UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

TFG del Grado en Ingeniería Informática

PDDetection
Aplicación de técnicas de minería de datos para la detección de la enfermedad del Parkinson
Documentación Técnica

Presentado por Adrián Arnaiz Rodríguez
en Universidad de Burgos — 2 de julio de 2019
Tutores: Jose Francisco Díez Pastor y César Ignacio García Osorio

Índice general

Índice de figuras

Índice de tablas

Apéndice A

Plan de Proyecto Software

A.1. Introducción

En este apartado se desarrollará la planificación temporal del proyecto, así como también un estudio de viabilidad.

A.2. Planificación temporal

Para la planificación temporal del proyecto, se sigue una metodología *Scrum*. Se realizarán diferentes *sprints*, en los que se marcaban objetivos. Idealmente, los *sprints* duran dos semanas. Debido a carga de trabajo externa a este proyecto, esta duración se ha visto afectada y se han realizado *sprints* de más y menos duración. Durante cada uno de ellos, se irán creando y realizando las diferentes tareas correspondientes a los objetivos fijados en cada *sprints*. Al finalizar cada *sprint*, se realizan reuniones con los tutores para revisar los objetivos cumplidos y marcar unos nuevos.

Se ha utilizado Github para el seguimiento de la aplicación, ayudados por el tablero Kanban de Zen-hub. Esta herramienta nos ha permitido llevar un seguimiento más detallado de la planificación en cada momento. El repositorio del proyecto, y las issues realizadas se encuentra en el repositorio del proyecto[1].

A su vez, cabe destacar que los primeros *sprints* no se muestran de manera correcta. Esto es debido al fallo de no cerrar los *sprints* en github, por lo que los gráficos *Burndown* están alterados.

[1] https://github.com/AdrianArnaiz/TFG-Neurodegenerative-Disease-Detection

Sprint 1

Idealmente, el comienzo de este sprint comenzaba el 15 de Diciembre. Debido a cargas de trabajo externas al proyecto, al final se comenzó el 14 de febrero. Duración del sprint: 14 de febrero 2019 al 28 de febrero 2019. Ver gráfico *Burndown* en la figura A.1.

Figura A.1: Burndown chart del sprint 1

Tareas realizadas:

- Instalación del cliente Github: GitTortoise.
- Crear la estructura de directorios del repositorio.
- Instalación de LaTeX y todos sus componentes.
- Lectura en profundidad de dos artículos de Giovanni Dimauro: [2] y [1] .

En este sprint se realizó la fase inicial del proyecto, que comprendía desde la instalación y creación de elementos principales, hasta la iniciación de la investigación.

Sprint 2

Duración del sprint: 28 de febrero de 2019 al 13 de marzo de 2019. Ver gráfico *Burndown* en la figura A.2.

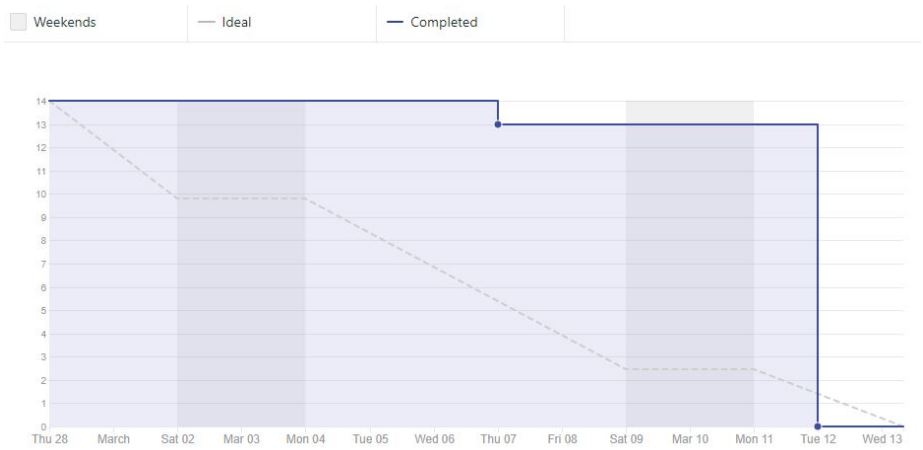

Figura A.2: Burndown chart del sprint 2

Tareas realizadas:

- Realizar la taxonomía de las características extraídas de cada audio, qué características de cada tipo...
- Identificar las herramientas de extracción de características de audio.
- Realizar taxonomía de correspondencia entre las herramientas y las características concretas a extraer.
- Realizar un arreglo en la preservación de privacidad de los audios del conjunto de datos. Los audios son audios privados definidos en [5]. Por un error con los archivos ignorados por *Git*, se subieron algunos audios. Se tuvieron que borrar para mantener la privacidad de los mismos.

En este sprint, seguimos explorando el estado del arte ahora de manera más precisa: empezar a ver en profundidad el proceso de extracción de características y medidas de los audios, explorando en diferentes artículos que características extraer y buscando las herramientas necesarias para su extracción.

Sprint 3

Duración del sprint: 13 de marzo 2019 al 29 de marzo 2019. Ver gráfico *Burndown* en la figura A.3.

Figura A.3: Burndown chart del sprint 3

Tareas realizadas:

- Búsqueda de audios, peticiones y realización de taxonomía de audios encontrados.
- Lectura en profundidad de los artículos más importantes del estado del arte, i.e. [6].
- Finalización de exploración del estado del arte: realizar taxonomía y resúmenes.

En este sprint se llevó a cabo la finalización de estado del arte y su resumen de aspectos y artículos más importantes. Un aspecto importante fue la búsqueda de audios, trabajo bastante arduo dentro del proyecto debido a que son muy difíciles de encontrar. Empezamos el proceso de extracción de características descrito en [6]: instalación y familiarización de librerías, segmentación de audios, etc...

Sprint 4

Duración del sprint: 29 de marzo 2019 al 11 de abril 2019. Ver gráfico *Burndown* en la figura A.4.

Figura A.4: Burndown chart del sprint 4

Tareas realizadas:

- Instalar herramientas de manejo de audio como Disvoice.
- Explorar el funcionamiento de estas herramientas.
- Preprocesar los audios: eliminar silencios inicial y final.
- Extracción de diferentes características de los audios con Disvoice.
  - Extracción medidas prosódicas de audios completos.
  - Extraer medidas de fonación de *voiced frames* de vocales y frase.
  - Extraer medidas de articulación (cepstrales y de energía de transiciones).

Se realizó el proceso de extracción de características. Se recorre toda la estructura de audios para crear las correspondientes matrices de características (en numpy) de cada tipo de audio y así dejar preparados los datos para entrenar modelos. Hay una matriz de características por cada tipo de audio y cada tipo de características. Para ello se realizó desde la instalación y comprensión del funcionamiento de las herramientas de manejo de audio, hasta extraer las medidas completas con la herramienta Disvoice.

Sprint 5

Duración del sprint: 15 de abril 2019 al 2 de mayo 2019. Ver gráfico *Burndown* en la figura A.5.

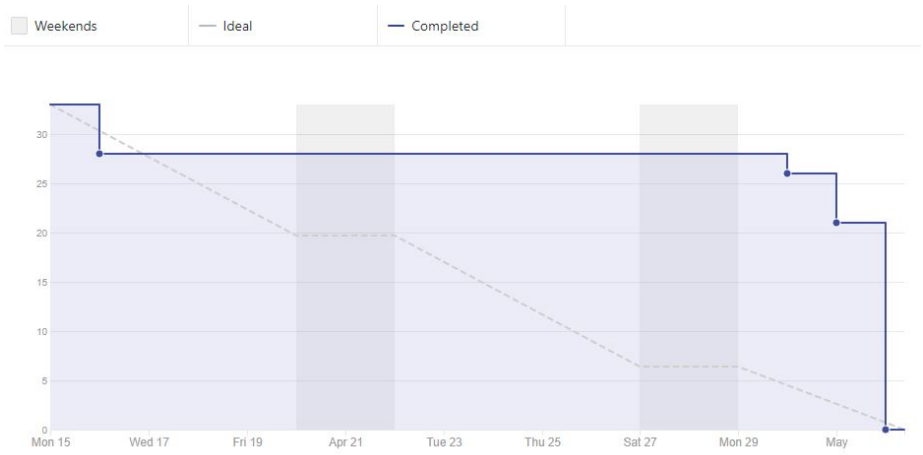

Figura A.5: Burndown chart del sprint 5

Tareas realizadas:

- Documentación de avances hasta el momento.
- Realización de los **primeros experimentos** con clasificadores, incluyendo la limpieza de características necesaria para ello. Se realiza, como mínimo, el proceso de [6]. Se utilizan las características extraídas con *Disvoice*.
- Realización proyección de características.
- Investigación, documentación y aprendizaje sobre *Deep Learning*.

En este sprint, se documentó el trabajo realizado hasta el momento. También se realizó el estudio con clasificadores para los conjuntos de características extraídos con Disvoice. Probamos proyecciones de datos de estas características para graficarlos. En este sprint se comenzó la investigación y aprendizaje sobre Deep Learning, para su aplicación a posteriores sprints.

Sprint 6

Duración del sprint: 2 de mayo 2019 al 17 de mayo 2019. Ver gráfico *Burndown* en la figura A.6.

Figura A.6: Burndown chart del sprint 6

Tareas realizadas:

- Búsqueda de librerías de *Deep Learning* para audios.
- Modificación de las características Disvoice.
- División del conjunto de datos por sexo.
- Realización de **segundos experimentos** con características Disvoice modificadas. Se describe la segunda fase en la memoria en el apartado Aspectos Relevantes. Se realiza, como mínimo, el proceso de [6], pero añadiendo edad y sexo a las características extraídas y dividiendo los conjuntos por sexo.

En este *sprint* realizamos la modificación de las características de *Disvoice* (adición de edad y sexo, división por sexos...) y realizamos los experimentos con estas nuevas características. También, siguiendo la línea del anterior *sprint*, profundizamos más en el *Deep Learning*, Buscando bibliotecas para la extracción de características de nuestros audios.

Sprint 7

Duración del sprint: 17 de mayo 2019 a 14 de junio 2019. Ver gráfico *Burndown* en la figura A.7.

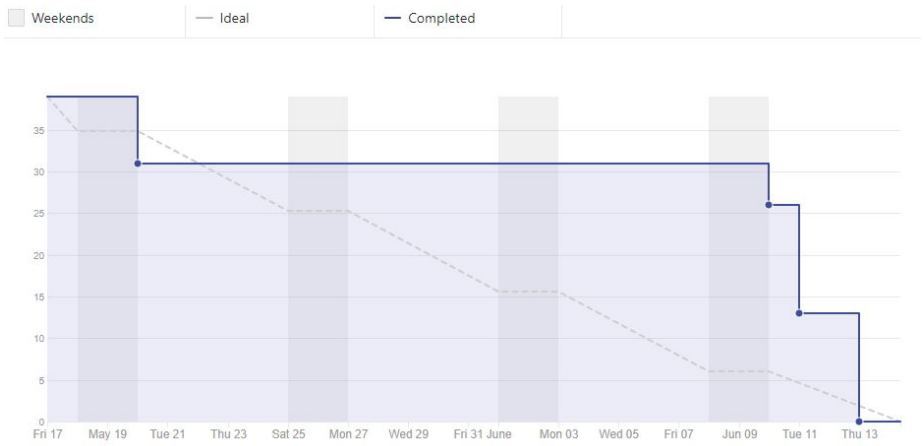

Figura A.7: Burndown chart del sprint 7

Tareas realizadas:

- Arreglo de fallo encontrado.
- Repetición de primeros y segundos experimentos: incluye repetición de análisis y resumen de resultados.
- Instalación biblioteca VGGish (y VGGish2Keras).
- Realizar extracción de características con VGGish (VGGish2Keras).
- Realización **terceros experimentos** con características VGGish. Se hace los experimentos con los embeddings y espectros extraídos con *VGGish* [4] .
- Análisis de viabilidad comercial: *lean canvas*.

En la reunión previa a este sprint, se revisó el código y encontramos un fallo de **data snooping**. No hacíamos validación cruzada anidada (*nested CV*). Debido a esto, tuvimos que arreglar el error y repetir los experimentos realizados hasta el momento (primeros y segundos con características Disvoice). Tras finalizar, tuvimos que volver a analizar y resumir todos los resultados de los experimentos, aunque estos habían variado menos de lo esperado. También, siguiendo con la línea de anteriores *sprints*, realizamos la instalación de la biblioteca *VGGish* de *Deep Learning*, extrajimos las características de los audios con esta biblioteca y realizamos los experimentos con estas nuevas características. Para finalizar, y por pedido de la OTRI

(Oficina de transmisión de información de la Universidad de Burgos), se realizó un análisis de viabilidad comercial, concretamente un *lean canvas* de nuestro proyecto.

Cabe destacar que la duración de este *sprint* es bastante mayor de lo normal. Se debe a dos aspectos. El primero de ellos es que la carga de trabajo de este *sprint* era sustancialmente superior a los demás. Se debe a que en fallo encontrado requería de una repetición de gran parte del trabajo y además no podíamos frenarnos en los experimentos con *Deep Learning*. Por otra parte, coincidió con las dos últimas semanas del curso, las cuales estaban repletas de trabajos finales y exámenes de convocatoria, que hacían que el tiempo dedicado al proyecto tuviera que ser menor.

Sprint 8

Duración del sprint: 14 de junio 2019 a 24 de junio 2019. Ver gráfico *Burndown* en la figura A.8.

Figura A.8: Burndown chart del sprint 8

Tareas realizadas:

- Aprendizaje del patrón de diseño mediador-fachada.
- Realización de la aplicación de escritorio.

En este sprint se realizó la aplicación demostrativa de escritorio. Para ello, se sugirió hacer mediante el patrón de diseño mediador-fachada. Por ello

antes de realizar la aplicación se tuvo que comprender ese patrón. Una vez comprendido se realizó la aplicación. Como ya hemos comentado en la memoria principal, esta aplicación, aunque es la parte más vistosa, personalmente no la considero la parte más importante del proyecto. Únicamente es una aplicación que muestra una posible aplicación del resultado de este investigación, y que con las líneas de investigación futuras pudiera llegar a un producto funcional y estable.

Sprint 9

Duración del sprint: 24 de junio 2019 a 1 de junio 2019. Ver gráfico *Burndown* en la figura A.1.

Figura A.9: Burndown chart del sprint 9

Tareas realizadas:

- Revisión de la memoria hasta el momento.
- Finalización de la memoria.
- Búsqueda de documentación de pruebas unitarias en Minería de Datos.
- Realización documentación de las clases.
- Realización test unitarios.
- Finalización de los anexos y el anexo de investigación donde detallamos las taxonomías.

