Universidad Internacional de Valencia

Máster Universitario en Inteligencia Artifical
Trabajo Fin de Máster

Implementación y comparativa de métodos semi-supervisados

UNIVERSIDAD DE BURGOS

Titulación:
Máster Universitario en Inteligencia Artifical

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

Convocatoria:
Septiembre

Curso académico:
2023-2024

Directora de TFM: Irma Sanabria

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

Eleanor Roosevelt

Agradecimientos

Me gustaría agradecer...

También quiero destacar...

Por último...

Índice general

Índice de figuras

Índice de tablas

Índice de algoritmos

Resumen

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 a las similitudes/disimilitudes de los ejemplos (por ejemplo, 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 encuentra a caballo entre el 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.

Marco teórico

2

2.1. Aprendizaje automático

El aprendizaje automático (*machine learning*) según Martel (2020) 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 van Engelen y Hoos (2020).

En la figura 2.1 se presenta una clasificación del aprendizaje automático.

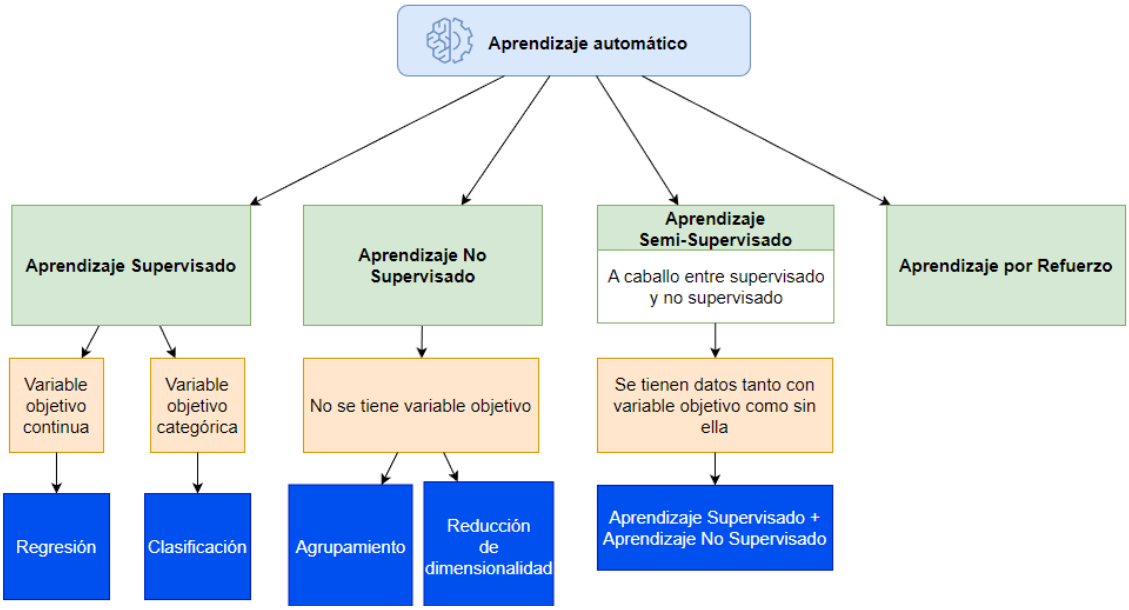

Figura 2.1: ***Clasificación de aprendizaje automático, basado en Solutions (2018).***

2.1.1. Aprendizaje supervisado

Los algoritmos de aprendizaje supervisado utilizan datos etiquetados durante su proceso de entrenamiento Alexander S. Gillis (2021). 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 Dridi (2021a).

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) Wikipedia contributors (2024). Ningún dato de este último conjunto ha sido “visto” por el modelo previamente.

En la figura 2.2 se encuentra un diagrama con el funcionamiento general.

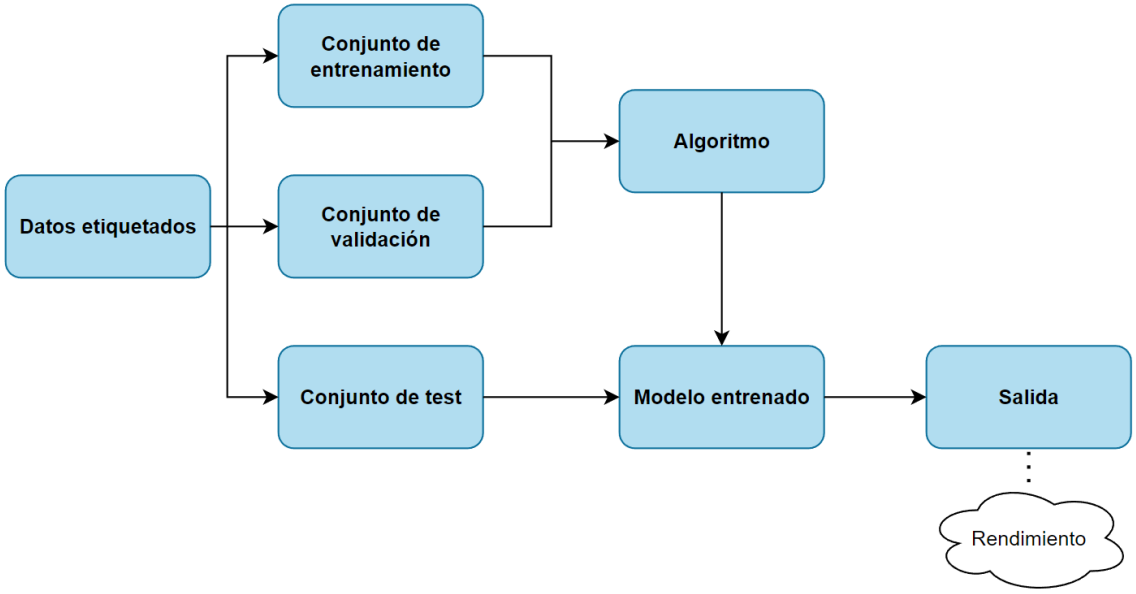

Figura 2.2: ***Funcionamiento general del aprendizaje supervisado, basado en Dridi (2021a).***

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.

2.1.2. Aprendizaje no supervisado

A diferencia del aprendizaje supervisado, el no supervisado no trabaja con datos etiquetados y clases. Según Dridi (2021b) 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 Dridi (2021b).

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 javaTpoint 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.

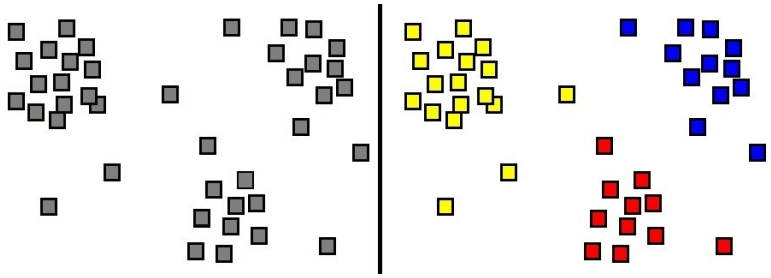

Figura 2.3: Clusters. *A la izquierda los datos sin agrupar y a la derecha los datos coloreados según la pertenencia a los distintos grupos. By hellisp - Own work, Public Domain, https://commons.wikimedia.org/w/index.php?curid=36929773.*

2.1.3. Aprendizaje semi-supervisado

Según van Engelen y Hoos (2020), 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.

