UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

TFG del Grado en Ingeniería Informática

Herramienta docente para la visualización en Web de algoritmos de aprendizaje Semi-Supervisado Documentación técnica

Presentado por David Martínez Acha
en Universidad de Burgos — 15 de abril de 2023
Tutor: Álvar Arnaiz González
Cotutor: César Ignacio García Osorio

Índice general

Índice de figuras

Índice de tablas

Apéndice A

Plan de Proyecto Software

A.1. Introducción

En el presente apartado de los anexos se analizará la gestión del proyecto software desarrollado. Este proyecto será organizado mediante la metodología Scrum en la que el trabajo estará dividido en Sprints. Por cada Sprint se realiza una reunión para la revisión del avance y los objetivos para el siguiente. Con esta metodología se mantendrá en todo momento lo que se conoce como *Product Backlog* que es una lista de las tareas a realizar. Esta lista será actualizada, en principio, en cada reunión para así mantener un desarrollo constante. Las reuniones en un principio se realizan cada dos semanas, intensificando a cada semana en el momento del inicio del periodo temporal del segundo cuatrimestre.

El objetivo de este plan es servir como herramienta para registrar el avance del proyecto y también para poder cumplir con el objetivo final del desarrollo.

A.2. Planificación temporal

La planificación temporal se comenzó mediante Sprints de dos semanas. En la presente sección se comentará el desarrollo realiza en cada uno de ellos.

Sprint 0

Desde el punto de vista temporal corresponde desde el inicio del curso del primer cuatrimestre académico (septiembre) hasta el Sprint 1. El día 15 de septiembre se tuvo la primera reunión con los tutores sobre el trabajo presente donde se establecieron las líneas generales y temática sobre el mismo.

Se creó el repositorio del TFG en Github: https://github.com/dma1004/TFG-SemiSupervisado y se añadió la plantilla de la documentación.

Sprint 1

Corresponde con el periodo temporal del 5 al 19 de octubre de 2022.

El mismo día 5 tuvo lugar una reunión de seguimiento del trabajo. Durante el sprint se realizaron unos arreglos de la plantilla y una lectura de conceptos teóricos para posteriormente añadirlos a la documentación. Concretamente se crearon las tareas «Añadir conceptos teóricos aprendizaje» y «Trabajos relacionados» a día 9 de octubre.

Sprint 2

Corresponde con el periodo temporal del 19 de octubre al 2 de noviembre de 2022.

Durante el sprint se implementó un prototipo del algoritmo Self-Training en el que posteriormente se hicieron unas correcciones en el código. También se comenzó con la redacción de conceptos teóricos (tarea «In progess»), concretamente, sobre el aprendizaje automático.

Sprint 3

Corresponde con el periodo temporal del 16 al 30 de noviembre de 2022.

Durante el sprint se aumentaron los conceptos teóricos sobre el aprendizaje supervisado, no supervisado y semi-supervisado. Se refactorizó el prototipo para su documentación (PEP), evitar datos duplicados y modularizando el código.

La memoria fue parcialmente modificada basándose en las correcciones propuestas de los tutores.

Sprint 4

Corresponde con el periodo temporal del 25 de enero al 1 de febrero de 2023. En este momento las duraciones de los Sprints cambiaron a una semana, iniciando así el periodo temporal real del desarrollo del proyecto (segundo cuatrimestre)

Durante el sprint se retomaron las tareas y el desarrollo general del proyecto. Se mejoró el algoritmo de `SelfTraining` que estaba como prototipo y se avanzó en la tarea de primera aproximación en la aplicación. mediante Flask. Sobre esto último, se creó una visualización del proceso de entrenamiento muy básica por cada iteración.

Se creó un prototipo del algoritmo `CoTraining` sin cumplir con todas sus condiciones que posteriormente se completaron a falta de revisión.

Sobre estos dos algoritmos se propuso la versión 1.0.

Continuando con la Web, se realizó la interfaz general funcional. Incluye:

- Página de Inicio donde seleccionar el algoritmo.
- Página de subida de archivos en formatos ARFF y CSV de los conjuntos de datos
- Páginas correspondientes para SelfTraining y CoTraining: Cada una tiene sus parámetros específicos con la posibilidad de seleccionar si utilizar PCA (Principal Component Analysis) o dos componentes que elija el usuario.
- Página de visualización del algoritmo (su entrenamiento): Se tiene la vista principal que será común a todos los algoritmos (con algunas variaciones en caso necesario) con la posibilidad de avanzar en la visualización (con controles) y barra de progreso. Desde el punto de vista del gráfico los colores están automatizados dependiendo del número de clases, leyenda y etiqueta de ejes.

En el servidor (Flask) a nivel de programación se añadieron los «endpoints» correspondientes (subida, configuración, visualización...) y un control de acceso a las páginas muy básico (por ejemplo, si no se configuró el algoritmo, no se puede visualizar y le redirecciona a la configuración con un mensaje de error)

Sprint 5

Corresponde con el periodo temporal del 1 al 8 de febrero de 2023.

En la reunión del 1 de febrero se revisó lo realizado en el anterior y se fijaron una serie de mejoras/modificaciones y nuevas tareas:

1. Modificación de los algoritmos para trabajar con la convención de «-1s» en el conjunto de datos para los datos no etiquetados. Así el usuario podrá subir un archivo ya *Semi-Supervisado*.
2. Permitir al usuario seleccionar los porcentajes de no etiquetados y de test (para las futuras estadísticas).
3. Sobre la página general de la visualización de los algoritmos: volver a la configuración, el «feedback» de la iteración actual y el nombre del conjunto de datos utilizado.
4. Del gráfico de la visualización: Diferenciar en el algoritmo CoTraining cuál de los dos clasificadores han etiquetado cada punto y los puntos «etiquetados» en la iteración 0 deben mostrarse de forma diferente.
5. Avanzar con los trabajos relacionados.
6. Avanzar con la documentación teórica y anexos.

El punto 1 ha llevado unas 12 horas de compresión y desarrollo. Esto es debido a que los dos algoritmos implementados hasta ahora debían ser modificados para trabajar con la nueva convención. Además, el problema principal fue (aunque no implementado en este Sprint) dejar preparado una forma de carga del conjunto de datos que permita tratar datos no etiquetados («?» por ejemplo en el caso de ARFF) pues además de los algoritmos (su correcto funcionamiento) se han probado con ficheros. También conllevó la creación de un codificador de etiquetas propio para ignorar los no etiquetados en clases categóricas (y no realizar la conversión en esos casos)

El punto 2 volvió a causar bastantes problemas tanto en la ejecución de los algoritmos como en la Web. Hasta el momento, el usuario no seleccionaba los porcentajes de las divisiones. Al incluir esto, los algoritmos ya no se encargan de esta tarea y había que modificar tanto los algoritmos como aquellas rutas de la Web que debían encargarse de esto. Aproximadamente 4 horas.

El punto 3 no resultó demasiado difícil más allá de seguir habituándose a JavaScript/HTML. Unas 3 horas.

El punto 4 requirió unas 10 horas, en un principio se perdió mucho tiempo intentando solucionarlo de una forma que resultó inútil, pero finalmente ahora en el algoritmo se diferencian los datos clasificados por cada uno.

Los trabajos relacionados (no terminados) se realizaron en varios días con un tiempo aproximado de 6 horas.

Sprint 6

Corresponde con el periodo temporal del 8 al 15 de febrero de 2023.

En la reunión del 8 de febrero se revisó lo realizado en el anterior y se comentaron algunas tareas a realizar:

1. En la línea del anterior, los algoritmos deben poder ejecutarse directamente con conjuntos de datos semi-supervisados.
2. Permitir al usuario introducir ese tipo de conjuntos de datos.
3. Realizar alguna visualización de estadísticas.
4. Valor por defecto en las configuraciones.
5. Sobre el gráfico: mejorar la diferenciación de los puntos, información útil en los «tooltips» y colocación leyenda.
6. Avanzar con los trabajos relacionados.
7. Avanzar con la memoria y anexos.

Los puntos 1 y 2 estaban muy avanzados gracias al trabajo adicional del sprint anterior, ya que ya estaba prácticamente implementada la forma en la que detectar datos no etiquetados de forma automática. Unas 5 horas para terminar de implementar, corregir errores sobre la marcha y realizar alguna prueba confeccionando ficheros semi-supervisados.

Al realizar las pruebas anteriores se encontró un error en la visualización provocando que los datos que, por la iteración máxima, no se habían clasificado ni siquiera eran retornadas a la Web. Entre descubrir cómo hacerlo y sus modificaciones se tardó unas 3 horas.

El punto 3 fue el más complicado, pese a que era una idea sencilla, se optó por visualizar la gráfica de la evolución de la precisión. Cada punto del gráfico está unido por una serie de líneas. Este tipo de gráficos (según la documentación) se suelen hacer mediante «paths» o caminos, que son una

única línea, pero como en este caso era necesario no visualizar todo, sino por cada iteración, no se encontró una solución rápida. Unas 6 horas para probar muchas posibilidades hasta encontrar la que funcionó, acoplarla a los controles del paso de iteración e incluir alguna animación.

Adicionalmente se retocó por completo toda la Web mediante los estilos de `Bootstrap` para establecer ya una base vistosa y bonita. Unas 5 horas (la mayor parte del tiempo para probar y adquirir algo de soltura con estos estilos).

Sprint 7

Corresponde con el periodo temporal del 15 al 22 de febrero de 2023.

Puntos a desarrollar:

1. Implementación Democratic Co-Learning.
2. Profiling (tiempos de ejecución).
3. Estadísticas en la aplicación.
4. Test de las implementaciones.
5. Avanzar con la memoria y anexos.