En este último sprint se hicieron las últimas tareas del proyecto. Se terminó toda la documentación, lo que comprendía desde revisar lo hecho anteriormente, realizar lo que faltaba por terminar y buscar información sobre pruebas unitarias en el campo de la minería de datos.

Frecuencia de código

Añadimos la gráfica de frecuencia de código, donde se ve como se han distribuido las adiciones y modificaciones de código a lo largo del proyecto. Ver Figura A.10

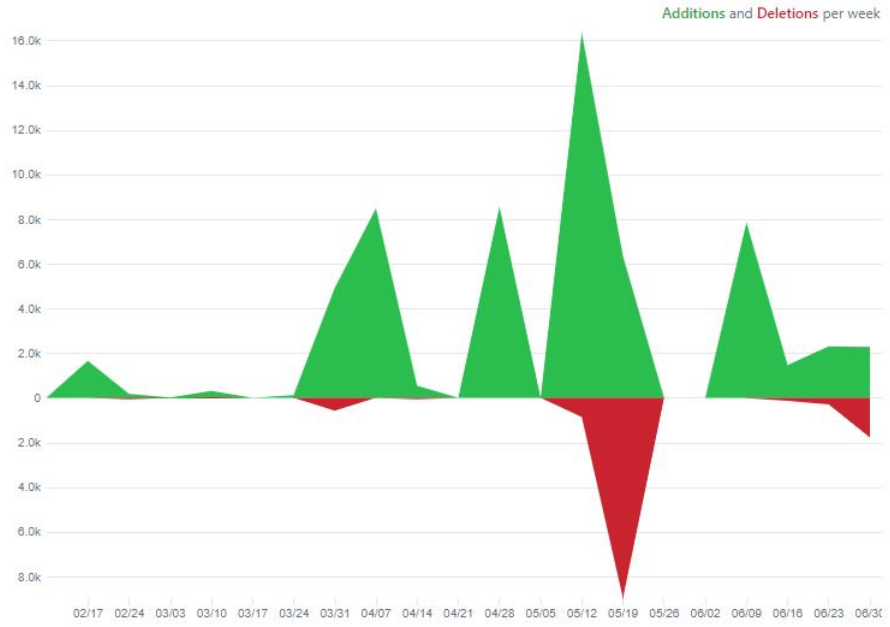

Figura A.10: Frecuencia de código

A.3. Estudio de viabilidad

En este apartado se abordará la viabilidad económica, comercial y legal de este proyecto.

Viabilidad económica

En este apartado, simularemos el coste del proyecto que tendría en una empresa, o en una venta al público.

Coste personal

Este proyecto cuenta con dos **profesores contratados** durante 6 meses para este proyecto. Por cada profesor se asignan 0,5 créditos, por lo que el coste[2] por tutor consideramos que es el siguiente:

- Sueldo ayudante doctor: 26445,24 euros anuales. Más 2 trienios más el complemento de mejora: $26445,24 + 1260,58 + 1110,71 = 28816,52$. En total imparte 24 créditos por lo que el coste anual por crédito es de 1200,68 euros. El coste total es el siguiente:

$$1200,68 \cdot 0,5 créditos = 600,34 euros \tag{A.1}$$

- Sueldo medio anual de un titular: 42220,34 euros anuales. En total imparte 24 créditos por lo que el coste anual por crédito es de 1759,18 euros. El coste total es el siguiente:

$$1759,18 \cdot 0,5 créditos = 879,59 euros \tag{A.2}$$

También cuanta con un desarrollador del proyecto. Supondremos un salario bruto de 2000 euros para el desarrollador. Habrá que tener en cuenta los gastos de seguridad social[3].

- Cotización por parte de la empresa: 23.6 %
- Cotización por parte del empleado: 4.7 %
- **Total: 28.3 %**

Por lo tanto el coste del desarrollador se detalla en A.1

Concepto	**Coste**
Salario mensual neto	1217.25
Retención IRPF (15 %)	216.75
Seguridad social (28.3 %)	566
Salario mensual bruto	2000
Coste total (6 meses)	12000

Tabla A.1: Costes de personal

[2] https://www.ubu.es/sites/default/files/portal_page/files/pdi_laboral_2019.pdf

[3] http://www.seg-social.es/wps/portal/wss/internet/Trabajadores/CotizacionRecaudacionTrabajadores/

Coste informático

El coste informático es mínimo. Se ha utilizado un único portátil personal. Su coste fue de 800 euros y se supone una amortización de 5 años. Para ello se contabilizará únicamente el cose amortizado.

$$(800/(12 \cdot 5)) \cdot 6 = 80 euros \tag{A.3}$$

Ingresos

Para este proyecto se ha contado con la concesión de la beca prototipos de la Oficina de Transmisión de Información de la Universidad de Burgos. La cuantía de la beca ha sido de 800 euros, por lo que estos serán los ingresos del proyecto.

Coste Total

El coste total se detalla en la tabla A.2

Concepto	**Coste**
Coste desarrollador	12000
Costes Tutores(15 %)	1479.93
Hardware	80
Beca prototipos	-800
Coste total (6 meses)	12679.93

Tabla A.2: Coste Total

Viabilidad comercial

Para analizar la viabilidad comercial, realizaremos un *lean canvas*, A.11, la cual es una herramienta de análisis de negocio desarrollada por Alex Osterwalder [7].

Viabilidad legal

Todo el código utilizado es de dominio público. Hemos utilizado multitud de librerías del lenguaje Python, las cuales son todas de dominio público. Las herramientas más concretas que hemos utilizado han sido las siguientes.

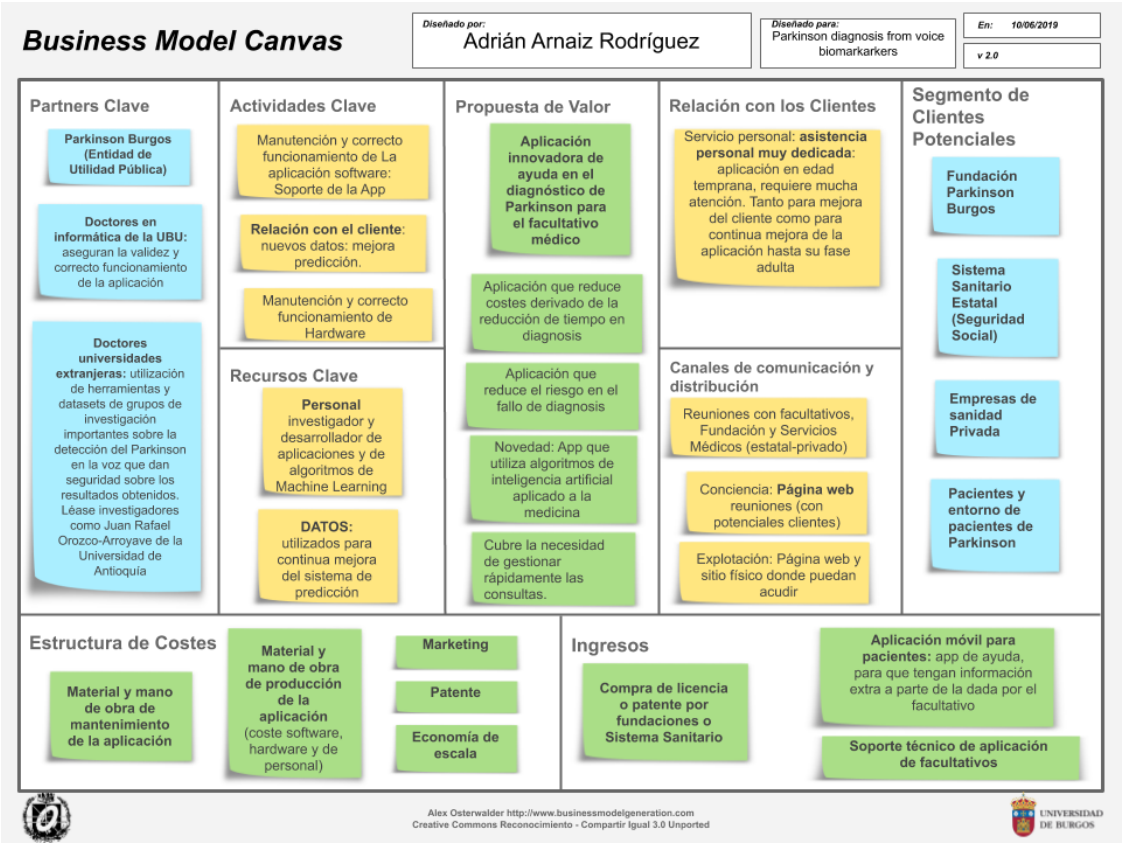

Figura A.11: Lean Canvas del proyecto

- **Disvoice**: alojado en repositorio público[4] de Github. **Tiene la licencia pública MIT**.
- **VGGish**: alojado en repositorio público de Github[5].Pertenece al proyecto de Tensorflow. **Tiene la licencia pública Apache license 2.0**.

Todas las demás librerías utilizadas son propias de Python. También mencionar que no utilizamos el repositorio Disvoice original, sino que le hemos modificado para la ejecución en Windows. Esto ha sido posible ya que la licencia MIT permite tanto el uso comercial como la modificación el mismo.

[4] https://github.com/jcvasquezc/DisVoice
[5] https://github.com/tensorflow/models/tree/master/research/audioset

Apéndice B

Especificación de Requisitos

B.1. Introducción

En esta sección se abordarán los diferentes objetivos y requisitos del proyecto. Se presentarán tanto los requisitos globales del proyecto, como los requisitos funcionales y casos de uso de la aplicación.

B.2. Objetivos generales

- Investigar y condensar el estado del arte de la investigación sobre la detección del Parkison a través de la voz resumiendo los artículos científicos más importantes, identificando las tecnologías, herramientas y procesos actuales utilizadas en ellos, realizando taxonomías, etc.

- Recopilación de bases de datos adecuadas para la investigación, tanto para uso en este proyecto como para su uso en proyectos posteriores, cerciorando que son *datasets* de audios correctos para las tareas necesitadas (i.e. audios etiquetados).

- Realización de un estudio comparativo en cuanto a la utilización de diferentes modelos de clasificación y conjuntos de características extraídas de los audios. Se compararán resultados, tanto entre los diferentes experimentos que nosotros realicemos, como entre nuestros mejores experimentos y resultados científicos de anteriores experimentos (publicados en artículos científicos).

- Aportación de una nueva perspectiva a este campo de investigación: extracción de las características de los audios mediante *Deep Learning*.
- Finalmente, utilizar todo lo descrito anteriormente para realizar una aplicación la cual permita la monitorización de la diagnosis de la enfermedad del Parkinson a través de la voz. La aplicación será capaz de discernir, mediante un clasificador, si la persona de un audio subido a la aplicación web tiene Parkinson o no. Esta aplicación será un ejemplo de como poder plasmar los resultados de esta investigación en una herramienta funcional.

B.3. Catalogo de requisitos

En esta sección se enumerarán los requisitos funcionales de nuestra aplicación y los actores que la realizan.

Actores:

Usuario El usuario será quien utiliza la aplicación. Idealmente, será el facultativo médico en su consulta.

Científico de datos Este actor será quien realiza la investigación, quien realiza los experimentos con los clasificadores.

- **RF.1** Crear una herramienta que nos permita detectar con que probabilidad tiene Parkinson la persona de un audio dado.
  - **RF.1.1** El usuario podrá elegir el audio que quiere predecir.
  - **RF.1.2** El usuario podrá añadir la edad y el sexo del paciente.
  - **RF.1.3** El usuario podrá ver la probabilidad en porcentaje de PArkinson de la persona del audio.
- **RF.2** Crear una herramienta que permita ver las gráficas de amplitud de onda y el espectrograma de frecuencias de un audio.
  - **RF.2.1** El usuario podrá elegir el audio del que quiere ver las gráficas.
  - **RF.2.2** El usuario podrá ver las gráficas en la pantalla.
  - **RF.2.3** El usuario podrá navegar en las gráficas (zoom, moverse).
  - **RF.2.4** El usuario podrá guardar las gráficas.

- **RF.3** Crear una herramienta que permita reproducir un audio.
  - **RF.3.1** El usuario podrá elegir el audio que quiere reproducir.
  - **RF.3.2** El usuario podrá reproducir un audio dando al play.
  - **RF.3.3** El usuario podrá pausar el audio dando al pause.
  - **RF.3.4** El usuario podrá rebobinar el audio dando a rebobinar.
  - **RF.3.5** El usuario podrá cambiar el volumen deslizando una pestaña.
  - **RF.3.6** El usuario podrá enmudecer el audio pulsando al mute.
- **RF.4** Crear una herramienta que permita cargar y borrar audios al entorno para trabajar con ellos.
  - **RF.4.1** El usuario podrá elegir los audios de cualquier ruta que quiere cargar en la aplicación.
  - **RF.4.2** El usuario podrá elegir los audios cargados en la aplicación que quiere borrar.
- **RF.5** Crear modelos de clasificación para los audios con Python.
  - **RF.5.1** El científico de datos podrá cargar datos de los audios.
  - **RF.5.2** El científico de datos podrá elegir el tipo de modelo que queremos entrenar.
  - **RF.5.3** El científico de datos podrá elegir la validación cruzada que queremos realizar.
  - **RF.5.4** El científico de datos podrá particionar los datos de manera estratificada.
  - **RF.5.5** El científico de datos podrá entrenar el modelo con la anterior configuración.
- **RF.6** Evaluar clasificadores.
  - **RF.6.1** El científico de datos podrá obtener medidas de rendimiento de los clasificadores.
  - **RF.6.2** El científico de datos podrá resumir medidas de rendimiento de todos los clasificadores.
  - **RF.6.3** El científico de datos podrá mostrar medidas de rendimiento en tablas o gráficas para ayudar a la decisión.

