UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

TFG del Grado en Ingeniería Informática

Visor de espectros
Documentación Técnica

Presentado por Iván Iglesias Cuesta
en Universidad de Burgos — 28 de junio de 2018
Tutor: Dr. José Francisco Díez Pastor
Cotutor: Dr. César Ignacio García Osorio

Índice general

Índice de figuras

Índice de tablas

Apéndice A

Plan de Proyecto Software

A.1. Introducción

Para que un proyecto se desarrolle con normalidad y con el menor número de imprevistos posibles es esencial que cuente con una fase de planificación. Aquí se estima el tiempo, trabajo y dinero necesario para realización del proyecto.

Para ello, se debe analizar en detalle cada parte del proyecto. De cara al futuro, el análisis del proyecto puede servir para predecir como de bien puede desarrollarse una continuación del mismo.

La planificación del proyecto consta de dos partes:

- **Planificación temporal**: en esta parte se analiza y planifica el tiempo que se va a dedicar a cada parte del proyecto, fecha de inicio y final aproximado, teniendo en cuenta el trabajo necesario para cada parte.
- **Estudio de viabilidad**: en esta parte se analiza como de viable es la realización del proyecto, se divide a su vez en dos apartados:

    - **Económica**: en esta parte se estiman los costes y los beneficios que puede suponer el proyecto.
    - **Legal**: en esta parte se analizan los conceptos legales del proyecto, como podrían ser las licencias del proyecto o la política de protección de datos.

A.2. Planificación temporal

La planificación temporal se organiza mediante *sprints*. Cada *sprint* dura una o dos semanas. Al terminar cada *sprint* se realiza una reunión con los tutores para dar por terminado

Sprint 0

En este *sprint* se marcó el inicio del proyecto. La lista de tareas está disponible en Sprint 0[1]. En reuniones previas se habló con los tutores en que iba a consistir en proyecto, pero no estaba claro con que tecnologías desarrollarlo. Se decidió hacer una evaluación de las tecnologías posibles y crear unos prototipos básicos. Todas las tareas se completaron a tiempo.

Sprint 1

En este *sprint* se habló sobre el despliegue de la aplicación. Los tutores comentaron que en proyectos webs anteriores el despliegue se solía dejar para las etapas finales del proyecto, haciendo que todos los problemas asociados surjan en esas etapas finales, retrasando el despliegue y, a veces, impidiendo desplegar la aplicación.

Se conocía la plataforma Heroku así que fue la primera opción que se probó., adicionalmente se buscaron otras alternativas. También se usó el prototipo escogido para crear el proyecto definitivo y se trabajó en la memoria. La última parte fue estudiar las guías de estilo de Python, aplicarlas en los prototipos y documentar su código.

La lista de tareas está disponible en Sprint 1[2]. Todas las tareas se completaron a tiempo.

[1] https://github.com/IvanBeke/TFG-Visor-de-espectros/milestone/1?closed=1

[2] https://github.com/IvanBeke/TFG-Visor-de-espectros/milestone/2?closed=1

Sprint 2

Dado que en la aplicación se necesita que los usuarios suban contenido, se necesita control de usuarios, en este *sprint* se investigaron formas de ofrecerlo. También debido a la necesidad de disponer almacenamiento persistente, se tuvo que mirar otras formas de despliegue e investigar como Heroku lo ofrece, dado que por defecto no lo hace. Al final se decidió cambiar a Digital Ocean con Nanobox.

La lista de tareas está disponible en Sprint 2[3]. Todas las tareas se completaron a tiempo.

Sprint 3

En este *sprint* se podría decir que comienza el desarrollo del proyecto, basado en el prototipo. Como tal se mejoró el aspecto visual de la aplicación, se cambio la estructura para tener partes diferenciadas y mantenibles, añadir control de usuarios y mejorar la subida de ficheros.

También se planteó añadir, subir y visualizar un *dataset* completo, escribir el manual de despliegue y los casos de uso.

La lista de tareas está disponible en Sprint 3[4]. El manual de despliegue y la subida de *datasets* no se pudieron completar. El gráfico *burndown* del *sprint* se ve en la figura A.1.

[3] https://github.com/IvanBeke/TFG-Visor-de-espectros/milestone/3?closed=1

[4] https://github.com/IvanBeke/TFG-Visor-de-espectros/milestone/4?closed=1

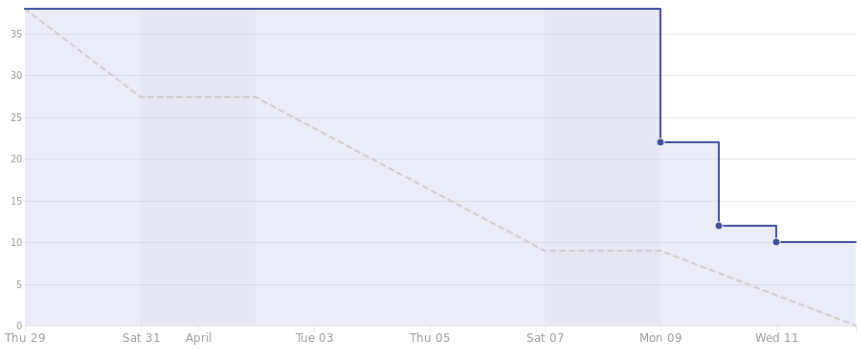

Figura A.1: Burndown del *sprint* 3

Sprint 4

En este *sprint* se completan las tareas que no habían dado tiempo del *sprint* anterior y se planteó usar una base de datos en lugar de almacenamiento para guardar los datos, por lo que se realizó una comparación entre formas de almacenar los datos. También se añadió la opción de borrar un *dataset* ya almacenado.

Similar a lo ocurrido en el *sprint* anterior, se cambió la estructura de la aplicación, en este *sprint* se estructuró la parte de visualización. También se arreglaron dos errores que se introdujeron en el *sprint* anterior en la aplicación desplegada.

La lista de tareas está disponible en Sprint 4[5]. Todas las tareas se completaron a tiempo. El gráfico *burndown* del *sprint* se ve en la figura A.2.

[5] https://github.com/IvanBeke/TFG-Visor-de-espectros/milestone/5?closed=1

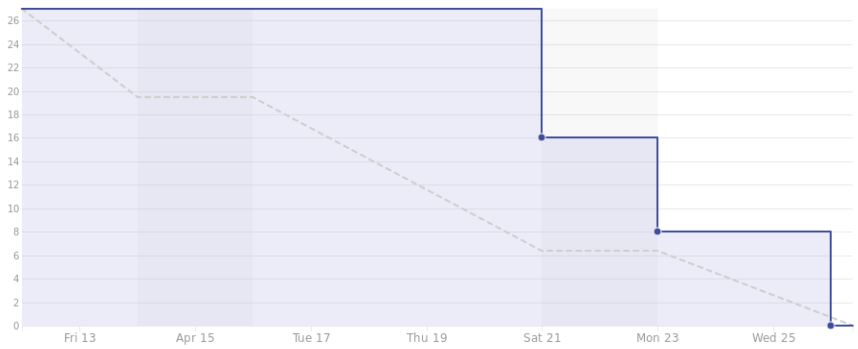

Figura A.2: Burndown del *sprint* 4

Sprint 5

De la comparación del *sprint* anterior se decidió usar MongoDB para el almacenamiento, por lo cual la mayoría de los esfuerzos se centraron en adaptar la aplicación para usar MongoDB en todos sus aspectos: guardar, borrar y coger los datos. También se solucionó un error en la parte de visualización.

La lista de tareas está disponible en Sprint 5[6]. Todas las tareas se completaron a tiempo. El gráfico *burndown* del *sprint* se ve en la figura A.3.

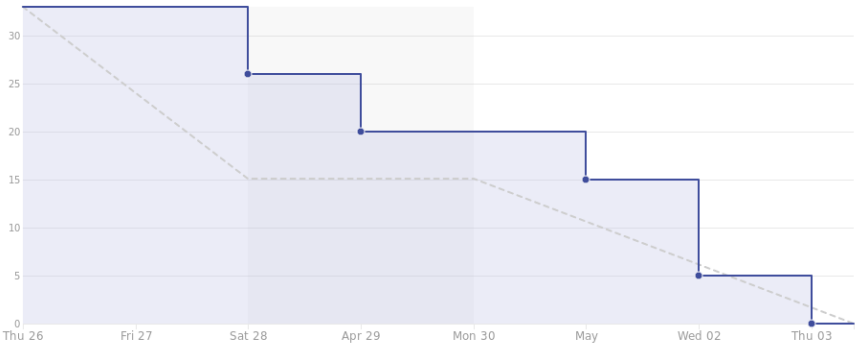

Figura A.3: Burndown del *sprint* 5

[6] https://github.com/IvanBeke/TFG-Visor-de-espectros/milestone/6?closed=1

Sprint 6

Para este *sprint* se planteó cambiar la forma en la que se guardan los *datasets*, de forma que cuando el sistema de aprendizaje automático esté implementado, sea más sencillo pasar los datos para el entrenamiento. También se añadió soporte de comentarios en el *dataset*. Debido a un cambio en como la geóloga organizaba sus datos, se tuvo que adaptar la subida de *datasets* a este cambio. Por último, se empezó a añadir controles en la parte de visualización para el procesado de los espectros.

