Universidad Internacional de Valencia

Implementación y comparativa de métodos semi-supervisados

Titulación:
Máster Universitario en Inteligencia Artificial

Curso académico:
2023-2024

Alumno/a: Martínez Acha David
D.N.I: 71310644H

Director/a de TFM: Irma Sanabria

Convocatoria:

Primera

septiembre 2024

De:
Planeta Formación y Universidades

El ayer es historia, el mañana es un misterio y el hoy es un regalo. . . por eso se llama presente.

Eleanor Roosevelt

Agradecimientos

A mi tutora, Irma Sanabria, que pese a las dificultades, me ha ayudado, corregido y respondido a mis dudas en todo momento.

A Alvar Arnáiz y Cesar Ignacio García, mis tutores por parte de la Universidad de Burgos, que me han apoyado y dedicado su tiempo para que este trabajo saliera a delante. Del mismo modo, agradecer también a mis compañeros de despacho.

Gracias a mis padres y a mi hermana por todo su apoyo. Gracias, mamá y papá, por darme todas las facilidades de hacer este máster y vuestra calma. Gracias a mi hermana, por tu apoyo incondicional y compresión, siempre recurro a ti en los momentos complicados. Esto es tan vuestro como mío. Gracias al resto de la familia, con especial cariño a mis primos, que me han mantenido entretenido y con los pies en el suelo.

Por último, gracias a mis amigos, que aunque sin ser partícipes directos en este trabajo, son una fuente de ánimo constante. Gracias por las risas que generan momentos de desconexión y que lo han hecho todo mucho más llevadero.

Índice general

Índice de figuras

Índice de tablas

Índice de algoritmos

Resumen

En este proyecto se aborda el aprendizaje semi-supervisado, una rama del *machine learning* a la que no se le da tanta importancia como al aprendizaje supervisado y no supervisado, pero que puede ofrecer una ventaja muy grande al aprovechar los datos no etiquetados. A través de una revisión bibliográfica a partir del material encontrado y también recopilado por el grupo de investigación de la Universidad de Burgos, se seleccionaron los métodos de árboles y grafos. Se ha desarrollado el método *SSLTree*, basado en árboles de decisión (CART) y la teoría de **?** y se han implementado dos algoritmos de construcción de grafos, GBILI **?** y RGCLI **?**, junto con el método de inferencia LGC **?**. Los métodos han sido probados en 24 conjuntos de datos, mostrando que *SSLTree* es competitivo, aunque con un rendimiento peor en *ensembles*, mientras que los métodos de grafos tienen peor rendimiento de base respecto al resto de métodos comparados. Como objetivo adicional, se ha creado una aplicación web "docente" para la visualización de los algoritmos basados en grafos desarrollados, disponible en https://dmacha.dev/gssl.

Introducción 1

El aprendizaje automático o *machine learning* como disciplina de la inteligencia artificial resulta ser uno de los campos más cotizados y que despierta más interés en prácticamente cualquier aplicación (investigación, automatización, sistemas de ayuda, detección...). Existe una división muy clara del aprendizaje automático que consta de: aprendizaje supervisado y el no supervisado. Pero existe otra división que no suele mencionarse, y que puede ser muy beneficiosa, este es el aprendizaje semi-supervisado.

De forma resumida, el aprendizaje supervisado trata de aprender de datos de los que se sabe lo que representan para después poder inferir este conocimiento para nuevos datos (por ejemplo, dadas las características de una flor, se intenta predecir de qué clase concreta es), el aprendizaje no supervisado trata de aprender de datos de los que **no** se sabe lo que representan, se utiliza en tareas en las que es necesario realizar agrupaciones o divisiones con base en las similitudes/disimilitudes de los ejemplos (es decir, podría distinguir entre animales que tienen plumaje de los que no sin tener el conocimiento de qué animales son concretamente). En el caso del aprendizaje supervisado, el etiquetado de los datos suele ser un proceso costoso (es posible imaginar, por ejemplo, la cantidad de tiempo y recursos que podría suponer el etiquetado masivo de millones de muestras de posibles cánceres). En la realidad, la mayor parte de los datos no están etiquetados. Ante esta necesidad aparece el aprendizaje semi-supervisado, que se sitúa en la intersección del supervisado y no supervisado y que permite aprovechar los escasos datos etiquetados para inferir su conocimiento a los no etiquetados.

Como se ha comentado, los algoritmos semi-supervisados suponen un área de mucha utilidad dentro del *machine learning*, sin embargo, así como para otras ramas (como el aprendizaje supervisado y no supervisado) es posible encontrar numerosas bibliotecas y algoritmos bien desarrollados y probados, para el semi-supervisado todavía hay una gran cantidad de investigación que no se ha materializado (o que si lo ha hecho, no se ha publicado). Se pretende contribuir en el desarrollo de estos algoritmos. Para ello, el objetivo consiste en realizar una revisión bibliográfica de métodos semi-supervisados con la posterior implementación de algunos de los más prometedores junto con su comparación exhaustiva con otros métodos afianzados.

La organización de la documentación es la siguiente: se comienza con el estado del arte para tener un contexto de panorama de los métodos implementados, se introduce el marco teórico del aprendizaje automático, de los árboles de decisión y los grafos, se plantean

los objetivos del proyecto alineados con los propuestos por la Universidad de Burgos y se expone el desarrollo completo de los métodos seleccionados, experimentos, resultados y conclusiones. Por último, se analizan las posibles perspectivas de futuro.

Estado del arte

2

En este apartado se comenta la revisión bibliográfica realizada. Es conveniente señalar que, debido a la naturaleza de este proyecto (en colaboración con la Universidad de Burgos y con ciertos objetivos pre-fijados), se parte de una bibliografía acotada para los métodos de árboles y también de una revisión sistemática sobre los métodos del Aprendizaje semi-supervisado basado en grafos o *Grahp-based semi-supervised learning* (GSSL).

2.1. Árboles de decisión en aprendizaje semi-supervisado

Debido a la versatilidad, interpretabilidad y las múltiples ventajas que ofrecen los árboles de decisión, su uso se ha extendido a numerosos dominios de aplicación.

En **?** se propone el *Tree-Based Bayesian (TBB)* que asume la existencia de un árbol construido con la ayuda de los datos no etiquetados. La idea principal de la teoría bayesiana es calcular la probabilidad de que un dato sea positivo (en dominios binarios, aunque indican que es generalizable a multiclase). Se define un conjunto de hipótesis, cada una representa una asignación de clase a todo el conjunto de datos (para cada ejemplo). Dicha probabilidad (positiva) se calcula como la suma de todas las probabilidades de todas las hipótesis consistentes (consistentes con las etiquetas conocidas) que lo clasifican como positivo entre la suma total de las probabilidades de todas las hipótesis consistentes (sin limitar si es positivo o negativo). También introducen el concepto de las mutaciones, que resultan en un cambio aleatorio (con base en un ratio) de la etiqueta de un nodo del árbol. Estas mutaciones permiten introducir otro enfoque denominado *Tree Nearest Neighbor (TNN)* para etiquetar según el nodo más cercano.

Otras propuestas como **?** aplican conceptos de semi-supervisado como el *self-learning* en el que se crea un *Random Forest* (conjunto de árboles de decisión). Este método utiliza un proceso iterativo junto al *out-of-bag-error* para detectar si los ejemplos no etiquetados podrían mejorar la clasificación. Durante el entrenamiento, cada uno de los árboles de decisión es reentrenado con ejemplos pseudo-etiquetados (obtenidos por el conocimiento de la iteración previa). De hecho, si el *out-of-bag-error* del *Random Forest* final es mayor que el obtenido con solo los ejemplos etiquetados, se retorna este último. Otra perspectiva en esta línea es **?**, que aplica este mismo *self-learning* al *Rotation Forest* **?** (miembro del grupo de investigación de la Universidad de Burgos), que es una evolución de los *Random Forest* y considerado como estado del arte. Del mismo modo, el *Rotation Forest* es entrenado a partir de los ejemplos

etiquetados y, mediante un proceso iterativo, se reentrena con nuevas instancias pseudo-etiquetadas.

Continuando con *self-learning*, en **?** se aplica directamente a un árbol de decisión individual. Muy parecido al algoritmo denominado *Self-Training*, utiliza los ejemplos pseudo-etiquetados con mayor confianza para re-entrenar. Los árboles de decisión suelen calcular probabilidades de etiqueta en función de la distribución de las clases en las hojas pero esto no mejora el rendimiento. En cambio, además del *self-training*, también proponen el uso de *Laplacian correction* para estimaciones más fiables y *no-pruning* que evita precisamente el subajuste en conjuntos de datos pequeños.

Es en **?** en el que aparece un método de **construcción** de un árbol semi-supervisado. El propio árbol es capaz de manejar los datos no etiquetados sin necesidad de procesos iterativos que los incorporan. La propia construcción del árbol aprovecha los datos no etiquetados. Esto lo realiza incorporando la variabilidad de las características a la componente de impureza en las etiquetas (generalmente el coeficiente de *gini*). La premisa es que grupos con poca variabilidad tendrán etiquetas similares. Los mismos autores en **?** extienden este enfoque a los problemas de *multi-target regression* (se predicen varios valores continuos por cada ejemplo).

2.2. GSSL (Graph-based semi-supervised learning)

Como punto de partida, en **?** se realizó una revisión del estado del arte en GSSL. Sin embargo, como se indica en **?**, la primera revisión no establecía relaciones entre los métodos, incluían modelos basados en redes neuronales que no estaban basados en grafos y además, no establecen un marco general.

En **?** se ha encontrado la mejor y más actualizada revisión del panorama GSSL que además establece una nueva taxonomía para estos métodos. Dicha taxonomía propone una división de todos los métodos en dos pasos: construcción de grafo e inferencia de etiquetas.

2.2.1. Construcción del grafo

Es el primer paso crucial para los métodos GSSL, el objetivo es representar los ejemplos como nodos y las aristas con ciertos pesos representando la similitud entre ellos. En la literatura existen varios algoritmos divididos según el uso o no de la información de etiquetas.

En el caso de los no supervisados, aparecen los basados en K vecinos más cercanos o *k-Nearest Neighbor* (kNN) que conectan de forma voraz un nodo con sus k vecinos más cercanos (en base a una métrica de distancia). Aparecen otros métodos basados en ϵ-*neighborhood* en los que solo se conectan dos nodos si la distancias entre ellos es menor a ϵ. Incluso otros que aprovechan KNN en combinación con los vecinos mutuos de tal forma que dos nodos están unidos si son mutuamente vecinos cercanos **?**. Esto último nace debido a que los *hubs* o vértices con un gran número de aristas que llegan a él (grado) empeoran el *accuracy* en tareas de clasificación. Aparece también otro tipo de métodos similar a KNN

conocido como *b-Matching* que aborda el problema de los grados de los vértices limitando que cada nodo solo pueda tener exactamente b vecinos.

Para los métodos supervisados en los que se quiere utilizar la información de etiquetado, aparecen estudios como en **?** para estudiar dicha posibilidad. Con base en estos estudios, aparecen GBILI **?** como primera versión y RGCLI **?** como continuación de los mismos autores que permiten mejorar el *accuracy* en clasificación.

2.2.2. Inferencia de etiquetas

Este paso es el más importante en las tareas de clasificación, pues permite obtener las etiquetas de ejemplos no etiquetados.

2.2.2.1. Graph regularization

El objetivo de la *regularización de grafos* es encontrar una función f que permita obtener etiquetas para ejemplos. Esta función busca cumplir dos objetivos: la función debe estar lo más cerca posible a las etiquetas conocidas y debe variar suavemente en todo el grafo construido (la intuición es que nodos que se consideran similares tendrán la misma etiqueta hasta que cierta disimilitud haga cambiar esta predicción). Para lograr esto, estos métodos toman como base una función de pérdida compuesta por la pérdida supervisada (para intentar cumplir ese primer objetivo) y la pérdida de regularización (para el segundo objetivo).

Como primera familia de métodos de regularización se encuentra ***Label Propagation***. Parte del grafo construido en el paso anterior, donde algunos vértices corresponden con ejemplos etiquetados y a continuación propagan este ”conocimiento“ a los ejemplos no etiquetados a partir de la similitud. Existen dos aproximaciones principales **?**:

1. *Gaussian random fields*: En GRF se parte de una función de energía (que resulta ser la función de pérdidas) que intenta asegurar que nodos muy similares (aristas pesadas) tendrán las mismas etiquetas **?**.

2. *Local and global consistency (LGC)*: LGC extiende GRF a problemas multiclase para poder ser aplicado a más problemas y además añade alguna relajación en cuanto a la restricción de mismas etiquetas si dos nodos son muy similares (de esta forma puede ajustarse mejor al ruido de las etiquetas) e incluso penalizan la etiqueta de un nodo cuando su grado es demasiado grande (en comparación con el resto, en grafos irregulares) **?**.

Otros métodos son los de *Directed regularization*, que permiten trabajar con grafos dirigidos de tal forma que se tenga en cuenta la direccionalidad de las aristas en la función de pérdidas como se propone en **?**, que introduce un *random walk* para obtener una distribución de probabilidad (probabilidad de llegar a un nodo en un grafo dirigido) y se incorpora en la función de pérdidas.

A partir de estos métodos aparecen otros cuya teoría aborda otros paradigmas como *Manifold regularization* que combina la teoría de grafos espectral con *manifold learning* (una

forma de reducción de dimensionalidad no lineal) **?**, *Label Prediction via Deformed Graph Laplacian (LPDGL* que partiendo de *Label propagation* y *Directed regularization* incluye un término de suavidad **?**, y por último, el *Poisson learning* que nace de la idea de lidiar con tasas de etiquetado muy bajas **?**.

2.2.2.2. Graph embedding

La idea de *Graph embedding* es construir representaciones de los grafos con menor dimensionalidad. Se trabaja a nivel de nodo, codificando dichos nodos como vectores en menor dimensión pero manteniendo posición y vecindad. La aproximación general es la de *encoder-decoder*. El *encoder* se encarga de transformar los nodos en vectores. El *decorder* se encarga de reconstruir la información original. El objetivo es que la reconstrucción sea lo mejor posible (pues el *encoder* estará realizando una reducción fiable).

2.2.2.3. Shallow graph embedding

Como su nombre indica, son métodos más simples (menos profundos) en los que se distinguen métodos basados en factorización como *Locally linear embedding (LLE)* donde el *embedding* resulta ser la combinación lineal de los nodos de la vecindad **?**. También *Laplacian eigenmaps* en los que se asegura que nodos muy cercanos también estarán cerca en el nuevo espacio **?** y *Graph factorization* que emplea la matriz de adyacencia (los pesos de las aristas) **?** entre otros. También existe otra familia basada en *random-walks* como *DeepWalk* que está basado en el modelo *skip-gram* **?**. Este tipo de *embedding* tiene la problemática principal de no utilizar las características de los nodos.

2.2.2.4. Deep graph embedding

En *Deep graph embedding* se emplean redes neuronales para aprender esas representaciones comentadas. En este caso sí que utiliza tanto la estructura del grafo como las características. Existe una familia de métodos basados en *Autoencoder* y otros basados en redes neuronales de grafos (*Graph Neuronal networks*).

2.3. Líneas de investigación seleccionadas

En el caso de los árboles, los trabajos más interesantes desde el punto de vista de la utilidad e implementación son aquellos que proponen métodos de construcción de árboles semi-supervisado. Se ha seleccionado el trabajo realizado en **?** para clasificación. En este, no se propone una implementación explícita de un algoritmo, pero los conceptos comentados permiten desarrollar uno propio. El método que sus autores implementaron a raíz de los conceptos parece no ser accesible. La intención, por tanto, es crear una implementación basada en los conceptos propuestos, que sea accesible y bien probada en diferentes ámbitos. Además, el resultado de este trabajo (un árbol intrínsecamente semi-supervisado) permite ser utilizando en otros proyectos del grupo de investigación de la Universidad de Burgos, como

por ejemplo, el intento de crear un Rotation Forest semi-supervisado sin utilizar *self-training* u otros enfoques.

Para el caso de los métodos GSSL, se han seleccionado ciertos métodos/algoritmos: *Graph-based on informativeness of labeled instances (GBILI)*, *Robust Graph that Considers Labeled Instances (RGCLI)* y *Local and global consistency (LGC)*. El principal motivo de su elección es la disponibilidad de *software* desarrollado y probado a disposición de la comunidad. Tanto para GBILI y RGCLI en **?** se realiza una búsqueda de los algoritmos implementados pero parece no existir implementaciones *open-source* de ambos algoritmos. LGC resulta ser el método más general y extendido que propagación de etiquetas, es el punto de partida para comparar estos algoritmos entre ellos y con otros modelos.

Por otro lado, en los artículos de GBILI **?** y RGCLI **?** se realizan experimentaciones con relativamente pocos conjuntos de datos. Además, se desconoce si han sido seleccionados convenientemente de tal forma que puedan alterar/beneficiar los resultados (debido al cumplimiento de las suposiciones 3.1.3.1). Con este proyecto se pretende aplicar ambos algoritmos en una mayor variedad de dominios (i.e múltiples conjuntos de datos). Además, la comparación con otros modelos no se realiza contra otros modelos bien establecidos y eficaces en el aprendizaje semi-supervisado como para justificar el uso de estos algoritmos (y no otros no basados en grafos). En este proyecto también se comparará con dichos otros modelos.

Marco teórico 3

3.1. Aprendizaje automático

El aprendizaje automático (*machine learning*) según **?** es una rama de la Inteligencia Artificial y se trata de una técnica de análisis de datos que enseña a las computadoras a aprender de la **experiencia** (como los humanos). Para llevar a cabo este proceso, el aprendizaje automático requiere de una amplia cantidad de datos, o los necesarios para el problema específico en cuestión. Estos datos son procesados mediante algoritmos, los cuales se alimentan de ejemplos (también conocidos como instancias o prototipos). A través de estos ejemplos, los algoritmos tienen la capacidad de generalizar comportamientos ocultos.

Estos algoritmos mencionados mejoran su rendimiento iterativamente y de forma automática durante su entrenamiento e incluso también durante su aprovechamiento/explotación. El aprendizaje automático ha adquirido una gran relevancia en una amplia variedad de áreas como la visión artificial, automoción, detección de anomalías o automatización, entre otras. El aprendizaje automático generalmente se clasifica en tres tipos: aprendizaje supervisado, aprendizaje no supervisado y aprendizaje por refuerzo. Sin embargo, ha surgido una nueva disciplina que se sitúa entre el aprendizaje supervisado y el no supervisado, utilizando tanto datos etiquetados como no etiquetados durante el proceso de entrenamiento **?**.

En la figura 3.1 se presenta una clasificación del aprendizaje automático.

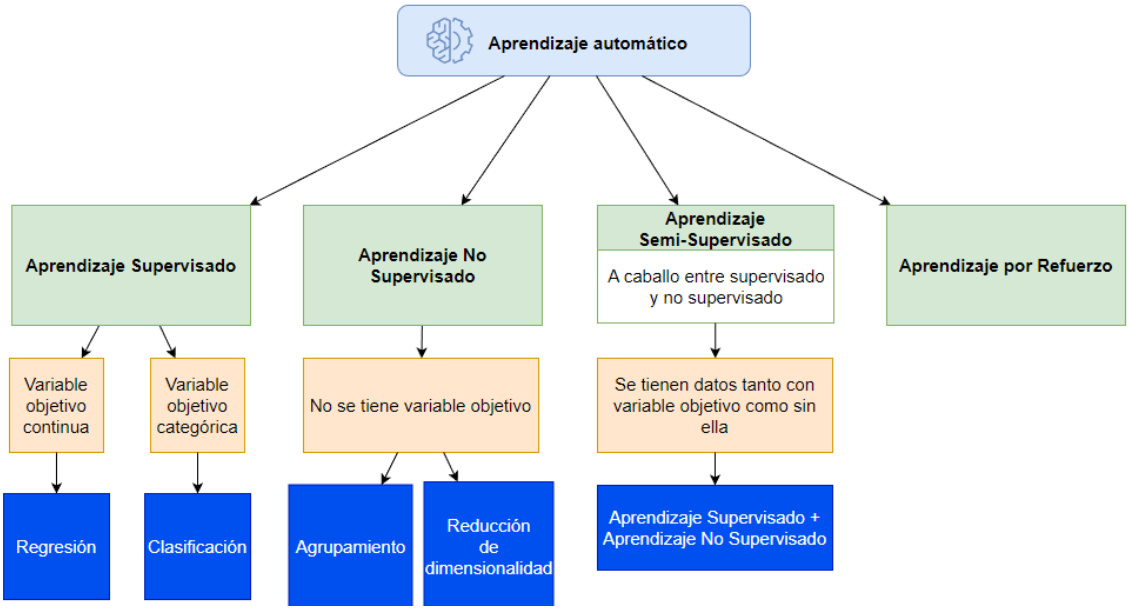

Figura 3.1: ***Clasificación de aprendizaje automático, basado en ?.***

3.1.1. Aprendizaje supervisado

Los algoritmos de aprendizaje supervisado utilizan datos etiquetados durante su proceso de entrenamiento **?**. Un ejemplo popular de datos etiquetados podría ser el conjunto flores de iris y las posibles etiquetas podrían ser: setosa, versicolor y virginica. Estos datos estarán formados por un conjunto de características (en el caso de las flores de iris podrían ser la longitud y ancho del sépalo y del pétalo). Estas características podrían ser categóricas, continuas o binarias **?**.

Para generar un modelo correcto, estos datos son divididos en varios subconjuntos: conjunto de entrenamiento (*training data set*), conjunto de validación (*validation data set*) y conjunto de test (*test data set*). El conjunto de entrenamiento corresponde con la porción de los datos que el algoritmo utilizará para aprender un modelo que generalice los patrones ocultos subyacentes. El conjunto de validación permite comprobar, durante el proceso de entrenamiento, que el modelo que se está generando no memoriza los datos (fenómeno conocido como sobreajuste), también sirve para finalizar el entrenamiento (e.g. el error en validación aumenta durante varias iteraciones). Una vez que el algoritmo ha generado un modelo, se utiliza el conjunto de test para comprobar el rendimiento real (una estimación) **?**. Ningún dato de este último conjunto ha sido "visto" por el modelo previamente.