B.4. Especificación de requisitos

Diagrama de casos de uso

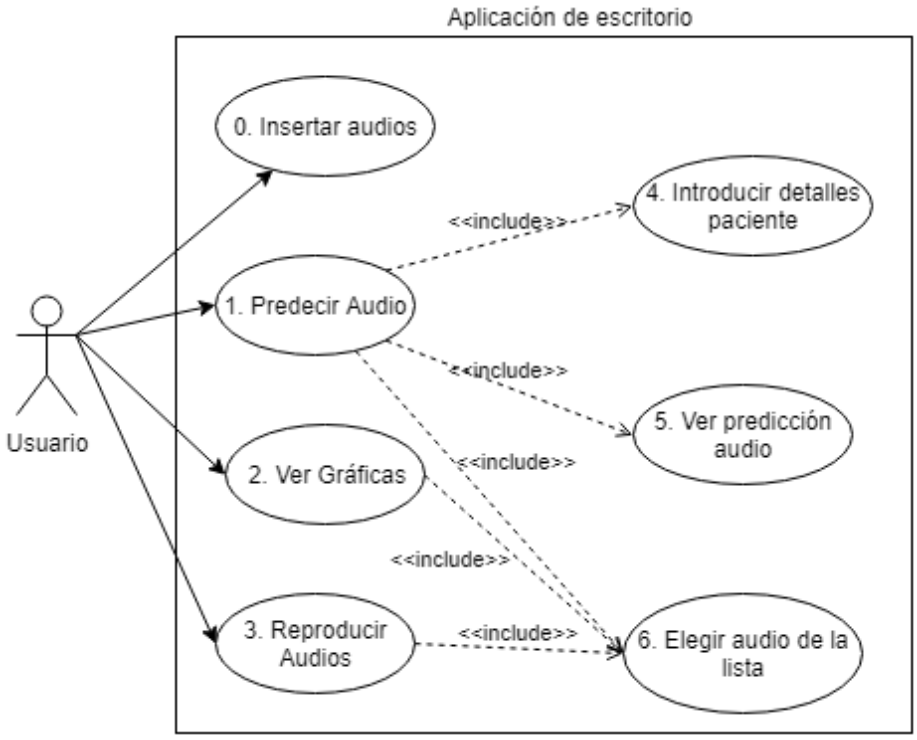

Figura B.1: Diagrama de casos de uso de la aplicación.

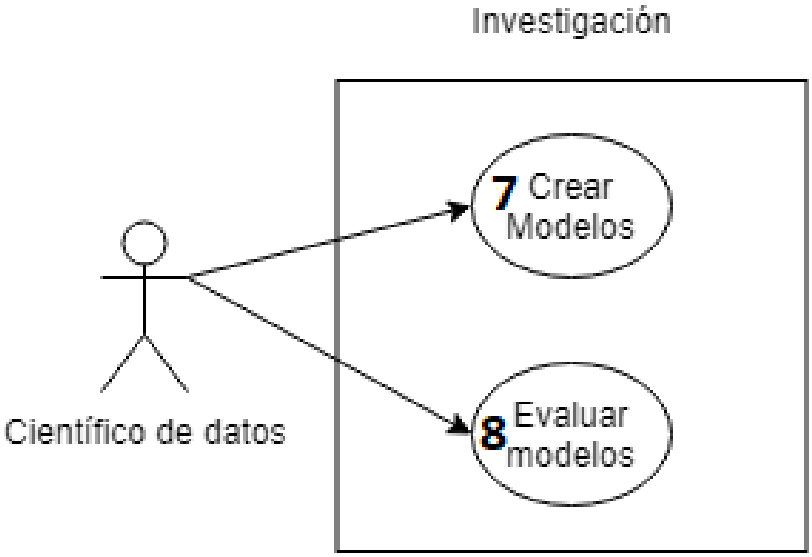

Figura B.2: Diagrama de casos de uso de la investigación.

Especificación de los casos de uso

Caso de uso 0: Insertar audios.		
Descripción	Permite al usuario cargar audios de su equipo a la aplicación.	
Requisitos	RF-4	
	RF-4.1	
	RF-4.2	
Precondiciones	Ninguna	
Secuencia normal	Paso	Acción
	1	El usuario pincha el botón 'Añadir'.
	2	Se despliega un menú de búsqueda de archivos.
	3	Se eligen los audios y se da a 'Abrir'
	4	Se cargan las imágenes dentro de la aplicación.
Postcondiciones	Los audios aparecen en el recuadro de audios cargados en la aplicación, listos para reproducir, predecir o mostrar gráficas.	
Excepciones	Audio no encontrado	
Importancia	Alta	
Urgencia	Alta	

Tabla B.1: Caso de uso 0: Insertar audios.

Caso de uso 1: Predecir Audio.		
Descripción	Permite al usuario obtener la predicción de parkinson de un audio cargado en la aplicación.	
Requisitos	RF-1	
	RF-1.1	
	RF-1.2	
	RF-1.3	
Precondiciones	Tener un audio cargado	
Secuencia normal	Paso	Acción
	1	El usuario elige un audio de la lista.
	2	El usuario introduce edad y sexo del paciente.
	3	El usuario pincha el botón predecir.
	4	Aparece la predicción en pantalla.
Postcondiciones	Aparece la predicción del audio en porcentaje en la pantalla.	
Excepciones	Audio no seleccionado. Edad no introducida. Edad no está en formato válido.	
Importancia	Alta	
Urgencia	Alta	

Tabla B.2: Caso de uso 1: Predecir Audio.

Caso de uso 2: Ver gráficas.		
Descripción	Permite al usuario obtener las gráficas de un audio.	
Requisitos	RF-2	
	RF-2.1	
	RF-2.2	
	RF-2.3	
	RF-2.4	
Precondiciones	Tener un audio cargado y con los detalles del paciente introducidos.	
Secuencia normal	Paso	Acción
	1	El usuario elige un audio de la lista.
	2	El usuario pincha el botón predecir.
	3	Aparecen las gráficas en pantalla.
	4	El usuario puede navegar o guardar las gráficas.
Postcondiciones	Aparece las gráficas de amplitud y espectrograma de frecuencias en la pantalla.	
Excepciones	Audio no seleccionado.	
Importancia	Alta	
Urgencia	Alta	

Tabla B.3: Caso de uso 2: Ver Gráficas.

Caso de uso 3: Reproducir audios.		
Descripción	Permite al usuario reproducir un audio.	
Requisitos	RF-3	
	RF-3.1	
	RF-3.2	
	RF-3.3	
	RF-3.4	
	RF-3.5	
	RF-3.6	
Precondiciones	Tener un audio cargado	
Secuencia normal	Paso	Acción
	1	El usuario elige un audio de la lista.
	2	El usuario pincha el botón play.
	3	El sonido comienza a sonar.
	4	El usuario puede pinchar el botón pause, volumen o rebobinar.
	5	El audio se para, cambia de volumen o vuelve a empezar.
Postcondiciones	Ninguna	
Excepciones	Audio no seleccionado.	
Importancia	Alta	
Urgencia	Alta	

Tabla B.4: Caso de uso 3: Reproducir audios.

Caso de uso 4: Introducir detalles del paciente		
Descripción	Permite al usuario introducir la edad y el sexo del paciente para su posterior predicción.	
Requisitos	RF-1	
	RF-1.2	
Precondiciones	Tener un audio cargado y seleccionado	
Secuencia normal	Paso	Acción
	1	El usuario pincha en hombre o mujer.
	2	El usuario introduce la edad (entero mayor que 0).
	3	El usuario puede dar a predecir.
Postcondiciones	La predicción se realiza de manera satisfactoria ya que los datos del paciente del audio se han leído correctamente.	
Excepciones	Edad no insertada. Edad en formato erróneo.	
Importancia	Alta	
Urgencia	Alta	

Tabla B.5: Caso de uso 4: Introducir detalles del paciente

Caso de uso 5: Ver predicción del audio		
Descripción	Permite al usuario ver el porcentaje de parkinson con el color indicado.	
Requisitos	RF-1	
	RF-1.3	
Precondiciones	Tener un audio cargado, seleccionado y con los datos del paciente introducidos	
Secuencia normal	Paso	Acción
	1	El usuario pincha en predicción.
	2	Se clasifica el audio.
	3	Se muestra la predicción del audio: porcentaje y si tiene parkinson o no.
Postcondiciones	Se muestra el texto del resultado de la predicción.	
Excepciones	Ninguna.	
Importancia	Alta	
Urgencia	Alta	

Tabla B.6: Caso de uso 5: Ver predicción del audio

Caso de uso 6: Elegir audio de la lista		
Descripción	Permite al usuario elegir un audio de los cargados para su procesamiento.	
Requisitos	RF-1.1	
	RF-2.1	
	RF-3.1	
	RF-4.2	
Precondiciones	Tener un audio cargado	
Secuencia normal	Paso	Acción
	1	El usuario elige pinchando un audio de la lista.
	2	El audio se remarca en azul.
Postcondiciones	Es posible la reproducción o predicción del audio seleccionado.	
Excepciones	Audio no seleccionado.	
Importancia	Alta	
Urgencia	Alta	

Tabla B.7: Caso de uso 6: Elegir audio de la lista

Caso de uso 7: Crear modelos		
Descripción	El científico de datos crea los modelos de predicción.	
Requisitos	RF-5	
	RF-5.1	
	RF-5.2	
	RF-5.3	
	RF-5.4	
	RF-5.5	
Precondiciones	Tener las características de los audios preextraídas	
Secuencia normal	Paso	Acción
	1	El científico de datos elige un modelo de python a entrenar.
	2	El científico de datos elige los datos con los que entrenar.
	3	El científico de datos elige el tipo de validación cruzada.
	4	El científico de datos elige como particionar los datos.
	5	El modelo se entrena.
Postcondiciones	Tenemos un modelo ya entrenado.	
Excepciones	Ninguna	
Importancia	Alta	
Urgencia	Alta	

Tabla B.8: Caso de uso 7: Crear modelos

Caso de uso 8: Evaluar modelos		
Descripción	Permite al científico de datos realizar una comparativa de resultados de los modelos.	
Requisitos	RF-6	
	RF-6.1	
	RF-6.2	
	RF-6.3	
Precondiciones	Haber entrenado los modelos	
Secuencia normal	Paso	Acción
	1	El científico de datos extrae las medidas de rendimiento de varios modelos.
	2	El científico de resume las medidas de rendimiento en gráficas o tablas.
	3	El científico de datos visualiza las medidas de rendimiento en gráficas o tablas.
Postcondiciones	Vemos los resultados de la evaluación de los modelos.	
Excepciones	Ninguna.	
Importancia	Alta	
Urgencia	Alta	

Tabla B.9: Caso de uso 8: Evaluar modelos

Apéndice C

Especificación de diseño

C.1. Introducción

En éste apéndice, se describirán cómo están implementados los datos de esta aplicación, cuales son los procedimientos más relevantes de la aplicación, y cómo se organizan los proyectos en paquetes.

C.2. Diseño de datos

En este sección, explicaremos como están organizados tanto el conjunto de audios obtenido, cómo son y como se organizan las características extraídas de los audios, y el diagrama de clases de la aplicación.

Conjunto de audios

El conjunto de audios es el descrito en [5]. También lo hemos descrito en la sección 5.3 de la memoria principal. Estos audios no se encuentran en el repositorio, ya que son de carácter privado. En [5] se describe como: "(...)incluye grabaciones de voz de 50 personas con PD y 50 controles sanos, 25 hombres y 25 mujeres en cada grupo. Todos los participantes son hablantes nativos de español colombiano. La edad de los hombres con PD varía de 33 a 77 años (media 62.2 $\pm$ 11.2), la edad de las mujeres con PD varía de 44 a 75 años (media 60.1 $\pm$ 7.8). Para el caso de los controles sanos, la edad de los hombres varía de 31 a 86 (media 61.2 $\pm$ 11.3) y la edad de las mujeres de 43 a 76 años (media 60.7 $\pm$ 7.7). Por lo tanto, la base de datos está bien equilibrada en términos de edad y género. Las grabaciones fueron capturadas en condiciones de control de ruido, en una cabina de prueba de sonido que

se construyó en la Clínica Noel, en Medellín, Colombia. Los registros se muestrearon a 44100 Hz con 16 bits de resolución, utilizando un micrófono omnidireccional dinámico (Shure, SM 63L) que se usa comúnmente para aplicaciones profesionales. Las grabaciones se capturaron utilizando una tarjeta de audio profesional de hasta 24 bits y de tal forma que admite hasta 96 Kbps de frecuencia de muestreo (M-Audio, Fast Track C400). Estos audios están en formato wav." (p.343). Tendremos 100 audios de una frase y 100 audios de cada una de las 5 palabras. En caso de las vocales, tenemos 300 audios para cada vocal, en vez de 100, ya que hay 3 audios para cada vocal de cada persona. Aunque en el total hacen un número de audios de 2100 audios, como hemos comentado en la memoria, no se puede entrenar con todos a la vez. Tenemos que entrenar los modelos con las características extraídas de cada grupo de 100 o 300 audios (si son vocales) por separado. Esto hace que entrenemos los modelos con bases de datos pequeñas y corran riesgo de sobreajustarse.

Destacar también, que tenemos un archivo tipo excel donde se indica: edad,sexo, UPDRS, HY y tiempo desde la detección de la enfermedad del paciente. Este archivo se encuentra en `doc/masRecursos/PCGITA_metadata.xlsx`.

Características extraídas

Como se ha comentado el la memoria hemos extraído diferentes características de los audios. Todas, se encuentran el el directorio o subdirectorios de `src/CaracterísticasExtraídas`. También, todas ellas están guardadas de manera serializada en formato tipo *numpy* con la herramienta *pickle*. Las características extraídas son las siguientes

- **Características Disvoice (primera fase):** se encuentran en `src/CaracterísticasExtraídas` y se extraen en el *notebook* `Extracción de características.ipynb` del directorio `src`. Están descritas más en detalle en la memoria en la sección 5.5.2 (Aspectos Relevantes - Primera Fase: atributos Disvoice - Modelado del discurso). A grandes rasgos, de los audios de la frase sacamos medidas de fonación, articulación y prosodia; de los audios de cada palabra sacamos medidas de fonación y articulación, y de los audios de cada vocal sacamos únicamente medidas de fonación. Hay un total de **18 conjuntos** diferentes de características. El tamaño de las matrices de características es el siguiente
  - Fonación: 100×30 y 300×30 (para vocales). Se sacan 29 medidas más la clase.

  - Articulación: 100×489. Se sacan 488 medidas más la clase.
  - Prosodia: 100×39. Se sacan 28 medidas más la clase.

- **Características MODIFICADAS Disvoice (segunda fase):** se encuentran en `src/ CaracterísticasExtraídas/EdadYSexo` y en `src/CaracterísticasExtraídas/DivisionSexo` y se extraen en `src/ Extracción de características MODIFICADAS Disvoice.ipynb`. Están descritas más en detalle en la memoria en la sección 5.6.1 (Aspectos Relevantes - Segunda Fase - Modelado del discurso). A grandes rasgos, primero se ha añadido los atributos edad y sexo a las anteriores características, por lo que tendremos otros 18 conjuntos de datos nuevos nuevos, cuya medida de las matrices es igual que la primera fase, pero teniendo 2 columnas más (2 características más). Por otro lado, se han dividido entre hombres y mujeres, por lo que tendremos otros 35 conjuntos de datos más, cada uno con la mitad de instancias que el original (tamaño: tendrán la mitad de filas que los de la fase 1, y tendrán 1 columna más: la edad). Por último, tenemos los conjuntos de características exactos a los de Orozco, solo los MFCC. Estos también están divididos por sexo. De estos últimos tenemos 24 conjuntos, cuyo tamaño es 50×35 (34 medidas más la clase). Por lo hque forman un total de **78 conjuntos** de características diferentes.