La lista de tareas está disponible en Sprint 6[7]. La tarea de los controles no se completó a tiempo. El gráfico *burndown* del *sprint* se ve en la figura A.4.

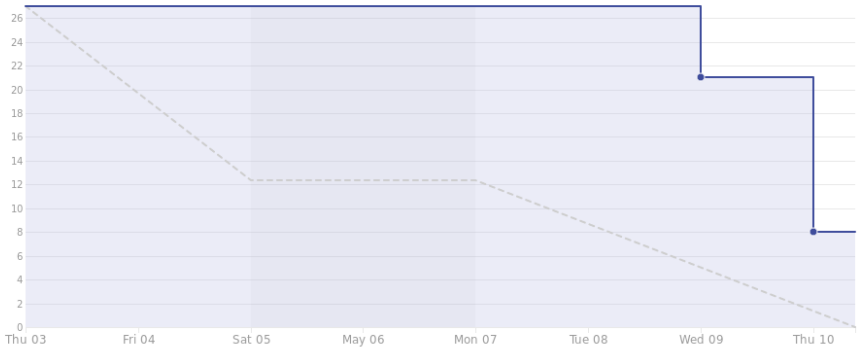

Figura A.4: Burndown del *sprint* 6

Sprint 7

Este *sprint* se centró en completar la tarea del *sprint* anterior, cambiar la parte de visualización para que se muestre en un tabla los ejemplos subidos y sus metadatos, documentar el código y avanzar en la memoria y anexos.

Durante el *sprint*, se vio que la tarea sobre procesamiento de datos era demasiado extensa, dividiéndola en dos partes, la primera que sería para añadir los controles para el procesamiento en la interfaz, y una segunda para añadir el código de procesamiento de datos, para realizar en el siguiente

[7] https://github.com/IvanBeke/TFG-Visor-de-espectros/milestone/7?closed=1

sprint.

La lista de tareas está disponible en Sprint 7[8]. No dio tiempo a escribir la introducción de la memoria. El gráfico *burndown* del *sprint* se ve en la figura A.5.

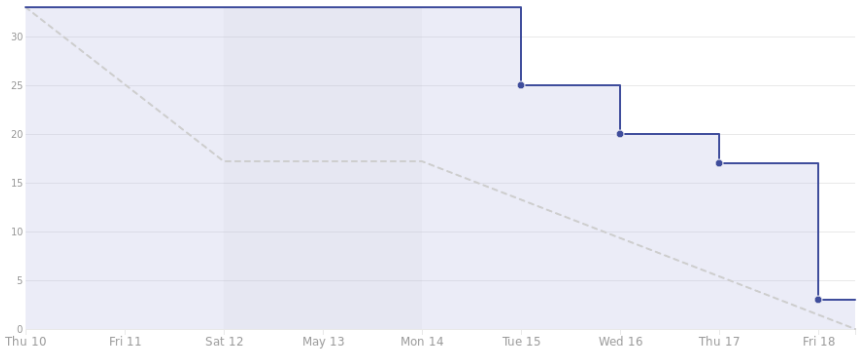

Figura A.5: Burndown del *sprint* 7

Sprint 8

Este *sprint* se centró en completar la tarea del *sprint* anterior, arreglar un error del despliegue, añadir al proyecto el código de procesamiento de datos y enlazarlo a los controles, añadir instrucciones de esta parte y ampliar la planificación temporal con links al repositorio y capturas de los gráficos *burndown*.

Se recuerda que el código de procesamiento es gran parte de los resultados del proyecto previo en las colaboraciones.

La lista de tareas está disponible en Sprint 8[9]. Todas las tareas se completaron a tiempo. El gráfico *burndown* del *sprint* se ve en la figura A.6.

[8] https://github.com/IvanBeke/TFG-Visor-de-espectros/milestone/8?closed=1

[9] https://github.com/IvanBeke/TFG-Visor-de-espectros/milestone/9?closed=1

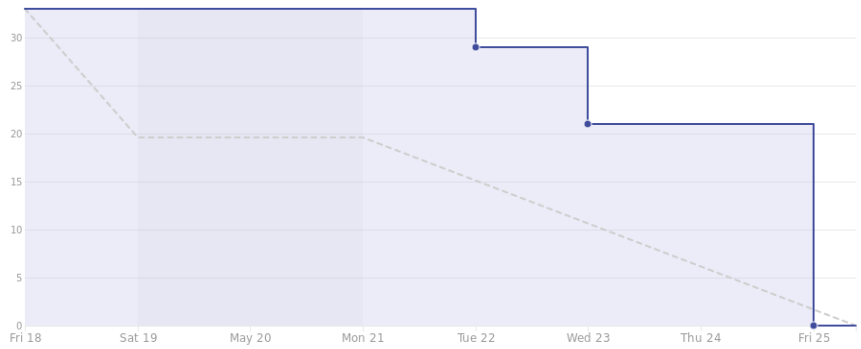

Figura A.6: Burndown del *sprint* 8

Sprint 9

Este *sprint* se centró en añadir la funcionalidad de los espectros individuales: subida, visualización, procesado y borrado. También se trabajó en los objetivos de la memoria.

La lista de tareas está disponible en Sprint 9[10]. Todas las tareas se completaron a tiempo. El gráfico *burndown* del *sprint* se ve en la figura A.7.

Figura A.7: Burndown del *sprint* 9

[10] https://github.com/IvanBeke/TFG-Visor-de-espectros/milestone/10?closed=1

Sprint 10

Este *sprint* se centró en añadir la funcionalidad de minería de datos respecto a creación de clasificadores y varias mejoras visuales de la interfaz.

Este *sprint* duró dos semanas por desarrollarse a la vez que la época de exámenes. Las tareas relacionadas con la mejora de la web no hubo problemas en completarlas, pero las relacionadas a la creación de clasificadores llevaron más tiempo del esperado y no se pudieron completar a tiempo.

La lista de tareas está disponible en Sprint 10[11]. El gráfico *burndown* del *sprint* se ve en la figura A.8.

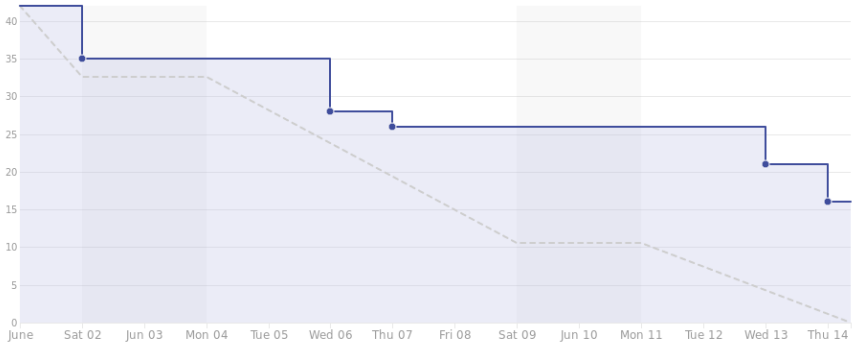

Figura A.8: Burndown del *sprint* 10

Sprint 11

En este *sprint* se completaron las tareas pendientes del *sprint* anterior, además se realizó también la tarea de codificar la predicción de nuevos espectros. También se avanzó bastante en la memoria, aunque no se llegaron a terminar completamente todas las secciones previstas.

Debido a la cercanía de la entrega este *sprint* tuvo más carga de trabajo que los anteriores. Se puede considerar el último *sprint* de desarrollo porque

[11] https://github.com/IvanBeke/TFG-Visor-de-espectros/milestone/11?closed=1

se terminaron de implementar los objetivos del proyecto.

La lista de tareas está disponible en Sprint 11[12]. El gráfico *burndown* del *sprint* se ve en la figura A.9. Debido a que gran parte de las tareas eran de documentación, se avanzaba en ellas en paralelo y se cerraron al final del *sprint*, de ahí la forma del gráfico.