En la figura 3.2 se encuentra un diagrama con el funcionamiento general.

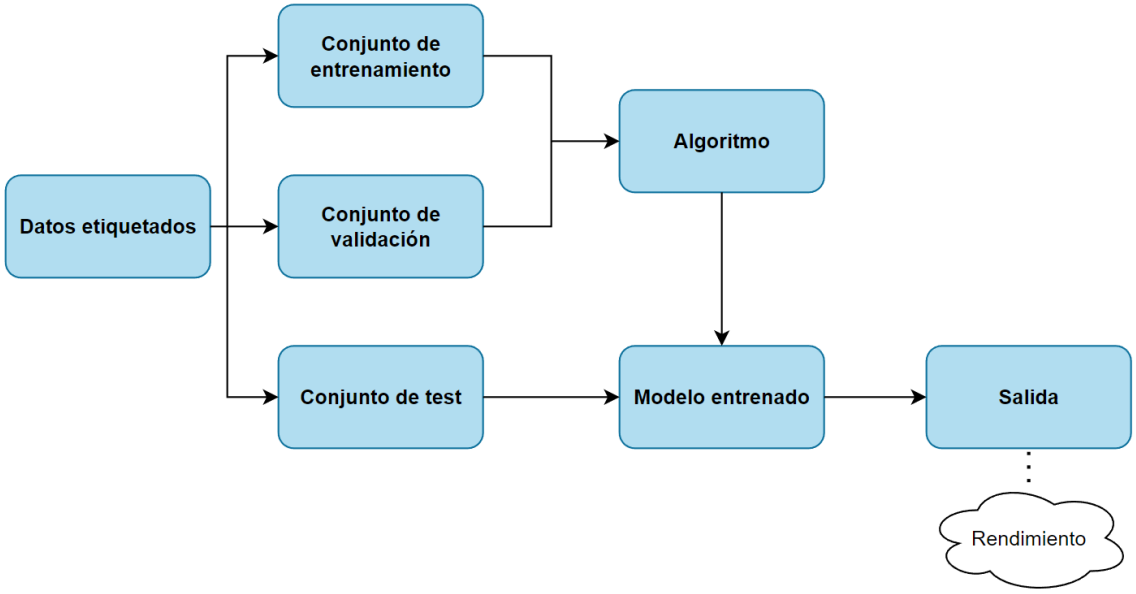

Figura 3.2: ***Funcionamiento general del aprendizaje supervisado, basado en ?.***

Partiendo del concepto de etiqueta de un dato, el problema será de **clasificación** si los valores que puede tomar la etiqueta representan un conjunto finito. Por otro lado, si estos valores son continuos, el problema será de **regresión**.

- **Clasificación**: Un modelo entrenado en un problema de clasificación se denomina clasificador. Ante un nuevo dato, el clasificador predecirá su etiqueta correspondiente. Por lo general, a cada valor de etiqueta se le suele llamar clase. Dependiendo de la cantidad de valores, se referirá a un problema binario o multiclase.
- **Regresión**: En este caso, ante un nuevo dato, el modelo predecirá un valor continuo. La idea subyacente es evaluar una función (ajustada/aprendida durante el entrenamiento) dado un dato como variables de entrada.

3.1.2. Aprendizaje no supervisado

A diferencia del aprendizaje supervisado, el no supervisado no trabaja con datos etiquetados y clases. Según **?** esto quiere decir que nosotros no "supervisamos" el algoritmo. No se le añade ese conocimiento extra. Estos algoritmos intentarán descubrir patrones que se encuentren en la propia estructura de los datos (de sus características). La idea del aprendizaje no supervisado es estudiar las similitudes/disimilitudes que hay entre los datos y, por ejemplo, obtener una separación o agrupación de los mismos (e.g. separación de especies en imágenes de animales sin conocer el animal concreto).

Entre las principales aplicaciones del aprendizaje no supervisado se encuentran las siguientes:

1. **Agrupamiento (Clustering)**: Divide los datos en grupos. Los ejemplos de un grupo tendrán cierta similitud entre ellos, mientras que todos los ejemplos de ese grupo serán

disimilares a los de otro grupo (y por eso se genera esa división). Algunos algoritmos necesitan conocer de antemano el número de grupos en los que dividir lo datos, otros son capaces de descubrir cuántos grupos existen **?**. En la figura 3.3 puede verse un ejemplo de agrupamiento.

2. **Reducción de la dimensionalidad**: Los conjuntos de datos generalmente tiene un número bastante grande de características. Esto hace que los algoritmos de aprendizaje sean más lentos. La reducción de dimensionalidad hace referencia a la reducción de número de características tratando de no perder información al hacerlo. Según **?** se denomina como una forma de convertir conjuntos de datos de alta dimensionalidad en conjunto de datos de menor dimensionalidad, pero garantizando que proporciona información similar.

   Algunos ejemplos concretos de reducción de dimensionalidad son:

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

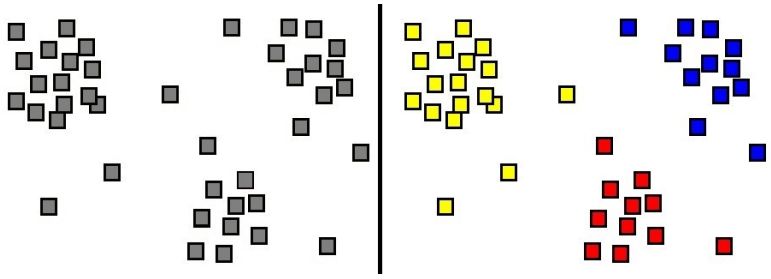

Figura 3.3: Clusters. *A la izquierda los datos sin agrupar y a la derecha los datos coloreados según la pertenencia a los distintos grupos. By hellisp - Own work, Public Domain, https://commons.wikimedia.org/w/index.php?curid=36929773.*

3.1.3. Aprendizaje semi-supervisado

Según **?**, el aprendizaje semi-supervisado es la rama del aprendizaje automático que utiliza tanto datos etiquetados como no etiquetados durante el entrenamiento. Es por esto que se dice que está a medio camino entre el aprendizaje supervisado y el no supervisado. Como se ha comentado, el problema al que todos los algoritmos se enfrentan en la realidad es a la escasez de datos etiquetados, pues es un proceso costoso. Gracias a la naturaleza del semi-supervisado, hace que sea una buena aproximación para esos casos. Por lo general, suele aplicarse en problemas de clasificación.

3.1.3.1. Suposiciones

¿Y por qué utilizar aprendizaje semi-supervisado? Lo cierto es que algunos de los algoritmos existentes de aprendizaje supervisado funcionan bastante bien incluso con pocos datos

etiquetados. Sin embargo, los datos no etiquetados podrían aprovecharse para mejorar el rendimiento.

El objetivo, por tanto, del aprendizaje semi-supervisado será obtener clasificadores que obtengan mejores resultados que los del aprendizaje supervisado. En **?** se especifican unas condiciones que han de cumplirse.

La primera premisa que se debe cumplir es que la distribución $p(x)$ de entrada contenga información sobre la distribución posterior $p(y|x)$ **?**.

Smoothness assumption

Probablemente, si dos ejemplos se encuentran próximos en el espacio, comparten la misma etiqueta.

Low-density assumption

La frontera de decisión en un problema de clasificación se encontrará en una zona del espacio en el que existan pocos ejemplos.

Manifold assumption

Los ejemplos suele encontrarse en una estructuras de dimensionalidad baja (algunas características no son útiles), denominadas *manifolds*. Los ejemplos que se encuentren en una misma *manifold* comparten la misma etiqueta **??**.

Cluster assumption

Los ejemplos que se encuentren en un mismo grupo compartirán la misma etiqueta.

El concepto clave de todas estas suposiciones es el de la “similitud” (ejemplos próximos en el espacio, ejemplos en misma manifold, mismo grupo...). Es por esto que la *Cluster assumption* es una generalización del resto (o el resto son versiones de esta).

Por esto, para que el **el aprendizaje semi-supervisado mejore al supervisado** es necesario que se cumpla dicha suposición generalizada. Si no fuese así (i.e. datos no agrupables), el aprendizaje semi-supervisado no mejorará al supervisado **?**.

En la figura 3.4 se presenta la taxonomía general del aprendizaje semi-supervisado.

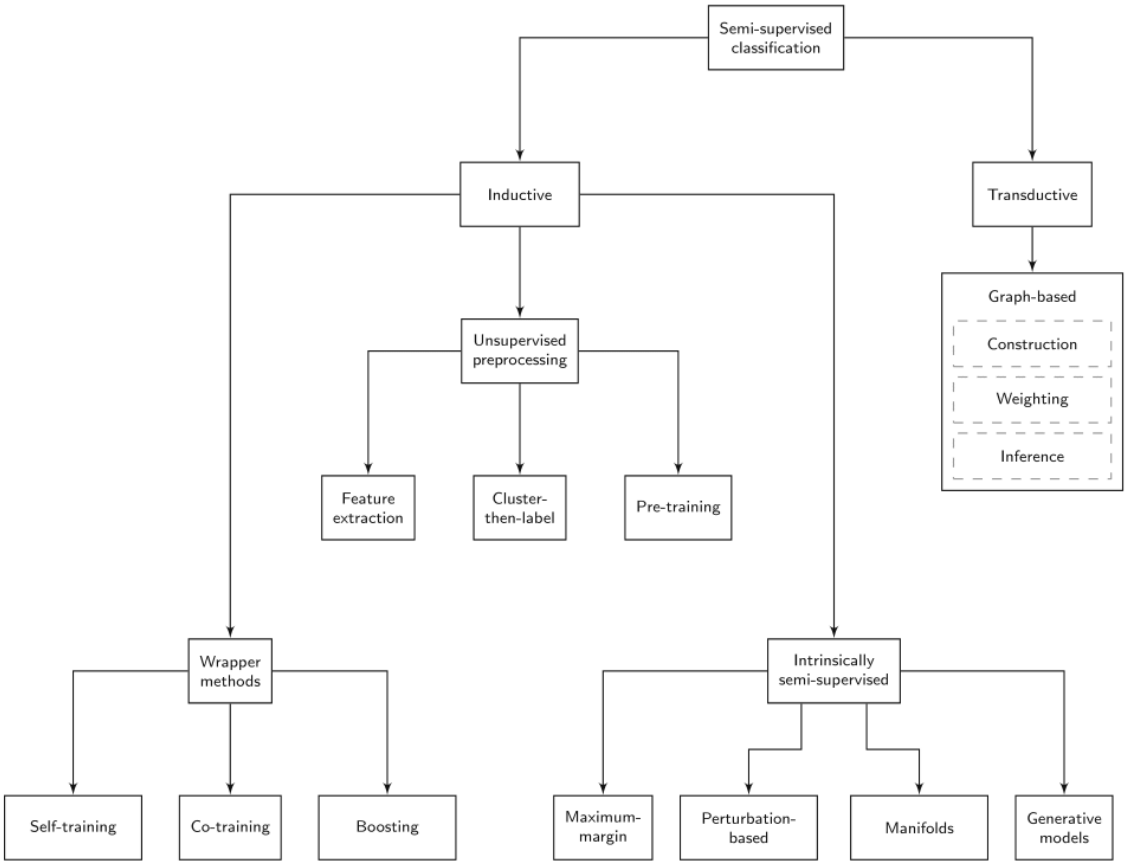

Figura 3.4: ***Taxonomía de métodos semi-supervisados* ?.**

Sin pérdida de generalidad, este trabajo estará centrado en métodos semi-supervisados basados en grafos y árboles (intrínsecamente semi-supervisados) con la comparación con otros métodos enmarcados en esta taxonomía.

3.2. Árboles de decisión (CART)

Antes de entrar en las explicaciones teóricas es conveniente indicar que existen multitud de algoritmos que permiten la creación de árboles (ID3 **?**, C4.5 **?**, C5.0 **?** y CART **?**, entre otros). El algoritmo en el que se centrará este desarrollo será CART (Classification and regression trees) para árboles de clasificación.

Árbol de decisión Los árboles de decisión son clasificadores que predicen etiquetas para instancias. Para clasificar, los árboles plantean sucesivas preguntas sobre las características de los ejemplos. Cada pregunta se realiza en uno de los nodos y se ramifica hacia un hijo por cada posible respuesta **?**. La clasificación se completa partiendo desde la raíz del árbol y "contestando" a las preguntas hasta llegar a una hoja (nodo sin hijos). Las hojas tendrán asociada la clase correspondiente.

La clave de los árboles de decisión es la formulación de las preguntas, que pueden ser bastante complicadas. Por lo general serán preguntas de si/no que contendrán una comparación (<, <=, > o =>) de una característica con un cierto valor **?**. El rendimiento del árbol dependerá de las preguntas que se establezcan en el proceso de entrenamiento.

Algoritmo CART El algoritmo CART permite construir árboles de decisión. Este algoritmo se centra en la selección de la mejor pregunta, o dicho de otra forma, la división del conjunto de datos en particiones. Los nodos deben ser vistos como particiones del conjunto de datos obtenidas por las sucesivas preguntas.

Aunque no se ha comentado, los nodos de los árboles de decisión podrían tener varios nodos hijos (incluso más de dos). En el caso de CART, genera árboles binarios. Es decir, las divisiones son solo en dos grupos cada vez.

La construcción del árbol comienza por la raíz, donde se encuentra todo el conjunto de datos de entrenamiento. A partir de este nodo, se obtiene cuál es la mejor división. Para ello, considera todas las posibles características junto con todos los posibles valores (observados en los datos) **?**. El proceso continúa recursivamente hasta obtener las hojas del árbol. El algoritmo 1 muestra el pseudocódigo simplificado de CART.

En la figura 3.5 puede verse el resultado de aplicar este algoritmo a un conjunto de datos particular. El conjunto de datos solo contiene dos características. Para seleccionar la pregunta del primer nodo, CART consideró ambas características y todos los valores que se observaron en el conjunto de datos hasta obtener la pregunta "Eje X <= 11.662".

Algoritmo 1: Algoritmo CART simplificado

1. Crear la raíz del árbol con los datos E.
2. Si se cumple un criterio de parada, devolver la raíz (sin subárboles).
3. Encontrar la mejor división de E en dos grupos E_{izq} y E_{der}.
4. Asignar el subárbol izquierdo llamando recursivamente al algoritmo CART con los datos E_{izq}.
5. Asignar el subárbol derecho llamando recursivamente al algoritmo CART con los datos E_{der}.
6. Devolver la raíz.

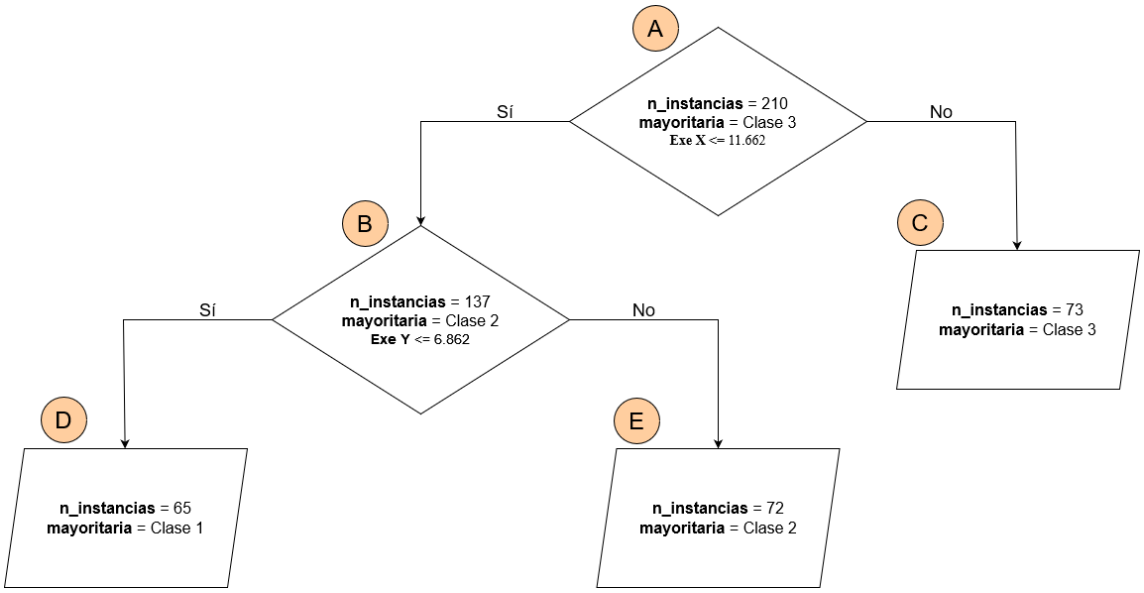

Figura 3.5: ***Árbol de decisión.***

La premisa de los árboles de decisión es obtener nodos puros. Es decir, nodos en los que los datos que corresponden a él son de una misma clase. Realmente no es necesaria esa perfección (eso resultaría en *sobreajuste*), se busca que la clase mayoritaria en esos datos tenga una proporción mucho mayor que el resto.

La pregunta “Eje X <= 11.662” de la raíz del ejemplo proviene de una comparación con todas las posibilidades restantes, para otros valores y para la otra característica. La comparación se realiza mediante un **criterio de división** o función de división **?**. Esta función arroja una medida de la **impureza** de los grupos resultantes. Por lo que, sabiendo que lo que se quiere es maximizar la pureza, se debe minimizar la impureza.

3.2.1. Criterios de impureza

En este desarrollo se han considerado dos criterios distintos. Por un lado **Gini** y por otro **Entropy**. Los dos tienen el mismo objetivo.

Gini index La función de Gini o índice Gini se calcula a partir de un conjunto de datos E y utiliza las proporciones de las clases. El objetivo es minimizar.

$$\text{Gini}(E) = 1 - \sum_{i=1}^{c} p_i^2 \tag{3.1}$$

Donde p_i es la proporción/probabilidad de la clase i.

Tomando el ejemplo de la figura 3.5, los coeficientes de Gini serían los de la tabla 3.1.

Nodo	Frecuencia			Proporción			Gini
	Clase 1	Clase 2	Clase 3	Clase 1	Clase 2	Clase 3	$1 - p_1^2 - p_2^2 - p_3^2$
A	65	72	73	0.3095	0.3429	0.3476	0.6658
B	65	72	0	0.4745	0.5255	0	0.4987
C	0	0	73	0	0	1	0
D	65	0	0	1	0	0	0
E	0	72	0	0	1	0	0

*Truncado a 4 decimales (los cálculos sí consideran todos los decimales).

Tabla 3.1: *Ejemplo Gini*

Ahora bien, se han calculado los coeficientes por cada nodo, sin embargo, esta información no es la que sea utiliza directamente para elegir “Eje X <= 11.662”, si no que se utilizan los coeficientes de los dos nodos hijos generados por esa pregunta.

Volviendo a la idea principal, se quiere obtener los grupos más homogéneos (puros) con esas preguntas. Para poder hacer una estimación se calculan estos coeficientes de Gini y después se realiza una suma ponderada de los grupos resultantes según la proporción de ejemplos.

Por ejemplo, durante la construcción del árbol se evaluaron todas las posibles preguntas en el nodo A. Al llegar a “Eje X <= 11.662” se calcularon los índices Gini para los nodos B y C. Esa estimación de lo “buena” que es la división se realiza de la siguiente manera:

$$\frac{137}{210} \times \text{Gini}(B) + \frac{73}{210} \times \text{Gini}(C) = \frac{137}{210} \times 0{,}4987 + \frac{73}{210} \times 0 = \frac{137}{210} \times 0{,}4987$$

Como el nodo C es completamente puro, comparando con el resto de todas las posibles preguntas, esta es la que minimiza los cálculos de impureza.

Entropy El cálculo de la entropía es muy similar al de Gini, se calcula a partir de un conjunto de datos E y utiliza las proporciones de las clases. El objetivo sigue siendo minimizar.

$$\text{Entropy}(E) = -\sum_{i=1}^{c} p_i \cdot \log_2(p_i) \tag{3.2}$$

Donde p_i es la proporción/probabilidad de la clase i.

Nodo	Frecuencia			Proporción			Entropy
	Clase 1	Clase 2	Clase 3	Clase 1	Clase 2	Clase 3	$-(p_1 \cdot \log_2(p_1) + p_2 \cdot \log_2(p_2) + p_3 \cdot \log_2(p_3))$
A	65	72	73	0.3095	0.3429	0.3476	1.5831
B	65	72	0	0.4745	0.5255	0	0.9981
C	0	0	73	0	0	1	0
D	65	0	0	1	0	0	0
E	0	72	0	0	1	0	0

*Truncado a 4 decimales (los cálculos sí consideran todos los decimales).

Tabla 3.2: *Ejemplo Entropy*

De la misma forma que con Gini, el cálculo de la entropía viene acompañado de la suma ponderada. Por ejemplo, continuando con el mismo ejemplo, la suma ponderada de los nodos B y C sería:

$$\frac{137}{210} \times \text{Entropy}(B) + \frac{73}{210} \times \text{Entropy}(C) = \frac{137}{210} \times 0{,}9981 + \frac{73}{210} \times 0 = \frac{137}{210} \times 0{,}9981$$

Para comprobar que esta intuición de la homogeneidad funciona, en la figura 3.6 se pueden observar las fronteras de decisión para una clasificación perfecta mediante Gini y su cálculo de impureza.