- **Características VGGish (tercera fase):** se encuentran en:

  - `src/CaracterísticasExtraídas/vggish/embeddings`
  - `src/CaracterísticasExtraídas/vggish/espectros`.

  Están descritas más en detalle en la memoria en la sección 5.7.1 (Aspectos Relevantes - Tercera Fase - Modelado del discurso), y en los *notebooks*. Se extraen para los tipos de audio de las 5 vocales y la frase los *embeddings*, que genera unas matrices de características de tamaño 100×256 para la frase y 300×256 para cada vocal. Esto es debido a que *VGGish* nos saca 128 *embeddings* para cada fragmento de 25ms de un audio, y nosotros, hemos decidido que para formar un único array de cada audio, hacer la media y desviación de los *embeddings*. Por otro lado, sacamos también los espectros de frecuencia del audio, con la herramienta del pre-procesado de VGGish. Los detalles se encuentran en la memoria, en el apartado recientemente indicado. El tamaño de las matrices de datos de los espectros es de 100×128 para la frase y 300×128 para cada vocal. Al igual que antes, es debido a que VGGish nos da 64 espectros de cada fragmento de 25ms, y hemos hecho la media y la desviación para tener un único array.

Diagrama de clases de la aplicación

En este apartado se comentara las clases realizadas por nosotros en el proyecto. Es decir, las clases que se han creado para ayudar a los experimentos y realizar la aplicación. Señalamos que, únicamente se analizaran las realizadas por nosotros, todas la estructura de *VGGish* no será analizada. De echo, por ejemplo, *VGGish2Keras* es un conjunto de módulos con funciones, que no contienen ninguna clase.

Clases de la etapa de experimentación

Cabe destacar que, en los experimentos con los clasificadores, se ha utilizado los *notebooks* de *Jupyter* para su realización. Por ello, en la etapa de realización de experimentos únicamente se realizaron 3 clases, sin ninguna relación entre ellas, las cuales únicamente se usaban en los *notebooks* tanto de extracción de características como en los experimentos con clasificadores.

De estas clases no se hará diagrama, ya que son clases independientes, y se explicarán en el apartado Manual del programador, ver capítulo D.3. De estas 3 clases, 2 se alojan en el directorio `src` y son: `extractorCcas.py` y experimenter.py. Las otra está en el directorio `src/vggish` y se corresponde con el nombre `extractor_ccas_vggish.py`. También tenemos diferentes módulos de carga de datos, estos no son clases, siguen la estructura tipo los cargadores de datos de *Scikit-Learn* (i.e load_iris[1]), funciones dentro de módulos que devuelven objetos de tipo *Bunch*

Clases de la aplicación

Para la aplicación se han creado las siguientes clases, alojadas en `src/demo`:

- `VentanaInicio` dentro de `main.py`: Clase que contiene toda la estructura gráfica de la ventada de la aplicación. Como suele pasar con los objetos de *tkinter*, no tiene funciones, únicamente se inicializan todos los elementos gráficos (*frames*, botones...), las variables necesarias para cada uno, y los mediadores. En esta clase, todos los elementos gráficos se guardarán como atributos (botones, *frames*...) para un acceso más fácil a ellos y, por tanto, una modificación más sencilla.

- `MediadorVentana` dentro de `MediadorVentana.py`: Clase que tiene como único atributo la ventana anterior. Contiene todos los métodos

[1] https://github.com/scikit-learn/scikit-learn/blob/7813f7efb/sklearn/datasets/base.py#L326

necesarios para la reproducción de los audios y muestra de detalles de la carga y reproducción de los mismos. **Encapsula la comunicación de los objetos gráficos de `VentanaInicio`**, en lo relativo a reproducción y carga de archivos. Debido a que comunica elementos, y los modifica, la mayoría de funciones no devuelve nada.

- `MediadorPrediccion` dentro de `MediadorPrediccion.py`: Tiene dos atributos, la ventana de la aplicación y un objeto tipo *FachadaPrediccion*. Las funciones de este mediador llaman a los métodos para predecir y mostrar gráficos de la fachada y se encarga de cambiar los elementos de la ventana. **Encapsula la comunicación de los objetos gráficos de `VentanaInicio`**, en lo relativo a la predicción y muestra de gráficos.

- `FachadaPrediccion` dentro del fichero `FachadaPrediccion.py` del paquete predicción: Tiene dos atributos, el modelo *keras VGGish* para la extracción de características de los audios y el modelo *MLP* con 10 neuronas para la predicción. Se encarga de interactuar con los módulos de *VGGish2Keras* para devolver los resultados de la predicción al mediador.

- Módulos propios de *VGGish* (*VGGish2Keras*): `mel_features.py`, `vggish_keras.py`, `vggish_params.py`. No son clases, son módulos Python que únicamente contienen métodos.

El diagrama de clases se puede ver en la Figura C.1.

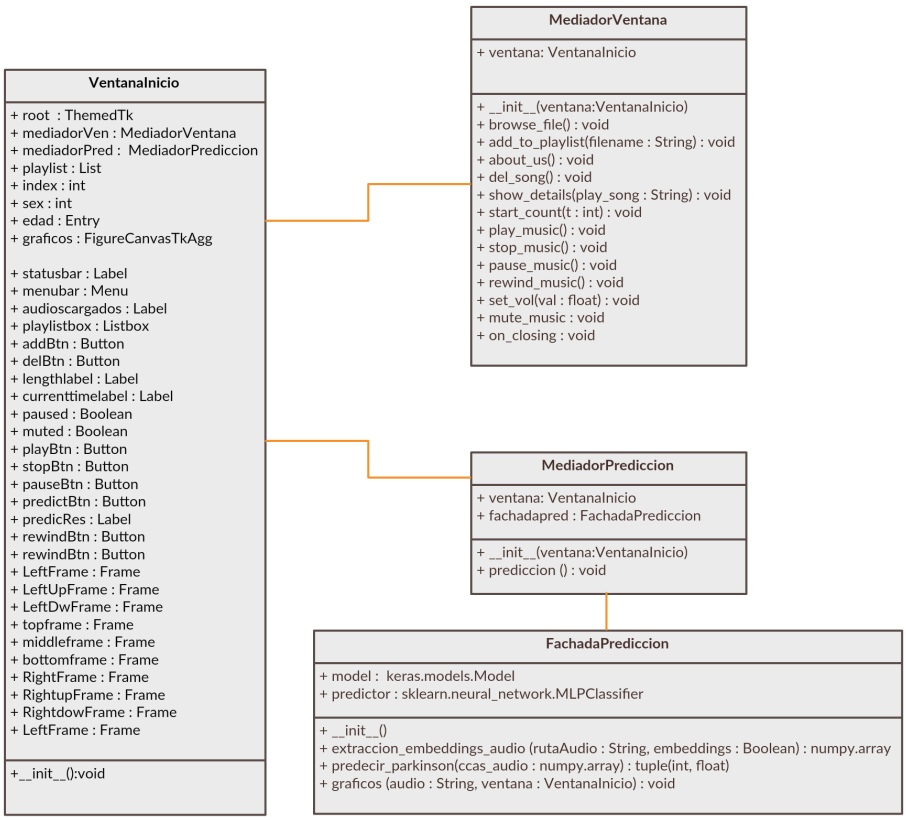

Figura C.1: Diagrama de clases de la aplicación

C.3. Diseño procedimental

En esta sección se mostrarán los diagramas de secuencia respectivos a 2 tareas principales de la aplicación: cargar audios (ver Figura C.2) y predecir de audios-muestra de gráficos (se ejecutan a la vez, cuando predices un audio, automáticamente se muestra sus gráficas, ver Figura C.3).

Diagramas de secuencia

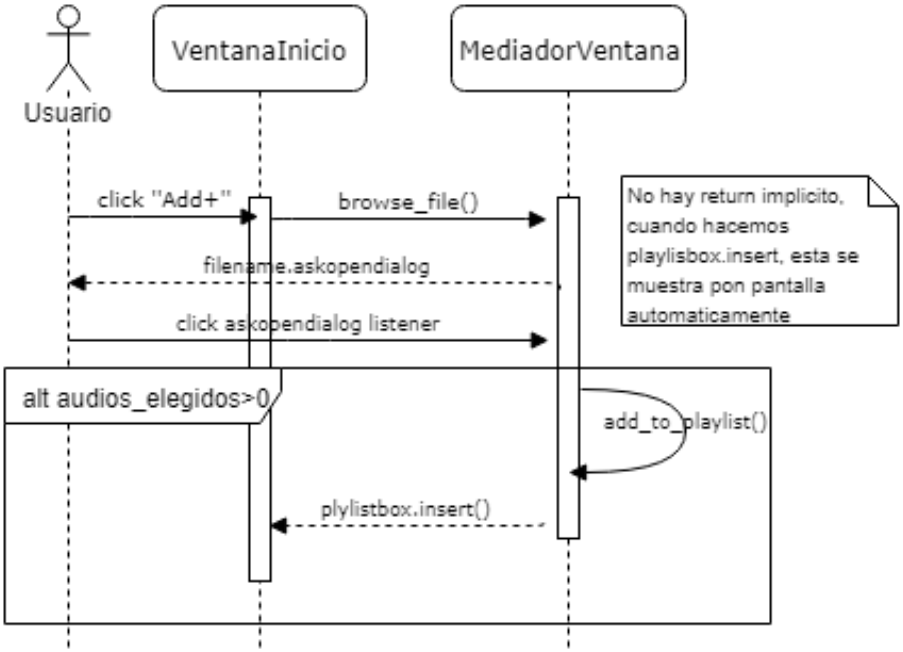

Figura C.2: Diagrama de secuencia para cargar un audio

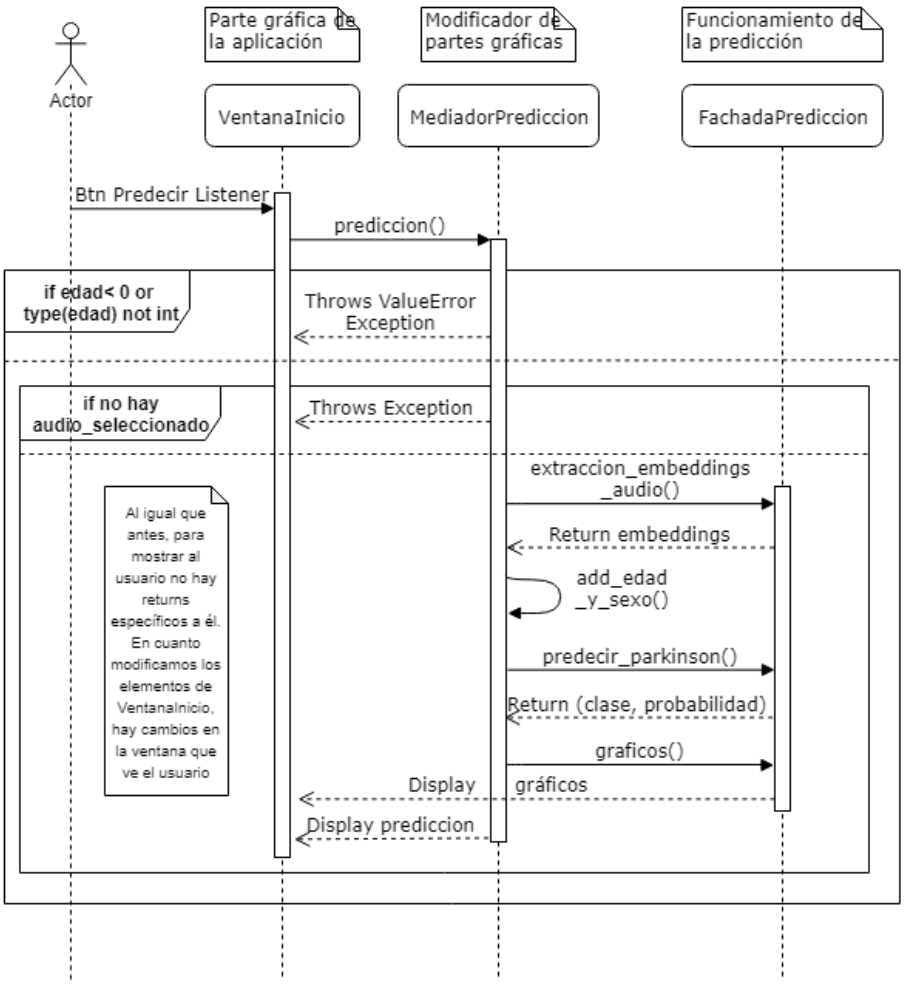

Figura C.3: Diagrama de secuencia para predicción y muestra de gráficos

C.4. Diseño arquitectónico

En este apartado se definirá como está **estructurado el código en paquetes**. Únicamente describiremos los paquetes que contienen código, los demás directorios y paquetes están definidos en la sección D.2. Adicionalmente, vamos a definir el diseño arquitectónico de la aplicación, la cual sigue los patrones de diseño Mediador y Fachada.

Patrón Mediador

El patrón de diseño Mediador es un patrón que encapsula la comunicación entre diferentes elementos del programa, normalmente en tiempo de ejecución. [9] Es nuestro caso, se ha utilizado para encapsular la comunicación entre diferentes elementos de la ventana de inicio. Es decir, encapsular la comunicación entre los diferentes *frames*, para que la clase `VentanaInicio` se ocupara únicamente de definir los elementos gráficos. Se han utilizado dos mediadores, uno que encapsula la comunicación entre elementos derivada de la carga y reproducción de datos (`MediadorVentana`) y otro que encapsula la comunicación entre elementos derivada de las tareas de predicción (`MediadorPrediccion`).

Patrón Fachada

El patrón *Facade*, o Fachada, proporciona una interfaz simple para un sistema de comportamiento más complejo. Este patrón reconoce que elementos del subsistema son los encargados de realizar una determinada labor, y los elementos por encima de la fachada únicamente se comunican con la fachada, no con todos los elementos del subsistema. [8]. En nuestro caso ha sido usado para encapsular el funcionamiento de los módulos *VGGish*. En un diagrama de clases de un patrón de fachada estándar, la fachada se comunicaría con las clases del subsistema. En nuestro caso, no se aprecia en el diagrama de clases la comunicación con el subsistema *VGGish* ya que, como hemos dicho antes, son una serie de módulos que contienen funciones y, por tanto, no son clases. Nuestra `FachadaPrediccion` se comunicará con los módulos: `mel_features.py, vggish_keras.py, vggish_params.py`.