2.1.3.1. 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 van Engelen y Hoos (2020) 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 $p(y|x)$ van Engelen y Hoos (2020).

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 Lukas Huber (2022); van Engelen y Hoos (2020).

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 van Engelen y Hoos (2020).

En la figura 2.4 se presenta la taxonomía general del aprendizaje semi-supervisado.

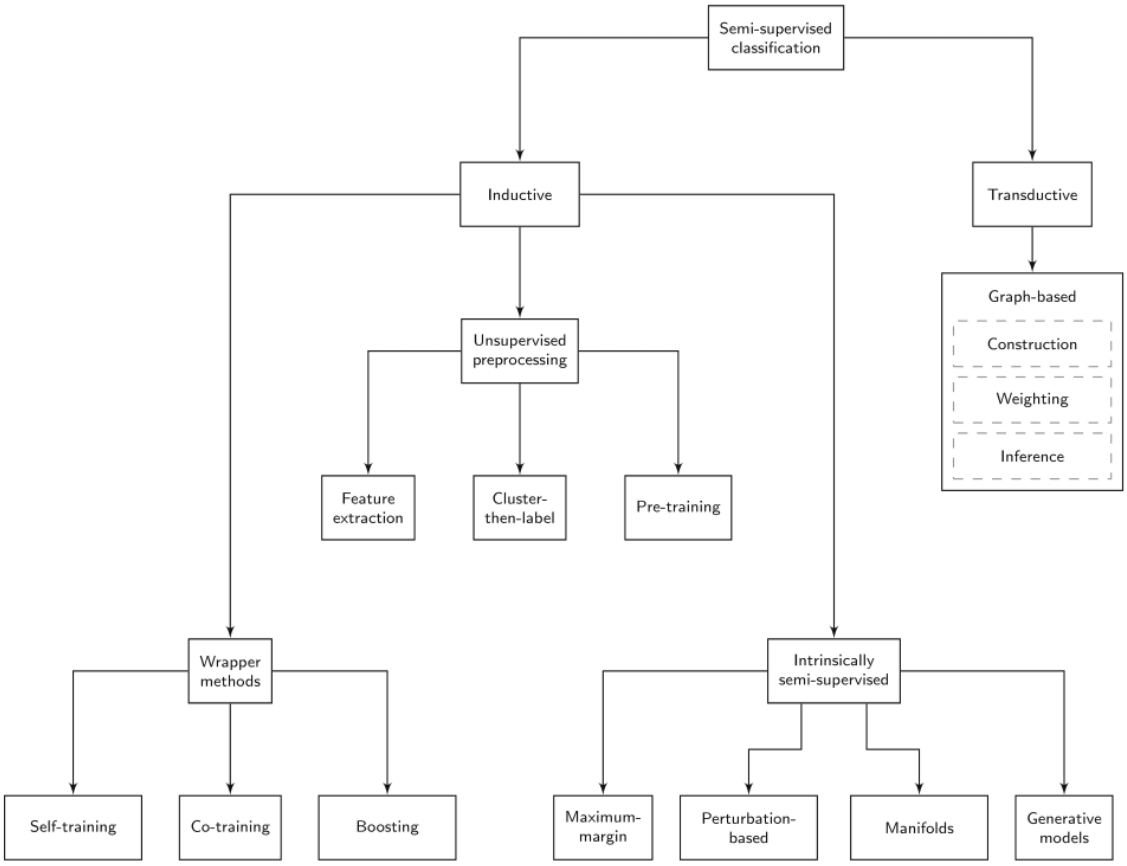

Figura 2.4: ***Taxonomía de métodos semi-supervisados van Engelen y Hoos (2020).***

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.

2.2. 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 Quinlan (1986), C4.5 Quinlan (2014), C5.0 Quinlan (2004) y CART Breiman (2017), 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 Kingsford y Salzberg (2008). 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 Kingsford y Salzberg (2008). 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) Lewis (2000). El proceso continúa recursivamente hasta obtener las hojas del árbol. El algoritmo 1 muestra el pseudocódigo simplificado de CART.

En la figura 2.5 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.

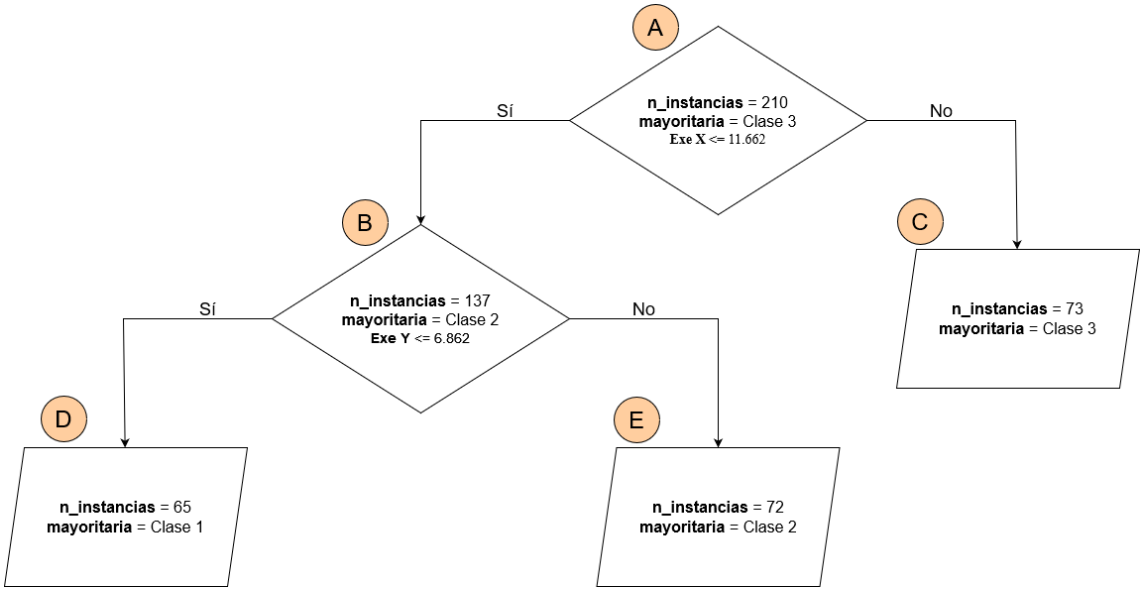

Figura 2.5: ***Árbol de decisión.***

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 Lewis (2000). 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.

2.2.1. 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.

$$\text{Gini}(E) = 1 - \sum_{i=1}^{c} p_i^2 \tag{2.1}$$

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

Tomando el ejemplo de la figura 2.5, los coeficientes de Gini serían los de la tabla 2.1.

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 2.1: *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.

$$\text{Entropy}(E) = -\sum_{i=1}^{c} p_i \cdot \log_2(p_i) \tag{2.2}$$

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 2.2: *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 2.6 se pueden observar las fronteras de decisión para una clasificación perfecta mediante Gini y su cálculo de impureza.

