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

Presentado por David Martínez Acha
en Universidad de Burgos
el 2 de junio de 2023

Tutores: Dr. Álvar Arnaiz González
Dr. César Ignacio García Osorio

UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

El Dr. Álvar Arnaiz González, junto a el Dr. César Ignacio García Osorio, profesores del departamento de Ingeniería Informática, área de Lenguajes y Sistemas informáticos.

Exponen:

Que el alumno D. David Martínez Acha, con DNI 71310644H, ha realizado el Trabajo final de Grado en Ingeniería Informática titulado Herramienta docente para la visualización en Web de algoritmos de aprendizaje Semi-Supervisado.

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección del que suscribe, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, 2 de junio de 2023

Vº. Bº. del Tutor:

Dr. Álvar Arnaiz González

Vº. Bº. del co-tutor:

Dr. César Ignacio García Osorio

Resumen

En este primer apartado se hace una **breve** presentación del tema que se aborda en el proyecto.

Descriptores

aprendizaje automático, aprendizaje semi-supervisado, visualización de algoritmos, web

Abstract

A **brief** presentation of the topic addressed in the project.

Keywords

machine learning, semi-supervised learning, algorithm visualization, web

Índice general

Índice de figuras

Índice de tablas

Introducción

El ámbito del aprendizaje automático «machine learning» es un campo muy interesante y que cada vez recibe más atención. La realidad es que la mayor parte del conocimiento está muy centrado en dos tipos de aprendizaje automático: el supervisado y el no supervisado. En cuanto se indaga un poco en «machine learning» aparecen estos dos conceptos. Pero del que no se oye tanto, y puede ser muy beneficioso, es el aprendizaje semi-supervisado.

El aprendizaje supervisado, en pocas palabras, permite aprovechar situaciones en las que se sabe qué representa un dato (por ejemplo, dado una animal, se sabe si el animal es un perro o un pato), el no supervisado no tiene esta «suerte», se utiliza en casos en los que no se tiene ese conocimiento, sino que es él mismo el que intenta extraer las representaciones (por ejemplo, para un conjunto de animales, podría distinguir entre los que tiene pico y alas y los que tienen cuatro patas sin necesidad de saber qué animal concreto es). En la realidad (obviando los ejemplos tan sencillos comentados), el «etiquetado» de los datos suele ser muy costoso y se tienen muchos más datos que no se sabe qué representan, el aprendizaje semi-supervisado es el ideal en estos casos dado que permite inferir conocimiento para estos últimos y determinar a qué corresponden.

Centrando más el objetivo final de este trabajo, no nos consta que existan aplicaciones que permitan compaginar la teoría de estos conceptos con visualizaciones interesantes que ayuden a comprender su funcionamiento y mucho menos, para los algoritmos semi-supervisados que incluso en muchos casos suelen obviarse. Vista la carencia en este ámbito, este trabajo pretende crear una aplicación amigable y atractiva que permita, mediante visualizaciones, facilitar la comprensión de cómo funcionan realmente los principales algoritmos semi-supervisados cuando se compagina con los conceptos teóricos.

Las herramientas de fácil acceso, como las páginas Web que no requieren de instalación por parte del usuario, van de la mano de la globalización de internet. Es por ello que esta herramienta, categorizada ya como *docente*, será accesible desde internet. La idea de esto es permitir a los usuarios la rapidez y facilidad de acceder simplemente a una «URL» sin necesidad de herramientas auxiliares.

Además, los datos de las visualizaciones serán proporcionados por una biblioteca propia donde se implementen estos algoritmos. Estarán adaptados para la obtención de la información del entrenamiento y estadísticas relevantes, pero en su caso también están pensados para que puedan ser utilizados de forma general independientemente de que no haya una visualización posterior a su utilización.

Objetivos del proyecto

Este apartado explica de forma precisa y concisa cuáles son los objetivos que se persiguen con la realización del proyecto.

Los **objetivos generales** del proyecto son los que siguen:

1. Implementación de una biblioteca con cuatro algoritmos de aprendizaje semi-supervisado más comunes: *Self-Training*, *Co-Training*, *Democratic Co-Learning* y *Tri-Training*.
2. Diseño y creación de una aplicación Web desde la accesibilidad y enfoque docente (ayudas contextuales, explicaciones, pseudocódigos, instrucciones...).
3. Integración de los cuatro algoritmos en la aplicación Web para su visualización.
4. Crear un sistema de usuarios que les permita controlar sus ficheros y ejecuciones.
5. Internacionalización de la página web de la aplicación.

Como **requisitos técnicos** y más particulares del proyecto se encuentran los siguientes:

1. Implementación de los algoritmos en Python 3.10.
2. Utilización del *framewrok* Flask para el desarrollo de la aplicación web.
3. Diseño de la web basado en Bootstrap 5.

4. Creación de las visualizaciones mediante JavaScript y la biblioteca `D3.js`.

5. Optimizar al máximo posible el procesamiento de datos y entrenamiento.

6. Internacionalización mediante Babel (Flask-Babel).

7. Creación y manejo de una base de datos para usuarios (y relativos) con independencia de la tecnología utilizada (SQLAlchemy).

8. Realización de pruebas sobre el software desarrollado (comparativa/validación con otras librerías y test unitarios mediante `pytest`).

9. Crear una documentación de usuario y programador precisa y completa.

Conceptos teóricos

En esta sección se presentarán los principales conceptos teóricos del proyecto. Estos incluyen el aprendizaje automático y más específicamente el aprendizaje semi-supervisado.

3.1. Aprendizaje automático

Según [18], el aprendizaje automático (*machine learning*) es una rama de la Inteligencia Artificial como una técnica de análisis de datos que enseña a las computadoras a aprender de la **experiencia** (es decir, lo que realizan los humanos). Para ello, el aprendizaje automático se nutre de gran cantidad de datos (o los suficientes para el problema concreto) que son procesados por ciertos algoritmos. Estos datos son ejemplos (también llamados instancias o prototipos), [28] mediante los cuales, los algoritmos son capaces de generalizar comportamientos que se encuentran ocultos.

La característica principal de estos algoritmos es que son capaces de mejorar su rendimiento de forma automática basándose en procesos de entrenamiento y también en las fases posteriores de explotación. Debido a sus propiedades, el aprendizaje automático se ha convertido en un campo de alta importancia, aplicándose a multitud de campos como medicina, automoción, visión artificial... Los tipos de aprendizaje automático se suelen clasificar en los siguientes: aprendizaje supervisado, aprendizaje no supervisado y aprendizaje por refuerzo. Sin embargo, aparece una nueva disciplina que se encuentra a caballo entre el supervisado y no supervisado (utiliza tanto datos etiquetados como no etiquetados para el entrenamiento) [30].

En la figura 3.1 se puede ver una clasificación de aprendizaje automático.

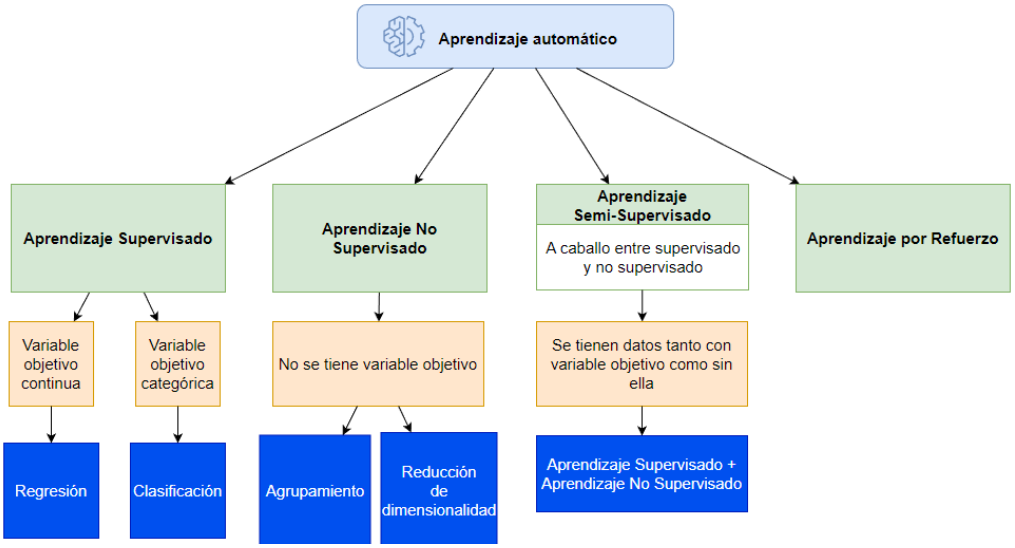

Figura 3.1: Clasificación de aprendizaje automático [27].

Aprendizaje supervisado

El aprendizaje supervisado es una de las aproximaciones del aprendizaje automático. Los algoritmos de aprendizaje supervisado son entrenados con datos que han sido etiquetados para una salida concreta [23]. Por ejemplo, dadas unas biopsias de pacientes, una posible etiqueta es si padecen de cáncer o no. Estos datos tienen una serie de características (e.g. en el caso de una biopsia se tendría la edad, tamaño tumoral, si ha tenido lugar mitosis o no...) y todas ellas pueden ser binarias, categóricas o continuas [11].

Es común que antes del entrenamiento, estos datos sean particionados en: conjunto de entrenamiento, conjunto de test y conjunto de validación. De forma resumida, el conjunto de entrenamiento serán los datos que utilice el propio algoritmo para aprender y generalizar los comportamientos ocultos de los mismos. El conjunto de validación se utilizará para tener un control de que el modelo está generalizando y no sobreajustando (memorizando los datos) y también para decidir cuando finalizar el entrenamiento. Por último, el conjunto de test sirve para estimar el rendimiento real que podrá tener el modelo en explotación [34]. En la figura 3.2 puede visualizarse el funcionamiento general.

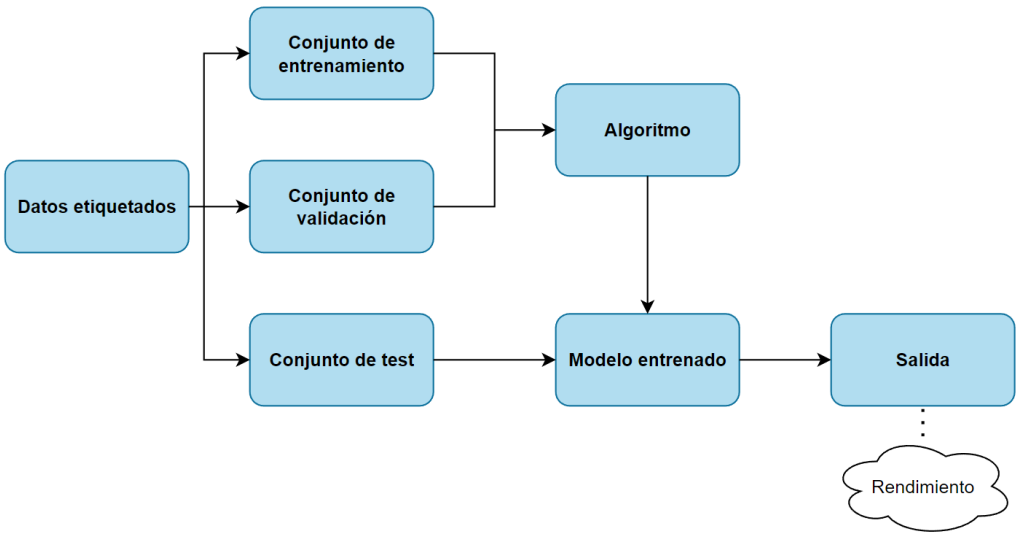

Figura 3.2: Funcionamiento general del aprendizaje supervisado [11].

El aprendizaje supervisado está altamente influenciado por esto. Por un lado, si el valor a predecir es uno entre un conjunto finito, el modelo será de **clasificación** y por otro, si el valor a predecir es un valor continuo, el modelo será de **regresión**.

- **Clasificación**: Los modelos de clasificación (generados a partir de algoritmos de aprendizaje), a veces denominados simplemente como clasificadores, tratan de predecir la clase de una nueva entrada a partir del entrenamiento previo realizado. Estas clases son discretas y en clasificación pueden referirse a clases (o etiquetas) binarias o clases múltiples.
- **Regresión**: En este caso, el modelo asigna un valor continuo a una entrada. Es decir, trata de encontrar una función continua basándose en las variables de entrada. Se denomina también ajuste de funciones.

Aprendizaje no supervisado

A diferencia del aprendizaje supervisado, en el no supervisado, los algoritmos de aprendizaje no se nutren de datos etiquetados. En otras palabras, los usuarios no «supervisan» el algoritmo [12]. Esto quiere decir que no aprenderán de etiquetas, sino de la propia estructura que se encuentre en los datos (patrones). Por ejemplo, dadas unas imágenes de animales, sin especificar cuál es cuál, el aprendizaje no supervisado identificará las similitudes entre imágenes y como resultado podría dar la separación de las especies (o separaciones entre colores, pelaje, raza...).

Como principales usos del aprendizaje no supervisado, suele aplicarse a:

1. **Agrupamiento (Clustering)**: Este tipo de algoritmo de aprendizaje no supervisado trata de dividir los datos en grupos. Para ello, estudia las similitudes entre ellos y también en las disimilitudes con otros. Estos algoritmos pueden tanto descubrir por ellos mismos los «clústeres» o grupos que se encuentran o indicarle cuántos debe identificar [12].

2. **Reducción de la dimensionalidad**: Para empezar, el término «dimensionalidad» hace referencia al número de variables de entrada que tienen los datos. En la realidad, los conjuntos de datos sobre los que se trabaja suelen tener una dimensionalidad grande. Según [19] la reducción de dimensionalidad se denomina como:

   > «Una forma de convertir conjuntos de datos de alta dimensionalidad en conjunto de datos de menor dimensionalidad, pero garantizando que proporciona información similar.»

   Es decir, simplificar el problema pero sin perder toda esa estructura interesante de los datos. Algunos ejemplos pueden ser:

   - Análisis de Componentes Principales (PCA).
   - Cuantificación vectorial.
   - Autoencoders.

