



UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática



TFG del Grado en Ingeniería
Informática

Ampliación web para la
docencia de métodos de
aprendizaje semisupervisado



Presentado por Mario Sanz Pérez
en Universidad de Burgos — 13 de junio
de 2024

Tutor: Dr. Álgvar Arnaiz González



UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática



El Dr. Álgar Arnaiz González, profesor del departamento de Ingeniería Informática, área de Lenguajes y Sistemas informáticos.

Expone:

Que el alumno D. Mario Sanz Pérez, con DNI 71482918E, ha realizado el Trabajo final de Grado en Ingeniería Informática titulado «Ampliación web para la docencia de métodos de aprendizaje semisupervisado».

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, 13 de junio de 2024

Vº. Bº. del Tutor:

Dr. Álgar Arnaiz González

Resumen

En la práctica, el aprendizaje semi-supervisado es crucial debido a los retos y altos costos que implica obtener datos etiquetados de calidad. A pesar de su relevancia, estos algoritmos suelen ser ignorados, tanto en la investigación académica como en la educación, donde el énfasis generalmente recae en el aprendizaje supervisado o no supervisado.

En un proyecto anterior, se desarrolló una biblioteca con cuatro de los más populares algoritmos semi-supervisados: Self-Training, Co-Training, Democratic Co-Learning y Tri-Training, además de una aplicación web para visualizar el proceso de entrenamiento de estos algoritmos.

En este trabajo, se ha continuado la herramienta docente, añadiendo nuevos algoritmos para ampliar su alcance y funcionalidad. Se ha incorporado el algoritmo Co-Forest para métodos de *ensembles* y, para métodos transductivos basados en grafos, se han implementado los algoritmos GBILI y RGCLI en la fase de construcción de grafos, y el algoritmo LGC (*Local and Global Consistency*) en la fase de inferencia de etiquetas.

El objetivo principal sigue siendo el mismo: mostrar cómo van cambiando los datos no etiquetados en la fase de entrenamiento, facilitando su comprensión. La visualización de las diferentes fases de estos algoritmos se han realizado en la misma aplicación web utilizada anteriormente, aunque con modificaciones para adaptarse a los nuevos algoritmos incorporados.

Se parte de la aplicación web <https://vass.dmacha.dev> y se evoluciona a una segunda versión disponible en [URL](#)

Descriptores

aprendizaje automático, aprendizaje semisupervisado, ensembles, grafos, web

Abstract

In practice, semi-supervised learning is crucial due to the challenges and high costs involved in obtaining quality labeled data. Despite its importance, these algorithms are often overlooked, both in academic research and education, where the focus generally falls on supervised or unsupervised learning.

In a previous project, a library was developed with four of the most popular semi-supervised algorithms: Self-Training, Co-Training, Democratic Co-Learning, and Tri-Training, along with a web application to visualize the training process of these algorithms.

In this work, the educational tool has been continued, adding new algorithms to expand its scope and functionality. The Co-Forest algorithm for ensemble methods has been incorporated, and for graph-based transductive methods, the GBILI and RGCLI algorithms have been implemented in the graph construction phase, and the LGC (Local and Global Consistency) algorithm in the label inference phase.

The main objective remains the same: to show how the unlabeled data changes during the training phase, facilitating its understanding. The visualization of the different phases of these algorithms has been carried out in the same web application used previously, with modifications to accommodate the newly incorporated algorithms.

The web application <https://vass.dmachacha.dev> has been used as a starting point and has evolved into a second version available at [URL](#).

Keywords

machine learning, semi-supervised learning, ensembles, graphs, web

Índice general

Índice general	iii
Índice de figuras	v
Índice de tablas	vii
1. Introducción	1
2. Objetivos del proyecto	5
2.1. Objetivos técnicos	5
2.2. Objetivos de desarrollo <i>software</i>	6
3. Conceptos teóricos	7
3.1. Aprendizaje automático	7
3.2. Aprendizaje semisupervisado	11
3.3. Ensembles	25
3.4. Métodos de evaluación de algoritmos de Aprendizaje Automático	28
4. Técnicas y herramientas	33
4.1. Técnicas	33
4.2. Herramientas	39
5. Aspectos relevantes del desarrollo del proyecto	45
5.1. Lectura artículos científicos	45
5.2. Trabajo Preexistente	46
5.3. Algoritmos	49
6. Trabajos relacionados	61

7. Conclusiones y Líneas de trabajo futuras	65
Bibliografía	67

Índice de figuras

3.1. Reducción de dimensionalidad	10
3.2. <i>Smoothness assumption</i> y <i>Low-density assumption</i> [21]	12
3.3. <i>Manifold assumption</i> [14]	13
3.4. <i>Cluster assumption</i> [14]	13
3.5. Clasificación de los diferentes algoritmos que pretenden incorporar datos no etiquetados a métodos de clasificación. Basado en [21]	14
3.6. Concepto de bagging [20]	25
3.7. Concepto de boosting [20]	26
3.8. Curva AUC-ROC.	31
4.1. Procedimiento en la metodología SCRUM [16]	34
4.2. Resultado de ejecutar Lighthouse en una página web.	41
4.3. Resultado de diagnósticos al ejecutar Lighthouse en una página web.	42
5.1. Prototipo de visualización de algoritmo Co-Forest	48
5.2. Comparación algoritmo CoForest utilizando diversos datasets	50
5.3. Visualización correcta de las 4 fases de construcción del grafo con el algoritmo GBILI utilizando el <i>dataset</i> de <i>iris data</i> . El color de la instancia representa su clase	54
5.4. Mapa de calor que representa el <i>accuracy</i> para el algoritmo <i>LGC</i> con rangos de valores <i>tolerance</i> =[0.1, 8] y <i>alpha</i> =[0.01, 1]. Se muestran los resultados para los <i>datasets</i> : <i>iris</i> , <i>breast cancer</i> y <i>wine</i>	56
5.5. Mapa de calor que representa el <i>accuracy</i> para el algoritmo <i>LGC</i> con rangos de valores <i>alpha</i> =[0.90, 0.99] y <i>tolerance</i> =[0.00001, 1]. Se muestran los resultados para los <i>datasets</i> : <i>iris</i> , <i>breast cancer</i> y <i>wine</i>	58

6.1. Captura de pantalla de la web <i>ML Algorithm Visualizer</i>	62
---	----

Índice de tablas

3.1. Comparación aprendizaje supervisado y no supervisado [18]. . .	10
3.2. Matriz de Confusión	30

1. Introducción

En los últimos años, el aprendizaje automático ha avanzado significativamente, destacando su capacidad para resolver problemas complejos en una variedad de dominios. Sin embargo, uno de los desafíos más persistentes es la obtención de datos etiquetados de alta calidad, los cuales son esenciales para el entrenamiento de modelos supervisados. En este contexto, el aprendizaje semisupervisado se presenta como una solución viable, al aprovechar tanto los datos etiquetados como los no etiquetados para mejorar la precisión de los modelos.

El aprendizaje semi-supervisado incluye una variedad de algoritmos que permiten inferir etiquetas para datos no etiquetados, basándose en la información disponible de los datos etiquetados. Entre estos algoritmos se encuentran los métodos de *ensembles* y los métodos basados en grafos, cada uno con sus propias ventajas y aplicaciones específicas. Los métodos de *ensembles* combinan múltiples modelos de aprendizaje para mejorar el rendimiento general. La idea principal es que, al combinar las predicciones de varios modelos, se puede reducir la varianza y el sesgo, logrando una mejor precisión y robustez. Por otro lado, los métodos basados en grafos representan los datos como nodos en un grafo, donde los enlaces (aristas) entre los nodos indican relaciones o similitudes entre los datos. Este enfoque es especialmente poderoso para capturar la estructura intrínseca de los datos.

Este proyecto se centra en la continuación y ampliación de una herramienta docente previamente desarrollada, la cual incluye una biblioteca de algoritmos semi-supervisados y una aplicación web para la visualización del proceso de entrenamiento. En la versión anterior, se implementaron cuatro algoritmos populares: *Self-Training*, *Co-Training*, *Democratic Co-Learning* y *Tri-Training*.

En esta nueva fase del proyecto, se han añadido nuevos algoritmos para ampliar la funcionalidad de la herramienta. Específicamente, se ha incorporado el algoritmo *Co-Forest* para métodos de ensembles y, para métodos transductivos basados en grafos, se han implementado los algoritmos *GBILI* y *RGCLI* en la fase de construcción de grafos, así como el algoritmo *LGC* (*Local and Global Consistency*) en la fase de inferencia de etiquetas.

La estructura de este documento se organiza de la siguiente manera:

- **Memoria:** documento principal del proyecto que se divide en:
 1. **Introducción:** se ofrece una visión general del proyecto junto con la estructura de la documentación.
 2. **Objetivos:** se enumeran los objetivos generales, técnicos y de desarrollo *software*.
 3. **Conceptos teóricos:** se explica en detalle los conceptos teóricos claves para comprender el desarrollo del proyecto (tanto en la documentación como en la aplicación final).
 4. **Técnicas y herramientas:** se describe las tecnologías y herramientas de desarrollo empleadas en el proyecto.
 5. **Aspectos relevantes del proyecto:** se discuten los aspectos más importantes del desarrollo del proyecto, como los estudios realizados o las dificultades encontradas en pleno desarrollo.
 6. **Trabajos relacionados:** Se revisan otros trabajos y proyectos relevantes en el mismo ámbito que puedan servir de ayuda en el desarrollo.
 7. **Conclusiones y líneas de trabajo futuras:** se evalúa el cumplimiento de los objetivos propuestos y se sugieren posibles mejoras para el futuro.
- **Anexos:** documento que contiene contenido adicional del desarrollo del proyecto, dividido en:
 1. **Plan de proyecto *software*:** se detalla el plan de desarrollo del *software*, incluyendo recursos y estrategias de gestión del proyecto.
 2. **Especificación de requisitos:** se enumeran y describen los requisitos que el sistema debe cumplir usando diagramas y tablas.

3. **Especificación de diseño:** se presenta el diseño detallado de la arquitectura del sistema, describiendo los componentes principales y su interacción.
4. **Documentación técnica de programación:** se proporciona documentación técnica detallada, con el objetivo de simplificar la integración de nuevos desarrolladores en el proyecto y acelerar su comprensión del mismo.
5. **Documentación de usuario:** se ofrece una guía detallada para los usuarios de la aplicación, explicando cómo utilizar las diferentes funcionalidades.

Como recursos adicionales se incluyen:

- **Repositorio del proyecto:** <https://github.com/msp1015/TFG-Semi-Supervised-Learning>
- **Web desplegada:**
 - **Versión 1.0:** <https://vass.dmachacha.dev/>
 - **Versión 2.0:** [URL](#)

2. 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.

Para comprender los objetivos concretos es útil dar un contexto y contar el objetivo general del proyecto. Este trabajo se encuentra en el ámbito del aprendizaje automático, más concretamente en el aprendizaje semisupervizado, tratando de comprender su utilidad y profundizar en algunos de sus algoritmos; y en el ámbito del desarrollo web, con la intención de mejorar una página web ya existente que permite visualizar los algoritmos anteriores. Por lo tanto se podrían detallar tres objetivos generales:

- Investigación exhaustiva sobre aprendizaje semisupervisado y sus algoritmos.
- Implementación de los algoritmos elegidos.
- Desarrollo web para la visualización del resultado de estos algoritmos.

2.1. Objetivos técnicos

Se definen los siguientes objetivos técnicos:

1. Implementar distintos algoritmos de aprendizaje semisupervisado, basados en artículos científicos para conseguir una mayor eficiencia.
2. Comparar las implementaciones propias con otras ajenas para comprobar su funcionamiento

3. Continuar el desarrollo de la web ya implementada: para mejorar su funcionamiento y añadir nuevas funcionalidades.
4. Aprender a realizar experimentaciones de Aprendizaje Automático de forma rigurosa.

2.2. Objetivos de desarrollo *software*

Se definen los siguientes objetivos de desarrollo de software:

1. Implementación de nuevos algoritmos utilizando librerías como *Scikit-Learn* y la última versión de Python a fecha de inicio del proyecto (*Python 3.11.5*)
2. Implementar un código limpio y estandarizado.
3. Crear nuevas interfaces de usuario de visualización de algoritmos, basadas en la idea original.
4. Mejorar ciertas funcionalidades de la web anterior.
5. Conocer métodos de internacionalización, como *babel*.
6. Documentar el proceso de desarrollo: de manera resumida y con información útil.
7. Familiarizarse con la metodología ágil *Scrum*.

3. Conceptos teóricos

En esta sección se resumirán los conceptos teóricos básicos y necesarios para comprender el trabajo. Principalmente se presentará el aprendizaje automático y luego se profundizará en el aprendizaje semisupervisado.

3.1. Aprendizaje automático

El aprendizaje automático (*Machine Learning* en inglés) es el campo de la inteligencia artificial (IA) que se centra en el uso de datos y en el desarrollo de algoritmos para imitar la manera de aprender de los humanos [12]. La esencia radica en la capacidad de los sistemas informáticos para aprender de datos y realizar tareas sin intervención humana directa, si no descubriendo patrones y tendencias en los mismos. A estos sistemas se les conoce como **modelos**, los cuales pueden mejorar su rendimiento y adaptarse a nuevas situaciones basándose en la experiencia pasada.

Según [8], existen cuatro etapas principales en el desarrollo de un modelo. El primer paso consiste en seleccionar y preparar el conjunto de datos (*dataset*) que utilizará el modelo para aprender a resolver el problema para el que se ha diseñado. En el segundo paso se selecciona el algoritmo para ejecutar sobre el *dataset*. Este dependerá del tamaño y el tipo de los datos de entrada y del tipo de problema que se está resolviendo. El tercer paso consiste en entrenar el algoritmo hasta que la mayoría de los resultados sean los esperados. El cuarto y último paso trata de usar el modelo sobre nuevos datos y hacer una evaluación para una posible mejora.

Según los datos que se seleccionen en el primer paso, podemos tener dos ramas distintas en el aprendizaje automático [15]:

- **Predictiva:** también caracterizada por utilizar el aprendizaje supervisado, es decir, datos de entrada etiquetados.
- **Descriptiva:** al contrario, utiliza el aprendizaje no supervisado, con datos de entrada no etiquetados.

Aprendizaje supervisado

El aprendizaje supervisado es un tipo de aprendizaje automático en el que los modelos son entrenados utilizando conjuntos de datos etiquetados, en los que se basarán las decisiones y predicciones. Los conjuntos de datos contienen ejemplos emparejados de variables de entrada (o características) y de salida (o etiquetas). La esencia de este tipo de aprendizaje se basa en la capacidad del modelo para aprender la relación funcional entre las entradas y las salidas, permitiéndole hacer predicciones precisas sobre nuevos datos no vistos anteriormente [9]. De ahí su clasificación como «predictiva» en la sección anterior. Dos tareas habituales dentro del aprendizaje supervisado son:

- **Clasificación:** los modelos asignan categorías o clases a las entradas no etiquetadas. Dentro de este tipo se puede encontrar la clasificación binaria y la multi-clase. La primera se ve en un caso como la clasificación de correos electrónicos marcadas como spam o no spam (solo una etiqueta). Y la segunda se puede ver en cualquier ejemplo en el que haya mas de dos clases, como al establecer si un paciente tiene alto, medio o bajo riesgo de muerte ante una operación.
- **Regresión:** es similar a la clasificación, pero en vez de asignar un valor discreto, ahora es un valor continuo. Un ámbito común en el que se suele dar es en la economía, con la predicción de acciones o ventas.

También es importante comentar las principales fases que forman este aprendizaje y los posibles problemas o desafíos que pueden surgir, ya que pueden servir para tener en cuenta en los algoritmos concretos a implementar. En la mayoría de algoritmos que utilizan datos etiquetados, estos se dividen en tres conjuntos: entrenamiento, validación y prueba. El conjunto de entrenamiento se utiliza para ajustar los parámetros del modelo, el conjunto de validación para ajustar los hiperparámetros y prevenir el sobreajuste y el conjunto de prueba para evaluar el rendimiento final. El sobreajuste u *overfitting* es uno de los principales problemas del aprendizaje automático y ocurre cuando el modelo se ajusta demasiado a los datos de entrenamiento, es decir, los memoriza en vez de generalizar.