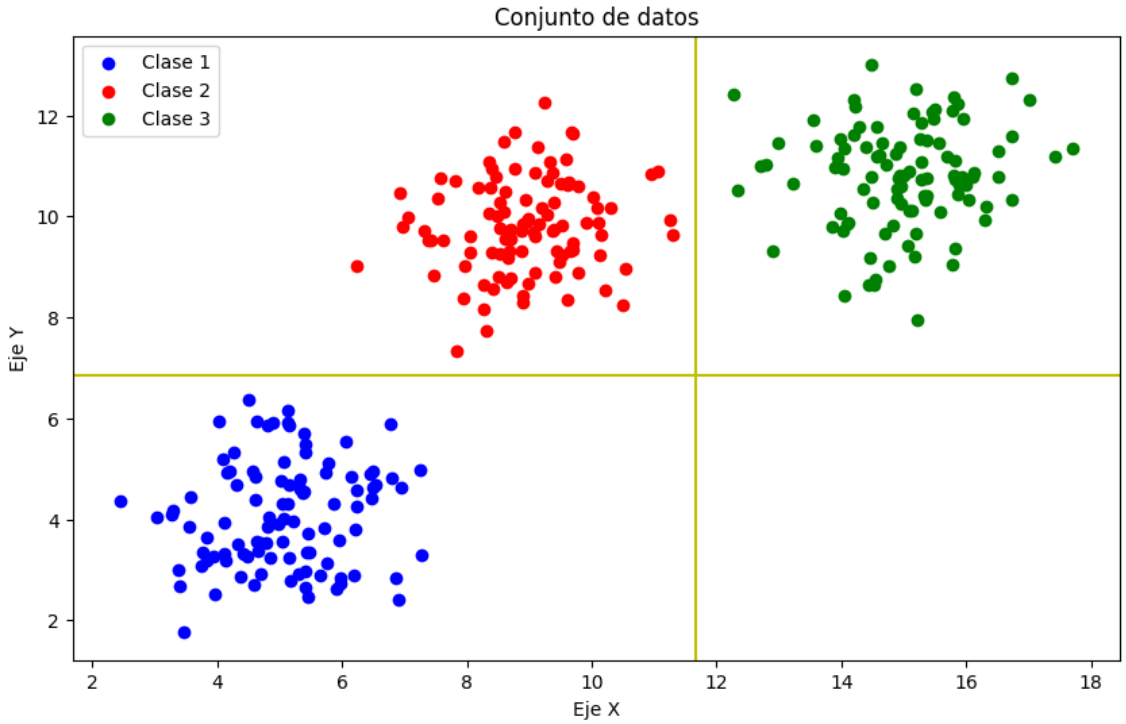

Figura 2.6: ***Fronteras de decisión.***

2.3. 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* 2.1.3.1. 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* Song et al. (2022).

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 $E = \{e_1, e_2, ...\}$ Deo (2017).

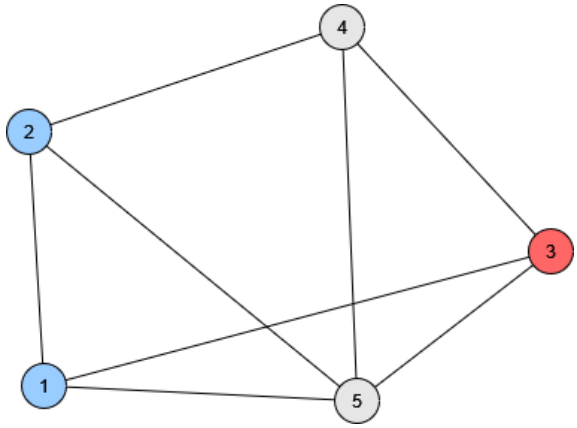

Figura 2.7: ***Ejemplo de grafo.***

En GSSL, cada ejemplo del conjunto de datos representa un vértice o nodo y estarán conectados por aristas ponderadas que representarán la similitud. En la figura 2.7 puede verse un ejemplo de un grafo (los nodos grises representan ejemplos no etiquetados).

Los métodos GSSL se fundamentan en dos pasos Song et al. (2022):

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.

Objetivos

3

3.1. Objetivo general

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 Self-Training o Co-Forest.

3.2. 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). Incluye también de la adición de algoritmos complementarios como *post-pruning*.

3. Implementar desde cero varios algoritmos de construcción de grafo 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, tanto los que cumplen las suposiciones del aprendizaje semi-supervisado como los que no, para una validación exhaustiva que refleje la utilización de estos algoritmos 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.

Metodología y Desarrollo

4

En este aparatado se expondrán todas las decisiones tomadas para el desarrollo de los algoritmos seleccionados. En las primeras secciones se comentarán las decisiones generales que afectarán a los algoritmos. A continuación, se desarrollarán los algoritmos seleccionados.

4.1. 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) Derrac et al. (2015) 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

4.2. Árbol intrínsecamente semi-supervisado (SSLTree)

En la figura 2.4 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 van Engelen y Hoos (2020).

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 pretendían construir árboles semi-supervisados, muchos de ellos con objetivos de aplicación específicos (clasificación de imágenes o análisis de supervivencia, entre otros). Sin embargo, y como se buscaba, en el artículo Levatić et al. (2017) proponen un cálculo de impureza **genérico** que incorpora en sus cálculos los datos no etiquetados.

4.2.1. 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 Levatić et al. (2017) 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):

$$\text{Impurity}(E) = \text{Gini}(E, Y) \tag{4.1}$$

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):

$$\text{Impurity}_{\text{SSL}}(E) = \underbrace{w \cdot \text{Impurity}(E_l, Y)}_{\textit{Target space}} + \underbrace{\frac{1-w}{D} \cdot \sum_{i=1}^{D} \text{Impurity}(E, X_i)}_{\textit{Descriptive space}} \tag{4.2}$$

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:

$$\text{Impurity}(E_l, Y) = \frac{\text{Gini}(E_l, Y)}{\text{Gini}(E_l^{\text{train}}, Y)} \tag{4.3}$$

$$\text{Impurity}(E, X_i) = \frac{\text{Var}(E, X_i)}{\text{Var}(E_{\text{train}}, X_i)} \tag{4.4}$$

La variabilidad se calcula como:

$$\text{Var}(E, X_i) = \frac{\sum_{j=1}^{N} (x_i^j)^2 - \frac{1}{N} \cdot (\sum_{j=1}^{N} x_i^j)^2}{N} \tag{4.5}$$

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

[2] Y es la variable de etiquetas.

se está calculando la medida.

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 4.4. En el artículo original Levatić et al. (2017) la ecuación 4.4 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.