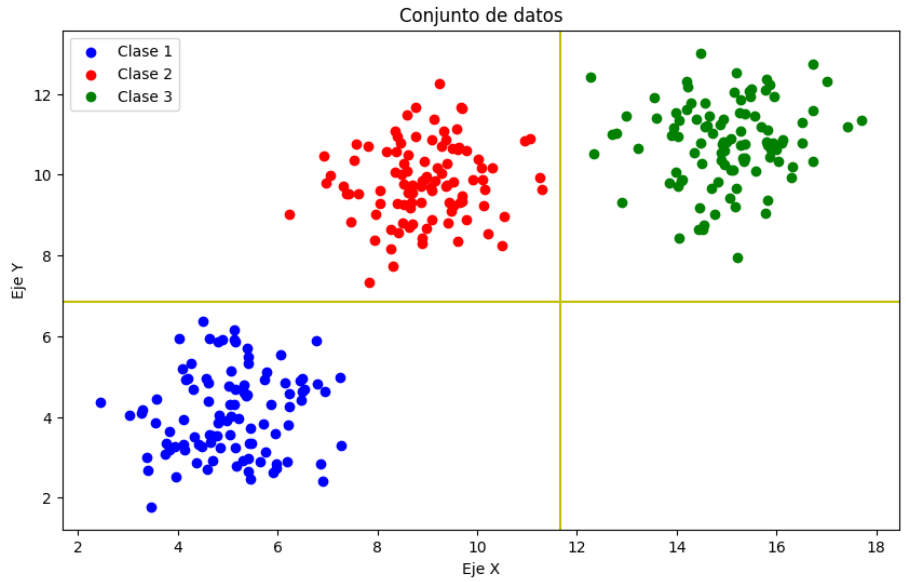

Figura 3.6: ***Fronteras de decisión.***

3.3. Aprendizaje semi-supervisado basado en grafos

Una variante para enfocar el aprendizaje semi-supervisado son los métodos basados en grafos (GSSL del inglés Graph-based Semi-Supervised Learning). Estos métodos resultan prometedores debido a que la construcción de estructuras de grafos tienen una relación natural con la *manifold assumtion* 3.1.3.1. Cuando se construye un grafo, los nodos conectados con conexiones con pesos altos suelen tener las mismas etiquetas, lo que corresponde con esa *manifold assumtion* **?**.

Grafo Un grafo es un par $G = (V, E)$ donde V es en un conjunto de objetos llamados vértices $V = \{v_1, v_2, ...\}$ y E otro conjunto de elementos denominados aristas $E = \{e_1, e_2, ...\}$ **?**.

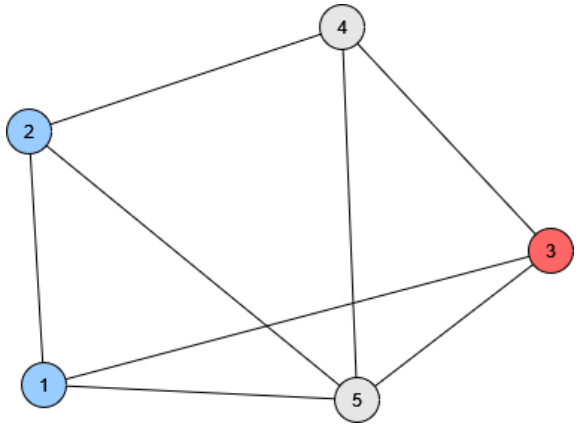

Figura 3.7: ***Ejemplo de grafo.***

En GSSL, cada ejemplo del conjunto de datos representa un vértice o nodo y estarán conectados por aristas ponderadas con un peso que representarán la similitud. En la figura 3.7 puede verse un ejemplo de un grafo (los nodos grises representan ejemplos no etiquetados).

Para la tarea del aprendizaje, los métodos GSSL se fundamentan en dos pasos:

1. Construcción del grafo: Se construye el grafo que indica la similitud de los ejemplos del conjunto de entrenamiento. Tiene en cuenta tanto datos etiquetados como no etiquetados.

2. Inferencia de etiquetas: La información de etiquetado se propaga desde los etiquetados a los no etiquetados a partir de la información del grafo construido.

Debido al gran alcance de los métodos GSSL se cree conveniente comentar la taxonomía propuesta en **?**. Este trabajo estará centrado en los elementos resaltados de la figura 3.8.

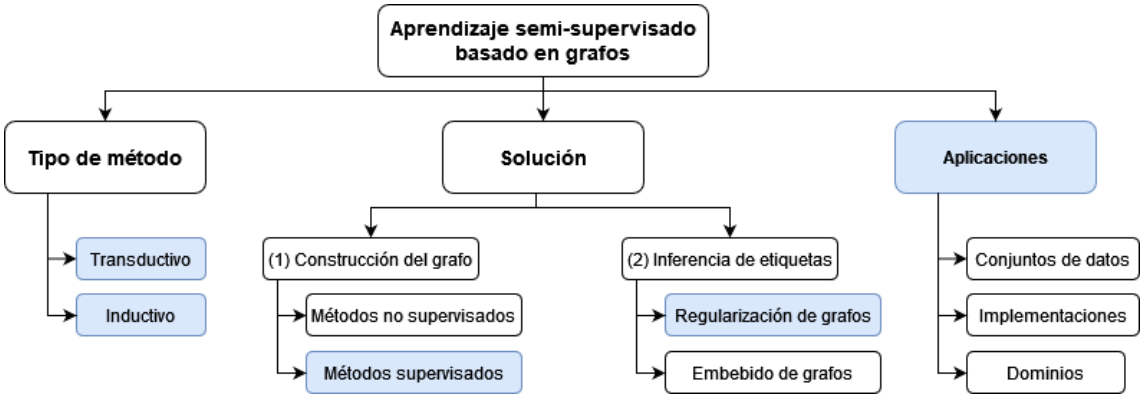

Figura 3.8: ***Taxonomía GSSL, basada en ?.***

El tipo de método hace referencia a la capacidad de predecir cualquier nuevo ejemplo no etiquetado.

Se parte de un conjunto de datos de **entrenamiento** $D = \{\{x_i, y_i\}_{i=1}^{n_l}, \{x_i\}_{i=l}^{n_u}\}$ con datos etiquetados y no etiquetados. El método transductivo aprende una función $f : X \rightarrow Y$ de tal forma que f pueda predecir los ejemplos no etiquetados $\{x_i\}_{i=l}^{n_u}$ (son los del conjunto de datos de entrenamiento). El método inductivo aprende una función $f : X \rightarrow Y$ a partir de D de tal forma que f pueda predecir cualquier ejemplo $x \in X$ (todo el dominio).

Como se ha comentado anteriormente, la solución mediante un método GSSL consta de dos pasos. El objetivo de la construcción del grafo es obtener ese par $G = (V, E)$ con la adición de una matriz W que serán los pesos de las aristas y representarán la similitud comentada. Se tienen en cuenta tres suposiciones:

1. Grafo no dirigido, W simétrica y $W_{ij} \geq 0, \forall i \neq j$
2. $w_{ij} = 0$ significa que no hay arista entre i y j.
3. Sin bucles en el grafo, $W_{ii} = 0, \forall 1 \leq i \leq n$

Dentro de todos los posibles algoritmos de construcción de grafos, se clasifican en **no supervisados** y **supervisados**. En el caso de los primeros, no utilizan información de etiquetado. Estos son algoritmos como *KNN* o *b-Matching*, entre otros. Los algoritmos supervisados aprovechan la información de las etiquetas mientras se construye el grafo (manteniendo o heredando conceptos de los algoritmos no supervisados sobre vecindad). Estos últimos son los que se han implementado en este proyecto. En cuanto a la inferencia de etiquetas, este proyecto se ha centrado en la regularización de los grafos durante dicho proceso de inferencia. Uno de los paradigmas de estos algoritmos se conoce como *Label Propagation* y sobre este se desarrolla uno de ellos.

Ambos pasos de la solución serán convenientemente desarrollados particularmente en la sección 5 con la implementación de dichos algoritmos.

Objetivos 4

4.1. Objetivo general

Este proyecto es una propuesta del grupo de investigación de la Universidad de Burgos, para analizar ramas del aprendizaje semi-supervisado que no se habían contemplado hasta el momento. El objetivo general del presente trabajo es realizar una implementación y validación de algunos métodos de aprendizaje semi-supervisado, centrándose en los ámbitos de grafos y árboles y compararlos con otros algoritmos bien afianzados como RandomForest, Self-Training o Co-Forest.

4.2. Objetivos específicos

1. Realizar una revisión bibliográfica (guiada[1]) para establecer los métodos más prometedores y útiles.

2. Implementar desde cero un algoritmo de construcción de árboles que permita trabajar con datos etiquetados y no etiquetados (semi-supervisado).

3. Implementar desde cero varios algoritmos de construcción de grafos y de *label propagation* pues debido a la naturaleza de los algoritmos basados en grafos, necesitan de dos pasos separados (construcción de grafo y propagación de etiquetas).

4. Seleccionar los conjuntos de datos adecuados para la experimentación de estos algoritmos para una validación exhaustiva que refleje su utilización en muy diversos ámbitos. Al menos, 20 *datasets*.

5. Codificar experimentos adecuados para los algoritmos. Previsiblemente incluirá: preprocesado de datos, codificación de validaciones cruzadas, experimentación de parámetros específicos (influencia) o graficar resultados, entre otros.

6. Comparar los algoritmos desarrollados con algoritmos afianzados del estado del arte para la extracción de resultados y conclusiones.

[1] Gracias a información previa recopilada por el grupo de investigación de la Universidad de Burgos.

En las etapas finales del proyecto, una vez analizada la información obtenida, se consideró en conjunto con el grupo de investigación de la Universidad de Burgos, crear una página web con el propósito de permitir la visualización de los algoritmos de grafos desarrollados. En el Anexo B se muestra la documentación de la aplicación web.

Metodología y Desarrollo

5

En este aparatado se exponen todas las decisiones tomadas para el desarrollo de los algoritmos seleccionados. En las primeras secciones se comentan las decisiones generales que afectan a los algoritmos y a continuación, se desarrollan los algoritmos seleccionados.

5.1. Datasets utilizados

Debido a la naturaleza de este estudio semi-supervisado, los conjuntos de datos típicos que suelen utilizarse no son válidos. La mayor parte de ellos están preparados para el aprendizaje supervisado, donde todos los datos tienen etiquetas.

Una posibilidad es utilizar esos conjuntos de datos "supervisados" y des-etiquetar manualmente los datos para realizar el estudio. Sin embargo, existe una herramienta open-source denominada Keel (Knowledge Extraction based on Evolutionary Learning) **?** en la que, además de la propia herramienta, sus autores incorporaron una serie de datasets semi-supervisados públicos[1]. Es decir, varios conjuntos de datos con algún dato etiquetado y el resto no etiquetados.

El formato que han propuesto es crear cuatro proporciones de etiquetados: 10 %, 20 %, 30 % y 40 %. Se asume que estas proporciones son las de interés puesto que el aprendizaje semi-supervisado surge con la idea de lidiar con **pocos** datos etiquetados. A partir de estas proporciones los métodos supervisados serán ya suficientemente buenos.

Además del trabajo del etiquetado, por cada uno de los datasets, incorporaron la división en *Folds* para la validación cruzada de los métodos. La validación cruzada permitirá controlar la "aleatoriedad" de los experimentos de cara a extraer conclusiones reales. Con solo la extracción del rendimiento en una sola ejecución no sería posible extraer resultados válidos.

Para la experimentación se han seleccionado 24 datasets de Keel, cantidad suficiente para poder extraer conclusiones genéricas y encontrando un compromiso entre el tiempo de ejecución y la efectividad de la experimentación. Los datasets no han sido modificados, aunque sí que ha sido necesario un pequeño preprocesado corrigiendo "errores" de formato.

[1] Es posible acceder a ellos desde la web de la Universidad de Granada: https://sci2s.ugr.es/keel/datasets.php

5.2. Árbol intrínsecamente semi-supervisado (SSLTree)

En la figura 3.4 se comentaba una taxonomía de los algoritmos semi-supervisados. La implementación que se desarrolla a continuación, al no poder se enmarcada en ninguna otra categoría, se trata de un método intrínsecamente semi-supervisado, que por otro lado coincide perfectamente con la definición que se da en **?**.

Un método intrínsecamente semi-supervisado es aquel que no utiliza pasos intermedios utilizando otros algoritmos supervisados. Por ejemplo, el bien conocido Self-Training (*wrapper method*) opera mediante un bucle continuo en el que un algoritmo supervisado es entrenado, predice etiquetas para los no etiquetados y se reentrena con estas. Generalmente, los intrínsecamente semi-supervisados son modificaciones de algoritmos supervisados para poder trabajar con datos no etiquetados en su operación.

Partiendo de esta definición, la idea de este árbol con nombre *SSLTree* (Semi-Supervised Learning Tree) parte de la teoría comentada en 1 en la que se construían árboles de decisión de forma supervisada para clasificación (regresión también aunque no se considera en este desarrollo).

Para realizar una implementación que siga el pseudocódigo de CART y que además permita trabajar con datos no etiquetados es necesario incorporar, de algún modo, el conocimiento que puede proporcionar dichos datos. En otras palabras, los datos no etiquetados puede contener relaciones interesantes que permitan construir árboles más puros (mejores) que simplemente utilizando datos etiquetados.

Para incorporar estos datos no etiquetados, no es necesario modificar la estructura de los árboles que se generan. En la construcción de los árboles, el único momento donde se trabaja con los datos es en la ramificación de un nodo. Utilizando medidas de impureza como *gini* o *entropy* se obtiene una estimación de lo buena que puede ser una de esas divisiones realizadas (cuanto menor valor de estas medidas, más homogéneas son las ramas generadas). Debido a esto, parece tener sentido encontrar alguna forma de incorporar los datos no etiquetados en estas medidas. De hecho, la implementación realizada del algoritmo CART para aprendizaje semi-supervisado no dista mucho del algoritmo original para supervisado.

Durante la revisión bibliográfica se encontraron numerosos artículos que abordaban los árboles semi-supervisados con diferentes aproximaciones. Sin embargo, y como se buscaba, en el artículo **?** proponen un cálculo de impureza **genérico** que incorpora en sus cálculos los datos no etiquetados.

5.2.1. Cálculo de impureza modificado

A partir de este punto, se considera que los datos contienen dos ejes. Tradicionalmente, los árboles se construyen solo considerando el primero de los ejes, el del espacio de etiquetas (*target space*). Adicionalmente para este nuevo cálculo de impureza, se añade el eje de las características o atributos (*descriptive space*).

Los autores **?** proponen el nuevo cálculo considerando la **homogeneidad** de esas ramas

generadas en base a estos dos ejes. La idea de la homogeneidad proviene del *Predictive Clustering* (PC), que considera a los árboles de decisiones como una jerarquía de *clusters*. Por ejemplo, la raíz del árbol es un grupo con todos los datos de entrenamiento, y cuando se crean las dos ramificaciones de la raíz, se generan dos nuevos grupos. Lo que se busca es que estos nuevos grupos sean lo más homogéneos posibles. En aprendizaje supervisado esto suponía que ese grupo contenga la mayor parte de etiquetas de una misma clase.

El objetivo de este desarrollo será aplicar ese cálculo de impureza que se aplicó a los árboles del *Predictive Clustering* para el algoritmo de CART y estudiar su posible efecto beneficioso.

El cálculo de la impureza para aprendizaje supervisado para un conjunto de datos E[2] es (se utiliza Gini para ser fiel a la literatura, pero podrían utilizarse otras medidas):

$$\text{Impurity}(E) = \text{Gini}(E, Y) \tag{5.1}$$

El primer paso para modificar este cálculo es considerar que E ahora contiene datos no etiquetados, esto es: $E = E_l \cup E_u$ donde E_l es la parte de los datos con etiquetas y E_u es la parte de los datos sin etiquetas.

El segundo paso es modificar el cálculo de esa función *Impurity*. Será una suma ponderada de dos impurezas (la del eje de etiquetas y la del eje de características):

$$\text{Impurity}_{\text{SSL}}(E) = \underbrace{w \cdot \text{Impurity}(E_l, Y)}_{\textit{Target space}} + \underbrace{\frac{1-w}{D} \cdot \sum_{i=1}^{D} \text{Impurity}(E, X_i)}_{\textit{Descriptive space}} \tag{5.2}$$

Donde $E = E_l \cup E_u$, D es el número de atributos (en datos tabulares, el número de columnas), X_i es la i-ésima característica (columna) y $w \in [0, 1]$ es un parámetro que controla el peso de cada "eje".

Donde cada función se descompone como:

$$\text{Impurity}(E_l, Y) = \frac{\text{Gini}(E_l, Y)}{\text{Gini}(E_l^{\text{train}}, Y)} \tag{5.3}$$

$$\text{Impurity}(E, X_i) = \frac{\text{Var}(E, X_i)}{\text{Var}(E_{\text{train}}, X_i)} \tag{5.4}$$

La variabilidad se calcula como:

$$\text{Var}(E, X_i) = \frac{\sum_{j=1}^{N} (x_i^j)^2 - \frac{1}{N} \cdot (\sum_{j=1}^{N} x_i^j)^2}{N} \tag{5.5}$$

Es importante aclarar que cuando se hace referencia a E^{train}, representa todo el conjunto de entrenamiento. A diferencia de E, que representa el conjunto de datos del nodo en el que se está calculando la medida.

[2] Y es la variable de etiquetas.

La idea es que los denominadores que tienen E^{train} sirvan como normalización para que ambos términos contribuyan de igual manera (Gini podría dar valores mucho más grandes o pequeños que las variabilidades, o viceversa). Es decir, los cálculos con E^{train} serán cálculos para todo el conjunto de datos.

Es conveniente aclarar que aunque en ciertos conjuntos de datos las características sean categóricas, la implementación realizada de CART solo admite características numéricas, esto supone también una simplificación en el cálculo de la ecuación 5.4. En el artículo original **?** la ecuación 5.4 se calcula de distinta forma si las características son nominales. En este trabajo, esto no se ha considerado.

El parámetro w resulta ser el más importante en todo el cálculo. Controla la cantidad de supervisión del método. Es decir, controla cuánto se tiene en cuenta la parte de las etiquetas contra la parte de las características. Con $w = 1$ sería como si el árbol fuera completamente supervisado, con $w = 0$ sería como si fuera completamente no supervisado. La clave está en encontrar un valor para w que pueda tener en cuenta las etiquetas pero incorporar también el "conocimiento" que proporciona la variabilidad de las características.

5.2.2. Intuición del nuevo cálculo

Antes de estudiar la influencia del parámetro w se pretende explicar el nuevo término de la variabilidad.

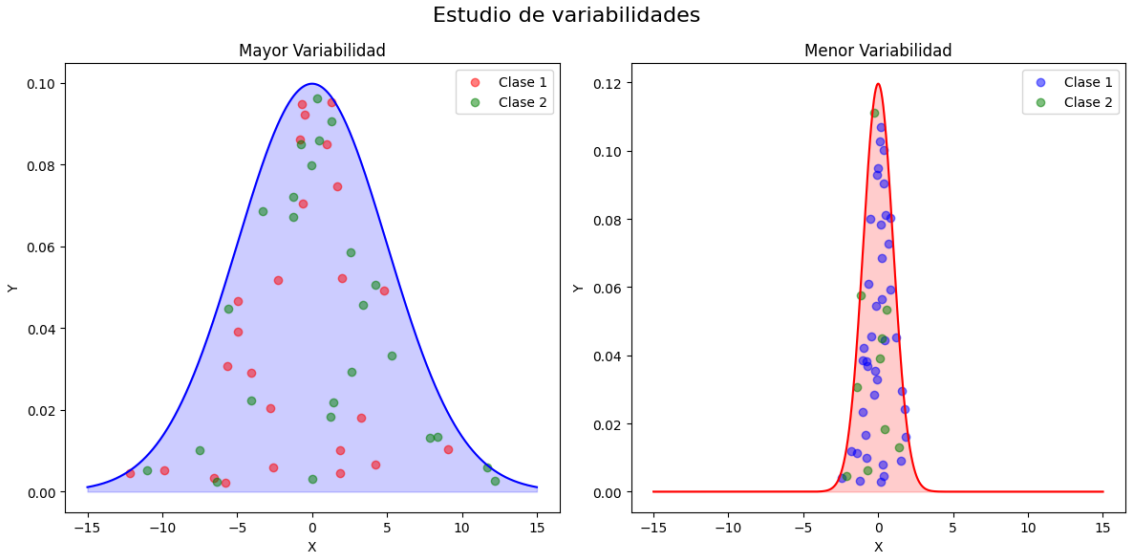

Figura 5.1: ***Suposición de la variabilidad. No representa un ejemplo real, solo es ilustrativo.***

Como se ha comentado, el uso de la impureza mediante Gini o Shannon pretende minimizar su cálculo para conseguir mayor homogeneidad (por eso se denomina impureza, lo que interesa es la pureza). Este cálculo se realiza considerando la proporción de las etiquetas en la división de los datos. Aquella partición que de un menor valor, será la decisión tomada en el nodo concreto.

Para considerar los datos no etiquetados, obviamente no se puede utilizar esas funciones de impureza, hay que, de alguna forma, extraer información de su variabilidad.

La intuición de este término es que, si existe una variabilidad "grande" en las características, es de suponer que si los datos fueran etiquetados, las proporciones de las etiquetas serían más bien parecidas (no homogéneas). Es decir, algo como el gráfico de la izquierda en la figura 5.1, donde las etiquetas parecen tener una proporción muy similar.

Por el contrario, si la variabilidad de las características es menor, es previsible que las etiquetas que tienen esos datos sean más homogéneas. Algo parecido al gráfico de la derecha en la figura 5.1. Esto es porque **datos similares en las características** tienden a tener las **mismas etiquetas** (muy relacionado con las suposiciones en 3.1.3.1).

Como se ha comentado, esto solo es una suposición (al igual que las del aprendizaje semi-supervisado) y por lo tanto no se cumplirá en todos los casos. Podría ocurrir algo similar a la figura 5.2, donde claramente hay poca homogeneidad y la clasificación no sería buena.