Aprendizaje no supervisado

El aprendizaje no supervisado hace referencia a los tipos de problemas en los que se utiliza un modelo para caracterizar o extraer relaciones en los datos. A diferencia del aprendizaje supervisado, estos algoritmos descubren la estructura implícita de un conjunto de datos utilizando únicamente características de entrada y no clases o categorías. Ya que no existen etiquetas en los datos, los métodos no supervisados se utilizan normalmente para crear una representación concisa de los datos, posibilitando la generación de contenido creativo a partir de ellos. Por ejemplo, si tenemos una gran cantidad de fotografías sin clasificar, un modelo no supervisado encontraría relaciones entre las características para poder organizar automáticamente las imágenes en grupos [18]. Principalmente, se pueden clasificar en tres diferentes tareas:

- **Clustering:** segmentación o agrupamiento. Consiste en la identificación de grupos o *clusters* en función de sus similitudes y diferencias. Dentro de este tipo, se puede diferenciar un agrupamiento exclusivo, donde los datos pertenecen a un único grupo, y un agrupamiento superpuesto, donde los datos pueden pertenecer a varias agrupaciones. El ejemplo de las fotografías entra dentro de esta categoría.
- **Reglas de asociación:** utiliza una medida de interés para obtener un conjunto de reglas sólidas que permitan descubrir asociaciones interesantes entre las características de un conjunto de datos. La principal aplicación es el «análisis de cestas de compra», que se usa para determinar los patrones de compra de los clientes en función de las relaciones entre productos.
- **Reducción de dimensionalidad:** estos algoritmos buscan reducir la complejidad de un conjunto de datos de alta dimensión a espacios de baja dimensión sin perder propiedades fundamentales de los datos originales [3]. Este tipo de algoritmos se utiliza en la fase de análisis de datos, facilitando la representación gráfica. Se puede ver un ejemplo en la figura 3.1.

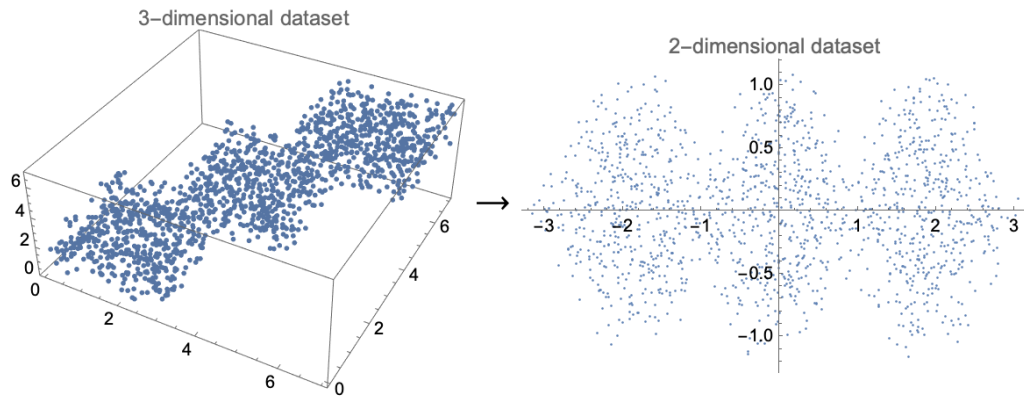


Figura 3.1: Reducción de dimensionalidad

En la tabla 3.1 se resumen las principales diferencias entre aprendizaje supervisado y no supervisado:

	Supervisado	No supervisado
Objetivo	Aproximar una función que asigna entradas a salidas a partir de un conjunto de datos clasificados.	Crear una representación concisa de los datos, posibilitando la generación de contenido creativo a partir de ellos.
Complejidad	Simple	Mayor
Entrada	Se conoce el número de clases (datos etiquetados).	No se conoce el número de clases (datos no etiquetados).
Salida	Genera un valor de salida esperado.	No se tienen valores de salida asociados
Tareas	Clasificación, Regresión	Clustering, Reglas de asociación, Reducción de dimensionalidad

Tabla 3.1: Comparación aprendizaje supervisado y no supervisado [18].

3.2. Aprendizaje semisupervisado

Como el nombre sugiere, el aprendizaje semisupervisado se encuentra entre los dos tipos vistos anteriormente. Los algoritmos dentro de esta estrategia se basan en extender cualquiera de los aprendizajes, supervisado o no supervisado, para añadir información adicional que el otro no proporciona [25].

Los métodos de clasificación semi-supervisada intentan utilizar puntos de datos no etiquetados para generar un modelo cuyo rendimiento supere el de los modelos obtenidos al utilizar solo datos etiquetados [21].

Por ejemplo, imaginemos que se está trabajando en la clasificación de imágenes médicas para identificar diferentes tipos de enfermedades. En este caso, consideramos específicamente la detección temprana de ciertos tipos de cáncer a partir de imágenes de tomografías. En un enfoque supervisado, podríamos entrenar un modelo utilizando un conjunto de datos etiquetado que incluye imágenes con diagnósticos de cáncer y sin cáncer. Sin embargo, la obtención de un gran conjunto de datos etiquetado puede ser costosa y consume tiempo. En un escenario de aprendizaje semisupervisado, además de los datos etiquetados, podríamos tener un conjunto de datos mucho más grande que incluye imágenes no etiquetadas. Algunas de estas pueden contener señales sutiles o características asociadas con el cáncer que no han sido previamente etiquetadas.

El modelo de aprendizaje semisupervisado podría analizar estas imágenes no etiquetadas y descubrir patrones que podrían indicar la presencia temprana de cáncer. Por ejemplo, podría aprender a reconocer características microscópicas específicas de las imágenes que no son evidentes para el ojo humano. Cuando se encuentra con nuevas imágenes no etiquetadas que comparten estas características, el modelo podría clasificarlas como indicativas de la presencia de cáncer, incluso si no ha visto exactamente esas características en el conjunto de datos etiquetado. Existe una condición necesaria en el aprendizaje semisupervisado: la distribución marginal subyacente $p(x)$ sobre el espacio de entrada debe contener información acerca de la distribución posterior $p(x|y)$ [21]. Es decir, la naturaleza de los datos no etiquetados debe contener información útil para inferir las etiquetas correspondientes. Esta suposición es básica y en la mayoría de los ejemplos se cumple. Aún así, como la manera de interactuar entre $p(x)$ y $p(x|y)$ no es siempre la misma, se pueden tomar malas decisiones que conllevarían un rendimiento cada vez peor. Por esta razón, existen tres principales suposiciones que

todo conjunto de datos de un algoritmo semisupervisado debe cumplir para funcionar correctamente.

- ***Smoothness assumption***: traducida como suposición de suavidad, consiste en que para dos puntos x_1 y x_2 que están cerca en una región densa, entonces sus correspondientes salidas (o etiquetas) y_1 y y_2 deben ser las mismas. Esto es útil sobretodo con datos no etiquetados, ya que por la propiedad transitiva, dos puntos que no estén relativamente cerca, pueden ser de la misma clase.
- ***Low-density assumption***: esta suposición está definida sobre la distribución de datos de entrada $p(x)$ y dice que el límite de decisión en la clasificación debe pasar antes por un área de poca densidad que por una de mayor densidad. Esto se puede observar en la figura 3.2.

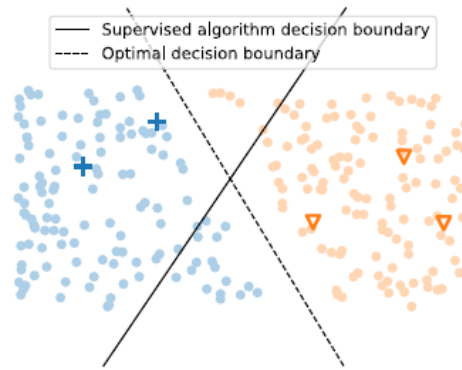
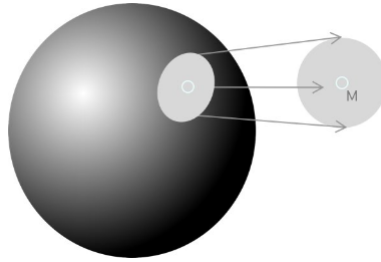
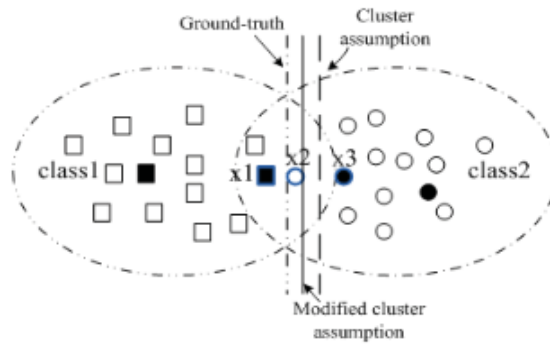


Figura 3.2: *Smoothness assumption* y *Low-density assumption* [21]

- ***Manifold assumption***: esta suposición afirma que los datos utilizados se encuentran en un *manifold* de baja dimensión incrustado en un espacio de mayor dimensión. En otras palabras, los datos, en lugar de proceder de cualquier parte del espacio, deben proceder de estos *manifolds* de dimensiones más bajas [14].

Figura 3.3: *Manifold assumption* [14]

En algunas ocasiones, aparece una cuarta suposición: ***cluster assumption***. Esta indica que dos datos que pertenecen a un mismo *cluster*, pertenecen también a la misma clase. Se tomará esta suposición como una generalización de las tres anteriores [21].

Figura 3.4: *Cluster assumption* [14]

Taxonomía

No hay una clasificación oficial de algoritmos de aprendizaje semisupervisado, pero sí se pueden encontrar aproximaciones teniendo en cuenta las suposiciones en las que están basados los algoritmos y en cómo se relacionan con los algoritmos supervisados y no supervisados.

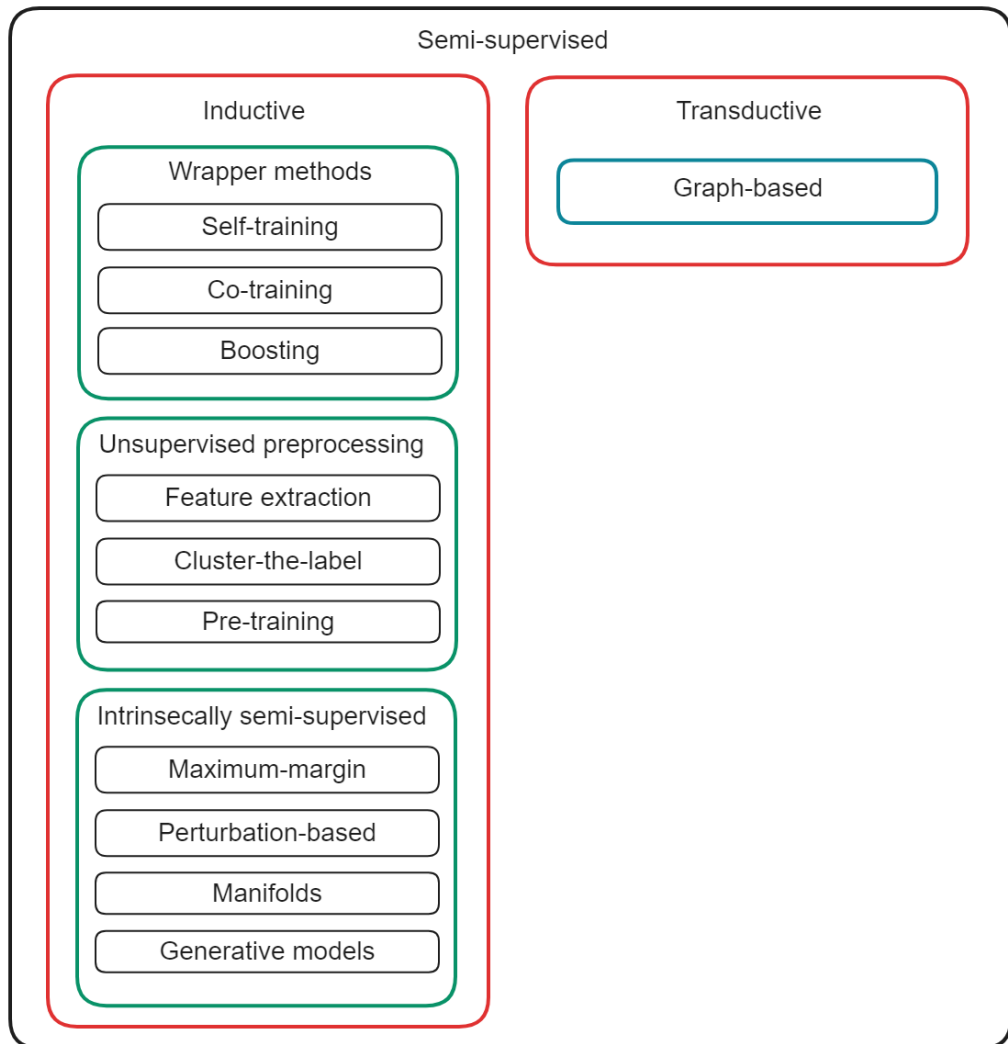


Figura 3.5: Clasificación de los diferentes algoritmos que pretenden incorporar datos no etiquetados a métodos de clasificación. Basado en [21]

Métodos inductivos

Los métodos inductivos en aprendizaje semi-supervisado tienen como objetivo construir un clasificador que pueda generar predicciones para cualquier objeto dentro del espacio de entrada. Durante la fase de entrenamiento de este clasificador, se pueden utilizar datos no etiquetados junto con los datos etiquetados disponibles. Una vez que el entrenamiento del modelo se ha completado, las predicciones que el clasificador realiza para nuevos datos son independientes entre sí. Esto significa que el modelo puede ge-

neralizar y hacer predicciones sobre nuevos ejemplos que no fueron vistos durante el entrenamiento, basándose en el conocimiento adquirido de los datos etiquetados y no etiquetados utilizados durante el entrenamiento [21].

- ***Wrapper methods***: Estos métodos inicialmente entrenan clasificadores utilizando únicamente los datos etiquetados. Después, las predicciones generadas por estos clasificadores se utilizan para etiquetar datos no etiquetados adicionales, creando así un conjunto de datos pseudo-etiquetados. El clasificador se vuelve a entrenar con estos datos adicionales, mejorando su capacidad para generalizar a partir de la información combinada de los datos originalmente etiquetados y los pseudo-etiquetados. En este grupo entrarían los cuatro algoritmos implementados en el trabajo anterior.
- ***Unsupervised preprocessing***: En este enfoque, los métodos no supervisados se utilizan para extraer características útiles, agrupar datos o determinar parámetros iniciales de aprendizaje antes de entrenar el clasificador supervisado. Este preprocesamiento con datos no etiquetados mejora el rendimiento del clasificador supervisado, aprovechando las estructuras y patrones presentes en los datos no etiquetados.
- ***Intrinsically semi-supervised***: Estos métodos integran directamente los datos no etiquetados en la función objetivo o en el procedimiento de optimización del modelo de aprendizaje. Al incorporar datos no etiquetados en la función objetivo, se maximiza la información obtenida durante el proceso de aprendizaje, lo que lleva a una mejor generalización y rendimiento del clasificador.

Métodos transductivos

A diferencia de los métodos inductivos, los métodos transductivos no construyen un modelo para todo el espacio de entrada. En su lugar, su poder predictivo se limita exactamente a los objetos que encuentra durante la fase de entrenamiento. Por lo tanto, los métodos transductivos no tienen fases de entrenamiento y predicción distintas [21].

El aprendizaje transductivo puede ahorrar tiempo y es preferible cuando el objetivo se orienta a mejorar nuestro conocimiento sobre el conjunto de datos sin etiquetar. Sin embargo, este enfoque tiene limitaciones, especialmente cuando queremos entender causas y efectos dentro de los datos. Básicamente, aunque el aprendizaje transductivo es útil para hacer inferencias específicas sobre los datos no etiquetados basándose en los datos etiquetados, no es

adecuado ni efectivo para estudiar o predecir relaciones causales, es decir, cómo un factor directamente provoca otro. Esto se debe a que el aprendizaje transductivo se centra sólo en los datos presentes durante el entrenamiento y no generaliza más allá de estos [14]. Los métodos transductivos suelen definir un grafo sobre todos los puntos de datos, tanto etiquetados como no etiquetados, codificando la similitud entre pares de puntos de datos con aristas posiblemente ponderadas. Para definir más en detalle el aprendizaje semisupervisado basado en grafos (GSSL por sus siglas en inglés), se ha utilizado como base los conceptos del artículo [21] y para detallar más información se ha utilizado uno de los artículos más actualizados en este ámbito hasta la fecha [19]. Ambos coinciden en que estos algoritmos se dividen en dos pasos: creación del grafo y fase de inferencia (se refiere al proceso de asignar etiquetas o categorías a los nodos no etiquetados en un grafo utilizando la información de los nodos etiquetados y la estructura del grafo).