4.2.2. Intuición del nuevo cálculo

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

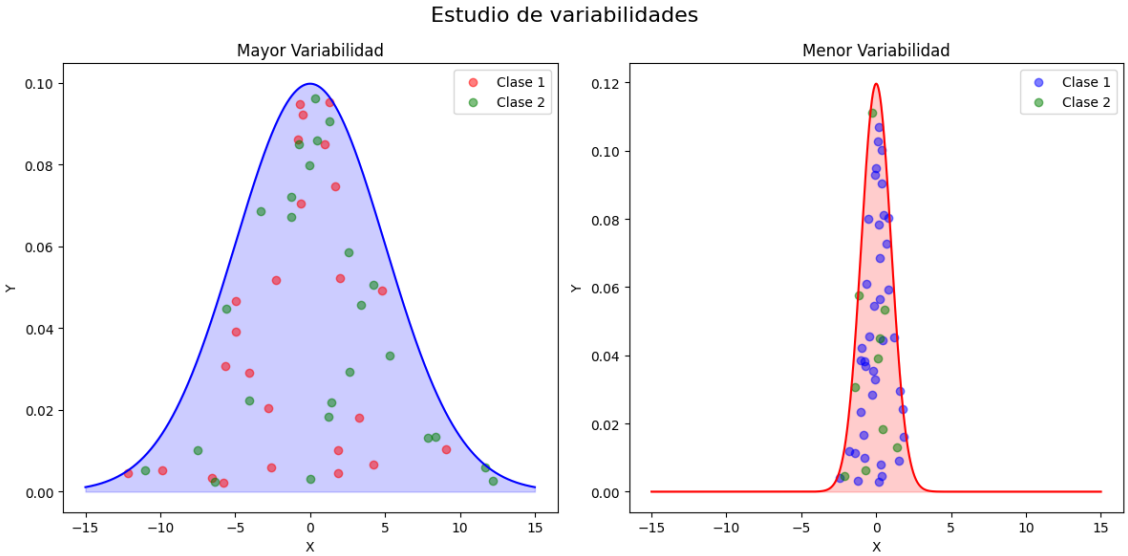

Figura 4.1: ***Suposición de la variabilidad. No representa un ejemplo real, solo es ilustrativo.***

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 4.1, 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 4.1. Esto es porque **datos similares en las características** tienden a tener las **mismas etiquetas** (muy relacionado con las suposiciones en 2.1.3.1).

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 4.2, donde claramente hay poca homogeneidad y la clasificación no sería buena.

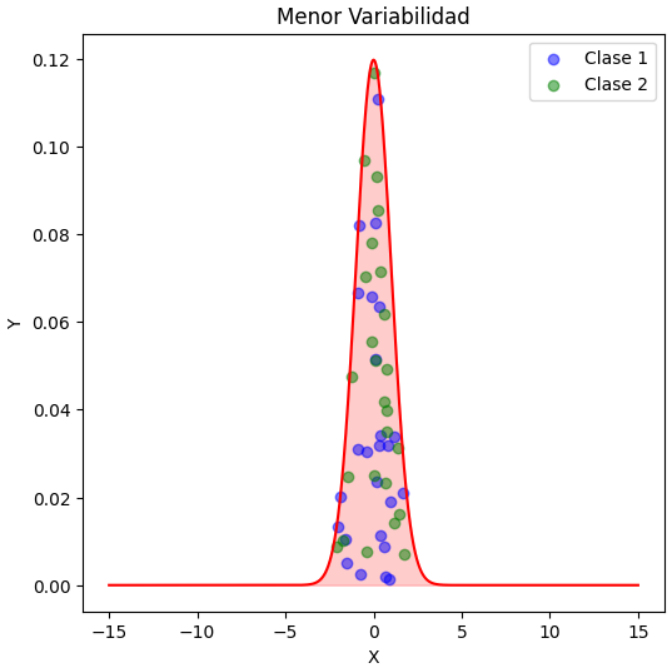

Figura 4.2: ***Incumplimiento de la suposición de la variabilidad.***

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 (en los casos en los que se cumplan las suposiciones). 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.

4.2.3. 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 va a realizar 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.

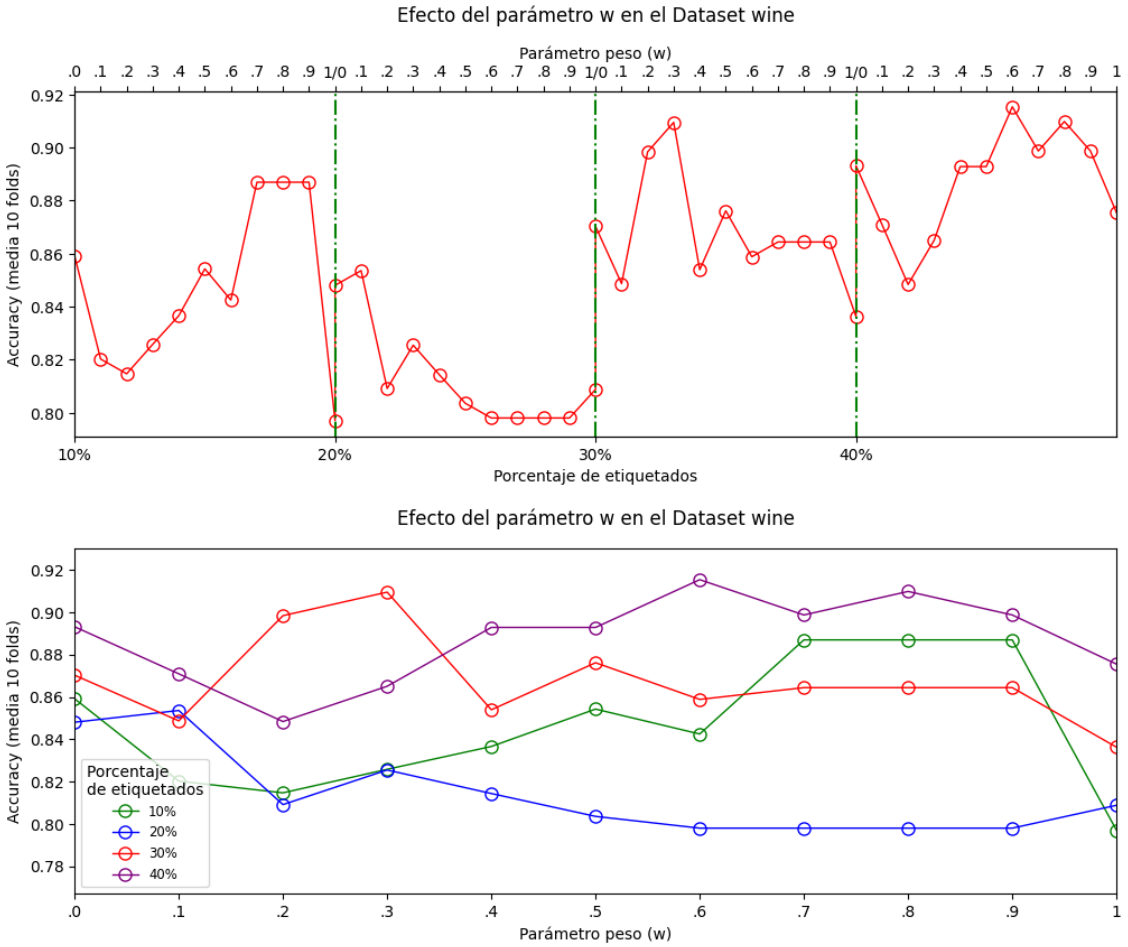

Figura 4.3: ***Ejemplo del efecto de w en el dataset Wine.***

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.