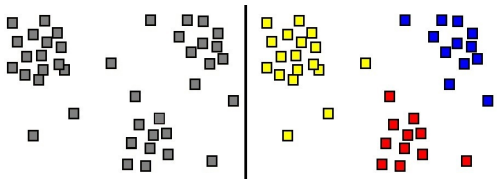

Figura 3.3: *Clusters*. Ejemplo de agrupamiento, a la izquierda los datos no etiquetados y a la derecha los datos coloreados según las clases identificadas por el algoritmo de clustering. By hellisp - Own work, Public Domain, https://commons.wikimedia.org/w/index.php?curid=36929773.

Aprendizaje semi-supervisado

Según [30], el aprendizaje semi-supervisado es la rama del aprendizaje automático referido al uso simultáneo de datos tanto etiquetados como no etiquetados para realizar tareas de aprendizaje. Se encuentra a caballo entre el aprendizaje supervisado y el no supervisado. Concretamente, los problemas donde más se aplica, y donde más investigación se realiza es en clasificación. Los métodos semi-supervisados resultan especialmente útiles cuando se tienen escasos datos etiquetados, que, aparte de ser una situación común en problemas reales, hacen que el proceso de etiquetado sea una labor compleja, que consume tiempo y es costosa.

Suposiciones

El objetivo de usar datos no etiquetados es construir un clasificador que sea mejor que el que se obtendría utilizando aprendizaje supervisado, donde solo se tienen datos etiquetados. Pero para que el aprendizaje semi-supervisado mejore a lo ya existente, tiene una serie de suposiciones que han de cumplirse.

En primera instancia se dice que la condición necesaria es que la distribución $p(x)$ del espacio de entrada contiene información sobre la distribución posterior $p(y|x)$ [30].

Pero la forma en el que interactúan los datos de una distribución y la posterior, no siempre es la misma:

Smoothness assumption

Esta suposición indica que si dos ejemplos (o instancias) de la entrada están cerca en ese espacio de entrada, entonces, probablemente, sus etiquetas sean las mismas.

Low-density assumption

Esta suposición indica que en clasificación, los límites de decisión deben encontrarse en zonas en las que haya pocos de estos ejemplos (o instancias).

Manifold assumption

Los datos pueden tener una dimensionalidad alta (muchas características) pero generalmente no todas las características son completamente útiles. Los datos a menudo se encuentran en unas estructuras de más baja dimensionalidad. Estas estructuras se conocen como *manifolds*. Esta suposición indica que si los datos del espacio de entrada se encuentran en estas *manifolds* entonces aquellos puntos que se encuentren en el mismo *manifolds* tendrán la misma etiqueta. [21, 30]

Cluster assumption

Como generalización de las anteriores, aquellos datos que se encuentren en un mismo clúster tendrán la misma etiqueta.

De estas suposiciones se extrae el concepto de «similitud» que está presente en todas ellas. Y en realidad, todas son versiones de la *Cluster assumption*, que dice que los puntos similares tienden a pertenecer al mismo grupo.

Además, la suposición de clúster resulta necesaria para que el aprendizaje semi-supervisado mejore al supervisado. Si los datos no pueden ser agrupados, entonces no mejorará ningún método supervisado [30].

Para tener un punto de vista general, en la figura 3.4 se presenta la taxonomía de los métodos de aprendizaje semi-supervisado.

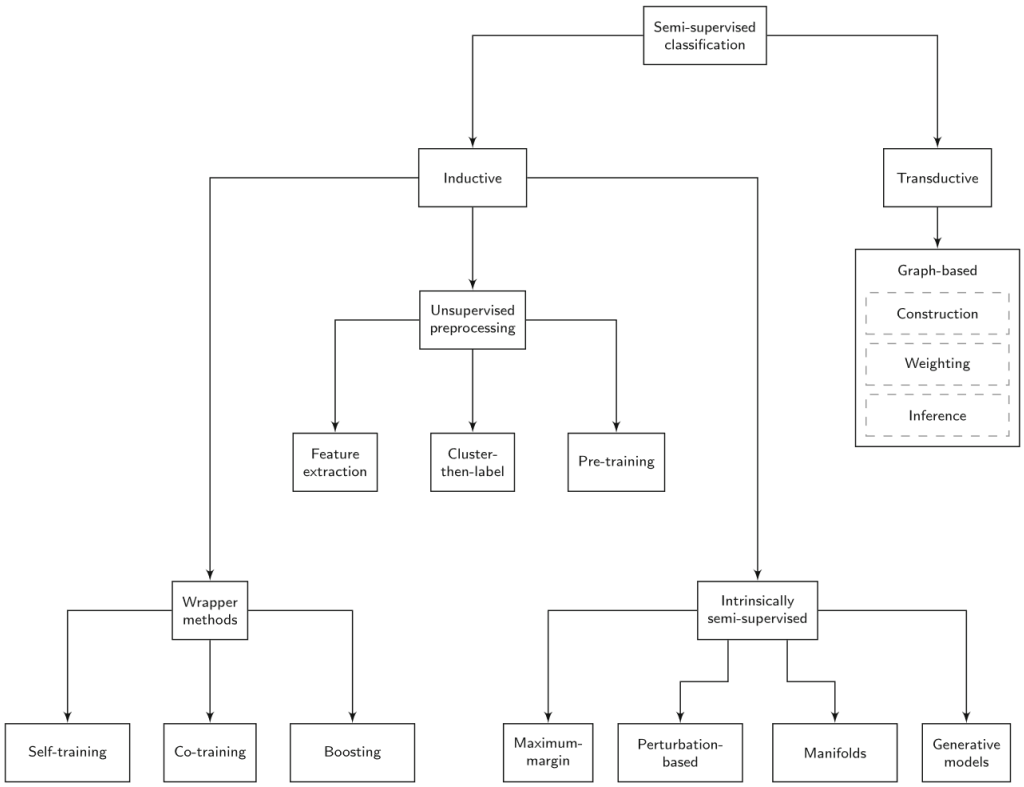

Figura 3.4: Taxonomía de métodos semi-supervisados [30].

El núcleo de este proyecto está basado en los métodos inductivos. Su idea es muy sencilla y está altamente relacionada con el objetivo del aprendizaje supervisado, trata de crear un clasificador que prediga etiquetas para datos nuevos. Por lo tanto, los algoritmos construidos tendrán este objetivo, aunque con un punto más de concreción: el proyecto se centrará en los métodos *wrapper* o de envoltura.

Los conocidos métodos *wrapper* se basan en el *pseudo etiquetado* («pseudo-labelling»), es el proceso en el que los clasificadores entrenados con datos etiquetados generan etiquetas para los no etiquetados. Una vez completado este proceso, el clasificador se vuelve a entrenar pero añadiendo estos nuevos datos. La gran ventaja que suponen estos métodos es que pueden utilizarse con casi todos los clasificadores (supervisados) existentes [30].

3.2. Self-Training

Se trata del método de aprendizaje semi-supervisado más sencillo y «directo». Este método «envuelve» un único clasificador base, que entrena con los datos etiquetados iniciales y aprovecha el proceso de pseudo etiquetado comentado para continuar su entrenamiento.

El método comienza por entrenar ese clasificador con los datos etiquetados que se tienen. A partir de este aprendizaje inicial, se etiqueta el resto de datos. De todas las nuevas predicciones se seleccionan aquellas en las que el clasificador más confianza tiene (de haber acertado). Una vez seleccionados, el clasificador es reentrenado con la unión de los ya etiquetados y estos recién etiquetados. El proceso continúa hasta que se verifica un criterio de parada (generalmente hasta etiquetar todos los datos o un número máximo de iteraciones).

En este proceso, el paso más importante es la incorporación de nuevos datos al conjunto de etiquetados porque, **probablemente**, la predicción sea la correcta. Es importante entonces que el cálculo de la probabilidad se realice correctamente para asegurar que los nuevos datos son de interés. En caso contrario, no es posible aprovechar los beneficios que ofrece self-training [30]. Todo este proceso queda descrito en el algoritmo 1.

Algoritmo 1: Self-Training

Input: Conjunto de datos etiquetados $\boldsymbol{L}$, no etiquetados $\boldsymbol{U}$ y clasificador $\boldsymbol{H}$
Output: Clasificador entrenado

while $|U| \neq 0$
    Entrenar $\boldsymbol{H}$ con $\boldsymbol{L}$
    Predecir etiquetas de $\boldsymbol{U}$
    Seleccionar un conjunto $\boldsymbol{T}$ con aquellos datos que tenga la mayor probabilidad
    $\boldsymbol{L} = \boldsymbol{L} \cup \boldsymbol{T}$
    $\boldsymbol{U} = \boldsymbol{U} - \boldsymbol{T}$
endwhile
Entrenar $\boldsymbol{H}$ con $\boldsymbol{L}$
return $\boldsymbol{H}$

Sobre esta base, el algoritmo tiene muchas formas de diseñarse. En algunos casos la condición de parada suele tomarse como un número máximo de iteraciones. También, la cantidad de datos que se incorporan al conjunto

L (con mayor confianza) puede ser fija, o mediante un límite mínimo de confianza/probabilidad (todas las instancias con mayor probabilidad se añadirían).

3.3. Co-Training

Basado fuertemente en Self-Training, en este caso **varios** clasificadores (normalmente dos) se encargan del proceso e «interactúan» entre sí. Del mismo modo, una vez entrenados predicen las etiquetas de los no clasificados y todos los clasificadores añaden las mejores predicciones.

En [8], Blum y Mitchel propusieron el funcionamiento básico de Co-Training con dos vistas sobre los datos (*multi-view*). Estas vistas corresponden no con subconjuntos de las instancias, sino con subconjuntos de las características de las mismas. Es decir, cada clasificador va a entrenarse teniendo en cuenta características distintas. Idealmente estas vistas tendrían que ser independientes y servir por sí solas para predecir la etiqueta (aunque no siempre se cumplirá). Cuando los clasificadores predicen etiquetas sobre los datos, se seleccionan de ambos los de mayor confianza y se construye el nuevo conjunto de entrenamiento para la siguiente iteración.

Algoritmo 2: Co-Training

Input: Conjunto de datos etiquetados $\boldsymbol{L}$, no etiquetados $\boldsymbol{U}$, clasificadores $\boldsymbol{H}_1$ y $\boldsymbol{H}_2$, p (positivos), n (negativos), u (número de datos iniciales), k (iteraciones)

Output: Clasificadores entrenados

1 Crear un subconjunto $\boldsymbol{U'}$ seleccionando u instancias aleatorias de $\boldsymbol{U}$
2 **for** *k iteraciones*
3 Entrenar $\boldsymbol{H}_1$ con $\boldsymbol{L}$ solo considerando un subconjunto ($\boldsymbol{x}_1$) de las características de cada instancia (x)
4 Entrenar $\boldsymbol{H}_2$ con $\boldsymbol{L}$ solo considerando el otro subconjunto ($\boldsymbol{x}_2$) de las características de cada instancia (x)
5 Hacer que $\boldsymbol{H}_1$ prediga p instancias positivas y n negativas de $\boldsymbol{U'}$ que tengan la mayor confianza
6 Hacer que $\boldsymbol{H}_2$ prediga p instancias positivas y n negativas de $\boldsymbol{U'}$ que tengan la mayor confianza
7 Añadir estas instancias seleccionadas a $\boldsymbol{L}$
8 Reponer $\boldsymbol{U'}$ añadiendo $2p + 2n$ instancias de $\boldsymbol{U}$
9 **endfor**
10 **return** $\boldsymbol{H}_1, \boldsymbol{H}_2$

3.4. Democratic Co-Learning

Yan Zhou y Sally Goldman presentaron en [38] un algoritmo de aprendizaje semi-supervisado en la línea del Co-Training (varios clasificadores). La diferencia sustancial es que los clasificadores base no trabajan con dos (o más) conjunto de atributos (para que cada clasificador utilice uno de ellos), en este caso solo se tiene un único conjunto de atributos (*single-view*).

Partiendo de los datos etiquetados, varios clasificadores realizan votación ponderada sobre los no etiquetados. Lo que quiere decir que, para una instancia, su nueva etiqueta será la que vote la mayoría. Además, para aquellos clasificadores que no votan como la mayoría, la instancia se añade a su conjunto de entrenamiento junto a la etiqueta mayoritaria, de esta forma se «obliga» en la siguiente iteración a «aprenderla». Todo el proceso se repite hasta que no se añadan más instancias a ningún conjunto de entrenamiento, esto ocurrirá cuando no se mejora la precisión pese a la adición de nuevas instancias pseudo-etiquetadas.

Aclaración sobre la predicción final (en combinación): una vez calculadas las confianzas de la instancia a predecir ($\mathbf{x}$) y por cada posible etiqueta, la idea de la combinación es obtener la etiqueta (k, posición en el grupo) con mayor confianza.