Construcción de grafos

En el contexto del aprendizaje semi-supervisado basado en grafos (GSSL), la construcción del grafo es un paso crítico. Durante esta fase, se crea un grafo que representa todos los datos disponibles, tanto etiquetados como no etiquetados. Los nodos representan muestras y las aristas ponderadas reflejan la similitud entre pares de muestras. Existen varios métodos para construir estos grafos, que pueden clasificarse en métodos no supervisados y supervisados.

- **Métodos no supervisados:** Los métodos no supervisados ignoran la información de las etiquetas durante la construcción del grafo. Entre los enfoques más populares se encuentran:
 - ***k*-Nearest Neighbours:** cada nodo se conecta a sus k vecinos más cercanos según alguna medida de distancia. Puede tener dos variantes: *symmetric k nearest neighbours*, que construye una arista si i o j están en el “vecindario” de k y *mutual k-nearest neighbours* que la construye si ambos están en la k vecindad del otro, es decir, la relación es bidireccional.
 - ***b*-Matching:** Asegura que cada nodo tenga exactamente b vecinos, lo que resulta en un grafo regular donde las etiquetas se propagan de manera equilibrada.
- **Métodos supervisados:** Estos métodos utilizan la información de las muestras etiquetadas para refinar el grafo durante su construcción. Algunos ejemplos son:

- ***GBILI (Graph-based on Informativeness of Labeled Instances)*** [3.2]: Utiliza información de las etiquetas para crear grafos más precisos y robustos.
- ***RGCLI (Robust Graph that Considers Labeled Instances)*** [36]: Optimiza la construcción del grafo para mejorar la precisión en tareas de inferencia posteriores.

Dentro de esta fase, un proceso importante es ponderar el grafo (dar pesos a las aristas). En primer lugar, se construye una matriz de adyacencia completa utilizando una función k y después se obtiene la matriz de pesos W mediante *sparsification*¹. Uno de los métodos más populares y sencillos, y en el que se profundizará en secciones más adelante, consiste en ponderar los enlaces entre los nodos con un 1 si existe relación entre ellos y con un 0 en caso contrario. Ponderar las aristas es crucial ya que refleja la fuerza de las relaciones entre nodos, lo que puede influir significativamente en la propagación de etiquetas y en la precisión final del algoritmo.

Fase de inferencia

En la fase de inferencia del aprendizaje semisupervisado basado en grafos (GSSL), se utilizan dos enfoques principales: la regularización en grafos y el *embedding* en grafos. Estos métodos se centran en la propagación de etiquetas desde los nodos etiquetados a los no etiquetados mediante la estructura del grafo construido.

- **Regularización en grafos:** La regularización en grafos busca encontrar una función de predicción que sea lo más cercana posible a las etiquetas conocidas y que mantenga una suavidad en todo el grafo. Esto significa que nodos conectados con una fuerte relación (representada por el peso de los bordes) deberían tener etiquetas similares. Los métodos de regularización en grafos se pueden clasificar en varias categorías:
 - **Propagación de etiquetas (*Label propagation*):** Este método es uno de los más populares para la inferencia de etiquetas en GSSL. Se basa en la idea de que las etiquetas de algunos nodos (semillas) se pueden propagar a los nodos no etiquetados basándose en la similitud representada por el grafo.

¹Sparsification es el proceso de eliminar elementos menos significativos de una matriz o aristas en un grafo, con el fin de hacer la estructura más esparcida y eficiente en términos de almacenamiento y procesamiento.

- **Campos Aleatorios Gaussianos (*Gaussian Random Fields*)**: Utiliza una función de predicción que debe cumplir con ciertas restricciones para asegurar que los nodos conectados compartan etiquetas similares.
 - **Consistencia Local y Global (*Local and Global Consistency*)** [36]: Este método extiende el enfoque de GRF a un contexto multiclase y trata de minimizar una función objetivo que penaliza las diferencias entre las etiquetas inferidas y las etiquetas conocidas.
- **Regularización dirigida**: Este enfoque extiende la regularización a grafos dirigidos, utilizando caminos aleatorios para manejar la direccionalidad de los bordes.
- **Regularización de variedad (*Manifold Regularization*)**: Esta técnica aprovecha la geometría intrínseca de los datos.
- ***Embedding* en grafos**: El *embedding* en grafos transforma la información estructural del grafo en un espacio de menor dimensión, facilitando la tarea de inferencia de etiquetas. Este enfoque se divide en tres componentes principales:
 - **Codificador**: transforma cada nodo del grafo en un vector de características de baja dimensión, preservando la estructura del grafo y la información de los nodos.
 - **Decodificador**: Reconstruye la información del grafo a partir de los vectores de características generados por el codificador.
 - **Reconstrucción**: El objetivo es minimizar la diferencia entre las medidas de similitud en el grafo original y las predicciones del decodificador.

En resumen, los métodos transductivos en el aprendizaje semi-supervisado basado en grafos ofrecen un enfoque potente y escalable para aprovechar tanto datos etiquetados como no etiquetados. Aunque presentan ciertas limitaciones en la generalización y el estudio de relaciones causales, su capacidad para mejorar el conocimiento sobre conjuntos de datos no etiquetados los hace altamente valiosos en diversas aplicaciones.

Graph-based on informativeness of labeled instances **(GBILI)**

El algoritmo ***GBILI*** o Construcción de Grafos Basada en la Informatividad de Instancias Etiquetadas [4] es un método de construcción de grafos para el aprendizaje semisupervisado. Su principal característica es que se basa en la informatividad de las instancias etiquetadas para relacionar nodos dentro del grafo, pudiendo aprovechar después esta conectividad para predecir las etiquetas de los datos no etiquetados. Se utilizan los métodos de k -vecinos más cercanos y k -vecinos más cercanos mutuos para inicializar el grafo. En el pseudocódigo 1, propuesto en, se puede ver el proceso detallado.

Metodología

El algoritmo aprovecha la información de las etiquetas disponibles para priorizar las conexiones entre los vértices, especialmente aquellos que están más cerca de un punto etiquetado, convirtiendo así estos puntos etiquetados en hubs a medida que aumenta el valor de k . Los pasos que sigue son:

1. Generación de la Matriz de Distancias: Se crea una matriz de distancias D usando la distancia euclidiana para determinar los k vecinos más cercanos de cada elemento.
2. Configuración de Parámetros: Se establece el parámetro k con un valor natural y se genera una lista de puntos etiquetados L .
3. Búsqueda de Vecinos Más Cercanos: Para cada vértice v_i , se encuentran sus k vecinos más cercanos.
4. Determinación de Vecinos Mutuos: Se identifican los k vecinos mutuos para v_i .
5. Cálculo de la Informatividad: Se calcula la suma de las distancias desde v_i a cada elemento de vecinos mutuos y desde estos elementos hasta un punto etiquetado. Se establece una conexión entre v_i y v_j que minimice esta suma. Este aspecto puede verse confuso y por ello se comenta en la sección 19.
6. Post-procesamiento del Grafo: Se conectan los componentes aislados mediante una búsqueda en anchura (*BFS*) para encontrar componentes en la red. Los componentes sin puntos etiquetados se conectan con componentes vecinos que tienen puntos etiquetados, limitando el número de nuevas conexiones para evitar una red demasiado densa.

Algoritmo 1: GBILI

Input: Conjunto de datos etiquetados L , conjunto de datos no etiquetados U , número de vecinos más cercanos K

Output: Grafo G

```

1  Generar matriz de distancias  $D$  entre todos los puntos de datos
2  Establecer el parámetro  $K$ 
3  for  $i=1; i < |V|; i++$  do
4      for  $k=1; k < K; k++$  do
5          for  $j=1; j < |V|; j++$  do
6              if  $D(v_i, v_j)$  es el  $kNN$  then
7                   $listakNN(v_i) \leftarrow v_j$ 
8              end if
9          end for
10     end for
11     for  $j=1; j < listakNN(v_i); j++$  do
12         for  $k=1; k < K; k++$  do
13             if  $D(v_j, v_i)$  es el  $kNN$  then
14                  $listaMutuoskNN(v_i) \leftarrow v_j$ 
15             end if
16         end for
17     end for
18     for  $j=1; j < listaMutuoskNN(v_i); j++$  do
19         for  $l=1; l < L; l++$  do
20             if  $D(v_i, v_j) + D(v_j, v_i)$  es mínima then
21                  $G \leftarrow e_{i,j}$ 
22             end if
23         end for
24     end for
25 end for
26  $Componentes = BFS(G)$ 
27 for  $i = 1; i < |V|; i++$  do
28     if  $Componentes(v_i) \notin L$  then
29         for  $k = 1; k < listakNN(v_i); k++$  do
30             if  $Componentes(v_k) \in L$  then
31                  $G \leftarrow e_{i,k}$ 
32             end if
33         end for
34     end if
35 end for
36 return  $G$ 

```

Como se comenta en uno de los pasos, hay ciertos puntos en este algoritmo que deben ser comentados para evitar confusión en la implementación. Además, para este trabajo en concreto, el algoritmo sufre ciertas modificaciones (con la misma idea teórica) que se comentan en la sección 19.

Robust Graph that Considers Labeled Instances (RGCLI)

El algoritmo **RGCLI** (Grafo Robusto que Considera Instancias Etiquetadas) es una mejora del método GBILI, diseñado para la construcción de grafos robustos en el aprendizaje semi-supervisado [5]. Este método aprovecha la informatividad de las instancias etiquetadas para generar grafos más eficaces en la propagación de etiquetas, asegurando una mejor representación de la estructura de los datos y optimizando el rendimiento de las tareas de clasificación. GBILI tiene limitaciones significativas, como una alta complejidad temporal cuadrática, lo que dificulta su aplicación en conjuntos de datos grandes. RGCLI mejora GBILI al reducir la complejidad temporal de cuadrática a $O(nk \log n)$, o que permite manejar conjuntos de datos grandes de manera más eficiente. Además, RGCLI demuestra matemáticamente que el grafo construido es óptimo para modelar la suposición de suavidad.

El algoritmo original de RGCLI sigue un enfoque optimizado para construir grafos que satisfacen las suposiciones de consistencia del aprendizaje semisupervisado, tanto a nivel local como global. Utiliza hilos para realizar una ejecución concurrente y estructuras poco costosas como *kdtrees*². Como se puede observar en el algoritmo 2, recibe dos parámetros, k_e y k_i que indican el número de vecinos a encontrar en cada fase del algoritmo. Después de inicializar todas las estructuras necesarias, el algoritmo se divide en dos fases:

1. **Búsqueda de vecinos más cercanos:** Se construye un *kdtree* a partir de los datos de entrada y se utiliza para buscar los k_e vecinos más cercanos para cada punto. Además, se almacena información sobre los vecinos etiquetados más cercanos.
2. **Construcción del grafo RGCLI:** Se calculan los vecinos mutuos más cercanos (MkNN) y se asignan puntuaciones basadas en las distancias a los vecinos y a los puntos etiquetados. Se establecen conexiones entre los puntos utilizando los k_i vecinos que minimizan las puntuaciones calculadas. La construcción del grafo prioriza las conexiones con puntos cercanos y etiquetados, generando un grafo esparso y eficiente para la propagación de etiquetas.

²«En ciencias de la computación, un Árbol kd es una estructura de datos de particionado del espacio que organiza los puntos en un espacio euclídeo de k dimensiones, empleando solo planos perpendiculares a uno de los ejes del sistema de coordenadas» [23].

Algoritmo 2: RGCLI

Input: número de k vecinos más cercanos k_e , número de RGCLI vecinos más cercanos k_i , lista de datos etiquetados L , dataset X , número de hilos nt

Output: Grafo G

```

1  $V \leftarrow$  vértices a partir de  $X$ 
2  $E, W \leftarrow \emptyset$ 
3  $G \leftarrow (V, E, W)$ 
4  $kdtree \leftarrow$  a partir de  $X$ 
5  $kNN \leftarrow \text{map}$ 
6  $F \leftarrow \text{map}$ 
7  $L \leftarrow \text{map}$ 
8  $T \leftarrow \{T_i : \bigcup_{i=1}^{nt} T_i = V, \bigcap_{i=1}^{nt} T_i = \emptyset\}$ 
9 for  $T_i \in T$  do
10    $t \leftarrow \text{Hilo}(\text{SearchKNN}(T_i, k_e, kdtree, kNN, L))$ 
11    $t.\text{start}()$ 
12 end for
13 for  $T_i \in T$  do
14    $t \leftarrow \text{Hilo}(\text{SearchRGCLI}(G_L, T_i, k_i, kNN, L))$ 
15    $t.\text{start}()$ 
16 end for
17 return  $G$ 

18 Function  $\text{SearchKNN}(T, k_e, kdtree, kNN, L)$ :
19   for  $v_i \in T$  do
20      $kNN[v_i] \leftarrow kdtree.\text{query}(v_i, k_e)$ 
21      $L[v_i] \leftarrow$  encontrar puntos etiquetados más cercanos en  $L$ 
22      $F[v_i] \leftarrow$  encontrar el  $k$ -ésimo vecino más lejano de  $v_i$ 
23   end for

24 Function  $\text{SearchRGCLI}(G_L, T, k_i, kNN, L)$ :
25   for  $v_i \in T$  do
26      $\epsilon \leftarrow \text{mapa}$ 
27     for  $v_j \in kNN[v_i]$  do
28       if  $\text{dist}(v_i, v_j) \leq \text{dist}(v_j, F[v_j])$  then
29          $e \leftarrow (v_j, v_i)$ 
30          $\epsilon[e] \leftarrow \text{dist}(v_i, v_j) + \text{dist}(v_j, L[v_j])$ 
31       end if
32     end for
33   end for
34    $E^* \leftarrow$  obtener  $k_i$  aristas con la menor puntuación de  $\epsilon$ 
35    $E \leftarrow E \cup E^*$ 
36    $w(e) \leftarrow 1 \forall e = (v_i, v_j) \in E^*$ 

```

En resumen, el algoritmo RGCLI mejora significativamente la construcción de grafos para SSL al reducir la complejidad temporal y utilizar instancias etiquetadas de manera efectiva. El uso de *kdtrees* optimiza la búsqueda de vecinos más cercanos, y las fases del algoritmo garantizan la creación de un grafo robusto y esparso que facilita la propagación de etiquetas y mejora el rendimiento del aprendizaje semisupervisado.

Local and Global Consistency

Este algoritmo aborda el problema general de aprender a partir de grafos, utilizando tanto datos etiquetados como no etiquetados. Dado un conjunto de puntos X y un conjunto de etiquetas L , los primeros l puntos tienen etiquetas y los puntos restantes no están etiquetados. El objetivo es predecir las etiquetas de los puntos no etiquetados. Este proceso se conoce como la fase de inferencia en los algoritmos de aprendizaje semi-supervisado basados en grafos (GSSL, por sus siglas en inglés). El algoritmo está fuertemente basado en dos suposiciones fundamentales:

- **Homogeneidad Local:** Los puntos cercanos tienden a tener la misma etiqueta.
- **Homogeneidad Global:** Los puntos en la misma estructura (como un *cluster*) probablemente tengan la misma etiqueta.

La idea principal es que cada punto propague iterativamente su información de etiqueta a sus vecinos hasta que se alcance un estado global estable. El pseudocódigo 3 y las aclaraciones siguientes están basados en el artículo [24].

