #creo el metodo de resample que voy a usar (repeated cross_validation)
13 resampling = rsmp("repeated_cv", folds=5, repeats=2)
14 #Lo aplico
15 rr = resample(task = task_Opto, learner = learner_rpart, resampling =
   resampling)
16 #Veo los resultados estadisticos
17 rr$aggregate(msr("classif.acc"))
18
19 #para ver los resultados en cada iteracion
20 acc = rr$score(msr("classif.acc"))
21 acc[, .(iteration, classif.acc)]

```

Listing 2.1: MLR3 con GMLNET

La media de precisión obtenida es 0,305364; un resultado negativo.

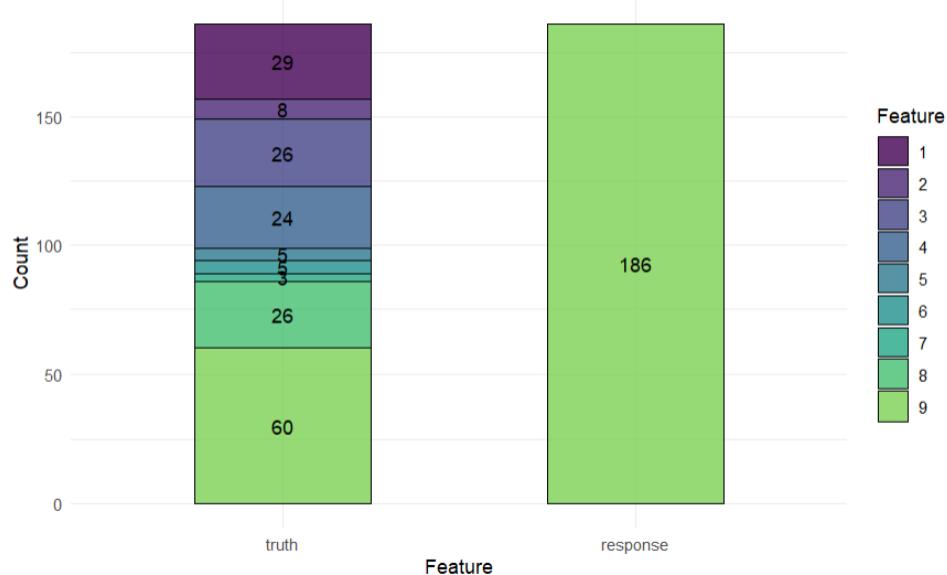


Figura 2.12: Resultado de predicción GMLNET

Podemos ver como la clase *Otro* invade totalmente todas las demás y el algoritmo predice todas como *Otro*. El principal problema parece ser el desbalanceo de clases.

Realizando undersampling a la clase 9 (*Otro*), conseguíamos que el dominio pasara a otra clase, apareciendo una predicción constante en la clase 1 ('Fascitis'). A pesar de hiperparametrizar (con Grid Search, Random Search) y usar varios entrenamientos con diferentes modelos (Support Vector Machine, Decision Trees, Neural Networks...) los resultados seguían igual de pobres.

```

1 measure=msr("classif.mauc_aunp") #La medida por la cual se va a
   hiperparametrizar
2 learner = lrn("classif.svm", #algoritmo y parametros
3   cost = tune(1e-1, 1e5),
4   gamma = tune(1e-1, 1),
5   predict_type = "prob",
6   type="C-classification",
7   kernel = "radial"
8   )
9 instance = ti( #pipeline

```