Algoritmo 3: Democratic Co-Learning — entrenamiento

Input: Conjunto de datos etiquetados $\boldsymbol{L}$, no etiquetados $\boldsymbol{U}$ y algoritmos de aprendizaje $\boldsymbol{A}_1$,..., $\boldsymbol{A}_n$

for i = 1,...,n
  $L_i = L$
  $e_i = 0$
endfor
repeat
  for i = 1,...,n
    Calcular $\boldsymbol{H}_i$ entrenando $\boldsymbol{A}_i$ con $\boldsymbol{L}_i$
  endfor
  for *cada instancia no etiquetada* $x \in U$
    for *cada posible etiqueta* j = 1,...,n
      $c_j = |\{H_i | H_i(x) = j\}|$
    endfor
    $k = arg\ max_j\{c_j\}$
  endfor
  /* Instancias propuestas para etiquetar */
  for i = 1,...,n
    Utilizar $\boldsymbol{L}$ para calcular el intervalo de confianza al 95 %, $[l_i, h_i]$ de $\boldsymbol{H}_i$
    $w_i = (l_i + h_i)/2$
  endfor
  for i = 1,...,n
    $L'_i = \emptyset$
  endfor
  if $\sum_{H_j(x)=c_k} w_j > max_{c'_k \neq c_k} \sum_{H_j(x)=c'_k} w_j$
    $L'_i = L'_i \cup \{(x, c_k)\},\ \forall i$ tal que $H_i(x) \neq c_k$
  end
  /* Estimar si añadir L'_i a L_i mejora la exactitud */
  for i = 1,...,n
    Utilizar $\boldsymbol{L}$ para calcular el intervalo de confianza al 95 %, $[l_i, h_i]$ de $\boldsymbol{H}_i$
    $q_i = |L_i|(1 - 2(\frac{e_i}{|L_i|})^2)$ /* Tasa de error */
    $e'_i = (1 - \frac{\sum_{i=1}^{d} l_i}{d})|L'_i|$ /* Nueva tasa de error */
    $q'_i = |L_i \cup L'_i|(1 - \frac{2(e_i + e'_i)}{|L_i \cup L'_i|})^2$
    if $q'_i > q_i$
      $L_i = L_i \cup L'_i$
      $e_i = e_i + e'_i$
    end
  endfor
until L_1,..., L_n *no cambien*
return *Combinar*($\boldsymbol{H}_1, \boldsymbol{H}_2, ..., \boldsymbol{H}_n$)

Algoritmo 4: Democratic Co-Learning — predicción (combinación)

Input: $\boldsymbol{H}_1, \boldsymbol{H}_2, ..., \boldsymbol{H}_n$ y $\mathbf{x}$ (instancia)
Output: Hipótesis combinadas (predicción)

for *i = 1,...,n*
    Utilizar $\boldsymbol{L}$ para calcular el intervalo de confianza al 95 %, $[l_i, h_i]$ de $\boldsymbol{H}_i$
    $w_i = (l_i + h_i)/2$
endfor
for *i = 1,...,n*
    if $H_i(\mathbf{x})$ *predice* c_j *y* $w_i > 0.5$
        Añadir H_i al grupo G_j /* j es etiqueta */
    end
endfor
for *j = 1,...,r*
    $\bar{C}_{G_j} = \frac{|G_j|+0{,}5}{|G_j|+1} * \frac{\sum_{H_i \in G_j} w_i}{|G_j|}$
endfor
H predice con el grupo G_k con $k = \arg\max_j(\bar{C}_{G_j})$
return H

3.5. Tri-Training

En Co-Training, los algoritmos son entrenados con el conjunto de datos etiquetados para luego ser reentrenados con los datos no etiquetados que han pasado a estarlo.

En [39] (Tri-Training), Zhi-Hua Zhou y Ming Li, comentan la idea de que para determinar qué ejemplo no etiquetado debe seleccionarse (para etiquetar) y qué clasificador debe tener más influencia, se debe calcular la *confianza de etiquetado* de cada uno de los clasificadores. Sin embargo, esto puede ser muy costoso de calcular.

En este caso, se tendrán tres clasificadores base. La idea es que, en línea de lo anterior, para que un clasificador etiquete un ejemplo sin etiquetar, los otros dos clasificadores deben coincidir en la etiqueta de ese ejemplo. Aunque no es necesario calcular esa *confianza de etiquetado* de cada clasificador en particular.

Una consecuencia de lo anterior es que, si los otros dos clasificadores fallan en su predicción (y coinciden), se añadirá esa etiqueta y se estaría introduciendo un ejemplo mal etiquetado. Sin embargo, en el peor caso, el ruido introducido puede ser compensado si hay suficientes datos nuevos etiquetados (bajo ciertas condiciones [39]).

Una particularidad de Tri-Training es que tanto la cantidad como los ejemplos concretos no etiquetados que son seleccionados para ser etiquetados no será siempre el mismo en cada iteración. Para comprender esto, cada uno de los clasificadores tiene, aparte del conjunto de entrada etiquetado L, un conjunto con los datos recién etiquetados en cada iteración L_i[1]. Y de hecho, este conjunto es «vaciado» entre una iteración y otra. Es utilizado para reentrenar al clasificador al final de la iteración uniendo L y L_i.

Proceso general El primer paso es entrenar a cada uno de los tres clasificadores (h_i, h_j y h_k) mediante una muestra aleatoria del conjunto de entrada L.

A continuación, el algoritmo entra en un bucle cuya condición de parada es que ningún clasificador obtenga nuevos ejemplos para entrenar (ejemplos que eran no etiquetados).

Dentro del bucle y suponiendo la perspectiva del clasificador i lo primero que se hace es «vaciar» el conjunto de datos seleccionados para él, L_i. En

[1] Constituido por esos ejemplos en los que los otros dos clasificadores determinan la misma etiqueta

segundo lugar, se realiza una estimación del error de la combinación de las hipótesis h_j y h_k[2]. Si el error no es menor que el de la iteración anterior e_i, se vuelve al inicio del bucle.

Si el error sí era menor, el conjunto L_i es rellenado con aquellos ejemplos del conjunto de no etiquetados U en los que h_j y h_k predicen la misma etiqueta.

A partir de aquí, se realizan comprobaciones sobre ecuaciones del ratio de ruido y errores [39] que determinarán si los ejemplos realmente serán utilizados para reentrenar. De hecho, es posible también que el conjunto L_i sea reducido.

Al final del bucle si las comprobaciones realizadas determinaron que L_i contiene ejemplos de interés para h_i, dicho clasificador es entrenado con la unión de L y L_i.

Predicción La predicción de una nueva instancia es un proceso sencillo. Cada uno de los clasificadores h_i, h_j y h_k predicen la etiqueta de esta nueva instancia y se retorna la mayoritaria.

[2]El error de clasificación se aproxima dividiendo el número de instancias (de entrenamiento) en las que h_j y h_k se equivocan entre el número de instancias en las que predicen la misma etiqueta [39].

Algoritmo 5: Tri-Training

Input: Conjunto de datos etiquetados $\boldsymbol{L}$, no etiquetados $\boldsymbol{U}$ y algoritmo de aprendizaje *Learn*

for $i \in \{1.,3\}$
  $S_i \leftarrow BootstrapSample(L)$
  $h_i \leftarrow Learn(S_i)$
  $h_i \leftarrow ,5;\ l'_i \leftarrow 0$
endfor
repeat
  for $i \in \{1.,3\}$
    $L_i \leftarrow \emptyset$
    $update_i \leftarrow False$
    $e_i \leftarrow MeasureError(h_j \& h_k)\ (j,k \neq i)$
    if $e_i < e'_i$
      for *every* $x \in U$
        if $h_j\ (x) = h_k\ (x)\ (j,k \neq i)$
          $L_i \leftarrow L_i \cup \{(x, h_j\ (x))\}$
        end
      endfor
      if $l'_i = 0$ /* h_i *no ha sido actualizado antes* */
        $l'_i \leftarrow \lfloor \frac{e_i}{e'_i - e_i} + 1 \rfloor$
      end
      if $l'_i < |L_i|$
        if $e_i |L_i| < e'_i l'_i$
          $update_i \leftarrow True$
        end
        else if $l'_i > \frac{e_i}{e'_i - e_i}$
          $L_i \leftarrow Subsample(L_i, \lceil \frac{e'_i l'_i}{e_i} - 1 \rceil)$
          $update_i \leftarrow True$
        end
      end
    end
  endfor
  for $i \in \{1.,3\}$
    if $update_i = True$
      $h_i \leftarrow Learn(L \cup L_i); e'_i \leftarrow e_i; l'_i \leftarrow |L_i|$
    end
  endfor
until *ningún* $h_i\ (i \in \{1.,3\})$ *cambie*
return $h(x) \leftarrow arg\ max_{y \in label} \sum_{i:h_i(x)=y} 1$

3.6. Técnicas de tratamiento de datos

En el proceso de minería de datos y aprendizaje automático una de las primeras etapas, y principales, es el preprocesamiento de datos. Este concepto hace referencia a la manipulación, eliminación y/o transformación de datos antes de ser usados para mejorar el rendimiento del proceso [35].

El tratamiento de datos también es particularmente útil en el área de la interpretación y la visualización de datos. En ocasiones, el conocimiento obtenido es difícil de interpretar con reglas o puras matemáticas.

En esta sección se van a comentar algunos conceptos que se han aplicado en el área de tratamiento de datos (tanto en la parte de preprocesamiento como visualización).

Codificación de variables categóricas

Este es el único tratamiento que el proyecto tiene incluido dentro del preprocesamiento de los datos (el usuario es el encargado de tratar el resto de cuestiones). Además, solo se aplica al atributo de la clase.

Muchos algoritmos de aprendizaje automático no son capaces de trabajar con variables categóricas (no numéricas). Además, la categorización está presente en muchos conjuntos de datos, ya que son muy útiles para aportar significado (los números serían más difíciles de entender). Debido a las limitaciones de los algoritmos, estos valores categóricos deben ser codificados como unos valores numéricos.

Una de las ideas más directas (y la aplicada en el proyecto) es la «Codificación de etiquetas» (*Label encoding*). Esta técnica simplemente trata cada valor único de etiqueta[3] como un valor numérico también único.

Etiquetas originales	Etiquetas codificadas
Amarillo	0
Rojo	1
Verde	2
Amarillo	0
Azul	3

Tabla 3.1: Codificación de etiquetas

[3]Entendiendo etiqueta como un valor del atributo de clase, no de una característica.

Desde el punto de vista del algoritmo, no necesita tener esa información que aportaba la categoría (y que a nosotros sí nos ayuda). Simplemente, con mantener la misma codificación para cada etiqueta, es suficiente.

PCA (*Principal Component Analysis*)

Desde el punto de vista de la interpretación de los datos, PCA puede ser muy útil (también se utiliza en el preprocesamiento en algún proceso de minería).

El análisis de componentes principales es un algoritmo matemático que reduce la dimensionalidad de los datos manteniendo la mayoría de las variaciones en los datos. Esto lo realiza identificando componentes principales [24]. Una componente principal es una nueva variable que es combinación lineal de las variables originales con la particularidad de que esa combinación de variables mantiene la mayor cantidad de varianza del conjunto de datos original. La idea es mantener los patrones escondidos en los datos, en las componentes principales, para que, si se aplica a un proceso de minería de datos, se pueda seguir extrayendo conocimiento.

Aplicar PCA permite utilizar solo unas pocas componentes que pueden ser graficadas y hacer posible la visualización de diferencias y similitudes.

Esta idea es la que se ha usado para las visualizaciones del proyecto. En muchos casos se tendrá una dimensionalidad alta y por lo tanto no se puede visualizar en dos dimensiones.

Estandarización

La estandarización en el campo de tratamiento de datos hace referencia a la transformación de los valores para centrarlos en una media y con una cierta desviación estándar (típicamente media cero y desviación 1).

$$z = (x - u)/s \tag{3.1}$$

z es el nuevo valor, x el valor original, u la media y s la desviación típica.

La estandarización es útil cuando no se quiere establecer un rango fijo de valores (como en la normalización), y además, los *outliers* (en castellano valores atípicos) no afectan demasiado, ya que se considera la media de todos ellos.

En el proyecto, es una opción que se le da al usuario para mostrar los datos finales, ya que permite mantener las relaciones entre los datos pero compactándolos lo suficiente para una buena visualización.

HTTP (Descripción general)

Al ser una aplicación Web, este proyecto tiene alta relación con este protocolo. En el resto del apartado se van a documentar los conceptos que se han utilizado en el desarrollo.

Hypertext Transfer Protocol (HTTP) o Protocolo de Transferencia de Hipertexto es un protocolo de la capa de aplicación[4] para transmitir documentos. Se trata de un modelo cliente-servidor clásico, el cliente realiza una petición al servidor, y espera a que este le devuelva una respuesta [10].

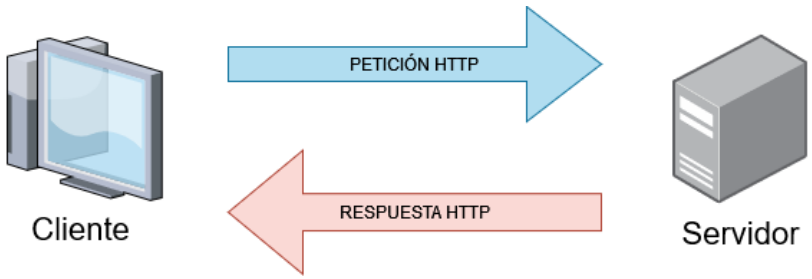

Figura 3.5: Diagrama cliente-servidor.

Características de HTTP [2]:

- Protocolo sin conexión: esto significa que la comunicación antes comentada ocurre sin realizar un acuerdo previo entre el cliente y el servidor.
- Independiente al tipo de contenido: quiere decir que HTTP no limita los datos que se pueden enviar, siempre y cuando, tanto el cliente como el servidor sepan manejar esos datos.
- Protocolo sin estado: cliente y servidor solo saben de su existencia durante la comunicación, no retienen la información de peticiones anteriores.

[4]Permite a las aplicaciones trabajar con los servicios de las capas inferiores y define protocolos para el intercambio de datos [33]

Estructura de las peticiones y respuestas

La realidad es que, peticiones y respuestas, tienen una estructura muy similar. Para las peticiones:

- **Línea de inicio**, formada por:

  El método HTTP (se describirán a continuación), indica la operación a realizar.

  El objetivo, generalmente una URL. Para algunos métodos, aquí ser incorporan también los parámetros de la petición (`?param=X`).

  Versión del protocolo.

- **Cabeceras**: Son «metadatos» que proporcionan información sobre la petición. Cada una de ellas (puede haber varias) es un par nombre-valor. Por ejemplo, especificar el tipo de contenido de la petición se haría con «`Content-Type: application/json`».

- **Cuerpo**: Contiene información adicional para el servidor, pudiendo enviar cualquier dato. No todas las peticiones llevan cuerpo, una petición GET no lo necesita, pero sí un POST.

Para las respuestas:

- **Línea de estado**, formada por:

  Versión del protocolo.

  Código de estado que indican el estado de la petición (se describirán a continuación).

  Descripción textual del código de estado (para poder interpretarlo).

- **Cabeceras**: Del mismo modo que las peticiones, son «metadatos» que proporcionan información sobre la petición. Cada una de ellas (puede haber varias) es un par nombre-valor.