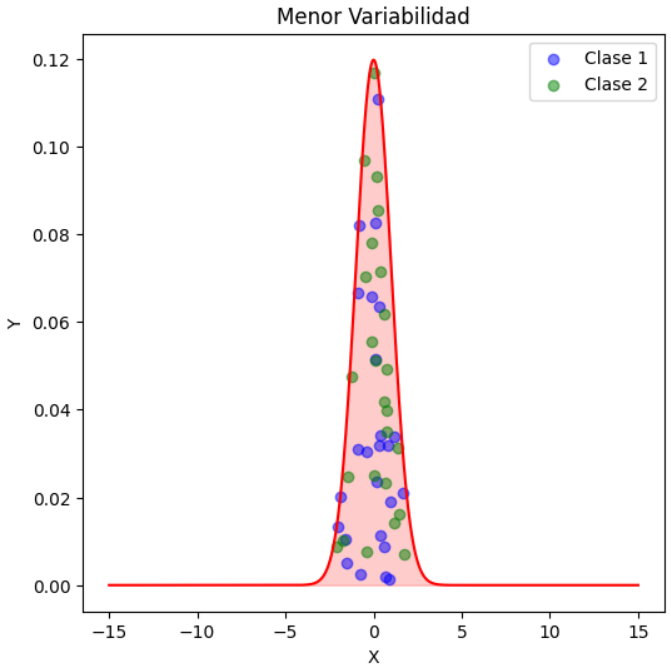

Figura 5.2: ***Incumplimiento de la suposición de la variabilidad.***

En definitiva, se intenta conseguir la homogeneidad en las etiquetas conocidas y en las características. Aunando ambas aportaciones se ha demostrado que es posible mejorar el rendimiento de métodos supervisados. Se verá reflejado en la experimentación.

Criterio supervisado utilizado

En la implementación de SSLTree se han incluido tanto Gini como Entropy. Sin embargo, para esta documentación se ha elegido arbitrariamente hacer referencia al cálculo mediante Entropy. Los resultados obtenidos, además de muy similares, derivan en las mismas conclusiones. Algunos de los experimentos con Gini se añadirán, por completitud, en el anexo A.
Por lo tanto, a partir de este apartado, las experimentaciones contendrán gráficos obtenidos al utilizar Entropy como criterio.

5.2.3. Influencia del parámetro w

El parámetro w es el que proporciona la importancia a cada eje del nuevo cálculo. Siguiendo la suposición comentada en el apartado anterior, el método resulta muy dependiente del caso concreto con el que se trabaja. Es posible que tener más en cuenta la variabilidad que las etiquetas sea mejor, o viceversa.

Se realiza un estudio exhaustivo para todos los conjuntos de datos, donde se verá la influencia de este parámetro. Para este análisis se ha obtenido el rendimiento (*accuracy*) del modelo para cada conjunto de datos y para cada posible valor de w entre 0 y 1, con pasos de 0.1.

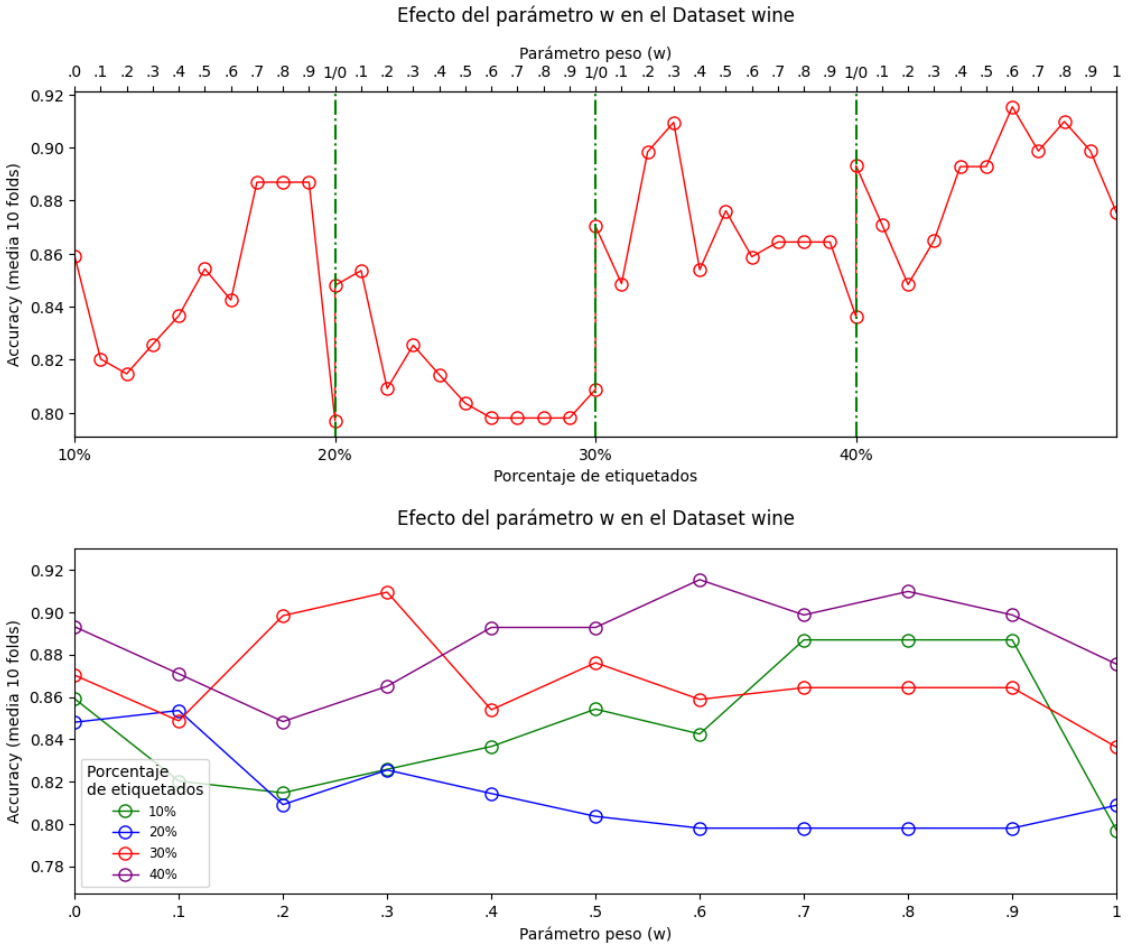

Figura 5.3: ***Ejemplo del efecto de w en el dataset Wine.***

Con independencia del criterio utilizado, es muy claro que dependiendo del valor de w, el comportamiento del modelo se ve alterado. Esto es precisamente lo que se está buscando, si los gráficos fueran planos o con poca alteración, no se podría evaluar el efecto.

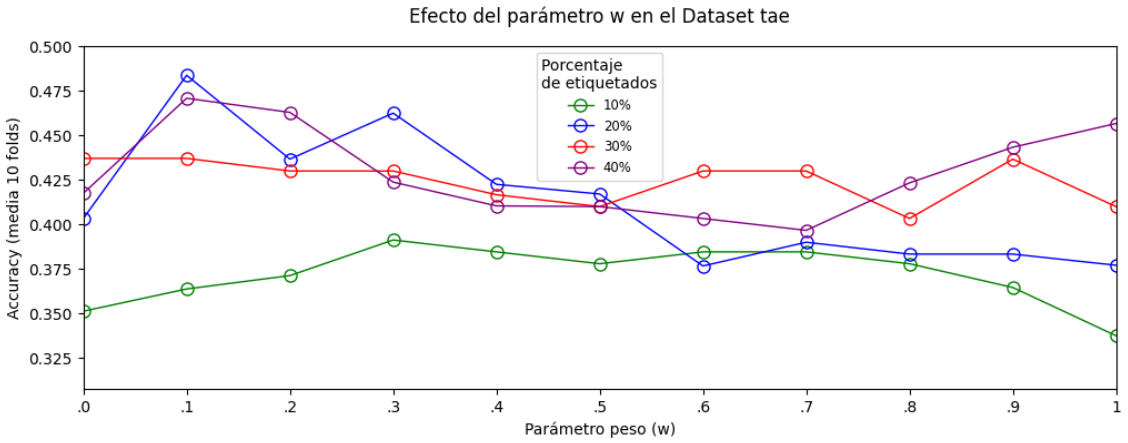

Figura 5.4: ***Ejemplo del efecto de w en el dataset Tae.***

En esta última imagen 5.4 es mucho más interesante ver que, si se quisiera aplicar el modelo a este conjunto de datos, se podrían seleccionar varios valores de w. Quizás sería interesante aplicar un valor $w = 0{,}3$ si se conoce que hay muy pocos datos etiquetados (10 %) o $w = 0{,}1$ si hay más. Lo importante es que se obtienen mejores resultados (en general) para valores menores que $w = 1$, lo que indica que si se aplica un modelo supervisado (i.e $w = 1$), sería peor (no en todos los casos y bajo las suposiciones del semi-supervisado).

En definitiva, estos resultados indican dos conclusiones:

- El parámetro w es dependiente del conjunto de datos.
- El parámetro w es beneficioso y **podría** mejorar el rendimiento del modelo.

5.2.4. Post-poda del árbol

El concepto de poda, por su nombre, consiste en eliminar subárboles del árbol original convirtiendo la raíz de ese subárbol en una hoja.

La motivación de implementar un algoritmo de poda es que en **?** se realiza un estudio comparativo de métodos de poda para árboles de decisión. La conclusión del estudio fue que aplicar la poda **puede** mejorar el *accuracy* hasta en un 25 %. Por lo que, aplicar la poda a un árbol permite, en algunos casos, mejorar su rendimiento.

Al igual que el parámetro w, en toda la experimentación posterior en 6.1 no se aplica el algoritmo de post-poda a ningún árbol (y por esta razón se encuentra en este apartado). Sería una situación injusta para el resto de modelos con los que se compara.

Para *SSLTree* se ha implementado un algoritmo de post-poda conocido como *Cost-Complexity Pruning* de **?**.

En *cost-Complexity pruning* se desea optimizar la función *cost-complexity*:

$$R_\alpha(T) = R(T) + \alpha \cdot |f(T)| \tag{5.6}$$

donde $R(T)$ es el error de entrenamiento 5.7, $f(T)$ devuelve el conjunto de hojas del árbol T y α es el parámetro de regularización (hiper-parámetro fijado).

$$R(T) = \sum_{t \in f(T)} r(t) \cdot p(t) = \sum_{t \in f(T)} R(t) \tag{5.7}$$

$\sum_{t \in f(T)} R(t)$ *es la suma de los errores de clasificación en cada hoja*

$r(t) = 1 - \max_k p(C_k \mid t)$ *error de clasificación*

$p(t) = \frac{n(t)}{n}$ *n(t) es el número de ejemplos en el nodo t y n el total de ejemplos*

Para entender el funcionamiento de este algoritmo se supone un árbol original T y un subárbol de él T_t (t es un nodo de T). La variación de la función *cost-complexity* del árbol resultante de podar T por T_t (es decir, $T - T_t$) es:

$$R_\alpha(T - T_t) - R_\alpha(T)$$

Como la función *cost-complexity* tiene en cuenta el error (*cost*) y la complejidad (número de hojas), lo que interesa es que la variación sea negativa. Es decir, pasar de un árbol más complejo y errático a otro mejor. El parámetro α es el punto de comparación para determinar si un árbol podado es mejor.

Al desarrollar la ecuación 5.7 con el árbol podado, la variación queda delimitada por un α' (no es el parámetro original):

$$R_\alpha(T-T_t) - R_\alpha(T) = R(T-T_t) - R(T) + \alpha(|f(T-T_t)| - |f(T)|) = R(t) - R(T_t) + \alpha(1 - |f(T_t)|)$$

$$\alpha' = \frac{R(t) - R(T_t)}{|f(T_t)| - 1}$$

Los valores de dicha variación son:

- Nula si $\alpha = \alpha'$
- Negativa si $\alpha < \alpha'$
- Positiva si $\alpha > \alpha'$

Dado un α, un árbol original T y un árbol podado $T - T_t$, al calcular α' mostrará si ese nuevo árbol es menos complejo y errático que el original. Se puede entender cómo si se trabajara en términos relativos de α. Para otro valor de α como hiper-parámetro, un árbol que antes era mejor, podría ser peor, o viceversa.

El algoritmo 2 muestra el proceso de generación de las posibles podas y la selección del mejor árbol podado según el hiperparámetro α. Se trata de una interpretación de la explicación teórica de ML Wiki[3]. En la figura 5.5 puede verse el efecto beneficioso al aplicarse el algoritmo de post-poda en el dataset Breast Cancer para los distintos valores de α que el algoritmo evaluó. Encontrando el valor α adecuado pueden obtenerse mejores resultados en el test al reducir el sobreajuste del modelo.

[3]Creada por Alexey Grigorev, puede accederse desde http://mlwiki.org/index.php/Cost-Complexity_Pruning

Algoritmo 2: Cost-complexity Pruning

1. Inicialización:
   a) Dejar que T^1 sea el árbol obtenido con $\alpha^1 = 0$ (árbol original).
2. Repetir los siguientes pasos para cada i hasta que T^i sea la raíz:
   a) Paso 1:
      1) Seleccionar el nodo $t \in T^1$ que minimiza
      $$g_1(t) = \frac{R(t) - R(T_t^1)}{|f(T_t^1)| - 1}$$
      2) Dejar que t_1 sea este nodo.
      3) Dejar que $\alpha^2 = g_1(t_1)$ y $T^2 = T^1 - T_{t_1}^1$.
   b) Paso i:
      1) Seleccionar el nodo $t \in T^i$ que minimiza
      $$g_i(t) = \frac{R(t) - R(T_t^i)}{|f(T_t^i)| - 1}$$
      2) Dejar que t_i sea este nodo.
      3) Dejar que $\alpha^{i+1} = g_i(t_i)$ y $T^{i+1} = T^i - T_{t_i}^i$.
3. Devolver la secuencia de árboles $T^1 \supset T^2 \supset \ldots \supset T^k \supset \ldots \supset \{\text{raíz}\}$ y la secuencia de parámetros $\alpha^1 \leq \alpha^2 \leq \ldots \leq \alpha^k \leq \ldots$.
4. Seleccionar el T^{k+1} árbol tal que $\alpha \in [\alpha^k, \alpha^{k+1})$. Se asume la existencia de un $\alpha^0 = 0$ que permita seleccionar T^1.

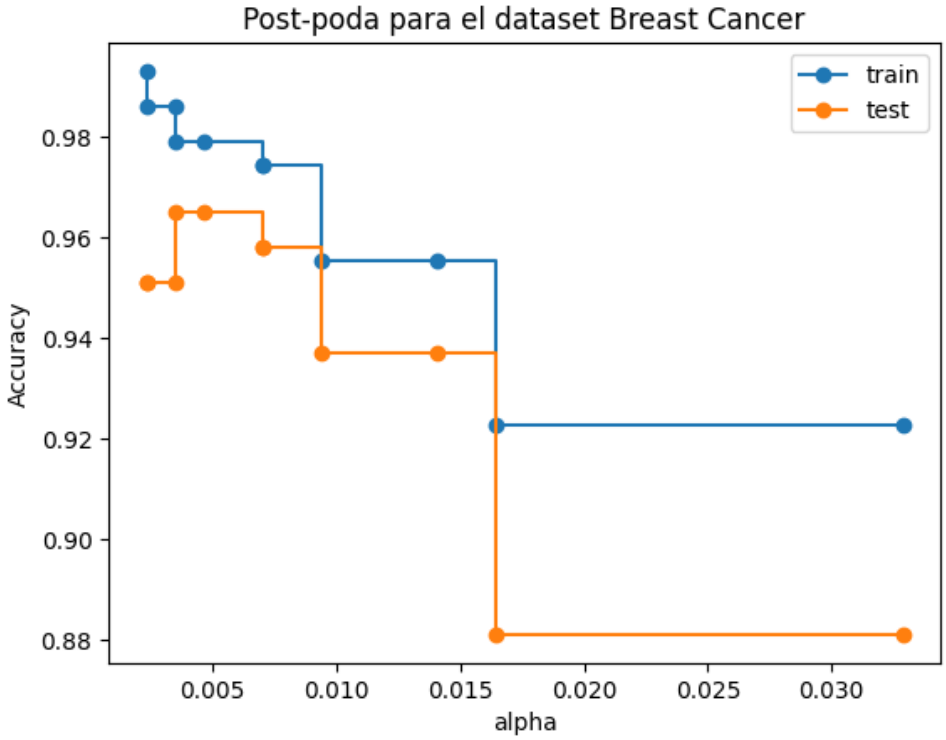

Figura 5.5: ***Beneficio de la post-poda en SSLTree para el dataset Breast Cancer.***

5.3. Grafos semi-supervisados

5.3.1. GBILI (Graph-based on informativeness of labeled instances)

Los métodos de construcción de grafos suelen ser, por lo general, no supervisados. Esto hace que solo se utilice la información de similitud (como lo hace el algoritmo kNN o derivados). Sin embargo, los autores de *Graph-based on informativeness of labeled instances* (GBILI) **?** parten del hecho de que utilizar la información de las etiquetas puede mejorar el grafo construido (i.e el proceso de inferencia posterior).

El algoritmo GBILI construye grafos no dirigidos y se basa en ciertas premisas. En primer lugar, las conexiones están influenciadas por los *k vecinos más cercanos mutuos*, es decir, para un punto a, un vecino más cercano b es mutuo si además de ser uno de lo k más cercanos, a también es un k vecino más cercano de b. Esto previene juntar subgrafos que sean de diferentes grupos.

$$\text{kNN-Mutuo: } b \in \text{kNN}(a) \wedge a \in \text{kNN}(b) \tag{5.8}$$

La otra premisa es que las conexiones también están influenciadas por las etiquetas de los vértices. Esto prioriza que se formen conexiones entre vértices cercanos a un punto etiquetado. Para ello se minimiza la ecuación 5.9.

$$\text{mín} \sum_i \sum_j \left(D_{ij} + \sum_l D_{jl} \right) \tag{5.9}$$

La intuición de la ecuación 5.9 es comparar distintos puntos i y j de tal forma que si la distancia entre ellos, junto a la distancia de ese j a un punto etiquetado es mínima, entonces habrá una conexión $i - j$ para que i aproveche la información del etiquetado a través de j. Con estas dos premisas, todo el proceso puede verse en el pseudocódigo 3.

En las líneas 5 a 8 se obtienen los vecinos más cercanos de cada vértice (nodo). A continuación, en las líneas 9 a 12 se obtienen los vecinos más cercanos mutuos (ver 5.8). En la líneas 12 a 16 se minimiza la ecuación 5.9 conectando vecinos que estén cerca de puntos etiquetados. Este paso genera componentes en el grafo, que, o tiene algún ejemplo etiquetado o ninguno. En este último caso, no se puede aprovechar la información cercana de ejemplos etiquetados. En la línea 17 se localizan esos componentes mediante una búsqueda en anchura en el grafo construido a partir del paso anterior. En las líneas 18 a 22 se conectan esas componentes aisladas que no tienen un ejemplo etiquetado con otra componente que sí lo tenga, para ello también se utilizan los vecinos más cercanos (no se unen aleatoriamente, siempre se utilizan nodos cercanos).

5.3.2. RGCLI (Robust Graph that Considers Labeled Instances)

Los métodos semi-supervisados se benefician de las suposiciones de clúster y *manifold* pues generalizan todas las demás. Estas indicaban que aquellos puntos que estén cerca, que

Algoritmo 3: GBILI

```
generar una matriz de distancias D
generar una lista de puntos etiquetados L
establecer el parámetro K
for i = 1; i < |V|; i++ do
    for k = 1; k < K; k++ do
        for j = 1; j < |V|; j++ do
            if D(v_i, v_j) es el k-ésimo vecino más cercano then
                Guardar v_j en la Lista-kNN(v_i)
    for j = 1; j < Lista-kNN(v_i); j++ do
        for k = 1; k < K; k++ do
            if D(v_j, v_i) es el k-ésimo vecino más cercano then
                Guardar v_i en la M-kNN(v_j)
    for j = 1; j < M-kNN(v_i); j++ do
        for l = 1; l < |L|; l++ do
            if D(v_i, v_j) + D(v_j, v_l) es mínimo then
                Guardar e_ij en G
Realizar BFS y devolver Componente(G)
for i = 1; i < |V|; i++ do
    if Componente(v_i) ∉ L then
        for k = 1; k < Lista-kNN(v_i); k++ do
            if Componente(v_k) ∈ L then
                Guardar e_ik en G
devolver G
```

pertenezcan al mismo grupo, clúster o *manifold*, tendrán la misma etiqueta. En el caso de GBILI, aunque era prometedor, sus autores no consiguieron demostrar que se ajustara a las suposiciones comentadas. *Robust Graph that Considers Labeled Instan* (RGCLI) **?** supone una mejora de GBILI con la que sus autores demuestran que cumple con las suposiciones.

Un método tradicional de kNN podría cumplir la suposición local de puntos que estén cerca, pero para un valor grande de k enlazaría con puntos que pertenecen a otros grupos. RGCLI utiliza los vecinos más cercanos mutuos para establecer conexiones. Estos vecinos se seleccionan por su cercanía a puntos etiquetados y como también se limita el número de conexiones a unas pocas (fijadas por parámetro), permite cumplir las suposiciones establecidas. En el pseudocódigo 4 puede verse el proceso completo.

Primero se inicializan las variables que almacenan la información del grafo y de vecindad. RGCLI se basa en dos procedimientos de búsqueda:

- *SearchKNN*: Se establecen los k_e vecinos más cercanos de todos los puntos así como el más lejano y el más cercano etiquetado.