- **Crear la matriz de afinidad:** Calcula una matriz W que mide la similitud entre cada par de puntos en X . Si dos puntos están muy cerca entre sí, su valor en la matriz W será alto. Los valores en la diagonal de la matriz (que corresponden a la similitud de un punto consigo mismo) se establecen en cero.
- **Normalizar³ la matriz de afinidad:** Ajusta la matriz W para obtener una nueva matriz S . Esto es necesario para asegurar que los valores se propaguen correctamente en los pasos siguientes.
- **Propagar la información de etiquetas:** Comienza con una matriz inicial F que contiene las etiquetas conocidas. Repite un proceso en el que cada punto actualiza su etiqueta basada en las etiquetas de sus puntos vecinos y sus etiquetas iniciales. Este proceso se repite hasta que las etiquetas dejan de cambiar significativamente.
- **Asignar etiquetas finales:** Una vez que la propagación de etiquetas ha convergido (ya no cambia mucho), asigna a cada punto la etiqueta

³**Normalizar** es un proceso en el que se ajustan los valores de una matriz o conjunto de datos para que sean más comparables y manejables

de la clase con la que tiene mayor afinidad. Esto se hace eligiendo la etiqueta que corresponde al valor más alto en la matriz F para cada punto.

Algoritmo 3: Local and Global Consistency

Input: Conjunto de puntos $X = \{x_1, \dots, x_l, x_{l+1}, \dots, x_n\} \subseteq R^m$, conjunto de etiquetas $L = \{1, \dots, c\}$

Output: Etiquetas predichas para los puntos no etiquetados

```

1 Paso 1: Formar la matriz de afinidad  $W$ 
2 for  $i = 1$  to  $n$  do
3   for  $j = 1$  to  $n$  do
4     if  $i \neq j$  then
5        $W_{ij} \leftarrow \exp(-\|x_i - x_j\|^2 / 2\epsilon^2)$ 
6     end if
7     else
8        $W_{ii} \leftarrow 0$ 
9     end if
10  end for
11 end for

12 Paso 2: Construir la matriz  $S = D^{-1/2} W D^{-1/2}$ 
13  $D$  es una matriz diagonal con el elemento  $(i, i)$  igual a la suma de la  $i$ -ésima fila de  $W$ 

14 Paso 3: Iterar  $F^{(t+1)} = \alpha S F^{(t)} + (1 - \alpha) Y$  hasta la convergencia
15  $\alpha$  es un parámetro  $\in (0, 1)$ 

16 Paso 4: Asignar etiquetas
17 Deja que  $F^*$  denote el límite de la secuencia  $\{F^{(t)}\}$ 
18 for  $i = 1$  to  $n$  do
19    $y_i \leftarrow \arg \max_{j \leq c} F_{ij}^*$ 
20 end for
21 return Etiquetas predichas para los puntos no etiquetados

```

Este algoritmo, como se ha podido llegar a observar, podría no usar directamente la unión física entre nodos del grafo, es decir, no utilizar el paso anterior de construcción del grafo, y aún así, funcionar correctamente, ya que dispondría de la información que da la matriz de distancias. Por ello, para el interés de este trabajo, es necesario buscar una manera de utilizar esta información, la cual se comenta en la sección 19.

3.3. Ensembles

Los sistemas basados en *ensembles* se refieren al proceso de combinar las opiniones de un conjunto de modelos diferentes para tomar una decisión final. Se ha descubierto que los *ensembles* pueden producir resultados más favorables en comparación con los sistemas de un solo experto en una amplia gama de aplicaciones. La creación de un *ensemble* implica generar componentes individuales del sistema y luego combinar sus clasificaciones. En este contexto, se destacan dos técnicas principales: *bagging* y *boosting* [17].

Bagging

El *Bagging*, o *Bootstrap Aggregating*, es una técnica que mejora la estabilidad y precisión de los algoritmos de aprendizaje automático. Funciona mediante la creación de múltiples versiones de un predictor y utilizando estos para obtener un conjunto agregado. Se generan diferentes conjuntos de entrenamiento, cada uno mediante muestreo con reemplazo del conjunto original, y se entrena un modelo en cada uno de ellos. La decisión final se toma por mayoría de votos para clasificación o el promedio en regresión. Esta técnica es particularmente efectiva con modelos que tienen alta varianza.

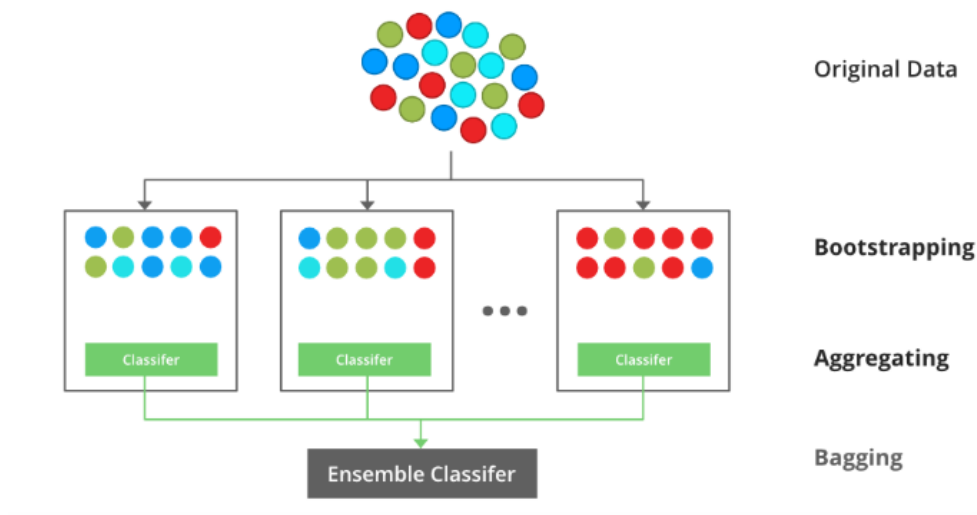


Figura 3.6: Concepto de bagging [20]

Boosting

El *Boosting* es un enfoque que construye secuencialmente un conjunto de modelos; cada nuevo modelo se enfoca en corregir los errores cometidos por los modelos anteriores. AdaBoost, uno de los algoritmos de *boosting* más conocidos, ajusta los pesos de las instancias incorrectamente clasificadas para que modelos posteriores se enfoquen más en ellas. A diferencia del *bagging*, el *boosting* puede aumentar el riesgo de sobreajuste si el conjunto de datos es ruidoso, pero generalmente produce modelos más precisos.

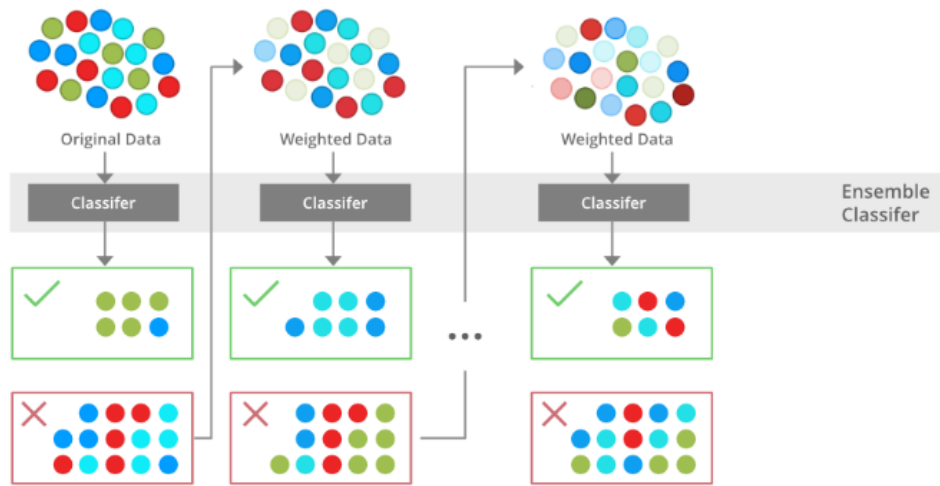


Figura 3.7: Concepto de boosting [20]

Co-Forest

El co-forest es una versión semisupervisada del método de clasificación *random forest*, diseñado para usar tanto datos etiquetados como no etiquetados. En este método, los árboles de decisión (que conforman el *random forest*) se entrenan en un proceso iterativo utilizando subconjuntos de datos etiquetados junto con pseudo-etiquetas seleccionadas de los datos no etiquetados basadas en su alta confiabilidad. La confiabilidad de estas pseudo-etiquetas es evaluada por todos los árboles del ensemble excepto el árbol que está siendo entrenado en ese momento, denominado el conjunto concomitante. El entrenamiento continúa hasta que se alcanza un criterio de parada definido como el *Out Of Bag Error* (OOBE), que mide el error de predicción del conjunto concomitante usando solo aquellos árboles que no incluyeron una

muestra específica de los datos etiquetados en su entrenamiento [13]. A continuación se muestra el pseudocódigo y una explicación más extensa:

Algoritmo 4: Co-Forest

Input: Conjunto de datos etiquetados L , conjunto de datos no etiquetados U , número de árboles n , umbral de confianza θ , sumatorio de confianzas inicial $W_{inicial}$ y parámetros para los árboles de decision p

Output: *Ensemble* de árboles entrenado H

```

1 for  $i = 0, \dots, n - 1$  do
2    $L_i \leftarrow \text{Bootstrap}(L)$ 
3    $h_i = \text{EntrenarArbol}(L_i, p)$ 
4    $\hat{e}_{i,t} \leftarrow 0,5$ 
5    $W_{i,0} \leftarrow W_{inicial}$ 
6 end for
7  $t \leftarrow 0$ 
8 while Algún árbol recibe pseudo-etiquetas do
9    $t \leftarrow t + 1$ 
10  for  $i = 0, \dots, n - 1$  do
11     $\hat{e}_{i,t} \leftarrow \text{EstimateError}(H_i, L)$ 
12     $L'_{i,t} \leftarrow \emptyset$ 
13    if  $\hat{e}_{i,t} < \hat{e}_{i,t-1}$  then
14       $W_{max} = \hat{e}_{i,t-1} W_{i,t-1} / \hat{e}_{i,t}$ 
15       $U'_{i,t} \leftarrow \text{Submuestrear}(U, H_i, W_{max})$ 
16       $W_{i,t} \leftarrow 0$ 
17      foreach  $x_j \in U'_{i,t}$  do
18        if  $\text{Confianza}(H_i, x_j) > \theta$  then
19           $L'_{i,t} \leftarrow L'_{i,t} \cup x_j, H_i(x_j)$ 
20           $W_{i,t} \leftarrow W_{i,t} + \text{Confianza}(H_i, x_j)$ 
21        end if
22      end foreach
23    end if
24  end for
25  for  $i = 0, \dots, n - 1$  do
26    if  $(\hat{e}_{i,t} * W_{i,t} < \hat{e}_{i,t-1} * W_{i,t-1})$  then
27       $h_i = \text{ReentrenarArbol}(L_i \cup L'_{i,t})$ 
28    end if
29  end for
30 end while
31 return  $H$ 

```

Inicialización de Variables:

- Cada árbol del *ensemble*, h_i , se inicializa entrenándolo con una muestra *bootstrap* del conjunto etiquetado L .
- Se establece un error estimado inicial, $\hat{e}_{i,t}$, a 0.5 para cada árbol.
- Se asigna el sumatorio de confianzas inicial a cada árbol. Basado en el estudio comentado en la sección 5.3.

Proceso iterativo:

- El bucle continúa mientras al menos un árbol pueda recibir nuevas pseudo-etiquetas para entrenamiento.
- Se estima el error actual (*OOBE*) del árbol usando el conjunto etiquetado.
- Se inicializa un conjunto temporal $L'_{i,t}$ para acumular nuevas pseudo-etiquetas aceptadas.
- Si el error estimado del árbol mejora (disminuye respecto a la iteración anterior), se procede a submuestrear el conjunto no etiquetado U basándose en el peso máximo W_{max} , que ajusta la cantidad de datos a submuestrear en función de la mejora en el error.
- Cada dato submuestreado se evalúa, y si la confianza en su pseudo-etiqueta supera el umbral θ , se añade al conjunto temporal $L'_{i,t}$ con su correspondiente pseudo-etiqueta, y se actualiza el sumatorio de confianzas para ese árbol.

Reentrenamiento de árboles: Después de evaluar y potencialmente agregar nuevas pseudo-etiquetas, se decide si reentrenar el árbol. Esto se basa en una comparación del producto de error y sumatorio de confianzas actual contra el de la iteración anterior. Si el producto actual es menor, se procede a reentrenar el árbol incorporando las nuevas pseudo-etiquetas. Esto ayuda a mantener la calidad del modelo y a evitar el sobreajuste.

3.4. Métodos de evaluación de algoritmos de Aprendizaje Automático

La evaluación de algoritmos de *Machine Learning* es fundamental para asegurar su rendimiento y capacidad de generalización. A continuación, se presenta un resumen de los conceptos y métodos más utilizados, siguiendo la guía [11].

Random Split

Random Split consiste en muestrear aleatoriamente un porcentaje de datos en conjuntos de entrenamiento, prueba y, preferiblemente, validación.

La ventaja de este método es que asegura una representación adecuada de la población original en los tres conjuntos, evitando un muestreo sesgado.

Es importante usar el conjunto de validación en la selección de modelos, como paso intermedio a la evaluación final, para ajustar parámetros en la fase de entrenamiento. El conjunto de prueba, que contiene datos completamente no vistos, se usa para la evaluación final del modelo después de la selección de características y ajuste de parámetros [7].

Validación cruzada

La validación cruzada es una técnica que permite evaluar el rendimiento del modelo de manera más robusta.

- ***K-Fold Cross-Validation***: En la validación cruzada *K-Fold*, el conjunto de datos se divide en k pliegues (*folds*). El modelo se entrena k veces, cada vez utilizando $k - 1$ pliegues para el entrenamiento y el pliegue restante para la prueba. El rendimiento final del modelo se obtiene promediando los resultados de las k iteraciones.
- ***Stratified K-Fold Cross-Validation***: Este método es similar a *K-Fold*, pero asegura que cada pliegue tenga una distribución similar de la variable objetivo. Esto es especialmente útil en conjuntos de datos con desequilibrio de clases.

Bootstrap

El *Bootstrap* es una técnica de remuestreo que estabiliza el modelo. Implica seleccionar un tamaño de muestra (generalmente igual al tamaño del conjunto de datos original), y luego seleccionar aleatoriamente puntos de datos con reemplazo para crear la muestra bootstrap. El modelo se entrena en esta muestra y se evalúa en los puntos de datos que no se incluyeron en la muestra (conocidos como muestras fuera de bolsa). Esto permite obtener una estimación robusta del rendimiento del modelo y ayuda a mitigar el sobreajuste. Esta técnica se utiliza dentro de los ensembles como ya se ha comentado en la sección anterior.

Matriz de Confusión

Para cada predicción de un modelo de clasificación, se puede construir una matriz de confusión que demuestra el número de casos de prueba correctamente e incorrectamente clasificados.

La matriz de confusión se ve de la siguiente manera (considerando que 1 es Positivo y 0 es Negativo para las clases objetivo):

	Actual 0	Actual 1
Predicted 0	True Negatives (TN)	False Negatives (FN)
Predicted 1	False Positives (FP)	True Positives (TP)

Tabla 3.2: Matriz de Confusión

- **TN**: Número de casos negativos clasificados correctamente.
- **TP**: Número de casos positivos clasificados correctamente.
- **FN**: Número de casos positivos incorrectamente clasificados como negativos.
- **FP**: Número de casos negativos incorrectamente clasificados como positivos.

Exactitud (Accuracy)

La exactitud es la métrica más simple y se puede definir como el número de casos de prueba correctamente clasificados dividido por el número total de casos de prueba.

$$\text{Exactitud} = \frac{TP + TN}{TP + TN + FP + FN}$$

Sin embargo, no es muy útil para conjuntos de datos desequilibrados. Por ejemplo, en la detección de fraudes, si la proporción de fraude a no fraude es 1:99, la exactitud no sería un buen indicador del rendimiento del modelo.

Precisión (Precision)

La precisión es la métrica utilizada para identificar la corrección de la clasificación positiva.

$$\text{Precisión} = \frac{TP}{TP + FP}$$

Una mayor precisión significa una mejor capacidad del modelo para clasificar correctamente la clase positiva.

Recall

El recall nos dice el número de casos positivos correctamente identificados del número total de casos positivos.

$$\text{Recall} = \frac{TP}{TP + FN}$$

Un alto valor de recall indica que se identificaron muchos casos de fraude del total de fraudes.