- **Cuerpo**: Para aquellas respuestas que tengan cuerpo (algunas no lo tienen), contiene los datos solicitados o generados por el servidor.

```
acha@DESKTOP-9MJIVL5:~$ curl -i https://vass.dmacha.dev
HTTP/1.1 200 OK
Server: nginx/1.18.0 (Ubuntu)
Date: Mon, 08 May 2023 17:39:52 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 7380
Connection: keep-alive
Vary: Cookie

<!DOCTYPE html>
<html lang="es">
<head>
```

Figura 3.6: Ejemplo de respuesta con su estructura.

Como se ha visto en las operaciones HTTP, todas tienen un indicativo, llamado método, que permite distinguir qué tipo de operación se quiere realizar. El protocolo HTTP soporta una enorme cantidad de métodos, algunos de ellos son para contextos muy específicos. Los más utilizados (constantemente en entornos reales) son [36]:

- **GET**: Solicitud de un recurso. Solo para recuperar datos, sin otras operaciones. Generalmente a la URL se le añaden parámetros que configuran esta solicitud para que el servidor pueda personalizar las respuestas.
- **POST**: Solicitud que envía datos en el cuerpo de la petición (es el método de los formularios HTML, por ejemplo). Tiene diversas funciones, como crear un recurso o actualizarlo. Sin embargo, pese a que no está pensado para ello, en muchas ocasiones, también se utiliza para solicitar información.

  Entre otras cosas, solicitar información con POST, permite reducir la longitud de las URLs para operaciones GET demasiado extensas (incluso que puedan sobrepasar el límite de caracteres) y también permite ocultar información comprometida [3].

  Las peticiones GET son parametrizadas mediante la propia URL, mientras que en POST, los parámetros pueden incorporarse al cuerpo de la petición, ocultándolos (interesante, por ejemplo, si no se quiere compartir identificadores).
- **PUT**: Método similar a POST, también envía datos al servidor, pero en este caso está orientado específicamente a la actualización de los recursos.

- **DELETE**: Método para eliminar recursos.

Los códigos de estado antes mencionados son parte de la respuesta del servidor a los clientes. Indicarán, por lo general, el éxito o fracaso de las operaciones.

Hay aproximadamente 60 códigos distintos, cada uno brinda la posibilidad de personalizar las respuestas dependiendo del contexto. Todos estos códigos pueden ir acompañados de la estructura antes comentada. Por ejemplo, para una petición exitosa, la respuesta de un POST podría tener un `cuerpo` con más información junto con el código *200*.

Los códigos más utilizados son:

- **200**: *OK*, respuesta a peticiones HTTP exitosas.
- **400**: *Bad Request*, el servidor no puede procesar la petición debido a una petición mal construida.
- **401**: *Unauthorized*, no se tiene las credenciales de autenticación válidas para acceder al recurso solicitado.
- **403**: *Forbidden*, el servidor entiende la petición pero la rechaza.
- **404**: *Not Found*, el recurso no puedo ser encontrado.
- **405**: *Method Not Allowed*, el método no es soportado para acceder al recurso. Por ejemplo, utilizar un GET cuando está pensado para utilizarse un POST.
- **500**: *Internal Server Error*, respuesta genérica para errores internos, sin mensaje específico de los mismos.

HTTP Cookies El concepto de las *cookies* es sencillo, son pequeños bloques de datos que el servidor manda al navegador del usuario mientras realiza peticiones HTTP a él. Una vez que el cliente ha almacenado esta información, puede usarla para agilizar ciertos pasos en las siguientes peticiones. Cuando vuelva a acceder, junto con la petición «normal» enviará esta porción de datos, que el servidor puede interpretar y reconocer. Por ejemplo, pueden guardarse los inicios de sesión, preferencias o, en entornos comerciales, el comportamiento del usuario.

La idea interesante de las *cookies* es que permite almacenar **información de estado** para el protocolo HTTP, que es **sin estado** (por sí solo no puede guardar esta información).

3.7. CSRF (Cross-Site Request Forgery)

Cross-Site Request Forgery (falsificación de petición en sitios cruzados) es un ataque sobre aplicaciones web vulnerables en las que se utiliza a un usuario autenticado en la misma (o en el que la web confía) para enviar una petición maliciosa. Los ataques CSRF explotan la confianza que una web tiene con un usuario, al no poder diferenciar entre una petición generada por el usuario (la persona) y la petición generada por el usuario, pero sin su consentimiento [1] (generalmente de forma cruzada, proveniente de otro contexto).

Se presenta un ejemplo explicativo:

Situación

Web vulnerable: `https://ejemplo.es`

Usuario: David (con sesión iniciada)

URL de cambio de contraseña: `https://ejemplo.es/cambiarcontrasena` (permite un parámetro «nueva» mediante POST para especificar la nueva contraseña)

El servidor ha «securizado» parcialmente la URL anterior comprobando que el que accede a esa URL es el propio usuario con sesión iniciada.

Atacante

1. Ha hecho ingeniería social para saber el Email de David: `david@ejemplo.es`.
2. Envía un correo electrónico con un enlace que parece de Google.
3. En realidad, todos los botones de esa página envían un formulario (POST) hacia `https://ejemplo.es/cambiarcontrasena`.
4. El formulario tiene un campo oculto:

```
<input type="hidden" name="nueva" value="hackeado"/>
```

David

1. Recibe el correo en su bandeja de entrada.
2. No se percata de que no es la página original de Google y entra en ella.
3. Busca información en Google y pincha en un botón.
4. Se realiza la petición POST a la aplicación web vulnerable en la que ha iniciado sesión.
5. El servidor procesa la petición pues cree que es el propio usuario el que ha construido la petición conscientemente.
6. La contraseña ha cambiado a «hackeado».

Atacante

Con el Email «david@ejemplo.es» y la contraseña «hackeado», accede a la Web «ejemplo.es» como si fuera David.

Solución Dada la vulnerabilidad comentada anteriormente, la solución pasa por idear una forma de saber que esa petición se ha creado desde la propia aplicación (no por terceros). Por lo tanto, la idea más efectiva es la creación de un *CSRF token* único para los usuarios, que el servidor (aplicación) genera y «guarda» en la sesión de los mismos. Este «token» es incrustado como un parámetro oculto en las peticiones (en los formularios) y al realizar el envío del formulario, el servidor comprobará que el «token» recibido era el correcto para el usuario.

Técnicas y herramientas

En este apartado se presentarán las técnicas y herramientas que se han utilizado para el desarrollo del presente proyecto. En algunos de los casos la experiencia u otros aspectos han hecho decantarse por una u otra o simplemente se seleccionaron directamente.

4.1. Técnicas

Scrum Es un marco de trabajo de gestión de proyectos ágiles. Aunque aplicado también a otros trabajos, se utiliza principalmente para el desarrollo de *software* de calidad y de la forma más rápida posible.

Scrum se define como un marco de trabajo ligero, fácil de entender y difícil de dominar [29]. Scrum permite la entrega de valor de forma incremental y colaborativa.

Además, Scrum define una serie de reuniones, herramientas y roles que permiten la gestión del trabajo de forma eficiente.

Las **reuniones** (también llamadas ceremonias) son las siguientes:

- ***Sprint***: periodo de tiempo en el que se realizan las tareas para crear un incremento del producto.
- ***Sprint planning*** (planificación del *sprint*): es una reunión que inicia un *sprint* con una duración máxima de 8 horas, en ella se establece el objetivo del sprint.
- ***Daily Scrum***: es una reunión realizada cada día para comprobar el progreso del trabajo marcado para el Sprint, su duración varía entre 10 y 15 minutos.

- ***Sprint review***: es una demostración del trabajo realizado durante el *sprint* para comprobar que el progreso es el adecuado para alcanzar el objetivo.
- ***Sprint retrospective***: es la última reunión del *sprint* para determinar que es lo que ha ido como se planeaba y qué es lo que no.

Los **artefactos** (o documentos) más relevantes son los siguientes:

- ***Product Backlog***: por su nombre, hace referencia al producto completo y contiene todas las funcionalidades y requisitos que debe tener. Este documento se actualiza periódicamente.
- ***Sprint Backlog***: contiene las funcionalidades y requisitos del *product backlog* que se planean cumplimentar en el *sprint*.
- ***Incremento***: es el resultado de todos los elementos completados durante el *sprint*, que añade valor y es potencialmente entregable.

Y por último, los **roles** que intervienen en todo este proceso ágil son [26]:

- ***Scrum Master***: es el encargado de establecer Scrum, de que todos lo entiendan y lo apliquen correctamente.
- ***Product Owner***: se encarga de maximizar el valor del producto. Esto lo realiza principalmente manejando el *product backlog* (ordenando) y asegurando que es visible y entendido.
- ***Desarrolladores***: son los encargados de llevar a cabo el incremente de cada *Sprint*.

PEP8 Las siglas de PEP corresponden con *Python Enhacement Proposal*. Un PEP es un documento que la comunidad Python utiliza para describir características de Python.

PEP8 se trata de uno de estos documentos, describe una guía sobre cómo escribir código en Python junto con las mejores prácticas [13]. En definitiva, es una guía de estilos. Fue escrita por Guido van Rossum, Barry Warsaw, and Nick Coghlan en 2001. Es la guía de estilo que se ha usado en todo el desarrollo Python del proyecto.

La necesidad de utilizar PEP8 recae concretamente en la legibilidad. La idea subyacente de esto es que el «código es leído mucho más a menudo de lo que es escrito» (Guido van Rossum). Cuando se ha escrito un código, lo lógico sería que al volver a él, el desarrollador sepa qué es lo que esa porción de código hace. Sin seguir unas guías de estilo esta tarea es mucho más difícil (nombres de variables sin contexto, código amontonado, comentarios en línea y de funciones escasos e inútiles...).

A grandes rasgos, PEP8 considera los siguientes aspectos [31]:

- Diseño de código: como las indentaciones de 4 espacios o longitud máxima de líneas de 79 caracteres.
- Cadenas de caracteres: con libertad si utilizar comillas dobles o simple.
- Espacios en blanco: recomendaciones sobre cuándo dejar o no espacios.
- Comentarios: mantener los comentarios actualizados, claros y fácilmente entendibles. Los comentarios en línea deberían ser útiles (sin obviedades) y los comentarios de documentación siempre deben aparecer en todas las funciones (no privadas) después del «`def`».
- Convenciones de nombres: con nombres a evitar (como 'O') o cómo poner nombres correctos de paquetes, módulos, clases, funciones, constantes...

Algo que ha facilitado mucho la aplicación de esta guía de estilo es la capacidad de la propia herramienta utilizada (Pycharm) de detectar automáticamente la violación de algunas de estas recomendaciones.

Diseño adaptable Una de las herramientas utilizadas (ver siguiente sección) es Bootstrap, un conjunto de estilos ya predefinidos que pueden utilizarse sin necesidad de tener que codificar CSS. La idea de la aplicación web desarrollada es que tenga la máxima accesibilidad. Esto se consigue construyendo un diseño que sea apto tanto para los dispositivos móviles como para los ordenadores, lo que se denomina estilo adaptable («responsive»). Se trata de una técnica de diseño web para adaptar la visualización de la página al dispositivo desde el que se accede, haciendo que sea atractiva a una mayor cantidad de usuarios. El diseño «responsive» se consolida como una de las mejores prácticas en el diseño web [4]. En toda la aplicación (excepto las visualizaciones que requieren de mucho más espacio), cuando el tamaño de la pantalla se reduce lo suficiente, los contenidos se adaptan y se posicionan de tal forma que puedan seguir utilizándose.

Internacionalización (i18n[5]) Se trata de una técnica/procedimiento por el que se desarrolla el software para poder ser adaptado y localizado posteriormente a otras culturas y lenguajes [20]. Concretamente, este proyecto se está centrando en la parte lingüística, con Inglés y Español como los idiomas principales. La idea de esta técnica es conseguir esto, pero haciendo un código completamente neutral, es decir, permitir que todo lo que se ha hecho para un idioma, pueda hacerse a posteriori con la misma facilidad. En este entorno de Python y Flask existe la biblioteca Babel que permite capturar el texto (previamente anotado) y sustituirlo en cada momento por las palabras adecuadas a la localización del usuario (o bien con un cambio manual mediante botones). Por así decirlo, en vez de codificar a un solo idioma, las palabras del idioma original se convierten en claves, que luego son sustituidas cuando el usuario accede.

4.2. Herramientas

PyCharm Se trata de un entorno de desarrollo integrado («IDE») desarrollado por JetBrains. Está creado para la programación en lenguaje Python y aunque no se ha usado en este proyecto también Java. Este «IDE» se ha convertido junto con Visual Studio Code y Jupyter en el más utilizado por los desarrolladores.

Ofrece multitud de funcionalidades (https://www.jetbrains.com/es-es/pycharm/features/):

- Inspección de código.
- Indicación de errores (compilación).
- Refactorización de código automático (rápidas y seguras).
- Depuración.
- Pruebas.
- Herramientas para bases de datos.
- Integración con Git.

Estas son solo algunas de las muchas funcionalidades que permite y que han hecho que se haya seleccionado para este proyecto. Además, dado que

[5]Denominado así por las 18 letras entre la «i» y la «n» en Internationalization

el proyecto tiene una fuerte componente de desarrollo Web, PyCharm tiene una integración completa con estos ámbitos, pudiendo desarrollar también de forma nativa con JavaScript o lenguajes de marcas (HTML o CSS).

También se ha de destacar otra funcionalidad que ha sido de gran utilidad. JetBrains tiene una herramienta específica para la conexión con bases de datos `DataGrip`. Sin embargo Pycharm, de forma nativa, tiene ciertas de estas utilidades. En concreto, en este proyecto se ha usado la visualización del contenido de la base de datos junto con el diagrama de tablas/entidades (que muestra relaciones, claves primarias, foráneas e incluso los tipos de datos).

Finalmente, el factor decisivo para usar PyCharm y concretamente gracias a la Universidad de Burgos, es que el alumnado tiene acceso a la edición «Professional», que habilita ese desarrollo web, por ejemplo.

Esta aplicación ya había sido usada con anterioridad y por tanto su aprendizaje básico no era necesario.

Visual Studio Code Se trata de un editor de código fuente desarrollado por Microsoft. Es el editor por excelencia en todo el ámbito de la programación pues permite la programación en casi cualquier lenguaje de programación, haciéndole muy versátil y un «todo en uno».