- *SearchRGCLI*: Con la información de vecindad obtenida, este paso crea el grafo final. Para ello minimiza una puntuación o *score* de los enlaces. Para generar un posible enlace se consideran los vecinos más cercanos (j) de cada vértice (i). La idea más interesante es la condición impuesta para considerar un posible enlace. Se evalúa si la

Algoritmo 4: RGCLI

Input: número de vecinos más cercanos k_e, número de vecinos RGCLI k_i, lista de ejemplos etiquetados L, conjunto de datos X, número de hilos nt
Output: GL
$V \leftarrow$ crear un conjunto de vértices a partir de X
$E, W \leftarrow \emptyset$
$GL \leftarrow (V, E, W)$
$kdtree \leftarrow$ crear un Kd-tree a partir de X
$kNN \leftarrow$ dict
$\mathcal{F} \leftarrow$ dict
$\mathcal{L} \leftarrow$ dict
$\mathcal{T} \leftarrow \{T_i : \cup_{i=1}^{nt} T_i = V, \cap_{i=1}^{nt} T_i = \emptyset\}$
for $T_i \in \mathcal{T}$ **do**
    $t \leftarrow$ Thread(SearchKNN($T_i, k_e, kdtree, kNN, \mathcal{L}$))
    t.start()
for $T_i \in \mathcal{T}$ **do**
    $t \leftarrow$ Thread(SearchRGCLI($GL, T_i, k_i, kNN, \mathcal{L}$))
    t.start()
Procedure SearchKNN(T, k_e, $kdtree$, kNN, $\mathcal{L}$)
    for vértice $v_i \in T$ **do**
        $kNN[v_i] \leftarrow$ kdtree.query(v_i, k_e)
        $\mathcal{L}[v_i] \leftarrow$ encontrar puntos etiquetados más cercanos en $\mathcal{L}$
        $\mathcal{F}[v_i] \leftarrow$ encontrar el k-ésimo vecino más lejano de v_i
End Procedure
Procedure SearchRGCLI(GL, T, k_i, kNN, $\mathcal{L}$)
    for vértice $v_i \in T$ **do**
        $\mathcal{E} \leftarrow$ dict
        for vértice $v_j \in kNN[v_i]$ **do**
            if dist$(v_i, v_j) \leq$ dist$(v_j, \mathcal{F}[v_j])$ **then**
                $e \leftarrow (v_i, v_j)$
                $\mathcal{E}[e] \leftarrow$ dist(v_i, v_j) + dist$(v_j, \mathcal{L}[v_j])$
        $E^* \leftarrow$ obtener k_i aristas con menor puntuación de $\mathcal{E}$
        $E \leftarrow E^* \cup E$
        $w(e) \leftarrow 1 \quad \forall\, e = (v_i, v_j) \in E^*$
End Procedure

distancia ente (i) y (j) es menor que la de (j) con el más lejano de (j). De esta forma es improbable crear enlaces con puntos lejanos. Si se cumple esta condición, se calcula su puntuación como la suma de distancias de (i) con (j) y (j) con su vecino más cercano etiquetado. Una vez considerados todos los vecinos cercanos (j) de (i), se almacenan los k_i enlaces con menor puntuación[4] (y se continúa con el siguiente vértice).

[4]Como la puntuación está basada en la distancia, cuanto menor puntuación, más prometedor podría ser un enlace.

5.3.3. LGC (Local and Global Consistency)

Como se ha comentado, el aprendizaje semi-supervisado se basa en dos suposiciones clave: puntos cercanos tendrán la misma etiqueta, y puntos que pertenezcan al mismo clúster o *manifold*, también. En el primer caso la suposición es local y en el segundo es global. La primera de las suposiciones es fácil de cumplir y muchos algoritmos supervisados lo consiguen (como kNN). El método *Local and Global Consistency* (LGC) **?** trata de propagar las etiquetas de forma iterativa a los vecinos de forma suave, es decir, sin cambios bruscos (entre dos puntos cercanos no debería haber un cambio de etiqueta) para cumplir las suposiciones local y global al mismo tiempo.

Se parte de un conjunto de datos $\mathcal{X} = \{x_1, \ldots, x_l, x_{l+1}, \ldots, x_n\} \subseteq \mathbb{R}^m$ y un conjunto de etiquetas $\mathcal{L} = \{1, \ldots, c\}$. Los l primeros puntos x_i $(i \leq l)$ tienen etiquetas $y_i \in \mathcal{L}$ y el resto de puntos $x_u (l + 1 \leq u \leq n)$ no tienen etiquetas.

Existe un conjunto $\mathcal{F}$ de matrices $n \times c$. Una matriz $F = [F_1^T, \ldots, F_n^T]^T \in \mathcal{F}$ es la clasificación de un conjunto de datos $\mathcal{X}$ tomando como $y_i = arg\,max_{j \leq c} F_{ij}$. Es decir, F se puede entender como una función $\mathcal{X} \rightarrow \mathbb{R}^c$ que para un x_i devuelve un vector F_i.

$Y \in \mathcal{F}$ también será un matriz del mismo estilo en el si $Y_{ij} = 1$ entonces x_i tiene etiqueta $y_i = j$ y $Y_{ij} = 0$ en caso contrario.

Como ejemplificación de lo anterior, en la tabla 5.3.3 se puede ver un ejemplo de una matriz $M \in \mathcal{F}$. Por ejemplo, si se quiere conocer la etiqueta de x_3, se retornaría un vector $[0, 0, 1]$. Su etiqueta sería $y_i = j = 2$ pues la posición $M_{32} = 1$ (la etiqueta es obtenida con el $arg\,max_{j \leq c} M_{ij}$).

Con estas definiciones, en el pseudocódigo 5 se puede ver el proceso iterativo de inferencia.

	j = 0	**j = 1**	**j = 2**
$\mathbf{x_0}$	0	1	0
$\mathbf{x_1}$	1	0	0
$\mathbf{x_2}$	0	1	0
$\mathbf{x_3}$	0	0	1

Tabla 5.1: *Ejemplo de una matriz de etiquetado.*

Algoritmo 5: Local and Global Consistency

1: Construir la matriz de similitud W.
2: Construir la matriz $S = D^{-1/2} W D^{-1/2}$ donde D es una matriz diagonal con su elemento (i, i) igual a la suma de la fila i-ésima de W.
3: Iterar $F(t + 1) = \alpha S F(t) + (1 - \alpha) Y$ hasta la convergencia, donde α es un parámetro en $(0, 1)$.
4: Asumir un F^* como el límite de la secuencia $\{F(t)\}$. Etiquetar cada punto x_i con etiqueta $y_i = \arg \max_{j \leq c} F^*_{ij}$.

El primer paso es construir la matriz de afinidad, que generalmente se construye como $W_{ij} = exp(-||x_i - x_j||^2 / 2\sigma^2)$ si $i \neq j$ y $W_{ii} = 0$. Sin embargo, los métodos de construcción de

grafos generalmente suelen establecer, sin pérdida de generalidad, que si existe un enlace entre i y j entonces $W_{ij} = 1$ (y $W_{ji} = 1$) simplemente.

En el siguiente paso, esta matriz W es normalizada de forma simétrica pues el método diseñado lo requiere para converger.

En el tercer paso ocurre el proceso iterativo. Este consiste en utilizar la información de los vecinos (término $\alpha SF(t)$) e información inicial (término $(1 - \alpha)Y$). El parámetro α determina la cantidad de información que se tiene en cuenta de cada término.

Al final del proceso iterativo (límite de la secuencia realizada), se obtiene una matriz similar a la de la tabla 5.3.3 que permite realizar el etiquetado de los x_u (no etiquetados), aunque en realidad el proceso se realiza para todos los $x \in \mathcal{X}$.

Resultados y Discusión 6

En esta sección se describen los resultados de la experimentación de los métodos desarrollados en la sección anterior.

6.1. Experimentación *SSLTree*

Código de la experimentación

Toda la codificación de los experimentos realizados puede encontrarse en https://github.com/dmacha27/TFM-VIU/tree/main/metodos/SSLTree/experimentos.

Para comprobar su funcionamiento, se han seleccionado dos métodos con una comparación directa a *SSLTree*. Estos son *DecisionTreeClassifier* y *SelfTrainingClassifier*, ambos implementados en la librería *Scikit-Learn*.

En el caso de *DecisionTreeClassifier*, es un buen modelo de referencia por ser la implementación clásica de **CART** para aprendizaje **supervisado**. La idea es que *SSLTree* debería ser mejor para algunos conjuntos de datos y para el resto, al menos, no introducir ruido y tener un rendimiento muy similar. Para el caso de *SelfTrainingClassifier*, este es un *wrapper method* que envuelve un modelo supervisado, que también será *DecisionTreeClassifier*. SelfTraining es considerado el método más sencillo del aprendizaje semi-supervisado y, aunque sea capaz de trabajar con datos no etiquetados, el modelo SSLTree debería ser mejor que él por su naturaleza intrínseca.

Antes se ha analizado la importancia y la influencia que puede tener el parámetro w. Los métodos que se van a comparar poseen parámetros por defecto que habrán sido probados y seleccionados acordemente. Para el modelo SSLTree es necesario fijar este parámetro w.

Podría estudiarse el parámetro w por cada conjunto de datos. Sin embargo, a la hora de realizar la comparación, no sería justo. Se deberían ajustar los hiper-parámetros del resto de modelos también para cada conjunto de datos.

Para una comparación justa, el estudio de w se realizará para todos los datasets seleccionados y, a partir de los resultados, se seleccionará un valor que obtenga un compromiso para cada porcentaje de etiquetado. La idea es que este valor se convierta en el valor por defecto de SSLTree.

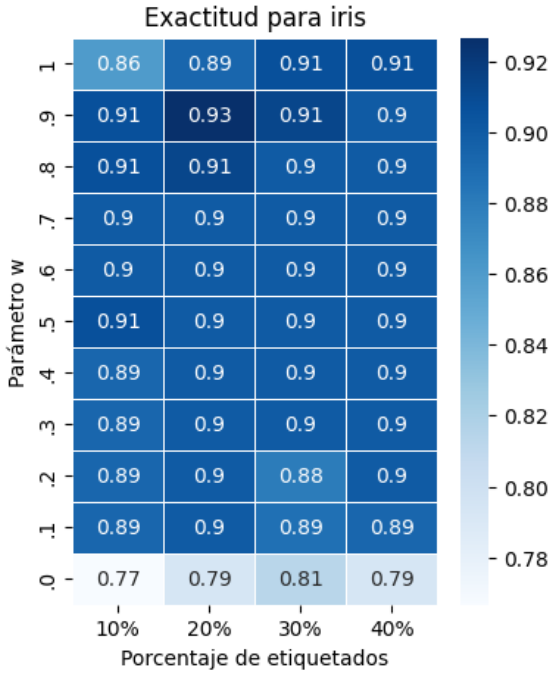

Figura 6.1: ***Mapa de calor de w para el dataset Iris.***

La figura 6.1 representa los resultados de la evaluación de cada posible valor de w para cada proporción de etiquetados (similar a los gráficos anteriores) en forma de mapa de calor.

Para elegir cual es el parámetro w adecuado se han obtenido esos mapas de calor por cada conjunto de datos. A partir de esos cálculos se realiza el ranking medio por cada porcentaje de etiquetado.

Pasos del ranking medio:

1. Se calcula el ranking por columna (porcentaje de etiquetado) por cada conjunto de datos. Se obtendrán otras 24 matrices con dichos rankings.

2. Se realiza el promedio de cada celda en todos los conjuntos de datos. Por ejemplo, para el 10 % y $w = 1$ se calcula el promedio de esa posición a lo largo de las 24 matrices (conjuntos de datos).

El resultado será el de la figura 6.2. Un resultado muy similar con el cálculo de Gini puede encontrarse en A.1.

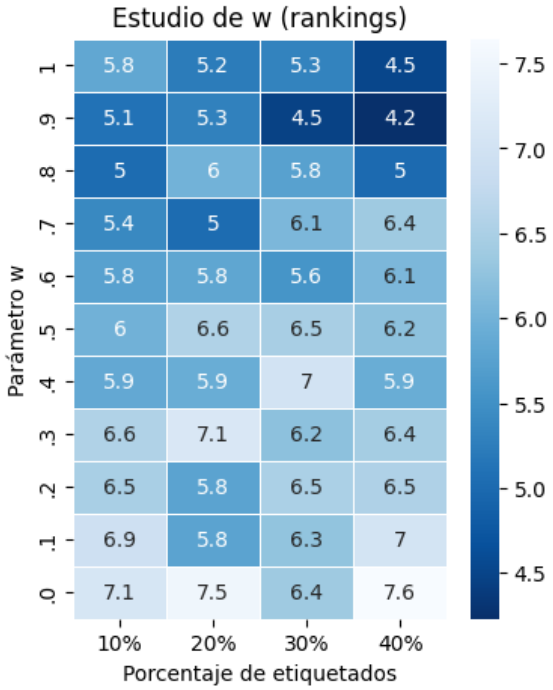

Figura 6.2: ***Mapa de calor de los rankings medios de*** w***.***

La interpretación de ese resultado es que, por columnas, cada celda representa el ranking promedio que esa combinación de porcentaje y valor de w ha tenido en los 24 datasets. Por ejemplo, el 30 % y $w = 0{,}9$, en promedio, ocupa el ranking 4.5 en todos los datasets (cuanto menor valor de ranking, mejor).

Recurriendo de nuevo a un promedio es posible ver cuál es el ranking que ocupa cada valor de w:

1	**.9**	.8	.7	.6	.5	.4	.3	.2	.1	0
5.22	**4.77**	5.44	5.72	5.80	6.31	6.17	6.59	6.32	6.5	7.15

Tabla 6.1: *Ranking promedio de cada valor de* w

Con estos resultados, el mejor valor de w es 0.9. Consigue, en promedio, obtener los mejores resultados en todos los porcentajes de etiquetados.

Continuando con la comprobación del modelo, se realiza la comparación de SSLTree con *DecisionTreeClassifier* y *SelfTrainingClassifier*. De nuevo, para cada conjunto de datos se ha obtenido el rendimiento (exactitud) para cada porcentaje de etiquetado (ver figura 6.3).

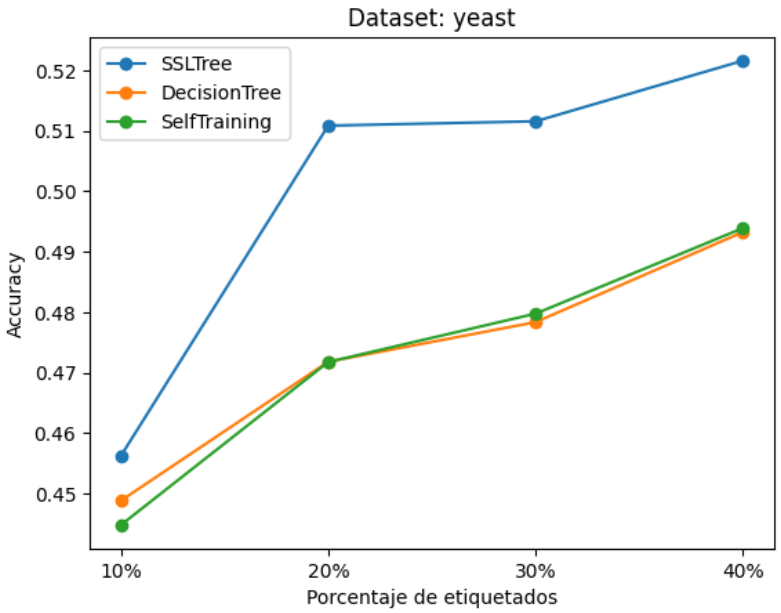

Figura 6.3: ***Comparativa en dataset Yeast.***

La figura 6.3 se ha seleccionado convenientemente. Existirán conjuntos de datos en los que SSLTree sea igual o mínimamente peor cuando no se cumplan las suposiciones comentadas o no se tiene la información suficiente.

Para compactar los resultados de todos los conjuntos de datos, se realiza otro ranking promedio. Es decir, para cada conjunto de datos se hace el ranking de los tres modelos por cada porcentaje y se realiza después el promedio.

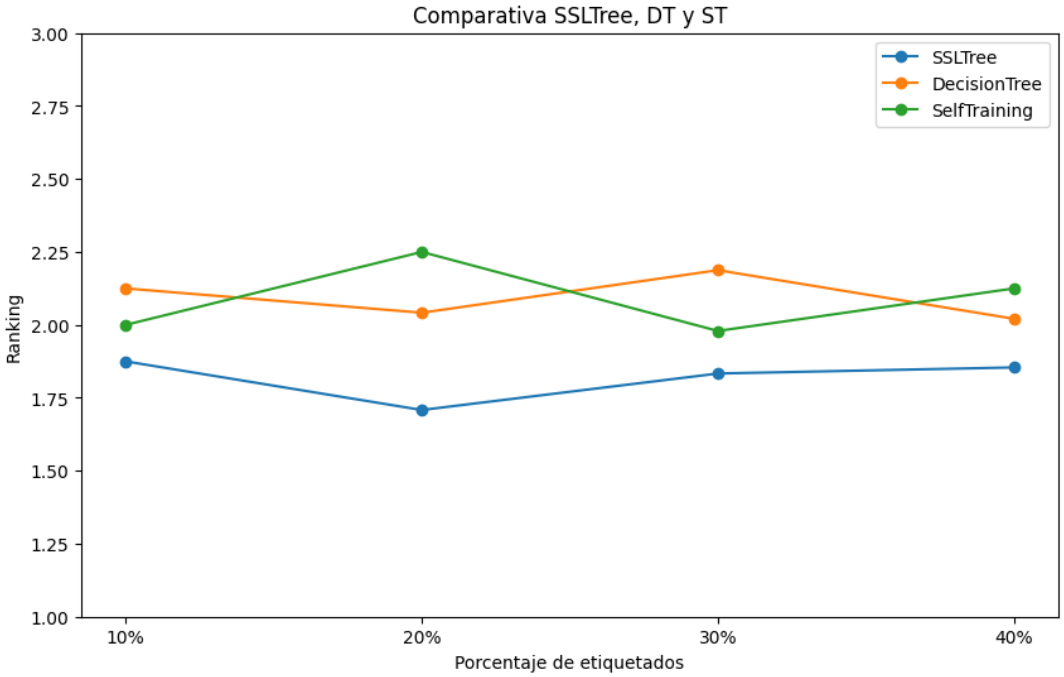

Figura 6.4: ***Ranking promedio de cada modelo para todos los datasets.***

Los resultados obtenidos[1] indican que la implementación de SSLTree funciona correctamente y en general obtiene mejores resultados que el resto de modelos.

Aunque empíricamente ofrezca mejores resultados, es posible que la diferencia entre los modelos no sea significativa. Si un modelo es mejor que otro por pequeñas diferencias, pueden considerarse similares.

Para realizar este estudio, se obtienen los resultados de un test de Nemenyi. El test de Nemenyi permite determinar si grupos de datos (en este caso medidas de exactitud) son diferentes estadísticamente. A partir de esos resultados (p-valores), pueden representarse las diferencias críticas. Los resultados representados para la comparativa anterior pueden verse en las figuras 6.5, 6.6, 6.7 y 6.8. Las líneas horizontales unen los grupos que no tienen diferencias significativas.

En todas ellas, SSLTree es el mejor modelo, sin embargo, las diferencias no parecen ser significativas para considerarlo como un modelo mucho mejor que el resto.

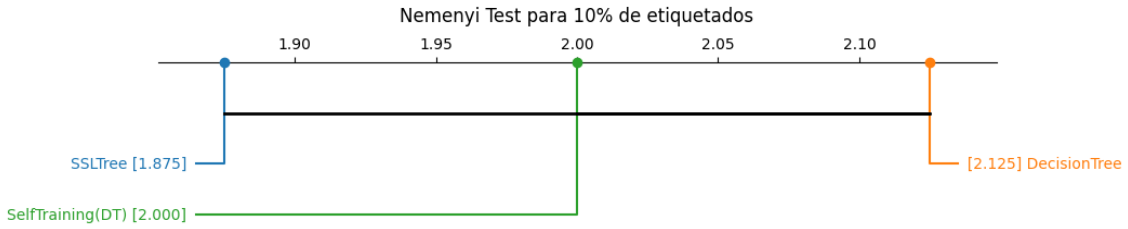

Figura 6.5: ***Comparativa básica: Nemenyi Test para 10 % de etiquetados.***

[1] Pueden comprobarse resultados similares para Gini en 6.4.

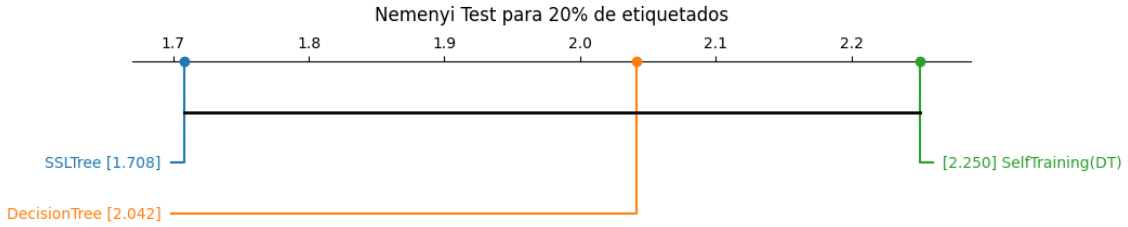

Figura 6.6: ***Comparativa básica: Nemenyi Test para 20 % de etiquetados.***

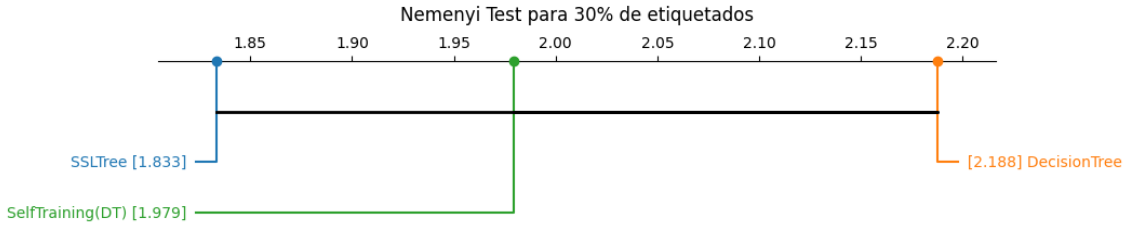

Figura 6.7: ***Comparativa básica: Nemenyi Test para 30 % de etiquetados.***

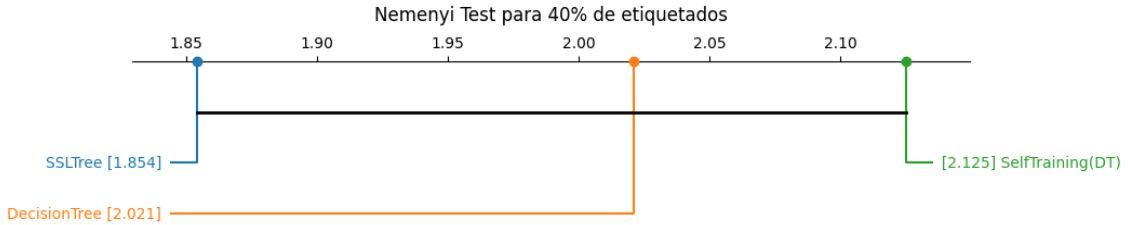

Figura 6.8: ***Comparativa básica: Nemenyi Test para 40 % de etiquetados.***

Aunque estos resultados supongan que *SSLTree* no es un modelo significativamente mejor que el resto, es un resultado muy prometedor. En las conclusiones se analizará el alcance que supone disponer de un modelo basado en árboles semi-supervisado.