```
10 task=task_Opto ,  
11 learner=learner ,  
12 resampling=resampling ,  
13 measures=measure ,  
14 terminator = trm("none") #Uso trm none para hacer una busqueda de  
15 hiperparametros exhaustiva , que se para sola al ser un grid search  
16  
17 # Ahora elegimos un metodo que hara la seleccion de hiperparametros ( Grid Search )  
18 tuner = tnr("grid_search" , resolution = 5 , batch_size = 10) #  
19 Resolution es el numero de valores que se prueban por cada  
hiperparametro y batch_size es cuantas configuraciones se evaluan  
al mismo tiempo  
tuner$optimize(instance)
```

Listing 2.2: Hiperparametrización con SVC

Obteniendo aún resultados muy bajos: MAUC (Multiclass Area Under the Curve) = 0,5.

De aquí en adelante prosigo con Python en el entorno Jupyter por haber mayor documentación y mayor resolución de errores por la comunidad.

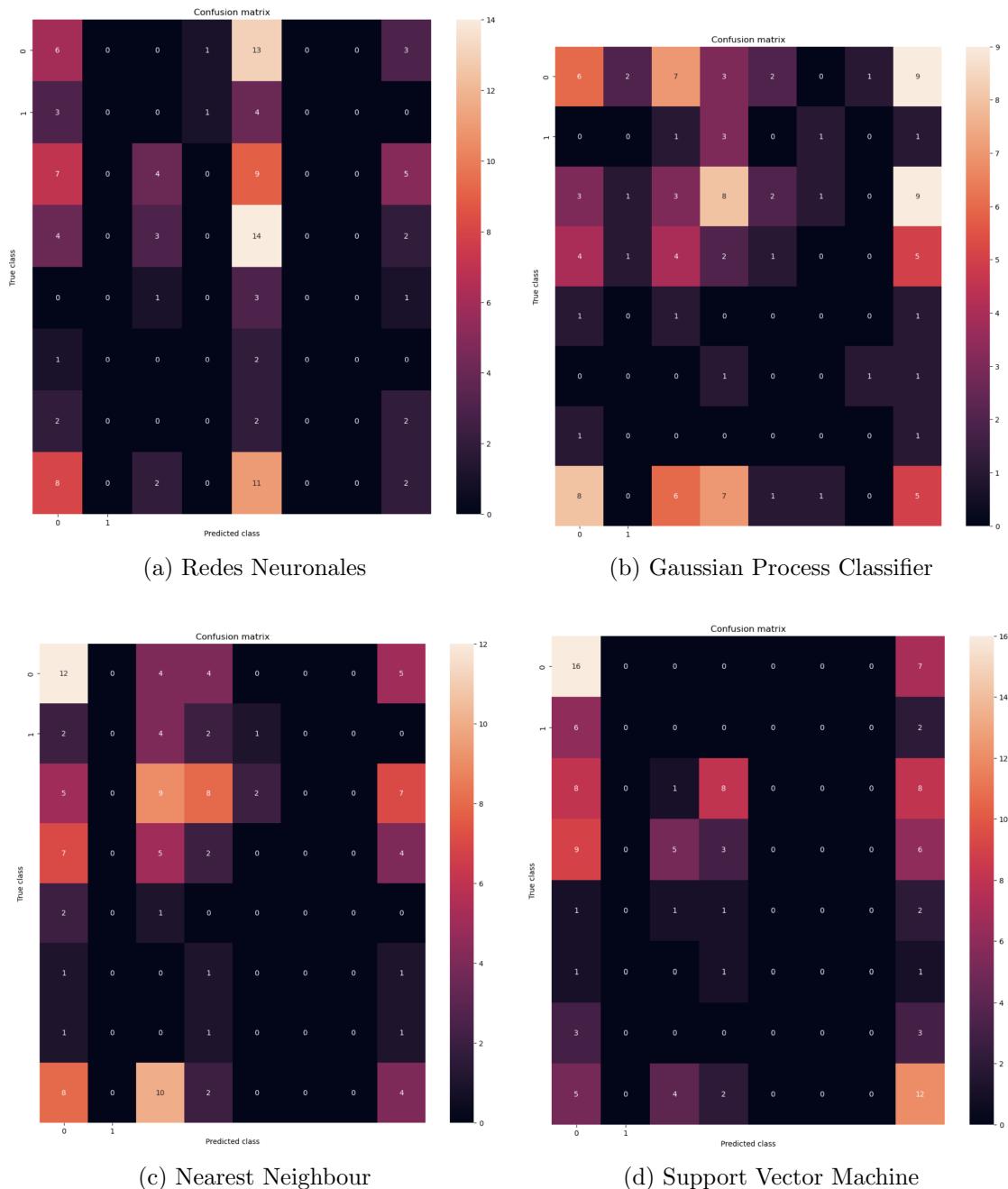


Figura 2.13: Matrices de confusión con ocho patologías

Para continuar eliminé la clase *Otro* puesto que tras pensarla, no tenía mucho sentido mantenerla ya que eran aquellos pacientes de cuyas historias médicas no se podían sacar ninguna de las patologías que se querían estudiar, es decir, pacientes con otras enfermedades, sanos, historias incomprensibles, medicaciones que toma o datos diferentes a diagnósticos. Esto lo único que hacía era confundir al modelo en sus predicciones.

En la figura 2.13 se ve el resultado de cuatro diferentes algoritmos, con PCA previamente aplicado, entrenados con las ocho clases restantes (Fascitis, Esguince, Pie plano, Pie cavo, Aquiles, Síndrome de Predislocación, Retropie Varo, Retropie Valgo).

En el eje vertical está la clase verdadera y en el eje horizontal la predicha, cuanto mayor sean los valores en la diagonal de las matrices, mejor será la predicción. Como podemos ver, todos estos resultados no son nada provechosos, pues la predicción apenas ha sido

certera. Obtuvimos valores de **precisión**, **recall**, **f1-score** de apenas 0,2.

El problema sigue siendo el alto número de clases y el que estén tan desbalanceadas, no es posible hacer buenas predicciones si en una clase tenemos 15 muestras y en otra tenemos 130 por ejemplo. La segunda acabará dominando sobre la primera y el modelo predecirá todos los registros como la segunda clase.

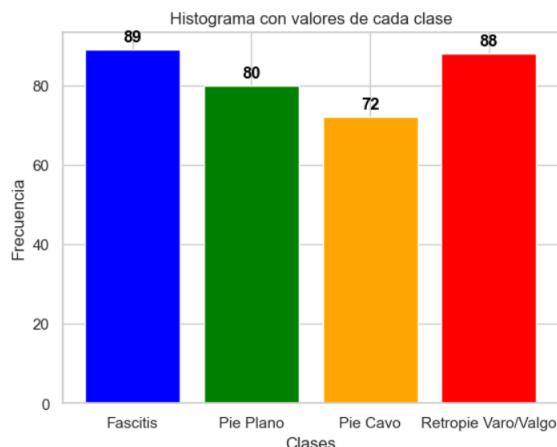
Como solución están las técnicas de resampling, que ofrecen varias opciones, poder hacer undersampling a aquellas clases con muchos números de muestras (pero no es muy favorable porque perdemos información útil para entrenar el modelo) o oversampling a esas clases que tienen muy pocos datos en nuestro caso *Esguince*, *Aquiles*, *Síndrome de Predislocación* y *Retropie Varo*.

Aún así la diferencia de datos entre las clases es tan grande que no tiene sentido aplicar oversampling. Estas técnicas siguen diferentes métodos para crear nuevas muestras, algunos de estos métodos pueden ser los mismos algoritmos que usamos para entrenar el modelo más adelante (SVC, Neural Networks...), por lo que también necesitan suficientes valores para, burdamente hablando, aprender el patrón de tales datos y crear nuevos sintéticos.

En nuestro caso tenemos clases con **ocho** valores (Retropie Varo) o con **dieciséis** (Aquiles y Síndrome de Predislocación), frente a las demás que rondan las **ochenta** muestras.

No queda otra opción que reducir clases por un problema de datos: no hay suficientes.

Dataset con cuatro patologías



Probamos a borrar las clases *Esguince*, *Aquiles*, *Síndrome de Predislocación* y uniflico las clases *Retropie Varo* y *Retropie Valgo*

Figura 2.14: Histograma con las clases

Ahora el dataset parece algo más fácil de entrenar y robusto. Se compone de cuatro clases con alrededor de 80 valores cada uno, por lo que no hay desbalanceo y podemos prescindir de hacer oversampling.

Vamos a ver como se comportan los datos ahora:

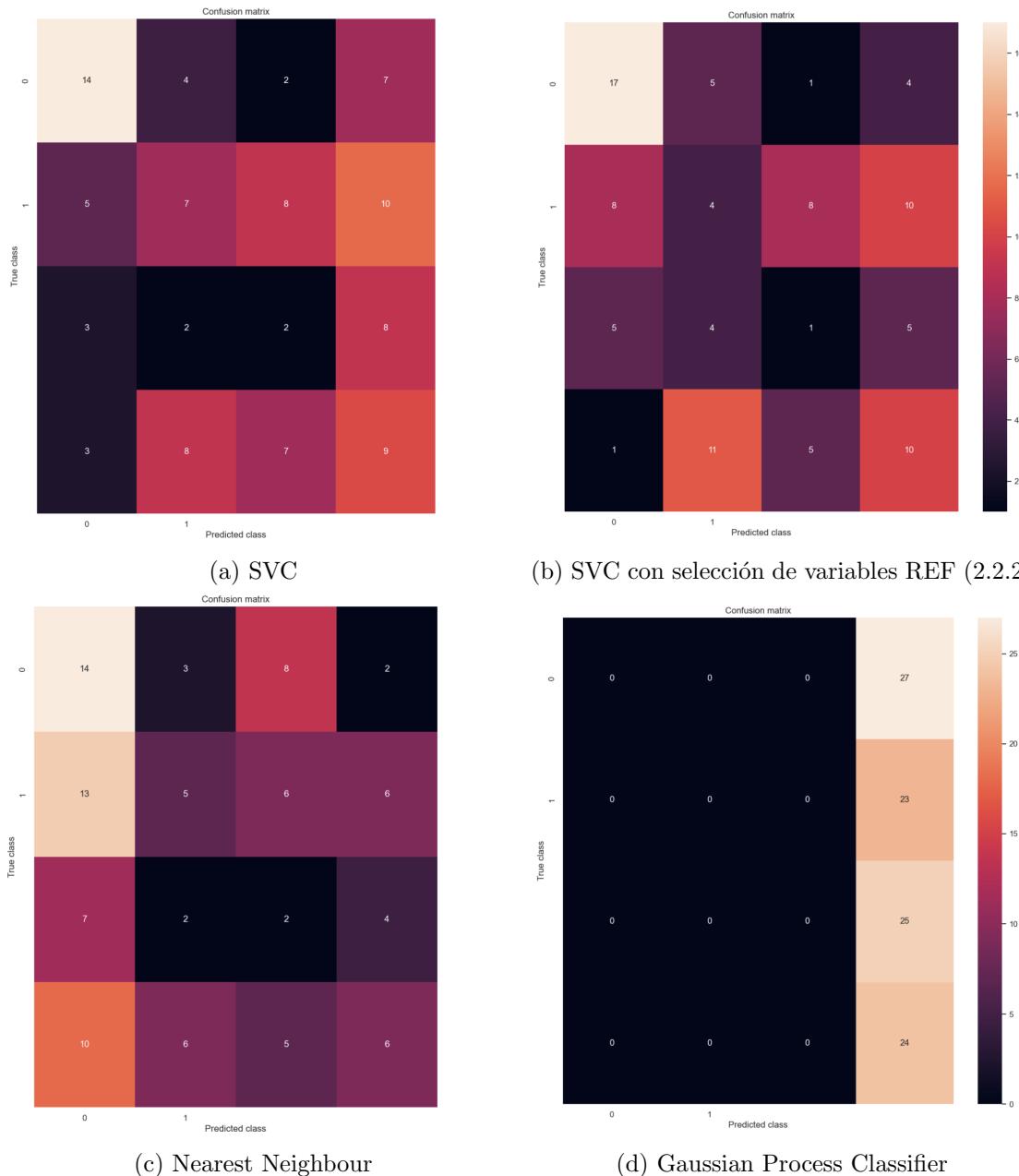


Figura 2.15: Matrices de confusión con cuatro patologías

Observamos de nuevo otras cuatro matrices de confusión, esta vez de dimensiones 4x4. En general, los resultados siguen sin ser útiles puesto que ninguna de ellas tienen muchos aciertos, cosa que podemos observar simplemente viendo el color de la diagonal de cada una. Si fuera de tonos más claros, como podemos ver en la leyenda, significaría un mayor valor, pero no es el caso. Analicémosla una a una:

- La primera matriz corresponde al algoritmo Support Vector Classification. Donde más ha acertado ha sido en la primera clase, (0 = 'Fascitis'). En la que ha predicho satisfactoriamente 14/27 casi la mitad. En cuanto a la segunda clase (1 = 'Pie Plano'), la predicción ha sido de muy mala calidad con un acierto de 7/30. La tercera clase ('Pie Cavo') 2/15 y la cuarta clase ('Retropie Vago') 9/27.

Solamente con ver que la clase *Fascitis* es la que mayor número de aciertos ha tenido con la mitad de su total, nos muestra los malos resultados de este primer modelo. En el que ni la **precisión** ni el **recall** superan el valor de 0,3.

- En cuanto a la segunda matriz, parece que la patología *Fascitis* ha sido un poco mejor predicha, pero algo insignificante, siguen siendo predicciones de muy mala calidad. En este caso también hemos aplicado el algoritmo Support Vector Classification, pero habiendo sido aplicado previamente un método de reducción de variables (REF). Este método ha seleccionado las 30 mejores variables del dataframe (ver 2.2.2), sin embargo podemos ver como no ha implicado nada en el modelo final, da unos resultados muy similares.
- La tercera matriz también ofrece unas predicciones pésimas, sin embargo difiere de las demás en que la predicción constante ha sido *Fascitis* en lugar de la última. Se ha usado el algoritmo 'Vecino más cercano' (Nearest Neighbour).
- Por último, el algoritmo del clasificador por procesos gaussianos (Gaussian Process Classifier) predice todas y cada una de las muestras como si fueran *Retropie Varo-/Valgo*, podríamos descartarlo sin problema puesto que no da ninguna información.

La cuarta clase, *Retropie Varo/Valgo*, parece haber dominado sobre las demás. Si nos fijamos, el modelo ha entendido la mayoría de las muestras como *Retropie Varo/Valgo*, excepto en el algoritmo de Nearest Neighbour, lo que me lleva a pensar que estas patologías no influyen nada en la marcha del paciente y son datos totalmente normales y muy generales por lo que el modelo no es capaz de distinguirlo. Por otro lado, en *Fascitis* suele haber un alto porcentaje de aciertos lo que pueden parecer buenas noticias, podríamos seguir investigando por esta etiqueta porque parece ser diferenciable por los modelos.

	precision	recall	f1-score	support	
1	0.55	0.63	0.59	27	
2	0.17	0.13	0.15	30	
3	0.07	0.07	0.07	15	
4	0.34	0.37	0.36	27	
accuracy			0.32	99	Aquí podemos ver los valores obtenidos en
macro avg	0.28	0.30	0.29	99	la primera matriz. La media (tanto 'macro
weighted avg	0.30	0.32	0.31	99	'avg' como 'weighted avg') de los valores se
					encuentra alrededor de 0.3.

Figura 2.16: Valores de SVC

Siendo estos los mejores datos obtenidos de las cuatro matrices, podemos asumir que necesitamos reducir las clases del problema aún más. Por muy balanceadas que estén, los modelos confunden todas entre ellas y no hay manera de encontrar ningún algoritmo (aquí se han mostrado cuatro pero se han probado además, Redes Neuronales, Decision Tree, Random Forest, AdaBoost, Naive Bayes, QDA, Logistic Regression y Linear Regression). El siguiente paso será eliminar la clase que hemos observado que añade más ruido, *Retropie Varo/Valgo*, además para añadirle peso a tal decisión, no es una patología grave como lo puede ser una fascitis o padecer de síndrome de predislocación por lo que influye menos en la vida de una persona. Es más, esta etiqueta fue añadida por observación de alta

frecuencia en los historiales médicos de los pacientes y curiosidad de estudio, no por indicación de los profesionales podólogos.

Dataset con tres patologías

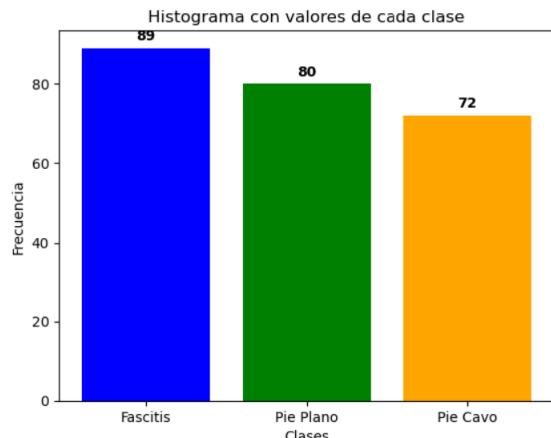


Figura 2.17: Histograma con tres patologías

Igual que antes, el dataset está balanceado, las clases tienen un número de datos parecido por lo que no nos tenemos que molestar en aplicar resampling.

Comparemos los mismo algoritmos que antes pero con este nuevo dataset (Ver figura 2.18):

- En la primera matriz, en la que hemos aplicado el algoritmo Gaussian Process Classifier, todos los valores son predichos como si fuesen Pie Cavo por lo que los valores de accuracy y recall son de 0.33, que es lo mismo que si hubiésemos clasificado todos los datos aleatoriamente (ya que tenemos tres clases, $\frac{1}{n^o \text{ de clases}}$).
 - En la segunda matriz de confusión hemos aplicado el algoritmo de C-Support Vector Classification. No hay un dominio de una clase sobre las demás como pasaba en la anterior pero de igual manera siguen siendo resultados muy bajos con valores de precision de 0.39, accuracy de 0.4 y recall de 0.41, casi bordeando la aleatoriedad.
 - En la tercera se ha utilizado el algoritmo de Nearest Neighbour, que no muestra tampoco una buena clasificación. Sólo clasifica las dos primeras clases olvidándose de la tercera a la que le asigna solamente 12 valores cuando la cantidad real de valores de Pie Cavo es 27, además de esos 12 únicamente ha acertado 3.
- Para las otras dos clases el índice de aciertos tampoco es alto; en *Fascitis* ha predicho 33 y solamente ha acertado 10 de esos. Para *Pie Plano* ha predicho 28 y de esas, solo 8 son ciertas. Estamos ante peores resultados que los dos algoritmos anteriores, es decir, peor que la aleatoriedad.
- En cuanto a la última matriz, se ha usado el algoritmo de Redes Neuronales, sin embargo, al principio, ejecutándolo con los parámetros por defecto de las funciones, ocurría que la clase *Fascitis* dominaba sobre las demás, por tal razón he buscado los parámetros más óptimos usando técnicas de hiperparametrización.