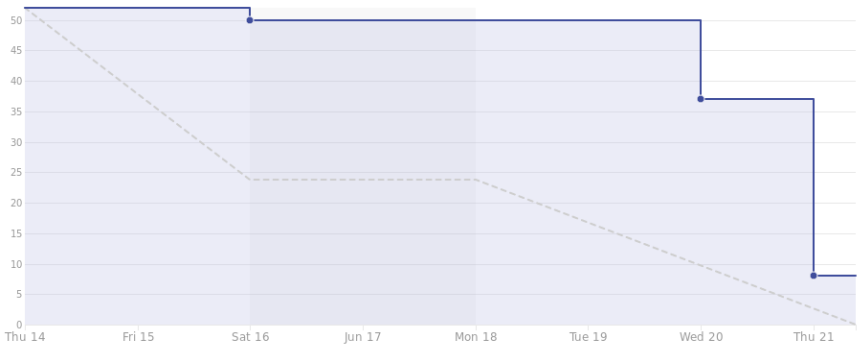

Figura A.9: Burndown del *sprint* 11

Sprint 12

Último *sprint* del proyecto. Se realizaron mejorar menores en la aplicación, se implementaron unos tests unitarios y se terminó de escribir la memoria y anexos. Se completaron todas las tareas.

La lista de tareas está disponible en Sprint 12[13]. El gráfico *burndown* del *sprint* se ve en la figura A.10.

[12] https://github.com/IvanBeke/TFG-Visor-de-espectros/milestone/12?closed=1

[13] https://github.com/IvanBeke/TFG-Visor-de-espectros/milestone/13?closed=1

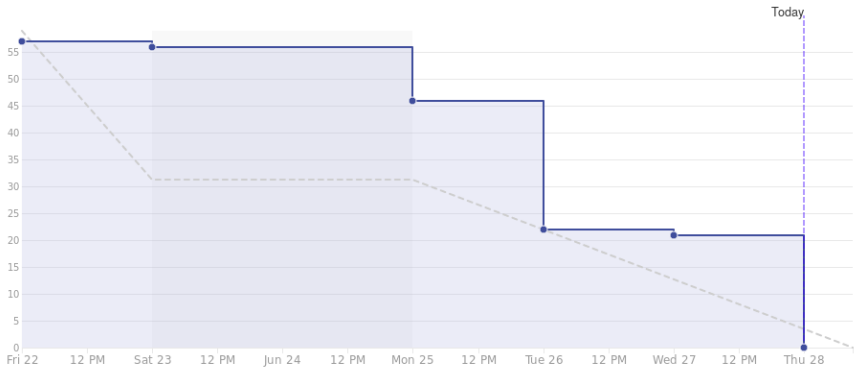

Figura A.10: Burndown del *sprint* 12

A.3. Estudio de viabilidad

Viabilidad económica

Costes de personal

El proyecto se ha llevado a cabo por un desarrollador contratado a tiempo parcial durante 4 meses. Se considera un salario neto de 1000 € mensuales (ver tabla A.1).

La cotización a la seguridad social se ha calculado como horas comunes, según el régimen general de 2018 (28,30 %) [7].

Concepto	**Coste**
Salario neto	1000 €
Retención IRPF (19 %)	360,53 €
Seguridad social (28,30 %)	537,00 €
Salario bruto (mensual)	1897,53 €
Total 4 meses	7590,12 €

Tabla A.1: Costes de personal

Costes de hardware

En esta sección se enumeran los costes del hardware usado durante el desarrollo.

Para el desarrollo se ha usado un ordenador portátil valorado en 800€, con amortización en 4 años (ver tabla A.2).

$$\frac{800€}{4\ años * 12\ meses} = 16{,}67€$$

Concepto	**Coste**	**Amortización**
Ordenador portátil	800€	16,67€
Total 4 meses	66,67€	

Tabla A.2: Costes de hardware

Costes de software

El proyecto se ha desarrollado usando el sistema Ubuntu, por lo que en este aspecto no habría costes. La licencia de PyCharm Professional supone un gasto de 8,90€ mensuales [6]. La licencia de GitKraken Pro supone un pago único anual de 49$, que aproximadamente corresponde a 42€ [2]. Los costes se resumen en la tabla A.3.

Concepto	**Coste**
PyCharm Professional (mensual)	8,90€
GitKraken Pro	42€
Total 4 meses	77.60€

Tabla A.3: Costes de software

Costes del servidor

Como servidor se ha elegido un *Droplet* estándar con 2GB de memoria, 1 CPU virtual, 50GB de disco duro y 2TB de transferencia cuyo coste es de 10$ mensuales, aproximadamente 8,60€ mensuales [1]. Los costes se resumen en la tabla A.4.

Costes totales

En la tabla A.5 se agrupan todos los costes calculados del proyecto, dando el total de 7768,99€.

Concepto	Coste
Digital Ocean (mensual)	8,60 €
Total 4 meses	34,40 €

Tabla A.4: Costes del servidor

Concepto	Coste
Personal	7590,12 €
Hardware	66,67 €
Software	77,60 €
Servidor	34,60 €
Total	7768,99 €

Tabla A.5: Costes totales del proyecto

Beneficios

Para obtener beneficios del proyecto se podría plantear añadir a la aplicación las siguientes alternativas:

- Limitaciones de almacenamiento: la cantidad de ficheros que puede subir un usuario estaría limitada por un plan de pago.
- *Freemium* [9]: este modelo funciona ofreciendo unas funcionalidades básicas a todos los usuarios, pero bloqueando algunas a usuarios que no hayan pagado, por ejemplo, la creación de clasificadores.
- Publicidad: podría incluirse en la aplicación publicidad relacionado con el tema.

Para obtener el máximo beneficio estas opciones podrían combinarse.

Viabilidad legal

A lo hora de añadir una licencia al proyecto hay que tener en cuenta a que licencias están sometidas las dependencias. Con la ayuda de la herramienta Requires.io[14], se han listado las dependencias, versión usada y licencia (ver tabla A.6).

[14] https://requires.io/

Dependencia	Versión	Licencia
dash	0.21.1	MIT
dash-core-components	0.22.1	MIT
dash-html-components	0.10.1	MIT
dash-table-experiments	0.6.0	MIT
dash_renderer	0.11.3	MIT
Flask	1.0.2	BSD
Flask-Bootstrap	3.3.7.1	BSD
Flask-Dance	0.14.0	MIT
Flask-PyMongo	0.5.1	BSD
Flask-WTF	0.14.2	BSD
gunicorn	19.8.1	MIT
ipython	6.3.1	BSD
numpy	1.14.3	BSD
numpydoc	0.8.0	BSD
pandas	0.23.0	BSD
plotly	2.5.1	MIT
pymongo	3.6.1	Apache License 2.0
pyOpenSSL	17.5.0	Apache License 2.0
PyWavelets	0.5.2	MIT
scikit-learn	0.19.1	BSD 3-Clause
scipy	1.1.0	BSD
Werkzeug	0.14.1	BSD
WTForms	2.2.1	BSD
xlrd	1.1.0	BSD

Tabla A.6: Dependencias del proyecto

De las dependencias usadas, todas son licencias bastante permisivas que permiten el uso con libertad, por lo que no tenemos que preocuparnos de incompatibilidades con la licencia que se escoja.

Con ayuda de las recomendaciones GNU [4], se ha escodigo la licencia GPL-3.0 [3]. Esta licencia permite la modificación, uso y distribución del software, siempre que esto se haga bajo la misma licencia, se indiquen los cambios y se mencione al autor original.

Apéndice B

Especificación de Requisitos

B.1. Introducción

En este apéndice se describen los objetivos generales de la aplicación y se detallan sus requisitos, tanto funcionales como no funcionales.

B.2. Objetivos generales

- Ofrecer control de usuarios.
- Permitir a los usuarios subir *datasets* y espectros.
- Que los *datasets* y espectros se puedan visualizar.
- Que sobre la visualización se pueda aplicar operaciones de procesamiento.
- Poder entrenar modelos de aprendizaje automático con los *datasets* subidos.
- Poder usar los modelos mencionados anteriormente para predecir nuevos ejemplos subidos.
- Que la aplicación final sea útil para la investigadora, que hace las veces de cliente en este proyecto.

B.3. Catálogo de requisitos

Requisitos funcionales

- **RF-1 Control de usuarios**: la aplicación debe permitir controlar usuarios.
  - **RF-1.1 Integración con Google**: la aplicación debe poder hacer uso de cuentas de Google para el control de usuarios.
  - **RF-1.2 Inicio de sesión**: el usuario debe poder iniciar sesión con una cuenta de Google.
  - **RF-1.3 Cierre de sesión**: el usuario debe poder cerrar sesión cuando haya terminado.
- **RF-2 Datasets**: la aplicación debe poder almacenar y gestionar conjuntos de espectros.
  - **RF-2.1 Subida**: el usuario debe poder subir un *dataset*.
  - **RF-2.2 Eliminado**: el usuario debe poder eliminar un *dataset* almacenado.
  - **RF-2.3 Visualización**: el usuario debe poder visualizar un *dataset* almacenado.
- **RF-3 Espectros**: la aplicación debe poder almacenar y gestionar espectros.
  - **RF-3.1 Subida**: el usuario debe poder subir un espectro.
  - **RF-3.2 Eliminado**: el usuario debe poder eliminar un espectro.
  - **RF-3.3 Visualización**: el usuario debe poder visualizar un espectro.
- **RF-4 Procesamiento**: el usuario debe poder aplicar operaciones de preprocesamiento.
  - **RF-4.1: Procesamiento de *dataset***: el usuario debe poder aplicar operaciones de preprocesamiento sobre un *dataset* visualizado.
  - **RF-4.2: Procesamiento de espectro**: el usuario debe poder aplicar operaciones de preprocesamiento sobre un espectro visualizado.