A continuación, partiendo de que SSLTree no es un modelo peor, se va a estudiar el funcionamiento de varios *ensembles*. Un *ensemble* trata ponderar varios clasificadores individuales (opiniones individuales) y combinarlos para obtener un clasificador que supere a todos ellos (decisión final) **?**.

Para esta comparativa final se van a utilizar los siguientes *ensembles*:

1. Random Forest con SSLTree como estimador base. El algoritmo Random Forest genera un bosque de dichos árboles con *bagging*[2] y subespacios aleatorios en cada árbol (cada árbol, aleatoriamente, solo utilizará parte de las características). La implementación utilizada es propia desarrollada para este proyecto.

2. CoForest con SSLTree como estimador base. CoForest se puede entender como la implementación del Random Forest en el aprendizaje semi-supervisado **?**. Implemen-

[2]La técnica de bagging consiste en generar subconjuntos aleatorios del conjunto de datos original para entrenar al estimador.

tación desarrollada en *sslearn* por José Luis Garrido-Labrador (Universidad de Burgos) **?**.

3. Self-Training con Random Forest como estimador base. El Random Forest, a su vez, tendrá el Decision Tree **supervisado** como estimador base.

Al trabajar con aleatoriedad, se ha fijado una semilla para la replicabilidad de los experimentos. Todos los modelos tienen los parámetros por defecto para una comparación justa.

Los resultados de la comparativa pueden visualizarse en la figura 6.9. Estos resultados indican que los *ensembles* con SSLTree como estimador base son peores que la versión supervisada (convertida a semi-supervisada mediante Self-Training). En la comparativa básica anterior, SSLTree sí parecía comportarse mejor que el Decision Tree supervisado (aunque sin diferencias significativas). Sin embargo, este Decision Tree, para algunos conjuntos de datos, seguía siendo mejor que el resto de modelos semi-supervisados, y para otros conjuntos, no se alejaba mucho de SSLTree a pesar de que disponer de pocos datos etiquetados.

Partiendo de esta idea y conociendo que los Random Forest (que en principio son supervisados) arrojan un rendimiento sobresaliente comparable con los mejores clasificadores (*state-of-the-art accuracy* según **?** y **?**), pueden superar a muchos de los modelos semi-supervisados solo con la porción etiquetada como ha ocurrido en este experimento.

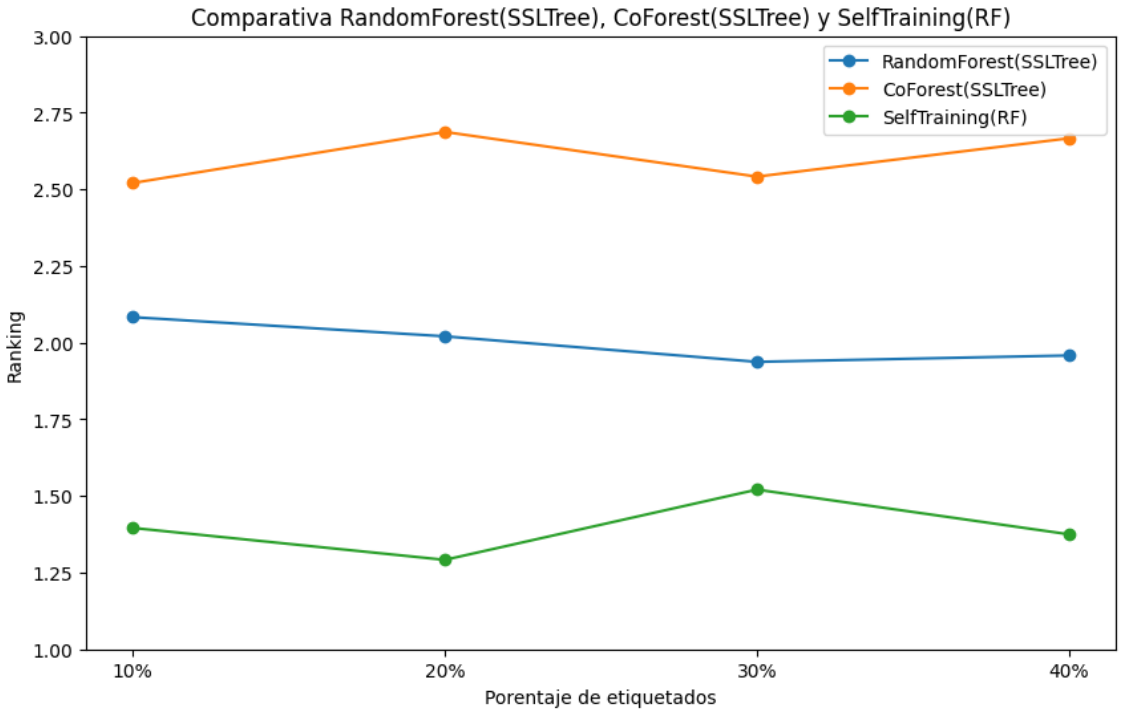

Figura 6.9: ***Ranking promedio de cada ensemble para todos los datasets.***

De hecho, realizando un nuevo test de Nemenyi entre estos modelos, se observan diferencias significativas en 10 % y 20 % de etiquetados. Estos resultados pueden verse en las figuras 6.10, 6.11, 6.12 y 6.13.

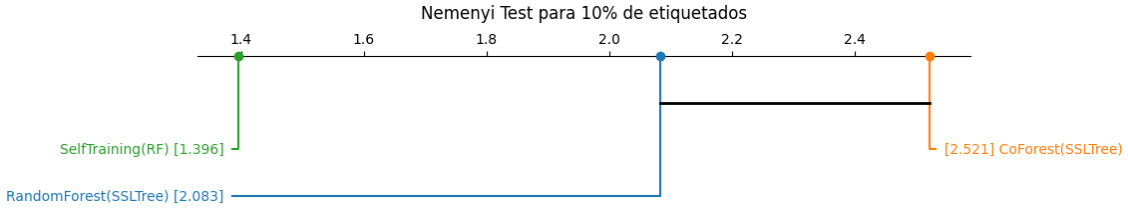

Figura 6.10: ***Comparativa ensembles: Nemenyi Test para 10 % de etiquetados.***

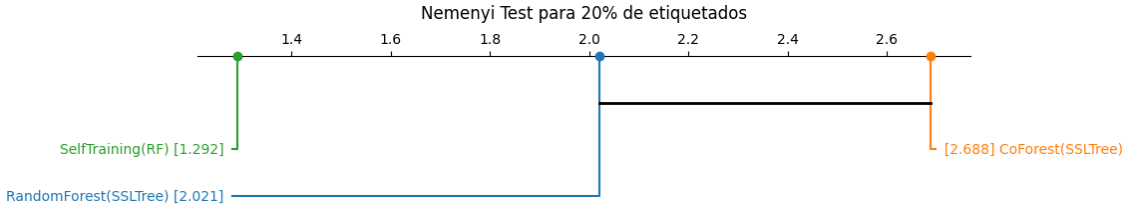

Figura 6.11: ***Comparativa ensembles: Nemenyi Test para 20 % de etiquetados.***

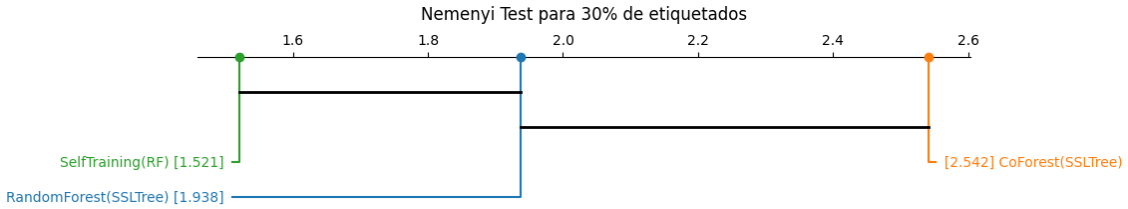

Figura 6.12: ***Comparativa ensembles: Nemenyi Test para 30 % de etiquetados.***

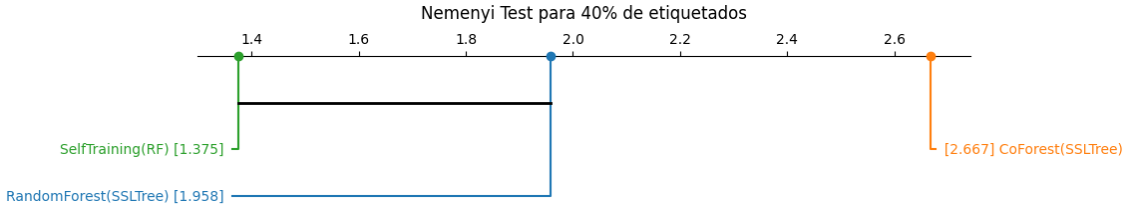

Figura 6.13: ***Comparativa ensembles: Nemenyi Test para 40 % de etiquetados.***

6.2. Experimentación GSSL

Código de la experimentación

Toda la codificación de los experimentos realizados puede encontrarse en https://github.com/dmacha27/TFM-VIU/tree/main/metodos/GSSL/experimentos.

Al igual que en *SSLTree*, para poder realizar una comparación justa con respecto a los modelos seleccionados, es necesario fijar los hiper-parámetros de los algoritmos desarrolla-

dos. Además, quieren ser probados como una unidad, es decir, se quiere comparar RGCLI + LGC y GBILI + LGC como modelos semi-supervisados.

El algoritmo común a estos dos nuevos modelos es LGC (el algoritmo de inferencia). El primer paso será fijar el hiper-parámetro α que posee. Para ello se necesita tener grafos construidos y para ello se utilizarán ambos algoritmos como métodos de creación de grafos. Para GBILI, en el estudio realizado por sus autores **?**, concluyen que el *accuracy* se estabiliza para un valor de $k > 10$ por lo que se ha fijado a $k = 11$. Para RGCLI, los autores **?** fijan el parámetro $k_e = 50$ y $k_i = 2$ para problemas de clasificación.

A partir de aquí, la experimentación consistirá en realizar un estudio del parámetro α de forma muy similar al parámetro w en *SSLTree* analizando los resultados obtenidos por ambos algoritmos.

Por cada porcentaje de etiquetados, se ejecutan pruebas para valores de α comprendidos entre 0.1 y 0.99 (ambos incluidos) con intervalos de 0.1. Estas pruebas (basadas en validación cruzada) se realizan por cada conjunto de datos y se realiza el ranking promedio que ocupa cada combinación de porcentaje de etiquetados y posible α (del mismo modo que en *SSLTree*). Los rankings obtenidos pueden verse en la figura 6.14.

Conjuntos de datos utilizados

Los métodos de construcción de grafos tienen un coste computacional muy alto. En estos experimentos se ha decidido obviar uno de los conjuntos de datos (se utilizarán 23 en vez de 24). El conjunto de datos en cuestión requería aproximadamente de cinco días de ejecución solo para el primer experimento más sencillo. Debido a las limitaciones temporales de este estudio, no se utilizará.

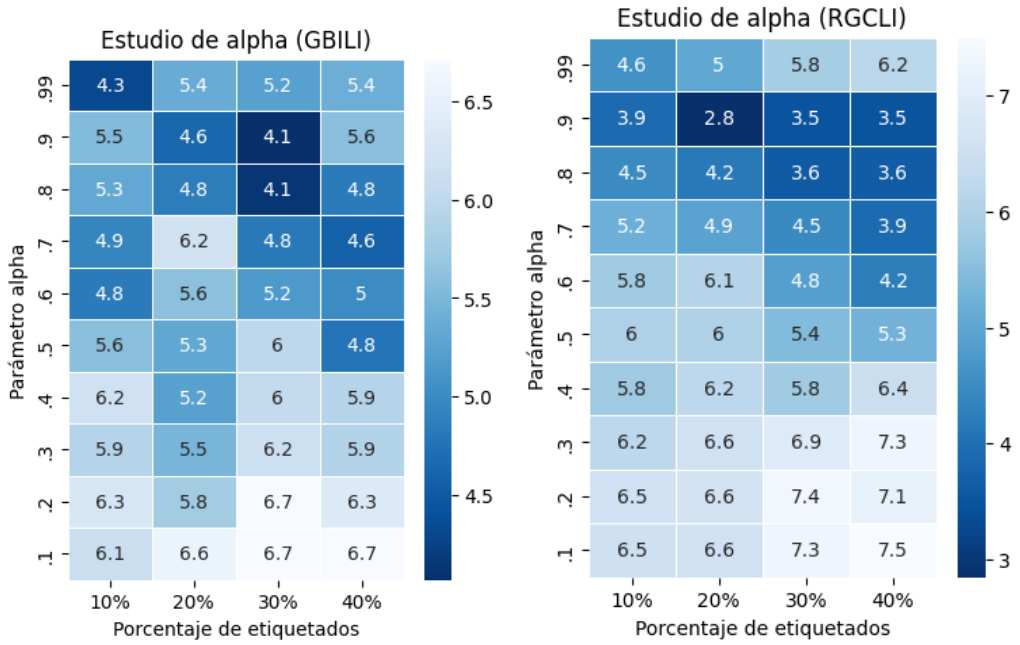

Figura 6.14: ***Rankings medios de α.***

Recurriendo de nuevo a un promedio es posible ver cuál es el ranking que ocupa cada valor de α:

ALPHA	0.99	0.9	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1
GBILI	5.076	4.957	4.783	5.125	5.147	5.408	5.842	5.859	6.277	6.527
RGCLI	5.408	3.429	3.967	4.62	5.239	5.668	6.043	6.734	6.918	6.973

Figura 6.15: ***Ranking promedio de cada valor de α para ambos métodos.***

En este punto, los dos algoritmos parecen obtener los mejores resultados para distintos valores de α, aunque bastante próximos. Para tomar una decisión de qué valor usar (si .9 o .8), se estudian las diferencias críticas existentes entre cada valor de cada algoritmos. El estudio de Nemenyi puede verse en las figuras 6.16 y 6.17.

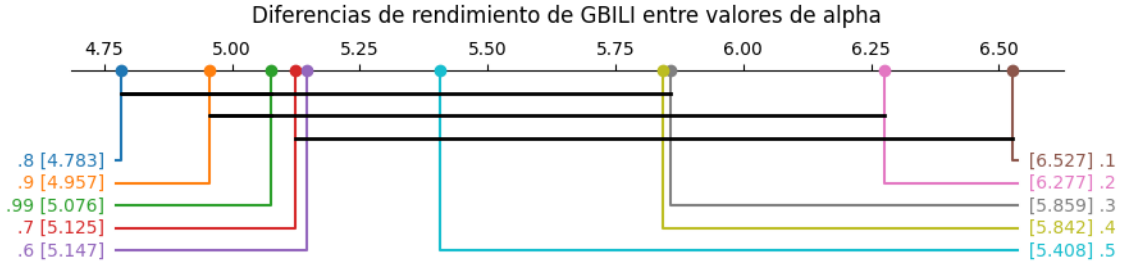

Figura 6.16: ***Nemenyi Test para valores de α en GBILI.***

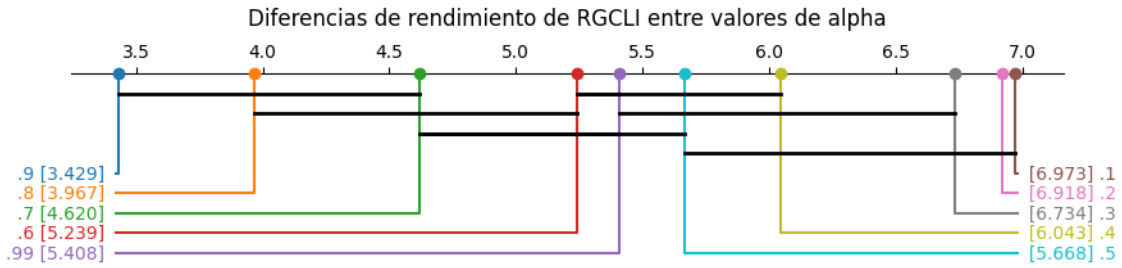

Figura 6.17: ***Nemenyi Test para valores de α en RGCLI.***

El estudio no arroja un decisión clara para seleccionar un valor u otro, podría tomarse cualquiera de los dos pues no existen diferencias críticas significativas. Sin embargo, la diferencia que existe entre .8 y .9 en GBILI es menor que la que existe en RGCLI. Si el valor seleccionado fuera .8, la penalización para RGCLI sería mayor que la que tendría GBILI si se selecciona .9. Por lo tanto, el valor seleccionado de α para LGC es .9.

A partir de esta hiperparametrización de LGC se continúa con la comparación de estos nuevos modelos con otros relevantes. Se han seleccionado dos modelos con semejanza directa a los métodos basados en grafos. Estos son:

1. Modelo *k-nearest-neighbor (kNN)*. Se trata de un algoritmo que se basa en la teoría de los vecinos más cercanos, como muchos de los algoritmos basados en grafos (GBILI y RGCLI). Clasifica nuevos ejemplos considerando los k vecinos más cercanos (asignando la etiqueta más común de ellos).

2. Self-Training con kNN como estimador base. Como se comentaba en *SSLTree*, es el algoritmo más básico en la familia de los algoritmos semi-supervisados, permite convertir el estimador base (en este caso knn) en un algoritmo semi-supervisado.

Por lo tanto, se realiza la comparación de GBILI + LGC, RGCLI + LGC con *KNeighborsClassifier* y *SelfTrainingClassifier*. Para cada conjunto de datos se ha obtenido el rendimiento (exactitud) para cada porcentaje de etiquetado, similar a lo realizado en *SSLTree*. A partir de los resultados obtenidos por cada dataset, se realiza el ranking promedio. Esto es, para cada conjunto de datos, se hace el ranking de los 4 modelos en cada porcentaje. Con eso se obtendrían 23 matrices en las que cada columna representa los rankings (cada celda de la columna es el ranking que ocupa el modelo) y a continuación se realiza la media entre los 23 datasets. La información final se resume en la gráfica 6.18

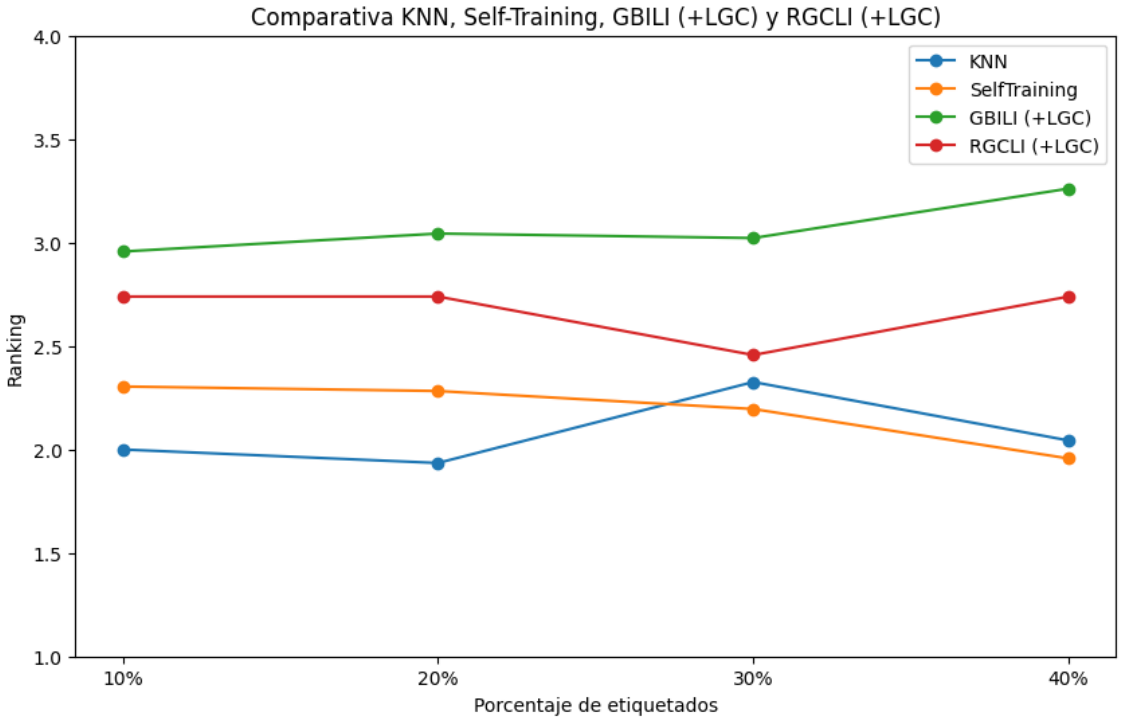

Figura 6.18: ***Ranking promedio de cada modelo en cada porcentaje de etiquetados.***

Los resultados indican que los mejores modelos son kNN y *Self-Training*. Al igual que con *SSLTree*, parece que un algoritmo supervisado obtiene mejores resultados que los semi-supervisados. Para evaluar si estos nuevos modelos son realmente peores, se evalúan de nuevo las diferencias críticas. Los test de Nemenyi pueden verse en las figuras 6.19, 6.20, 6.21 y 6.22.