```
1 hidden_layer_sizes=[(1,3),(1,4),(1,5),(1,10),(2,3),(2,5)
,(2,10),(2,20),(2,50),(3,3),(3,5),(3,10),(3,15),(3,20)
,(4,5),(4,7),(4,10),(4,20),(4,25),(5,5),(5,7),(5,10)]
```

```
1   ,(6,10),(7,5),(7,10),(8,5),(9,10),(9,20),(10,5),(10,10)
2   ,(10,50)
3   ]
4 activation =["tanh","logistic","relu"]
5 solver = ["lbfgs","sgd","adam"]
6 alpha = [0.00001,0.0001,0.001, 0.005, 0.01, 0.05, 0.1, 1, 10]
7 learning_rate = ["constant","adaptive"]
8 learning_rate_init = [0.0001, 0.001, 0.01, 0.1, 1, 10, 20]
9 max_iter = [1000,2000]
10 early_stopping=[True]
11
12 nn = MLPClassifier()
13
14 param_dist = {'hidden_layer_sizes': hidden_layer_sizes,
15                 'activation': activation,
16                 'solver': solver,
17                 'alpha': alpha,
18                 'learning_rate': learning_rate,
19                 'learning_rate_init': learning_rate_init,
20                 'max_iter': max_iter,
21                 'early_stopping': early_stopping }
22
23 # Se ejecutara 100 veces por el num de cross validations
24 rs = RandomizedSearchCV(nn,
25                         param_dist,
26                         n_iter = 100,
27                         cv = 3,
28                         verbose = 1,
29                         n_jobs=-1,
30                         random_state=0)
```

Primero, de las líneas 1-9 declaro las variables que serán los parámetros con todas las posibilidades que quiero estudiar en este algoritmo.

En la línea 11 se crea el clasificador, en este caso, red neuronal.

Línea 13-20 asigno las variables creadas a los parámetros.

Creo el hiperparametrizador que en este caso será Random Search Cross Validation y le doy los parámetros mostrados.

Los mejores parámetros obtenidos han sido:

```
solver : lbfgs,    max_iter : 1000,
learning_rate_init : 0,001,
learning_rate : 'adaptive',
hidden_layer_sizes : (3, 3),
early_stopping : True,
alpha : 0,05,
activation : 'tanh'
```

Sin embargo, a pesar de haber buscado los mejores parámetros la clasificación sigue sin ser de buena calidad, de la primera clase ha predicho 35 valores cuando en realidad solo hay 27 y de esos 35 sólo ha acertado 15. En la segunda clase ha

predicho 27 de los cuales solamente ha acertado 8 y realmente de la segunda clase hay 20 valores. En cuanto a la tercera, ha fallado casi plenamente, ha predicho 11 de los que ha acertado 3 y verdaderamente hay 26 muestras de la tercera clase.

Tener el modelo con tres etiquetas no ha solucionado nada, ninguno de los cuatro modelos ha conseguido predicciones útiles. Tampoco cambian los resultados si aplicamos hiperparametrización, métodos de selección de variables, métodos de resampling...

De igual manera que pasaba cuando el problema tenía cuatro clases, se han puesto en práctica más algoritmos cuyos resultados han sido iguales o peores que los aquí mostrados.

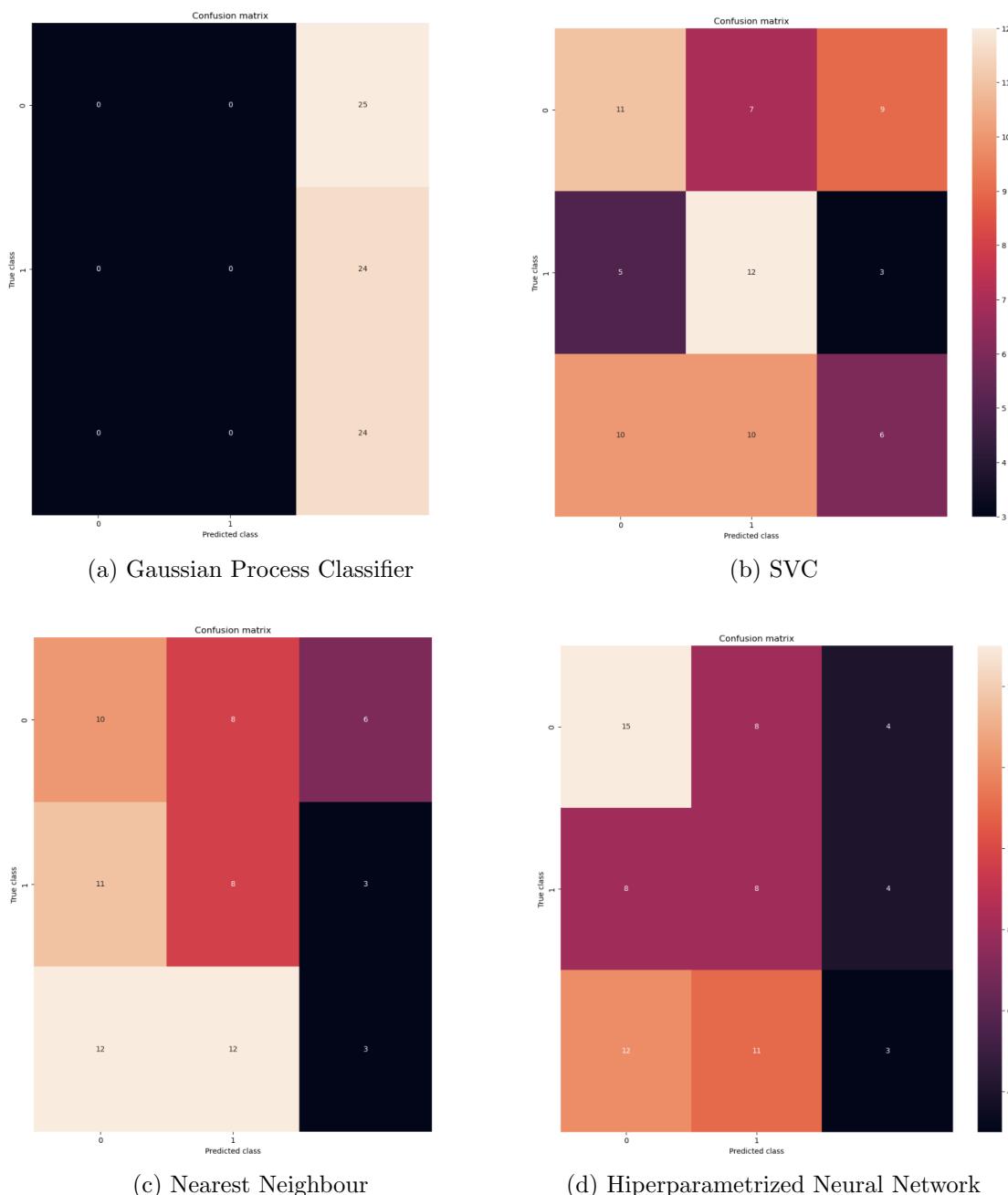


Figura 2.18: Matrices de confusión con tres patologías

Analizando las tres clases, podemos ver como la tercera ha sido la más problemática, los modelos son incapaces de aprender las muestras de *Pie Cavo*. En cuanto a *Fascitis* y *Pie Plano*, no tenemos mucha información. Hasta ahora los modelos los han clasificado

pero no con mucho acierto. Por el momento parece que la clase candidata a eliminar del problema será *Pie Cavo*, sin embargo, tener un modelo que prediga si un paciente tiene fascitis o pie cavo no es el tipo de problema que estábamos buscando solucionar, ambas patologías no tienen nada que ver y no es algo eficiente que prediga una cosa u otra. Más acorde a los intereses de un principio es hacer un modelo predictivo que sepa indicar si el nuevo paciente que ha realizado el test sufre de fascitis o por otro lado que indique que puede ser otra patología o ninguna. Es decir, volaremos a añadir una clase *otro*.

Dataset con dos patologías

Al final, se ha ido reduciendo el problema hasta acabar en uno con sólo dos etiquetas, es decir, un modelo binario en el que hay dos opciones para predecir, en este caso si el paciente padece de fascitis o si en cambio parece ser que no lo es y puede ser otra cosa. A aquellas muestras del dataset cuya patología sea fascitis, se le asignará el valor 1 y a todos los demás que hemos ido quitando serán agrupados en el grupo *otro* con el valor 0 asignado.



Figura 2.19: Histograma de Fascitis y Otro

Para componer la clase *Otro* se han fusionado las demás clases que había desde un principio a excepción de *Retropie Valgo/Varo* y la clase original *Otro*.

$$25 \text{ (Esguince)} + 80 \text{ (Pie Plano)} + 72 \text{ (Pie Cavo)} + 16 \text{ (Aquiles)} + 16 \text{ (S. Predislocación)} = 209$$

Probablemente el lector haya apreciado que de nuevo volvemos al problema del desbalance de clases. Tenemos una clase con más del doble de muestras que la otra. Tendremos que aplicar *resampling* para disminuir la diferencia de datos en cada una. Se ha decidido aplicar un *oversampling* con SMOTE (Synthetic Minority Over-sampling Technique) sobre la clase minoritaria y después un *undersampling* con 'Random Under Sampler' sobre la clase mayoritaria.