- **RF-5 Minería de datos**: el usuario debe poder usar técnicas de minería de datos.
  - **RF-5.1 Creación**: el usuario debe poder crear modelos personalizados.
  - **RF-5.2 Evaluación**: la aplicación debe ofrecer métricas del modelo entrenado.
  - **RF-5.3 Predicción**: el usuario debe poder usar los modelos que ha creado para predecir nuevos espectros.

Requisitos no funcionales

- **RNF-1 Usabilidad**: la aplicación debe ser intuitiva y fácil de usar.
- **RNF-2 Escalabilidad**: el rendimiento debe poder aumentar al incrementar los recursos.
- **RNF-3 Manteninibilidad**: debe ser sencillo añadir funcionalidad nueva a la aplicación.
- **RNF-4 Compatibilidad**: la aplicación debe poder funcionar en los principales navegadores.
- **RNF-5 Responsividad**: la aplicación debe adaptarse al tamaño de la pantalla.
- **RNF-6 Facilidad de despliegue**: la aplicación debe poder desplegarse en un servidor de forma sencilla.

B.4. Especificación de requisitos

En esta sección se desarrollan los casos de uso de la aplicación. A continuación, se expone el diagrama de casos de uso que los resume.

CU-1	**Iniciar sesión**
Versión	1.0
Autor	Iván Iglesias Cuesta
Requisitos asociados	RF-1, RF-1.1, RF-1.2
Descripción	El usuario inicia sesión en la aplicación.

CU-1	**Iniciar sesión**
Precondición	Navegador web abierto y página de la aplicación cargada.
Acciones	1. Pulsar botón “Iniciar sesión con Google”. 2. Introducir o seleccionar cuenta de Google.
Postcondición	Redirección a la aplicación con sesión iniciada.
Excepciones	▪ Cuenta no existente. ▪ Combinación de nombre y contraseña incorrecta.
Importancia	Alta

Tabla B.1: Iniciar sesión.

CU-2	**Cerrar sesión**
Versión	1.0
Autor	Iván Iglesias Cuesta
Requisitos asociados	RF-1, RF-1.3
Descripción	El usuario cierra sesión en la aplicación.
Precondición	Sesión iniciada en la aplicación.
Acciones	1. Pulsar en el botón cuyo texto es el correo. 2. Pulsar "Cerrar sesión".
Postcondición	Redirección a la aplicación con sesión cerrada.
Excepciones	▪ La sesión no estaba iniciada.
Importancia	Alta

Tabla B.2: Cerrar sesión.

CU-3	**Subir dataset**
Versión	1.0
Autor	Iván Iglesias Cuesta
Requisitos asociados	RF-2, RF-2.1
Descripción	El usuario sube un dataset a la aplicación para su guardado.
Precondición	Sesión iniciada en la aplicación.
Acciones	1. Pulsar en el botón "Subir dataset". 2. Descargar y rellenar la plantilla. 3. Crear un fichero `.zip` con los datos y la plantilla. 4. Rellenar el formulario de subida. 5. Seleccionar el fichero creado. 6. Presionar el botón "Subir".
Postcondición	Redirección a la página de los archivos guardados.
Excepciones	▪ El formato del dataset no es correcto. ▪ Existe un dataset con el mismo nombre.
Importancia	Alta

Tabla B.3: Subir dataset.

CU-4	**Subir espectro**
Versión	1.0
Autor	Iván Iglesias Cuesta
Requisitos asociados	RF-3, RF-3.1
Descripción	El usuario sube un espectro a la aplicación para su guardado.
Precondición	Sesión iniciada en la aplicación.
Acciones	1. Pulsar en el botón “Subir espectro”. 2. Rellenar el formulario de subida. 3. Seleccionar el espectro. 4. Presionar botón “Subir”.
Postcondición	Redirección a la página de los archivos guardados.
Excepciones	▪ El formato del espectro no es correcto. ▪ Existe un espectro con el mismo nombre.
Importancia	Alta

Tabla B.4: Subir espectro.

CU-5	**Visualizar dataset**
Versión	1.0
Autor	Iván Iglesias Cuesta
Requisitos asociados	RF-2, RF-2.3, RF-4, RF-4.1
Descripción	El usuario visualiza un espectro en la aplicación.
Precondición	Sesión iniciada en la aplicación, dataset guardado en la aplicación y estar en la página de "Mis ficheros".
Acciones	1. Escoger un dataset que visualizar. 2. Pulsar el botón "Visualizar" en el dataset escogido.
Postcondición	Se muestra la página de visualización.
Excepciones	▪ No se ha podido cargar el dataset.
Importancia	Alta

Tabla B.5: Visualizar dataset.

CU-6	**Visualizar espectro**
Versión	1.0
Autor	Iván Iglesias Cuesta
Requisitos asociados	RF-3, RF-3.3, RF-4, RF-4.2
Descripción	El usuario visualiza un espectro en la aplicación.
Precondición	Sesión iniciada en la aplicación, espectro guardado en la aplicación y estar en la página de "Mis ficheros".
Acciones	1. Escoger un espectro que visualizar. 2. Pulsar el botón "Visualizar" en el espectro escogido.
Postcondición	Se muestra la página de visualización.
Excepciones	▪ No se ha podido cargar el espectro.
Importancia	Alta

Tabla B.6: Visualizar espectro.

CU-7	**Guardar clasificador**
Versión	1.0
Autor	Iván Iglesias Cuesta
Requisitos asociados	RF-5, RF-5.1, RF-5.2
Descripción	El usuario crear y entrena un modelo usando como datos un dataset guardado.
Precondición	Sesión iniciada en la aplicación, dataset guardado en la aplicación y estar en la página de "Mis ficheros".
Acciones	1. Escoger el dataset que se quiera usar como base. 2. Pulsar el botón "Crear modelo" en el dataset escogido. 3. Seleccionar en el desplegable el modelo a usar. 4. (Opcional) Rellenar el formulario con los parámetros del modelo. 5. Pulsar el botón "Crear y evaluar modelo". 6. Para guardar el clasificador: *a*) Completar el formulario. *b*) Pulsar el botón "Guardar". 7. Para no guardar el clasificador: *a*) Pulsar el botón "Descartar".
Postcondición	Se redirige a los archivos guardados.
Excepciones	▪ Los parámetros del modelo introducidos son erróneos. ▪ Ya existe un clasificador con el nombre introducido.
Importancia	Media

Tabla B.7: Guardar clasificador.

CU-8	**Predecir espectro**
Versión	1.0
Autor	Iván Iglesias Cuesta
Requisitos asociados	RF-5, RF-5.3
Descripción	El usuario usa un clasificador creado para predecir un espectro.
Precondición	Sesión iniciada en la aplicación, dataset guardado en la aplicación, algún clasificador creado y estar en la página de "Mis ficheros".
Acciones	1. Escoger el espectro que se quiera predecir. 2. Pulsar el botón "Predecir" en el espectro escogido. 3. Seleccionar en el desplegable el clasificador a usar. 4. Pulsar el botón "Predecir espectro". 5. Para guardar el clasificador:
Postcondición	Se muestra la predicción.
Excepciones	▪ No se puede cargar el clasificador.
Importancia	Media

Tabla B.8: Predecir espectro.

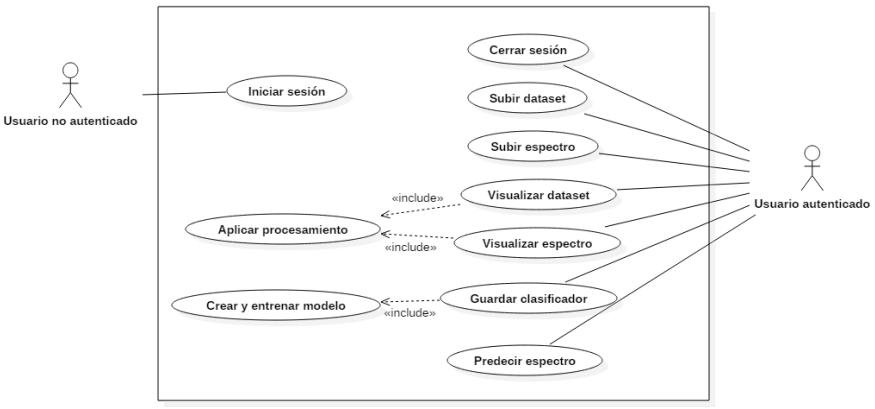

Figura B.1: Diagrama de casos de uso

Apéndice C

Especificación de diseño

C.1. Introducción

En este apéndice se expone el diseño que ha dado lugar a la aplicación. Se incluye el diseño de datos, diseño procedimental y diseño arquitectónico.