Paquetes del proyecto

Paquetes del proyecto (ver Figura C.4):

- **src**: contiene todo el código del proyecto. En este paquete están los *notebooks* de extracción de características *Disvoice*, la clase extractora de características de *Disvoice*, los módulos de carga de datos, las clase del experimentador, los 6 *notebooks* de experimentos y las pruebas de los cargadores de datos.
    - **demo**: contiene el código de la aplicación del proyecto. En este directorio están las clases de la ventana de inicio, los mediadores y las pruebas.
        - **prediccion**: contiene el código necesario para la prediccion: *VGGish* (*VGGish2Keras*) y `FachadaPrediccion`.
- **Disvoice**: Contiene el proyecto *Disvoice* modificado por nosotros. Contiene todos los *scripts* necesarios para la extracción de características con esta herramienta. No analizaremos los subpaquetes en profundidad, ya que no es una herramienta propia. Sin embargo caben destacar 3 paquetes: `articulacion`, `phonation` y `prosody`. Estos contienen los *scripts* para extraer los 3 diferentes tipos de características.
- **VGGish**: Contiene los archivos del repositorio VGGish[2] y del repositorio VGGish2Keras[3]. También contiene el archivo resultante de la conversión del modelo *VGGish* a *Keras*: `vggish_weights.ckpt`. Por último, contiene el *notebook* de extracción de características de VGGish.

[2] https://github.com/tensorflow/models/tree/master/research/audioset
[3] https://github.com/antoinemrcr/vggish2Keras

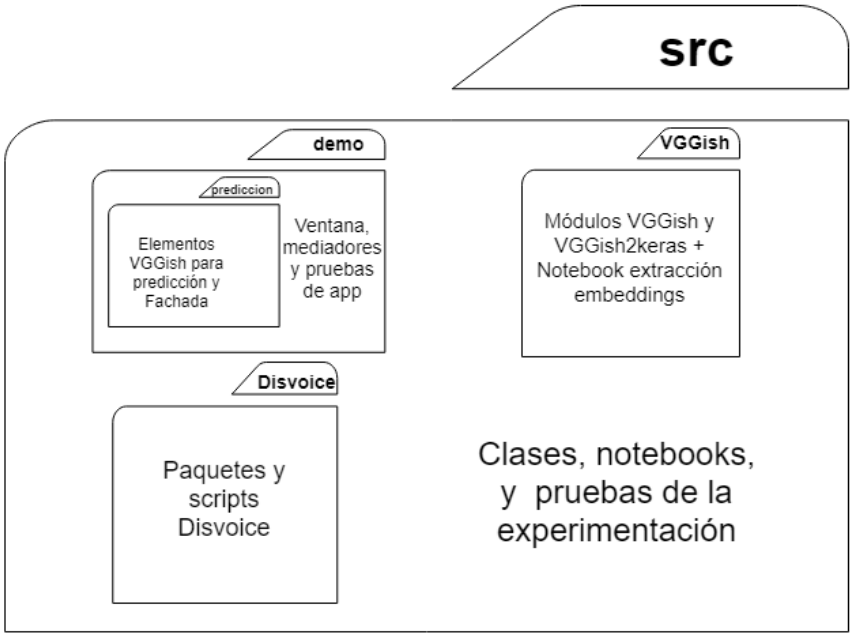

Figura C.4: Diagrama de paquetes del proyecto

Apéndice D

Documentación técnica de programación

D.1. Introducción

En esta sección se comentarán los aspectos más relevantes de la implementación del proyecto, el cual está alojado en un repositorio público de Github[1]. El objetivo es servir de manual para futuros desarrolladores, facilitando la labor de entendimiento, comprensión, modificación y extensión del código.

D.2. Estructura de directorios

El árbol de directorios del proyecto están definidos en la Figura D.1.

D.3. Manual del programador

Manual donde se va a explicar como son las diferentes clases y *notebooks* del proyecto, y con qué objetivo se han realizado. Este apartado servirá de ayuda y referencia a futuros desarrolladores que investiguen en este proyecto. Por ello nos disponemos a explicar que se ha realizado en cada fichero que contiene código del proyecto. Destacamos que el orden de la explicación de los ficheros, es el orden que hemos seguido en la creación de los mismos. Aunque, obviamente, aquí se explicará qué realiza cada elemento del proyecto

[1] https://github.com/AdrianArnaiz/TFG-Neurodegenerative-Disease-Detection

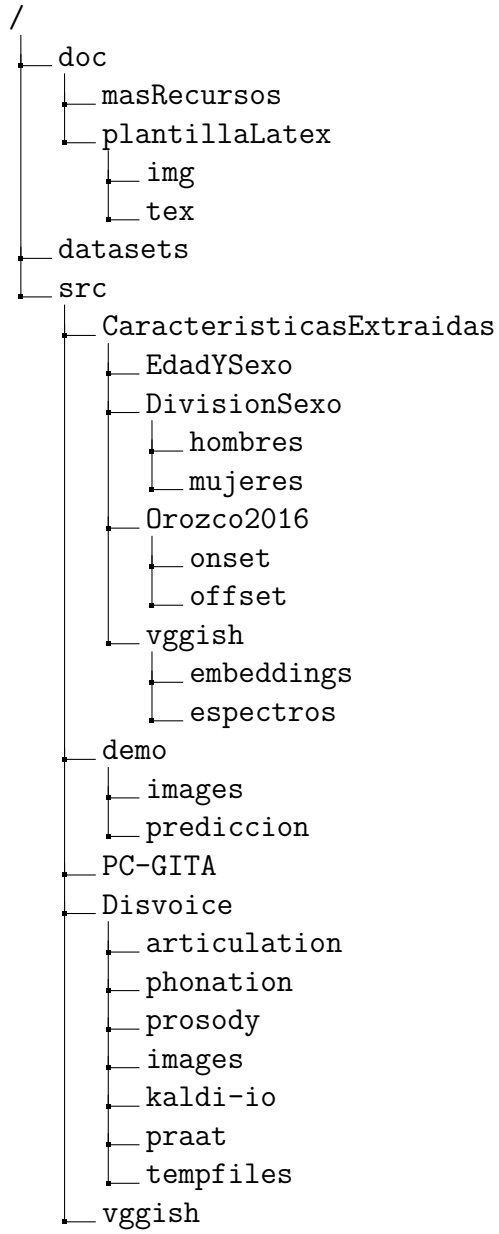

Figura D.1: Directorios del proyecto

para su futura comprensión y modificación, remarcamos que dentro de cada clase y cada *notebook*, hay documentación detallada de como funcionan. Se documenta, tanto en módulos *Pyton* como en *notebooks*, cada clase y cada método de una manera mucho más detallada (ver sección D.3).

Entorno del proyecto

Antes de realizar la explicación de lo que realiza cada elemento del proyecto, vamos a comentar el entorno en el que está realizado.

- **Ordenador**: Lenovo G50-80. Memoria 16GB RAM. Procesador Intel Core 17-5500U 2.4 Ghz. 64 bit.
- **Sistema operativo**: Microsoft Windows 10 [Versión 10.0.17134.829].
- **Python**: Versión 3.6.8 Anaconda Custom.
- **Anaconda**: Versión 4.6.8.
- Los demás requerimientos del proyecto se comentarán en la sección D.4.

Disvoice

Disvoice original[2] está implementado para su uso en Linux, y envuelve las funcionalidades de *Praat*, para darnos unos *scripts* que hacen más fácil la extracción de características de los audios. Sin embargo este proyecto se ha desarrollado en entorno Windows. A eso se le añade que *Disvoice* tiene una opción de tratamiento de datos para su visualización en interfaz gráfica a través de kaldi-io[3]. Esta biblioteca es de complicada instalación. Sin embargo, como a nosotros no nos interesaban las visualizaciones, decidimos no instalarla. Por ello se decidió hacer cambios en la biblioteca. Los cambios que realizamos fueron:

- Instalación de Microsoft Visual Studio c++ 2017 Build Tools (Arreglar error que daba únicamente en mi PC).
- Comentar líneas de código que utilizan *Kaldi* en estructura de Disvoice:
  - phonation.py lineas 71, 316, 330. Ejemplo: `from kaldi_io import write_mat, write_vec_flt`

[2] https://github.com/jcvasquezc/DisVoice
[3] https://github.com/vesis84/kaldi-io-for-python

    - Más líneas de código que utilizaban *kaldi* en glotal.py y articulation.py

- **Windows**: En praat_functions.py, cuando construye los comandos, cambiar `praat` por `../praat/praat.exe`. Esto es debido a que en *Windows*, `praat` es un ejecutable que se encuentra en la carpeta DisVoice/praat y nuestro script se ejecuta desde /Disvoice/phonation (y articulation o prosody). En cambio en *Linux* es un comando del sistema.

- El detalle específico de los cambios se encuentra en el commit de los cambios[4].

> `src/Ejemplo ejecución scripts Disvoice.ipynb`, contiene los detalles del cambio junto con un ejemplo detallado de su utilización: cómo ejecutar los *scripts*, qué hace cada uno, etc. En este *notebook* también se explican las diferentes opciones de ejecución de la biblioteca de *scripts*, junto con la información del tipo de características que saca cada *script* de cada audio. Por ejemplo, se explica que serie de *scripts* pueden procesar audios en bruto, o en cual debemos eliminar los silencios para que no den medidas erróneas.

La clase `src/extractorCcas.py` es una abstracción de la biblioteca *Disvoice*, la cual internamente utiliza los *scripts* de Disvoice, junto con más operaciones, para extraer las características. Esta clase se utiliza para extraer las características de los audios en los 2 siguientes *notebooks*:

- `src/Extracción de características.ipynb`
- `src/Extracción de características MODIFICADAS Disvoice.ipynb`

Clases y notebooks

Comentaremos que se realiza en cada clase y *notebook* del proyecto, analizaremos sólo los elementos utilizados en la fase de experimentación, ya que, los elementos utilizados en la fase de desarrollo de la aplicación, están explicados en la sección C.2. Resaltamos un aspecto comentado anteriormente: **todos ellos están documentados al detalle en su documentación interna**.

[4] https://github.com/AdrianArnaiz/DisVoice/commit/330c67fc3b7e9ab59bf4f06e5df3c79f8ff18165

- `src/Instalación_de_librerías.ipynb`: En este *notebook* realizamos las primeras instalaciones necesarias para los primeros pasos de la codificación del proyecto. Para ello instalamos librerías como *Praat*, y también se instala como submódulo el repositorio *Disvoice* modificado por nosotros, recientemente comentado.

- `src/Preprocesamiento de audios.ipynb`: *notebook* en el que eliminamos el silencio inicial y final de los audios para que no den medidas erróneas en el *script* `prosody.py` de *Disvoice*. Para ello creamos una función que detecta los segmentos silenciosos al principio del audio y los eliminamos.

- `src/extractorCcas.py`: modulo que contiene la clase ExtractorCaracteristicas. Esta es la clase encargada de la extracción de características de los audios con la herramienta *Disvoice*. Para ello tiene los métodos necesarios para extraer las características deseadas de los audios deseados con los *scripts* de *Disvoice*, añadir la clase a las características, añadir los atributos extra necesarios a esas características e identificar y eliminar audios que contengan *NaN*.

- `src/extractor_ccas_vggish.py`: Módulo que contiene la clase Extractor_Caracteristicas_Vggish. Clase encargada de abstraer el funcionamiento de *VGGish*. Igual que la anterior pero para la extracción de *embeddings* y espectros de los audios.

- `src/Ejemplo ejecución scripts Disvoice.ipynb`: contiene los detalles del cambio junto con un ejemplo detallado de su utilización: cómo ejecutar los *scripts*, qué hace cada uno, etc. En este *notebook* también se explican las diferentes opciones de ejecución de la biblioteca de *scripts*, junto con la información del tipo de características que saca cada *script* de cada audio. Por ejemplo, se explica que serie de *scripts* pueden procesar audios en bruto, o en cual debemos eliminar los silencios para que no den medidas erróneas. Es una especie de manual de uso de *Disvoice*.

- `src/vggish/Prueba_VGGish.ipynb`: *notebook* que tiene un ejemplo del funcionamiento detallado y explicado de *VGGish*, tras haberlo convertido a un modelo *Keras* con *VGGish2Keras*. Se explican los diferentes pasos de uso, con las explicaciones y visualizaciones de los pasos dados. Al igual que el anterior, una especie de manual de uso de *VGGish*.

- `src/Extracción de características.ipynb`: *notebook* en el que extraemos las características originales de *Disvoice* para cada tipo de audio. En el *notebook* se describe todo el proceso. Desde cómo se hace, cómo lo estructuramos hasta las características y tamaños concretos de los conjuntos de datos resultantes y su 'limpieza'.

- `src/vggish/Extracción características VGGish.ipynb`: *notebook* en el que extraemos las características de *embeddings* y espectros de los audios con VGGish. Se describe detalladamente el proceso, las características extraídas,etc.

- **Módulos de carga de datos**: `src/carga.py`: describiremos los 8 diferentes cargadores de datos juntos, ya que son de idéntico funcionamiento. Cada uno de ellos es un módulo de *Python* que contiene las diferentes funciones de carga de datos. Todas las funciones funcionan de la misma forma, solo cambia el conjunto de datos. Son funciones que simulan en estilo de los *loaders* de *Scikit-Learn* (como load_iris en base.py[5]). Hay una función para cada conjunto de datos diferente, 108 en total. Están repartidas por módulos según del tipo que es cada característica (*Disvoice, Disvoice+Edad+Sexo, DisvoiceHombres, DisvoiceMujeres, VGGishEmbeddings y VGGish espectros*). Todas estas funciones devuelven un objeto tipo

- `src/Extracción de características MODIFICADAS Disvoice.ipynb`: en este *notebook* se extraen las características de Disvoice de la segunda fase. Se añaden edad y sexo a todos los conjuntos de datos. También se separan por sexos. Se utilizan funciones de la clase ExtractorCaracteristicas para añadir estos atributos extra. *Bunch* con 2 elementos: *data* (contiene los datos del conjunto de datos, sin targets) y *target* (la clase de los datos anteriores). Todas las funciones las podemos ver listadas en la Figura reffig:PruebaLoaders de la sección donde definimos las pruebas, D.5.

- Experimentos con clasificadores:

  - `src/1os Experimentos clasificadores.ipynb`: Se realiza el entrenamiento y experimentación con múltiples clasificadores y procesos, entrenados con los datos extraídos con *Disvoice* por defecto. Se corresponde con los experimentos de la primera fase.