La implementación del algoritmo Democratic Co-Learning supuso unas 14 horas divididas en varios días. Al principio se dedicó un tiempo para leer el artículo en el que se presentaba su implementación en forma de pseudocódigo junto con sus explicaciones teóricas. La realidad es que en primera instancia parecía algo fácil de realizar y entender, pero una vez comenzada la implementación se encontraban muchas alternativas a la hora de resolverlo. Además, pese a que en el artículo estaba bien explicado, el formato de pseudocódigo (en el archivo encontrado) las indentaciones eran incorrectas y se perdió mucho tiempo comprobando si era una interpretación errónea o si realmente era un fallo.

Se realizaron algunas pruebas de rendimiento para comprobar si los algoritmos tardaban demasiado con conjuntos de datos muy grandes (5 000 instancias). Se observó que, dada la configuración que se tenía, tardaba alrededor de 40-50 segundos en terminar la ejecución. Es por esto que para este Sprint se añadió la tarea de hacer un pequeño estudio dedicado a medir los tiempos de ejecución para ver qué se podía optimizar. Este proceso fue

de unas 2 horas y el resultado fue que el código implementado no afectaba mucho, eran los propios algoritmos de entrenamiento de los clasificadores de Scikit-Learn los que tardaban tanto. Por ejemplo, para un estimador gaussiano el tiempo se reducía drásticamente.

Para el caso de las estadísticas, se modificaron un poco las plantillas y la generación de sus gráficas para incluir más y revisarlas en la reunión. Unas 2 horas.

Los tests son una parte importante para validar que el comportamiento que se espera de la implementación sea el correcto. Se realizaron unos casos de pruebas sobre las utilidades que se usan a lo largo de todo el proyecto con la intención de encontrar errores (todo esto sin ver cuál es el resultado y replicarlo en los casos, sino realizar los casos basándose en lo que se espera de esas utilidades). Se tardó unas 4 horas en realizar todos los tests.

Sprint 8

Corresponde con el periodo temporal del 22 de febrero al 1 de marzo de 2023. Además, aprovechando la herramienta Zenhub, se modificó la duración de los Sprints también en ella para poder extraer los gráficos del trabajo realizado.

Puntos a desarrollar:

1. Intervalo de confianza en Democratic Co-Learning.
2. Control reetiquetado en Democratic Co-Learning.
3. Correcciones sobre memoria y anexos.
4. Gráfico de estadísticas unificado.
5. Internacionalización Web.
6. Visualización principal de Democratic Co-Learning en la aplicación.

El primer punto fue muy sencillo, se proporcionó la implementación de los intervalos de confianza tanto de Álvar Arnaiz González como de César Ignacio García Osorio (tutores) y finalmente se implementó esta segunda.

El control del reetiquetado fue mal estimado (en puntos de historia), al principio parecía una idea sencilla, pero por un error de pensamiento, la implementación que se realizó en un principio no funcionaba. Como cada

clasificador tiene su propio conjunto de entrenamiento, se estaba tomando que el índice de la instancia sumado a la longitud de su conjunto de entrenamiento era la posición en la que actualizar la etiqueta, obviamente esto no funciona pues esa suma puede superar la longitud del propio conjunto. Había que almacenar la posición concreta sin realizar esos cálculos y se optó por un diccionario de «ids» para cada clasificador en el que el valor es la posición en las etiquetas del conjunto del clasificador.

Las correcciones de la documentación se incorporaron según las indicaciones de Álvar Arnaiz.

El gráfico de estadísticas fue unificado, permitiendo seleccionar al usuario mediante unos «checkboxes». Aproximadamente unas 4 horas para generar el formulario correspondiente que controle la aparición de cada línea y controlar los eventos de las iteraciones (por ejemplo, añadir siguiente punto solo si está activado su check).

En cuanto a la Internacionalización, al principio resultó sencillo, ya que gracias a Babel (Flask-Babel) detecta automáticamente las cadenas de texto dentro de «gettext», sin embargo, al indicar las traducciones en JavaScript, el intérprete tomaba «gettext» como una función que al no estar definida, lanzaba error. Al final se generó una función en la plantilla principal de tal forma que actúe como «gettext», llamada desde los distintos scripts.

La visualización de Democratic Co-Learning no dio tiempo a implementarse en este Sprint.

Además, a partir de este Sprint se incorporan todas las tareas en Zenhub para la generación de los gráficos Burndown (ver Imagen A.1).

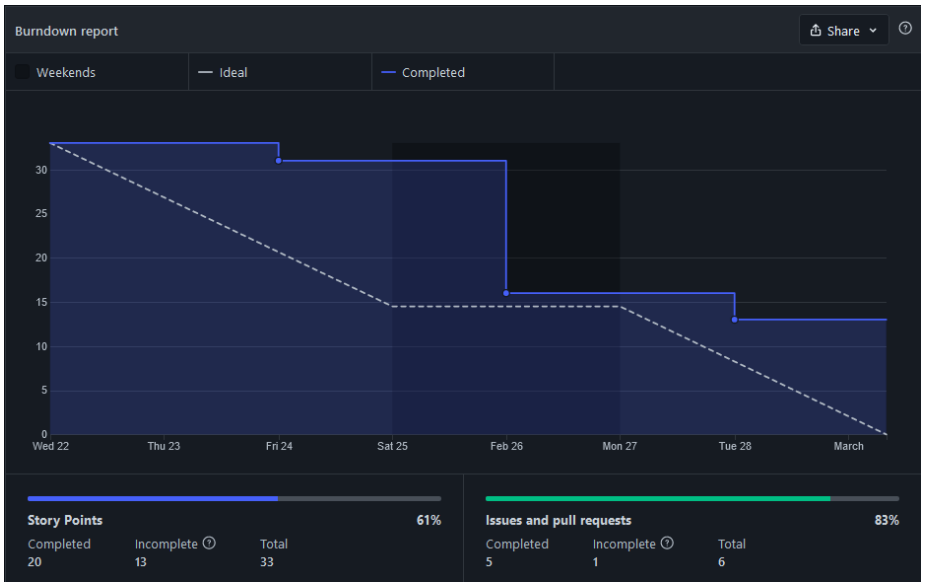

Figura A.1: Burndown chart del sprint 8.

Sprint 9

Corresponde con el periodo temporal del 1 de febrero al 8 de marzo de 2023.

Puntos a desarrollar:

1. Visualización principal de Democratic Co-Learning en la aplicación.
2. Formulario de los parámetros de los clasificadores.
3. Estadísticas generales en Democratic Co-Learning.
4. Estadísticas específicas en Democratic Co-Learning
5. Condición de parada reetiquetado

La visualización principal fue relativamente sencilla pues en realidad es muy similar a Co-Training. Hubo algún ajuste adicional para generar la información de la *tooltip* al pasar por encima de un punto. Como en cada posición podía haber varios clasificadores había que mostrar esa información.

Para el formulario de los parámetros se consideró el uso de Flask-WTF que al final se descartó. Se tenía como punto de partida utilizar un JSON del que leer los parámetros, así que lo primero fue pensar una forma sencilla de

codificarlos, con la información necesaria de las entradas («type»,«step»...). La ventaja de JSON es que aparte de que la lectura es muy sencilla, son diccionarios, muy fáciles de utilizar (tanto desde Python como desde las propias plantillas/JavaScript). Para generar el formulario se pensó en hacerlo de la forma más automática posible para así solo tener que cambiar el JSON, esto se hizo con un método de JavaScript que para cada clasificador genera el formulario correspondiente con los parámetros del JSON. El tiempo total fue unas 6 horas, pues también se perdió tiempo debido a que no era posible seleccionar elementos del DOM (pues no estaba cargado) y los elementos debían seleccionarse mediante CSS (algo que no se sabía por desconocimiento de JavaScript).

Se añadieron las estadísticas generales de Democratic Co-Learning de forma muy sencilla pues era exactamente igual que Co-Training (y Self-Training) esto se realizó añadiendo el *DataFrame* en el propio algoritmo con las estadísticas deseadas (como el resto de los algoritmos).

Había mucho código muy parecido o exactamente igual en las plantillas de los algoritmos, así que aprovechando esta tarea también se automatizaron por completo las estadísticas. Se pensó de tal forma que solo con las columnas de los *DataFrames* que retornan los algoritmos, la Web ya se encargue de generar el resto. A la vez que esto (e iniciando la tarea de las estadísticas individuales), otro punto importante que se tuvo en cuenta eran las futuras estadísticas individuales para Democratic Co-Learning, todas las funciones fueron transformadas para trabajar con elementos del DOM específicos, de esta forma al indicarle por ejemplo un «DIV», se generen las estadísticas sobre ese elemento. En total unas 8 horas.

Para estas estadísticas individuales, aparte de las funciones generalizadas se creó una nueva que sobre un elemento (un «DIV») se generase un selector con el nombre de los clasificadores que intervienen en el algoritmo junto con los contenedores dentro de él para las estadísticas de cada clasificador. Finalmente, se utilizan las funciones de la tarea anterior para añadir a esos contenedores nuevos los gráficos individuales. Unas 4 horas.

Sobre el último punto, se comentó que una etiqueta que es reetiquetada al mismo valor no debe «contar» como mejora (o cambio en el algoritmo). Si no se realiza así, el tiempo de ejecución aumenta demasiado.

El gráfico Burndown se visualiza en la imagen A.2.

Figura A.2: Burndown chart del sprint 9.

Sprint 10

Corresponde con el periodo temporal del 8 de febrero al 15 de marzo de 2023.

Puntos a desarrollar:

1. Comparación sslearn
2. Refactorización de plantillas.
3. Refactorizacion Javascript.
4. Refactorización Flask (app).
5. Añadir zoom a los gráficos.
6. Conceptos teóricos.

