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 — 22 de marzo de 2023
Tutor: Álvar Arnaiz González
Cotutor: César Ignacio García Osorio

UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

D. Álvar Arnaiz González, profesor del departamento de Ingeniería Informática, área de Lenguajes y Sistemas informáticos, junto a D. César Ignacio García Osorio, profesor 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, 22 de marzo de 2023

Vº. Bº. del Tutor:	Vº. Bº. del co-tutor:
Álvar Arnaiz González	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 es un campo muy interesante y del que cada vez hay más atención. La realidad es que la mayor parte del conocimiento está muy centrado en ciertos tipos de aprendizaje automático: el supervisado y el no supervisado. En cuanto se indaga un poco en «machine learning» estos dos conceptos empiezan a rondar. 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, dada 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 de los que no se sabe qué representan, el aprendizaje semi-supervisado es el ideal en estos casos y es que permite inferir conocimiento para estos últimos y determinar a qué corresponden.

Centrando más el objetivo final de este trabajo, no existen muchas aplicaciones que permitan compaginar la teoría de estos conceptos con visualizaciones interesantes que permitan 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 resultar en la creación de una aplicación amigable y atractiva que permita, mediante visualizaciones, comprender realmente cómo funcionan los principales algoritmos semi-supervisados.

Las herramientas de fácil acceso, como las páginas Web que no requieren de instalación por parte del usuario, van de la mano con 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 prácticamente tener que realizar otro trabajo.

Además, los datos de las visualizaciones será proporcionados por una biblioteca propia donde se implementen estos algoritmos. Estarán adaptados para la obtención de la información del entrenamiento y estadísticas relevantes, pero su caso están pensados para que puedan ser utilizados de forma general.

Objetivos del proyecto

Este apartado explica de forma precisa y concisa cuales son los objetivos que se persiguen con la realización del proyecto. Se puede distinguir entre los objetivos marcados por los requisitos del software a construir y los objetivos de carácter técnico que plantea a la hora de llevar a la práctica el proyecto.

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 [11], 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), [17] 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) [18].

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

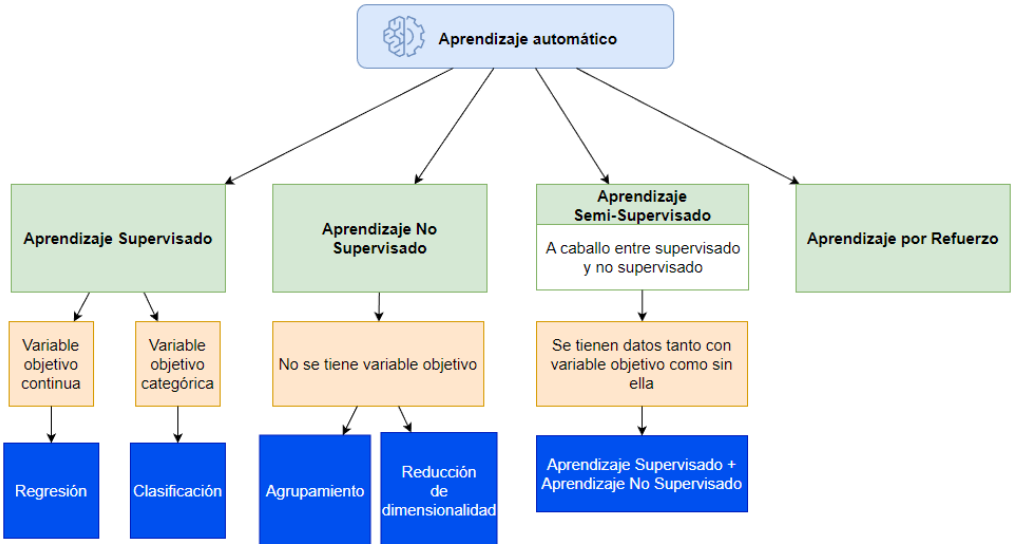

Figura 3.1: Clasificación de aprendizaje automático [16].

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 [14]. 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 [6].

Es común que antes del entrenamiento, estos datos son particionados en: conjunto de entrenamiento, conjunto de test o 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 por último, el conjunto de test sirve para estimar el rendimiento real que podrá tener el modelo en explotación [19]. En la figura 3.2 puede visualizarse el funcionamiento general.

