



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 — 24 de marzo
de 2023

Tutor: Álgvar Arnaiz González
Cotutor: César Ignacio García Osorio

Índice general

Índice general	i
Índice de figuras	iii
Índice de tablas	iv
Apéndice A Plan de Proyecto Software	1
A.1. Introducción	1
A.2. Planificación temporal	1
A.3. Estudio de viabilidad	11
Apéndice B Especificación de Requisitos	13
B.1. Introducción	13
B.2. Objetivos generales	13
B.3. Catálogo de requisitos	13
B.4. Especificación de requisitos	13
Apéndice C Especificación de diseño	15
C.1. Introducción	15
C.2. Diseño de datos	15
C.3. Diseño procedimental	15
C.4. Diseño arquitectónico	15
C.5. Diseño de la Web	15
Apéndice D Documentación técnica de programación	19
D.1. Introducción	19
D.2. Estructura de directorios	19

D.3. Manual del programador	21
D.4. Compilación, instalación y ejecución del proyecto	26
D.5. Pruebas del sistema	29
Apéndice E Documentación de usuario	31
E.1. Introducción	31
E.2. Requisitos de usuarios	31
E.3. Instalación	31
E.4. Manual del usuario	31
Bibliografía	33

Índice de figuras

A.1. Burndown chart del sprint 8.	9
A.2. Burndown chart del sprint 9.	11
C.1. Página inicial de la Web.	16
C.2. Página de configuración del algoritmo.	17
C.3. Página de ejecución del algoritmo.	18

Índice de tablas

B.1. CU-1 Nombre del caso de uso.	14
---	----

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 progress»), 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 Álar 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 Álar 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](#).

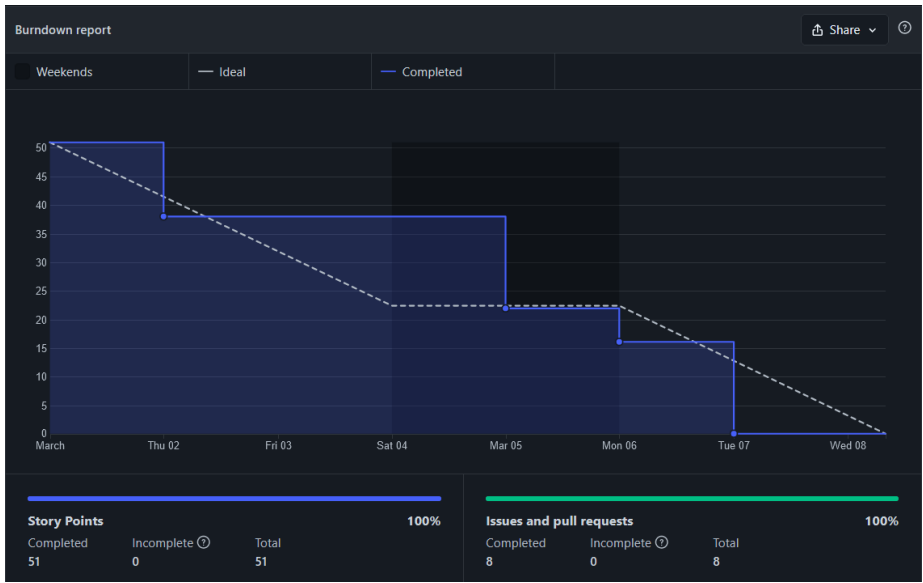


Figura A.2: Burndown chart del sprint 9.

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	<ol style="list-style-type: none"> 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

C.2. Diseño de datos

C.3. Diseño procedimental

C.4. Diseño arquitectónico

C.5. Diseño de la Web

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).