[5] https://github.com/scikit-learn/scikit-learn/blob/7813f7efb/sklearn/datasets/base.py#L326

- `src/2os Experimentos A - Disvoice + Edad y Sexo.ipynb`: Se realiza el entrenamiento y experimentación con múltiples clasificadores y procesos, entrenados con los datos extraídos con *Disvoice*, y añadidos la edad y el sexo del paciente en cada instancia. Se corresponde con los experimentos de la segunda fase.
- `src/2os Experimentos B - Disvoice Mujeres.ipynb`: Se realiza el entrenamiento y experimentación con múltiples clasificadores y procesos, entrenados con los datos extraídos con *Disvoice*, únicamente con las instancias de las mujeres. Se corresponde con los experimentos de la segunda fase.
- `src/2os Experimentos C - Disvoice Hombres.ipynb`: Se realiza el entrenamiento y experimentación con múltiples clasificadores y procesos, entrenados con los datos extraídos con *Disvoice* por defecto, únicamente con las instancias de los hombres. Se corresponde con los experimentos de la segunda fase.
- `src/3os Experimentos A - VGGish Embeddings.ipynb`: Se realiza el entrenamiento y experimentación con múltiples clasificadores y procesos, entrenados con los datos extraídos con *VGGish*. Concretamente, los datos son los *embeddings* extraídos de cada audio. Se corresponde con los experimentos de la tercera fase.
- `src/3os Experimentos B - VGGish Espectros.ipynb`: Se realiza el entrenamiento y experimentación con múltiples clasificadores y procesos, entrenados con los datos extraídos con *VGGish*. Se corresponde con los experimentos de la tercera fase. Concretamente, los datos son los espectros extraídos de cada audio, extraídos con la función de pre-procesamiento de *VGGish*.

- `src/Proyeccion_ccas_Disvoice.ipynb`: Se realizan diferentes visualizaciones de los datos. Para ello es necesario reducir su dimensionalidad para lo que hemos utilizado diferentes técnicas: PCA, kernel PCA, LDA y t-SNE.
- `src/ROC - best.ipynb`: Se ha realizado la curva ROC del mejor experimento conseguido. Ver Figura D.2.

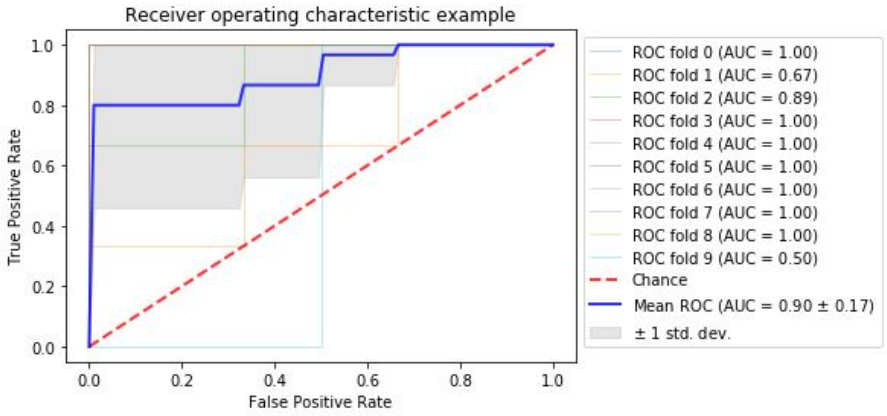

Figura D.2: Curva ROC del mejor experimento

- `src/Guardar modelo para Demo.ipynb`: Se ha realizado el que hemos considerado el mejor modelo para la aplicación. Para ello, hemos entrenado el modelo MLP (10 capas) con el conjunto de datos *embeddings* extraídos de la frase. A continuación, hemos guardado el modelo de manera serializada con la biblioteca *Pickle*, para poder utilizarlo en nuestra aplicación.

- `src/PruebasCargadoresDatos.py`: pruebas unitarias de los módulos de carga de datos. Ver sección D.5.

- `src/install_vggish_win10.cmd`: Pasos que hemos realizado nosotros para la instalación de *VGGish*. En nuestro proyecto ya está instalado, se encuentra en la carpeta `src/vggish` y sólo hace falta instalar los requerimientos. Sin embargo, este archivo **son los pasos que nosotros hemos realizado para su instalación**. Ver sección D.4.

- `src/vggish/vggish_weights.ckpt`: archivo resultante de la conversión del modelo *VGGish* a *Keras* con la biblioteca *VGGish2Keras*. Son los pesos extraídos de la red pre-entrenada *VGGish* de tipo *Tensorflow*, y ya aptos para cargar en un modelo tipo Keras.

VGGish

VGGish[6] es una modelo implementado con *Tensorflow*. Sin embargo, para facilitar la ejecución nuestros experimentos, decidimos investigar la conversión del modelo *VGGish* tipo *Tensorflow* a un modelo tipo *Keras*.

[6] https://github.com/tensorflow/models/tree/master/research/audioset

Para ello, hemos utilizado la biblioteca VGGish2Keras[7]. Esta biblioteca se encarga de extraer los pesos del modelo pre-entrenado *VGGish*, para crear un archivo de pesos adecuado para su carga a un modelo *Keras*.

- `src/vggish/vggish_weights.ckpt`: archivo de pesos resultante de la ejecución de la conversión.
- `src/vggish/convert_ckpt.py`: *script* de conversión.

VGGish2Keras, tiene una serie de funciones que crean la arquitectura del modelo *VGGish* en *Keras* y permiten cargar el archivo de pesos en ese modelo, teniendo como resultado el modelo *VGGish* pre-entrenado en tipo *Keras*.

Los pasos para realizar esta conversión se ven en la sección D.4.

Documentación de código

Un aspecto reseñable es que todo el código del programa ha sido minuciosamente comentado para su mejor comprensión. Los *notebooks*, se han comentado mediante celdas tipo *markdown* (además de comentarios de código). En estas celdas se ha explicado multitud de aspectos, desde todo el proceso que hemos seguido, el funcionamiento de los métodos y clases usadas, aspectos relevantes, análisis de resultados, etc. Para las clases y módulos (código escrito en ficheros `.py`), se ha realizado con la estructura que podemos ver en la Figura D.3.

Se ha realizado la documentación a:

- Módulos cargadores de datos (8 en total).
- Extractores de características (2 en total).
- Experimenters (1).
- Pruebas (2).
- Clases de la demo (4).

[7] https://github.com/antoinemrcr/vggish2Keras

```
class ClassName:
    '''
    definicion
    Atributos
    ----------
    atrib1 : tipo
        descripcion
    '''

    def methodName(p1):
        '''
        definicion
        Parametros
        ----------
                    pram1 : tipo
                        descripcion

        Return
        ------
                    r1 : tipo
                        descripcion
    '''
.   .   .
```

Figura D.3: Documentación en *Python*

Aspectos relevantes

Me gustaría destacar un fallo relativo a la implementación de un experimento. Este error se tuvo tras las dos primeras fases de experimentos, y una vez arreglado se repitieron todos los experimentos y análisis de resultados.

> Este error consistía en que no hacíamos validación cruzada anidada en la búsqueda de parámetros. Entrenábamos únicamente el objeto tipo `GridSearchCV`, devolviendo su rendimiento. Esto hacía que estuviéramos teniendo un bias optimista, que causaba que el rendimiento de los clasificadores era mejor que el real.

En resumen, el cambio realizado fue cambiar el entrenamiento del `GridSearchCV`, ver Figura D.4. Antes, le entrenábamos directamente con

su función fit. Ahora, le insertamos en la función `cross_val_score`, que detecta que es un objeto de búsqueda de parámetro y realiza la validación cruzada anidada.

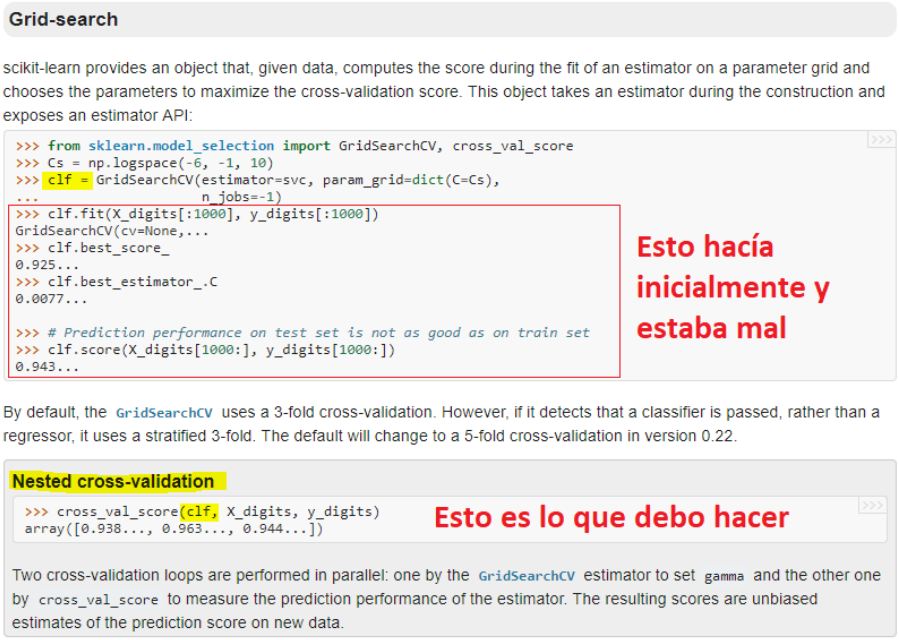

Figura D.4: Fallo nested cross validation

Los cambios realizados están explicados en la tarea pertinente de Github[8]. También, se muestra como, si no realizamos validación cruzada anidada, hay un sesgo optimista en el rendimiento del clasificador en la documentación de Scikit-Learn[9], ver Figura D.5.

[8] https://github.com/AdrianArnaiz/TFG-Neurodegenerative-Disease-Detection/issues/34

[9] https://scikit-learn.org/stable/auto_examples/model_selection/plot_nested_cross_validation_iris.html

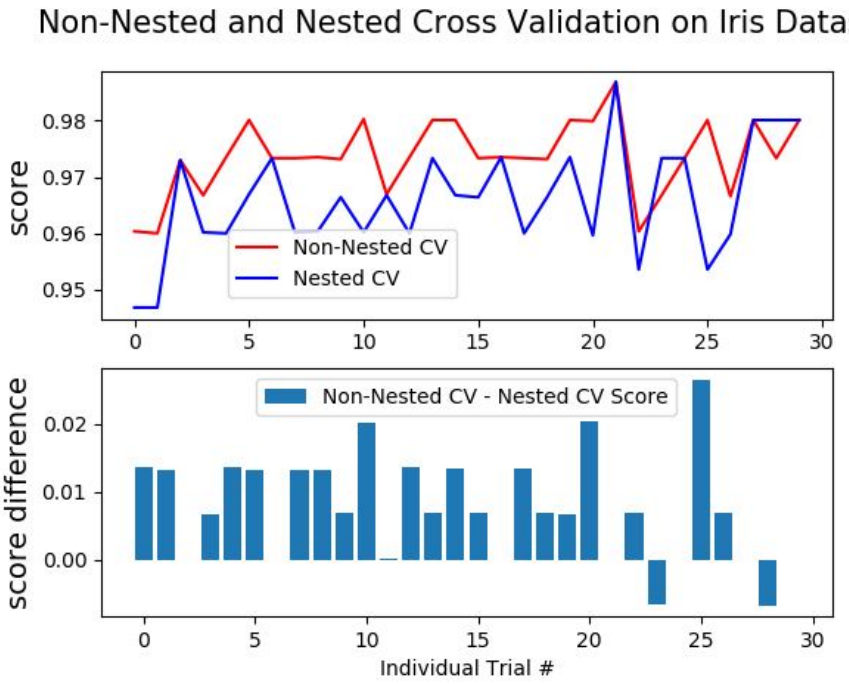

Figura D.5: CV anidada VS CV no anidada

D.4. Compilación, instalación y ejecución del proyecto

En esta sección explicaremos como instalar las diferentes dependencias del proyecto. Como hemos venido comentando, en la etapa de experimentación ha habido dos herramientas principales: Disvoice y VGGish. Aquí comentaremos como se realizó su instalación.

Repositorio general

1. Clonar repositorio.

   git clone https://github.com/AdrianArnaiz/TFG-Neurodegenerative-Disease-Detection.git

2. Inicializar submódulo Disvoice.

   git submodule update –init

3. Instalar dependencias. Todas las dependencias del repositorio quedan satisfechas si instalamos los requerimientos definidos en:

   - `src/Disvoice/requirements.txt`
   - `src/demo/requirements.txt`

   Podemos instalarlo de la siguiente forma:

   pip install -r src/Disvoice/requirements.txt && pip install -r src/demo/requirements.txt

Disvoice

Para instalar *Disvoice*, primero habrá que inicializar los submódulos de la manera definida en el anterior apartado. Posteriormente instalar las dependencias de *Disvoice*. Se podrá hacer de dos distintas maneras, estando en el directorio de *Disvoice*:

- pip install -r requirements.txt
- python install.py

Como ya hemos comentado, no se trata de *Disvoice* original, sino de *Disvoice* modificado por mí. Las únicas diferencias son su adaptación a *Windows* y los comentarios para que no se ejecute la biblioteca *kaldi-io*.

VGGish

Actualmente en el repositorio ya está listo para usar con *Keras*, es decir, ya está realizada la conversión del modelo *VGGish* a *Keras* con el archivo de pesos del modelo resultante: `vggish_weigths.ckpt`. Estas funciones de *VGGish* funcionarán siempre cuando estén instalados los requerimientos de la biblioteca *VGGish* original (ver en README[10] de VGGish). Estos requerimientos se satisfacen si instalamos las dependencias establecidas en *requirements.txt* de la aplicación. Sin embargo, vamos a comentar el proceso de instalación y conversión de la la biblioteca *VGGishKeras*.

`src/src/install_vggish_win10.cmd` contiene los pasos que dimos para la instalación y conversión. Este fichero podrá ser ejecutado en un proyecto cualquiera que no tenga *VGGish* y se realizará la instalación y conversión del modelo *VGGish* a *Keras*.

[10] https://github.com/tensorflow/models/blob/master/research/audioset/vggish/README.md

- Clonar el proyecto models de Tensorflow[11].
- Extraer el proyecto VGGish, que es un subproyecto dentro de *models*.
- Instalar requerimientos de *VGGish*.
- Descargar archivos de *checkpoint* de modelo y de los parámetros. Son dos archivos de gran peso, necesarios para las funciones de *VGGish2Keras*.
- Clonar el repositorio VGGish2Keras[12] y moverlo al mismo directorio que hemos puesto *VGGish*
- Ejecutar conversión y prueba de conversión correcta.