C.2. Diseño de datos

En esta sección se explican las entidades usadas por la aplicación:

- Dataset: esta entidad representa los conjuntos de espectros subidos por los usuarios.
- Spectrum: esta entidad representa un espectro subido por los usuarios.
- ClassifierSet: esta entidad representa los clasificadores entrenados por los usuarios, se compone de tres clasificadores, uno por cada atributo a predecir.

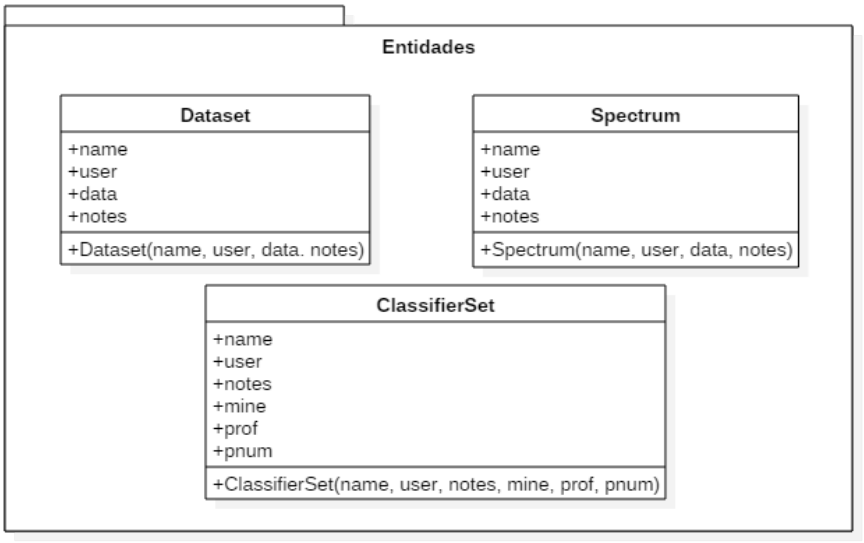

Figura C.1: Diagrama de clases

C.3. Diseño procedimental

En esta sección se explican las interacciones más importantes de la aplicación. Las interacción principal de la aplicación es, dentro de la visualización de datasets, cuando se selecciona un espectro para visualizar, ya que entran en juego todas las partes de la aplicación (ver figura C.2).

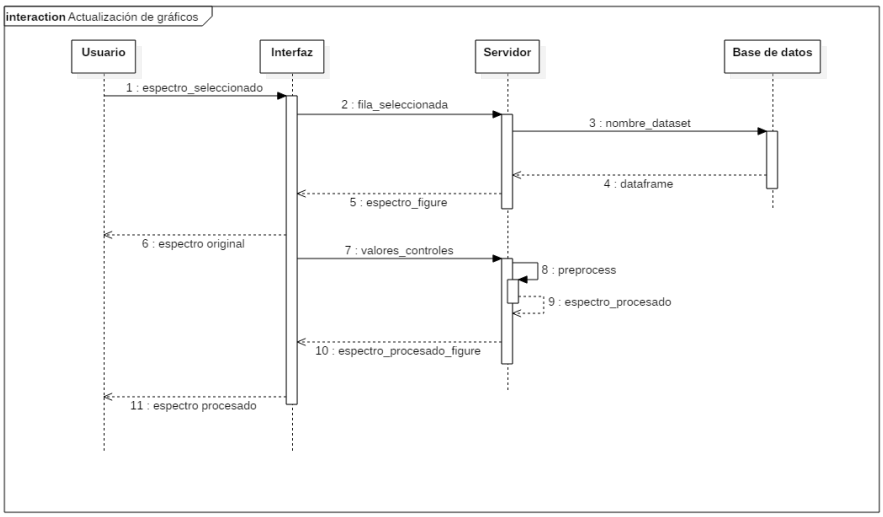

Figura C.2: Diagrama de secuencia de visualización de espectro

Cuando el usuario selecciona un espectro, se envía al servidor la fila seleccionada, el servidor pide a la base de datos el dataset que se está visualizando y lo devuelve, el servidor extrae del dataset el espectro seleccionado, lo convierte en una figura que se pueda visualizar y lo envía a la interfaz, que se lo muestra al usuario.

A continuación, para mostrar el espectro procesado, la interfaz envía al servidor los valores actuales de los controles, el servidor aplica las acciones de procesamiento al espectro, lo convierte en una figura para poder visualizarlo y lo envía a la interfaz, que lo muestra al usuario.

C.4. Diseño arquitectónico

Al haber desarrollado una aplicación web, la arquitectura del proyecto está condicionada por ello.

Modelo Vista Controlador (MVC)

Se ha seguido el patrón de MVC con el objetivo de separar la lógica de la aplicación (controlador), los datos (modelo) y la interfaz (vista). Este patrón permite facilitar la tarea del mantenimiento. Además, al ser un patrón conocido, se facilita la tarea de trabajar en la aplicación a futuros desarrolladores.

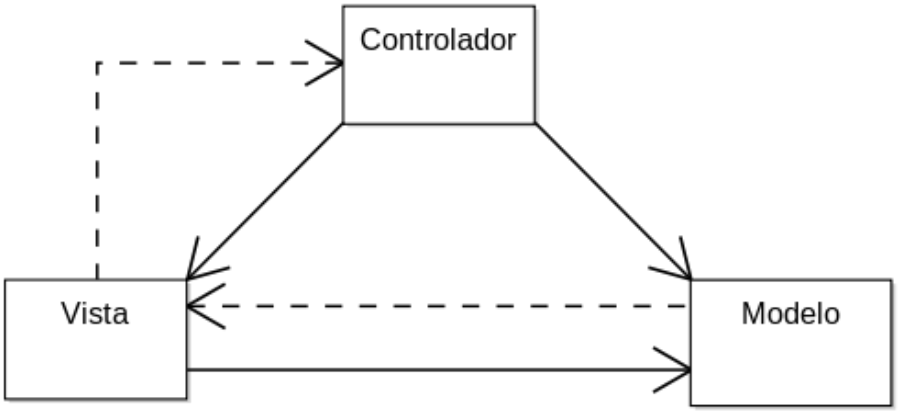

Figura C.3: Diagrama del patrón MVC [10]

Los modelos se encuentran presentes en `SpectraViewer/models.py`, los controladores se encuentran en `SpectraViewer/main/routes.py` y las vistas se encuentran en `SpectraViewer/templates/`.

Apéndice D

Documentación del programador

D.1. Introducción

En este apéndice se presenta todo lo que tiene que conocer un desarrollador para poder continuar con el desarrollo de la aplicación. Se describe la estructura de directorios del proyecto, como instalar la aplicación

D.2. Estructura de directorios

A continuación se presentan los directorios que contiene el proyecto con una breve descripción de cada uno.

```
/ Directorio raíz
├── Documentacion LaTeX/ -Documentación del proyecto
│   ├── img/ -Imágenes de la documentación
│   ├── tex/ -Secciones de la documentación
│   ├── anexos.pdf -Anexos del proyecto
│   └── memoria.pdf -Memoria del proyecto
├── etc/ -Configuración del servidor
└── SpectraViewer/ -Aplicación web
    ├── auth/ -Módulo de autenticación
    │   ├── __init__.py -Fichero principal del módulo
    │   └── routes.py -Rutas del módulo
    └── main/ -Módulo principal de la aplicación
        ├── __init__.py -Fichero principal del módulo
        └── errors.py -Manejadores de errores de la aplicación
```

```
        ├── forms.py -Formularios de WTForms
        └── routes.py -Rutas del módulo
    ├── processing/ -Fuentes de la librería superman ampliada
    ├── static/ -Contenido estático de la aplicación
    │   ├── css/ -Directorio con hojas de estilos
    │   ├── img/ -Imágenes usadas en la web
    │   └── js/ -Directorio con ficheros de JavaScript
    ├── templates/ -Plantillas de Jinja2
    │   └── errors/ -Plantillas específicas de errores
    ├── utils/ -Directorio con utilidades de la aplicación
    ├── visualization/ -Aplicaciones Dash para la visualización
    │   ├── __init__.py -Fichero principal del módulo
    │   ├── common.py -Cosas comunes en la visualización
    │   ├── dataset.py -Visualización de datasets
    │   └── spectrum.py -Visualización de espectros
    ├── __init__.py -Fichero principal del módulo
    ├── manual_de_uso.pdf -Manual de usuario
    ├── metadatos.xlsx -Plantilla de metadatos
    └── models.py -Modelos de la aplicación
├── tests/ -Directorio de las pruebas
├── boxfile.yml -Configuración de Nanobox
├── config.py -Configuración de la aplicación web
├── install_dependencies -Script para instalar dependencias
├── requirements.txt -Fichero con las dependencias del proyecto
└── start.py -Script para ejecutar la aplicación
```

D.3. Manual del programador

En esta sección se explican los puntos más importantes que los desarrolladores deben tener en cuenta para mantener o seguir ampliando este proyecto.