F1 Score

El F1 Score es la media armónica del Recall y la Precisión, y equilibra las fortalezas de cada una.

$$F1 = 2 \cdot \frac{\text{Precisión} \cdot \text{Recall}}{\text{Precisión} + \text{Recall}}$$

Es útil en casos donde tanto el recall como la precisión son valiosos.

AUC-ROC

La curva ROC (*Receiver Operating Characteristics*) es una gráfica de la tasa de verdaderos positivos (recall) contra la tasa de falsos positivos. El área bajo la curva (AUC-ROC) mide el rendimiento del modelo; cuanto mayor es el área, mejor es el rendimiento del modelo. Si la curva está cerca del eje diagonal (45 grados), sugiere que el modelo está prediciendo de manera aleatoria.

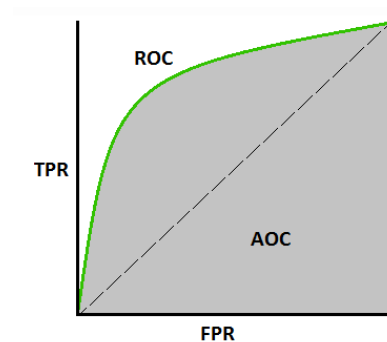


Figura 3.8: Curva AUC-ROC.

4. Técnicas y herramientas

En este apartado se tratará de presentar las técnicas llevadas a cabo para desarrollar el proyecto y también las herramientas utilizadas durante todo el proceso.

4.1. Técnicas

En el ámbito del desarrollo de software, la selección adecuada de técnicas es fundamental para la eficacia y la sostenibilidad del proyecto. Este apartado se enfoca en las técnicas específicas utilizadas en este trabajo.

SCRUM

Para explicar esta sección se utilizará el manual de la certificación oficial Scrum Master de Scrum Manager [16].

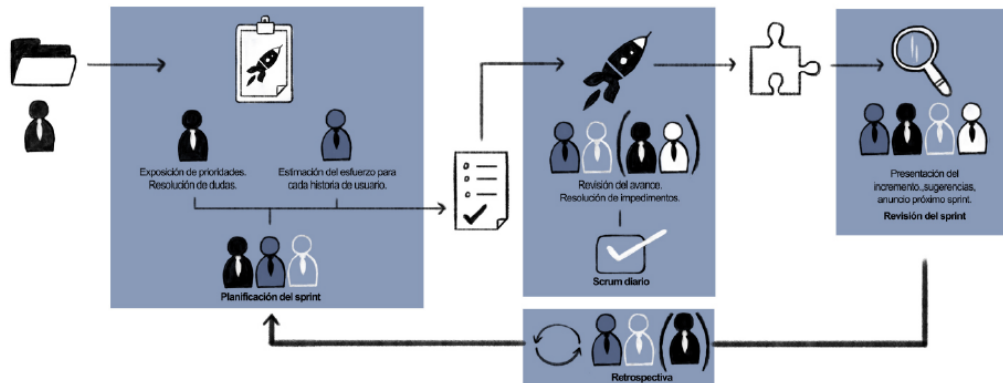


Figura 4.1: Procedimiento en la metodología SCRUM [16]

La metodología ágil Scrum se diferencia de las metodologías clásicas en su enfoque iterativo y flexible hacia la gestión de proyectos. Mientras que las metodologías clásicas, como el modelo en cascada, siguen un enfoque secuencial y predeterminado, Scrum promueve la adaptabilidad y la colaboración continua. Facilita respuestas rápidas a los cambios a través de ciclos cortos de desarrollo llamados Sprints, permitiendo a los equipos evaluar el progreso y ajustar el rumbo con frecuencia.

Roles

- **Desarrolladores:** Encargados de crear el producto, los desarrolladores son fundamentales para la ejecución técnica del proyecto, aportando habilidades específicas para alcanzar los objetivos del Sprint.
- **Propietario del producto (*Product Owner*):** Define el alcance y las prioridades del proyecto, manteniendo la pila del producto actualizada para reflejar las necesidades del negocio, asegurando que el trabajo del equipo de desarrollo aporte el máximo valor.
- ***Scrum Master*:** Facilitador y guía del equipo Scrum, el Scrum Master ayuda a implementar Scrum, asegurándose de que se sigan las prácticas y procesos, y trabaja para eliminar obstáculos que puedan impedir el progreso del equipo.

Artefactos

- **Pila del producto**(*Product Backlog*): Lista ordenada de todo lo necesario para el producto, gestionada por el *Product Owner*, que establece los requisitos y prioridades.
- **Pila del sprint**(*Sprint Backlog*): Conjunto de elementos seleccionados de la pila del producto para ser desarrollados durante el sprint, junto con un plan para entregar el incremento del producto y lograr el objetivo del Sprint.
- **Incremento**: La suma de todos los elementos de la pila del producto completados durante un sprint y todos los sprints anteriores, que cumple con los criterios de aceptación y asegura que el producto es potencialmente entregable.

Eventos

- **Sprint**: un periodo fijo durante el cual se crea un incremento del producto potencialmente entregable. Suele ser entre una y cuatro semanas.
- **Planificación del sprint**: sesión al inicio del sprint donde el equipo selecciona trabajo de la pila del producto para completar durante el sprint.
- **Reunión diaria**: breve reunión diaria para sincronizar actividades y crear un plan para el próximo día, facilitando la colaboración.
- **Revisión del sprint**: al final del sprint, el equipo presenta el incremento a los interesados, recopilando retroalimentación para futuras iteraciones.
- **Retrospectiva del sprint**: oportunidad para el equipo scrum de inspeccionarse a sí mismo y crear un plan de mejoras para el próximo sprint.

Optimización de Rendimiento de la Web

En el desarrollo web, la optimización del rendimiento es crucial para asegurar una experiencia de usuario rápida y eficiente. Mejorar el rendimiento de una web implica una serie de técnicas y prácticas que reducen el tiempo de carga, optimizan la entrega de contenido y mejoran la interacción del

usuario con el sitio. Una técnica esencial en este proceso es la auditoría de rendimiento de la web.

Auditoría de Rendimiento de la Web

La auditoría de rendimiento es un proceso sistemático de evaluación de una página web para identificar y resolver problemas que puedan afectar la velocidad y eficiencia del sitio. Hoy en día existen varias herramientas que permiten esto, siendo ***Lighthouse*** una de las más conocidas. Por ello se utilizará su guía para explicar el concepto de auditoría de web [10].

¿Qué se audita en una web?

1. Tiempo de carga:

- ***First Contentful Paint (FCP)***: mide el tiempo que transcurre desde que el usuario navega a la página por primera vez hasta que se renderiza en la pantalla cualquier parte del contenido de la página.
- ***Largest Contentful Paint (LCP)***: Tiempo hasta que se renderiza el contenido más grande visible en la ventana gráfica.
- ***Time to Interactive***: Tiempo hasta que la página es completamente interactiva.

2. Recursos estáticos:

- **Minificación de CSS y JavaScript**: Reducción del tamaño de los archivos eliminando espacios, comentarios y otros caracteres innecesarios.
- ***Lazy Loading***: Carga diferida de imágenes y otros recursos solo cuando son necesarios.

3. Optimización de imágenes:

- **Formato de imágenes**: Uso de formatos eficientes como WebP o AVIF.
- **Dimensiones correctas**: Asegurar que las imágenes no sean más grandes de lo necesario.

4. Reducción de peticiones HTTP:

- **Combinar archivos**: Agrupación de archivos CSS y JavaScript para reducir el número de peticiones.

- **Uso de *sprites*:** Combinar varias imágenes pequeñas en una sola imagen para reducir peticiones.

Resultados de la Auditoría

Una auditoría de rendimiento genera un informe detallado con las siguientes secciones:

- **Diagnósticos:** Información específica sobre problemas encontrados y cómo afectan el rendimiento.
- **Oportunidades:** Recomendaciones sobre cómo mejorar el rendimiento, incluyendo estimaciones de cuánto mejorará cada cambio sugerido.
- **Métricas:** Resultados cuantitativos que miden el rendimiento actual de la web, como el tiempo de carga de diferentes partes del contenido.

Estas auditorías proporcionan una hoja de ruta clara para optimizar el rendimiento de la web, ayudando a priorizar las acciones necesarias para mejorar la velocidad y eficiencia del sitio.

PEP 8

En el desarrollo de este proyecto, se ha seguido la guía de estilo **PEP8** para el lenguaje de programación Python, [22]. *PEP8* es un documento que proporciona convenciones para el código Python. El cumplimiento de estas convenciones es crucial para garantizar una base de código coherente, legible y eficiente. Entre sus características, esta guía de estilo cobra importancia en:

- **Legibilidad:** La claridad y la simplicidad del código se priorizan, haciendo que sea más fácil de leer y entender para cualquier desarrollador que lo lea.
- **Consistencia:** Seguir una guía de estilo común promueve la uniformidad en la base de código, lo que facilita la colaboración entre múltiples desarrolladores (pensando en posibles modificaciones futuras).
- **Mantenibilidad:** Un código bien estructurado y formateado según *PEP8* es más sencillo de mantener, depurar y ampliar.

Los aspectos clave de PEP8 adoptados en el proyecto son:

- **Formato de Código:** Se ha prestado especial atención al uso adecuado de espacios en blanco, sangrías y alineaciones para mejorar la legibilidad. Manteniendo un límite de línea máximo de 80.
- **Convenciones de Nomenclatura:** Las variables, funciones, clases y módulos se nombran siguiendo las recomendaciones de *PEP8*, lo que facilita la comprensión de su propósito y alcance.
- **Guía de Importaciones:** Los módulos se importan de una manera ordenada y coherente, evitando conflictos y facilitando la identificación de dependencias.
- **Comentarios y *Docstrings*:** Se utilizan comentarios y cadenas de documentación para explicar el propósito y el funcionamiento de todos los bloques de código, mejorando así su comprensibilidad.

Principios SOLID

Se ha adoptado los principios SOLID como fundamentos clave para un diseño de software eficiente y mantenible. Estos principios orientan la estructura del código, asegurando que sea flexible ante cambios y extensiones futuras.

- **S (Responsabilidad Única):** Cada componente tiene un solo propósito, facilitando las pruebas y minimizando las dependencias.
- **O (Abierto/Cerrado):** El código está preparado para la expansión sin necesidad de modificar lo existente, reduciendo los errores al introducir nuevas funcionalidades.
- **L (Sustitución de Liskov):** Las clases derivadas pueden sustituir a sus clases base sin alterar el comportamiento esperado, garantizando la consistencia del sistema.
- **I (Segregación de Interfaces):** Se utilizan interfaces específicas para evitar la implementación de métodos innecesarios, promoviendo un código más limpio y modular.
- **D (Inversión de Dependencias):** se debe depender de abstracciones en lugar de implementaciones concretas, lo que disminuye el acoplamiento y mejora la testabilidad.

4.2. Herramientas

En este proyecto, se ha empleado una variedad de herramientas esenciales que han facilitado un desarrollo eficiente y efectivo. A continuación, se detallan las herramientas clave utilizadas y cómo han ayudado en los resultados finales.

Programas

En esta sección se comentarán todos aquellos programas que se han utilizado durante el desarrollo, ya sea de código, documentación o planificación. Se incluyen aplicaciones tanto de escritorio como web.

Visual Studio Code

Visual Studio Code, ha sido el IDE principal utilizado en el proyecto, abarcando tanto el desarrollo web como la creación de algoritmos. Su amplia gama de extensiones lo convierte en una herramienta extremadamente versátil y potente, capaz de adaptarse a diversas necesidades de programación. Las funcionalidades como el resaltado de sintaxis, la depuración integrada, el control de versiones Git, la personalización a través de extensiones, como soporte para diferentes lenguajes de programación y herramientas de desarrollo, y la posibilidad de uso de \LaTeX , han sido las claves para que sea el entorno de programación elegido.

TeXstudio

Para la documentación del proyecto, se ha utilizado TeXstudio, un editor especializado en \LaTeX . Su interfaz intuitiva y las características como la vista previa en tiempo real, la comprobación ortográfica, el resaltado de sintaxis y los atajos de teclado para una escritura más rápida, hacen que sea una aplicación fácil de usar para principantes en el lenguaje.

Git (GitHub)

El control de versiones y la planificación del proyecto se han gestionado a través de Git, utilizando GitHub como plataforma de hospedaje para el código fuente.

Taiga

Taiga se ha utilizado para la planificación de sprints y la gestión ágil del proyecto. Esta herramienta permite organizar tareas, priorizar actividades y seguir el progreso de forma clara y estructurada. Quizá no sea la que más funcionalidades de, pero al tratarse de un único desarrollador, cumple con lo que se buscaba.

Zapier

Zapier se ha utilizada para conectar aplicaciones como GitHub y Taiga para agilizar el proceso de planificación. Mediante la creación de «*triggers*» (desencadenantes), se puede crear un *issue* en GitHub y que aparezca automáticamente en Taiga o viceversa, facilitando la creación de tareas, unificando su creación en un único lugar.

Excalidraw

Excalidraw ha sido utilizado para crear dibujos y diagramas de forma amena y sencilla. Esta herramienta web ofrece la capacidad de diseñar tanto diagramas que parecen hechos a mano como figuras más formales.

w3schools

Para la resolución de dudas específicas y el aprendizaje de nuevas tecnologías, se ha recurrido a menudo a w3schools. Esta plataforma en línea ha sido una fuente de tutoriales, ejemplos de código y referencias, facilitando el rápido entendimiento de nuevas técnicas de desarrollo web.

Lighthouse

Lighthouse es una herramienta automatizada de Google que ayuda a mejorar la calidad de las páginas web mediante auditorías detalladas [10]. Se puede ejecutar en cualquier página web, pública o que requiera autenticación.

Esta herramienta ofrece informes con sugerencias de mejora en áreas clave como:

- **Rendimiento:** Mide tiempos de carga y ofrece recomendaciones para mejorar la velocidad de la página.
- **Informe de accesibilidad:** Evalúa la accesibilidad de la página para usuarios con discapacidades y proporciona sugerencias para mejorarla.

- **Informe de prácticas recomendadas:** Verifica que la página siga las mejores prácticas de desarrollo web, incluyendo seguridad y estándares de codificación.
- **Informe de SEO:** Analiza la optimización de la página para motores de búsqueda y sugiere mejoras para aumentar la visibilidad.

Al ser una herramienta de Google, viene integrada en *Chrome DevTools*, pero también se puede usar como extensión del navegador o desde línea de comandos. Los resultados de las métricas anteriores las muestra con una puntuación de 0 a 100, como se puede ver en la imagen 4.2. Además, deja elegir entre ejecutar la auditoría en un dispositivo móvil o en un ordenador de escritorio, contextos que debe cubrir la web objetivo de este trabajo.



Figura 4.2: Resultado de ejecutar Lighthouse en una página web.

En la figura 4.2 se ve como cada área estudiada tiene sus propias métricas (algunas vistas en el apartado 4.1), las cuales también muestra con tiempos reales. Otro apartado relevante son los diagnósticos, donde muestra consejos para mejorar el rendimiento y además cuantifica esa mejora en unidades, como se puede ver en la figura 4.3.

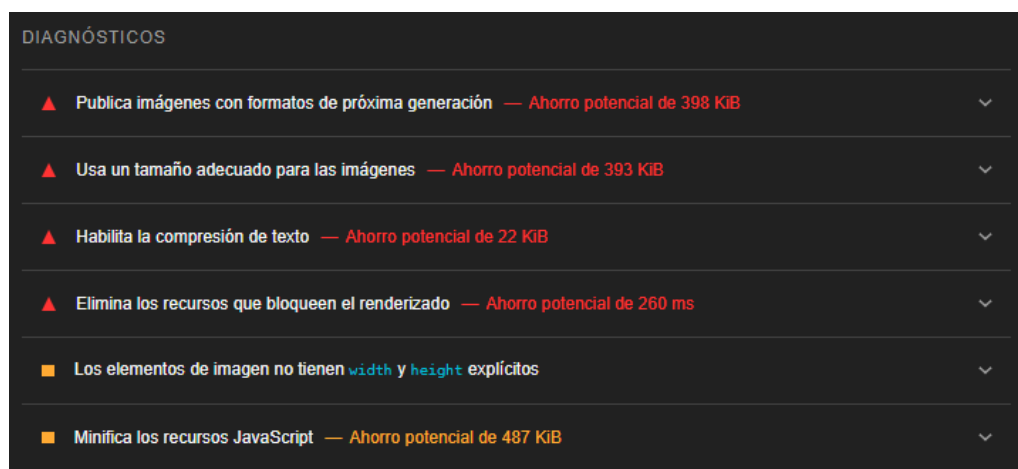


Figura 4.3: Resultado de diagnósticos al ejecutar Lighthouse en una página web.