Ofrece todas las funcionalidades que cabe esperar (incluyendo a las que ofrece PyCharm): sintaxis, depuración, personalización, integración git...

Además del núcleo propio del editor y su constante actualización, uno de sus puntos fuertes son las extensiones, que empresas o incluso la comunidad, desarrollan y que permiten agilizar en lo posible las tareas del programador.

A pesar de todas las ventajas, el manejo de PyCharm ha resultado más sencillo de utilizar (durante la experiencia previa a este proyecto) en el desarrollo del software. Pero debido a su versatilidad y al desarrollo de esta documentación en Latex, VS Code es el editor adecuado para ello. Gracias a las extensiones y a la instalación local de LaTeX, permite tener un control completo para la creación de este tipo de documentos.

GitHub Desktop Es una aplicación que permite interactuar con GitHub utilizando una interfaz gráfica respecto al manejo tradicional mediante la línea de comandos. Permite realizar las operaciones más comunes y básicas de Git.

Pese a que no tiene toda la funcionalidad que sí ofrece la línea de comandos no se previó un uso muy exhaustivo de Git y, por tanto, es suficiente («Commit»,«Push»,«Pull»,«Merge»...).

Además, puede visualizarse cada cambio respecto a la última versión de un vistazo y seleccionar dinámicamente los cambios que se deseen aplicar.

Aún con todo esto, no es más que una aplicación para agilizar el proceso de control de versiones.

Diagrams.net (Draw.io) Es una herramienta online y de escritorio para la creación de diagramas. Con ella se pueden crear diagramas de flujo, de UML o de red entre otras muchas posibilidades. Para crear estos diagramas, posee un listado de elementos (organizados) de cada tipo de diagrama (actores, cajas de clases, rombos de relaciones, bloques...). Pero aunque esté pensado específicamente para diagramas, existen muchos más elementos como flechas, figuras e iconos que permiten no solo crear diagramas, sino todo tipo de cosas.

Con esta herramienta se han creado prácticamente todos los diagramas de la documentación.

PuTTy La aplicación Web del proyecto se encuentra en funcionamiento en internet a través de `https://vass.dmacha.dev`. Para gestionar el servidor en el que se tiene lanzada se ha utilizado PuTTy.

Se trata de una aplicación de escritorio de código abierto y gratuita que permite emular una terminal de comandos. Soporta múltiples protocolos como SSH (*Secure Shell*), que es el que se ha usado mediante la generación de clave pública y privada. Una vez que se establece la conexión con el equipo remoto, se convierte en la misma terminal que podría visualizarse en la pantalla de este.

Además de esta funcionalidad, PuTTy alberga muchas otras utilidades como la de generación de claves o incluso la transferencia de ficheros con SFTP (*Secure File Transfer Protocol*).

Bibliotecas Python Para el proyecto se están utilizando múltiples bibliotecas que facilitan en gran medida el desarrollo del mismo. Implementan funcionalidades que pueden ser utilizadas directamente.

- **Pandas**: Biblioteca para el manejo de estructuras de datos que implementa muchas operaciones útiles (guardado/lectura CSV, reordenaciones, divisiones...) y de manera eficiente.
- **Babel**: Colección de utilidades para la internacionalización y localización de aplicaciones en Python.
- **Flask**: Es un micro Framework escrito en Python para el desarrollo de aplicaciones web. Impone además su formato de plantillas (Jinja2).
- **Flask-babel**: Es una extensión de Flask que permite la internacionalización concreta de aplicaciones Flask (con la ayuda de Babel).
- **Flask-SQLAlchemy**: Biblioteca que actúa de extensión de Flask y añade el soporte para SQLAlchemy. No modifica el comportamiento de este último.
- **Flask-Login**: Biblioteca complementaria a Flask para el manejo de sesiones de usuarios, automatizando los inicios y cierres de sesión entre otras cosas.
- **Scikit-learn**: Librería más extendida de aprendizaje automático («machine learning») para Python. Junto con Flask es el núcleo del proyecto, ya que utiliza muchos de sus paquetes implementados.
- **Numpy**: Es una Biblioteca especializada en el cálculo numérico y análisis de datos. Se caracteriza por su eficiente manejo de grandes volúmenes de datos gracias a su parcial implementación en lenguaje C (mucho más eficiente).
- **Scipy**: Biblioteca que incluye algoritmos matemáticos.
- **Setuptools**: Biblioteca que permite crear e instalar paquetes de software de Python.
- **Pytest**: Es un *framewrok* de pruebas unitarias para Python.
- **Sslearn**: Biblioteca para aprendizaje automático en conjuntos de datos Semi-Supervisados como extensión de Scikit-learn. Creada por José Luis Garrido-Labrador.
- **Werkzeug**: Biblioteca para aplicaciones WSGI (Web Server Gateway Inteface). Contiene una colección de utilidades para este tipo de aplicaciones.

- **Matplotlib**: Biblioteca para la creación de gráficos en 2D.
- **WTForms**: Biblioteca para la creación de formularios seguros. Permite añadir múltiples validaciones en los campos de los formularios. Además, son útiles para crear «plantillas» base existen muchos formularios comunes.
- **Flask-WTF**: Biblioteca que actúa de extensión de Flask y añade el soporte para WTForms.
- **SQLAlchemy**: Biblioteca que incorpora una serie de herramientas SQL junto con el «mapeo» de los objetos relacionales. Ha resultado especialmente importante para desvincular la instancia de la base de datos (SQLite, MySQL...) de las consultas y operaciones.

Bibliotecas Web Para el desarrollo web conviene comentar la utilización de dos bibliotecas que han resultado fundamentales.

Bootstrap: No es considerado una biblioteca al uso, sino que está categorizado como un *framework* de CSS. Fue lanzado en 2011, con gran cantidad de actualizaciones y de lanzamientos de nuevas versiones. La versión más reciente y la utilizada en el proyecto es Bootstrap 5, que mejora el código fuente de la versión 4 y añade propiedades nuevas.

La intención del equipo de Bootstrap es promover el desarrollo web con novedades de CSS y menores dependencias. Su principal objetivo es la creación de páginas y aplicaciones webs con diseño adaptable (adaptados al tamaño del dispositivo de acceso) de la forma más sencilla posible. Contiene infinidad de plantillas creadas con HTML, CSS y JavaScript para componentes de la interfaz que el usuario simplemente puede hacer uso de ellas. En HTML, Bootstrap se incorpora en las clases de los componentes y en su página oficial se tiene toda la documentación[6].

Bootstrap Icons: es una biblioteca aparte de Bootstrap con iconos SVG. Los iconos son tratados como clases que se incorporan a los elementos HTML. Desde https://icons.getbootstrap.com/ se pueden encontrar todos los iconos que incluye la librería junto con el código a utilizar para copiar y pegar directamente.

D3.js: su nombre significa Data-Drive-Documents, es una biblioteca de JavaScript que permite representar datos en distintas visualizaciones mediante HTML, SVG y CSS. D3 trabaja sobre el Modelo de Objetos

[6]Bootstrap: https://getbootstrap.com/

del Documento (*Document Object Model*) para aplicar transformaciones dirigidas por datos. Todas las visualizaciones presentadas en la aplicación web desarrollada se han realizado con D3. D3 permite un amplio abanico de posibilidades y creatividad, pueden crearse visualizaciones más bien estáticas y añadir estilos, anotaciones o leyendas entre otra cosas, pero también permite la creación de gráficos altamente interactivos incluyendo «zoom», aprovechar los eventos en el DOM o transiciones completamente personalizadas.

Para este caso se cree necesario comentar y justificar el uso de D3 ya que en el ámbito de las visualizaciones, existe una gran cantidad de librerías disponibles como JSAV (*JavaScript Algorithm Visualization Library*) o Algorithm-Visualizer (contexto concreto de este proyecto) o incluso otras grandes librerías como Anime.js, Chart.js o FusionCharts. Sin embargo, durante el periodo de estudio de las distintas herramientas disponibles, D3 es superior a la demás en su documentación. Tanto porque existen páginas dedicadas a mostrar ejemplos de uso de D3, como por la gran cantidad de foros que se han creado solucionando dudas y problemas. Sobre las últimas bibliotecas mencionadas (y otras grandes existentes), D3 está particularmente pensada para la completa personalización y aunque estas también, no al nivel de D3.

La desventaja de D3 es que tiene una curva de aprendizaje bastante grande, pero gracias a la documentación en internet, esto puede ser solventado sin grandes problemas.

DataTables: Es un complemento para la librería JQuery. Su objetivo es la creación de tablas dinámicas e interactivas de la forma más sencilla posible. Este complemento se ha utilizado en todos los apartados de los usuarios, tanto el espacio personal como el panel de administración. La elección de esta biblioteca es por recomendación de los tutores. Pese a que las tareas relativas a los usuarios fue compleja y se tardaron dos Sprints en terminar todo lo relativo a las tablas, DataTables tiene una muy buena documentación. Además, en el momento de consultar una duda, en su propio foro otros usuarios ya habían tenido los mismos problemas.

A la hora de crear las tablas, tiene multitud de opciones, tanto para la tabla general, como para celdas, columnas o filas concretas. Como más importante, existen opciones para «renderizar» código HTML en unas columnas (típica columna de Acciones, por ejemplo).

También es de destacar que se ha incluido la extensión «responsive» de DataTables para habilitar el estilo adaptativo de las tablas.

Aspectos relevantes del desarrollo del proyecto

En este apartado se van a comentar los aspectos más interesantes o que han influido en el desarrollo. Por lo general, estos aspectos vendrán acompañados de tomas de decisiones que se tuvieron que hacer, se argumentará y explicará el desarrollo final de estos aspectos.

5.1. Elección del proyecto

La idea de este proyecto no fue propia, ambos tutores tenían en mente realizar una aplicación enfocada a la docencia de algoritmos de aprendizaje semi-supervisado. Durante los meses anteriores aparecieron diversos proyectos de muy diferente índole que podía realizar. La realidad es que la inteligencia artificial es un campo que me atrae mucho y que incluso me abre la mente a proyectos personales. Dentro de la oferta de proyectos, el que iba en la línea de lo que quería hacer era este. Al principio era un poco reacio a la idea de realizar una aplicación tanto web como de escritorio. Demasiado enfocado en ello, quería aplicar los algoritmos a algo concreto. Sin embargo, pensándolo bien, el hecho de poder aprender algoritmos (e incluso la metodología que su desarrollo conlleva de investigación y entendimiento) pero además, poder desarrollar habilidades en web (o escritorio, aunque finalmente no ha sido así), fue la clave para decantarme por el proyecto. Además, dado que disfruto mucho «cacharreando», los tutores también me comentaron que este proyecto podría tenerme entretenido muchas horas para desarrollar ambas partes. Con todo ello, parecía muy interesante y muy provechoso para mi aprendizaje elegir este proyecto.

5.2. Versión de Python

Al comienzo del proyecto, se valoró la versión de Python en la que realizarlo. En un principio parecía importante abarcar el mayor número de equipos en los que el proyecto podía ser instalado y se pensó en alguna versión desde la 3.7. Sin embargo, pese a que en la biblioteca de algoritmos que se iba a programar sí que podía tener sentido aumentar los posibles usuarios, la realidad es que el objetivo del final proyecto es una aplicación web completamente nueva. Se supone además que, simulando un entorno completamente real y/o empresarial, el equipo podría tener a su disposición servidor/es propio (o por lo menos, configurables). En este sentido, dado que el usuario solo necesita acceder a la Web, no tiene por qué conocer ni tener instalada una versión u otra. Por todo esto, se pensó en utilizar versiones más recientes.

En el momento de inicio del proyecto, la más reciente era la 3.10, esto también es una ventaja en el medio plazo debido a que el periodo de actualizaciones y de soporte para esta versión termina a finales de 2026 (mientras que algunas anteriores finalizan en 2024 o incluso 2023).

5.3. Utilidades para los algoritmos

Antes de realizar la aplicación web, pero previendo lo que podía encontrarse. Surgieron dos grandes problemas, el tratamiento de los datos que utilizan los algoritmos (etiquetados, no etiquetados, particiones...) y por otro lado, cómo ajustar estos datos y comunicarlos a la Web cuando los requiera.

No resultaba correcto vincular todos estos pasos en el código de los algoritmos, ni tampoco en el de la propia aplicación. Tenía mucho más sentido crear unas utilidades que actuasen de intermediarias en ciertos pasos del procedimiento.

Para el primer problema se crearon tres utilidades principales:

Un «particionador» de datos que se encargara de dividir los datos en conjunto de entrenamiento y test. Pero que además, si el conjunto estaba pensado para aprendizaje supervisado, generase aleatoriamente datos no etiquetados.

Un codificador de etiquetas para transformar las etiquetas nominales en numéricas (necesarias para los algoritmos). De base, el codificador de etiquetas que propone SKlearn podría ser suficiente. Sin embargo, era necesario

por un lado no tratar los datos no etiquetados (internamente tratados como -1s) y por el otro devolver de alguna forma las transformaciones realizadas Es decir, a qué clase nominal corresponde cada número codificado.

Y por último, un cargador de conjuntos de datos («datasetloader») que automatizara toda la lectura de un fichero ARFF o CSV, conversión a DataFrame y las transformaciones de etiquetas (utilizando la utilidad anterior). Y que además devuelva los datos de los atributos ($\mathbf{x}$) y por separado las etiquetas (y). Es decir, un cargador cuyo resultado pueda ser introducido en los algoritmos.

Sobre el segundo problema (datos válidos para la aplicación web) se creó una utilidad encargada de transformar el conjunto de datos a dos dimensiones. Esto es, transformar los atributos (que pueden ser más de dos) a exactamente dos, para poder representarlo en dos dimensiones. Sin embargo, cuando se avanzó un poco en el desarrollo, era obvio que no interesaba modificar el conjunto de datos, sino el resultado de la ejecución de la aplicación web. Fue cuando se implementó y visualizó Self-Training cuando se pudo ver esta casuística. Finalmente, lo que se hace es, transformar la estructura de datos que retorna la ejecución de los algoritmos (consultar anexo D) que incluye todos los datos (y todos sus atributos), a dos dimensiones (solo los atributos, no el «target» ni el resto de columnas). El usuario puede seleccionar dos posibilidades, o realizar PCA o seleccionar él mismo dos atributos del conjunto.