En este Sprint no se contaba con demasiado tiempo así que aunque sí se realizó alguna tarea para añadir funciones, la idea era dedicarlo a mejorar el código y mantenimiento.

En la reunión del final del Sprint anterior se comentó el cómo se estaba validando los algoritmos y hasta ese momento solo se habían probado las

utilidades. Se sugirió compararlo contra *sslearn*, una biblioteca de José Luis Garrido-Labrador. Como primera aproximación se realizó una validación cruzada completamente manual en la que se ejecutaba al mismo tiempo las dos implementaciones de los distintos algoritmos (de momento sin extraer conclusiones). Se tardó unas 3 horas.

El segundo y tercer punto se iniciaron por separado, pero llegó un momento en el que las funciones que se había creado en JavaScript, si se modificaban un poco, podrían simplificarse a su vez las plantillas. Todos los métodos los gráficos estaban separados por cada uno de los algoritmos, esto lo hacía muy engorroso porque incluso algún método tenía el mismo nombre. Se modificaron las funciones de tal forma que fuesen específicas para cada algoritmo para así juntarlas en un único Script. Por parte de las plantillas, como había partes repetidas se crearon macros y se identificó una parte común a todos los algoritmos en su configuración, esto se añadió a la base de las configuraciones. Todo esto llevó unas 4 horas.

En `Flask` se tenían varios problemas. El primero era que las visualizaciones de cada algoritmo tenían un *endpoint* particular, pero esto no era necesario si se hacía un método para cada la obtención de los parámetros de cada algoritmo. Cuando se crearon estos métodos se generó código muy parecido porque todos ellos tenían dos partes: una en la que se obtenían los parámetros que no eran de los clasificadores base y otra en la que se incorporaban esos parámetros de los clasificadores. La primera parte es particular para cada algoritmo, pero la segunda es común a todos. Se creó otro método para ese paso. Por último, por cada algoritmo se tiene un *endpoint* para la ejecución y obtención de la información de entrenamiento, pero había una parte que todos hacían prácticamente igual. Se creó un método que engloba: carga de datos, separación de los datos para entrenamiento, entrenamiento y obtención de los algoritmos y la aplicación de PCA (o no).

La visualización principal de los algoritmos tenía el problema de que cuando los puntos estaban demasiado cerca, no se llegan a apreciar individualmente. Esto se solucionaría aplicando *zoom* al gráfico. Pese a que en cuanto a código no fuese un desarrollo largo, fue una tarea compleja. Se probaron unas tres implementaciones parecidas a ejemplos encontrados en la documentación, pero en todas ellas se perdía la ayuda contextual del *tooltip*. Al final se optó por volver a empezar de cero con el conocimiento adquirido y junto con un último ejemplo [1] y ciertas modificaciones se consiguió. Posteriormente se añadió el reinicio del *zoom* para volver a la posición original.

[1] Ejemplo D3: https://observablehq.com/@d3/zoom-with-tooltip

El proceso duró unas 5 horas pues cada intento parecía ser definitivo, pero al final siempre había ciertos límites.

Al final del Sprint se añadieron los conceptos teóricos de Democratic Co-Learning (conceptos y pseudocódigos).

Aparte de estas tareas se realizaron pequeñas modificaciones: se completó el estilo adaptable («responsive»), se arreglaron pequeños bugs de visualizaciones y ayuda contextual, actualización de traducciones y se añadió un fichero de prueba que el usuario puede descargar si solo quiere probar la aplicación.

El gráfico Burndown se visualiza en la imagen A.3.

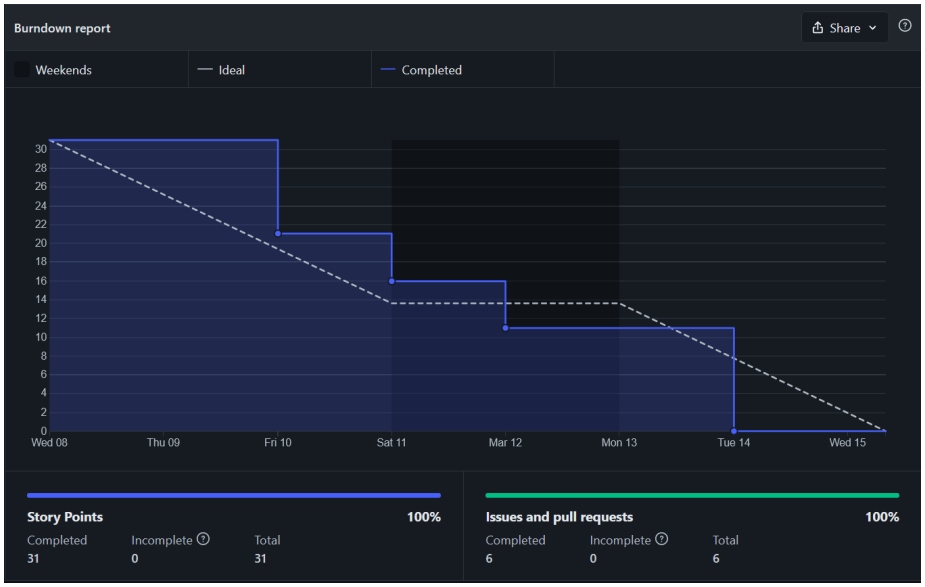

Figura A.3: Burndown chart del sprint 10.

Sprint 11

Corresponde con el periodo temporal del 15 al 22 de marzo de 2023.

Puntos a desarrollar:

1. Arreglos en Democratic Co-Learning.
2. Controlar los límites en los parámetros de los clasificadores.
3. Modificar las estadísticas generales.

4. Invertir controles en las estadísticas específicas.
5. Anexos: Manual del programador.
6. Modificar validación cruzada.
7. Comparación exhaustiva contra sslearn.
8. Añadir validación de los algoritmos a la memoria.

En Democratic Co-Learning se estaba obviando el caso en el que si una instancia ya estaba etiquetada y esta es reetiquetada, pero además cambiando de etiqueta, no se consideraba como cambio en el algoritmo. Se ha añadido esta casuística. Además, Álvar sugirió en el pseudocódigo de la memoria hacer el método de combinación de hipótesis (predicción) para una sola instancia. Esto se ha realizado así en la propia implementación (y así el método `predict` se encarga de iterar sobre un conjunto de instancias).

Al JSON que codifica los parámetros de los clasificadores base se añadieron los controles de mínimo y máximo. Esto es porque hay algunos de sus parámetros que requieren rangos específicos. Ahora la web (JavaScript) genera el formulario de configuración teniendo en cuenta estos límites.

En la reunión del Sprint anterior se sugirió modificar la visualización de estadísticas. Para el gráfico general (de estadísticas) no tenía sentido poder ocultar o no cada una de las estadísticas así que ahora siempre se muestran todas ellas. Particularmente para Democratic Co-Learning, las estadísticas individuales estaban manejadas mediante un selector (para seleccionar el clasificador base) y unos `checkboxes` para seleccionar las estadísticas a mostrar. Pero tiene mucho más sentido que el selector sea para las estadísticas. De esta forma el usuario selecciona una estadística y en el gráfico puede comparar esa estadística para todos los clasificadores. Además, los `checkboxes` se mantienen, pero ahora sirven para elegir qué clasificadores comparar. Esto llevó unas 6 horas. La organización de las funciones estaba altamente centrada en la versión anterior, fue un proceso complicado y lioso.

Pese a que en un principio no se comentó, una vez que se terminó el punto anterior, se vio necesaria una reestructuración de la página de las visualizaciones. Se mejoraron las leyendas de tal forma que ya no estaban dentro de los SVGs, ahora están en su propio cuadro. Esto ha conseguido no preocuparse por el tamaño de las palabras, centrarla y poder organizar mejor las columnas en las que se divide esa plantilla.

Sobre el manual del programador, se añadió la estructura de directorios con el paquete `dirtree` de LaTeX, se completó el manual del programador, centrado en qué es lo que debe saber un desarrollador para continuar con el proyecto y finalmente se describió el proceso de la compilación, instalación y ejecución del proyecto. En total fueron unas 6 horas.

El proceso de validación cruzada no estaba bien enfocado, se estaba realizando de forma manual (y todos los posibles fallos que puede suponer) aunque la librería scikit-learn incorpora ya utilidades para este proceso. Concretamente se tiene un método que genera los distintos `Folds` dado un conjunto de datos. El proceso manual se sustituyó por este nuevo. Se aprovechó para guardar los resultados como CSV.

Con el proceso de validación cruzada la idea era obtener métricas para compararlas en alguna gráfica/tabla y comprobar que las implementaciones son correctas (comparadas con `sslearn`). Para ello se creó una función que recogía la información de los CSVs y dibujaba una malla con distintos gráficos. En las columnas se distribuyen los algoritmos, separadas en dos para la implementación propia y la de `sslearn`. En las filas se tiene cada estadística. Cada uno de los gráficos de esta malla es un gráfico de cajas (para mostrar mínimos, máximos, medias, medianas...).

Sobre el último punto, aunque la idea era realizarlo en este Sprint, no se estimó bien la cantidad de trabajo no pudo ser realizado. Además, en las ejecuciones del punto anterior (la comparativa) se vio que el algoritmo Co-Training estaba funcionando por debajo que el de `sslearn`. Y esto impedí además justificar en la memoria la validación de los algoritmos.

Durante estas tareas fueron surgiendo pequeños arreglos: Se colorearon las etiquetas de la ayuda contextual (`tooltip`) para que quedara claro qué etiqueta lleva cada punto (no solo el nombre) y se actualizaron las traducciones.

El gráfico Burndown se visualiza en la imagen A.4.