IA Generativa

El uso de herramientas de IA generativa como ChatGPT, GitHub Copilot y DALL·E en proyectos de desarrollo como el de este proyecto aumenta la productividad. Permiten encontrar, reducir y solucionar errores de manera rápida, mejorar la calidad del trabajo y evitar tareas tediosas y poco significativas como la creación de imágenes.

Bibliotecas

El desarrollo de este proyecto se ha apoyado en una serie de bibliotecas esenciales tanto para el *backend* como para el *frontend*, proporcionando una amplia gama de funcionalidades, desde el desarrollo web hasta el análisis de datos. A continuación, se detalla una lista de las principales bibliotecas empleadas, mayormente heredadas del anterior trabajo:

- **Babel:** Utilizada para las traducciones en la web, permite llevar la internalización de manera ágil, generando todos aquellos textos nuevos y compilándolos para mostrarse en la web. La propia acción de traducir la lleva a cabo el desarrollador.
- **Flask:** Framework de Python para el desarrollo de aplicaciones web. Incluye herramientas y características básicas como enrutamiento, gestión de solicitudes y respuestas, y soporte para plantillas HTML.

- **Mypy**: Herramienta de verificación de tipos estáticos para Python, permite un desarrollo más seguro al detectar incompatibilidades de tipo.
- **Pylint**: Analiza el código Python en busca de errores de estilo, listándolos para ayudar al usuario. Permite un código más limpio y estándar. Ayuda a seguir la guía de estilo *PEP8* antes comentada.
- **SQLAlchemy**: SQLAlchemy es una biblioteca de mapeo objeto-relacional (ORM) para Python que facilita la interacción con bases de datos relacionales mediante el uso de objetos de Python en lugar de consultas SQL directas. Proporciona una capa de abstracción que permite a los desarrolladores trabajar con modelos de datos en Python, simplificando operaciones de creación, lectura, actualización y eliminación (CRUD).
- **Scikit-learn**: Biblioteca de aprendizaje automático para Python, incluye una amplia variedad de algoritmos y herramientas para tareas como clasificación, regresión, clustering y reducción de dimensionalidad entre otras. Usada para la construcción y evaluación de los modelos predictivos.
- **Numpy**: Biblioteca que proporciona soporte para arrays y matrices grandes y multidimensionales, junto con una colección de funciones matemáticas para operar con estos objetos. Se utiliza como base para otras bibliotecas como scikit-learn o Pandas.
- **Pandas**: Es una biblioteca poderosa y flexible para el análisis de datos en Python. Ofrece estructuras de datos de alto rendimiento, como DataFrames, que permiten la manipulación y análisis de datos de manera eficiente y expresiva.
- **Bootstrap**: Es un framework de desarrollo web diseñado para facilitar la creación de sitios y aplicaciones web «responsive», es decir, que se adapten de manera eficiente a diferentes tamaños de pantalla y dispositivos. Bootstrap incluye CSS, componentes de JavaScript o incluso iconos, lo que permite a los desarrolladores construir interfaces de usuario coherentes y atractivas con una mínima cantidad de código personalizado.

Bootstrap ofrece una gran cantidad de componentes predefinidos que aceleran el proceso de desarrollo, como: *Cards*, contenedores flexibles y

extensibles para mostrar contenido variado o *Navbars*, barras de navegación personalizables que facilitan la creación de menús y cabeceras de sitios web.

Cuenta con una amplia documentación en la web (<https://getbootstrap.com/docs/5.3/getting-started/introduction/>), que permite a los usuarios aprender de manera rápida y efectiva.

- **D3.js:** (*Data-Driven Documents*) es una biblioteca de JavaScript utilizada para crear visualizaciones de datos dinámicas e interactivas en navegadores web.

Uno de los usos más populares y que se usa en este trabajo es la construcción de grafos de fuerza (*force-directed graphs*), que son una forma de visualización utilizada para mostrar relaciones entre entidades. Estos gráficos utilizan simulaciones físicas para posicionar los nodos de manera que las conexiones entre ellos sean claras y comprensibles. Las simulaciones de fuerza en D3.js permiten ajustar diversas fuerzas, como la atracción entre nodos conectados y la repulsión entre nodos no conectados, para lograr una disposición equilibrada y legible del grafo.

D3.js también es ampliamente utilizada para crear otros tipos de visualizaciones de datos, como gráficos de barras, líneas, áreas, etc. Y todos ellos se encuentran en el sitio web <https://observablehq.com/@d3/gallery>.

5. Aspectos relevantes del desarrollo del proyecto

En la sección que sigue, se abordan los aspectos más significativos que han marcado el desarrollo de este proyecto. Se detalla cómo cada elección ha influido en la trayectoria y los resultados del proyecto.

5.1. Lectura artículos científicos

Se ha decidido incluir la lectura de artículos científicos en aspectos relevantes ya que son una parte significativa de todo el trabajo, llevando bastante tiempo en aquellos que son más técnicos. Además proporcionan una base sólida de teorías y métodos que pueden ser utilizadas tanto para el trabajo como para otras situaciones. Todos ellos están en inglés por lo que también se mejora esta habilidad, notando cierta mejora de comprensión en los últimos artículos. Un artículo científico es un informe escrito y publicado en una revista con cierto prestigio que describe resultados originales de investigación. Estos artículos son fundamentales para la difusión del conocimiento científico y suelen seguir un formato estructurado que incluye una introducción, metodología, resultados, discusión y conclusiones.

Para este trabajo se han leído 2 tipos de artículos, los *surveys* o resúmenes sobre un tema y los artículos de implementación de algoritmos:

- ***A Survey on Semi-Supervised Learning*** [21]: Este artículo proporciona una revisión exhaustiva del campo del aprendizaje semi-supervisado. Gracias a este artículo se consigue una base sólida en este

tipo de aprendizaje y se comprenden cuales son los desafios actuales y las direcciones futuras en la investigación en este campo.

- ***Ensemble Based Systems in Decision Making*** [17]: Este artículo revisa el uso de sistemas basados en *ensembles* para la toma de decisiones. Estos sistemas combinan múltiples clasificadores para mejorar la precisión y la robustez del proceso de decisión. Destaca cómo los sistemas de ensamblaje pueden superar las limitaciones de los clasificadores individuales y proporcionar decisiones más confiables y precisas.
- ***Improve Computer-Aided Diagnosis With Machine Learning Techniques Using Undiagnosed Samples*** [13]: Los autores presentan el método Co-Forest, una técnica semi-supervisada que combina múltiples clasificadores de árboles de decisión para mejorar la precisión del diagnóstico. Se discuten los beneficios de incorporar datos no etiquetados en el proceso de entrenamiento y se presentan resultados experimentales que demuestran la eficacia de esta metodología en varias aplicaciones médicas.
- ***Graph Construction Based on Labeled Instances for Semi-supervised Learning*** [4]: Los autores presentan un método para la construcción de grafos basado en instancias etiquetadas para el aprendizaje semi-supervisado. La idea principal es utilizar la información de las instancias etiquetadas para guiar la construcción del grafo.
- ***Learning with Local and Global Consistency*** [24]: Este artículo introduce un algoritmo para el aprendizaje con consistencia local y global. Los autores describen cómo el algoritmo itera para ajustar las etiquetas de las instancias no etiquetadas, combinando la información de las instancias vecinas y la información inicial. Se demuestra que este enfoque es eficaz para tareas de clasificación semi-supervisada.

5.2. Trabajo Preexistente

La elección de este trabajo se realiza por el gran interés en la inteligencia artificial y el aprendizaje automático, pero también por el hecho de que ya existía un trabajo realizado por otro alumno un año atrás (David Martínez Acha – <https://vass.dmachacha.dev/>). Esto ayudaría mucho en el desarrollo ya que serviría como referencia para muchas dudas.

El plan original era hacer mi propia página web educativa desde cero, pero mostrando otra serie de algoritmos semisupervisados (ensembles y grafos) en lugar de los ya existentes. Con el desarrollo del primer algoritmo surge la idea por parte del tutor de basar el proyecto en esta otra página desarrollada por David Martínez. De esta manera, el tiempo que hubiera empleado en aprender e implementar la web desde cero, se emplea en comprender todo el código programado por David, aprovechando también la gestión de cuentas de usuarios. Aún así, se deja total libertad para cambiar e implementar lo que haga falta para mejorar el proyecto original.

El tiempo empleado en familiarizarse con el nuevo código fue de dos sprints, ya que no solo trataba de leer código, sino de comprender las técnicas de HTML, css y javascript que se utilizan, junto con bibliotecas como *Bootstrap*. Aún así, el tiempo ganado es considerable y da pie a poder implementar más algoritmos y pensar en ideas que mejoran la web. Inicialmente se piensa que con un fork a su repositorio de GitHub, [2], se puede trabajar mejor, pero de esta manera se perderían las tareas y commits hechos hasta la fecha en el repositorio de este proyecto. Por esto se decide descargar el contenido y copiarlo a el proyecto ya en desarrollo.

La documentación de David [1], sirve de gran ayuda y también se heredan partes de ella, como los trabajos relacionados y conceptos teóricos. También se tiene en cuenta las líneas de trabajo futuras desarrolladas para poder implementarlas en este trabajo.

Cambios en la web

Gran parte de la web se reutiliza, pero a medida que se desarrolla el proyecto surgen nuevas ideas que modifican parte del comportamiento de la web que ya existía. Esta sección servirá para remarcar esas pequeñas o grandes modificaciones que sufre el diseño inicial y cual es la idea de esta nueva funcionalidad. La primera idea de cambio que surge es la de modificar la funcionalidad de configuración de un algoritmo. Visualizando páginas web, se da con una página que muestra algoritmos de aprendizaje automático [6], pero no de la misma manera que David. En esta página se permite configurar y ver resultados y estadísticas en una única ventana, con la gran diferencia de que no muestra el paso a paso en cada algoritmo, sino directamente el resultado final de la clasificación. El primer prototipo se hace pensando en esta idea, quedando algo parecido a la figura 5.1:

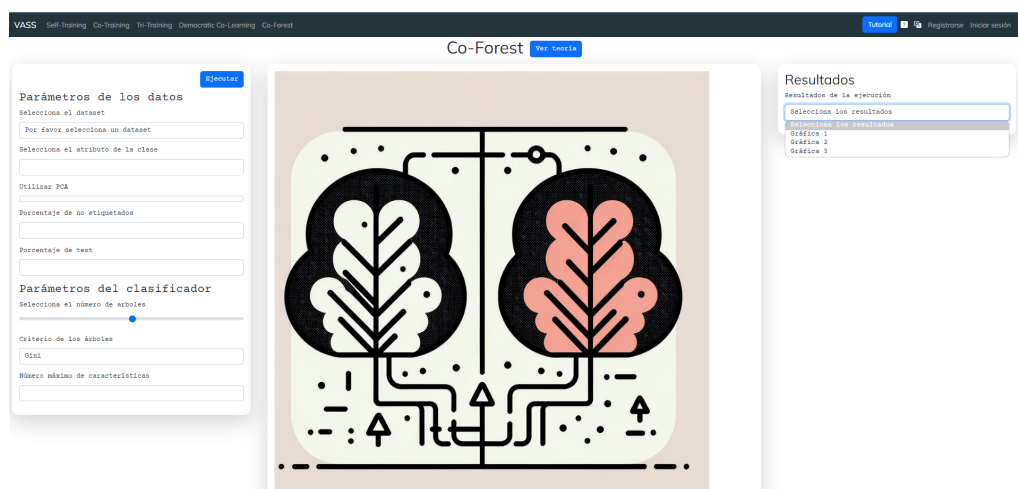


Figura 5.1: Prototipo de visualización de algoritmo Co-Forest

Posteriormente, gracias al tutor nos damos cuenta de que si la idea es mostrar la ejecución paso a paso, que los parámetros de configuración estén en la misma pantalla, no es de gran ayuda. Por esto, se decide volver a la versión del trabajo base y modificar las plantillas necesarias para adaptarlas a los nuevos algoritmos.

Configuración de parámetros: Cuando tratamos con archivos de datos preparados para estos algoritmos, siempre se suele dar el caso de que la clase o etiqueta suele ser la última columna. Por esto, se ha mantenido la opción de seleccionar el atributo deseado, pero saldrá por defecto la clase en la caja de entrada. Esto además permite mayor agilidad a la hora de hacer pruebas de visualización ya que no hay que gastar tiempo en seleccionar esta opción.

Gráfica de estadísticas específicas: el cambio que se ha realizado en esta parte ha sido la opción de marcar o desmarcar todos los clasificadores para ver su traza en la gráfica de estadísticas. Sirve sobretodo en el caso del Co-Forest por el hecho de que puede haber gran cantidad de árboles, y querer ver uno concreto con la configuración anterior llevaba una pérdida de tiempo innecesaria desmarcando uno por uno el resto de clasificadores.

Utilizar fichero por defecto: en la versión anterior se dedica una ventana entera a la selección del archivo, permitiendo descargar varios ficheros de prueba para luego subirlos. La idea aquí es agilizar el proceso de utilizar estos ficheros de prueba, aunque manteniendo la opción de

descarga, permitiendo que al pulsar en un dataset de prueba pase a la fase de configuración directamente, sin tener que cargarlo desde las descargas.

5.3. Algoritmos

En esta sección se comentarán los aspectos más relevantes en la implementación de los algoritmos semisupervisados.

Co-Forest

En la implementación de este algoritmo de nuevo se parte con una gran ventaja. El año anterior otra alumna había implementado el mismo algoritmo para otro proyecto. Dentro de esta aplicación, Patricia Hernando, hizo sus propios estudios, los cuales se han aprovechado en este trabajo. Basado fuertemente en el pseudocódigo del artículo [13], la implementación se ve alterada por el parámetro W , el cual establece la confianza en las muestras para ser seleccionada o no. Resumiendo el estudio de Patricia, como se puede ver en el pseudocódigo del artículo, hay una ecuación donde el valor de W esta dividiendo. Esto es un problema ya que según establece el algoritmo puede llegar a valer cero, provocando así una indeterminación. Uno de los estudios de Patricia determina que una de las mejores soluciones a esto es iniciar el parámetro W al mínimo entre 100 y el 10% de la cantidad de muestras etiquetadas que hay. Como se puede ver en el pseudocódigo de la sección tres, esto se aprovecha, evitando así posibles problemas. Para determinar si el algoritmo definitivo es bueno, se compara con el de Patricia, evaluando como varía el valor de *accuracy* en cada iteración del algoritmo. Los resultados se muestran en la figura 5.2

Grafos

En este apartado se comentarán todos los aspectos relevantes relacionados con la implementación de los algoritmos basados en grafos. Desde sus posibles interpretaciones y modificaciones con respecto al código original, a los diferentes estudios de comparación.