Definir funciones de actualización

Dentro de la parte de visualización, las funciones de actualización son una parte esencial, por lo que un punto importante es saber codificarlas correctamente.

Para empezar, son funciones de Python asociadas a elementos HTML de la interfaz que se ejecutan cuando una propiedad seleccionada del elemento se modifica. Esta asociación se realizado por medio de un decorador, que

recibe como parámetros un elemento que actualizar y una lista de elementos que tomar como entradas del método. El siguiente método se encarga de mostrar los espectros cuando se seleccionan en la tabla.

```
import dash
import dash_core_components as dcc
import dash_html_components as html
import dash_table_experiments as dt
from dash.dependencies import Input, Output

@app.callback(Output('spectrum-original', 'figure'),
      [Input('metadata', 'rows'),
      Input('metadata', 'selected_row_indices')])
def update_spectrum(rows, spectra_index):
  dataset = session['current_dataset']
  user_id = session['user_id']
  dataset_data = get_user_dataset(dataset, user_id)
  figure = {
    'layout': {
      'title': 'Espectro original',
      'xaxis': {'title': 'Raman shift'},
      'yaxis': {'title': 'Intensity'}
    },
    'data': list()
  }
  for i in spectra_index:
    name = rows[i]['Nombre']
    spectrum = dataset_data[dataset_data['Nombre'] ==
       name]
    spectrum = spectrum.drop(
      columns=['Nombre', 'Etiqueta', 'Mina', '
         Profundidad',
        'Profundidad_num'])
    figure['data'].append({
      'x': spectrum.columns.tolist(),
      'y': spectrum.values[0],
      'name': f'{name}'
    })
  return figure
```

Listing D.1: Carga de espectros al ser seleccionados

En el decorador se recibe una elemento `Output`, con parámetros `spectrum-original`, como elemento de salida, el grafico con el espectro original, y `figure` como la propiedad a modificar del elemento.

Como entradas se reciben dos `Input`, el primero corresponde a la tabla con los espectros, con la propiedad de las filas, y el segundo se refiere también a la tabla, pero con la propiedad asociada a las filas seleccionadas. Cada vez que se modifique una de esas propiedades se ejecutaría el método. Los parámetros que recibe el método son los valores de las propiedades definidas en los `Input`

Al terminar la ejecución del método, el valor devuelto ocupará el valor de la propiedad definida en el `Output`, en este caso la propiedad `figure` del espectro original.

Manual de despliegue

La configuración de despliegue tiene tres etapas, que se describen en esta sección.

Alojamiento

Para este proyecto el proveedor escogido ha sido Digital Ocean[1], pero puede servir cualquiera de los proveerdores compatibles[2] con Nanobox.

Nos dirigimos a la página del proveedor para registrarnos y poder conseguir un *token* que Nanobox nos pedirá más adelante. Es necesario asociar una tarjeta al servicio para que nos puedan cobrar el coste del servidor.

El *token* lo creamos en la sección API de la barra de navegación.

Nanobox

A continuación, crearemos una cuenta en el servicio Nanobox[3]. Con la cuenta ya confirmada, nos dirigimos a las opciones de la cuenta y asociamos la cuenta de Digital Ocean con el *token* obtenido previamente.

El siguiente paso es instalar la herramienta nanobox[4]. La primera que la usemos nos pedirá los datos de nuestra cuenta para iniciar sesión.

[1] https://www.digitalocean.com/
[2] https://docs.nanobox.io/providers/hosting-accounts/
[3] https://nanobox.io/
[4] https://docs.nanobox.io/install/

La configuración de este servicio se realiza por medio del fichero `boxfile.yml`, aquí se indican los componentes de la aplicación y su configuración.

Para poder desplegar la aplicación, necesitamos tener creado una aplicación en Nanobox, esto se realiza desde la web. Para ello, pulsamos el botón "Launch New App" y seguimos las instrucciones que se muestran.

Ahora estando todo configurado queda desplegar la aplicación, este paso se realiza con el siguiente comando:

```
$ nanobox deploy
```

Para cualquier otra duda con la herramienta se puede buscar en su documentación oficial [5].

D.4. Instalación y ejecución del proyecto

Dado que el proyecto se ha desarrollado usando el sistema operativo Ubuntu, las instrucciones de instalación están orientadas a ese sistema.

MongoDB

Antes de instalar el proyecto es necesario instalar la base de datos MongoDB[5].

Una vez instalado, para iniciar el servicio y poder conectarnos a la base de datos es necesario ejecutar el siguiente comando:

```
$ sudo service mongod start
```

Python

Este proyecto está desarrollado con la versión 3.6.3, pero con para desarrollar la mínima es 3.6. Python se puede descargar desde el siguiente enlace: https://www.python.org/downloads/

Instalación

La forma más cómoda de obtener el código del proyecto es mediante *git*, para ello usar el siguiente comando:

[5] https://www.mongodb.com/

```
$ git clone <url_del_repositorio>
```

Siendo https://github.com/IvanBeke/TFG-Visor-de-espectros.git la URL del repositorio.

Ya con el proyecto descargado, el siguiente paso es instalar las dependencias. Aunque se pueden instalar sobre la instalación global de Python, es recomendable usar un entorno virtual. Para crearlo usar el siguiente comando:

```
$ python3 -m venv <nombre_del_entorno>
```

Para activar el entorno usar el siguiente comando:

```
$ source <ruta/del/entorno>/bin/activate
```

Para instalar las dependencias se pueden ejecutar cualquiera de los siguientes comandos:

```
$ bash install_dependencies.sh
$ pip install -r requirementes.txt
```

Claves OAuth de Google

El inicio de sesión con Google usa la API de OAuth, para que funcione se necesitan el *client_id* y *client_secret*. Estos valores se obtienen desde https://console.developers.google.com/apis/credentials.

Variables de entorno

Para que la aplicación funcione correctamente hay que definir las siguientes variables de entorno:

- `GOOGLE_OAUTH_CLIENT_ID`: *client_id* obtenido anteriormente.
- `GOOGLE_OAUTH_CLIENT_SECRET`: *client_secret* obtenido anteriormente.
- `UPLOAD_FOLDER`: directorio donde se van a guardar los archivos que se suban a la aplicación.
- `SECRET_KEY`: clave necesaria por Flask.

- `OAUTHLIB_RELAX_TOKEN_SCOPE`: recomendable poner este valor a 1[6].
- `ENVIRONMENT`: entorno en el que se está actualmente, development, production.
- `DATA_MONGODB_HOST`: dirección para conectarse a MongoDB.

Ejecución

Para ejecutar la aplicación es necesario tener el entorno virtual activo y ejecutar el siguiente comando:

```
$ python start.py
```

Esto lanza el servidor Flask en el ordenador local y puerto 5000, se accede desde https://127.0.0.1:5000/. Al entrar seguramente el navegador avise de certificado incorrecto, esto se debe al uso de un certificado local, ya que OAuth requiere el uso de `https`.

D.5. Pruebas del sistema

Las pruebas del sistema se encuentran dentro del directorio `tests`. Se han codificado pruebas unitarias para probar los modelos de la aplicación, se encuentran en el fichero `unitarios`. Se ha usado el *framework Unittest* [8], que viene por defecto instalado con Python, por lo que no es necesario instalar ninguna dependencia.

Para ejecutar las pruebas, dirigirse al directorio raíz del proyecto y ejecutar el siguiente comando:

```
$ python -m unittest tests.unitarios
```

[6] https://flask-dance.readthedocs.io/en/v0.14.0/quickstarts/google.html

Apéndice E

Documentación de usuario

E.1. Introducción

En este apéndice se explica los requisitos que debe cumplir el usuario para ejecutar la aplicación, como lanzarla y como usarla.

E.2. Requisitos de usuarios

Al tratarse de una aplicación los requisitos que debe cumplir el usuario son los siguientes:

- Navegador web instalado.
- Cuenta de Google activa.
- JavaScript avtivo en el navegador.
- *Cookies* activas en el navegador.

La aplicación se ha probado en los siguientes navegadores y se certifica que funciona:

- Google Chrome 67.0.3396.99.
- Chromium 66.0.3359.181.
- Mozilla Firefox 60.0.2.