En ambos casos apareció la decisión de la estandarización. Al principio se realizaba en ambos casos directamente, pero finalmente se permitía al usuario elegir si estandarizar o no.

Como se puede ver, no se está normalizando sino estandarizando. Esta decisión es debida, principalmente, al efecto de los *outliers* (valores atípicos). Un valor extremo en normalización puede hacer que la visualización sea muy pobre, juntando la mayoría de los puntos a una zona. Con la estandarización este efecto es mucho menor, pues se tiene en cuenta la media de los valores.

5.4. Desarrollo de los algoritmos

Junto con la aplicación web, este desarrollo es el más importante realizado. Ha supuesto un verdadero reto documentarse con los artículos científicos de estos algoritmos. Al fin y al cabo, no se tienen absolutamente todos los conocimientos del aprendizaje y en muchas ocasiones se «perdía» más tiempo leyendo y entendiendo que programando. Sí que es verdad que los algoritmos que se han desarrollado tienen ideas bastante lógicas y directas.

Para desarrollar un algoritmo lo primero que se realizaba era leer y entender la teoría que se presentaba en los artículos, consultando conceptos que no se entendían en internet. En segundo lugar, en todos ellos siempre aparecía un pseudocódigo que describía formalmente todo el proceso del algoritmo y que luego se pasaba a la implementación. Exceptuando Self-Training, no era posible entender su funcionamiento sin entender la teoría y la dificultad se encontraba en que no se había trabajado con este tipo de notaciones particulares y era muy fácil perderse.

Entrando más en cómo se desarrollaban, la idea siempre era crear un objeto con ese algoritmo. De hecho, al principio del proyecto (con Self-Training) se realizaba como una función única. Sin embargo, la idea de la orientación a objetos permitía un manejo mucho más fino para crear funciones separadas (entrenamiento, estadísticas, predicciones...) pero del mismo contexto y facilitaban su uso en la parte de la aplicación Web (Flask).

Por lo general, se creaban unas versiones básicas en las que simplemente se quería tener una implementación funcional. En esas primeras fases no se consideraban «refactorizaciones» en cuanto a estructuras de datos más óptimas o exceso de complejidad, sino que era una toma de contacto para ajustar el pensamiento a cada uno de ellos en particular. Para probar estos algoritmos de forma rudimentaria, se utilizaban los conjuntos de datos de «juguete» de `Scikit-Learn`.

Cuando se pensaba que la implementación correspondía (en principio) a los artículos, se comenzaban esas tareas de depuración. Por ejemplo, tanto en Democratic Co-Learning como en Tri-Training se detectó que los tiempos de entrenamiento eran demasiado grandes. En el primer caso, resultó ser porque la condición de parada estaba mal planteada y ejecutaba muchas más iteraciones de las necesarias. En el segundo caso, resultó ser precisamente el uso de estructuras de datos más lentas (se subsanó utilizando `numpy`).

Validación

Aparte de intentar depurar el código generado, no era suficiente con la intuición de que estos algoritmos eran correctos. Para comprobarlo, se ha comparado con la biblioteca `sslearn` [15]. Esta biblioteca cubre todos los algoritmos desarrollados en este proyecto y que además está mantenida periódicamente.

Antes de presentar los resultados de esta validación, se definen las estadísticas que se han utilizado para comprobar que la implementación propia es comparable con `sslearn` y concluir que es correcta.

- ***Accuracy***: Esta métrica será el núcleo principal de la comparación. Representa la proporción de aciertos entre todos los datos.
- ***F1 score***: Esta métrica se calcula como la media armónica de la precisión[7] y `recall` (es igual que la sensibilidad).

$$F_1 = 2\frac{precisión \cdot recall}{precisión + recall}$$

- ***Geometric mean (Gmean)***: En este contexto, la media geométrica trata maximizar cada una de las medidas de *accuracy* de cada clase [22].

  Para realizar su cálculo, se tiene en cuenta si se trata de clasificación binaria o multiclase:

  - Binaria: la media geométrica se calcula como la raíz cuadrada del producto de la sensibilidad[8] y la especificidad[9].

$$Gmean = \sqrt{sensibilidad \cdot especificidad}$$

  - Multiclase: la media geométrica se calcula como la raíz n-ésima del producto de las sensibilidades de cada clase.

$$Gmean = \sqrt[n]{sensibilidad_1 \cdot sensibilidad_2 \ldots \ldots \cdot sensibilidad_n}$$

  El mejor valor es 1 y el peor 0.

[7] Precisión: proporción de clasificados positivos (verdaderos positivos + falsos positivos) que son clasificados positivos (verdaderos positivos)

[8] Sensibilidad: proporción de positivos correctamente identificados (clasificados como positivos respecto a todos los positivos)[32]

[9] Especificidad: proporción de negativos correctamente identificados (clasificados como negativos respecto a todos los negativos)[32]

La comparación consistirá en una validación cruzada con 10 *folds* con tres conjuntos de datos diferentes. Por cada iteración de la validación cruzada se tendrá el 90 % de datos para entrenar y 10 % de test.

Sobre ese 90 % de entrenamiento, solo el 20 % va a estar etiquetados, el resto serán no etiquetados. En conjunto esos serán los datos para entrenar el algoritmo.

Por cada uno de los conjuntos de datos se presenta un gráfico comparando cada algoritmo propio con el de `sslearn` mediante gráficos de caja. En el eje X se presentan los algoritmos que se están comparando y en el eje Y cada una de las estadísticas antes comentadas.

Contenido de cada gráfico de cajas:

- **Línea continua azul**: representa la media.
- **Línea discontinua negra**: representa la mediana (segundo cuartil).
- **Borde superior de la caja coloreada**: representa el tercer cuartil.
- **Borde inferior de la caja coloreada**: representa el primer cuartil.
- **Límite del bigote superior**: representa el máximo de los valores encontrados.
- **Límite del bigote inferior**: representa el mínimo de los valores encontrados.

En la tabla 5.1 se pueden ver los clasificadores base seleccionados para cada algoritmo. La selección es completamente arbitraria (con base en los clasificadores que se han usado personalmente en el proyecto y fuera de él).

Self-Training	**Co-Training**	**Democratic Co-Learning**	**Tri-Training**
	Árbol de decisión	Vectores de Soporte C (SVC)	Árbol de decisión
Naive Bayes Gaussiano	Naive Bayes Gaussiano	Naive Bayes Gaussiano	kNN
		Árbol de decisión	Naive Bayes Gaussiano

Tabla 5.1: Clasificadores base

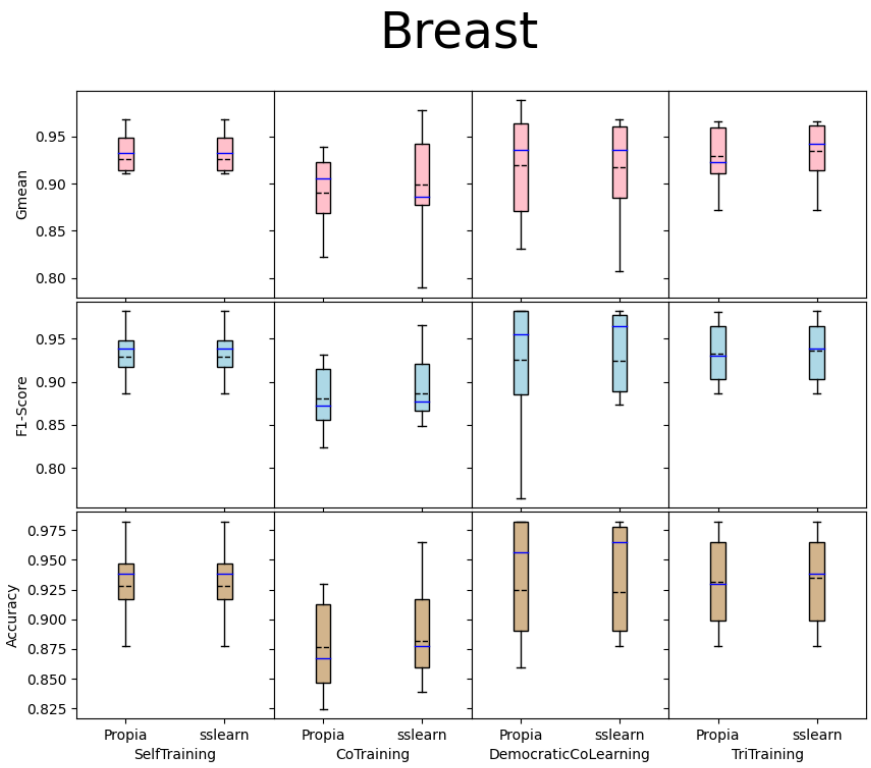

Figura 5.1: Comparación de métodos con el conjunto de datos *Breast Cancer*.

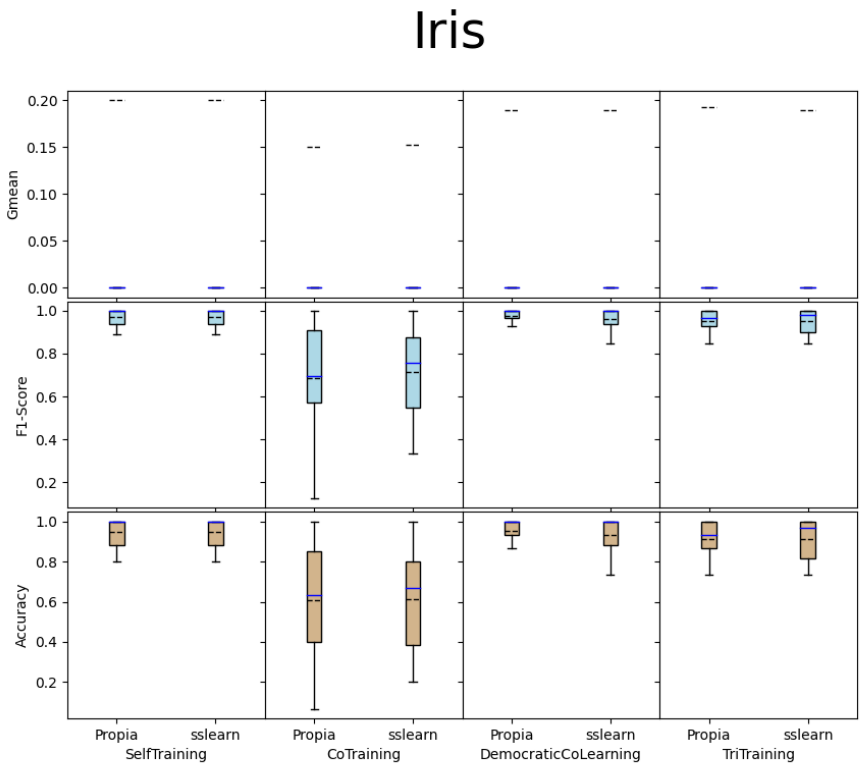

Figura 5.2: Comparación de métodos con el conjunto de datos *Iris*.

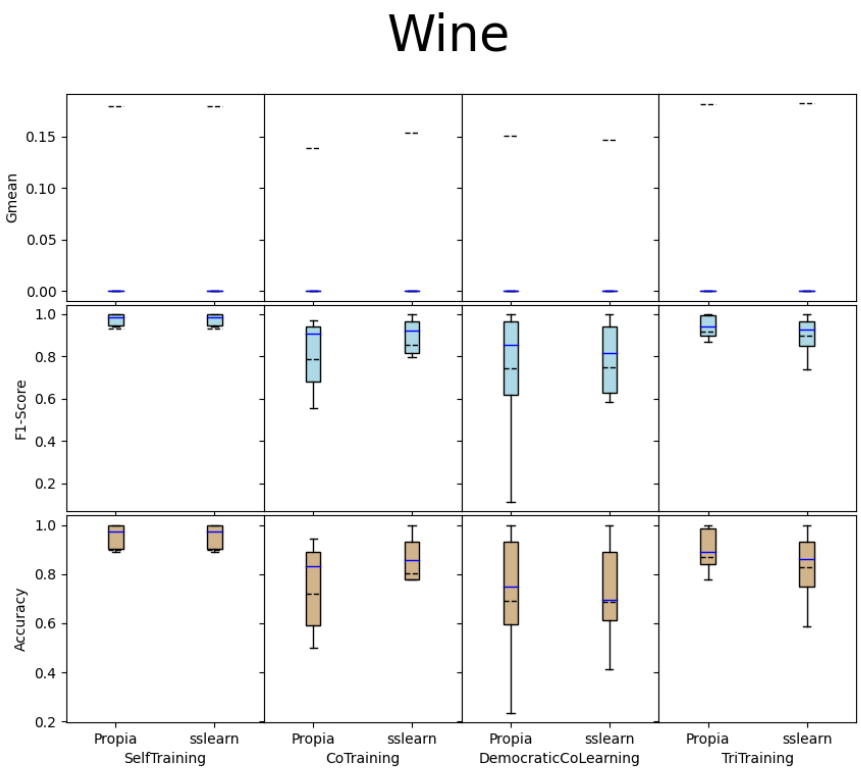

Figura 5.3: Comparación de métodos con el conjunto de datos *Wine*.

Por cada una de las figuras anteriores, lo interesante para comprobar si las implementaciones pueden considerarse como correctas es fijarse en `sslearn`[10].

Como se puede ver, en todos los casos los valores de la implementación propia coinciden en gran medida con las de `sslearn`. Es de destacar que en los casos de Self-Training, Democratic Co-Learning y Tri-Training se comportan de manera muy similar. En el caso de Co-Training en todas hay más variabilidad. Esto se debe a las diferencias de implementación.

Aunque haya pequeñas variaciones, los valores obtenidos son muy parecidos. Se concluye por tanto, que los algoritmos implementados se consideran correctos.