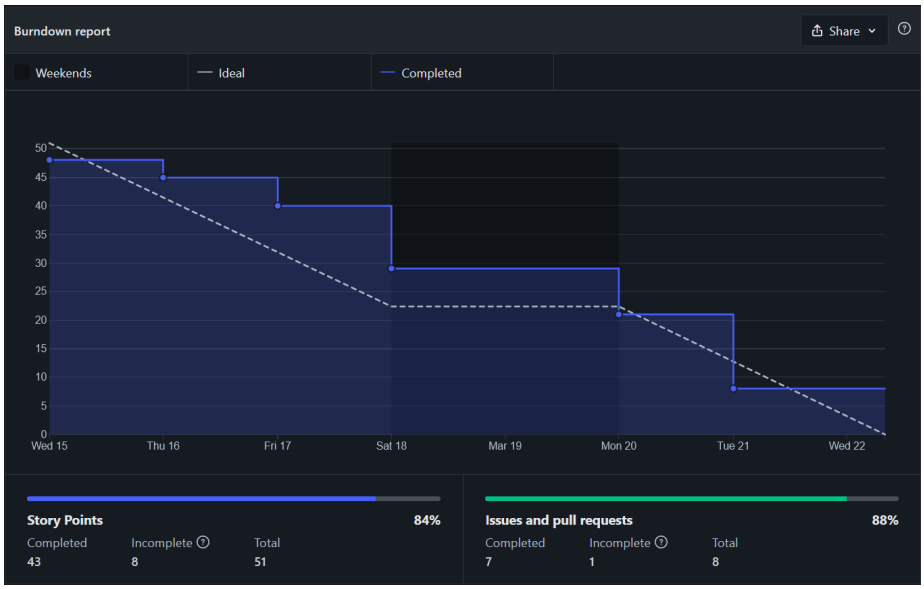

Figura A.4: Burndown chart del sprint 11.

Sprint 12

Corresponde con el periodo temporal del 22 al 29 de marzo de 2023.

Puntos a desarrollar:

1. Completar manual del programador.
2. Introducción memoria.
3. Añadir bibliotecas Web.
4. Añadir explicaciones de los algoritmos en la Web.
5. Añadir los pseudocódigos en la Web.
6. Algoritmo Tri-Training.

En el Sprint anterior se empezó el manual del programador, pero quedó pendiente la compilación, instalación y ejecución del proyecto. Lo primero que se hizo en este Sprint es finalizar este manual describiendo todos los pasos que un programador debe seguir para poner en funcionamiento la aplicación.

Para continuar con la memoria, se añadió la sección de la `Introducción` como presentación de todo este trabajo.

Las dos principales bibliotecas o recursos utilizados en la Web son `Bootstrap` y `D3.js`, el primer caso es un «framework» de CSS que ya viene con una gran cantidad de clases a utilizar. D3 es la biblioteca de JavaScript que se ha utilizado para la generación de todos los gráficos. Se añadió una descripción (y justificación) de uso en las `Técnicas y herramientas`.

Teniendo en cuanta la componente docente de este proyecto, se creyó conveniente la adición de unas pequeñas explicaciones de los algoritmos. Se añadieron en la fase de configuración de los algoritmos dado que al mismo tiempo el usuario debe configurar los parámetros y puede que le permita tener una mayor intuición a la hora de configurarlo.

En esta misma línea se añadieron los pseudocódigos en forma de imágenes, justo debajo de las explicaciones. También se creyó conveniente que el usuario pudiera verlo en la fase de visualización. Se creó un desplegable con la imagen del pseudocódigo.

Finalmente, y como desarrollo más grande de este Sprint, se creó el algoritmo Tri-Training. La primera versión (el primer `commit`) no fue validada de ninguna forma (se confirmó según se terminó). Esta versión no incluía ninguna forma de obtener el proceso de entrenamiento, simplemente era el esqueleto funcional del algoritmo. Obviamente, era claro que iba a haber algún que otro error. Al principio solo se detectó un error en el que el cálculo del error de clasificación siempre resultaba en 0. Era una cuestión de pura implementación en Python así que no fue difícil arreglarlo.

Antes de arreglar este error, la rama en la que se ejecutaba el «subsample» no se estaba realizando en ningún caso. En el momento que se arregló el anterior error se accedió a esta parte y apareció un nuevo error. Se confundió array de NumPy con lista nativa de Python y se estaban seleccionando ciertas posiciones de una lista de Python accediendo a ella con unos índices (List[indices]). La solución fue sencilla de igual manera, se recorría la lista en base la los índices y se creaba esta sublista mediante compresión de listas.

Al final de este desarrollo, se comparó con una sola ejecución con sslearn. El nuevo algoritmo tardaba mucho más tiempo que el resto (y que sslearn). Esto era porque al añadir las predicciones (cuando las otras dos hipótesis coincidían) se hacía de una en una. Esto se sustituyó para predecir todas las instancias no etiquetadas y con vectorización. El rendimiento se incrementó mucho.

Durante estas tareas fueron surgiendo otras más pequeñas: Nuevas traducciones, correcciones de memoria y anexos y una pequeña comparación contra sslearn.

La herramienta Zenhub dejó de proporcionar servicios gratuitos. Esta compañía actualizó sus planes gratuitos, ya no existe una versión de esas características. Durante este Sprint «caducó» la licencia que se tenía y ya no se podía acceder a ninguna de sus funcionalidades, tampoco a los gráficos «Burndown».

Sprint 13

Corresponde con el periodo temporal del 29 marzo al 5 de abril de 2023.

Puntos a desarrollar:

1. Configuración Tri-Training
2. Visualizar Tri-Training.
3. Eliminar todo el código duplicado restante de JavaScript.
4. Documentar JavaScript.
5. Reestructurar Flask para aplicaciones grandes.
6. Añadir media geométrica en la validación/comparación contra sslearn.

Para este Sprint el objetivo principal era dejar lista la visualización de Tri-Training en la Web.

El primer paso fue incluir las estructuras de datos necesarias en el esqueleto del algoritmo para ir almacenando la información de entrenamiento. La idea era igual a Democratic Co-Learning. La realidad es que la forma en la que se almacenan los datos es muy distinta a todas las demás. Los datos pueden ser clasificador varias veces por cada clasificador en diferentes iteraciones. Por lo tanto, para cada clasificador base se almacena una lista para las etiquetas y para las iteraciones en las que se clasifican. Las estadísticas sí que se almacenan exactamente de la misma forma que Democratic Co-Learning.

El siguiente paso fue crear la pantalla de configuración de Tri-Training, para ello simplemente se copiaron las plantillas ya existentes y la misma estructura que ya se tenía en Flask para las rutas.

Y finalmente, la pantalla de visualización de Tri-Training. Este fue uno de los pasos más complejos. Como se tenía una nueva estructura de los datos, se tenían que crear los métodos que permitiesen interpretarlos, de

forma general, los que se ha realizado es comprobar, en cada iteración, qué clasificador han añadido datos etiquetados nuevos junto con su etiqueta. Para que el usuario pudiera ver los cambios que ocurren en cada iteración, cuando pulsa en la siguiente iteración se descoloran los puntos (a gris) y después se colorean los nuevos, todo esto mediante animaciones y con un cierto tiempo para que pueda darse cuenta.

En cuanto a JavaScript, se documentó por completo todos los métodos que fueron creados para este proyecto. Además, se eliminó el código duplicado en cada fichero utilizando métodos comunes. También fue un proceso largo, sobre todo para comprobar que las modificaciones eran correctas y generales, en principio, todo parece correcto.

Se reestructuró completamente Flask, esto es porque tal y como se encontraba la aplicación, era muy básica y general. Los proyecto más grandes con Flask tienen una estructura más o menos concreta. Esta es la idea que se ha intentado aplicar, utilizando «Application Facotory» y «Blueprints». Pero sobre todo, para hacerla más mantenible y extensible. De hecho, aunque no está siendo usada, se tiene una pequeña base de datos para posibles adiciones futuras. Esta reestructuración causó muchos problemas en cuanto a las rutas de ficheros. Se tuvieron que especificar manualmente y usar la librería del sistema operativo de Python para independizar ciertos tratamientos de ficheros del sistema operativo concreto donde corra la aplicación (se ha probado en Windows y Linux).

Finalmente, se añadió la media geométrica como métrica de validación a la hora de comparar contra sslearn. Además, en Sprints anteriores se comentó la idea de utilizar `Violinplots`, que resultan más pertinentes para los casos de comparación.

A.3. Estudio de viabilidad

Viabilidad económica

Viabilidad legal

Apéndice B

Especificación de Requisitos

B.1. Introducción

Una muestra de cómo podría ser una tabla de casos de uso:

B.2. Objetivos generales

B.3. Catálogo de requisitos

B.4. Especificación de requisitos

CU-1	**Ejemplo de caso de uso**
Versión	1.0
Autor	Alumno
Requisitos asociados	RF-xx, RF-xx
Descripción	La descripción del CU
Precondición	Precondiciones (podría haber más de una)
Acciones	1. Pasos del CU 2. Pasos del CU (añadir tantos como sean necesarios)
Postcondición	Postcondiciones (podría haber más de una)
Excepciones	Excepciones
Importancia	Alta o Media o Baja...

Tabla B.1: CU-1 Nombre del caso de uso.

Apéndice C

Especificación de diseño

C.1. Introducción

En esta sección se van a describir las decisiones tomadas para llevar a cabo todos los objetivos y requisitos iniciales establecidos. Se presentará el formato que se utiliza para tratar los datos, cuál es el procedimiento interno que realiza la Web para ofrecer al usuario las visualizaciones y cómo se traduce todo ello en la arquitectura subyacente.

C.2. Diseño de datos

Información de entrenamiento de los algoritmos

Todas las visualizaciones de los algoritmos se nutren del proceso de entrenamiento. Es decir, se tuvo que diseñar una forma de aglutinar la información que ocurre durante este proceso.