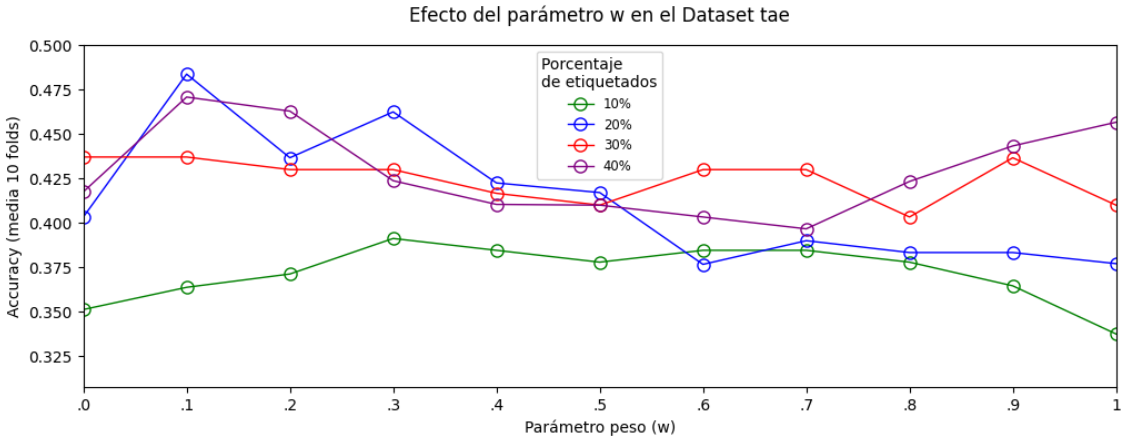

Figura 4.4: ***Ejemplo del efecto de w en el dataset Tae.***

En esta última imagen 4.4 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.

4.2.4. Experimentación

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*. Self-Training 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.

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.

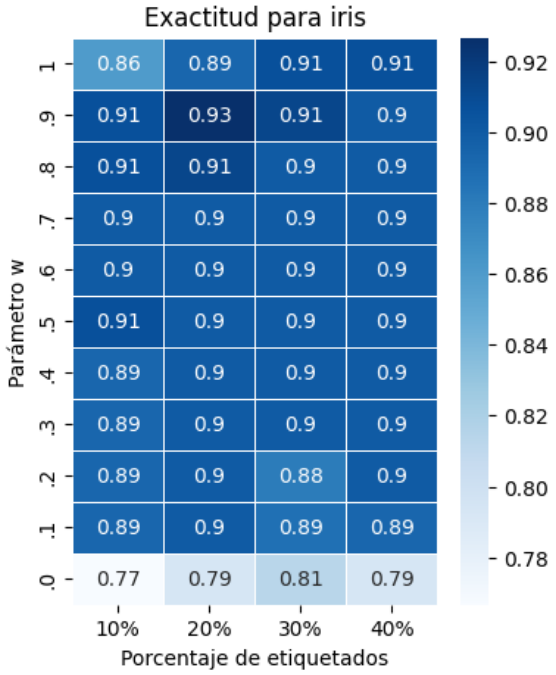

Figura 4.5: ***Mapa de calor de w para el dataset Iris.***

La figura 4.5 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 4.6. Un resultado muy similar con el cálculo de Gini puede encontrarse en A.1.

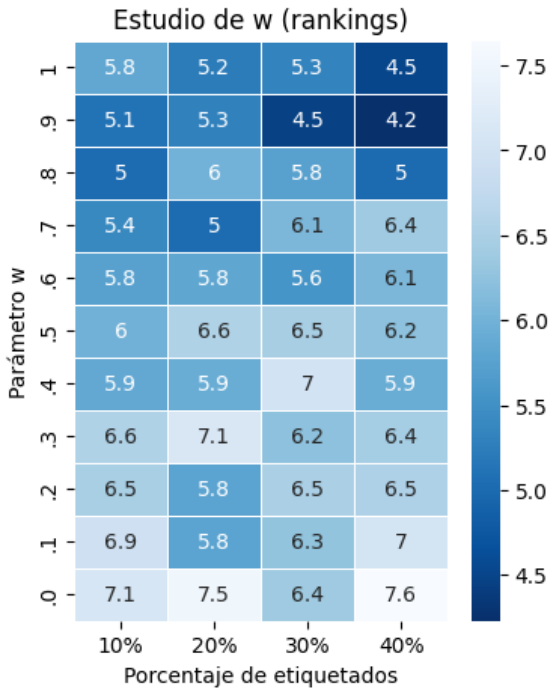

Figura 4.6: ***Mapa de calor de los rankings medios de*** w***.***

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 4.1: *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 4.7).

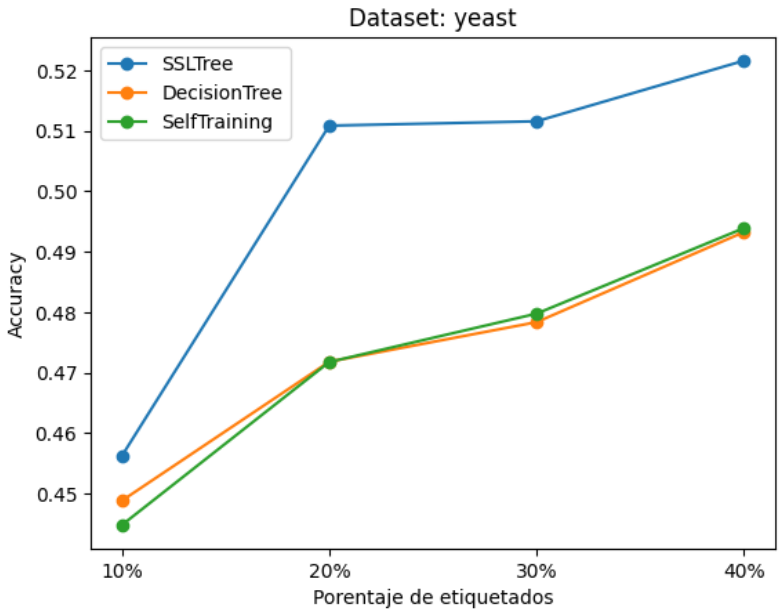

Figura 4.7: ***Comparativa en dataset Yeast.***

La figura 4.7 se ha seleccionado convenientemente. Existirán conjuntos de datos en los que SSLTree sea igual o mínimamente peor cuando no se cumplan las suposiciones comentadas.

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.