[10]Biblioteca de algoritmos semi-supervisados creada por José Luis Garrido-Labrador [15]

5.5. Desarrollo web

Pese a que los algoritmos han sido una buena parte de todo el proyecto, el desarrollo de la aplicación web es el que ha ocupado la mayor parte del tiempo (de forma aproximada, se calcula que un 70 %). El por qué esto ha sido así reside en dos cuestiones.

Por un lado, el tamaño de la aplicación, que pese a no ser una aplicación extremadamente grande, todo se ha realizado desde cero. No se adquirió ninguna plantilla o estructura existente y la única ayuda utilizada fue `Bootstrap` (que sí agiliza el estilado de la web). Y en segundo lugar, por el desconocimiento del desarrollo web. Durante los estudios no se han dado asignaturas de esta temática sea el desarrollo web. No ha sido hasta este mismo curso cuando se nos ha introducido el mundo web (Diseño y Mantenimiento del Software).

Durante todo el desarrollo, y hasta el final, se tomaron decisiones constantemente. El *framework* Flask fue el que se manipuló en esa asignatura, era interesante seguir en esa línea y desarrollar el proyecto con él.

De hecho, al comenzar el proyecto, se tomó las estructura básica del proyecto final de la asignatura dejando solo lo básico para arrancar una aplicación Web.

JavaScript

La toma de contacto con JavaScript fue compleja, no se había trabajado con este lenguaje y solo se tenían conceptos básicos de HTML y CSS (junto con *backend*, que al ser Python, ya se tenía cierta soltura). Para intentar avanzar lo máximo posible, se optó por ver tutoriales y cursos[11] al mismo tiempo que ya se iniciaban las visualizaciones de los algoritmos (completamente rudimentarias). En este sentido, se iba a utilizar la librería `D3.js` que, comparándola con otras, era algo más difícil de manejar con soltura.

Esta fue la línea general de desarrollo en JavaScript, en el momento que se presentaba un problema o el uso de una nueva librería, ir a fondo con ello directamente sobre la Web, hasta conseguir avances. Todo ello guiado por foros o páginas como *Stack Overflow*. Además, se optó por desarrollar la Web en JavaScript «Vanilla», sin librerías como jQuery. Se pensó que aprender las bases de JavaScript era más provechoso.

[11] https://www.w3schools.com/

Es destacable la cantidad de veces que hubo que refactorizar el código. Durante el manejo y aprendizaje, aparecían mejores soluciones a lo que ya se tenía. Por ejemplo, el uso de *imports* permitía centralizar y reutilizar código (que al principio se desconocía).

Aparte de todo esto, exceptuando las visualizaciones (que son bastante particulares) y considerando todos los problemas que aparecían, el resto del código de JavaScript no era extremadamente complejo y la mayoría de las veces existían soluciones parciales que simplemente podían adaptarse.

El avance, aunque algunas veces lento, era constante.

Rediseño completo

Uno de los puntos más destacables del desarrollo fue durante los sprints 13 y 14. La aplicación tuvo un rediseño completo (tanto estructura como apariencia) partiendo desde el «backend» hasta el «frontend».

Tal y como estaba la aplicación estructurada, todas las rutas estaban en un único fichero, que a su vez era el mismo que instanciaba la aplicación. Esto, para demos o aplicaciones muy pequeñas, es una solución rápida y buena. Sin embargo, lo que se vio de otros proyectos es que cuando el tamaño es grande, es necesario compartimentar rutas, modelos de bases de datos u otros elementos que crecen en número. Tener absolutamente todo en un único fichero es engorroso.

Es por eso que se utilizó la idea de *Application Factory* y *Blueprints*. Por un lado, *Application Factory* ha permitido hacer un único punto de creación de la aplicación donde se configura y se desvincula del resto del código.

Los *Blueprints* son como pequeñas aplicaciones independientes de Flask (que luego son unificadas en una, en esa *Application Factory* comentada). En cada *Blueprint* se han definido las rutas que intervienen en cada contexto. Por ejemplo, para la configuración de los algoritmos, se han definido sus rutas exclusivas, como si fuera una aplicación por sí sola. Esto permite aplicaciones **mantenibles y escalables**.

Tal y como está al final, con la cantidad de rutas y utilidades codificadas, hubiera sido imposible mantener un único fichero para toda la aplicación.

Babel

Un aspecto a destacar de la aplicación Web del proyecto es la internacionalización. En principio, dado que no era necesario abarcar un abanico

grande de idiomas, la aplicación está pensada para Inglés y Español, con la ventaja de que `Babel` hace muy fácil el manejo de las traducciones y es perfectamente escalable. Desde las primeras fases de desarrollo se incluyó esta forma de mantener unas traducciones de forma automática, y junto con el mantenimiento de las traducciones actualizadas cada poco tiempo, la tarea no ha resultado difícil, aunque sí laboriosa.

Trabajos relacionados

Con el aprendizaje automático y el uso de gran cantidad de datos, es completamente necesaria la visualización de los datos, procesos de entrenamiento y ciertas estadísticas. Al fin y al cabo, cuando se construye un modelo, se debe tener una realimentación de cómo de bien está funcionando para poder extraer conclusiones sobre el mismo.

De forma general, sin centrarse directamente en el aprendizaje automático, resulta interesante y conveniente que los visualizadores de algoritmos sean accesibles y que, como están apareciendo, se creen aplicaciones Web que resultan mucho más directas. Desde el punto de vista de la docencia y aprendizaje los visualizadores permiten culminar la comprensión los aspectos teóricos subyacentes.

6.1. Visualizadores

Seshat

Es una herramienta web que trata de facilitar el aprendizaje de la teoría de lenguajes y autómatas (análisis léxico) que utilizan los compiladores [6], puede accederse desde `http://cgosorio.es/Seshat`. La herramienta propone en primera instancia unas explicaciones teóricas de los algoritmos con sus conceptos generales, el concepto concreto de las expresiones regulares y qué es un autómata finito. A partir de la teoría, se encuentran implementados varios algoritmos que se visualizan paso a paso. En el momento de la ejecución también se tienen elementos de interés como la propia descripción o explicación del algoritmo. Los algoritmos implementados por la herramienta son:

1. Construcción de un autómata finito no determinista (AFND) a partir de una expresión regular.

2. Conversión de un autómata finito no determinista (AFND) a un autómata finito determinista (AFD).

3. Construcción de un autómata finito determinista (AFD) a partir de una expresión regular.

4. Minimización de un autómata finito.

Para su construcción se ha usado el *framework* Flask en Python que actúa como servidor. La interfaz de usuario está construida con HTML, SVGs y Javascript para proporcionar el contenido dinámico.

En este proyecto se realizó un estudio de la opinión de 42 estudiantes después de su uso. El estudio consistía en unas afirmaciones (5) respecto a la buena utilidad, buen diseño o el interés que puede producir al visualizar los algoritmos de esa forma. De forma general, en una escala del 1 al 5 (donde el 5 representa que están profundamente de acuerdo con la afirmación) el 95 % de los resultados se concentran entre el 4 y el 5 en la valoración. Esto indicó que la herramienta sí que resultó útil e incluso ellos mismos solicitaban este recurso para facilitar su aprendizaje.

Herramienta de apoyo a la docencia de algoritmos de selección de instancias

En [5] se presenta una herramienta de escritorio para la visualización de la ejecución y resultados de los algoritmos de selección de instancias, debido a la carencia de este tipo de aplicaciones para dichos algoritmos. La herramienta es altamente personalizable pudiendo subir el conjunto de datos o seleccionar qué característica visualizar en los ejes. En la visualización se pueden ver todos los pasos de los algoritmos junto, por ejemplo, a las regiones de Voronoi o el pseudocódigo. Esto hace que el alumno pueda conocer el progreso y los conceptos particulares (influencia, vecindad...). Los algoritmos implementados por la herramienta son:

1. Algoritmo Condensado de Hart (CNN) [17].

2. Algoritmo Condensado Reducido (RNN) [16].

3. Algoritmo Subconjunto Selectivo Modificado (MSS) [7].

4. Algoritmos Decremental Reduction Optimization Preocedure (DROP) [37].

5. Algoritmo Iterative Case Filtering (ICF) [9].

6. Algoritmo Democratic Instance Selection (DIS) [14].

Está desarrollado completamente en Java lo que lo hace portable a cualquier plataforma y sin instalación.

Como punto característico a tener en cuenta, la herramienta estaba muy orientada a ofrecer bastante granularidad en cada paso, de tal forma que mediante, por ejemplo, la visualización del pseudocódigo (al mismo tiempo que la visualización gráfica), permite para analizar el comportamiento subyacente.

Los estudiantes, de forma general, valoraron muy positivamente la herramienta pues les ayudó a comprender los algoritmos.

Towards Developing an Effective Algorithm Visualization Tool for Online Learning

En [25] se presenta una «guía» especialmente interesante con las consideraciones a tener en cuenta para el desarrollo de una correcta herramienta de visualización de algoritmos.

La realidad actual es que resulta muy difícil hacer que los alumnos entiendan todos los conceptos que involucran los algoritmos y más aún en la modalidad «a distancia». Para solventar este problema se han desarrollado múltiples herramientas de visualización de algoritmos (AV) que pretenden facilitar la compresión de los mismos, pero la realidad de estas herramientas es que se desarrollan con una fuerte componente «elegante» y no enriquecedora o pedagógica. Este artículo trata de abordar este último problema, el cómo construir herramientas de visualización de algoritmos que sean divertidas y atractivas, pero siendo fieles a los conceptos, y también cómo estas pueden ayudar a los docentes a asegurar el aprendizaje de sus alumnos.

Para lograr el objetivo de desarrollar un visualizador efectivo, se analiza desde el punto de vista de la pedagogía, usabilidad y accesibilidad. En la Tabla 6.1 se pueden ver algunas de las características que se deben tener en cuenta a la hora de desarrollar una herramienta de este estilo.

Característica	Cómo lograrlo
Incrementar la comprensión	Explicaciones textuales Realimentación con las acciones del usuario Ayuda integrada Comodidad en la entrada de datos
Promocionar interés	Permitir introducir datos al algoritmo Manipular la visualización Capacidad de volver «rebobinar» Variación de la velocidad Avanzar en la ejecución
Facilidad de aprendizaje	Controles familiares Generalidad con otros sistemas Predictibilidad de las acciones Interfaz simple
Facilidad de memorización	Interfaz común que soporte múltiples animaciones
Facilidad de uso	Interfaz común que soporte múltiples animaciones Claridad de los modelos y conceptos
Robustez	Explicaciones de la visualización Integración de ajustes predefinidos Recuperación de errores
Eficacia	Visibilidad del estado del sistema Control y libertad del usuario Prevención de errores Flexibilidad de uso Ayuda al usuario para reconocer, diagnosticar y recuperarse de errores
Accesibilidad	Globalmente disponible Plataforma y dispositivo independiente

Tabla 6.1: Características que influyen en los objetivos pedagógicos.

6.2. Otras herramientas/bibliotecas

Algorithm-Visualizer, https://algorithm-visualizer.org/. Se trata de una Web que incluye una cantidad enorme de algoritmos, todos están programados directamente en Javascript que genera una visualización de los mismos y también incluye una explicación corta de cada uno. Algunos de los algoritmos son: `Backtracking`, `Divide and Conquer`, `Greedy` o más concretamente `K-Means Clustering`, entre muchos otros.

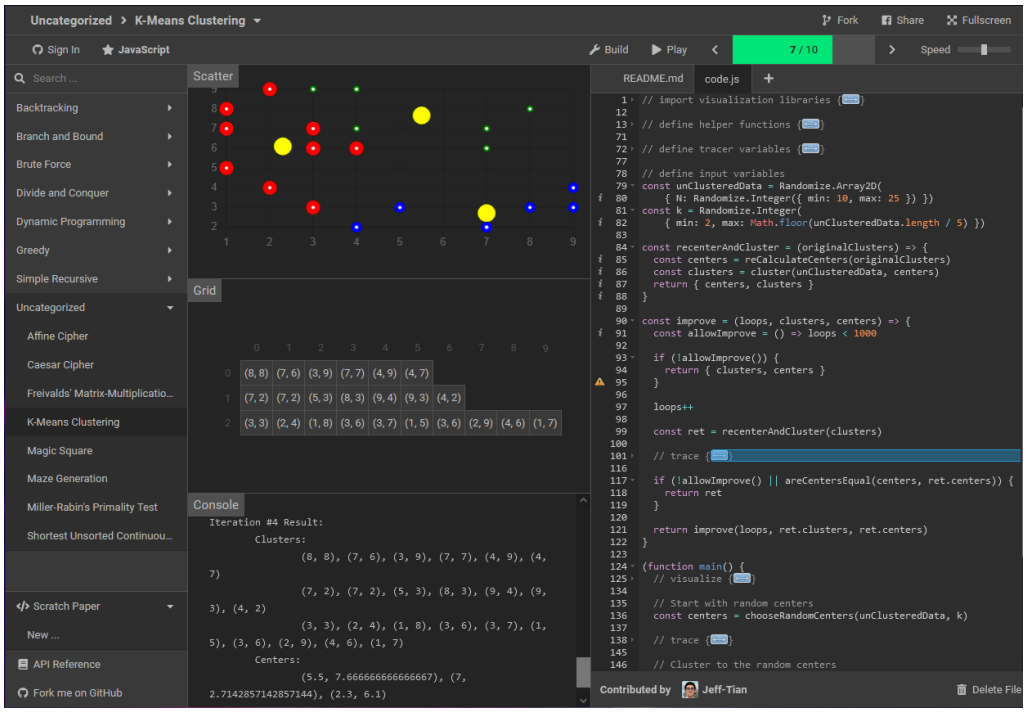

Figura 6.1: K-Means Clustering en algorithm-visualizer.

VISUALGO, https://visualgo.net/en. Es una página web creada por la Universidad Nacional de Singapur. Esta Web permite visualizar algoritmos básicos (como de ordenación o camino más corto) pero también estructuras de datos (como *hash tables* o conjuntos disjuntos).