Para adelantar un poco el funcionamiento de la Web, todos estos datos son transformados a JSON, el formato de texto con la sintaxis de JavaScript que resulta muy sencillo para el intercambio de datos. Que a la hora de trabajar con ello en dicho lenguaje resulta muy sencillo (como diccionarios en otros lenguajes de programación).

Pero antes de esa transformación, todos los datos son generados en Python. La estructura de datos por excelencia para almacenar muchos datos (gracias a su multitud de opciones) son los «DataFrames». Toda la información que generan los algoritmos son de este tipo.

Cada algoritmo retorna varios de estos «DataFrame», al menos uno del proceso de etiquetado y otro con las estadísticas generales. En el caso de Democratic Co-Learning y Tri-Training, estos además retornan los de las estadísticas específicas de cada clasificador base.

Etiquetas

Este «DataFrame» contiene todas las operaciones en las que los clasificadores han añadido nuevas etiquetas a instancias no etiquetadas. Por lo general indicarán los momentos en los que fueron etiquetadas (iteración) y el valor de las mismas.

Self-Training Este formato sirve como base para todos los siguientes. Es una idea muy sencilla (pues Self-Training también lo es).

La estructura de datos contendrá múltiples filas, cada un representa cada instancia de todo el conjunto de datos de entrenamiento. Lo que se entiende por conjunto de entrenamiento es, todos los datos etiquetados de los que aprenderá el clasificador base junto con los no etiquetados.

Ahora bien, la información útil que permitirá saber los momentos de clasificación y/o etiquetas estarán en las columnas:

- Columnas con los nombres de los atributos de las instancias. Por ejemplo, para el famoso conjunto de datos `Iris` se tendrán 4 columnas (una por cada atributo, **no se incluye la clase**): «sepal length», «sepal width», «petal length» y «petal width».
- Columna «iter»: Por cada fila, representa el número de la iteración en la que esa instancia fue clasificada. Si por el criterio de parada no llega a ser clasificada corresponderá con el número de iteraciones final + 1. Es decir, si en la iteración 9 el entrenamiento finalizó, corresponderá con 10. Obviamente, si la instancia es un dato inicial tendrá como iteración 0.
- Columna «target»: Representa la etiqueta de la instancia. Es importante destacar que esta siempre será de tipo entero. Esto es porque en pasos previos se eliminan los valores nominales (no están permitidos). Si tampoco llega a ser clasificado, corresponderá con «-1».

Ejemplo (entrenamiento finalizado en la iteración 9):

	petal.length	petal.width	sepal.length	sepal.width	iter	target
0	5.6	2.5	3.9	1.1	0	1
1	6.1	2.8	4.0	1.3	0	2
78	6.7	3.1	5.6	2.4	6	2
79	6.8	3.2	5.7	2.5	7	1
142	6.8	2.8	4.8	1.4	10	-1

Tabla C.1: Ejemplo de DataFrame de Self-Training

En la tabla anterior se pueden un extracto de lo que podría ser una ejecución. Las columnas de los atributos, la iteración (con iteración 10 para denotar que no fue etiquetado) y la etiqueta o target (con -1 para los que no fueron etiquetados).

Co-Training Es muy similar a Self-Trainig salvo que este algoritmo concreto envuelve dos clasificadores base. La consecuencia de esto a nivel del formato es que se necesita almacenar cuál de los dos clasificadores clasifica cada instancia. Pero más allá de esto, es el mismo formato.

Recopilando, las columnas para el «DataFrame» de Co-Training son:

- Columnas con los nombres de los atributos de las instancias.
- Columna «iter».
- Columna «target».
- Columna «clf»: Esta nueva columna indica el clasificador que le dio el valor a la etiqueta. Por convención (necesaria para otros procesos en Web) es que si el dato es inicial, «clf» valdrá «inicio», si es un dato clasificado durante el proceso valdrá «CLF(*nombre_clasificador*)» donde *nombre_clasificador* será el extraído de `scikit-learn` y si no llega a ser clasificado valdrá -1.

Ejemplo (entrenamiento finalizado en la iteración 9):

	petal.length	petal.width	sepal.length	sepal.width	iter	target	clf
0	5.6	2.5	3.9	1.1	0	1	inicio
1	6.1	2.8	4.0	1.3	0	2	inicio
78	6.7	3.1	5.6	2.4	6	2	CLF(SVC)
79	6.8	3.2	5.7	2.5	7	1	CLF(GaussianNB)
142	6.8	2.8	4.8	1.4	10	-1	-1

Tabla C.2: Ejemplo de DataFrame de Co-Training

Democratic Co-Learning Tomando como base Co-Training este algoritmo no añade ninguna columna más (ni elimina), solo modifica las existentes. Concretamente, tres última columnas («iter»,«target» y «clf») que estaban en singular ahora pasan a plural: «iters»,«targets» y «clfs».

La razón es simplemente para mantener una lógica y semántica interna, este algoritmo «comparte» las instancias entre tres clasificadores base. Cada uno de ellos puede clasificar cada instancia, incluso con distintas etiquetas y en la misma iteración que los otros clasificadores. Por lo tanto, para cada una de estas columnas y para cada instancia se tendrá ahora una lista.

Recopilando, las columnas para el «DataFrame» de Democratic Co-Learning son:

- Columnas con los nombres de los atributos de las instancias.
- Columna «iters»: Lista con la iteración en la que cada clasificador etiqueta la instancia. En el caso de un dato inicial, esta lista sol o tendrá una posición y contendrá 0 ([0]), para el resto siempre tendrá tres posiciones (por los tres clasificadores base). Para este último caso cada posición es independiente, es decir, si un clasificador base no ha etiquetado, contendrá -1 (**diferencia con los anteriores** [1]), pero para esa misma instancia otro clasificador base sí que puede haber etiquetado y contendrá dicha iteración (por ejemplo: [-1,4,-1])
- Columna «targets»: Exactamente igual a «iters» salvo que no indica la iteración, sino la etiqueta que asigna el clasificador base. De nuevo, si no la etiqueta, su posición contendrá -1.
- Columna «clfs»: Hasta ahora se ha hablado de posiciones en las dos columnas anteriores. Para saber a qué clasificador se refiere esta columna contiene otra lista con los tres nombres de los clasificadores.

[1]La razón de que se indique -1 en vez del número de iteraciones + 1, es porque toda esta estructura se genera antes del entrenamiento. En los algoritmos anteriores, si quedaba alguna sin etiquetar, se añadía al final y se sabía el número de iteraciones final.

Si es un dato inicial, contendrá [inicio], en otro caso contendrá algo como [CLF1(KNeighborsClassifier), CLF2(DecisionTreeClassifier), CLF3(GaussianNB)]

Ejemplo (entrenamiento finalizado en la iteración 9) [2]:

	petal.length	petal.width	sepal.length	sepal.width	iter	target	clf
0	5.6	2.5	3.9	1.1	[0]	[1]	[inicio]
1	6.1	2.8	4.0	1.3	[0]	[2]	[inicio]
78	6.7	3.1	5.6	2.4	[1, 4, -1]	[2, 2, -1]	[CLF1, CLF2, CLF3]
79	6.8	3.2	5.7	2.5	[-1, 2, -1]	[-1, 1, -1]	[CLF1, CLF2, CLF3]
142	6.8	2.8	4.8	1.4	[-1, -1, -1]	[-1, -1, -1]	[CLF1, CLF2, CLF3]

Tabla C.3: Ejemplo de DataFrame de Democratic Co-Learning

Tri-Training Tomando la idea de Democratic Co-Learning, en este algoritmo también se tienen tres clasificadores que pueden clasificar individualmente cada instancia. Sin embargo, cada iteración, el conjunto de datos **nuevos** etiquetados se vacía. Es decir, en una iteración se acaban etiquetando algunos, pero al principio de la siguiente se vacía ese conjunto y se vuelven a etiquetar de nuevo.

Entonces, el mecanismo de las listas de Democratic Co-Learning no funcionaría. Se debe añadir un nivel más de registro. Simplemente con añadir una lista en cada posición de las listas de «iters» y «targets» ya se puede registrar todos los momentos en los que una instancia se etiqueta (de nuevo, cada instancia podría ser etiquetada dos o más veces por un mismo clasificador base, al contrario del anterior).

Recopilando, las columnas para el «DataFrame» de Democratic Co-Learning son:

- Columnas con los nombres de los atributos de las instancias.
- Columna «iters»: Lista de listas con la iteración en la que cada clasificador etiqueta la instancia. En el caso de un dato inicial, esta lista sol o tendrá una posición y contendrá 0 ([0]), para el resto siempre tendrá una lista en las tres posiciones (por los tres clasificadores base). Para esto último caso las posiciones siguen siendo independientes solo que ahora si no se etiqueta la lista estará vacía (por ejemplo: [[2],

[2]Se han acortado los nombres de los clasificadores [CLF1, CLF2, CLF3] debería ser [CLF1(KNeighborsClassifier), CLF2(DecisionTreeClassifier), CLF3(GaussianNB)]

[], []], los dos últimos clasificadores no etiquetaron esa instancia en ningún momento). Para clarificar lo comentado anteriormente, los clasificadores podría etiquetar una misma instancia dos o más veces (por ejemplo: [[2], [2], [1, 2]], el último clasificador etiquetó la instancia en dos ocasiones).

- Columna «targets»: Exactamente igual a «iters» salvo que no indica la iteración, sino la etiqueta que asigna el clasificador base. Y de nuevo, si un clasificador no etiqueta en ningún momento esa instancia su lista interna estará vacía (por ejemplo: [[2], [], [2]]).

- Columna «clfs»: No se modifica respecto a Co-Training, contiene la lista de los nombres de los tres clasificadores.