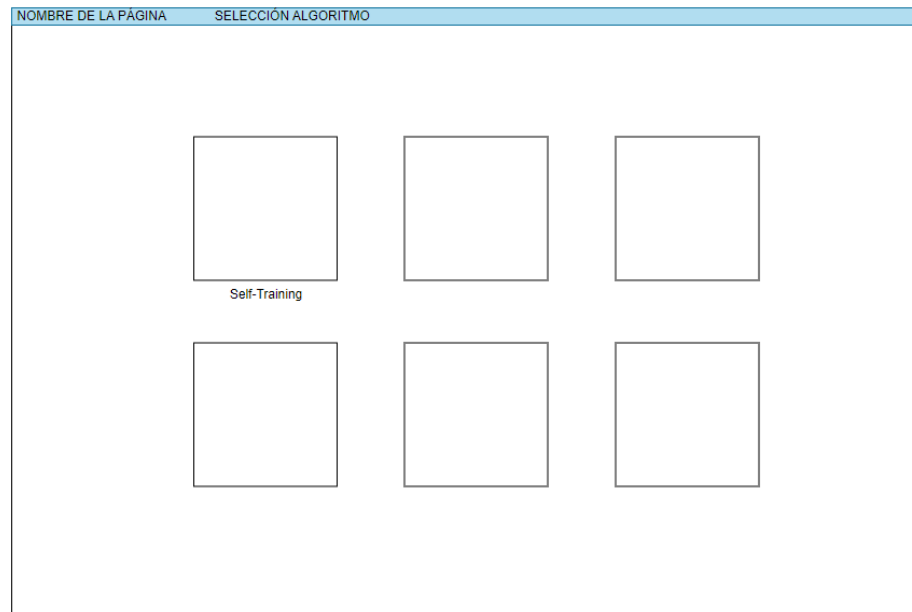


Figura C.1: Página inicial de la Web.

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.

NOMBRE DE LA PÁGINA SELF-TRAINING

SUBIR DATASET ARCHIVO SUBIDO

Precargado de atributos encontrados.
Usuario selecciona los parámetros

Datasets Locales ▾

Wine
Breast cancer
Otros...

Ejecutar

Explicación Self-Training

Pseudocódigo

Figura C.2: 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.

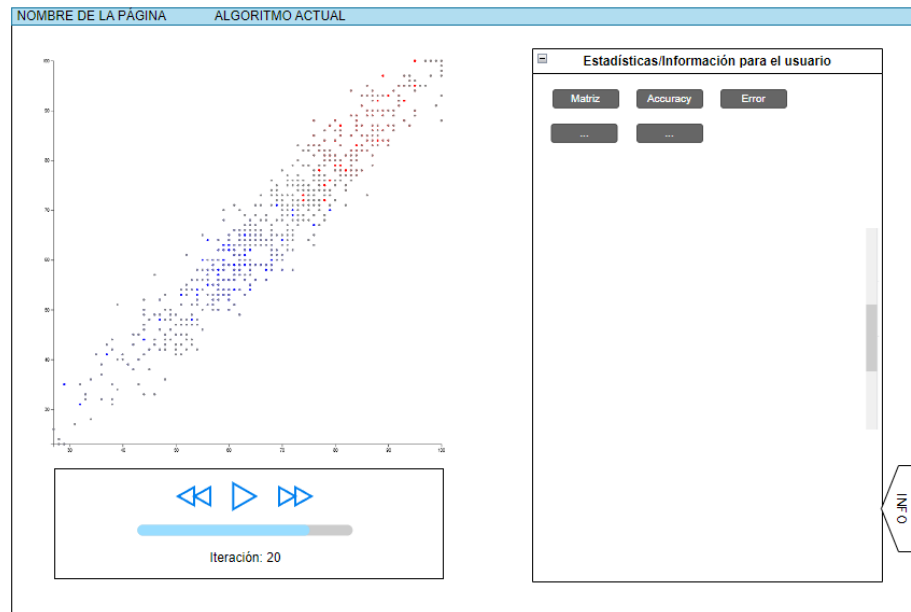


Figura C.3: Página de ejecución del algoritmo.

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.