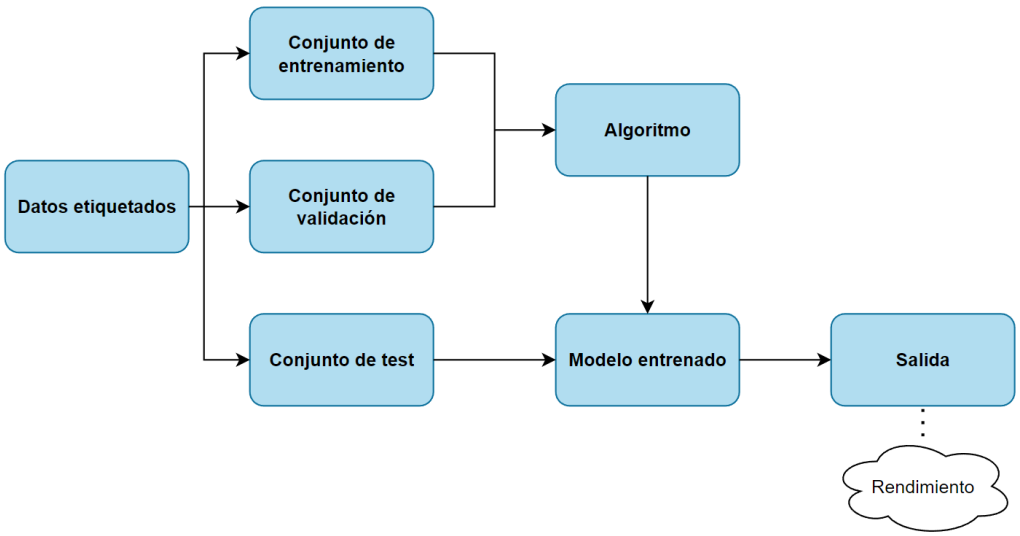

Figura 3.2: Funcionamiento general del aprendizaje supervisado [6].

El aprendizaje supervisado está altamente influenciado por esto. Por un lado, si las etiquetas son categóricas o binarias el modelo será de **clasificación** y por otro, si las etiquetas son continuas el modelo será de **regresión**.

- **Clasificación**: Los algoritmos de clasificación, 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 algoritmo 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 no se nutren de datos etiquetados. En otras palabras, los usuarios no «supervisan» el modelo [7]. 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 modelo 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 modelos pueden tanto descubrir por ellos mismos los «clústeres» o grupos que se encuentran o indicarle cuántos debe identificar [7].

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 [12] 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

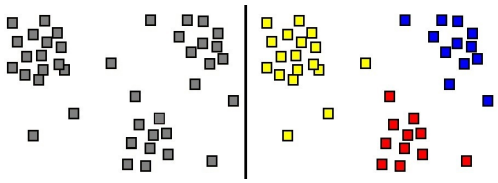

Figura 3.3: Clusters. Ejemplo de clustering, a la izquierda los datos no etiquetados y a la derecha los datos coloreados según las clases identificadas por el algoritmo de clustering. By hellisp - Own work, Public Domain, https://commons.wikimedia.org/w/index.php?curid=36929773.

Aprendizaje semi-supervisado

Según [18], el aprendizaje semi-supervisado es la rama del aprendizaje automático referido al uso de datos tanto etiquetados como no etiquetados simultáneamente para realizar tareas de aprendizaje. Se encuentra a caballo entre el aprendizaje supervisado y 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, el proceso de etiquetado es 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 aprendizaje supervisado, en el que 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 $p(y|x)$ [18].

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. [13, 18]

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» en el que en todas ellas se encuentra presente. Y en realidad, todas son versiones de *Cluster assumption* en la 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 [18].