Ejemplo (entrenamiento finalizado en la iteración 9)[3]:

	petal.length	petal.width	sepal.length	sepal.width	iter	target	clf
0	5.6	2.5	3.9	1.1	[0]	[1]	[inicio]
1	6.1	2.8	4.0	1.3	[0]	[2]	[inicio]
78	6.7	3.1	5.6	2.4	[[1], [1,2], [ ]]	[[2], [2, 2], [ ]]	[CLF1, CLF2, CLF3]
79	6.8	3.2	5.7	2.5	[[ ], [2], [ ]]	[[ ], [1], [ ]]	[CLF1, CLF2, CLF3]
142	6.8	2.8	4.8	1.4	[[ ], [ ], [ ]]	[[ ], [ ], [ ]]	[CLF1, CLF2, CLF3]

Tabla C.4: Ejemplo de DataFrame de Tri-Training

Interpretación sencilla de estos formatos Cuando se realiza su visualización, el primer paso es extraer los datos. Lo que se hace es crear un punto en el gráfico por cada instancia, con la particularidad de que cuando se entra en los algoritmos Democratic Co-Learning o Tri-Training, se añade más de un punto por cada instancia, representando la individualidad de cada clasificador base (cada uno de ellos puede haber clasificado esa instancia por separado e incluso más de una vez).

Cuando se genera el gráfico con sus puntos, cada punto lleva guardada la información de las iteraciones, etiquetas y clasificadores. Así se controla cuando ocultar/colorear/mostrar un punto. Por ejemplo, cuando se navega a la siguiente iteración se filtran todos los puntos que tenga esa iteración en la columna «iter» (o «iters»). Obviamente, cada algoritmo tiene sus particularidades, pero esta es la idea general.

[3]Se han acortado los nombres de los clasificadores [CLF1, CLF2, CLF3] debería ser [CLF1(KNeighborsClassifier), CLF2(DecisionTreeClassifier), CLF3(GaussianNB)]

Estadísticas generales

Las estadísticas generales son comunes a todos los algoritmos, no hay ninguna modificación entre ellos.

Se tiene un «DataFrame» con tantas filas como iteraciones se han ejecutado. En cuanto a las columnas, simplemente son los nombres de las estadísticas que se desean mostrar. Los nombres son los que se mostrarán en la Web.

Ejemplo:

	Accuracy	Precision	Error	F1_score	Recall
0	0.833333	0.888889	0.166667	0.822222	0.833333
1	1.000000	1.000000	0.000000	1.000000	1.000000
2	1.000000	1.000000	0.000000	1.000000	1.000000
3	1.000000	1.000000	0.000000	1.000000	1.000000
4	1.000000	1.000000	0.000000	1.000000	1.000000

Estadísticas específicas

Este tipo de estadísticas se refiere a la necesidad de mostrar también las estadísticas particulares de los clasificadores base. Esto aplica para Democratic Co-Learning y Tri-Training.

El núcleo de esta estructura de datos siguen siendo los «DataFrames», sin embargo, se han envuelto en un diccionario. El diccionario tiene como claves, cada uno de los nombres de los clasificadores base de la ejecución (por ejemplo: CLF3(GaussianNB)). Los valores serán esos «DataFrames» que son exactamente iguales que para las estadísticas generales. Guardarán por cada iteración (fila), las estadísticas (columnas) para el clasificador base concreto.

C.3. Diseño procedimental

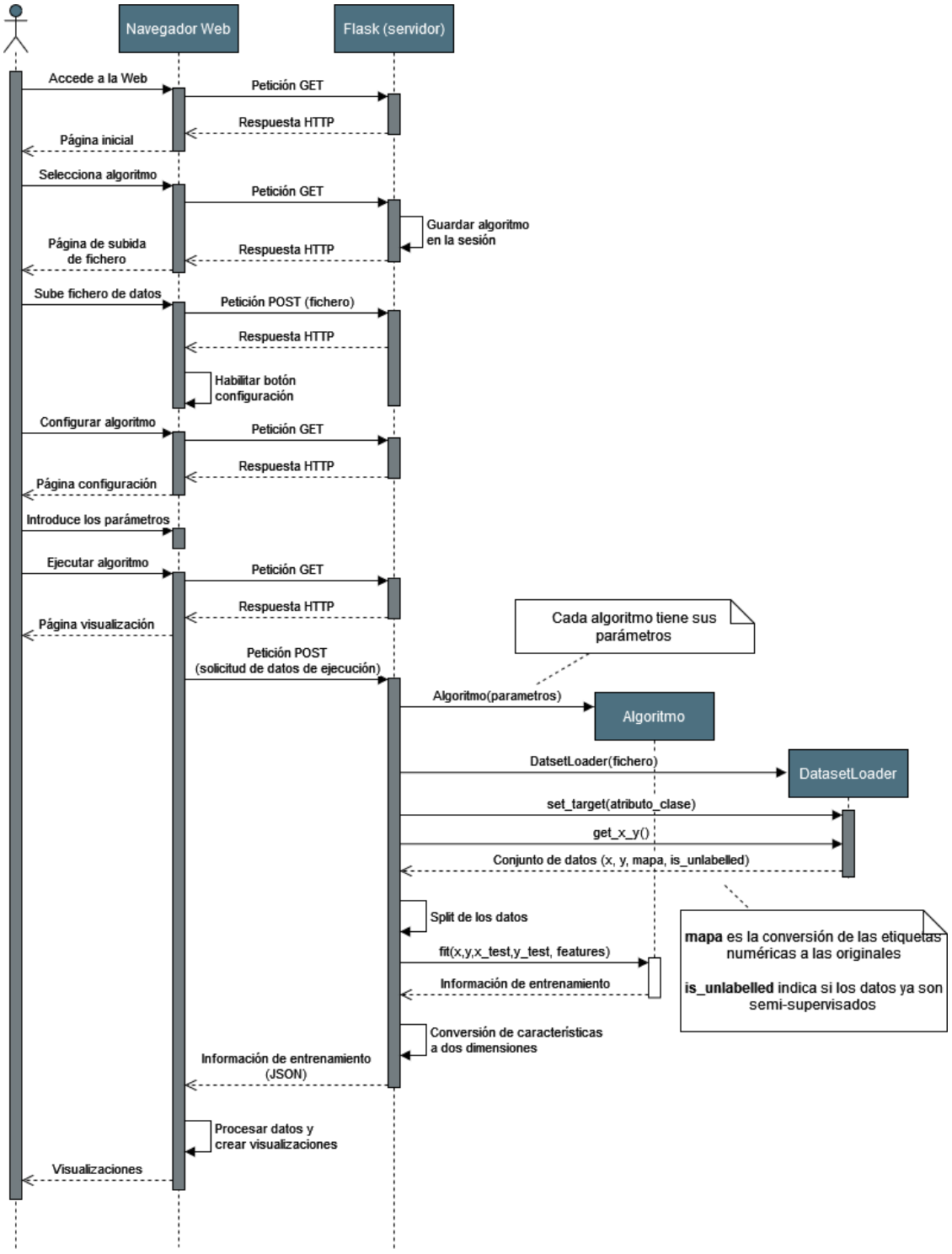

Figura C.1: Diagrama de secuencia de la interacción Web

C.4. Diseño arquitectónico

C.5. Diseño Web

Primer mockup o maqueta

Se presenta el primer Mockup o maqueta que se comentó de la página Web. Todas las páginas tendrán una base común en la que aparecerá información general como la Universidad de Burgos (barra superior).

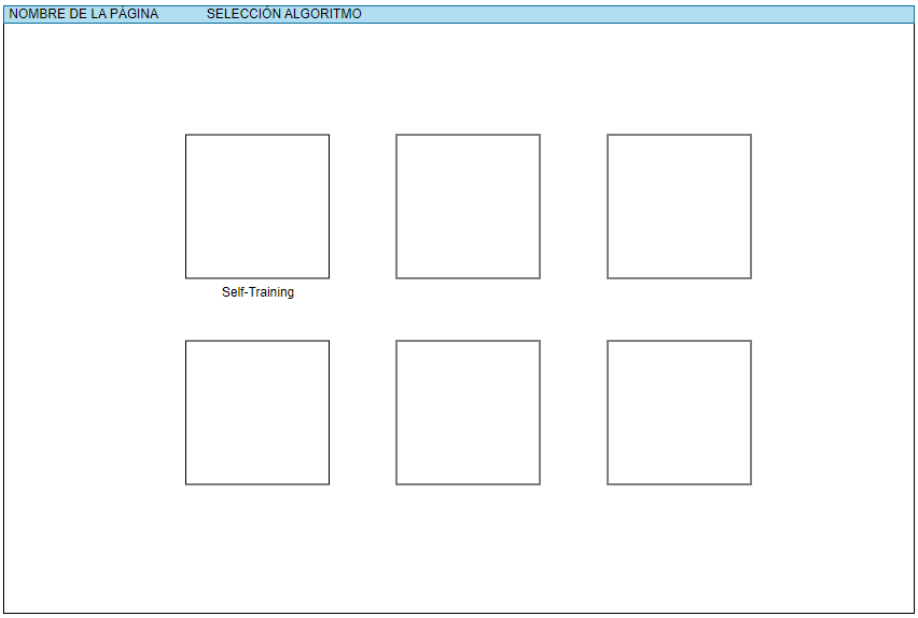

Figura C.2: Página inicial de la Web.

En esta página inicial el usuario podrá seleccionar el algoritmo que desea visualizar. En los cuadrados existirá un logo o imagen representativa del algoritmo junto con su nombre.