Comparativa de bibliotecas de grafos

Cuando se quiere implementar un algoritmo basado en grafos, lo ideal es utilizar una biblioteca que ayude a automatizar y mejorar el código. En *Python*, existen varias bibliotecas que ayudan en esta tarea, tres de ellas son:

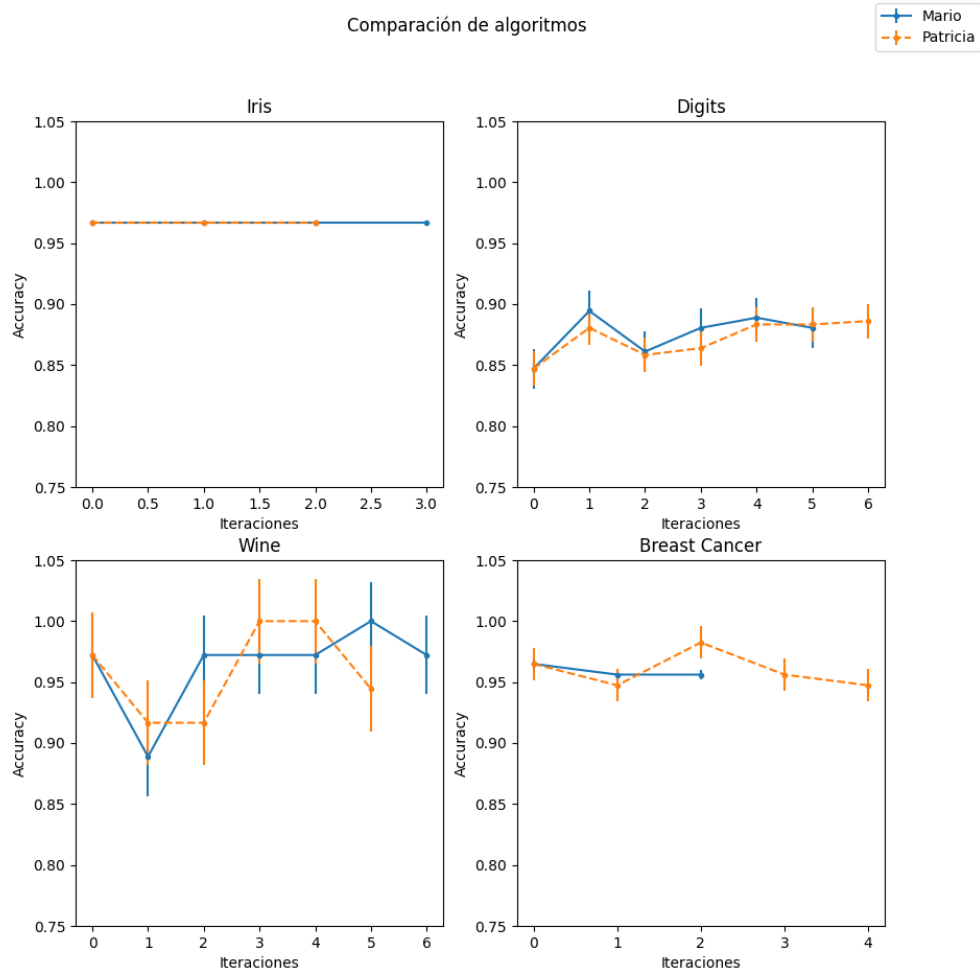


Figura 5.2: Comparación algoritmo CoForest utilizando diversos datasets

NetworkX, **igraph** y **graph-tool**. En esta sección se resumirá el estudio realizado para elegir la opción que mejor se adapte a las especificaciones.

Todas ellas ayudan en la construcción de grafos, pero para empezar es necesario dejar claro para qué se va a utilizar esta biblioteca. En cuanto al tamaño de los grafos, no necesariamente se necesita algo que maneje grafos muy grandes (más de 10000 nodos) de manera efectiva. La mayoría de *datasets* utilizados tendrán muchas menos instancias. En cuanto a la velocidad, se busca algo que sea efectivo pero sin necesidad de buscar lo mejor o más rápido, ya que los datos de entrada no van a suponer un gran esfuerzo. También hay que tener en cuenta la integración que se llevará a cabo posteriormente en la web, posiblemente con herramientas como *d3.js*.

Tras una primera búsqueda queda claro que la herramienta de *graph-tool* tiene un objetivo mucho más amplio y está pensado para proyectos con grafos grandes. De hecho es una herramienta que no se instala con *pip* sino que necesita otra instalación.

Por lo tanto, descartada una opción, se realizará una pequeña prueba para llegar a una conclusión. A continuación se muestra el pseudocódigo utilizado para la comparación de herramientas.

Algoritmo 5: *NetworkX vs igraph*

Input: Dataset de prueba L (*digits*)

Output: Grafo G en formato *JSON*

```

1  $timer \leftarrow startTimer()$ 
2  $G \leftarrow \emptyset$ 
3  $D \leftarrow pairwiseDistances(L)$  // Matriz de distancias
4 for  $i = 0$  to  $|L| - 1$  do
5     // Agregar vértices al grafo para cada muestra de  $L$ 
6      $G \leftarrow addNode(i)$ 
7 end for
8  $k \leftarrow 5$ 
9 for  $i = 0$  to  $|L| - 1$  do
10     $kNN \leftarrow getKNearestNeighbors(D[i], k)$ 
11    for  $j \in kNN$  do
12        // Añadir arista entre  $i$  y  $j$  con el peso de la distancia
13         $G \leftarrow addEdge(i, j, D[i][j])$ 
14    end for
15 end for
16  $timer \leftarrow stopTimer()$ 
17  $print(timer)$ 
18  $JSONGraph \leftarrow convertToJSON(G)$ 
19 return  $G$ 

```

Lo que se ha querido representar es el tiempo que tarda en construir el grafo y calcular los k vecinos más cercanos (kNN) para cada nodo. A su vez también se ha estudiado cuánta facilidad existe a la hora de convertir el grafo a formato *JSON*, para que pueda ser procesado después por herramientas como *d3.js*.

Los resultados obtenidos han sido los siguientes:

Ejecución 1:

NetworkX: Construction and kNN time: 0.0044 seconds

igraph: Construction and kNN time: 0.0134 seconds

Ejecución 2:

NetworkX: Construction and kNN time: 0.0020 seconds

igraph: Construction and kNN time: 0.0111 seconds

A su vez, en el uso del formato *JSON* se encuentra más útil el uso de *NetworkX* ya que incluye un método propio de exportación (*nx.readwrite.json_graph*). Por el contrario, con *igraph*, habría que construir un diccionario recorriendo los nodos y enlaces y posteriormente pasarlo a formato *JSON*.

En conclusión, ante los resultados obtenidos la idea era usar *NetworkX* para todas las fases del algoritmo, pero por simplicidad y facilidad en la implementación de los algoritmos, finalmente solo se utilizará para las ayudas en la visualización, y no como estructura principal de almacenamiento de los datos.

Modificaciones GBILI

Con respecto al algoritmo *GBILI*, surgen varias complicaciones o cuestiones de implementación que los autores no dejan claras. En este apartado se comentan las que se creen son más relevantes.

1. Visualización por pasos en la web: desde un principio se tiene claro que lo ideal es tener una visualización por pasos del grafo, localizando las principales fases y mostrando como van cambiando hasta llegar a la inferencia. En este caso, el pseudocódigo 1 muestra que hay un bucle principal en el que cada lista se va construyendo para cada nodo recorrido en este bucle externo. Para la visualización requerida esto no es lo ideal, ya que no podemos aislar las principales fases: lista de vecinos, lista de vecinos mutuos, grafo de distancias mínimas y grafo definitivo. Por todo esto, se decide seguir la misma estructura pero implementando todas estas estructuras de forma consecutiva.
2. Durante la implementación inicial del grafo, se asumió que cualquier enlace dentro del grafo sería bidireccional, asegurando así que el grafo fuera no dirigido. Sin embargo, en una de las primeras implementaciones, esta premisa no se respetó, lo que resultó en una estructura dirigida. Este error llevó a situaciones erróneas y confusas al realizar el seguimiento del código. Posteriormente, se corrigió la implementación

para garantizar que todos los enlaces sean bidireccionales, asegurando así la correcta representación de un grafo no dirigido.

3. Dificultades de interpretación del pseudocódigo: en el código del artículo pueden surgir ciertas dudas de como hace algún paso. Estas son: al hacer la búsqueda en anchura, dice que debe retornar la componente (subgrafo aislado) del grafo completo, posteriormente, esta sintaxis la utiliza para los nodos individuales. Después de consultarlo con el tutor se llega a la conclusión de que la búsqueda en anchura realmente devuelve todo el conjunto de componentes del grafo y posteriormente se accede a la que pertenece cada nodo individual. Otra sintaxis que puede llevar a confusión es la que encontramos en la línea 20. Cuando comprueba que la distancia es mínima, la suma la hace dentro de los dos bucles, lo que puede llevar a interpretación de que esa condición se debe cumplir dentro del bucle interno. En esta situación, pongamos el siguiente ejemplo: la lista actual de vecinos mutuos es 0: [1, 2], 1: [0]... y dos de los nodos etiquetados son [3, 4...]. Se accede a la lista de vecinos del nodo 0 y despues se recorre toda la lista de nodos etiquetados. Viendo esto sabríamos que obtendríamos una distancia mínima hasta un nodo etiquetado, pero siempre se estaría guardando el enlace entre los nodos vecinos, obteniendo la misma estructura que la lista de vecinos mutuos. Por ello, al implementarlo, hay que tener en cuenta que esta condición debe ir fuera del bucle interno, para poder coger de verdad la mínima distancia entre un nodo y sus vecinos.
4. Visualización con NetworkX: Para tener un primer acercamiento a una visualización y también para comprobar que el algoritmo ha seguido correctamente la influencia de los nodos etiquetados, se decide mostrar los 4 pasos gráficamente. Un aspecto importante en la implementación es que con networkX, a la hora de construir el grafo, debe estar ordenado de la misma manera que los datos de entrada al algoritmo, si no mostrará información falsa, como nodos pintados como etiquetados que no lo son.

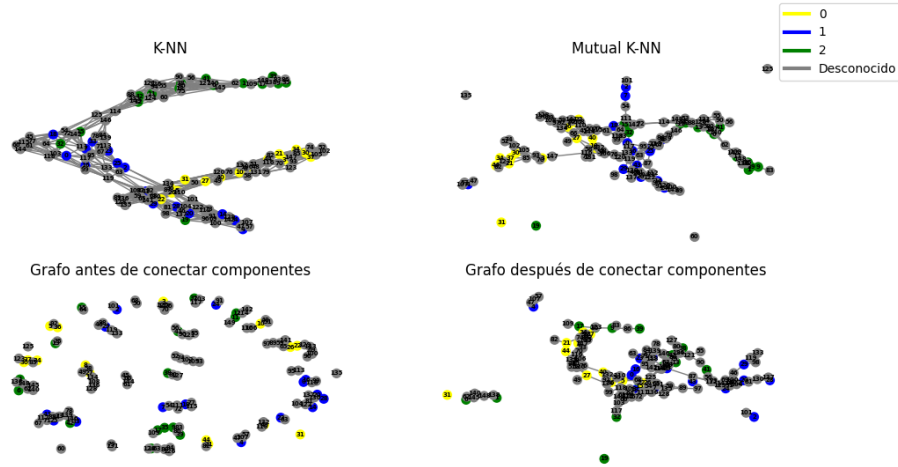


Figura 5.3: Visualización correcta de las 4 fases de construcción del grafo con el algoritmo GBILI utilizando el *dataset* de *iris data*. El color de la instancia representa su clase

LGC adaptado a grafos

En el algoritmo original de Consistencia Local y Global, la matriz de afinidad W se construye utilizando una función exponencial sobre la matriz de distancias entre los puntos. Sin embargo, dado que en este caso ya se ha construido un grafo G , utilizaremos esta información para definir la matriz de afinidad W . Específicamente, W será una matriz binaria donde $W_{ij} = 1$ si hay un enlace entre los nodos i y j en el grafo G , y $W_{ij} = 0$ de lo contrario. Esta modificación aprovecha la estructura del grafo previamente construida, además, es la misma seguida por los autores de [4]. Aunque es una opción algo «brusca», debe funcionar cuando los parámetros de los algoritmos son lo suficientemente buenos.

Este ajuste del algoritmo de Consistencia Local y Global permite aprovechar la estructura del grafo construido a partir de los puntos de datos, en lugar de basarse únicamente en la matriz de distancias. Esta modificación es especialmente útil cuando se tiene información topológica adicional, mejorando potencialmente la precisión en la inferencia de etiquetas para los puntos no etiquetados.

Un apunte importante es que cuando se estaba implementando esta modificación, se piensa que la matriz de afinidad se debe de construir aplicando la misma fórmula que antes, pero tomando como matriz de distancias las

nuevas medidas de 0s y 1s. Esto no debe ser así y fue corregido ya que si no la matriz de afinidad ya no sería esa matriz binaria que utilizan los autores del artículo original.

Si la construcción del grafo ocasiona nodos aislados (sin ninguna conexión), esto provoca que esta matriz de afinidad binaria no tenga ningún 1 conectado a este nodo. Si lo reflejamos en el pseudocódigo 3, en el paso 2, se requiere una división para realizar la inversa de la matriz D , que como indica el pseudocódigo, es una matriz diagonal que cada elemento (i, i) es igual a la suma de valores de la fila de la matriz de afinidad binaria. En el caso propuesto, esta suma sería 0, lo que ocasiona una indeterminación al calcular la división correspondiente. La solución aportada a este problema es la **regularización**, consiste en sumar a cada valor de la diagonal de la matriz donde ocurren los problemas, un valor cercano a 0, como 0.0001. De esta manera, no afecta con las siguientes operaciones ya que la influencia en los resultados es mínima, y no se producen indeterminaciones al no tener que dividir entre 0.

Para comprobar la buena funcionalidad del algoritmo *LGC* se realizará un estudio que tiene como objetivo evaluar la influencia de dos parámetros clave como son el parámetro α y la tolerancia. Para esto es necesario construir previamente el grafo, por ello se usará el algoritmo GBILI y se le dará un valor de k vecinos igual a 10, que es el valor que los autores de [4] indican con el cual a partir de él los resultados se estabilizan. La eficacia del modelo se medirá en términos de precisión (accuracy).

En una primera implementación, se quiere ver la zona de valores donde mejores resultados se obtienen. Los valores de *alpha* y *tolerance* en este caso tendrán un rango amplio para focalizarse después en una zona concreta. Después de tres ejecuciones con diferentes datasets, se pueden ver los resultados en la figura 5.4.

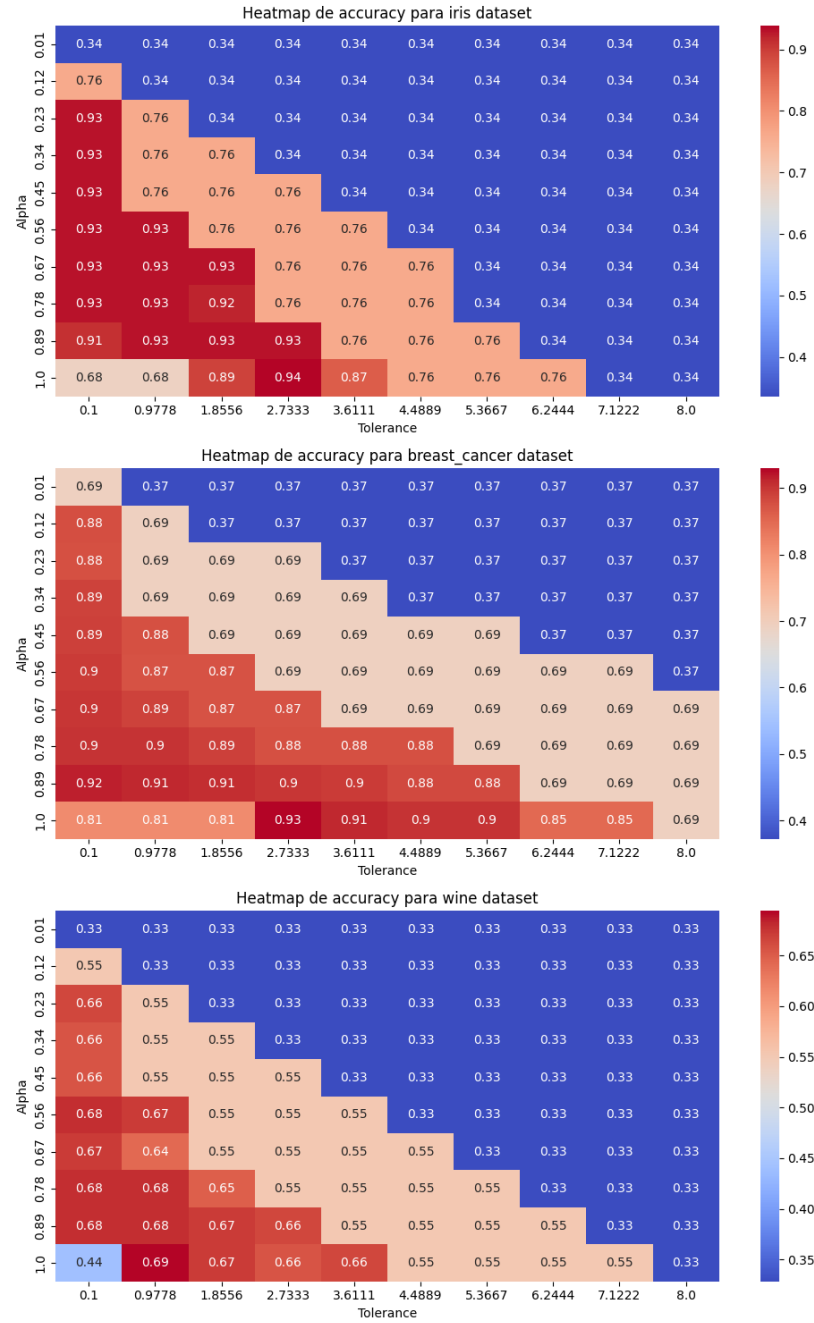


Figura 5.4: Mapa de calor que representa el *accuracy* para el algoritmo *LGC* con rangos de valores $tolerance=[0.1, 8]$ y $alpha=[0.01, 1]$. Se muestran los resultados para los *datasets*: *iris*, *breast cancer* y *wine*.