Aplicación

Para instalar la aplicación hay que seguir los pasos detallados en el archivo:

- `src/demo/README.md`

Como pre-condiciones, deberá estar instalado Python, mínimo 3.6.8 y Anaconda. Desde la carpeta `src/demo`:

- Descargar archivo de pesos del modelo *VGGish* y alojarlo en la carpeta `prediccion`. El fichero `vggish_weights.ckpt` se descargará desde un enlace de Mega[13].
- Ejecutar install.cmd. Este archivo contiene lo siguiente:

```
conda create -n myNewEnv python==3.6.8
conda activate myNewEnv
pip install -r requirements.txt
```

Para ejecutar la aplicación:

```
python main.py
```

[11] https://github.com/tensorflow/models.git
[12] https://github.com/antoinemrcr/vggish2Keras
[13] https://mega.nz/#!fRRFSKrT!0EMBqtYjogQrSQgudWifOAXm_A5Yx9UnX5Qk_Enanuk

D.5. Pruebas del sistema

Únicamente se han realizado pruebas de los *loaders*, cargadores de datos, y de la fachada de predicción. Los demás elementos de la investigación, están enfocados a *data mining*. Tenemos multitud de extracciones de características con herramientas externas (*Disvoice* y *VGGish*), que no podemos probar. No conocemos las tripas de su funcionamiento. Además. no conocemos que salidas nos va a dar un audio para poder comprobarlas. Destacamos que en el proceso de minería de datos, es muy difícil realizar pruebas unitarias. En todos los experimentos realizados, no tenemos manera de saber el resultado final. Además, muchos de ellos son estocásticos, por lo que en cada ejecución tienen una salida diferente.

La dificultad de pruebas unitarias en la minería de datos se explica, por la comunidad de científicos de datos Dominio[14], de la siguiente manera: creemos que el término "prueba unitaria" no es aplicable a todos los tipos de trabajo de minería de datos. Esto se debe a que probar implica que hay una respuesta correcta de lo que está probando: se debe saber de manera previa el resultado. No podemos saber la respuesta correcta cuando comenzamos a trabajar en un nuevo modelo o análisis. Dada la naturaleza de la investigación como tal, compleja y sin "respuestas correctas" conocidas por adelantado, se cree que la mejor manera de lograr estos objetivos es permitir la inspección visual y la comparación de resultados por parte de personas, las cuales pueden hacer análisis estadísticos, llevando a cabo la **validación de calidad de los modelos**. [3]. Como dijo Einstein: "*If we knew what it was we were doing, it would not be called research, would it?*".

Respecto a la aplicación, la clase `VentanaInicio` no tiene ningún método, solo inicializa y sostiene los elementos gráficos de *tkinter*. Además, no tenemos una manera sencilla de ejecutar pruebas automáticas sobre la interfaz *tkinter* (como podría utilizarse *Selenium* para una página web). Las clases mediadoras tienen métodos pero ninguno de ellos devuelve nada, no hay *returns*, solo se encargan de modificar partes gráficas de la ventana en ejecución. Por ello, de esta parte hemos probado únicamente la fachada de predicción.

Para ambas pruebas (cargadores de datos y fachada), se han utilizado clases que heredan de `unittest.TestCase`, las cuales tienen métodos de prueba.

Para los cargadores de datos, la prueba realizada ha sido el siguiente: comparar el archivo directamente cargado con la biblioteca *numpy*, con los

[14] https://blog.dominodatalab.com

datos cargados desde el cargador de datos. Comprobamos que el archivo *numpy* que nos devuelve la función es igual al archivo *numpy* que cargamos nosotros directamente y que sabemos que es el correcto (así se comprueban intrínsecamente valores y dimensiones). Esta prueba hace cargas de dato e importaciones dinámicas. Los resultados se pueden ver Figura D.6.

Para la fachada, la prueba ha sido la siguiente: probar los métodos. Como son de predicción, los métodos se corresponden con la extracción y predicción. Hemos probado que los datos extraídos con la función de extracción son los correctos. Para ello hemos extraído los datos de un audio cuyos datos ya teníamos guardados y hemos comprobado que eran iguales. También sabíamos la clase que se predecía para ese audio y su probabilidad, por lo que también se ha comprobado que se devolvían los valores correctos. Los resultados se pueden ver Figura D.7.

```
cargaDatosHombres
./CaracteristicasExtraidas/DivisionSexo/hombres
✓ - load_art_rt() OK
✓ - load_art_rt_ofset() OK
✓ - load_art_rt_onset() OK
✓ - load_art_w_atleta() OK
✓ - load_art_w_atleta_ofset() OK
✓ - load_art_w_atleta_onset() OK
✓ - load_art_w_braso() OK
✓ - load_art_w_braso_ofset() OK
✓ - load_art_w_braso_onset() OK
✓ - load_art_w_campana() OK
✓ - load_art_w_campana_ofset() OK
✓ - load_art_w_campana_onset() OK
✓ - load_art_w_gato() OK
✓ - load_art_w_gato_ofset() OK
✓ - load_art_w_gato_onset() OK
✓ - load_art_w_petaka() OK
✓ - load_art_w_petaka_ofset() OK
✓ - load_art_w_petaka_onset() OK
✓ - load_fon_rt() OK
✓ - load_fon_v_A() OK
✓ - load_fon_v_E() OK
✓ - load_fon_v_I() OK
✓ - load_fon_v_O() OK
✓ - load_fon_v_U() OK
✓ - load_fon_w_atleta() OK
✓ - load_fon_w_braso() OK
✓ - load_fon_w_campana() OK
✓ - load_fon_w_gato() OK
✓ - load_fon_w_petaka() OK
✓ - load_prs_rt() OK
-------------
cargaDatosMujeres
./CaracteristicasExtraidas/DivisionSexo/mujeres/
✓ - load_art_rt() OK
✓ - load_art_rt_ofset() OK
✓ - load_art_rt_onset() OK
✓ - load_art_w_atleta() OK
✓ - load_art_w_atleta_ofset() OK
✓ - load_art_w_atleta_onset() OK
✓ - load_art_w_braso() OK
✓ - load_art_w_braso_ofset() OK
✓ - load_art_w_braso_onset() OK
✓ - load_art_w_campana() OK
✓ - load_art_w_campana_ofset() OK
✓ - load_art_w_campana_onset() OK
✓ - load_art_w_gato() OK
✓ - load_art_w_gato_ofset() OK
✓ - load_art_w_gato_onset() OK
✓ - load_art_w_petaka() OK
✓ - load_art_w_petaka_ofset() OK
✓ - load_art_w_petaka_onset() OK
✓ - load_fon_rt() OK
✓ - load_fon_v_A() OK
✓ - load_fon_v_E() OK
✓ - load_fon_v_I() OK
✓ - load_fon_v_O() OK
✓ - load_fon_v_U() OK
✓ - load_fon_w_atleta() OK
✓ - load_fon_w_braso() OK
✓ - load_fon_w_campana() OK
✓ - load_fon_w_gato() OK
✓ - load_fon_w_petaka() OK
✓ - load_prs_rt() OK

cargaDatos
./CaracteristicasExtraidas/
✓ - load_art_rt() OK
✓ - load_art_w_atleta() OK
✓ - load_art_w_braso() OK
✓ - load_art_w_campana() OK
✓ - load_art_w_gato() OK
✓ - load_art_w_petaka() OK
✓ - load_fon_rt() OK
✓ - load_fon_v_A() OK
✓ - load_fon_v_E() OK
✓ - load_fon_v_I() OK
✓ - load_fon_v_O() OK
✓ - load_fon_v_U() OK
✓ - load_fon_w_atleta() OK
✓ - load_fon_w_braso() OK
✓ - load_fon_w_campana() OK
✓ - load_fon_w_gato() OK
✓ - load_fon_w_petaka() OK
✓ - load_prs_rt() OK
-------------
cargaDatosEdSx
./CaracteristicasExtraidas/EdadYSexo/
✓ - load_art_rt() OK
✓ - load_art_w_atleta() OK
✓ - load_art_w_braso() OK
✓ - load_art_w_campana() OK
✓ - load_art_w_gato() OK
✓ - load_art_w_petaka() OK
✓ - load_fon_rt() OK
✓ - load_fon_v_A() OK
✓ - load_fon_v_E() OK
✓ - load_fon_v_I() OK
✓ - load_fon_v_O() OK
✓ - load_fon_v_U() OK
✓ - load_fon_w_atleta() OK
✓ - load_fon_w_braso() OK
✓ - load_fon_w_campana() OK
✓ - load_fon_w_gato() OK
✓ - load_fon_w_petaka() OK
✓ - load_prs_rt() OK
-------------
cargaDatosVggishEmbeddings
./CaracteristicasExtraidas/vggish/embbedings/
✓ - load_vggish_embed_rt() OK
✓ - load_vggish_embed_v_A() OK
✓ - load_vggish_embed_v_E() OK
✓ - load_vggish_embed_v_I() OK
✓ - load_vggish_embed_v_O() OK
✓ - load_vggish_embed_v_U() OK
-------------
cargaDatosVggishEspectros
./CaracteristicasExtraidas/vggish/espectros/
✓ - load_vggish_espec_rt() OK
✓ - load_vggish_espec_v_A() OK
✓ - load_vggish_espec_v_E() OK
✓ - load_vggish_espec_v_I() OK
✓ - load_vggish_espec_v_O() OK
✓ - load_vggish_espec_v_U() OK
-------------
{'cargaDatos': True, 'cargaDatosEdSx': True,
'cargaDatosHombres': True, 'cargaDatosMujeres': True,
'cargaDatosVggishEmbeddings': True,
'cargaDatosVggishEspectros': True}
```

Figura D.6: Prueba de funciones de carga de datos

```
In [3]: runfile('C:/Users/usuario/Desktop/Ingenieria/TFG/TFG-Neurodegenerative-
Disease-Detection/src/demo/PruebaFachada.py', wdir='C:/Users/usuario/Desktop/
Ingenieria/TFG/TFG-Neurodegenerative-Disease-Detection/src/demo')
Reloaded modules: prediccion.FachadaPrediccion, prediccion.vggish_input,
prediccion.mel_features, prediccion.vggish_params, prediccion.vggish_keras
...
----------------------------------------------------------------------
Ran 3 tests in 8.500s

OK
```

Figura D.7: Prueba de la fachada de la aplicación

Apéndice E

Documentación de usuario

E.1. Introducción

En esta sección explicaremos lo necesario para que lo usuarios puedan instalar y utilizar la aplicación.

E.2. Requisitos de usuarios

El usuario, para poder utilizar la aplicación, deberá tener instalado Python, al menos la version 3.6.8, e instalar los requerimientos estipulados en `src/demo/requirements.txt`. Idealmente, el sistema se instalará en Windows 10, ya que la aplicación está optimizad para este sistema operativo. Sin embargo, funciona en todos los sistemas operativos probados (Windows, Linux y Mac).

E.3. Instalación

Los pasos para la instalación del proyecto se detallan en D.4. Les resumimos:

1. Clonar repositorio e ir al directorio `src/demo` o , en caso de tener el *release*, abrir únicamente el *release* en el directorio raíz.

2.

3. Descargar archivo de pesos del modelo[1] *VGGish* y alojarlo en la carpeta `prediccion`.

4. Ejecutar install.cmd o pip install -r requirements.txt

Para ejecutar la aplicación:

python main.py

E.4. Manual del usuario

Detallaremos como realizar las operaciones principales de la aplicación.

Cargar y borrar un audio

Cargar un audio

Para cargar un audio, ver Figura E.1:

1. Clickar botón '+ Añadir'

2. Elegir audio o audios tipo wav en el menú desplegable.

3. Los audios aparecerán listados.

[1] https://mega.nz/#!fRRFSKrT!0EMBqtYjogQrSQgudWifOAXm_A5Yx9UnX5Qk_Enanuk

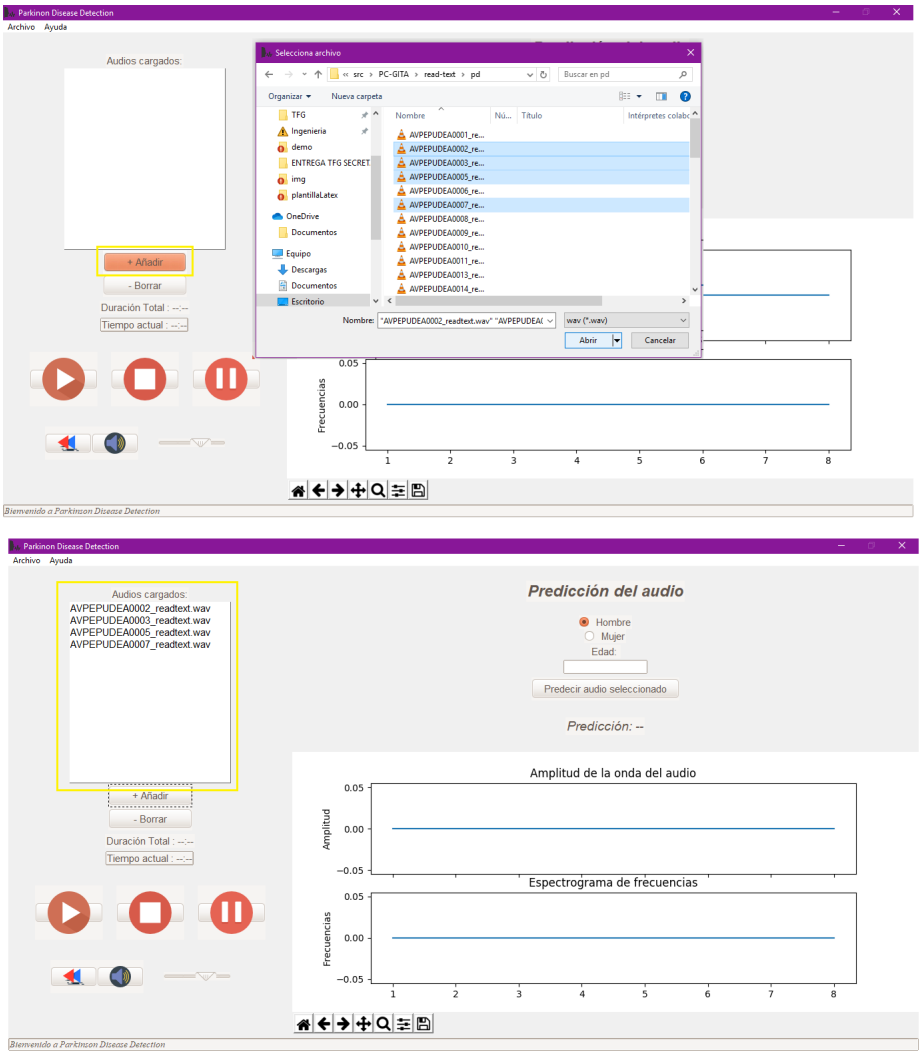

Figura E.1: Carga de audios en la aplicación

Borrar un audio

Para borrar un audio, ver Figura E.2:

1. Elegir un audio. Destacamos que cuando digamos elegir audio de aquí en adelante, nos referimos a clickar encima del audio. Es decir clickar encima del nombre del audio en la lista de audios cargados, y este se pondrá remarcado en azul. Cuando esté remarcado en azul, consideraremos que el audio está elegido.