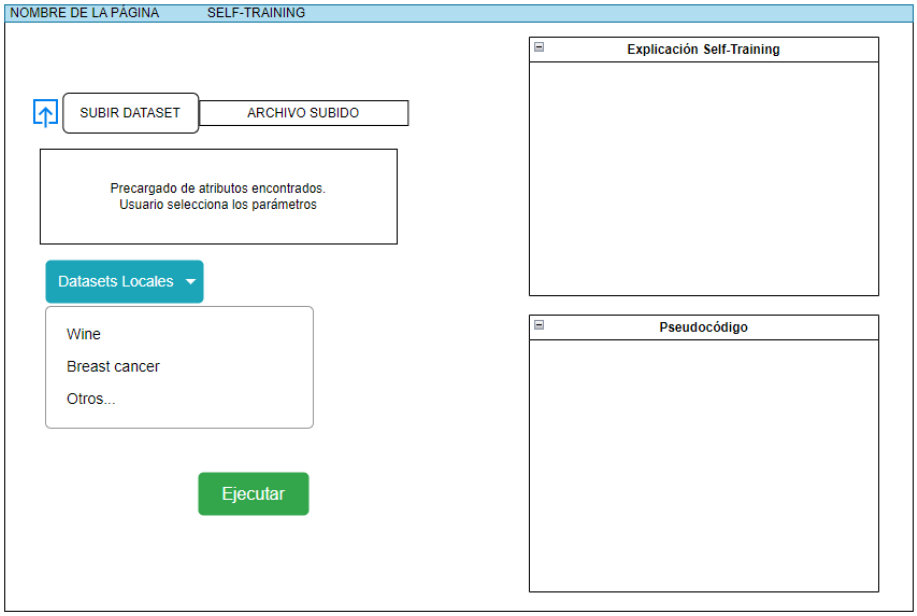

Figura C.3: Página de configuración del algoritmo.

En esta ventana el usuario podrá subir el conjunto de datos que desee o incluso seleccionar alguno de los almacenados localmente. Además, como los algoritmos tienen parámetros personalizables también habrá elemento para configurarlos.

Antes de iniciar, se muestra una explicación del algoritmo general y su pseudocódigo.

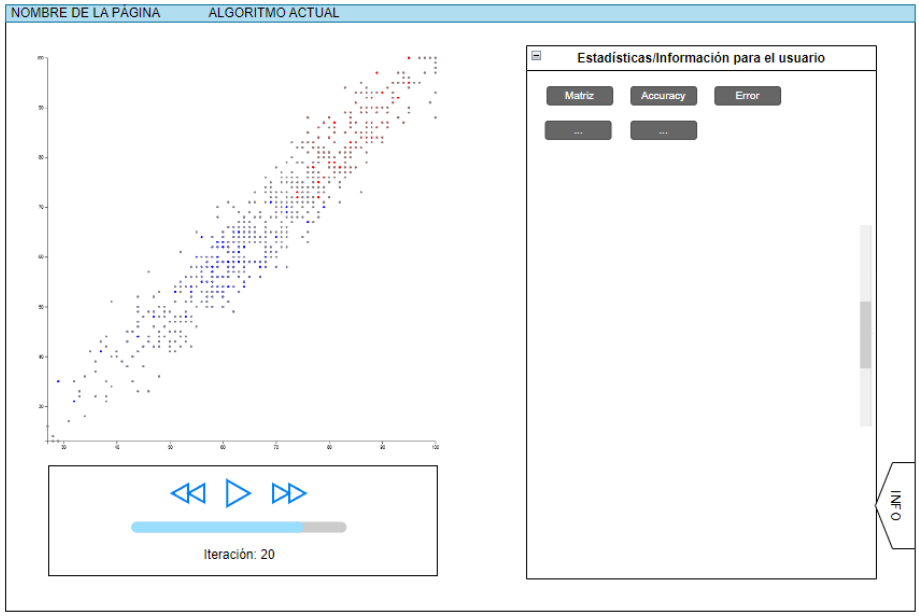

Figura C.4: Página de ejecución del algoritmo.

Mostrará la evolución del entrenamiento de los algoritmos con una vista principal (izquierda) de la clasificación y un compendio de métricas como la precisión o el error en su caso (derecha). Esto último principalmente planteado para ocultar/ver lo que el usuario desee en cada momento.

Diseño final

Apéndice D

Documentación técnica de programación

D.1. Introducción

En esta sección se presenta toda la documentación técnica del desarrollo del proyecto. Trata de ser una guía para entender cómo se ha hecho el proyecto comenzando por los directorios y su contenido, con un manual introductorio para un programador iniciado en el proyecto, explicación y ejemplificación de la instalación del mismo y las pruebas que se han realizado para validarlo.

Repositorio Github

https://github.com/dma1004/TFG-SemiSupervisado

D.2. Estructura de directorios

Estos son los directorios en los que se organiza el proyecto:

```
/
├── algoritmos: algoritmos Semi-Supervisados
│               implementados.
│
├── test: directorios y ficheros mediante los
│         que se valida el desarrollo software
│         correcto del proyecto.
│
│   ├── check_implementations: pruebas para la validación
│   │                          de los algoritmos.
│   │
│   │   └── results: ficheros CSV con resultados de
│   │                validación cruzada.
│   │
│   ├── check_utils: pruebas para la validación de las
│   │                utilidades.
│   │
│   │   └── test_files: ficheros de prueba (ARFF y CSV) para
│   │                   las pruebas.
│   │
│   ├── profiling: pruebas para medir el tiempo de
│   │              ejecución.
│   │
│   │   └── profile_results: resultados de los procesos de
│   │                        «profiling».
│   │
├── utilidades: utilidades (programas) que realizan
│               ciertos pasos de la aplicación y
│               de los algoritmos para centralizar
│               estos procedimientos (comunes).
│
├── docs: documentación teórica y técnica del
│         proyecto (hecha en LaTeX).
│
│   ├── img: imágenes utilizadas para la
│   │        generación de la documentación.
│   │
│   ├── tex: archivos de texto plano con código
│   │        LaTeX.
│
└── web: estructura o código de la aplicación
         Web.
```

- **web**: estructura o código de la aplicación Web.
  - **app**: contiene la definición de las rutas, modelos de la base de datos y la Application Factory que instancia la aplicación.
    - **datasets**: contiene (durante el funcionamiento de la aplicación) todos los conjuntos de datos que los usuarios introducen.
      - **seleccionar**: conjuntos de datos para seleccionar de prueba, principalmente durante el desarrollo de la aplicación. También almacena el fichero de prueba que los usuarios pueden descargar.
    - **static**: ficheros estáticos que utiliza la aplicación Web: CSS, Javascript, JSON o imágenes.
      - **css**: ficheros CSS.
      - **js**: ficheros JavaScript.
      - **json**: ficheros JSON.
      - **pseudocodigos**: imágenes de los pseudocódigos.
        - **en**: imágenes de los pseudocódigos en inglés.
        - **es**: imágenes de los pseudocódigos en español.
    - **templates**: plantillas HTML (Jinja2) que renderiza la aplicación Web (Flask).
    - **translations**: traducciones de los textos de la aplicación (por idiomas).
  - **instance**: contiene la base de datos de la aplicación (SQLite).

D.3. Manual del programador

El objetivo de este manual es dar al lector/desarrollador que comience a trabajar con el proyecto, el conocimiento necesario para continuarlo. Se ha de tener en cuenta que lo descrito a continuación es lo que se ha utilizado para el entorno desarrollo inicial y será explicado para este.

En primer lugar se listan las herramientas consideradas para el desarrollo:

- Python 3.10: Todo el proyecto, desde su inicio, ha sido desarrollado en la versión 3.10.
- Git: Necesario para continuar con el control de versiones del proyecto.
- Pycharm: Editor de código utilizado, podría utilizarse otro si así se considerase.

Python puede descargarse desde su página principal[1]. En las herramientas no se ha mencionado «pip» (el administrador de paquetes) pues desde la versión 3.4 de Python este está instalado con él, sin embargo, sería conveniente asegurarse de ello comprobado la versión pues en el futuro será necesario.

Comprobar pip

```
pip --version
python -m ensurepip --upgrade
```

Y de forma general, comprobar que los binarios pueden ser utilizados por lo menos en el entorno del programador (en su usuario o equipo completo).

En el caso de Git, desde Linux simplemente se puede realizar con el gestor de paquetes:

Instalar Git en Linux

```
sudo apt install git-all
```

Si el entorno es Windows, existe un instalador directo que puede descargarse desde la página de Git SCM (Source code management)[2].

[1] Descargas de Python: https://www.python.org/downloads/
[2] Git para Windows: https://git-scm.com/download/win

Pycharm se puede descargar tanto para Windows como Linux en la página oficial de JetBrains[3].

Comprensión de la estructura

Se recomienda leer la sección anterior donde se pueden consultar todos los directorios del proyecto con una breve descripción. La preparación del entorno virtual con el que trabajar se explicará en la próxima sección.

El proyecto se desarrolla en dos ramas comunicadas (de forma unidireccional): los algoritmos implementados y la aplicación Web. La aplicación es la que utiliza los algoritmos para obtener la información presentada en la Web.

Algoritmos semi-supervisados (contenidos en el directorio algoritmos): Los algoritmos desarrollados y nuevos han de situarse en este directorio como raíz. La idea es que cada fichero «.py», homónimo al algoritmo, contenga la definición de un objeto que encapsule el desarrollo del mismo, para que no haya confusión.

Estructura de los objetos (mínima):

Constructor: Donde se configuran los parámetros que necesita el algoritmo. Es recomendable realizar una validación de los mismos por si fueran utilizados de manera individual.

Método de entrenamiento (Fit): Dado que estos algoritmos están pensados no solo para entrenar, sino para almacenar el proceso de entrenamiento y estadísticas, siempre ha de recibir el conjunto de entrenamiento (x, y), el conjunto de test (x_test, y_test) y el nombre de las características de cada instancia.