```

1 #Smote Oversampling
2 oversample = SMOTE(sampling_strategy=0.7)
3 X_resampled, y_resampled = oversample.fit_resample(x, y)
4
5 # Random Under sampler
6 undersample = RandomUnderSampler(sampling_strategy=0.85)
7 X_resampled, y_resampled = undersample.fit_resample(X_resampled,
8 y_resampled)

```

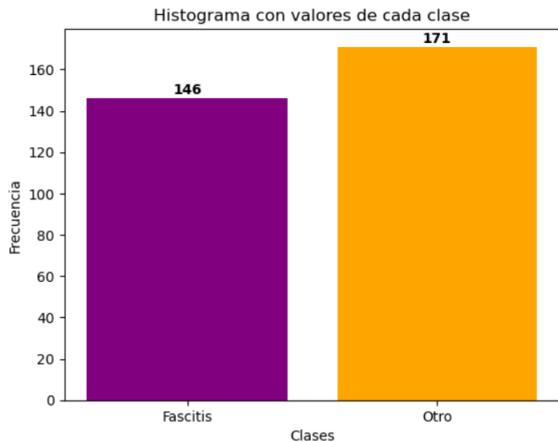


Figura 2.20: Histograma después de *Resampling*

Al ser estos los datos con los que trabajaremos en el modelo final, se mostrará cada uno de los métodos de reducción de variables así como los demás algoritmos con sus respectivas comparaciones hasta llegar al mejor modelo predictivo. Que será explicado en el capítulo **Resultados** y utilizado en la aplicación web creada y desarrollada en el apartado **RShiny app**.

CAPÍTULO 3

Resultados

En este capítulo de la memoria se mostrarán los mejores resultados obtenidos, así como sus características y una discusión sobre los mismos con comparaciones entre los diferentes algoritmos y sus respectivos valores de rendimiento.

Como se ha concluido en el anterior capítulo, el modelo final será binario, con solamente dos clases, *Fascitis* y *Otro*. De esta manera, ya empezamos a encontrarnos con resultados más asequibles y con los que se pueda trabajar. Pero para ello, primero tenemos que pasar por varios algoritmos y sus diferentes parámetros, así como la elección de un método u otro de reducción de variables, estandarización...

El objetivo, será encontrar un conjunto de valores de rendimiento alto. Nos fijaremos en: precisión (accuracy), precisión positiva (precision), sensibilidad (recall), F1-score. Usando gráficas para representarlas e interpretarlas más fácilmente.

Cuanto mayor sean estos valores, mejor será el modelo que hayamos encontrado, teniendo en cuenta la posibilidad de sobreajustamiento (ínfima, pero posible) y los casos de dominio de una clase sobre otra como hemos observado en los anteriores modelos multiclas.

Todos los datos han sido previamente estandarizados, puesto que era necesario por varias razones. La primera, el coste computacional; ejecutar las funciones de entrenamiento cuando los datos no estaban estandarizados se prolongaba varios segundos más que cuando sí estaban. Y, aunque en algunos algoritmos no mostraba mucho cambio, en otros la diferencia era de hasta 0.1 en los valores de rendimiento.

Como se ha mostrado antes, en varios modelos como por ejemplo *Redes Neuronales*, se ha hiperparametrizado si era necesario. En este último los parámetros necesitaban ser cambiados para encontrar unos mejores porque los que usaba la función por defecto daban resultados insatisfactorios.

En cuanto al método de selección de variables, teníamos cuatro métodos para elegir (Ver 2.2.2), habiendo sido probado todos, el que mejor valores ha resultado dar ha sido la *Eliminación de Variables Recursiva (RFE)* y ha sido la elegida para entrenar los diez algoritmos que se muestran más abajo.

Para todos los modelos, se ha utilizado una validación cruzada de cinco particiones y los valores de rendimiento mostrados en la 3.1 son la media de cada ejecución. Además, para aquellos modelos en los que ha sido necesario hiperparametrizar, se han buscado con

CAPÍTULO 3. RESULTADOS

el set de validación y luego comprobados con el set de testeo con el objetivo de no ofrecer una estimación optimista.

Es importante destacar la necesidad del uso de *Pipelines* para cada algoritmo cuando queremos conseguir los valores de rendimiento con la función *cross-validate*. Si no se hace uso de uno, no realizamos correctamente la predicción puesto que el preprocesamiento (estandarización, resampling...) se aplica también al set de validación (en caso de hiperparametrizar) y al set de test.

	SVC	NNeuron	NNeigh	GPC	Decision Tree	Random Forest	AdaBoost	Naive Bayes	QDA	Logistic Regression
Accuracy	0.65	0.67	0.6	0.64	0.58	0.62	0.65	0.5	0.67	0.66
Precision	0.62	0.57	0.56	0.6	0.54	0.54	0.61	0.51	0.57	0.61
Recall	0.64	0.61	0.57	0.6	0.55	0.54	0.61	0.52	0.52	0.62
F1 Score	0.62	0.58	0.56	0.59	0.53	0.54	0.61	0.47	0.51	0.61
ROC_AUC	0.66	0.62	0.64	0.65	0.55	0.56	0.65	0.58	0.54	0.67

Tabla 3.1: Tabla de resultados

Observemos los resultados, no resulta fácil decidir cuál ofrece mejores resultados y cuales peores. Nos hemos encontrado con la mala suerte de que los valores de cada modelo predictivo se asemejan, es decir, suelen andar todos en la misma centésima y no varían mucho entre ellos y en valores muy bajos. Algunos se pueden descartar a primera vista, sin embargo habrá que decidir cuál usar (sin haber mucha diferencia entre ellos).

Por ahora descartaremos, *Random Forest*, *Decision Tree* *QDA*, *Naive Bayes*, *Nearest Neighbors*, y *Redes Neuronales*.

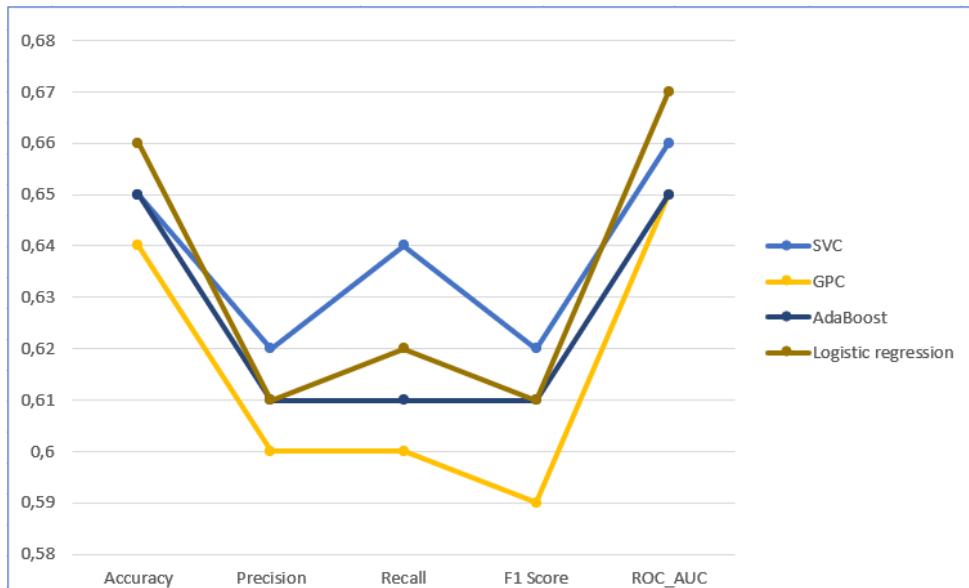


Figura 3.1: Valores de rendimiento de los modelos

Por encima de todos parece destacar el algoritmo *Support Vector Classification*, que ofrece una media de entre todos los resultados mayor. Después los modelos van empeorando aunque *Logistic Regression* o *AdaBoost* podrían servir como alternativa al primero. El algoritmo *Gaussian Process Classifier* a pesar de ser superior a todos los demás, nos muestra peores resultados que los anteriores tres.

Visto entonces, escojamos el algoritmo *SVC* y estudiemos cada uno de sus valores de rendimiento y su matriz de confusión.

		Predicted	
		Positive	Negative
Positivo Real	True Positive (36)	False Negative (23)	
	False Positive (11)	True Negative (20)	

Tabla 3.2: Support Vector Classifier Confusion Matrix

Positive = 0 = *Otro*Negative = 1 = *Fascitis*

Esta matriz de confusión corresponde al modelo hiperparametrizado, pero no el que se ha implementado en la aplicación, para eso se ha entrenado un modelo con todos los datos. La tabla mostrada anteriormente es el resultado de validación cruzada de varios modelos. Podemos observar que los verdaderos positivos y los verdaderos negativos son las celdas con mayor valor, lo cual nos indica un buen resultado en las predicciones, aunque los falsos positivos y los verdaderos negativos son casi iguales, lo que muestra que el modelo predictivo no es de muy buena calidad.

La cantidad real total de pacientes positivos ('Otro') es de 60, y de pacientes negativos ('Fascitis') es de 30. De los primeros 60, 42 han sido predichos de manera satisfactoria y de los 30, 17. Con estos datos (Verdaderos Positivos, Verdaderos Negativos, Falsos Positivos y Falsos Negativos) podemos calcular todos los valores de rendimiento anteriormente mostrados en la tabla 3.1.

$$\text{Accuracy} = \frac{\text{Número de Predicciones Correctas}}{\text{Número Total de Predicciones}} = \frac{TP + TN}{TP + TN + FP + FN}$$

Si hacemos el cálculo, obtenemos 0.62. El accuracy (precisión) indica el índice de aciertos que ha tenido el modelo, en este caso, ha sido del 62 %. Sin embargo, no es bueno fijarse solamente en la precisión puesto que no muestra si se ha acertado más en una clase que otra, puede que se haya fallado mucho en una clase pero al haber pocos valores no influye en el resultado de este dato.

$$\text{Precision} = \frac{\text{Predicciones Correctas Positivas}}{\text{Predicciones totales Positivas}} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Positives (FP)}}$$

Por otro lado está la precisión positiva (Precision) que muestra el índice de positivos que se han predicho correctamente frente a todos los positivos predichos. Este valor de rendimiento ya sí especifica una clase concreta (*Otro*). 0.77 indica un buen porcentaje de aciertos positivos.

$$\text{Recall} = \frac{\text{Predicciones Correctas Positivas}}{\text{Positivos Reales}} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Negatives (FN)}}$$

Recall es el índice de verdaderos positivos frente a los positivos reales; cuántos positivos de todos los verdaderos que hay han podido ser bien clasificados. 0.61 es el resultado de esta fórmula.

CAPÍTULO 3. RESULTADOS

$$\text{F1 Score} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

Por último tenemos la medida F1, que es mucho más sólida y mezcla varios valores de rendimiento dando una información más general. En este caso es de 0.68.

Hasta ahora, todas las medidas que hemos calculado han sido sobre los positivos, es decir, sobre la clase *Otro*. Sin embargo, deberíamos tener en cuenta también los negativos (*Fascitis*), a pesar de que a simple vista podamos ver en la matriz de confusión que han sido buenas predicciones también. Medidas como *True Negative Rate* (TNR, Specificity, o especificidad) o *Negative Recall* (Negative Recall, True Negative Recall), son útiles para este objetivo.

$$\text{Specificity} = \frac{\text{Predicciones Correctas Negativas}}{\text{Negativos Reales}} = \frac{\text{True Negatives (TN)}}{\text{True Negatives (TN)} + \text{False Positives (FP)}}$$

$$\text{Negative Recall} = \frac{\text{Predicciones Correctas Negativas}}{\text{Predicciones totales Negativas}} = \frac{\text{True Negatives (TN)}}{\text{True Negatives (TN)} + \text{False Negatives (FN)}}$$

Cuyos valores son 0.65 y 0.47, respectivamente. Juntando estos valores con los equivalentes en los positivos, se crea la precisión o recall medio.

Con estos dos últimos valores podemos ver que los resultados no llegan a ser suficientes. El modelo es capaz de predecir bien la clase 0 (*Otro*) pero no la clase 1 (*Fascitis*). A pesar de que la precisión media sea de 0.6, no podemos fiarnos plenamente de este modelo puesto que no es capaz de predecir mucho mejor que una moneda lanzada si un paciente tiene *Fascitis*.

CAPÍTULO 4

RShiny app

Con el objetivo de utilizar el modelo obtenido con los datos de la clínica, se ha creado una aplicación Web que permita al personal profesional de la clínica facilitar o agilizar el proceso de diagnóstico. Esta aplicación consiste en la introducción de los test de un nuevo paciente, exportados desde *Optogait* como un archivo XML. Al subir el documento, automáticamente se cargará en la página y tras añadir varios datos manualmente, permitirá usar el botón de predecir. Una vez clicado el botón, aparecerán las probabilidades, mostradas en un gráfico, de tener *Fascitis* u *Otro*.

4.1. Desarrollo de la aplicación Web

La aplicación ha sido desarrollada en RStudio con el lenguaje de programación R, en concreto con el paquete Shiny.

Shiny ofrece un framework para crear aplicaciones web sencillo e intuitivo que además posibilita la implementación de código en Python con la librería *reticulate*. Estiliza todo automáticamente con métodos implementados por defecto que usan la librería Bootstrap de CSS.

También se han implementado líneas de *HTML* y *JavaScript* para algunas ocasiones en que la personalización requería algo más específico. De igual, manera se ha asignado un archivo 'main.css' para cambiar el aspecto de algunos elementos de la aplicación RShiny puesto que la visualización gráfica que ofrecía por defecto no era la necesitada.

Toda aplicación de RShiny se compone de un archivo 'app.r' que se divide en un archivo 'server.r' (el servidor de la web) y 'ui.r' (y la interfaz gráfica que se muestra). Para lanzar la aplicación basta con ejecutar la función `runapp('nombre_del_archivo_app.r')`.

4.1.1. Interfaz gráfica (UI.R)

En RShiny se pueden elegir varios tipos de páginas para la interfaz (Fluidpage, navbarPage, fixedPage, dashBoardPage...). Para esta aplicación hemos escogido dashboardPage, que permite la creación de paneles laterales interactivos y un área de contenido principal. Es una opción popular para crear aplicaciones que manejan datos.

En el encabezado se ha añadido el título con el tamaño elegido:

```

1 ## Cabecera de la pagina ##
2 dashboardHeader(
3   title = "OptoPrediction",
4   titleWidth = 300
5 ),
6
7 ## Barra lateral ##
8
9 dashboardSidebar(
10
11   fileInput("xml_Optogait_file", label = "Selecciona un archivo xml:",
12   accept = c(".xml"))
13 )

```

En la barra lateral sólo se ha incluido el elemento para escanear el archivo.

En el cuerpo de la aplicación se ha incluido un link al archivo css, y dos paneles condicionales que se muestran cuando ocurren dos eventos. El primero, se muestra cuando se escanea un archivo y el segundo cuando se selecciona el botón de 'Predict' en el que inmediatamente después se muestra la gráfica con los resultados de la predicción.

```

1 dashboardBody(
2   tags$head(
3     tags$link(rel="stylesheet", type = "text/css", href="main.css") #No
4       hace falta poner la dirección absoluta, solo con poner la relativa,
5       el programa sabe que está en www
6     ),
7   useShinyjs(),
8   #Creo un panel condicional que se muestra solo si se ha subido un
9   #archivo valido
10  conditionalPanel(
11    condition="output$fileUploaded == true",
12
13    h1("Dataframe del nuevo paciente"), # Titular del panel
14
15    div(DTOutput("tabla"), style = "margin-bottom: 20px;"), # Tabla que
16      muestra el df
17
18    #Boton que se habilita cuando se introduce el Peso, Altura y N de
19    #Pie
20    div(
21      fluidRow(
22        column(width=2,actionButton("newPredictionButton","Predict",
23        disabled=TRUE)),
24        column(width=6,div(id="noPossiblePrediction",textOutput("noPossiblePrediction")))
25      )
26    ),
27    # Grafico con los resultados que se habilitar cuando se clica el
28    #boton de 'Predict'
29    conditionalPanel(
30      condition="input.newPredictionButton != 0",
31      div(plotOutput("prediction")) %>% withSpinner(color="#0dc5c1"),
32      style="height: 50px;")
33    ),    )

```

4.2. Servidor (Server.R)

En el servidor se irá manejando la información en línea conforme el usuario vaya usando las funcionalidades de la aplicación. También permite la edición de la interfaz gráfica al momento mostrando nuevos elementos dependiendo de unas acciones u otras.

Mediante la librería reticulate añadimos los dos scripts de python (el que procesa el archivo xml y el archivo que realiza la predicción con el modelo guardado),