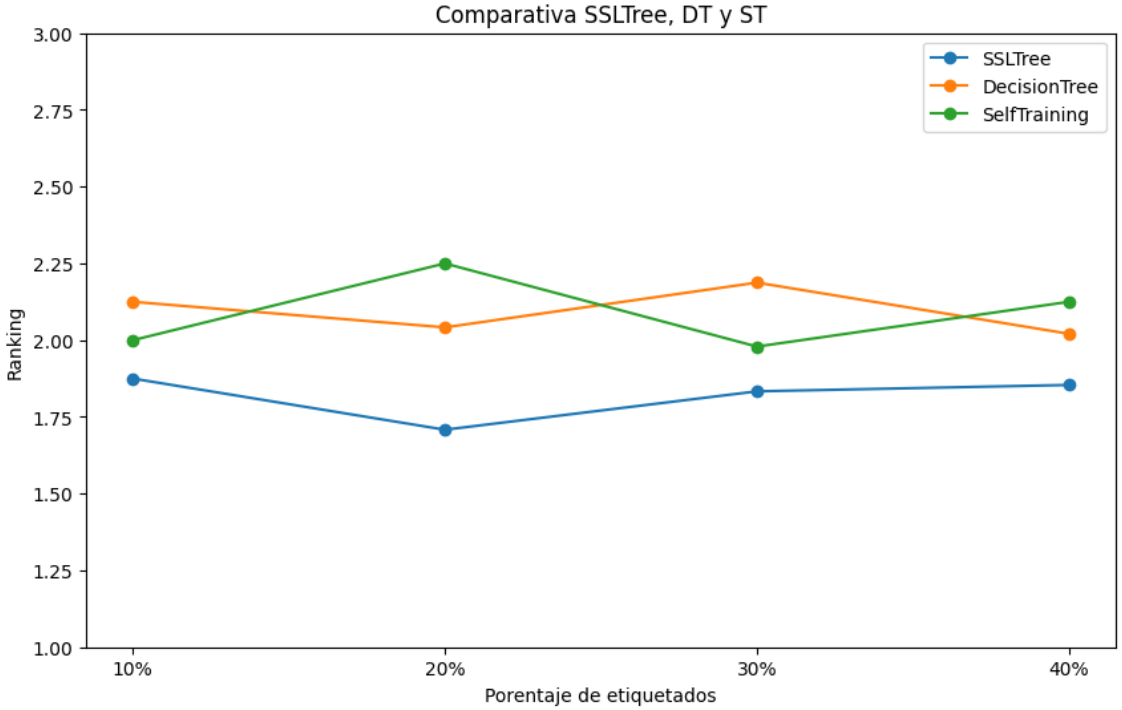

Figura 4.8: ***Ranking promedio de cada modelo para todos los datasets.***

Los resultados obtenidos[3] 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 4.9, 4.10, 4.11 y 4.12. 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.

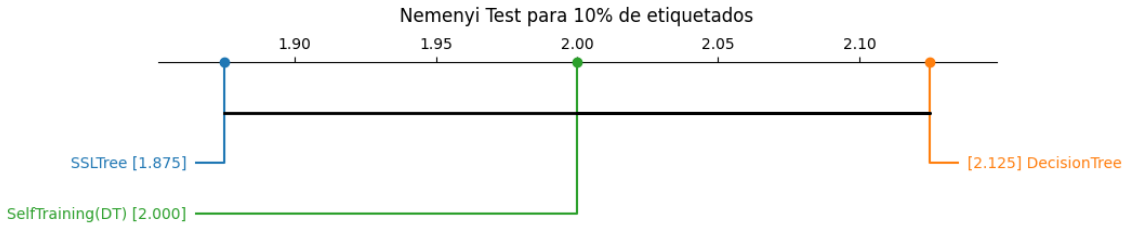

Figura 4.9: ***Comparativa básica: Nemenyi Test para 10 % de etiquetados.***

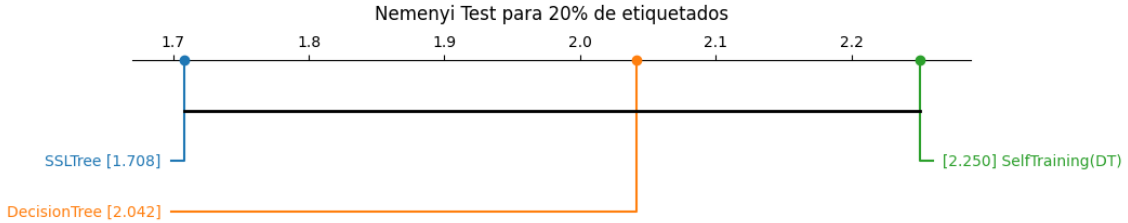

Figura 4.10: ***Comparativa básica: Nemenyi Test para 20 % de etiquetados.***

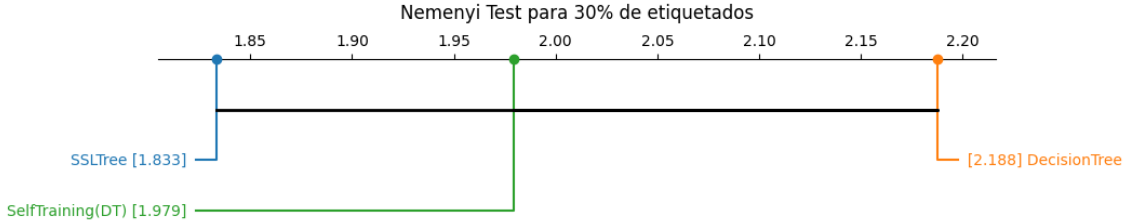

Figura 4.11: ***Comparativa básica: Nemenyi Test para 30 % de etiquetados.***

[3]Pueden comprobarse resultados similares para Gini en A.2.

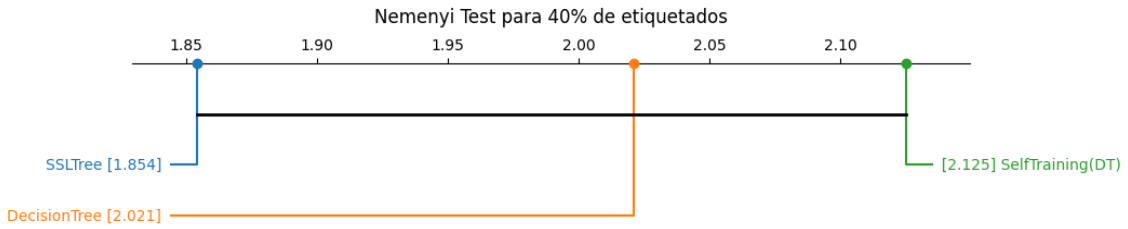

Figura 4.12: ***Comparativa básica: Nemenyi Test para 40 % de etiquetados.***

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) Rokach (2010).

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*[4] y subespacios aleatorios en cada árbol (cada árbol, aleatoriamente, solo utilizará parte de las características).

2. CoForest con SSLTree como estimador base. CoForest se puede entender como la implementación del Random Forest en el aprendizaje semi-supervisado Li y Zhou (2007).

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 4.13. 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 Salles et al. (2021) y Dorador (2024)), pueden superar a muchos de los modelos semi-supervisados solo con la porción etiquetada como ha ocurrido en este experimento.