Aunque la aplicación funciona en dispositivos móvil no se recomienda hacerlo debido a que la visualización sería demasiado pequeña y no estarían disponibles las acciones sobre los gráficos.

E.3. Instalación

Debido a que se proporciona una aplicación web no es necesario instalarla para poder usarla. Sin embargo, si se quiere proceder a la instalación se pueden seguir las instrucciones en la sección D.4.

De cara a probar la aplicación con ejemplos ya cargados se proporciona un cuenta con ejemplos cargados para su uso. El usuario proporcionado es `tfg.visor.ejemplos@gmail.com` y su contraseña es `tfg_visor_ejemplos`.

Hay que tener en cuenta que al ser una cuenta de prueba la pueden usar varias personas por lo que se recomienda encarecidamente no borrar los espectros ni los datasets que se encuentran subidos.

E.4. Manual del usuario

En esta sección se enseña al usuario como manejar la aplicación.

Inicio

Nada más entrar a la aplicación se muestra la bienvenida. Si no se ha iniciado sesión aparece el mensaje de la imagen E.1.

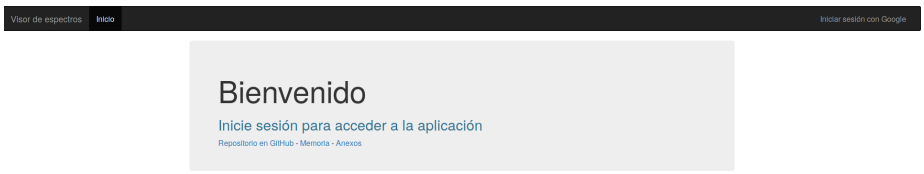

Figura E.1: Bienvenida

Después de iniciar sesión (botón en la esquina superior derecha), o si no se había cerrado sesión anteriormente, se muestra el mensaje de la imagen E.2.

Figura E.2: Bienvenida después de iniciar sesión

Las opciones presentadas en la barra de navegación y en las tarjetas situadas debajo del mensaje son las mismas, con la excepción de que en las tarjetas aparece una pequeña descripción de la acción.

Mis ficheros

Pulsando en "Mis ficheros" nos aparece la página con todo lo que tenemos asociado en nuestra cuenta (ver figura E.3), organizado en tres columnas, datasets, espectros y clasificadores. Para cada entidad se muestra el nombre, los comentarios y las opciones disponibles.

Figura E.3: Mis ficheros

Subir dataset

Pulsando en "Subir dataset" nos dirigimos a la página en la que podemos subir en conjunto de espectros. A la izquierda se muestran las instrucciones a seguir y unas notas respecto al formulario, presente a la derecha. En la imagen E.4 se puede ver la página con el formulario completo.

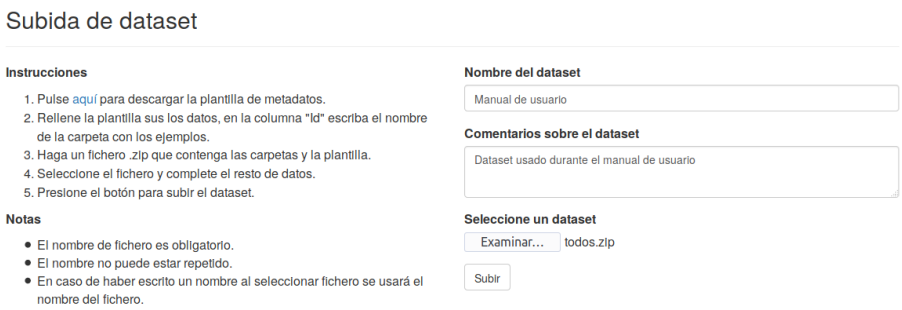

Figura E.4: Página para subir un dataset

Después de pulsar el botón "Subir" se muestra un mensaje de espera mientra se procesa la petición (ver figura **??**).

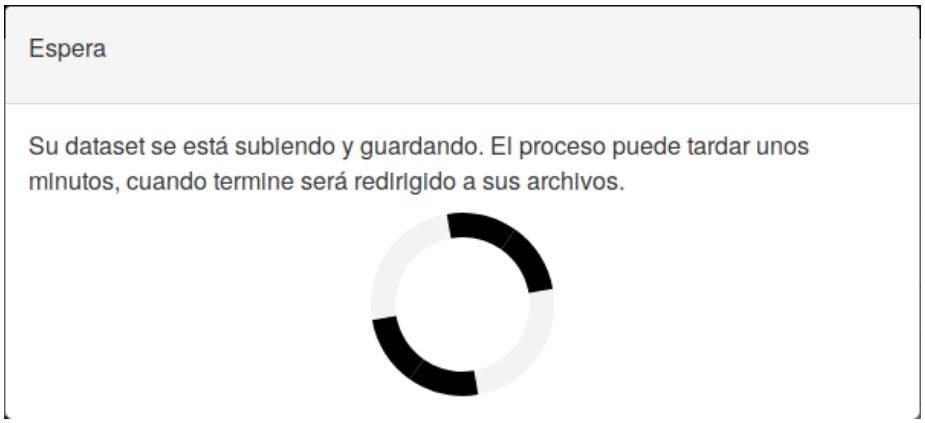

Figura E.5: Mensaje de espera al subir un dataset

Cuando la subida del fichero termine, mostrará la página con los ficheros guardados, indicando que el dataset se ha subido correctamente.

Subir espectro

Pulsando en "Subir espectro" nos dirigimos a la página en la que podemos subir un espectro. A la izquierda se muestran las indicaciones respecto al formato requerido en el fichero del espectros. En la imagen E.6 se puede ver la página con el formulario completo.

Figura E.6: Página para subir un espectro

Después de pulsar el botón "Subir" se muestra un mensaje de espera mientra se procesa la petición (ver figura **??**). Sin embargo, como esta operación suele tardar poco tiempo, el mensaje no llega a leerse.

Figura E.7: Mensaje de espera al subir un espectro

Cuando la subida del fichero termine, mostrará la página con los ficheros guardados, indicando que el espectro se ha subido correctamente.

Eliminado

Cuando se quiera eliminar cualquiera de los elementos guardados, es suficiente con presionar el botón "Eliminar". A continuación se muestra una ventana para confirmar la acción. Si se acepta la eliminación, se borra el elemento de la cuenta y se muestra un mensaje indicando que la acción se ha completado.

Visualización y procesamiento

Para visualizar un dataset o espectro hay que pulsar en el botón "Visualizar" en el elemento que se desee. Esta acción nos redirige a la vista de visualización.

Datasets

La visualización de datasets se compone de cuatro elementos (figura E.8), la tabla con los espectros contenidos, los controles de procesamiento, la visualización del espectro original y la visualización del espectro procesado. Se proporciona una ayuda integrada en esta vista.

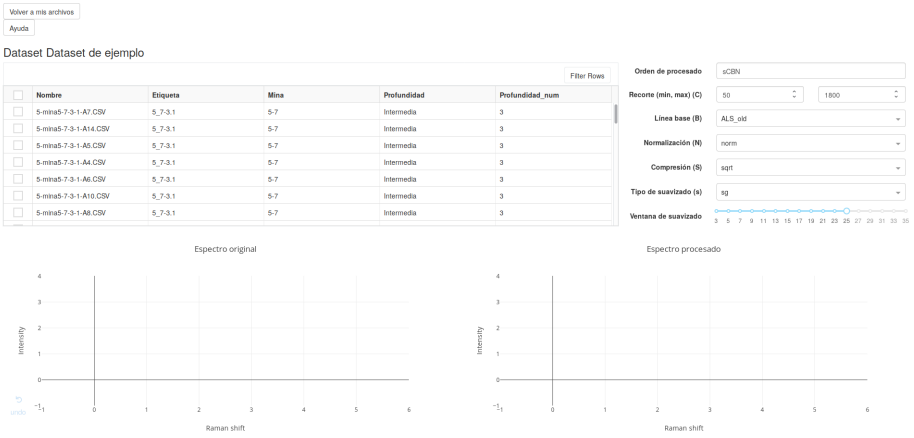

Figura E.8: Visualización de dataset

Para visualizar un espectro hay que seleccionarlo en la tabla. Pasados unos segundos, el espectro original y procesado aparecen en sus gráficos correspondientes. La aplicación soporta la visualización de varios espectros a la vez para poder compararlos (ver figura E.9).

Figura E.9: Visualización de dataset con espectros

El espectro procesado se actualiza automáticamente al modificar las opciones de procesamiento.

Espectro

La visualización de espectro se compone de tres elementos (ver figura **??**) (figura E.10), los controles de procesamiento, la visualización del espectro original y la visualización del espectro procesado. Se proporciona una ayuda integrada en esta vista.