Lo que se puede observar en estas representaciones es que la tolerancia debe tener un valor más pequeño para que el algoritmo pueda realizar más de una iteración. En los tres gráficos coincide que las zonas rojas (valores más altos) se encuentran en la esquina inferior izquierda, es decir, tolerancia más pequeña y un valor de *alpha* más cercano a 1. Partiendo de la premisa que nos dan los autores en el artículo [24], en el que indican que utilizan un valor de *alpha* de 0.99 para sus ejecuciones, la evaluación de momento va por buen camino. Cuánto mayor sea el conjunto de datos de entrada, como pasa con el ejemplo de *wine*, la *accuracy* disminuye para estos parámetros de entrada, una hipótesis ante esto es porque la tolerancia está frenando antes de que la matriz F converja a valores reales. Para comprobar esto, se realizará otro estudio ahora con un rango de valores entre $\alpha=[0.90, 0.99]$ y una tolerancia= $[0.00001, 1]$. Se puede observar en la figura 5.5.

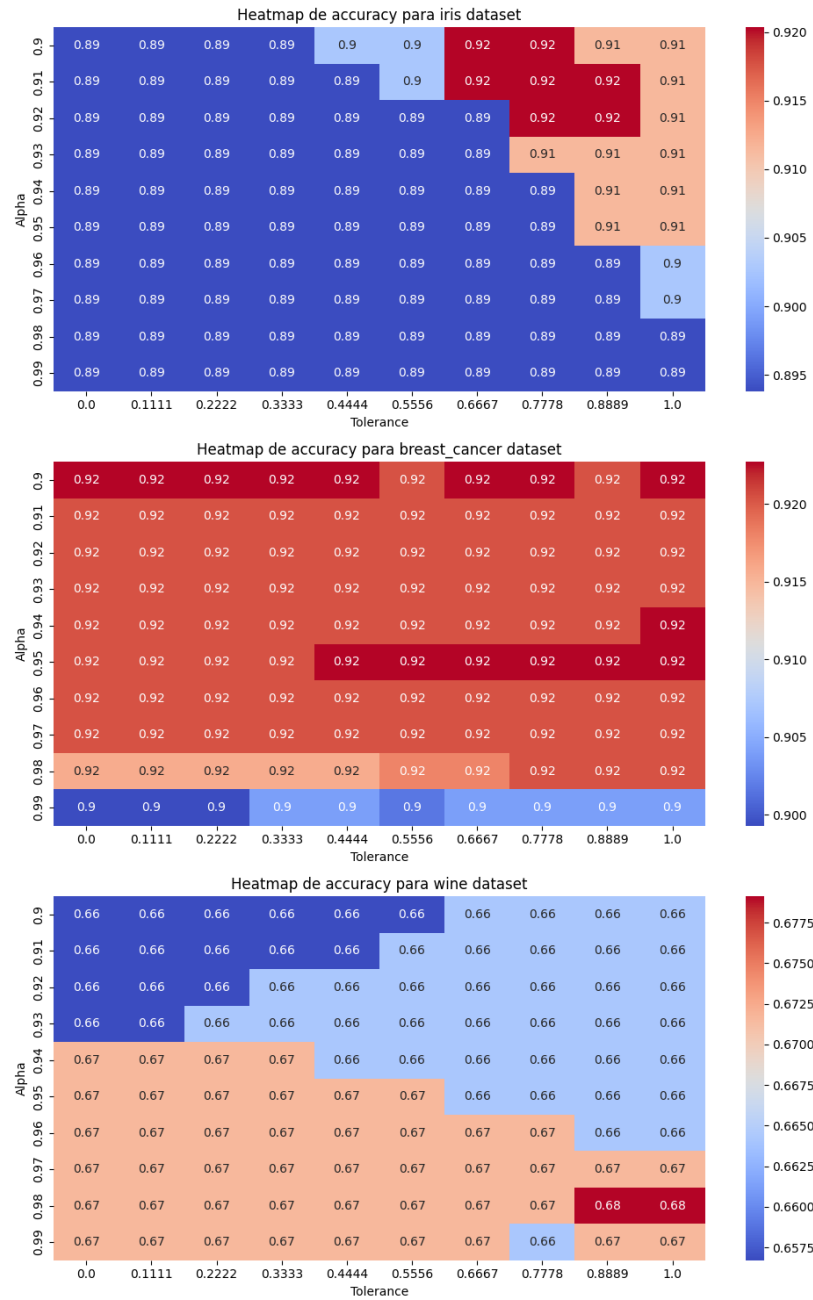


Figura 5.5: Mapa de calor que representa el *accuracy* para el algoritmo *LGC* con rangos de valores $\alpha=[0.90, 0.99]$ y $\text{tolerance}=[0.00001, 1]$. Se muestran los resultados para los *datasets*: *iris*, *breast cancer* y *wine*.

En cuanto a los dos primeros gráficos, se observa que los resultados son muy buenos a lo largo de todo el gráfico, con una *accuracy* cerca o por encima del 90 % en todos los casos. Esto indica que para estos conjuntos de datos, el algoritmo es altamente efectivo y logra clasificaciones muy precisas.

En cambio, al evaluar el último gráfico, se ve que, aun disminuyendo los valores de tolerancia, la precisión no ha conseguido mejorar como en los otros dos casos. Este comportamiento sugiere que el tercer conjunto de datos presenta un desafío mayor para el algoritmo, lo que impide alcanzar altos niveles de precisión. Es importante reflexionar que algunos conjuntos de datos (en este caso el dataset *wine*) son inherentemente más complejos que otros, ya sea por su distribución de clases, por sus atributos u otras características intrínsecas.

6. Trabajos relacionados

En esta sección, se analizarán los trabajos relacionados con el proyecto, destacando cómo cada uno de ellos ha influido y enriquecido el desarrollo. Es esencial reconocer y comprender estos trabajos, ya que ofrecen una base sobre la cual podemos construir, identifican oportunidades para la innovación y permiten evitar la repetición de errores pasados

Visualizador de Algoritmos SemiSupervisados (VASS)

Un pilar fundamental en el desarrollo de este proyecto es el trabajo realizado por David Martinez Acha. Este trabajo no solo ha sentado las bases conceptuales sino que también ha proporcionado una implementación práctica de gran valor. Se hereda directamente todo el esfuerzo de David, incluyendo sus investigaciones y los trabajos relacionados que identificó como relevantes en su estudio. Esta herencia es particularmente valiosa en lo que respecta a la implementación de la web, donde la infraestructura y el diseño preexistentes ofrecen una plataforma robusta desde la cual se puede avanzar sin partir de cero.

La implementación web desarrollada por David se destaca por su claridad, ofreciendo un punto de partida sólido para propias innovaciones. Aprovechar esta base preexistente permite enfocarse en la expansión y mejora de la funcionalidad, en lugar de invertir tiempo y recursos significativos en reconstruir lo que ya se ha logrado con éxito.

En resumen, la aplicación de David tiene 2 grandes funcionalidades separadas: los algoritmos y la gestión de usuarios. Lo que realmente importa para este proyecto es la implementación web de la parte de los algoritmos. Se tiene una pantalla principal donde permite seleccionar el algoritmo, una posterior donde se introduce el dataset, una ventana de configuración de

parámetros del algoritmo y por último la visualización de los resultados. La idea es mantener las mismas funcionalidades en el mismo orden, pudiendo modificar o extender ciertas partes gráficas que unifiquen o lo hagan más sencillo.

ML Algorithm Visualizer

Como ya se ha comentado en la sección anterior, esta página web fue importante en el planteamiento del primer diseño de la web. Se trata de una página web de visualización de algoritmos de aprendizaje automático, cuyo sitio web es <https://ml-visualizer.herokuapp.com/>. Con un diseño compacto permite seleccionar el modelo junto con la configuración de sus parámetros y unos segundos más tarde mostrará el resultado en la misma página, todo ello en una misma ventana como se puede ver en la imagen 6.1. Es aquí donde se diferencia del objetivo de este proyecto, donde se busca tener las ejecuciones separadas paso por paso.

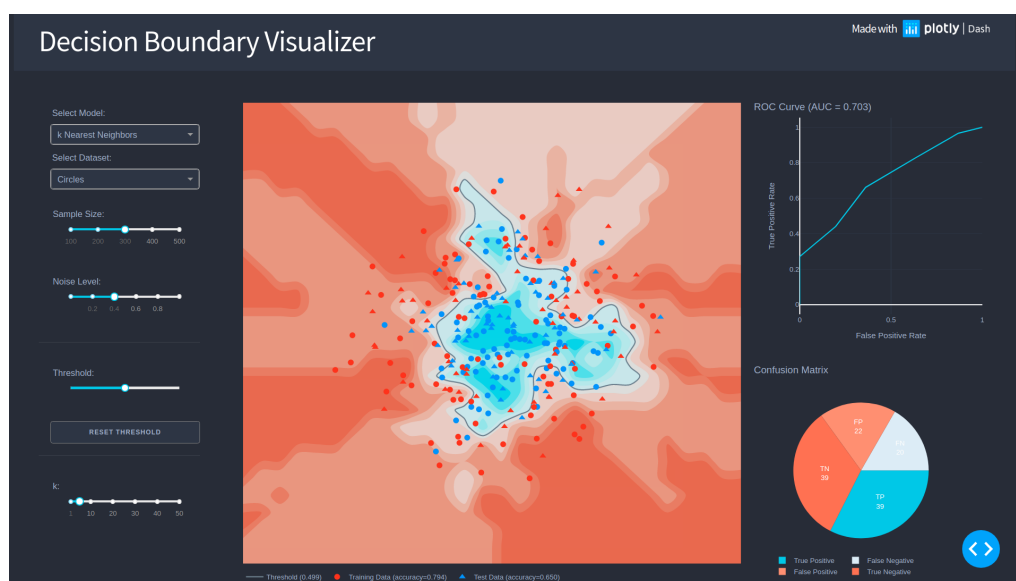


Figura 6.1: Captura de pantalla de la web *ML Algorithm Visualizer*.

Aprendizaje semisupervisado en ciberseguridad

Este trabajo corresponde con el Trabajo Fin de Grado de Patricia Hernando. Pese a ser una herramienta destinada a la ciberseguridad, tiene relación con el actual trabajo en el uso de algoritmos semisupervisados.

Permite ver la implementación y la explicación del código del Co-forest. Como ya se ha comentado, son de gran ayuda los estudios realizados para mejorar este algoritmo, ya que sirven para mostrar en la web una opción que permita seleccionar distintos valores y localizar diferencias en los resultados.

7. 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] David Martinez Acha. Herramienta docente para la visualización en web de algoritmos de aprendizaje semi-supervisado. *Universidad de Burgos*, 2023.
- [2] David Martinez Acha. Tfg-semisupervisado. <https://github.com/dmacha27/TFG-SemiSupervisado>, 2023. [Internet; descargado 20-marzo-2024].
- [3] Etienne Bernard. *Introduction to Machine Learning. Dimensionality Reduction*, chapter 7. Wolfram Media, 2021.
- [4] Lilian Berton and Alneu De Andrade Lopes. Graph construction based on labeled instances for semi-supervised learning. In *2014 22nd International Conference on Pattern Recognition*, pages 2477–2482, 2014.
- [5] Lilian Berton, Thiago de Paulo Faleiros, Alan Valejo, Jorge Valverde-Rebaza, and Alneu de Andrade Lopes. Rgcli: Robust graph that considers labeled instances for semi-supervised learning. *Neurocomputing*, 226:238–248, 2017.
- [6] Sagnik Bhattacharya. ML algorithm visualizer. <https://ml-visualizer.herokuapp.com/>, 2020. [Internet; descargado 1-abril-2024].
- [7] Jason Brownlee. How to evaluate machine learning algorithms. <https://machinelearningmastery.com/how-to-evaluate-machine-learning-algorithms/>, 2020. [Internet; descargado 12-junio-2024].

- [8] DataScientest. Machine learning: definicion, funcionamiento, usos. <https://datascientest.com/es/machine-learning-definicion-funcionamiento-usos>, 2021. [Internet; descargado 16-enero-2024].
- [9] emeritus. What is supervised learning in machine learning? a comprehensive guide. <https://emeritus.org/blog/ai-and-ml-supervised-learning/>, 2023. [Internet; descargado 17-enero-2024].
- [10] Chrome for Developers. Lighthouse. <https://developer.chrome.com/docs/lighthouse>, 2016. [Internet; descargado 12-junio-2024].
- [11] Samadrita Ghosh. The ultimate guide to evaluation and selection of models in machine learning. <https://neptune.ai/blog/ml-model-evaluation-and-selection>, 2024. [Internet; descargado 12-junio-2024].
- [12] ibm. What is machine learning? <https://www.ibm.com/topics/machine-learning>, 2019. [Internet; descargado 15-enero-2024].
- [13] Ming Li and Zhi-Hua Zhou. Improve computer-aided diagnosis with machine learning techniques using undiagnosed samples. *IEEE transactions on systems, man, and cybernetics - Part A: Systems and Humans*, 37(6):1088–1098, 2007.
- [14] Vidushi Meel. What is semi-supervised machine learning? a gentle introduction. <https://viso.ai/deep-learning/semi-supervised-machine-learning-models/>, 2021. [Internet; descargado 18-enero-2024].
- [15] César García Osorio and José Francisco Diez Pastor. *Aprendizaje automático. Introducción y problemas tipo*. Sistemas Inteligentes, 2022.
- [16] Marta Palacio. *Scrum Master*. Scrum Manager, 2021.
- [17] R. Polikar. Ensemble based systems in decision making. *IEEE Circuits and systems magazine*, 6(3):21–45, 2006.
- [18] Kurtis Pykes. Introduction to unsupervised learning. <https://www.datacamp.com/blog/introduction-to-unsupervised-learning>, 2024. [Internet; descargado 17-enero-2024].
- [19] Zixing Song, Xiangli Yang, Zenglin Xu, and Irwin King. Graph-based semi-supervised learning: A comprehensive review. *IEEE Transactions on Neural Networks and Learning Systems*, 34(11):8174–8194, 2023.

- [20] Soumya. Bagging vs boosting in machine learning. <https://www.geeksforgeeks.org/bagging-vs-boosting-in-machine-learning/>, 2022. [Internet; descargado 20-marzo-2024].
- [21] Jesper E. van Engelen and Holger H. Hoos. A survey on semi supervised learning. *Machine Learning*, 109:373–400, 2020.
- [22] Guido van Rossum, Barry Warsaw, and Alyssa Coghlan. Pep 8 - style guide for python code. <https://peps.python.org/pep-0008/>, 2013. [Internet; descargado 28 de marzo de 2024].
- [23] Wikipedia. Árbol kd — wikipedia, la enciclopedia libre. https://es.wikipedia.org/w/index.php?title=%C3%81rbol_kd&oldid=128605031, 2020. [Internet; descargado 20-agosto-2020].
- [24] Dengyong Zhou, Olivier Bousquet, Thomas Lal, Jason Weston, and Bernhard Schölkopf. Learning with local and global consistency. *Advances in neural information processing systems*, 16, 2003.
- [25] Xiaojin Zhu and Andrew B. Goldberg. *Introduction to Semi-Supervised Learning*. Morgan and Claypool, 2009.