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

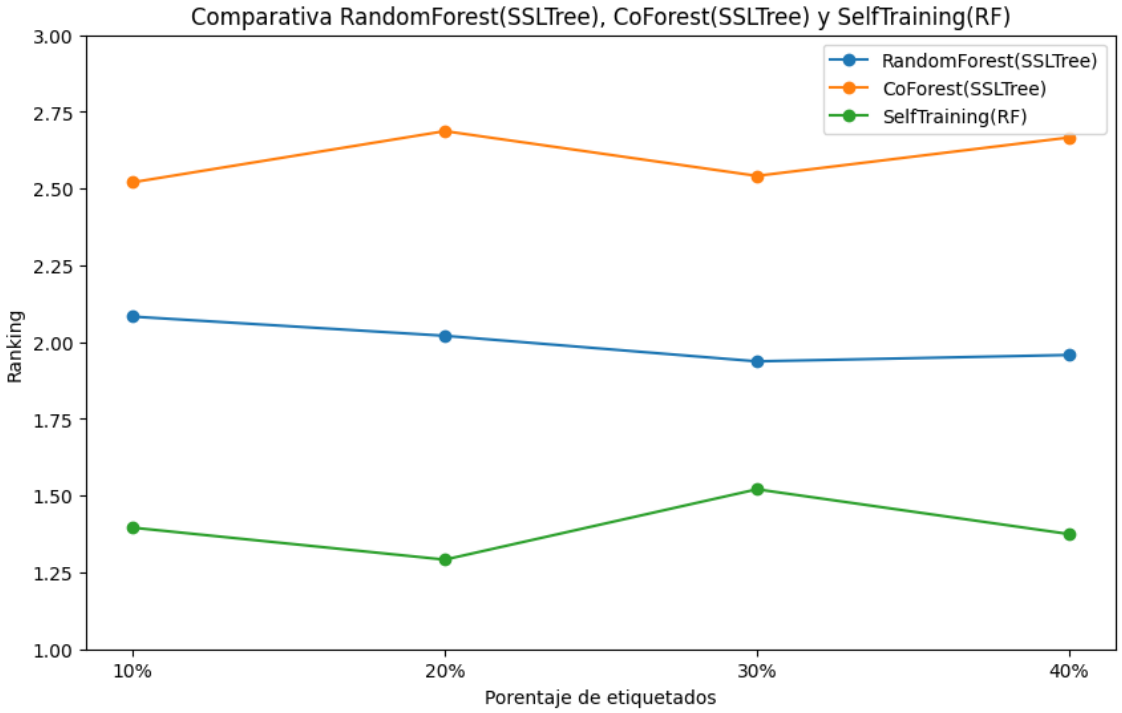

Figura 4.13: ***Ranking promedio de cada ensemble para todos los datasets.***

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 4.14, 4.15, 4.16 y 4.17.

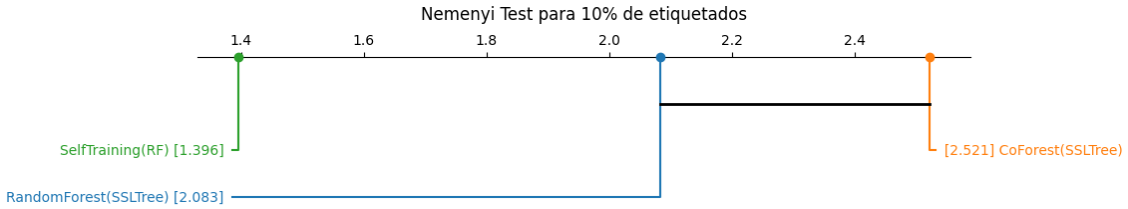

Figura 4.14: ***Comparativa básica: Nemenyi Test para 10 % de etiquetados.***

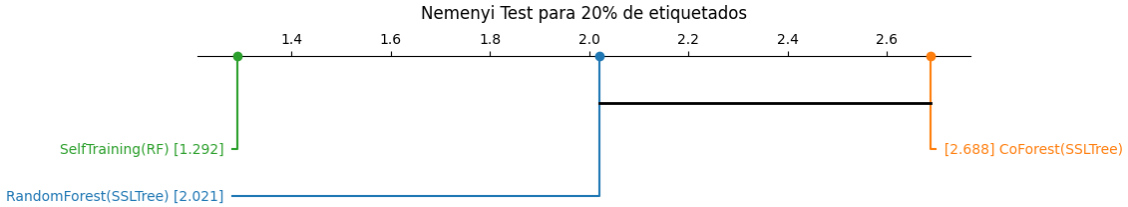

Figura 4.15: ***Comparativa básica: Nemenyi Test para 20 % de etiquetados.***

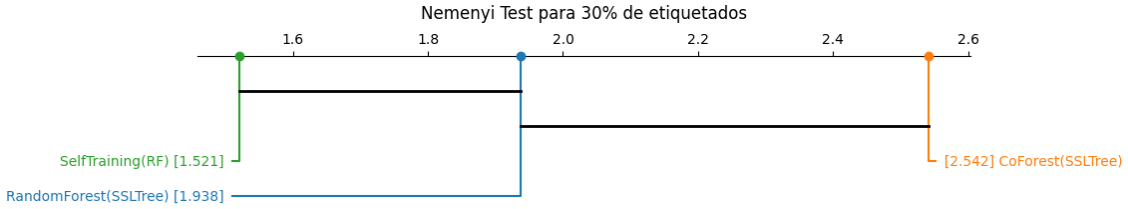

Figura 4.16: ***Comparativa básica: Nemenyi Test para 30 % de etiquetados.***

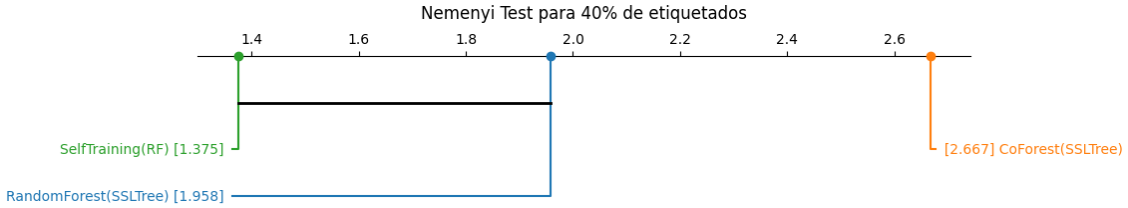

Figura 4.17: ***Comparativa básica: Nemenyi Test para 40 % de etiquetados.***

4.2.5. 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 Mingers (1989) 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 de 4.2.4 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 Gordon et al. (1984).

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

$$R_\alpha(T) = R(T) + \alpha \cdot |f(T)| \tag{4.6}$$

donde $R(T)$ es el error de entrenamiento 4.7, $f(T)$ devuelve el conjunto de hojas del árbol T y α es el parámetro de regularización (hiper-parámetro fijado).

$$R(T) = \sum_{t \in f(T)} r(t) \cdot p(t) = \sum_{t \in f(T)} R(t) \tag{4.7}$$

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

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

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

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 4.7 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[5]. En la figura 4.18 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.

[5]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 $\alpha^1 = 0$ minimizando $R(T)$.
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.