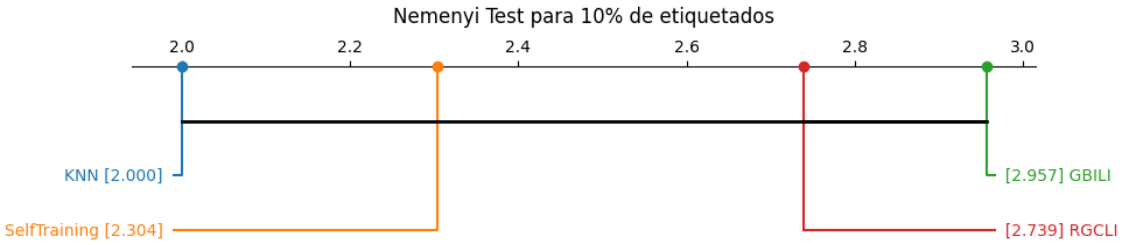

Figura 6.19: ***Comparativa grafos: Nemenyi Test para 10 % de etiquetados.***

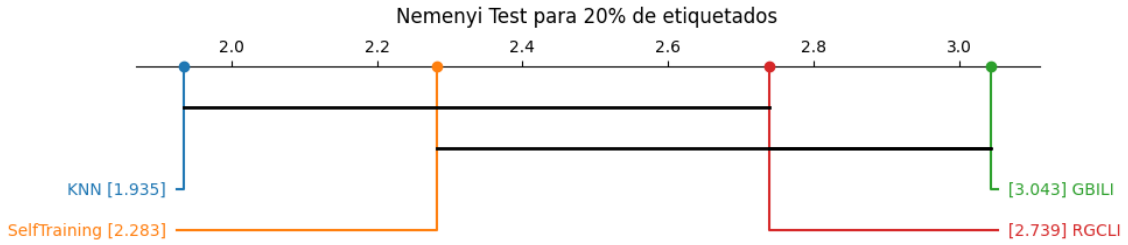

Figura 6.20: ***Comparativa grafos: Nemenyi Test para 20 % de etiquetados.***

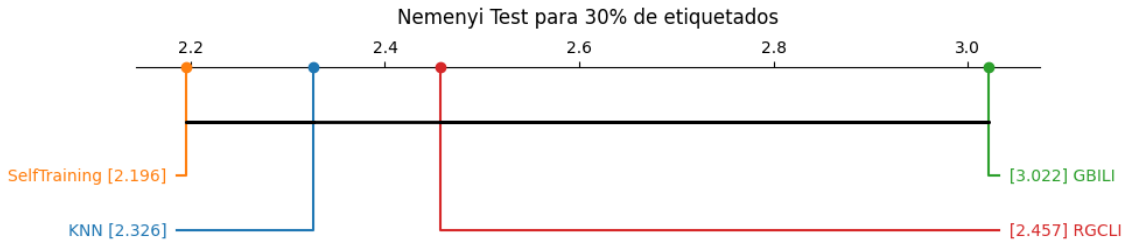

Figura 6.21: ***Comparativa grafos: Nemenyi Test para 30 % de etiquetados.***

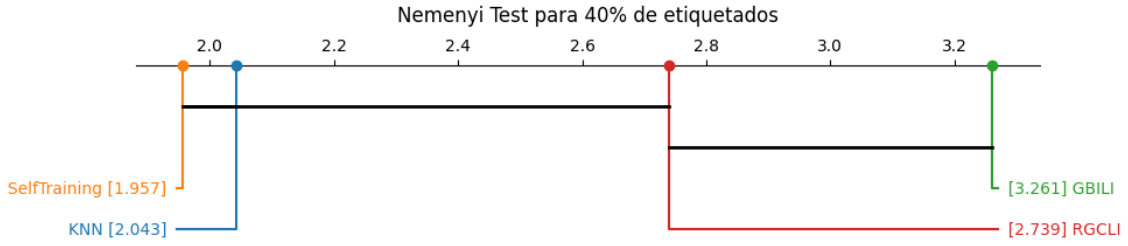

Figura 6.22: ***Comparativa grafos: Nemenyi Test para 40 % de etiquetados.***

RGCLI, como versión mejorada de GBILI es el mejor método basado en grafos. Además, GBILI tiene diferencias significativas en su rendimiento en 20 % y 40 % de etiquetados con respecto al resto. RGCLI por el contrario, no posee diferencias significativas con kNN ni con *Self-Training* y puede considerarse similar.

6.3. Discusión

Los resultados del proyecto se encuentran resumidos en las figuras 6.4, 6.9 y 6.18 (correspondencia con los objetivos 2, 5 y 6). *SSLTree* como nuevo método basado en la teoría de **?** parecía un método bastante prometedor por su naturaleza intrínseca. Comparado individualmente con otros métodos parece comportarse algo mejor (aunque sin diferencias significativas). En el caso de los ensembles, *SSLTree* no parece ser buen estimador base. Algo muy similar ocurre con los métodos basados en grafos (correspondencia con los objetivos 3, 5 y 6). Por su parte RGCLI como método de construcción produce resultados similares (no significativamente inferiores) al resto de métodos después de la inferencia (LGC).

Con este proyecto, estos algoritmos se han enfrentado a una cantidad suficiente de conjuntos de datos que permite evaluarlos en multitud de situaciones. La obtención de buenos resultados si se supone otra experimentación en la que se hayan podido seleccionar conjuntos de datos que sí cumplen las suposiciones del aprendizaje semi-supervisado no tendrían un valor sustancial. El número de conjuntos de datos que cumplen dichas suposiciones en el mundo real (que es el objetivo del desarrollo del *machine learning*) es muy reducido en comparación al resto. Una experimentación de ese estilo adulteraría el rendimiento real de los algoritmos.

Algo paradójico de esta experimentación es que, los algoritmos supervisados (y sus variantes en *ensembles*) se comportan mejor que los algoritmos semi-supervisados, incluso con algoritmos afianzados como *Self-Training*. Solo *SSLTree* parece ser comparable individualmente. Lo previsible era que aunque no se cumplieran las suposiciones para ningún conjunto de datos, aun así pudieran aprovechar algo más los datos no etiquetados. Esta situación lleva a considerar la utilidad real del aprendizaje semi-supervisado.

Concretamente acerca de los métodos basados en grafos, la principal cuestión a la que se quería dar respuesta es si es justificable el uso de esta rama algorítmica con respecto a algoritmos más tradicionales. En este marco, y para las configuraciones probadas, los métodos basados en grafos no parecen una mejor opción que el resto.

Por el contrario, la posible utilidad de *SSLTree* va más allá de su uso individual (que parece comportarse algo mejor). La disponibilidad de un método intrínsecamente semi-supervisado que además esté basado en árboles de decisión puede ser de gran utilidad en el desarrollo de otros algoritmos. De hecho, *SSLTree* está siendo utilizado por el grupo de investigación de la Universidad de Burgos para desarrollar un *Rotation Forest* semi-supervisado al utilizarlo como estimador base.

Aún con las distintas cuestiones comentadas, este proyecto pone a disposición de otros desarrolladores e investigadores un nuevo algoritmo basado en árboles de decisión e implementaciones públicas de los métodos basados en grafos.

Conclusiones

7

En este apartado se comentarán las conclusiones extraídas a partir del desarrollo y experimentación de los métodos implementados. Aunque en primer lugar, las conclusiones generales de los objetivos marcados se resumen en las siguientes:

1. Gracias a la información recopilada previamente y a nueva documentación bibliográfica buscada, se establecieron las líneas investigación del proyecto (objetivo 1).

2. Se ha implementado un nuevo modelo basado en árboles, *SSLTree*, así como tres algoritmos basados en grafos GBILI, RGCLI y LGC (objetivos 2 y 3).

3. Se ha generado una experimentación exhaustiva para todos los métodos desarrollados con respecto a métodos clásicos y con comparación directa (objetivos 4, 5 y 6). Toda la codificación de los experimentos es pública a través del repositorio del proyecto (Github).

4. Se ha creado una aplicación Web para la visualización y aprendizaje interactivo de los métodos basados en grafos desarrollados en el proyecto (objetivo adicional).

En cuanto a *SSLTree*:

5. *SSLTree* es competitivo. Los resultados obtenidos de la comparativa de *SSLTree* con el resto de modelos (no *ensembles*) indicaron que es mejor empíricamente (aunque sin diferencias significativas).

6. Debido a la primera conclusión (*SSLTree* parece mejor). La teoría en **?** y concretamente el parámetro w, resulta beneficioso al aplicarse a un árbol de decisión tipo *Classification And Regression Tree* (CART). Al haberse comparado con una implementación CART supervisada, cuando aparecen datos no etiquetados y se aplica *SSLTree*, se obtienen mejores resultados.

7. Resulta ser el peor estimador base al aplicarlo a *ensembles*.

8. Aplicar un algoritmo de poda (post-poda) puede mejorar los resultados del modelo para el contexto concreto.

En cuanto a los métodos basados en grafos, se utilizaron algoritmos ya encontrados en la literatura:

9. Tanto la combinación GBILI + LGC como RGCLI + LGC son peores empíricamente con respecto al resto de modelos. RGCLI no presenta diferencias significativas y podría considerarse similar. Sin embargo, no es justificable el uso de estos métodos con respecto al resto.

10. Los algoritmos de construcción de grafos tienen un alto coste computacional y esto hace que si el conjunto de datos crece, aunque sea relativamente poco, el tiempo de ejecución del modelo completo (+ LGC) sea desorbitado.

11. RGCLI consigue mejores resultados en clasificación con el mismo método de inferencia. Esto demuestra que el método de construcción de grafos influye directamente en la inferencia/clasificación.

Limitaciones y Perspectivas de Futuro

8

En esta sección se comentan las limitaciones observadas durante el proyecto e ideas que se han descartado como perspectivas de futuro.

- **Limitación temporal**: El desarrollo de los algoritmos no ha sido demasiado extenso (la mayor parte fue para SSLTree). Sin embargo, los experimentos de cada método, en varias variantes, en su versión individual o *ensemble*, entre otros, ha conllevado la mayor parte del tiempo. En este sentido, hubiera sido interesante haber podido probar más conjuntos de datos incluso seleccionando, a conciencia, algunos que pudieran cumplir las suposiciones para ver sí, en esos casos, el rendimiento es bueno. Además, habiendo probado que el método de construcción de grafos es determinante en la calidad de la clasificación, también se podrían haber implementado más algoritmos de construcción e incluso de inferencia.

- **Explorar rama de redes neuronales**: Algo que se comentó en las primeras reuniones (por parte de la Universidad de Burgos) era tratar también los métodos semi-supervisados basados en redes neuronales. Algo que tampoco se había probado en el grupo de investigación y que también se posee cierta recopilación bibliográfica sobre el tema. Sin embargo, se creyó suficiente con que, de momento, se abordaran solo árboles y grafos.

- **Añadir árboles a la Web**: La aplicación web desarrollada, aunque añadida como objetivo en la etapas finales, solo fue pensada para mostrar los pasos interesantes de los métodos basados en grafos. Esto fue así porque se cree que los árboles de decisión, con una pequeña aclaración teórica, son sencillos de comprender y muy interpretables. Sin embargo, debido a que el aprendizaje de cada estudiante es muy particular y esa decisión puede estar sesgada a poseer conocimientos previos, podrían incluirse unas visualizaciones similares de las construcción de los árboles.

Apéndice A: Más resultados experimentación SSLTree

A

Aquí se incluirán todos los resultados *adicionales* que, por completitud, se desean incluir de la experimentación propuesta en 5.

A.1. Estudio del parámetro w

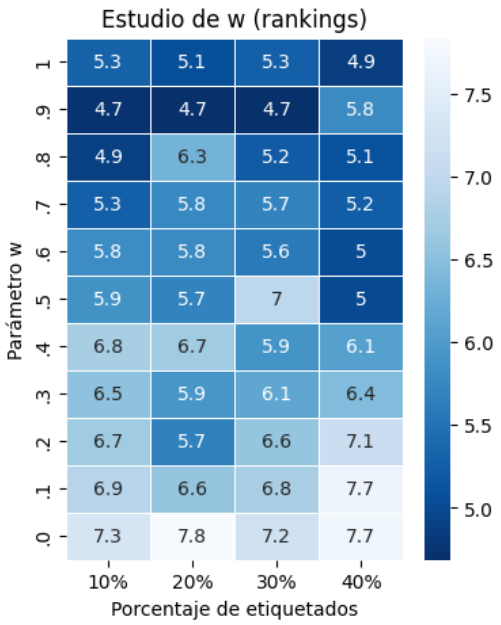

Figura A.1: ***Mapa de calor de los rankings medios (Gini).***

Recurriendo de nuevo a un promedio es posible ver cuál es el ranking que ocupa cada valor de w:

1	**.9**	.8	.7	.6	.5	.4	.3	.2	.1	0
5.12	**4.97**	5.38	5.49	5.55	5.88	6.35	6.26	6.51	6.98	7.51

Tabla A.1: *Ranking promedio de cada valor de w (Gini)*

La conclusión con Gini es la misma que para Entropy, el mejor valor de w parece ser 0.9 en todos los datasets.

A.2. Comparativa básica

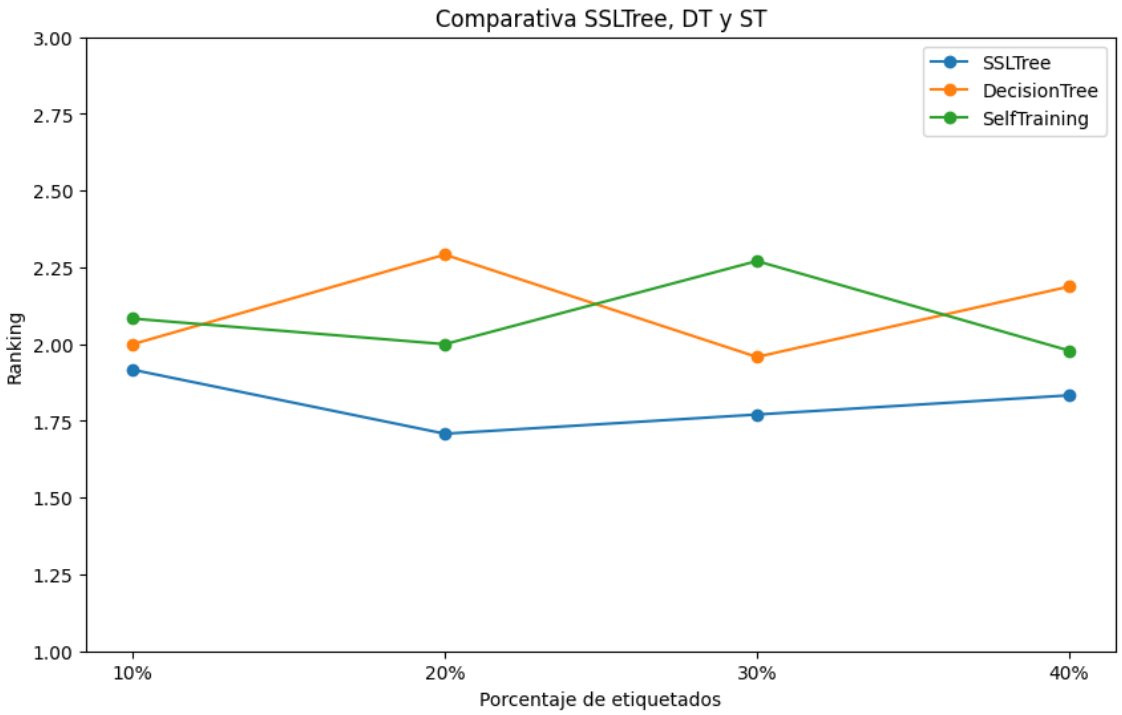

Figura A.2: ***Ranking promedio de cada modelo para todos los datasets (Gini).***

Los resultados obtenidos indican que la implementación de SSLTree funciona correctamente y en general obtiene mejores resultados que el resto de modelos.

El análisis de diferencias críticas/significativas para el criterio Gini resulta en la misma conclusión. Aunque SSLTree sea mejor, no hay diferencias significativas que supongan una mejora grande.

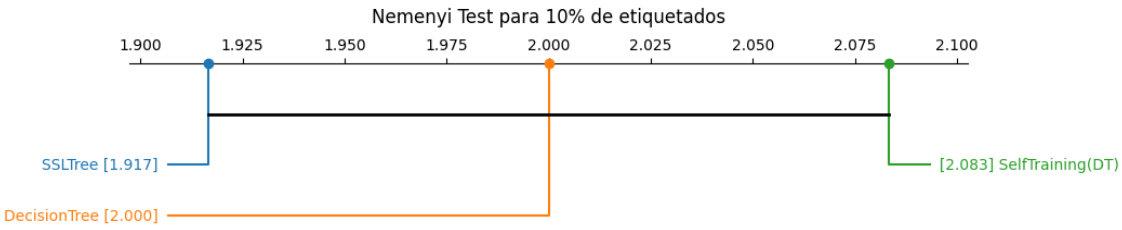

Figura A.3: ***Comparativa básica: Nemenyi Test para 10 % de etiquetados (Gini).***

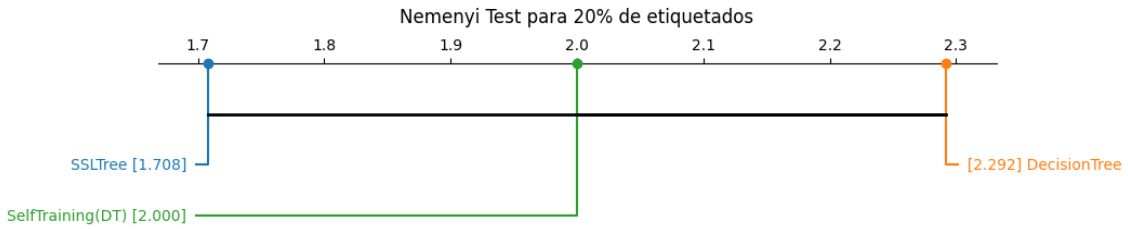

Figura A.4: ***Comparativa básica: Nemenyi Test para 20 % de etiquetados (Gini).***

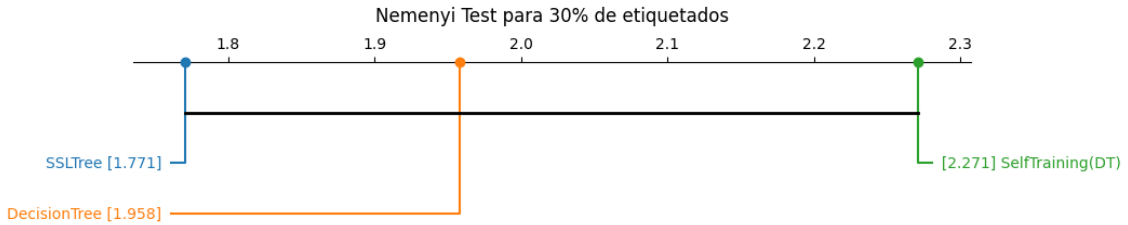

Figura A.5: ***Comparativa básica: Nemenyi Test para 30 % de etiquetados (Gini).***

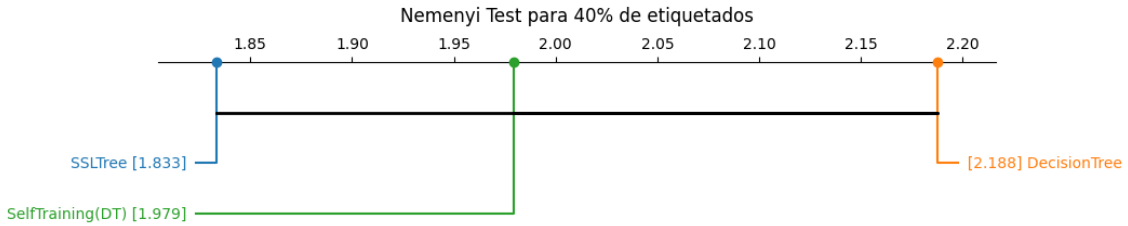

Figura A.6: ***Comparativa básica: Nemenyi Test para 40 % de etiquetados (Gini).***

A.3. Comparativa entre ensembles

Conjuntos de datos utilizados

Cierta función interna de la biblioteca *SSLTree* utilizada en la comparación (Co-Forest) parece no soportar algún dato. Existe un conjunto de datos que, pese a construirse el árbol de decisión, arroja un error en ejecución. Para el criterio Gini, simplemente se ha ignorado este conjunto de datos. Las conclusiones no se han visto afectadas.

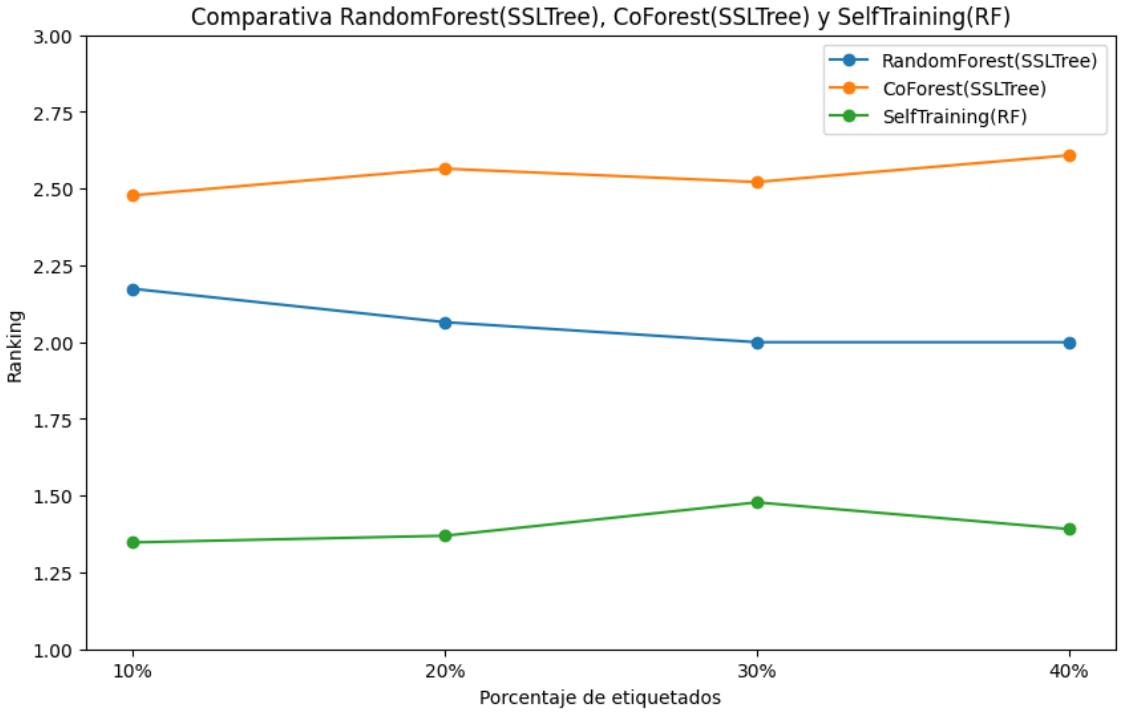

Figura A.7: ***Ranking promedio de cada ensemble para todos los datasets (Gini).***

Realizando un nuevo test de Nemenyi entre estos modelos, se observan prácticamente las mismas diferencias significativas en 10 % y 20 % de etiquetados que para *entropy*. Estos resultados pueden verse en las figuras A.8, A.9, A.10 y A.11.