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

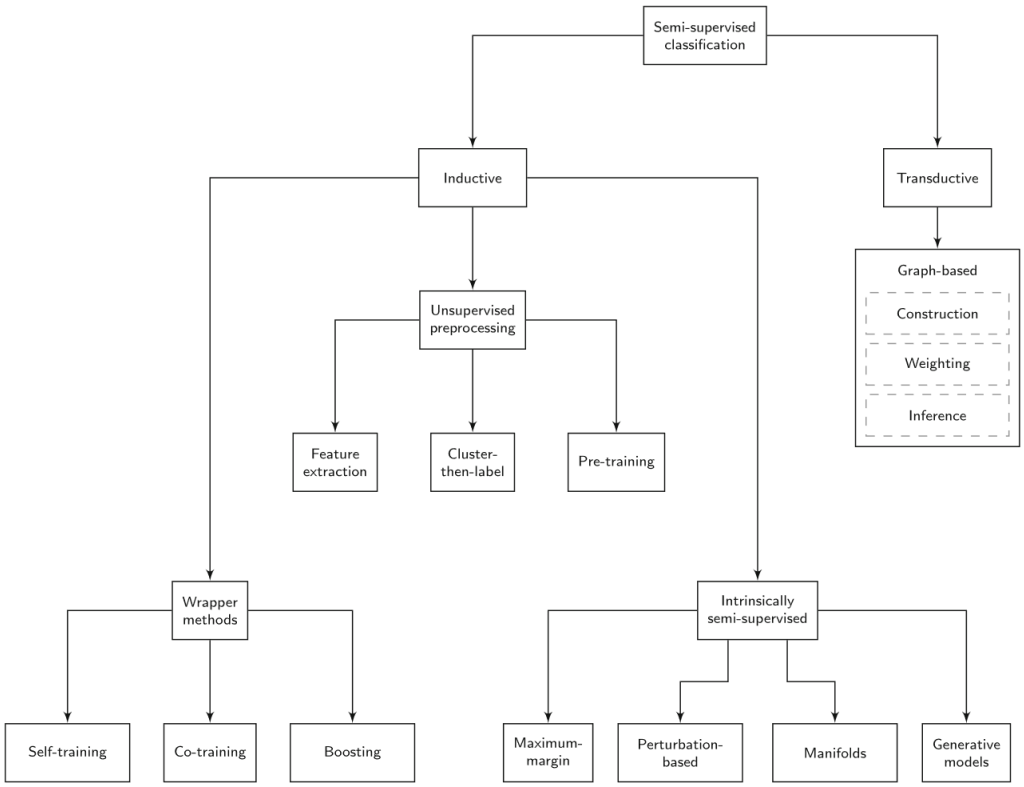

Figura 3.4: Taxonomía de métodos semi-supervisados [18].

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: 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 [18].

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 nuevos. El proceso continúa hasta 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 [18]. 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).

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 (mayor confianza/probabilidad).

En [4], 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 de subconjuntos de las características de las mismas. Es decir, cada clasificador va a entrenarse teniendo en cuenta características distintas. Idealmente estas vistas son independientes y pueden predecir por sí solas la etiqueta (aunque no siempre se cumplirá). Cuando los clasificadores predicen etiquetas sobre los datos, se seleccionan de ambos los de mayor confianza y construyen 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

```
Crear un subconjunto U' seleccionando u instancias aleatorias de U
for k iteraciones
    Entrenar H_1 con L solo considerando un subconjunto (x_1) de las características de cada instancia (x)
    Entrenar H_2 con L solo considerando el otro subconjunto (x_2) de las características de cada instancia (x)
    Hacer que H_1 prediga p instancias positivas y n negativas de U' que tengan la mayor confianza
    Hacer que H_2 prediga p instancias positivas y n negativas de U' que tengan la mayor confianza
    Añadir estas instancias seleccionadas a L
    Reponer U' añadiendo 2p + 2n instancias de U
endfor
return H_1, H_2
```

Democratic Co-Learning Yan Zhou y Sally Goldman presentaron en [21] un algoritmo de aprendizaje semi-supervisado en la línea de Co-Training (varios clasificadores). La diferencia sustancial es que 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 se alcanza cuando no se mejora la precisión pese a la adición de nuevas instancias pseudoetiquetadas.

Aclaración sobre la predicción final (en combinación): una vez calculadas las confianzas de la instancia a predecir (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 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 $H_i(x)$ *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

Técnicas y herramientas

En este apartado se presentarán las técnicas (procedimientos) 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. 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 automática (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 el proyecto tiene una fuerte componente de desarrollo Web así como frameworks, PyCharm tiene una integración completa con estos ámbitos, pudiendo desarrollar también de forma nativa con JavaScript o lenguajes de marcas (HTML o CSS).

Otra ventaja del uso de 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.

Por último, 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 ya por lo general rápido proceso de control de versiones.