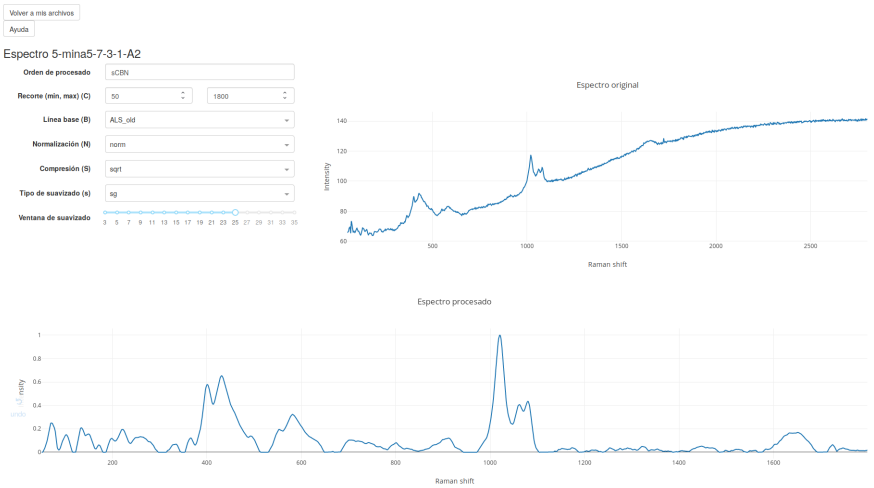

Figura E.10: Visualización de espectro

El espectro procesado se actualiza automáticamente al modificar las opciones de procesamiento.

Creación de modelos

Para crear un modelo primero hay que escoger que dataset se quiere usar como referencia. Una vez decidido, se pulsa en el botón “Crear modelo”, esto redirige a la página de creación de modelos (ver figura E.11).

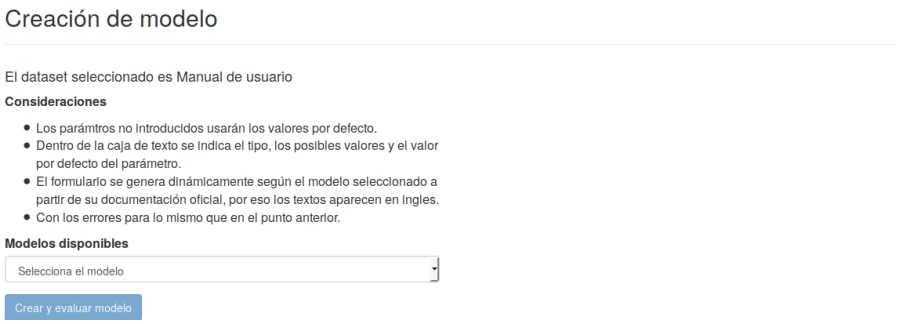

Figura E.11: Página de creación de modelos

Para poder crear el modelo hay que seleccionar uno de los disponibles en el desplegable, esta acción hace visible un formulario con los posibles parámetros del modelo (ver figura E.12 El formulario se actualiza cada vez que se cambia el modelo seleccionado.

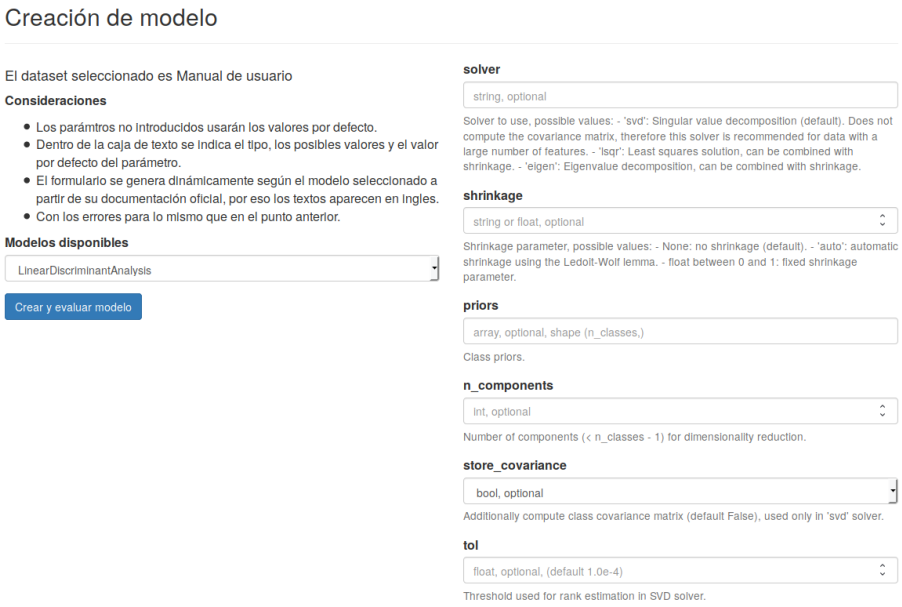

Figura E.12: Formulario de creación de modelos

Cuando se haya terminado de rellenar el formulario, todos los campos son opcionales, hay que pulsar el botón “Crear y evaluar modelo”. Esto crea el modelo con los parámetros introducidos, lo entrena con el dataset seleccionado y lo evalúa. Mientras se realiza este proceso, se le muestra un mensaje de espera al usuario hasta que se completa la acción (ver figura E.13).

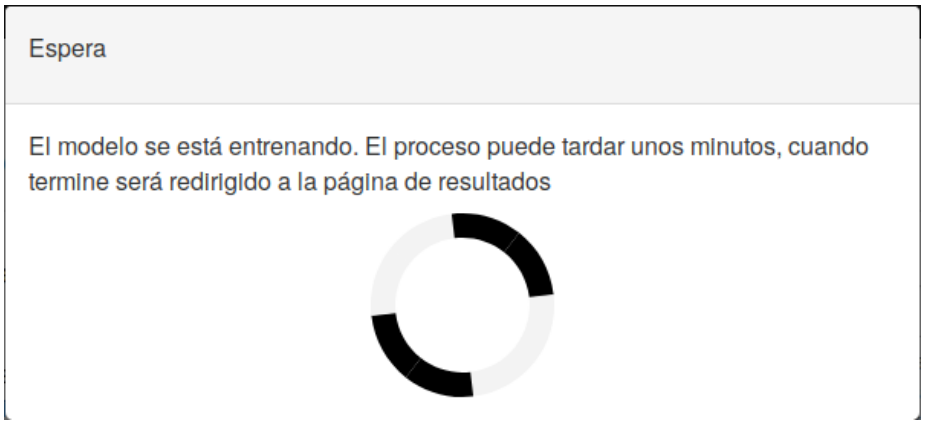

Figura E.13: Mensaje de espera en la creación de modelos

Cuando se termina el entrenamiento, se muestra una página con los

resultados de la evaluación y un formulario en caso de querer guardar el clasificador (ver figura E.14). Ambos botones, “Guardar” y “Descartar”, llevan de vuelta a la página con los ficheros guardados, pero al pulsar el de guardar, guarda el clasificador en la cuenta del usuario mientras que pulsar el botón de descarte no.

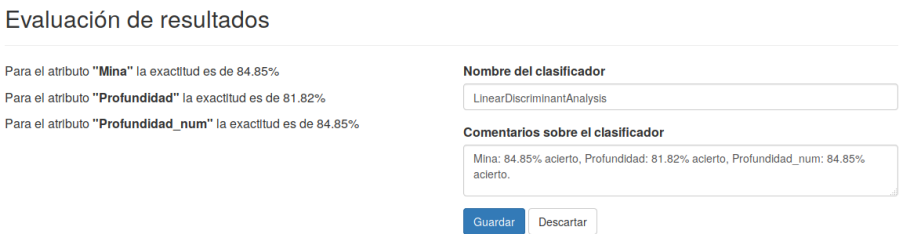

Figura E.14: Resultados de la evaluación

Bibliografía

[1] Digital Ocean Pricing. https://www.digitalocean.com/pricing/.

[2] GitKraken Pricing. https://www.gitkraken.com/pricing.

[3] Gnu general public license v3.0. https://www.gnu.org/licenses/gpl-3.0.en.html.

[4] How to choose a license for your own work. https://www.gnu.org/licenses/license-recommendations.en.html.

[5] Nanobox documentation. https://docs.nanobox.io/.

[6] PyCharm: JetBrains toolbox Subscription. https://www.jetbrains.com/pycharm/buy/#edition=personal.

[7] Seguridad Social: Bases y tipos de cotización 2018.

[8] Unit testing framweork. https://docs.python.org/3/library/unittest.html.

[9] Wikipedia. Freemium — wikipedia, la enciclopedia libre. https://es.wikipedia.org/w/index.php?title=Freemium&oldid=106331717, 2018. [Internet; descargado 27-junio-2018].

[10] Wikipedia. Modelo–vista–controlador — wikipedia, la enciclopedia libre, 2018. [Internet; descargado 27-junio-2018].