2. Clickar botón 'Borrar'.

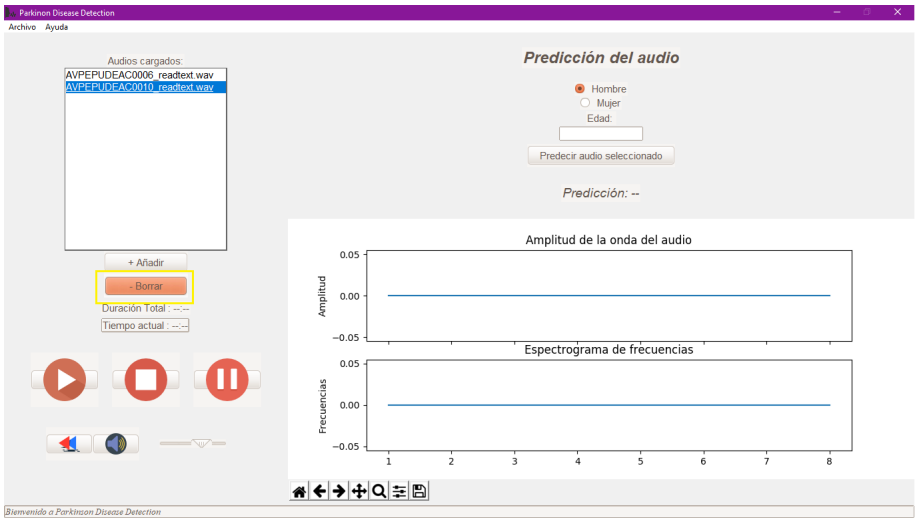

Figura E.2: Borra audios de la aplicación

Reproducir un audio

Para reproducir un audio, ver Figura E.3:

1. Elegir un audio.
2. Clickar botones de control de reproducción: *play*, *pause*, *stop*, *rewind*, *mute* o *volume*.
3. El audio se reproducirá y se detallará en pantalla un reloj de reproducción.

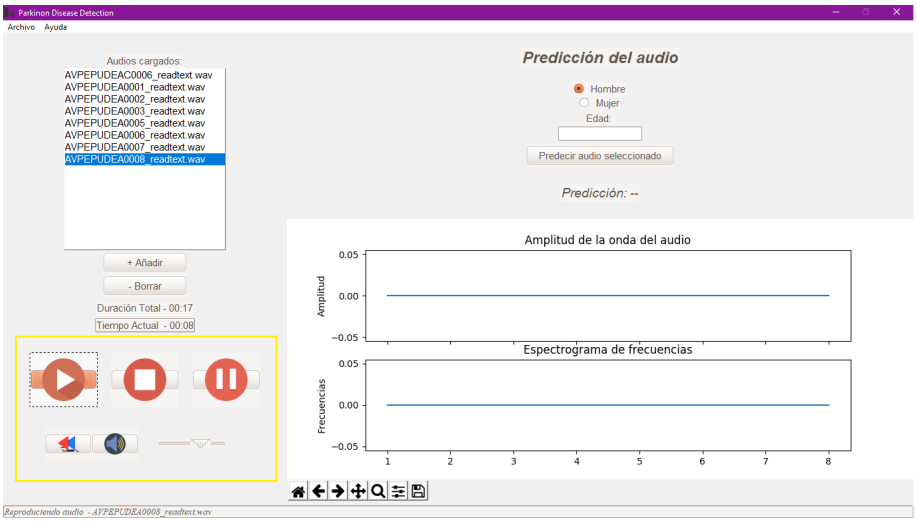

Figura E.3: Reproduce un audio de la aplicación

Predice un audio y vea gráficas

Para predecir un audio un audio, ver Figura E.4:

1. Elegir un audio.
2. Elegir el sexo.
3. Introducir edad en formato entero mayor que 0.
4. Clickar botón 'Predecir audio'.
5. Se mostrará en pantalla el resultado de la predicción en texto. Se mostrará si el paciente está sano o no y con que probabilidad (color de letra verde si sano, rojo si PD). Se mostrará en pantalla las dos gráficas de análisis del audio: amplitud y espectrograma de frecuencias.

Figura E.4: Predice un audio de la aplicación

Apéndice F

Anexo de investigación

F.1. Introducción

En esta sección detallaremos las taxonomías realizadas, donde se resume la investigación realizada. Se resumen tanto artículos, como características extraídas de los audios y las herramientas con las que se obtienen. La mayoría de las taxonomías son muy anchas, por lo que las tablas no caben en esta página, ni siquiera apaisadas. Por tanto, se referenciamos la ruta de la misma en el proyecto y se dará el link de acceso al archivo en la nube.

También describiremos el árbol de características extraídas, es decir, la estructura de directorios de las características extraídas.

F.2. Árbol de características

El árbol de directorios de características lo podemos ver en la Figura F.1. En el árbol se utilizan abreviaturas.

art articulación.

fon fonación.

prs prosodia.

ixc número de instancias por conjunto. Número de audios de cada conjunto de datos.

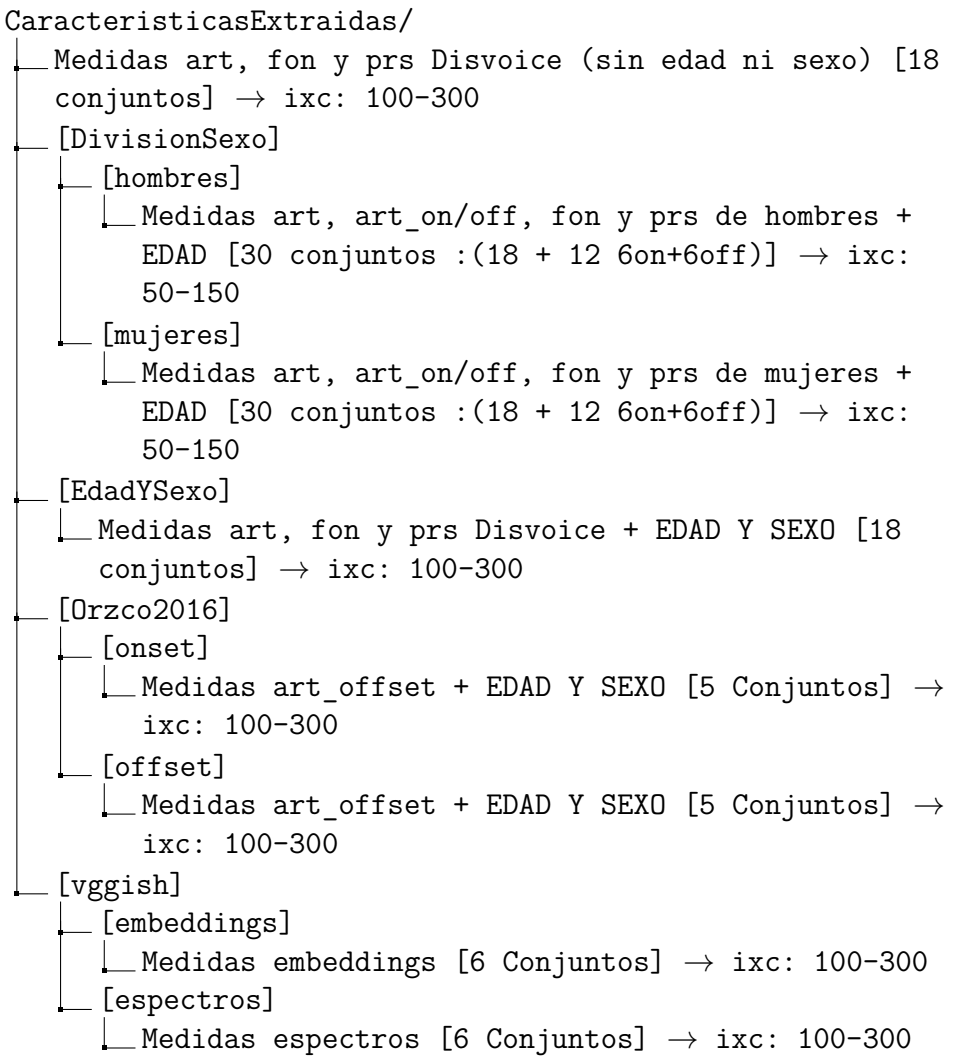

Figura F.1: Árbol de características

F.3. Taxonomías

Se explicarán las taxonomías realizadas en el proyecto.

Artículos relacionados

Se detallan los artículos relacionados. Se describe nombre, fuente (link o DOI), año, autor, revista, campo de la revista, número de citas, descripción (no en todos) y enfermedad tratada en el artículo.

Taxonomía:

- `doc/masRecursos/Articulos_Relacionados_Tabla_Resumen.csv`
- Enlace a Drive Spreadsheet[1]

Bases de datos de audios

Se detallan las bases de datos o conjunto de audios encontrados. Se describe nombre, descripción, autor, fuente (link o DOI), enfermedad, **privacidad**, precio, características técnicas de los audios, características extraídas del mismo, resultados de clasificación, año e idioma.

Taxonomía:

- `datasets/Tabla Resumen Datasets.csv`
- Enlace a Drive Spreadsheet[2]

Características de audios

Características de cada audio

Se detallan las características extraídas de cada tipo de audio. Se describe el tipo de audio, la segmentación del audio, las características extraídas, una nota de la característica, la herramienta de extracción de la misma y el artículo donde está descrita la herramienta.

- `doc/masRecursos/Taxonomía ccas completa.csv`

[1] https://docs.google.com/spreadsheets/d/10f7dg0UIAlxNedFbIpXQ2vrjXtpHhHqNO4hSk7RQBzw/edit?usp=sharing

[2] https://docs.google.com/spreadsheets/d/1yKu5-wWa_uh_VvmoD7Uo6nLbyFrLpeDXlRFeSGhPFak/edit?usp=sharing

- Enlace a Drive Spreadsheet[3]

Características de cada audio por artículo

Se detallan las características extraídas de cada tipo de audio **divididas por cual se sacan en cada artículo**. Se describe **el artículo donde se utilizan**, el tipo de audio, la segmentación del audio, las características extraídas, una nota de la característica, la herramienta de extracción de la misma y el artículo donde está descrita la herramienta.

- `doc/masRecursos/Taxonomia_ccas_POR_ARTICULO.csv`
- Enlace a Drive Spreadsheet[4]

Bibliotecas de extracción de características

Se detallan las bibliotecas de manejo de audios. Se describe el nombre de la herramienta, la fuente o link de descarga y una pequeña descripción de la misma (que características de las más importantes se pueden obtener con esa herramienta).

- `doc/masRecursos/Librerías_Tabla_Resumen.csv`
- Enlace a Drive Spreadsheet[5]

Relación característica → herramienta

Se describe con que herramienta se saca cada característica del audio. Se detalla el nombre de la característica, una nota de la misma y la o las herramientas con las que se puede extraer.

- `doc/masRecursos/Listado ccas-herramientas_Tabla_Resumen.csv`
- Enlace a Drive Spreadsheet[6]

---

[3] https://docs.google.com/spreadsheets/d/1uQ81sL3kpDlMKDWr0w2Sk8U1QC6e1yJUkTx_8f6cU2I/edit?usp=sharing

[4] https://docs.google.com/spreadsheets/d/1dLJq3So6EIyfl-udspz0htBf2Lx7AFqfdjMuFHX9s_c/edit?usp=sharing

[5] https://docs.google.com/spreadsheets/d/1WxdAnneBYJOd0tMNeatMloWAJ2XOTcDFqndSbUNOjvc/edit?usp=sharing

[6] https://docs.google.com/spreadsheets/d/1bUq36piHUQCQoivLOQCX7_Q8QrqQsXXfocesUeyRA_k/edit?usp=sharing

Bibliografía

[1] Giovanni Dimauro, Vincenzo Di Nicola, Vitoantonio Bevilacqua, Danilo Caivano, and Francesco Girardi. Assessment of speech intelligibility in parkinson's disease using a speech-to-text system. *IEEE Access*, pages 22199–22208, 2017.

[2] Giovanni Dimauro, Vincenzo Di Nicola, Vitoantonio Bevilacqua, Francesco Girardi, and Vito Napoletano. Voxtester, software for digital evaluation of speech changes in parkinson disease. *Proc. IEEE Int. Symp. Med. Meas. Appl. (MeMeA)*, pages 1–6, 2016.

[3] Dominio. Unit testing for data science, 2019. [Internet; descargado 30-junio-2019].

[4] Shawn Hershey, Sourish Chaudhuri, Daniel PW Ellis, Jort F Gemmeke, Aren Jansen, R Channing Moore, Manoj Plakal, Devin Platt, Rif A Saurous, Bryan Seybold, et al. Cnn architectures for large-scale audio classification. In *2017 ieee international conference on acoustics, speech and signal processing (icassp)*, pages 131–135. IEEE, 2017.

[5] J. R. Orozco-Arroyave, J. D. Arias-Londoño, J. F. Vargas-Bonilla, M. González-Rátiva, and E. Nöth. New spanish speech corpus database for the analysis of people suffering from parkinson's disease. *Proceedings of the 9th Language Resources and Evaluation Conference (LREC)*, pages 342–347, 2014.

[6] J. R. Orozco-Arroyave, F. Hönig, J. D. Arias-Londoño, J. F. Vargas-Bonilla, K. Daqrouq, S. Skodda, J. Rusz, and E. Nöth. Automatic detection of parkinson's disease in running speech spoken in three different languages. *The Journal of the Acoustical Society of America*, 139:481–500, Enero 2016.

[7] Alex Osterwalder. Business model design and innovation, 2007.

[8] Wikipedia. Facade (patrón de diseño) — wikipedia, la enciclopedia libre, 2019. [Internet; descargado 28-junio-2019].

[9] Wikipedia. Mediador (patrón de diseño) — wikipedia, la enciclopedia libre, 2019. [Internet; descargado 28-junio-2019].