En principio, el método de desarrollo seguido es el de primero implementar el algoritmo para después añadir, con la librería Pandas, un registro completo de los momentos de etiquetado y de las estadísticas de cada iteración.

Este método deberá retornar el registro de etiquetado, el estadístico y el número de iteraciones realizadas.

Cabecera fit

```
def fit(x, y, x_test, y_test, features)
```

[3]Pycharm: https://www.jetbrains.com/pycharm/download/

Es importante tener en cuenta que esta estructura puede variar en la medida de cómo sea el algoritmo. Por ejemplo, en el caso de Democratic Co-Learning se hacía imprescindible añadir estadísticas específicas para cada clasificador que encapsula y por tanto, retornaba más elementos.

Método de predicción: En el caso de algoritmos que trabajan con un único clasificador podría ser opcional, pero en el caso de varios, es necesario considerar cómo se predicen las etiquetas en combinación.

Cabecera predict

```
def predict(self, instances)
```

Métodos estadísticos: Dependiendo de lo que se desee mostrar en la aplicación, se incluirán ciertas estadísticas.

La convención utilizada hasta ahora es crear un método que comience por «get_» seguido del nombre de la estadística, por ejemplo `get_accuracy_score`.

Cabecera ejemplo estadística

```
def get_accuracy_score(self, x_test, y_test):
```

Utilidades En el directorio de las utilidades se han de alojar aquellos métodos que se reutilizan en el proyecto (y que intervenga algún paso del algoritmo, por ejemplo, la carga de datos).

Tests El desarrollo del software debe ser validado para asegurar su correcto funcionamiento ante las distintas casuísticas. Cuando se desarrolla código en esta sección, sus casos de prueba codificados deben incluirse en el directorio correspondiente. Se utiliza «pytest» como *framework* de pruebas.

Aplicación Web (contenida en el directorio web):

La aplicación está desarrollada con el «micro-framework» Flask, que permite la creación de aplicaciones Web en Python. A lo largo del desarrollo la estructura de la aplicación ha variado bastante. Finalmente, se ha modularizado completamente con el uso de `Blueprints` y una `Application Factory` que se encarga de instancia la aplicación, base de datos y pone en funcionamiento las distintas rutas accesibles.

run.py: Centraliza la ejecución de la aplicación (mediante la `Application Factory`)

instance: Donde se almacena la instancia de la base de datos.

app: Este directorio contiene toda la definición de la aplicación, el resto de los apartados comentados siguientes se encuentran dentro de este.

El fichero `__init__.py` contiene la creación de la aplicación con un método `create_app` (`Application Factory`). Aquí se específica toda la configuración, los elementos comunes (como los filtros de Jinja), se instancia la base de datos y se registran las rutas (organizadas con `Blueprints` como paquetes o extensiones de la aplicación básica).

Blueprints: Las rutas que tiene la aplicación están organizadas en función de su categoría mediante los Blueprints. Y es en los ficheros que terminan por «_routes.py» donde se definen. Si se añade una categoría nueva se debe crear un Blueprint la identifique dentro de su fichero y después debe ser registrado en la aplicación (en `__init__.py`).

Crear Blueprint

```
# El nombre es a elección propia
nuevo_bp = Blueprint('nuevo_bp', __name__)
```

Registrar Blueprint

```
# Seleccionando el prefijo deseado
app.register_blueprint(nuevo_bp, url_prefix='/')
```

La visualización de un algoritmo se centra en dos pasos: la configuración del mismo (en las rutas `/configuracion/<algoritmo>`) donde se debe renderizar una página con el formulario de configuración (gestionado por `configuration_routes.py`), y la visualización (`/visualizacion/<algoritmo>`) donde se renderiza la página donde se encuentran todos los gráficos de los algoritmos (gestionado por `visualization_routes.py`). Además, existe un paso intermedio en el que se obtienen los datos de la ejecución de los algoritmos, este paso no es una ruta que pueda visualizarse (gestionado por `data_routes.py`), es un método auxiliar que accede el propio usuario (su navegador) para obtener la información. Por lo general, la convención utilizada es que todos los métodos y rutas lleven el nombre del algoritmo.

Plantillas (templates): Flask utiliza el motor de plantillas Jinja2, que añaden instrucciones en HTML. Obviando que cada algoritmo tiene sus particularidades, están organizadas mediante la extensión de plantillas. De forma general, añadir un algoritmo podría solo intervenir la creación de una

plantilla de configuración y otra de visualización. En ambos casos se tiene una plantilla base de la que se debe extender.

Ficheros estáticos (static): El contenido dinámico debe ser creado mediante Javascript, de forma «vanilla» (sin utilizar frameworks). En menor medida, los estilos deben ser retocados en estos ficheros aunque en principio la aplicación está estilada con Bootstrap 5.

En estos ficheros estáticos se tienen unas imágenes con los pseudocódigos de los algoritmos. Estas imágenes deben verse tanto en la configuración como la visualización.

Otra parte muy importante es si se quieren añadir nuevos clasificadores base con sus parámetros, estos están almacenados en el fichero JSON «parametros.json». Hasta el momento del desarrollo solo existen dos tipos de entradas, los selectores y los numéricos.

Estructura para añadir clasificadores y sus parámetros

```
{
    "ClasificadorBase": {
        "parametro_numerico": {
        "label": "Nombre a mostrar",
        "type": "number",
        "step": 1,
        "min": 1,
        "max": "Infinity",
        "default": 5
        },
        "parametro_selector": {
        "label": "Nombre a mostrar",
        "type": "select",
        "options": ["lista", "de", "elementos"],
        "default": "elementos"
        }
    }
}
```

Con «step» se debe tener en cuenta la posibilidad de permitir números enteros (al introducir 1) o números decimales (al incluir un flotante, por ejemplo, 0.01)

Conjunto de datos (datasets): Los conjuntos de datos de los usuarios (vinculados a sus sesiones) se almacenan localmente en la aplicación con el «Timestamp» concatenado el nombre del fichero introducido.

Internacionalización: La aplicación está pensada para ser internacionalizada en cualquier idioma, aunque solo se ha incluido inglés y español. Una de las herramientas utilizadas es Babel, para incluir nuevo texto en la aplicación se debe utilizar la función `gettext()` (tanto en Jinja2 como Python si fuera necesario). El proceso de compilación está explicado en la siguiente sección.

D.4. Compilación, instalación y ejecución del proyecto

Para poner en funcionamiento la aplicación primero se debe preparar su entorno de ejecución. Se recuerda la necesidad de tener instalado Python (con el administrador de paquetes «pip»).

Código fuente de la aplicación Para obtener el código fuente de la aplicación, este debe ser descargado del repositorio Github utilizado[4].

Tanto en Windows como en Linux esto se puede realizar descargando el fichero comprimido de todo el repositorio desde el navegador.

Pero si se quiere realizar mediante consola de comandos, y suponiendo que se ha instalado Git:

Clonación de repositorio desde consola

```
$ git clone https://github.com/dma1004/TFG-SemiSupervisado.git
```

La localización del proyecto queda a discreción del programador.

Entorno virtual Python El entorno virtual no es estrictamente necesario aunque es **altamente** recomendable, pues en los próximos pasos se instalarán múltiples librerías que sin entorno virtual quedarían instaladas globalmente.

El proceso descrito a continuación puede ser sustituido en caso de que solo se esté trabajando con Pycharm. Este editor permite crear entornos virtuales e incluso realizarlo automáticamente al detectar los distintos requisitos del proyecto. Pero de forma general, suponiendo que se desea preparar un entorno

[4]Repositorio: https://github.com/dma1004/TFG-SemiSupervisado

de «pruebas» o «producción» y que la aplicación esté en funcionamiento, se realizan los siguientes pasos:

Creación del entorno virtual (dentro de la carpeta deseado)

```
$ python -m venv ./venv
```

Activación del entorno virtual

```
$ venv\activate.bat //Windows
```

```
$ source ruta/al/entorno/virtual/bin/activate //Linux
```

Para desactivar el entorno (una vez en él)

```
$ deactivate
```

Instalación de paquetes En este paso se van a instalar todas las librerías necesarias, el proyecto trae un fichero «requirements.txt» en el que vienen especificados estos paquetes y sus versiones. Además, los algoritmos implementados también están configurados como un paquete que puede ser instalado.

Instalar librerías externas

```
$ python -m pip install -r requieremens.txt
```

Instalar paquete de algoritmos

```
$ python -m pip install .
```

A partir de aquí ya se tiene configurado todo el entorno y los requisitos necesarios para poder ejecutar la aplicación.

Ejecución Referida a la ejecución de la aplicación Web (Flask), se debe estar situado en el directorio **/web** del proyecto.

Ejecución

```
$ set FLASK_APP=app.py //Windows
$ flask run
```

```
$ flask run //Linux
```

Existe una última cuestión que no es estrictamente necesaria para funcionar, pero que añade la internacionalización a la aplicación automáticamente.

Internacionalización En el caso de que se haya incluido más texto traducido, este debe ser compilado con Babel para que la aplicación pueda detectarlo.

Proceso de internacionalización (desde /web)

```
Extraer los texto encapsulados por gettext
$ pybabel extract -F babel.cfg -o messages.pot .

Actualizar los textos extraidos en el fichero de compilación
$ pybabel update -i messages.pot -d translations

Compilación de las traducciones
$ pybabel compile -d translations
```

A partir de aquí la propia aplicación utilizará las traducciones dependiendo del usuario que acceda.

D.5. Pruebas del sistema

Apéndice E

Documentación de usuario

E.1. Introducción

Es esta sección se presenta los requisitos que el usuario debe satisfacer para utilizar la aplicación junto con la Instalación y manual de la misma.

E.2. Requisitos de usuarios

E.3. Instalación

E.4. Manual del usuario

Bibliografía