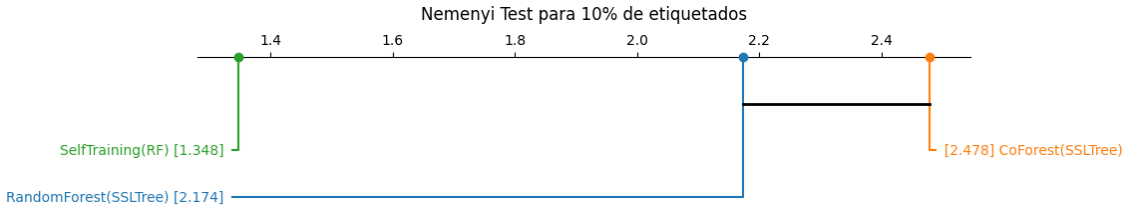

Figura A.8: ***Comparativa ensembles: Nemenyi Test para 10 % de etiquetados (Gini).***

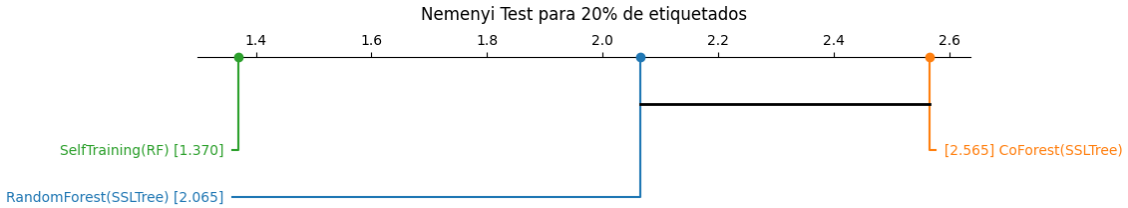

Figura A.9: ***Comparativa ensembles: Nemenyi Test para 20 % de etiquetados (Gini).***

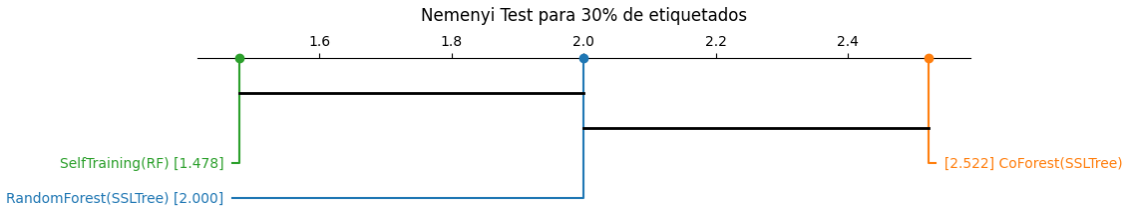

Figura A.10: ***Comparativa ensembles: Nemenyi Test para 30 % de etiquetados (Gini).***

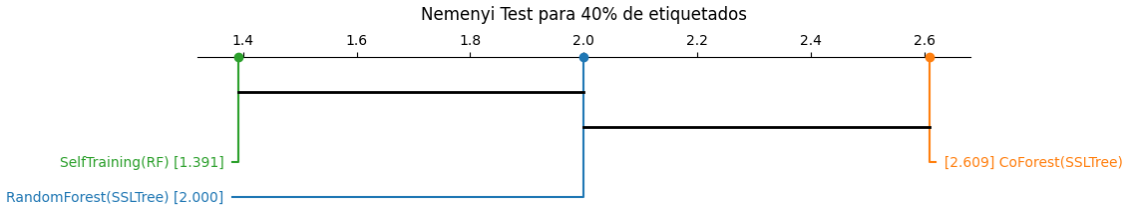

Figura A.11: ***Comparativa ensembles: Nemenyi Test para 40 % de etiquetados (Gini).***

Apéndice B: Visualizador GSSL

B

En este anexo se comenta el desarrollo y resultado de una aplicación web sencilla diseñada a partir de los algoritmos desarrollados en la investigación.

El objetivo es poder entender cómo funcionan los grafos, sus métodos de construcción y también cómo se propagan las etiquetas. Es accesible desde https://dmacha.dev/gssl.

B.1. Motivación

En la Universidad de Burgos se han realizado otras herramientas, tanto publicadas en revistas como proyectos de Trabajos de Fin de Grado (como VASS, realizada por el mismo autor que este trabajo), que están alineadas con la docencia de inteligencia artificial y *machine learning*. De esta manera, se puede disponer de una nueva herramienta que pueda ser de ayuda para el aprendizaje de estudiantes, con aplicación concreta a los grafos en el aprendizaje semi-supervisado.

B.2. Objetivos

- Modificar los algoritmos GBILI, RGCLI y LGC para la extracción de información de los pasos relevantes.
- Crear una *Application Programming Interface* (API) para la ejecución de los algoritmos y obtención de los pasos relevantes para su visualización Web.
- Crear la interfaz de usuario con la funcionalidad necesaria para la interpretación de la información.

B.3. Tecnologías utilizadas

Las tecnologías o herramientas utilizadas se enmarcan en dos grupos, por un lado, el *frontend*, y por otro, el *backend*.

En el *frontend* se han utilizado varias bibliotecas destacables (todas gratuitas y libres) que han permitido generar las visualizaciones, estas son:

- **Bootstrap**: Contiene código HTML, CSS y JavaScript que permite personalizar la interfaz del usuario actuando sobre los componentes mediante clases. Su objetivo es la creación de páginas web *responsive* (adaptables).

- **jQuery**: Biblioteca muy utilizada que simplifica la programación de código JavaScript.

- **DataTables**: Permite crear tablas interactivas completamente personalizables con datos dinámicos.

- **KaTeX**: Permite visualizar notación matemática en los navegadores y está basada en LaTeX.

- **Pseudocode**: Permite mostrar pseudocódigos de algoritmos tal y como se muestran en un documento generado por LaTeX. Utiliza KaTeX para renderizar fórmulas matemáticas.

- **D3.js**: Esta biblioteca es el núcleo de la aplicación. Permite generar, de forma automática, la visualización del grafo. Tiene un alto grado de libertad, lo que permite modificar completamente todos los aspectos del grafo (nodos, enlaces, colores, tamaños...).

- **introJS**: Esta biblioteca permite generar tutoriales interactivos en una página Web. En cada paso del tutorial, es posible seleccionar un elemento del la página junto com la explicación. El usuario puede navegar entre los pasos incluidos para comprender el funcionamiento.

Con respecto al *backend*, está implementado en Python y además de los propios algoritmos investigados y alguna utilidad creada para este proyecto, se han utilizado varias bibliotecas conocidas en *machine learning* y utilizadas en el resto de la maestría:

- **pandas**: Biblioteca muy utilizada sobre manipulación de datos que define nuevas estructuras de datos capaces de realizar operaciones de interés de manera eficiente.

- **scikit-learn**: Es la biblioteca más utilizada en *machine learning* en Python, incluye numerosos algoritmos de aprendizaje así como multitud de utilidades de procesamiento.

- **numpy**: Biblioteca dedicada al cálculo científico, implementa estructuras de datos tipo *arrays* y matrices y funciones matemáticas para realizar operaciones de forma eficiente (de hecho, buena parte de su implementación está codificada en lenguaje C).

- **scipy**: Biblioteca de algoritmos matemáticos.

- **flask**: Es un *framework* en Python para desarrollar aplicaciones web. Es el núcleo de toda la aplicación.

B.4. Desarrollo

En esta sección se describen todas las decisiones de diseño y desarrollo que se han realizado para cumplir los objetivos.

Código de la implementación

La implementación completa de la web puede encontrarse en https://github.com/dmacha27/TFM-VIU/tree/main/web.

B.4.1. Diseño arquitectónico

La aplicación posee una arquitectura de dos capas. Esto es así porque como se quería desarrollar una aplicación sencilla, no se ha utilizado una tercera capa de datos (bases de datos) que suele ser común y que se catalogaría como arquitectura de tres capas.

Arquitectura de dos capas

Esta arquitectura esta formada por dos capas: capa de presentación y capa de aplicación (también conocida como capa o "lógica de negocio"). En la figura B.1 se representa un diagrama con esta arquitectura.

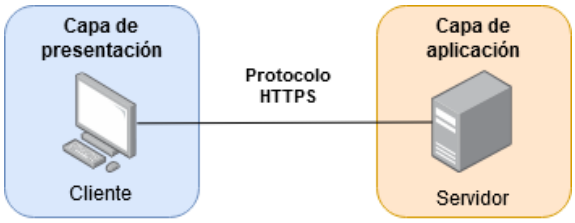

Figura B.1: ***Arquitectura de dos capas.***

- **Capa de presentación**: permite al usuario interactuar con el sistema. Se encarga de mostrar, y en este caso visualizar, la información que la aplicación desea. El contenido de esta capa es ejecutado en el propio navegador del usuario. El *framework* Flask comentado anteriormente es el encargado de generar, para el usuario, todo el conjunto de plantillas, estilos y código dinámico (JavaScript) que el navegador ejecuta. De forma práctica, Flask es la capa de presentación.
- **Capa de aplicación**: maneja las peticiones de la capa de presentación y envía las respuestas apropiadas. También conocida como lógica de negocio, recibe este nombre porque es aquí donde está definida toda la funcionalidad de la aplicación. Esta capa está implementada en Python y es todo el código que se ejecuta en las rutas de la API definida. En este caso, por ejemplo, sería la ejecución de un algoritmo concreto.

La comunicación entre estas capas se realiza mediante el protocolo HTTPS (Protocolo de Transferencia de Hipertexto Seguro).

B.4.2. Diseño procedimental

En esta sección se describe la interacción del usuario con el sistema para visualizar un algoritmo. Los pasos que el usuario realiza son los siguientes (queda reflejado en la figura B.2):

1. El usuario accede a la web mediante su navegador y el servidor generará la plantilla HTML (y estilado) principal (carga del conjunto de datos).

2. El usuario carga un conjunto de datos que será almacenado en el propio navegador. También se solicita al servidor la información básica de los datos (nombre de las columnas). El navegador permitirá pasar al apartado de configuración.

3. El usuario configura su ejecución seleccionando los parámetros generales (como el nombre del atributo de la clase) y los específicos de cada algoritmo.

4. El navegador realizará una petición al servidor con esa configuración a la espera de una respuesta con toda la información de la ejecución.

5. El servidor entonces carga de forma dinámica el conjunto de datos, ejecuta el algoritmo de construcción de grafos y ejecuta el algoritmo de inferencia de etiquetas. El servidor recopila y procesa la información de los pasos realizados y construye la respuesta (JSON).

6. EL navegador procesa la información y genera las visualizaciones correspondientes para el usuario.

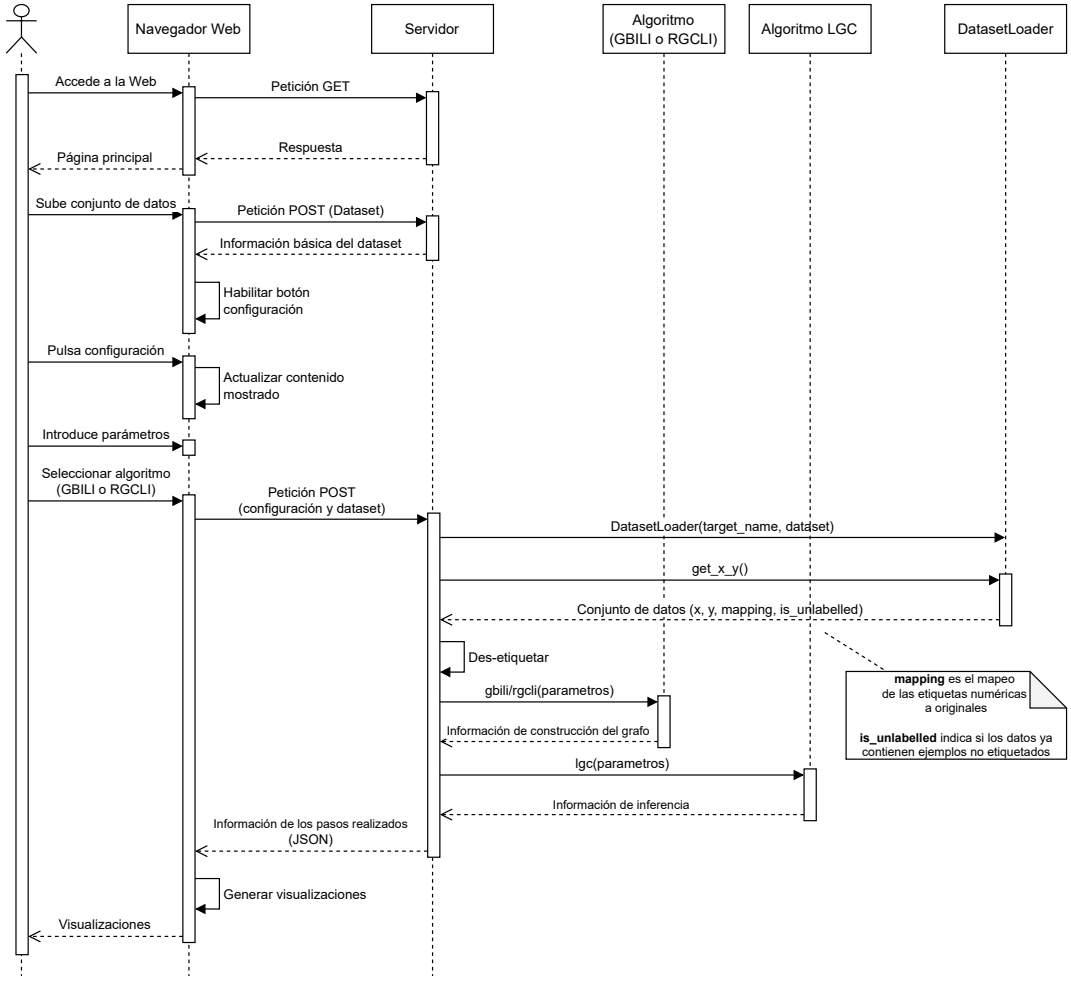

Figura B.2: ***Diagrama de secuencia.***

B.4.3. Diseño de datos

En esta sección se describe la información que fluye entre el servidor y el cliente. La figura B.2 representa esa última respuesta del servidor (“Información de los pasos realizados”).

El diseño de la lógica está pensado de tal forma que dependiendo qué algoritmo de construcción de grafo se quiera visualizar, se realiza una petición a una ruta u otra. Esto quiere decir que, salvo la información de los pasos de la construcción (GBILI o RGCLI), el resto es común (el algoritmo de inferencia LGC). El objetivo de ambas rutas es responder con un texto tipo *JavaScript Object Notation* (JSON), que JavaScript es capaz de manejar de forma nativa y directa.

El aspecto de una respuesta de las rutas es algo parecido a la figura B.3. Siempre existe una entrada que representa los pasos del algoritmo de construcción y otra para el algoritmo de inferencia. Sigue una estructura típica de diccionario con claves y valores. La estructura se organiza por niveles. En el primer nivel se hace referencia al algoritmo, en el segundo se hace referencia a cada paso dentro de un algoritmo y en el tercer nivel se encuentra la

información interesante del paso. En este último caso, es de destacar que siempre se informa de los nodos (los ejemplos en el conjunto de datos), de los enlaces que unen dichos nodos de cada paso (los enlaces de un paso pueden no ser los mismo que otro) e información adicional interesante del paso.

```
response = {

    "gbili": {
        "PASO X": {
            "nodes": nodes,
            "links": link_list,
            "+INFO": INFO
        }
    },

    "lgc": {
        "PASO X": {
            "nodes": nodes,
            "links": link_list,
            "+INFO": INFO
        }
    }
}

return jsonify(response)
```

Figura B.3: *Estructura de la respuesta*

Pasos de GBILI

Los pasos más relevantes que se han querido mostrar son: construcción de la matriz de distancias, obtención de los vecinos más cercanos, obtención de los vecinos mutuos, conectar vecinos cercanos a puntos etiquetados y el grafo final. Todo estos pasos están descritos en 5.3.1 y mostrados en el pseudocódigo 3.

1. ***dataset***: En este paso no existen enlaces todavía, se incluye la matriz de distancias D de cada par de nodos.

2. ***knn***: En este paso, los enlaces representan uniones entre los vecinos más cercanos. Un elemento de la lista de enlaces tiene la forma:

   ```
   {"source": nodo origen,
   "target": nodo destino,
   "value": fuerza del enlace}
   ```

   Además, se incluyen esos vecinos más cercanos de cada nodo como una lista (la posición i-ésima contiene los vecinos más cercanos del nodo i).

```
"gbili": {
    "dataset": {
        "nodes": nodes,
        "links": [],
        "distance": D.tolist()
    },
    "knn": {
        "nodes": nodes,
        "links": links_knn,
        "neighbors": D_argsort.tolist()
    },
    "m_knn": {
        "nodes": nodes,
        "links": links_m_knn,
        "mneighbors": m_knn
    },
    "semi_graph": {
        "nodes": nodes,
        "links": links_semi_graph,
        "components": [component_membership_semi
           [i] for i in range(len(X))],
        "components_with_labeled": list(
           components_with_labeled)
    },
    "graph": {
        "nodes": nodes,
        "links": links_graph,
        "unions": list(unions),
        "components": [
           component_membership_graph[i] for i
           in range(len(X))]

    }
}
```

Figura B.4: *Pasos GBILI*

3. ***m_knn***: En este paso, los enlaces representan uniones entre los vecinos más cercanos mutuos. Además, se incluyen esos vecinos más cercanos mutuos de cada nodo como una lista (la posición i-ésima contiene los vecinos más cercanos mutuos del nodo i).

4. ***semi_graph***: En este paso, los enlaces creados son parte del grafo final. Se incluye también un listado que indica la componente a la que pertenece cada nodo (la posición i-ésima contiene el número de componente a la que pertenece el nodo i). Por último, se construye un listado con las componentes que tienen algún punto etiquetado en ellas.

5. ***graph***: Los enlaces representan el grafo final construido. “unions” es una lista de tuplas de dos elementos en la que cada una contiene el número de las dos componentes unidas. Se incluye de nuevo un listado que indica la componente a la que pertenece cada nodo (en el grafo final).

Información de RGCLI

Para RGCLI, los pasos más relevantes que se han querido mostrar son: construcción de la matriz de distancias, obtención de los vecinos más cercanos, el más cercano etiquetado y el k-ésimo más lejano y, por último, el grafo final construido. Todo estos pasos están descritos en 5.3.2 y mostrados en el pseudocódigo 4.

```
"rgcli": {
    "dataset": {
        "nodes": nodes,
        "links": [],
        "distance": D.tolist()
    },
    "searchknn": {
        "nodes": nodes,
        "links": links_knn,
        "kNN": kNN.tolist(),
        "L": L.tolist(),
        "F": F_rgcli.tolist()
    },
    "graph": {
        "nodes": nodes,
        "links": links_graph
    }
}
```

Figura B.5: *Pasos RGCLI*

1. ***dataset***: En este paso no existen enlaces todavía, se incluye la matriz de distancias D de cada par de nodos.

2. ***searchknn***: En este paso, los enlaces representan uniones entre los vecinos más cercanos. El formato de la lista de enlaces es el mismo que para GBILI. Además, se incluyen esos vecinos más cercanos mutuos de cada nodo del mismo modo que en GBILI, el etiquetado más cercano de cada nodo y el k-ésimo más lejano.

3. ***graph***: Los enlaces representan el grafo final construido.

Información de LGC

Este algoritmo es común para los dos anteriores y los pasos más relevantes que se han querido mostrar son: construcción de afinidad, creación de la matriz S, el proceso iterativo de inferencia y, por último, el etiquetado de los datos. Todo estos pasos están descritos en 5.3.3 y mostrados en el pseudocódigo 5.

Algo común a todos los pasos es que los enlaces representan el grafo final construido (proveniente de unos de los algoritmos anteriores). La demás información se resume en:

1. ***dataset***: Se incluye la matriz de afinidad (W) y también una matriz de etiquetado similar a 5.3.3 (F), que representa las etiquetas de cada nodo.

```
"lgc": {
    "affinity": {
        "nodes": nodes,
        "links": links_graph,
        "F": F.tolist(),
        "W": W.tolist(),
    },
    "S": {
        "nodes": nodes,
        "links": links_graph,
        "D": D_diag.tolist(),
        "D_sqrt_inv": D_sqrt_inv.tolist(),
        "S": S.tolist(),
    },
    "iteration": {
        "nodes": nodes,
        "links": links_graph,
        "F_history": F_t_history.tolist(),
        "pred_history": pred_history.tolist()
    },
    "labels": {
        "nodes": nodes_final,
        "links": links_graph,
        "F_final": F_t_history[-1].tolist(),
        "pred_final": pred_history[-1].tolist()
    }
}
```

Figura B.6: *Pasos LGC*

2. ***S***: En este paso se calcula la matriz S (versión normalizada de W). Para su cálculo se utiliza D y $D^{-1/2}$. Se incluyen ambas matrices y la matriz S. En realidad, su inclusión solo es por completitud, es interesante ver el formato de esa nueva matriz D (que es diagonal).

3. ***iteration***: Incluye un historial tanto de las etiquetas de cada nodo “pred_history” (una lista de listas de una dimensión) como de la matriz de etiquetado “F_history” (una lista de matrices donde cada una es como 5.3.3 y se tiene una por cada iteración de LGC).

4. ***labels***: Incluye la última lista de “pred_history” (“pred_final”) y la última matriz de etiquetado “F_final” (“F_final”).

Universidad de Burgos

UNIVERSIDAD
DE BURGOS

Proyecto parcialmente tutorizado por la **Universidad de Burgos**.
Tutores UBU: Álvar Arnaiz González y César Ignacio García Osorio.