Librerías Python Para el proyecto se están utilizando múltiples librerías que facilitan en gran medida el desarrollo del mismo. Implementan funcionalidades que pueden ser utilizadas directamente. En el proyecto, las librerías usadas están especificadas en los requisitos (*requirements*).

- **pandas**: Librería 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 aplicacones Flask (con la ayuda de Babel).
- **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 librería 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**: Librería que incluye algoritmos matemáticos.
- **setuptools**: Librería que permite crear e instalar paquetes de software de Python.
- **pytest**: Es un Framework de pruebas unitarias para Python.
- **sslearn**: Librería para aprendizaje automático en conjuntos de datos Semi-Supervisados como extensión de scikit-learn. Creada por José Luis Garrido-Labrador.
- **matplotlib**: Librería para la creación de gráficos en 2D.

Aspectos relevantes del desarrollo del proyecto

Este apartado pretende recoger los aspectos más interesantes del desarrollo del proyecto, comentados por los autores del mismo. Debe incluir desde la exposición del ciclo de vida utilizado, hasta los detalles de mayor relevancia de las fases de análisis, diseño e implementación. Se busca que no sea una mera operación de copiar y pegar diagramas y extractos del código fuente, sino que realmente se justifiquen los caminos de solución que se han tomado, especialmente aquellos que no sean triviales. Puede ser el lugar más adecuado para documentar los aspectos más interesantes del diseño y de la implementación, con un mayor hincapié en aspectos tales como el tipo de arquitectura elegido, los índices de las tablas de la base de datos, normalización y desnormalización, distribución en ficheros3, reglas de negocio dentro de las bases de datos (EDVHV GH GDWRV DFWLYDV), aspectos de desarrollo relacionados con el WWW... Este apartado, debe convertirse en el resumen de la experiencia práctica del proyecto, y por sí mismo justifica que la memoria se convierta en un documento útil, fuente de referencia para los autores, los tutores y futuros alumnos.

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 construyen modelos, 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 [2] , 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 [1] 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 Boronoi 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) [10].

2. Algoritmo Condensado Reducido (RNN) [9].

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

4. Algoritmos Decremental Reduction Optimization Preocedure (DROP) [20].

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

6. Algoritmo Democratic Instance Selection (DIS) [8].

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 [15] 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
Aprendibilidad	Controles familiares
	Generalidad con otros sistemas
	Predictibilidad de las acciones
	Interfaz simple
Memorabilidad	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.

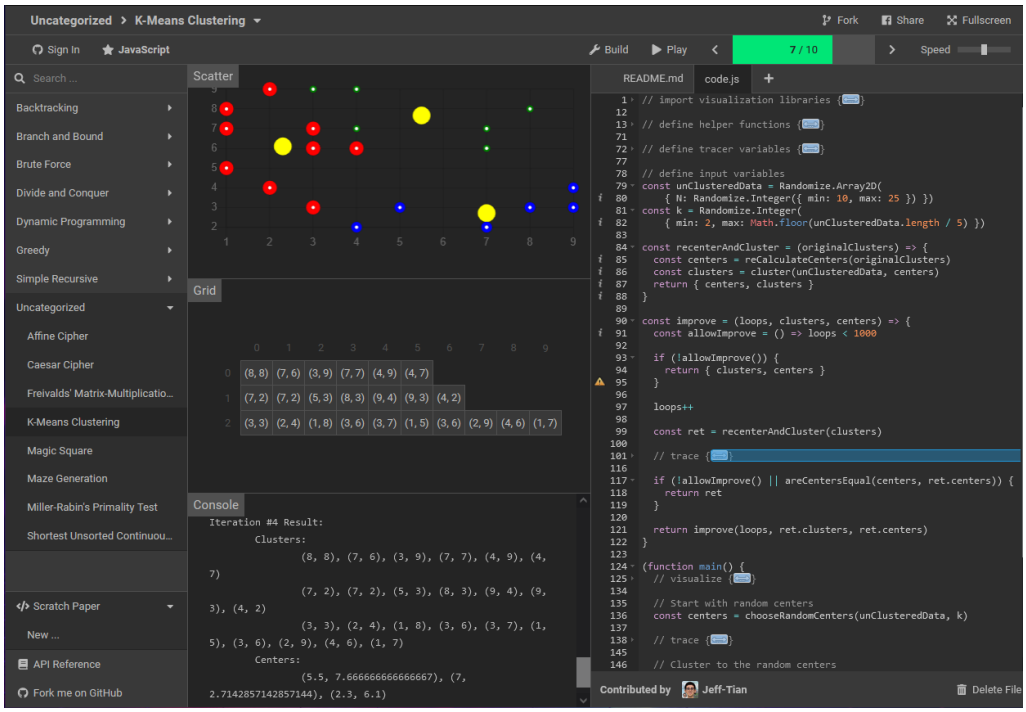

Figura 6.1: K-Means Clustering en algorithm-visualizer.