```

1 # Cargamos los scripts de python (el que procesa el archivo subido y el
2   que predice)
3 reticulate::source_python('python_funcs/xml_to_df.py')
4 reticulate::source_python('python_funcs/predict.py')
5
6 #Reactive value to get the Data from the file uploaded
7 getData <- reactiveValues(df=NULL)
8 output$noPossiblePrediction <- renderText("Introduce el Peso, Altura y
9   N de Pie")
10
11 observeEvent(input$xml_Optogait_file, {
12   if(is.null(input$xml_Optogait_file))
13     getData$df <- NULL
14
15   else{
16
17     # Obtener la ruta del archivo seleccionado
18     file_path <- input$xml_Optogait_file$datapath
19
20     withProgress(message="Cargando", value=100,{
21       # Cargar los datos de entrada a Python para convertirlos a un
22       # formato adecuado (Funcion obtenida del script anterior)
23       getData$df <- xml_to_df(file_path)
24     })
25   }
26 })

```

Con 'ObserveEvent', se espera a que se suba el archivo XML, en el momento que se hace, si el archivo es correcto (solo se permite subir XML), obtiene el dataframe correspondiente con la función de python importada con reticulate previamente.

Después se van editando el dataframe que ha sido mostrado en una tabla interactiva, en el que se puede cambiar cualquier dato y añadir las tres variables peso, altura y numero de pie.

4.2.1. Funcionamiento de la aplicación

La idea desde un principio era crear una aplicación que pudiese leer el archivo que crea el programa *Optogait* cuando exportas la prueba de marcha realizada a un paciente. Este archivo está en formato .XML pero tiene un formato difícil de interpretar y de aprovechar con otros programas (Véase figura 4.1).

Cada prueba consta de una medición de la marcha durante alrededor de treinta segundos (cada prueba puede variar). Cada paso es medido por el dispositivo Optogait y es representado en la hoja de cálculo en cada fila.

U	V	W	X	Y	Z	AA	AB	AC	AD	AE
Test	Fecha	Hora	#	L/R	Externo	TReaccion	TEspera	TVuelo	TContacto	TPaso
Exploración M 17/03/2023	17/03/2023	13:17:42	Iniciar en el eSTART						0,753	0,523
Exploración M 17/03/2023	17/03/2023	13:17:42		1 R					0,739	0,529
Exploración M 17/03/2023	17/03/2023	13:17:42		2 L					0,773	0,534
Exploración M 17/03/2023	17/03/2023	13:17:42		3 R					0,742	0,561
Exploración M 17/03/2023	17/03/2023	13:17:42		4 L					0,742	0,55
Exploración M 17/03/2023	17/03/2023	13:17:42		5 R					0,737	0,516
Exploración M 17/03/2023	17/03/2023	13:17:42		6 L					0,744	0,551
Exploración M 17/03/2023	17/03/2023	13:17:42		7 R					0,733	0,529
Exploración M 17/03/2023	17/03/2023	13:17:42		8 L					0,753	0,534
Exploración M 17/03/2023	17/03/2023	13:17:42		9 R					0,73	0,561
Exploración M 17/03/2023	17/03/2023	13:17:42		10 L					0,741	0,529
Exploración M 17/03/2023	17/03/2023	13:17:42		11 R					0,725	0,533
Exploración M 17/03/2023	17/03/2023	13:17:42		12 L					0,744	0,539
Exploración M 17/03/2023	17/03/2023	13:17:42		13 R					0,723	0,538
Exploración M 17/03/2023	17/03/2023	13:17:42		14 L						

Figura 4.1: Archivo XML Optogait

Los pasos se van ordenando en filas sucesivas, que comienzan siempre con el nombre del paciente y se van rellenando todas las variables. Además de las visibles en la imagen 4.1, aparecen las estudiadas en los modelos (LoadResponse, PreSwing, Zancada...) y bastantes más que están vacías sin información (Fecha de Nacimiento, Empleo, ID...) que son innecesarias.

Mediante técnicas de minería de textos, procesamos todo el archivo Excel reduciendo cada dato de cada paso en la media y la desviación, eliminando las columnas que no necesitamos (tanto vacías como con información innecesaria). Este primer procesamiento ha sido realizado con Python y las librerías *{unidecode, re, pandas y StringIO}*, después ha sido añadido con *reticulate* a la carpeta de la aplicación.

En este script de Python se ha definido una función que recibe la ruta de un archivo, después es cargado en una variable y comienza a procesarlo. Escanea cada línea, elimina todas las columnas innecesarias, quita espacios o caracteres corruptos y finalmente crea un dataframe con los datos sobre los que se añaden unas cuántas líneas de código más para devolver una tabla casi preparada para usarla como dato input en una predicción para el modelo, solo le hace falta que el profesional añada manualmente el peso, la altura y el nº de pie del paciente desde la aplicación web.

```

1 def xml_to_df(file_path):
2     #encoding="utf-8-sig" because the file is in UTF-8 with BOM
3     with open(file_path, 'r', encoding="utf-8-sig") as fp:
4         # read all text without written accents
5         text = unidecode(fp.read())
6
7         text=re.sub(r'</?Cell>[\n]', '', text)
8         #Splitting into lines
9         lines= text.splitlines()
10
11     # Removing first 12 lines
12     lines = [line for i, line in enumerate(lines, start=1) if i
13 not in range(0,13)]
14     #Creating the final variable with the final text needed
15     csv = ""
16     firstrow=True #So there is not a new line in the first row
17     columnnumber=1
18     for line in lines:

```