Apéndice D

Documentación técnica de programación

D.1. Introducción

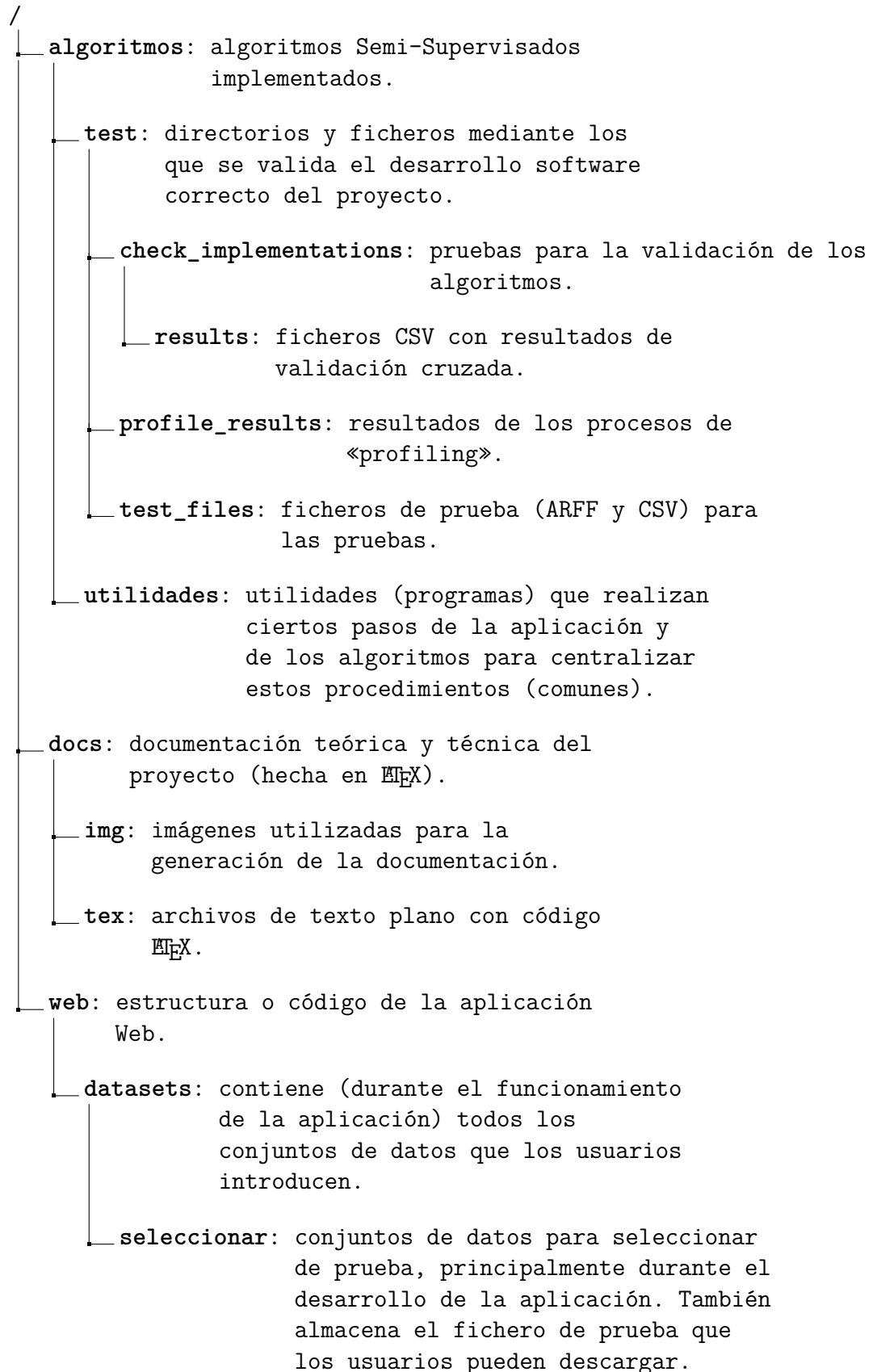
En esta sección se presenta toda la documentación técnica del desarrollo del proyecto. Trata de ser una guía para entender cómo se ha hecho el proyecto comenzando por los directorios y su contenido, con un manual introductorio para un programador iniciado en el proyecto, explicación y ejemplificación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



web: estructura o código de la aplicación Web.

- **static:** ficheros estáticos que utiliza la aplicación Web: CSS, Javascript o JSON.
- **templates:** plantillas HTML (Jinja2) que renderiza la aplicación Web (Flask).
- **translations:** traducciones de los textos de la aplicación (por idiomas).

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 principal¹. 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
```

¹Descargas de Python: <https://www.python.org/downloads/>

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.

Pycharm se puede descargar tanto para Windows como Linux en la página oficial de JetBrains³.

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,

²Git para Windows: <https://git-scm.com/download/win>

³Pycharm: <https://www.jetbrains.com/pycharm/download/>

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)
```

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 antes 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. Toda la lógica de la aplicación (exceptuando el contenido dinámico) está codificado en `app.py`.

app: En este fichero se definen todas las posibles rutas a las que los usuarios pueden acceder. Si se desea añadir alguna nueva, las anotaciones de Flask permiten especificarlas.

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, y la visualización (`/visualizacion/<algoritmo>`) donde se renderiza la página donde se encuentran todos los gráficos de los algoritmos. Además, existe un paso intermedio en el que se obtienen los datos de la ejecución de los algoritmos, este paso no es una ruta accesible, 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: 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: 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.

Otra parte muy importante es la adición de clasificadores base que están almacenados en el fichero JSON «parametros». Hasta el momento del desarrollo solo existen dos tipos de entradas, los selectores y los numéricos (para los clasificadores).

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: 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()` (en Jinja2). 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 utilizado⁴.

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 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
```

⁴Repositorio: <https://github.com/dma1004/TFG-SemiSupervisado>

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 requirements.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 Windows

```
$ 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
- E.2. Requisitos de usuarios
- E.3. Instalación
- E.4. Manual del usuario

Bibliografía