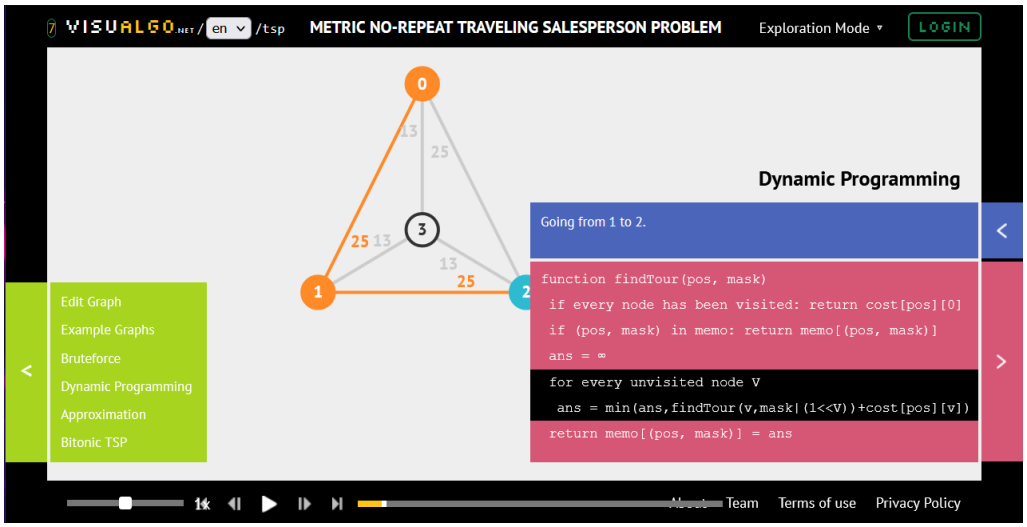

Figura 6.2: Viajante de comercio en VISUALGO.

Visualizer, por Zhenbang Feng, https://jasonfenggit.github.io/Visualizer/ [12]. Su autor lo define como una página web para proporcionar visualizaciones intuitivas e innovadoras de algoritmos generales y de inteligencia artificial. Las visualizaciones son muy atractivas, con las que se capta muy bien el funcionamiento de los algoritmos. Implementa algoritmos de búsqueda de caminos, ordenamiento e inteligencia artificial (como minimax).

[12]Github: https://github.com/JasonFengGit/Visualizer

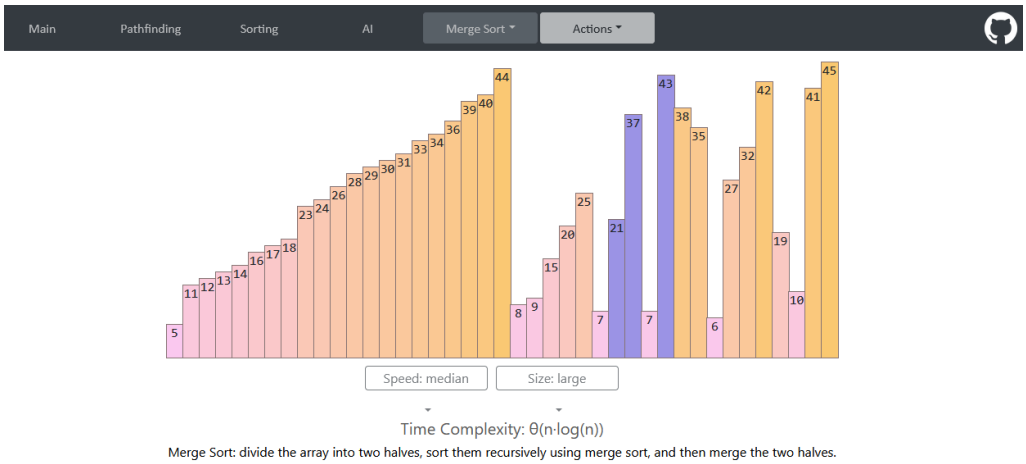

Figura 6.3: Mergesort en Visualizer (Zhenbang Feng).

Clutering-Visualizer, https://clustering-visualizer.web.app/. Es una página Web para visualizar algoritmos de clustering. Realmente, no tiene muchos implementados, pero las visualizaciones son muy descriptivas. En cada paso muestra cuál es la evolución/posición de los distintos *clusters* y mediante línea es posible ver las distancias en ellos y los datos.

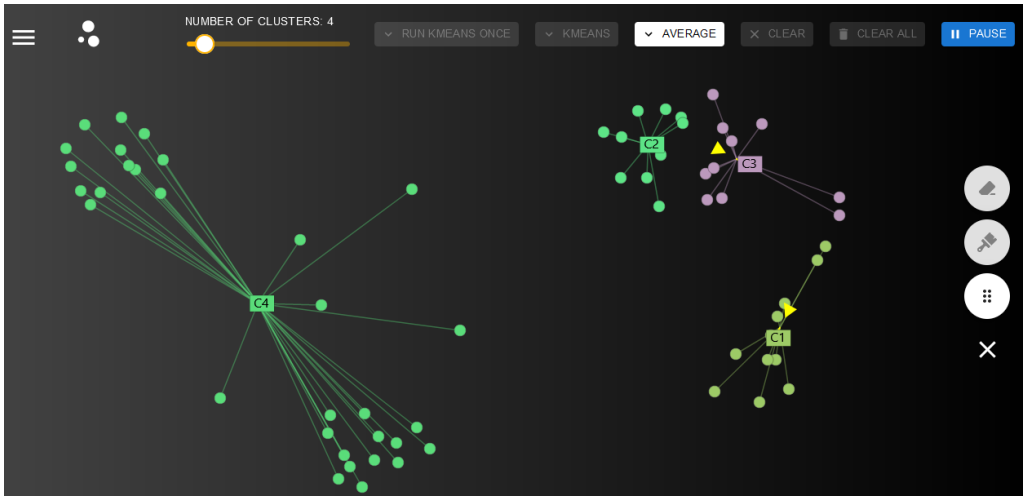

Figura 6.4: K-Means en Clutering-Visualizer.

MLDemos, puede visitarse desde https://basilio.dev/. Es una herramienta de visualización de código abierto para algoritmos de aprendizaje automático creada específicamente para ayudar en el estudio y compresión del funcionamiento de una gran cantidad de algoritmos. Entre ellos: clasificación, regresión, reducción de dimensionalidad y clustering.

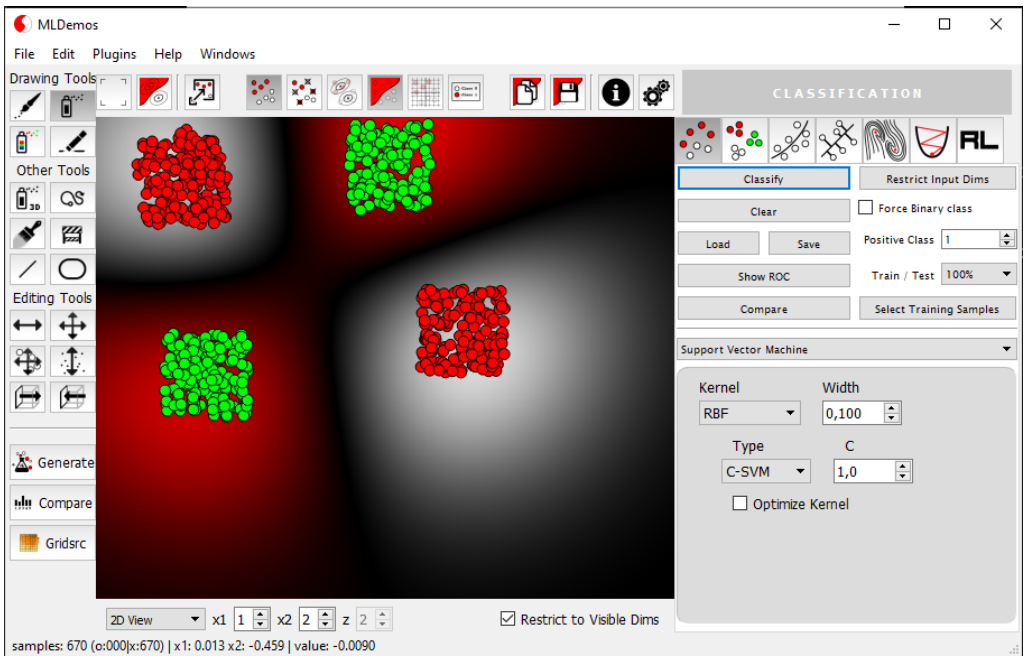

Figura 6.5: Clasificación mediante Support Vector Machine en MLDemos.

Conclusiones y Líneas de trabajo futuras

Todo proyecto debe incluir las conclusiones que se derivan de su desarrollo. Éstas pueden ser de diferente índole, dependiendo de la tipología del proyecto, pero normalmente van a estar presentes un conjunto de conclusiones relacionadas con los resultados del proyecto y un conjunto de conclusiones técnicas. Además, resulta muy útil realizar un informe crítico indicando cómo se puede mejorar el proyecto, o cómo se puede continuar trabajando en la línea del proyecto realizado.

Bibliografía

[1] Cross site request forgery. [Internet; accedido 8-May-2023].

[2] Http. [Internet; accedido 8-May-2023].

[3] Why would you use post instead of get for a read operation? [Internet; accedido 8-May-2023].

[4] 40defiebre. ¿qué es el diseño responsive? [Online; accessed 28-March-2023].

[5] Álvar Arnáiz González, José Francisco Díez Pastor, César I García Osorio, and Juan José Rodríguez Díez. Herramienta de apoyo a la docencia de algoritmos de selección de instancias. 2012.

[6] Álvar Arnaiz-González, Jose-Francisco Díez-Pastor, Ismael Ramos-Pérez, and César García-Osorio. Seshat—a web-based educational resource for teaching the most common algorithms of lexical analysis. *Computer Applications in Engineering Education*, 26(6):2255–2265, 2018.

[7] Ricardo Barandela, Francesc J. Ferri, and José Salvador Sánchez. Decision boundary preserving prototype selection for nearest neighbor classification. *IJPRAI*, 19(6):787–806, 2005.

[8] Avrim Blum and Tom Mitchell. Combining labeled and unlabeled data with co-training. pages 92–100, 1998.

[9] Henry Brighton and Chris Mellish. Advances in instance selection for instance-based learning algorithms. *Data Mining and Knowledge Discovery*, 6(2):153–172, 2002.

[10] MDN contributors. Http. [Internet; accedido 8-May-2023].

[11] Salim Dridi. Supervised learning - a systematic literature review. *ResearchGate*, 09 2021.

[12] Salim Dridi. Unsupervised learning - a systematic literature review. *ResearchGate*, 12 2021.

[13] Jasmine Finer. How to write beautiful python code with pep 8. [Internet; accedido 15-May-2023].

[14] César García-Osorio, Aida de Haro-García, and Nicolás García-Pedrajas. Democratic instance selection: A linear complexity instance selection algorithm based on classifier ensemble concepts. *Artificial Intelligence*, 174(5-6):410–441, 2010.

[15] José Luis Garrido-Labrador. jlgarridol/sslearn: v1.0.3.1, March 2023.

[16] G. Gates. The reduced nearest neighbor rule (corresp.). *Information Theory, IEEE Transactions on*, 18(3):431–433, May 1972.

[17] P. Hart. The condensed nearest neighbor rule (corresp.). *Information Theory, IEEE Transactions on*, 14(3):515 – 516, may 1968.

[18] Intelligent. Machine learning: qué es y cómo funciona, 2020. [Internet; descargado 27-octubre-2022].

[19] javaTpoint. Unsupervised machine learning. [Online; accessed 15-November-2022].

[20] lokalise. Internationalization vs. localization (i18n vs l10n): What's the difference? [Online; accessed 2-May-2023].

[21] Lukas Huber. A friendly intro to semi-supervised learning. [Online; accessed 15-November-2022].

[22] Metrics. The imbalanced-learn developers. [Internet; accedido 15-May-2023].

[23] David Petersson. Supervised learning, 2021. [Internet; descargado 15-noviembre-2022].

[24] Markus Ringnér. What is principal component analysis? *Nature biotechnology*, 26(3):303–304, 2008.

[25] Katarzyna Romanowska, Gurpreet Singh, M. Ali Akber Dewan, and Fuhua Lin. Towards developing an effective algorithm visualization tool for online learning. pages 2011–2016, 2018.

[26] Scrum.org. The scrum team. https://www.scrum.org/learning-series/what-is-scrum/the-scrum-team. [Internet; accedido 19-May-2023].

[27] Neova Tech Solutions. Machine learning algorithms: Beginners guide part 1, 2018. [Internet; descargado 27-octubre-2022].

[28] Pascual Parada Torralba. ¿qué es el machine learning? aprendizaje supervisado vs no supervisado, 2022. [Internet; descargado 27-octubre-2022].

[29] TutorialCampus. Scrum. [Internet; accedido 19-May-2023].

[30] Jesper E. van Engelen and Holger H. Hoos. A survey on semi-supervised learning. *Machine Learning*, 109(2):373–440, Feb 2020.

[31] Guido van Rossum y Barry Warsaw y Nick Coghlan. Pep 8 – style guide for python code. [Internet; accedido 15-May-2023].

[32] Wikipedia. Sensibilidad y especificidad — wikipedia, la enciclopedia libre, 2022. [Internet; descargado 16-agosto-2022].

[33] Wikipedia. Capa de aplicación — wikipedia, la enciclopedia libre, 2023. [Internet; descargado 19-febrero-2023].

[34] Wikipedia contributors. Training, validation, and test data sets — Wikipedia, the free encyclopedia, 2022. [Online; accessed 15-November-2022].

[35] Wikipedia contributors. Data pre-processing — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Data_pre-processing&oldid=1138293751, 2023. [Online; accessed 16-May-2023].

[36] Wikipedia contributors. List of http status codes — Wikipedia, the free encyclopedia, 2023. [Internet; accedido 8-May-2023].

[37] D. Randall Wilson and Tony R. Martinez. Reduction techniques for instance-based learning algorithms. *Machine Learning*, 38(3):257–286, 2000.

[38] Yan Zhou and Sally Goldman. Democratic co-learning. pages 594–602, 2004.

[39] Zhi-Hua Zhou and Ming Li. Tri-training: exploiting unlabeled data using three classifiers. *IEEE Transactions on Knowledge and Data Engineering*, 17(11):1529–1541, 2005.