```

18 #new column
19     if not re.search("<Row>",line):
20         firstrow=False
21         #if there is a index in a column, it means the
22         #previous ones are empty
23         if re.search("Index=",line):
24             #take the index to write empty cells in a for
25             tocolumn=int(re.findall(r'[^"]*',line)[0])
26             #write as many ";" as empty cells are between
27             #these two values
28             csv+=";"*(tocolumn-columnnumber)
29             columnnumber=tocolumn-1
30             #create a cell with its respective information
31             if re.search("Data",line):
32                 line=re.sub(r'<.*?>', ' ', line)
33                 csv+=line+";"
34                 columnnumber+=1
35             #new row
36         else:
37             columnnumber=1
38             if not firstrow:
39                 csv+="\n"
40             #delete spaces
41             csv=re.sub(r'[ \t\r\f\v]', ' ', csv)
42             #return dataframe
43             df = pd.read_csv(StringIO(csv),delimiter=";")

```

Todo eso ocurre desde que se escanea un archivo hasta que se selecciona submit:

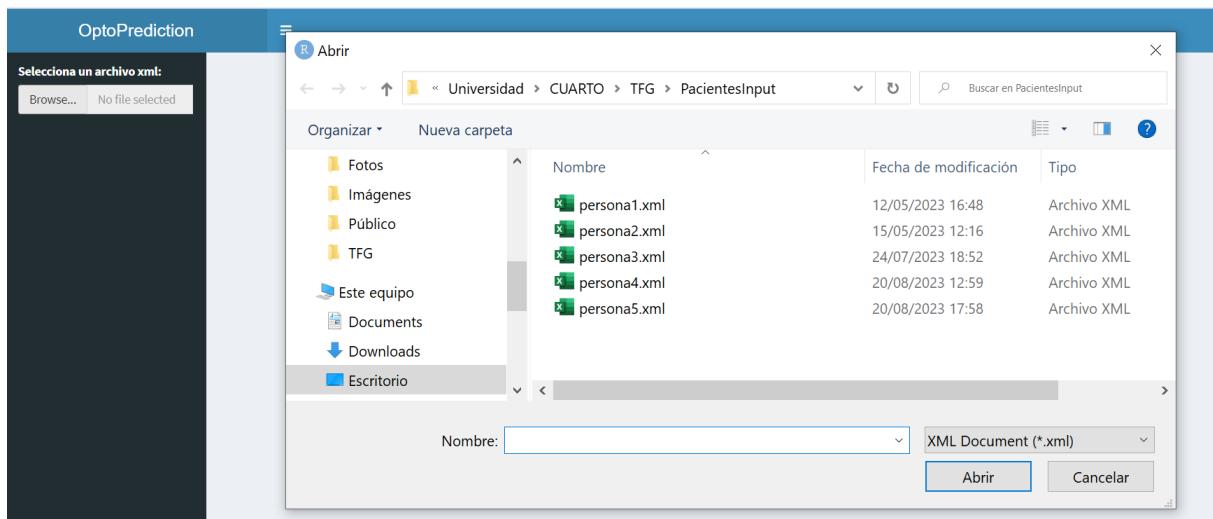


Figura 4.2: File Submition

El botón 'Browse' permite su uso en cualquier momento, lo cual reiniciará los datos guardados por los de un nuevo archivo escaneado.
Recargar la página hará que se pierdan los datos trabajados hasta el momento, habrá que subir de nuevo el fichero e introducir el Peso, Altura y Número de Pie.

CAPÍTULO 4. RSHINY APP



Figura 4.3: Uploaded File

Una vez introducidos estos tres valores, se habilitará el botón 'Predict' (4.3) que tras clicarlo aparecerá las solución con probabilidades de la predicción.

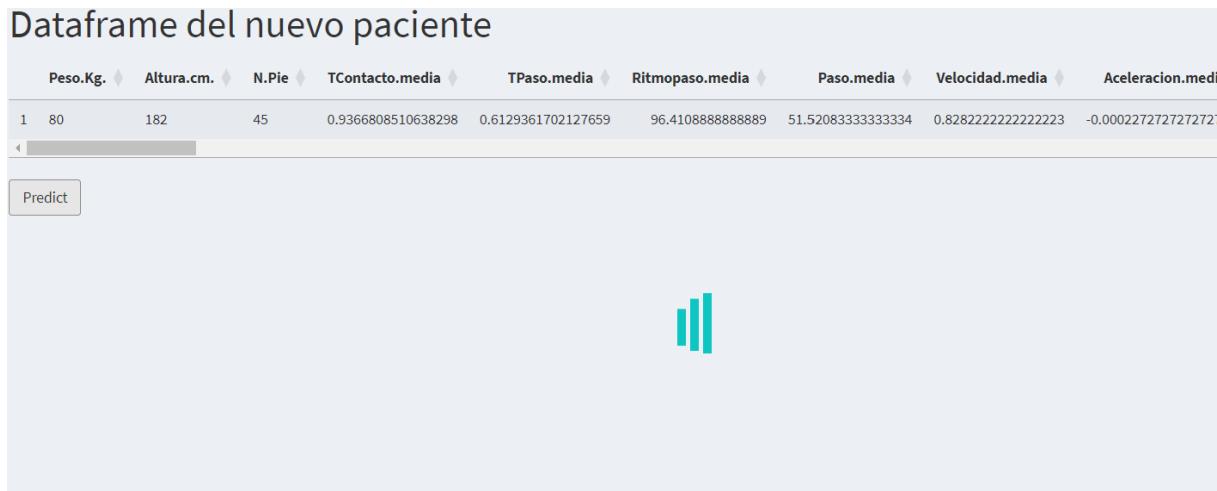


Figura 4.4: Prediciendo

Tras unos segundos de carga para realizar la predicción, aparecen los resultados.

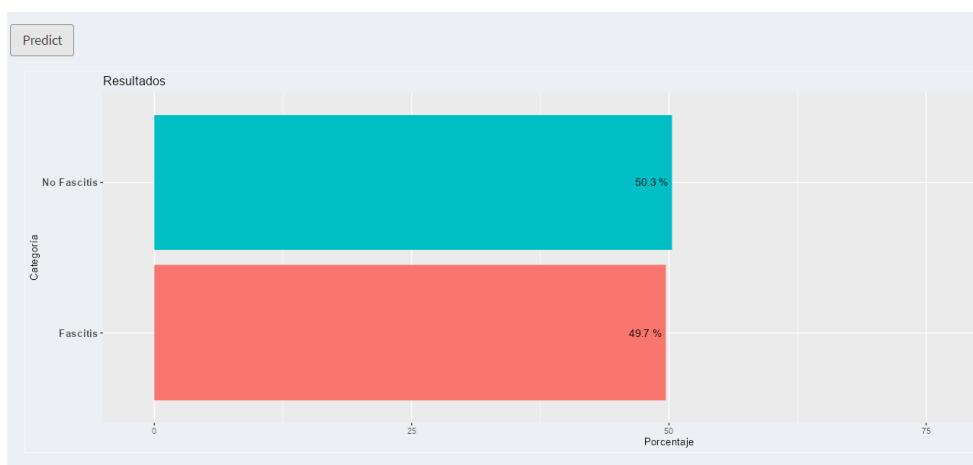


Figura 4.5: Resultados de la predicción

CAPÍTULO 5

Conclusiones y líneas futuras

5.1. Conclusiones

A pesar de los desafíos que enfrentamos durante estos meses de investigación, hemos llevado a cabo un riguroso proceso de limpieza de datos y mantenido un diálogo constante con los profesionales de la podología. A lo largo de este tiempo, hemos tomado importantes decisiones sobre variables y abordado preguntas fundamentales relacionadas con nuestros datos.

Aunque los resultados finales pueden no cumplir completamente con nuestras expectativas y por el momento no pueda llevarse al uso cotidiano laboral, es importante destacar que este proyecto ha proporcionado un primer contacto en la implementación de un modelo como este, haciendo más sencillo y creando una idea inicial de como puede ser el 'modus operandi' en la clínica una vez se haya desplegado la aplicación *OptoPredict*.

Es cierto que se han encontrado inconvenientes durante el proceso, que pueden ser debidos a:

1. Tamaño del Conjunto de datos. Inicialmente, contábamos con un conjunto de datos aparentemente suficiente. Sin embargo, tras el procesamiento, nos encontramos con un conjunto más pequeño de alrededor de 500 muestras. Esto significó complicaciones en la búsqueda de los mejores parámetros y validación cruzada, sin embargo se ha demostrado que incluso con recursos limitados, se puede realizar un estudio avanzado sobre unos datos, obtener información nueva e iniciar un proyecto. Con la adquisición de nuevas muestras, el modelo resulta prometedor para enfrentarse a casos de predicción reales.
2. Etiquetado Muestras. El etiquetado de las muestras no ha sido tarea fácil debido a la naturaleza ambigua de los registros médicos. Esto a la hora de entrenar altera mucho los resultados al ser el etiquetado tan irregular.

Una manera de solución sería un PLN (Procesamiento de Lenguaje Natural) que implicaría más trabajo pero sería muy eficaz, o etiquetar todos los pacientes manualmente. Esto último, además de ser muy costoso y requerir mucho tiempo, en el modelo estudiado y mostrado en este proyecto no mejoraría mucho porque mi capacidad para analizar e interpretar los historiales patológicos con mis conocimientos en podología es muy similar a la automatizada (leer 'fasc', asignar Fascitis).

Una solución para ambos problemas sería la recopilación de los datos de los pacientes que vayan asistiendo a la clínica por parte de los clínicos, teniendo en cuenta su posterior uso para un programa informático como este. Clasificar pacientes claros que padeczan fascitis aunque siempre teniendo en cuenta no aumentar el trabajo de los profesionales puesto que en un final idílico, el objetivo de estas investigaciones es mitigarle el esfuerzo a ellos.

Unificar los tres programas donde guardan los datos de los pacientes, otorgaría una mejora considerable ya que una proporción de la información se ha perdido cuando se han unificado todas las bases de datos y muchas de las tuplas no coincidían en la clave primaria.

Como última opción está la posibilidad de que estemos ante datos no modelables, sin embargo, por lo que hemos podido ver en este estudio, las predicciones han ido mejorando conforme se ajustaba y se desarrollaba el modelo hasta el límite que se ha alcanzado. Parece mostrar que conforme se aumenten los recursos aplicados aumentará la calidad de este resultado.