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).

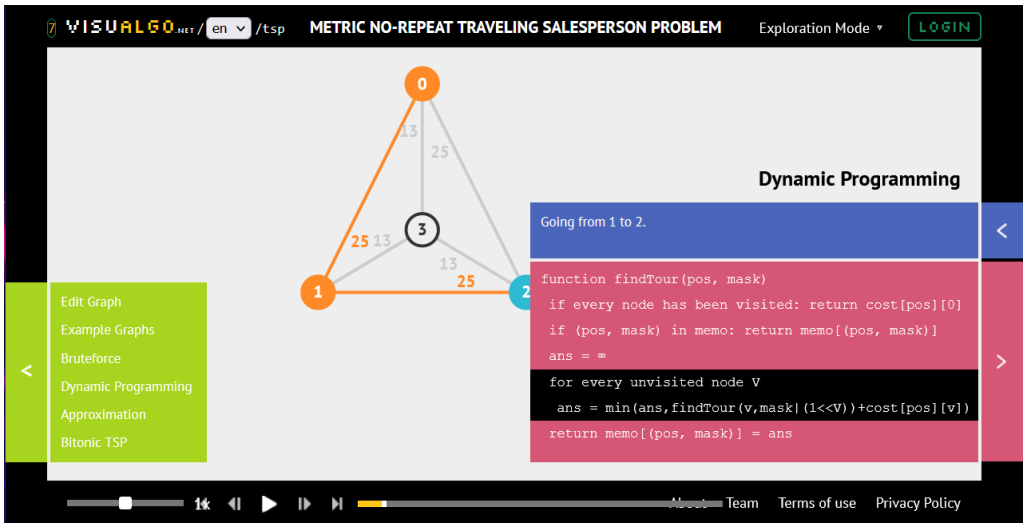

Figura 6.2: Viajante de comercio en VISUALGO.

Visualizer, por Zhenbang Feng, https://jasonfenggit.github.io/Visualizer/ [1]. 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 visuales con las que se capta muy bien el funcionamiento de los algoritmos. Implementa algoritmos de búsqueda de caminos, ordenamiento e inteligencia artificial (como minimax).

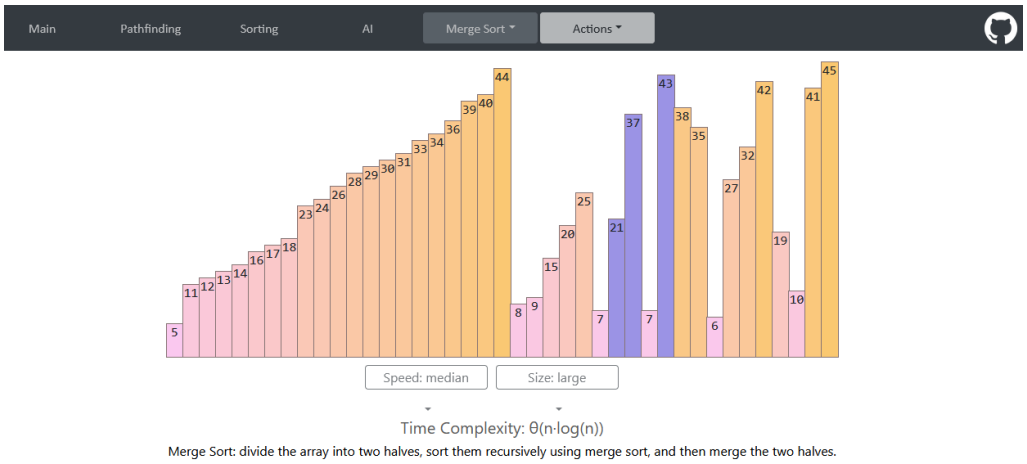

Figura 6.3: Mergesort en Visualizer (Zhenbang Feng).