5.2. Futuras líneas de trabajo

A través de este primer paso que he podido realizar en el intento de modelado de los datos de la Clínica Dr. Páez, se crea una pequeña idea de cómo son los datos. Se pueden descartar caminos que ya han sido tomados y plantear otras maneras para llegar a crear un modelo útil y que no solo prediga los casos de fascitis, si no que de varias patologías más también.

Según mi punto de vista, para poder tratar con este problema en un futuro, buscaría más datos de Optogait en otras clínicas para incrementar la dimensión de la información en un futuro, o simplemente esperar algún tiempo más hasta tener la suficiente cantidad de datos.

En este campo hay aún muchas opciones inexploradas que pueden ser investigadas y dar apertura a gran cantidad de aplicaciones en el ámbito de la podología con tal de facilitar el trabajo clínico y por supuesto, mejorar la condición física de las personas.

Bibliografía

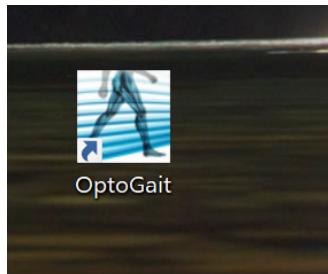
1. K. Zhao, *Pre-Process Data with Pipeline to Prevent Data Leakage during Cross-Validation*. (2023; <https://towardsdatascience.com/pre-process-data-with-pipeline-to-prevent-data-leakage-during-cross-validation-e3442cca7fdc>).
2. M. Corporation, *Que es OptoGait* (2023; <http://www.optogait.com/que-es-optogait>).
3. M. Corporation, *Manual de uso OptoGait* (2023; <http://www.optogait.com/optogaitportal/media/manuals/manual-es.pdf>).
4. Bernd Bischl, Raphael Sonabend, Lars Kotthoff, Michel Lang, ed., *Applied Machine Learning Using mlr3 in R* (CRC Press, 2024), ISBN: 9781032507545, (<https://mlr3book.mlr-org.com>).
5. scikit-learn contributors, *scikit-learn: Machine Learning in Python* (2023; <https://scikit-learn.org/>).
6. J. Brownlee, *Random Oversampling and Undersampling for Imbalanced Classification* (2023; <https://machinelearningmastery.com/random-oversampling-andundersampling-for-imbalanced-classification/>).
7. Wikibooks/Latex contributors, *Latex* (2023; <https://en.wikibooks.org/wiki/LaTeX>).
8. *GPT-3.5: OpenAI's Language Model* (2023; <https://www.openai.com/research/gpt-3.5>).
9. P. RStudio, *Shiny: Web Application Framework for R* (2023; <https://shiny.rstudio.com/>).
10. P. D. Team, *pandas: Powerful data analysis tools for Python* (2023; <https://pandas.pydata.org/>).
11. @kevinushey, *Shiny app crashes in connection with reticulate* (2023; <https://github.com/rstudio/reticulate/issues/683>).
12. P. S. Foundation, *documentación de Python - 3.11.5* (2023; <https://docs.python.org/es/3/>).
13. The R Project for Statistical Computing, *CRAN - The Comprehensive R Archive Network* (2023; <https://cran.r-project.org/>).
14. Jonghyeok Chae, Young-Jin Kang, Yoojeong Noh, A Deep-Learning Approach for Foot-Type Classification Using Heterogeneous Pressure Data. (2020).

Apéndices

APÉNDICE A

Guía de uso de la aplicación OptoPredict

Mostraremos una guía de uso completa de como utilizar la aplicación web desarrollada en este proyecto *OptoPredict* sacando los datos del programa de *Optogait*.



1. Primero, abrir la aplicación *Optogait*..

Figura A.1: Icono de Optogait

2. Una vez abierta, se despliega el menú principal:



Figura A.2: Menú de Optogait

APÉNDICE A. GUÍA DE USO DE LA APLICACIÓN OPTOPREDICT

3. Podemos ver varias opciones para clicar. Pulsamos en resultados.



Figura A.3: Menú de Optogait

4. Ahora se abrirá una lista con todos los pacientes. Desde aquí, debemos pulsar la flecha que aparece a la derecha de cada paciente, pero clicamos solo uno.

Figura A.4: Menú de Optogait

5. A continuación exportamos el paciente para conseguir el archivo .XML que tiene sus datos y que llevaremos a la app de predicciones:

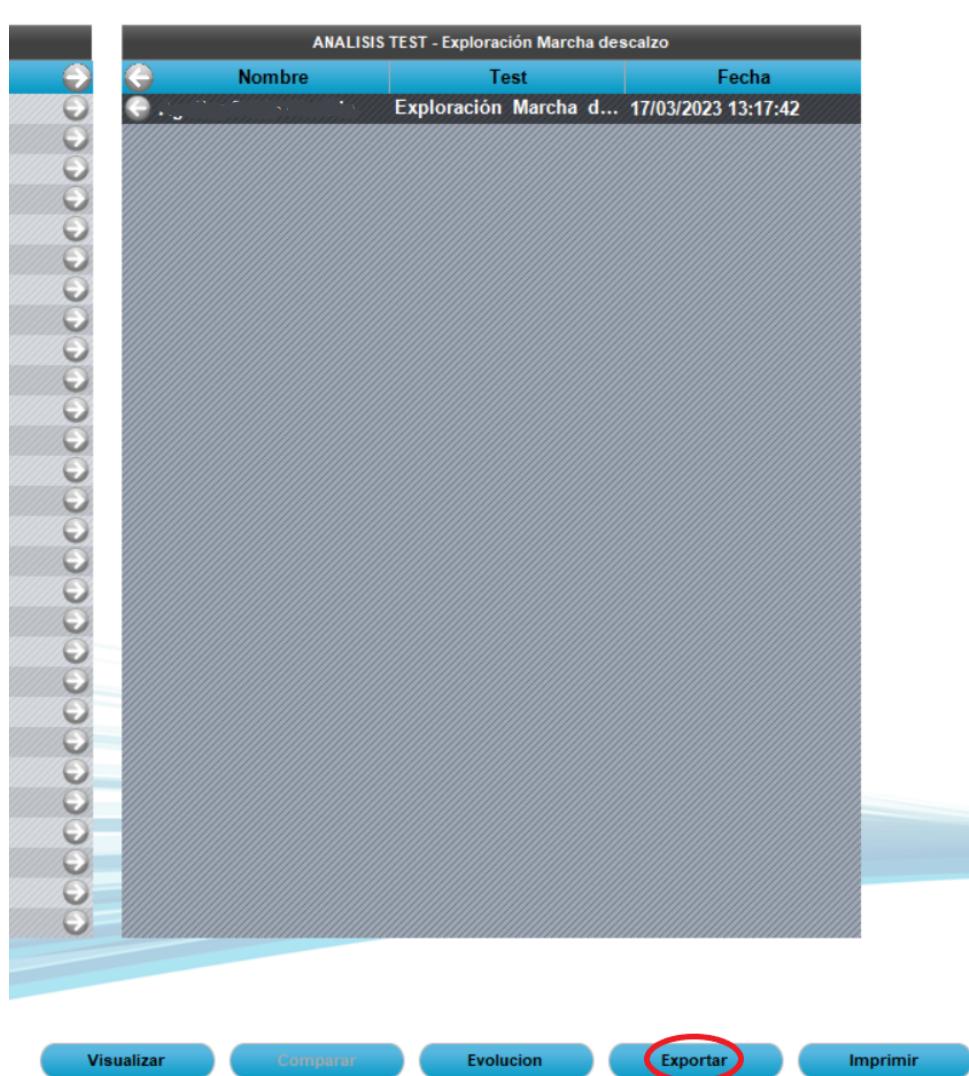


Figura A.5: Exportar

6. Una vez pulsamos ahí nos aparece una ventana, dejamos seleccionado la primera opción (Excel/XLM Extended) y seleccionamos 'Confirmar'.

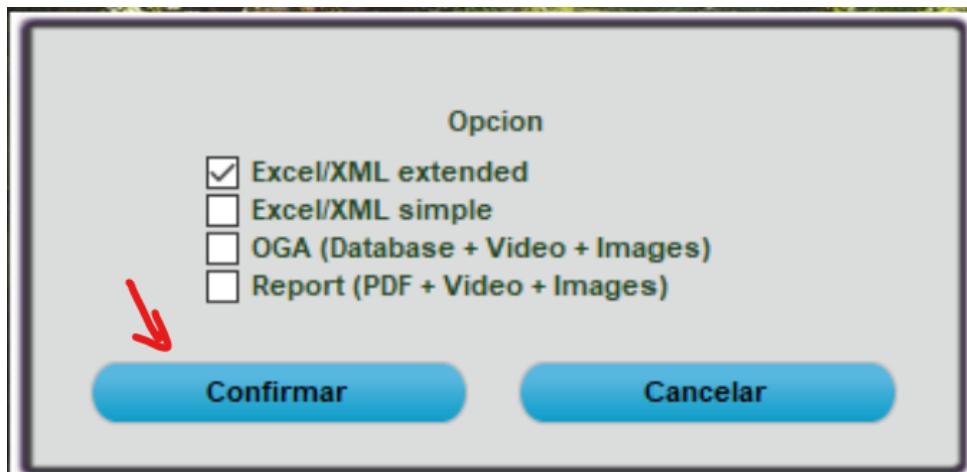


Figura A.6: Finalizar Exportación

Ahora ya habremos exportado nuestro archivo .XML y podremos abrir la otra aplicación para hacer la predicción.

7. Abrimos la app y desplegamos el menú.

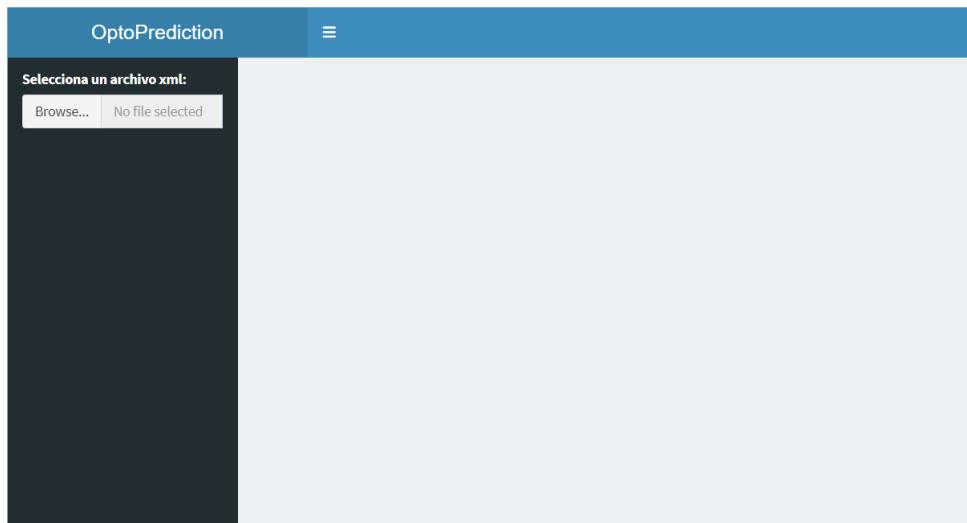


Figura A.7: Menú de la aplicación OptoPredict

8. Elegimos un archivo y lo subimos.

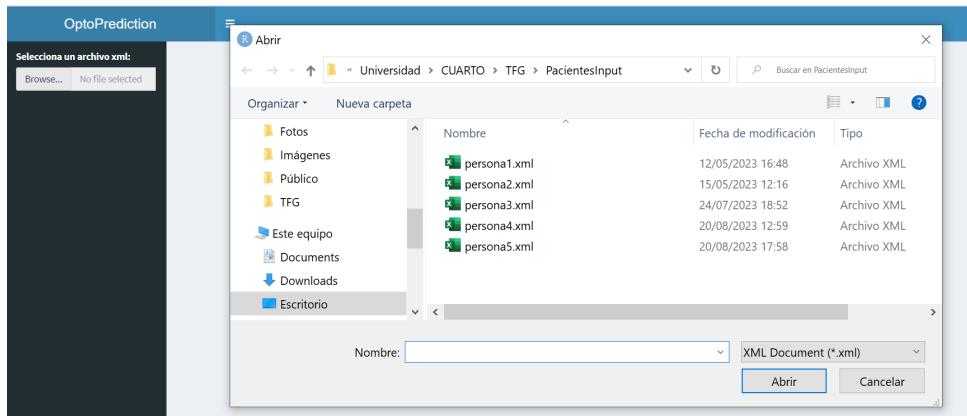


Figura A.8: Importar archivo

9. Ahora nos aparecerán los datos del paciente, podemos cambiar cualquiera, aunque debemos introducir valores para el Peso, Altura y Nº de Pie obligatoriamente para poder realizar la predicción. Una vez se haya hecho esto último, se podrá pulsar el botón de 'Predecir'.

	Peso.Kg.	Altura.cm.	N.Pie	TContacto.media	TPaso.media	Ritmopaso.media
1	<input type="text"/>	▲	▲	0.9366808510638298	0.6129361702127659	96.41088888888889

Introduce el Peso, Altura y Nº de Pie

Predict

Figura A.9: Introducir valores

10. Seleccionamos el botón de 'Predecir' una vez hecho el paso anterior.

	Peso.Kg.	Altura.cm.	N.Pie	TContacto.media	TPaso.media	Ritmopaso.media	Paso.media
1	80	182	45	0.9366808510638298	0.6129361702127659	96.41088888888889	51.52083333333333

Predict

Figura A.10: Pulsar botón Predecir

11. Obtenemos los resultados.

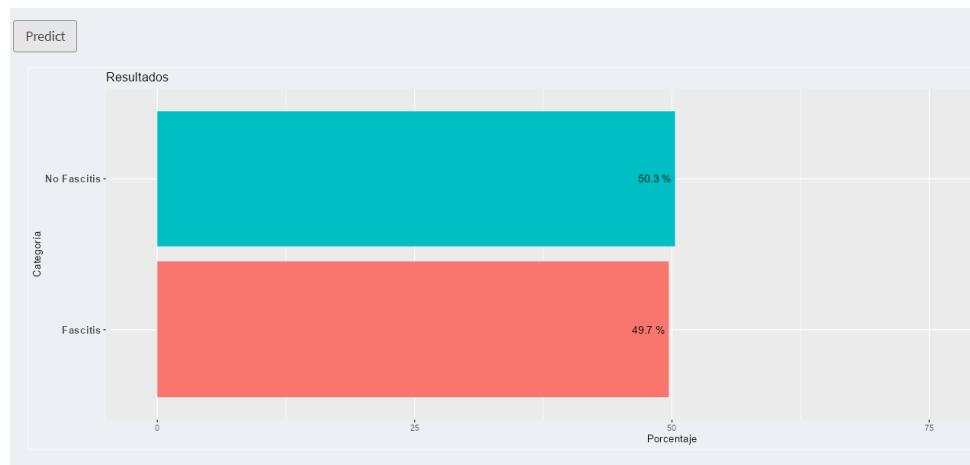


Figura A.11: Resultados