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

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.

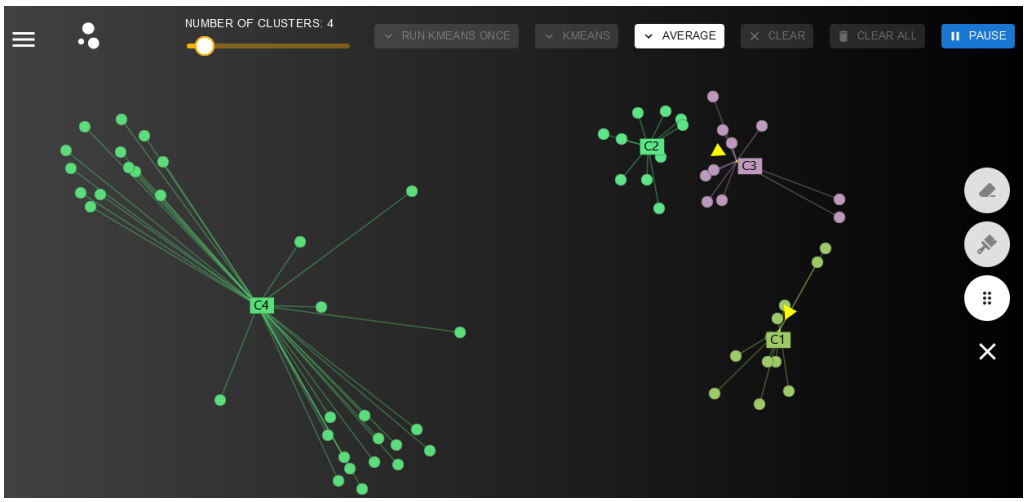

Figura 6.4: K-Means en Clutering-Visualizer.

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.

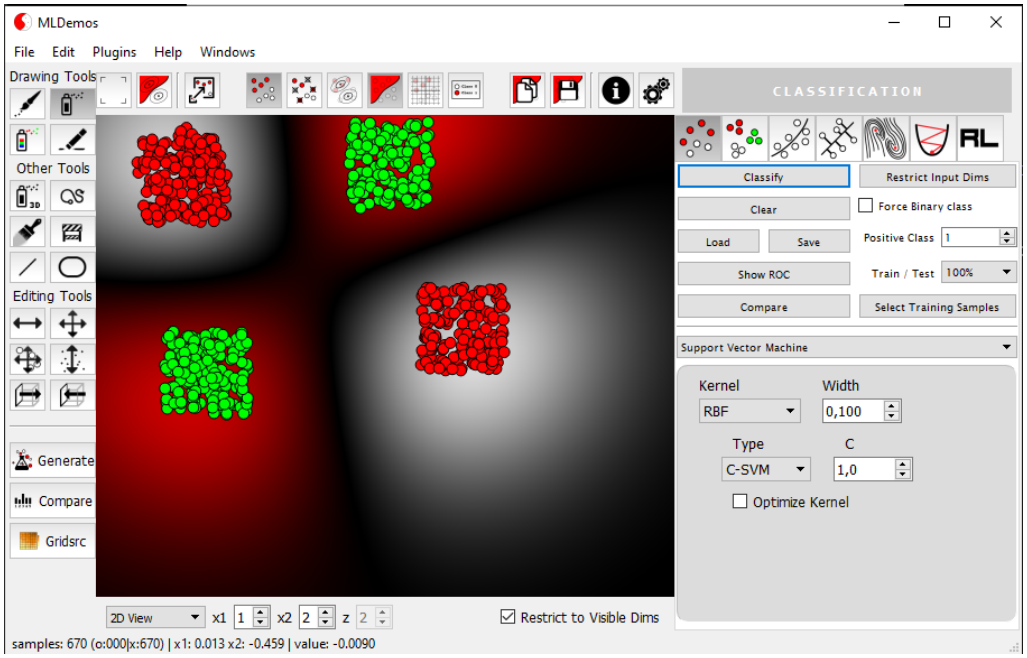

Figura 6.5: Clasificación mediante Support Vector Machine en MLDemos.

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] 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.

[2] 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.

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

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

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

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

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

[8] 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.

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

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

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

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

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

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

[15] 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.

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

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

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

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

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

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