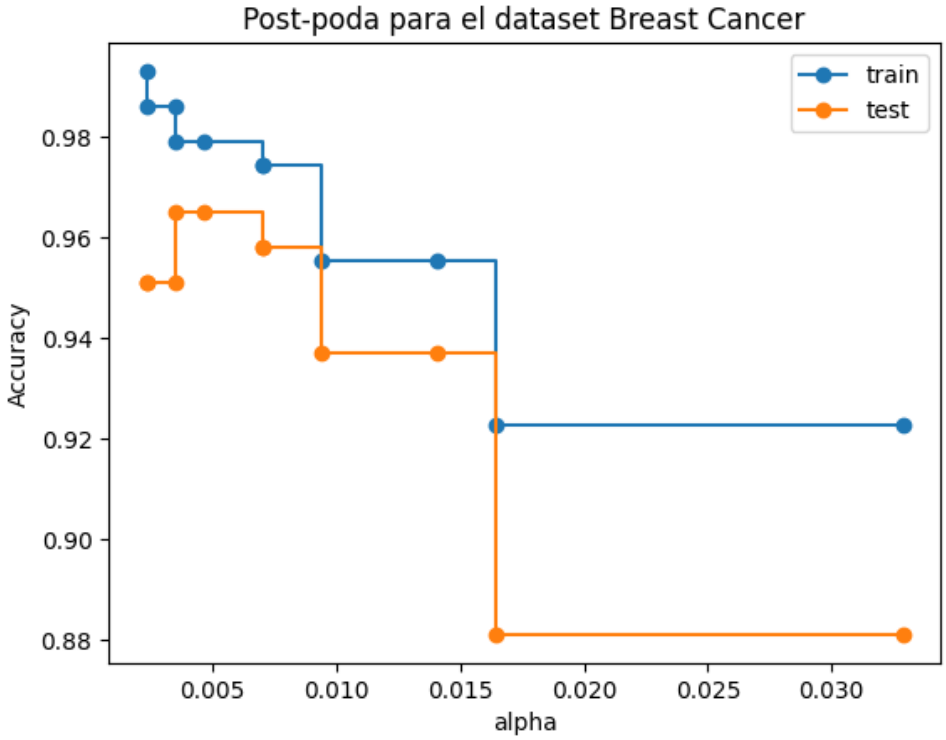

Figura 4.18: ***Beneficio de la post-poda en SSLTree para el dataset Breast Cancer.***

4.3. Grafos semi-supervisados

4.3.1. 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 en Berton y Lopes (2014) 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 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 de diferentes grupos.

$$b \in \text{kNN}(a) \wedge a \in \text{kNN}(b)$$

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. Ver ecuación 4.8.

$$\text{mín} \sum_i \sum_j \left(D_{ij} + \sum_l D_{jl} \right) \tag{4.8}$$

La intuición de la ecuación 4.8 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, el pseudocódigo del algoritmo es:

Algoritmo 3: Algoritmo 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) $\notin L$ **then**
    for $k = 1; k <$ Lista-kNN(v_i)$; k++$ **do**
      if Componente(v_k) $\in L$ **then**
        Guardar e_{ik} en G
devolver G

4.3.2. Experimentación

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 desarrollados.

Los algoritmos desarrollados quieren ser probados como una unidad. Es decir, no se quiere comparar RGCLI y GBILI entre ellos u otros métodos de construcción de grafos, si no 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 hiperparámetro α que posee.

Se necesita tener grafos construidos, para ello se utilizarán ambos RGCLI y GBILI como métodos de creación de grafos. Para RGCLI, los autores CITAAAAR fijan el parámetro $k_e = 50$ y $k_i = 2$ para problemas de clasificación. Para GBILI el estudio realizado por sus autores Berton y Lopes (2014) concluyen que el *accuracy* se estabiliza para un valor de $k > 10$ por lo que se ha fijado a $k = 11$.

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 ejecutarán 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 realizarán por cada conjunto de datos y se realizará 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 4.19.

Conjuntos de datos utilizados

Los métodos de construcción de grafos son muy costosos computacionalmente. 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.

4.3.2.1. Transductivo

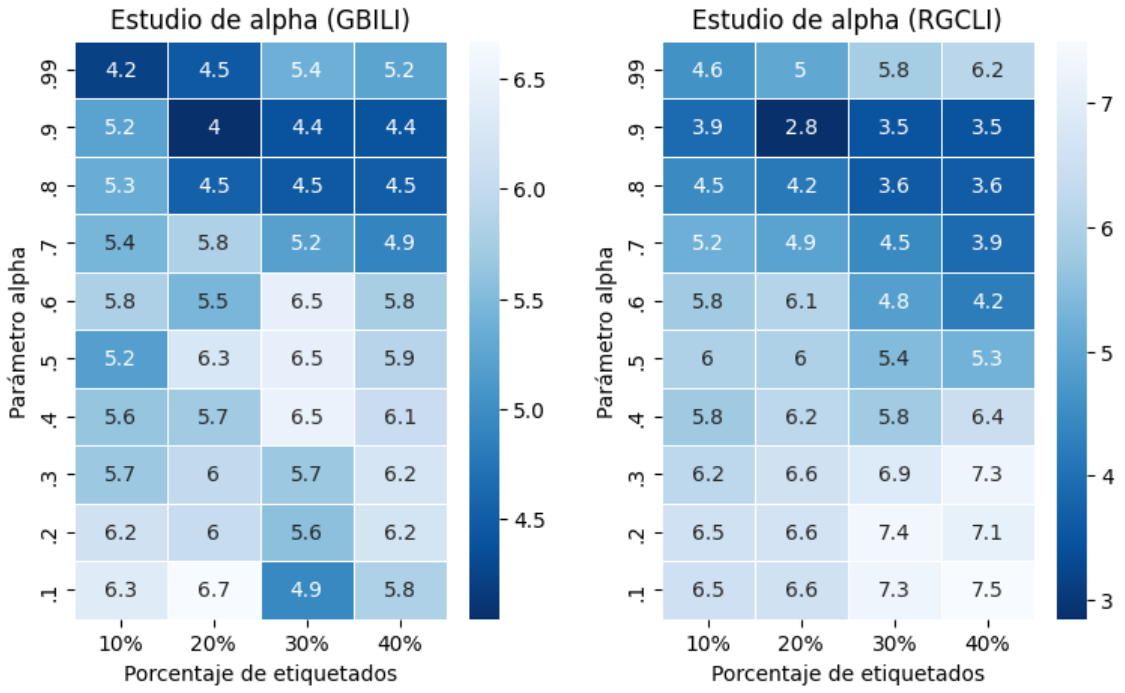

Figura 4.19: ***Mapas de calor de los rankings medios de α.***

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

.99	**.9**	.8	.7	.6	.5	.4	.3	.2	.1
4.82	**4.52**	4.71	5.34	5.87	5.95	5.98	5.89	5.99	5.95

Tabla 4.2: *Ranking promedio de cada valor de α con GBILI*

.99	**.9**	.8	.7	.6	.5	.4	.3	.2	.1
5.41	**3.43**	3.97	4.62	5.24	5.67	6.04	6.73	6.92	6.97

Tabla 4.3: *Ranking promedio de cada valor de α con RGCLI*

Para .99, .9 y .8, y sobretodo para GBILI, los resultados son bastante similares y aunque el valor .9 parezca el mejor, puede que disminuyendo la distancia de los intervalos pueda obtenerse un valor incluso más concreto. Para ello, se realizarán las mismas pruebas pera esta vez para valores de α entre .8 y .99 con intervalos de .01. El proceso es el mismo descrito anteriormente. Los resultados pueden verse en la figura 4.20.

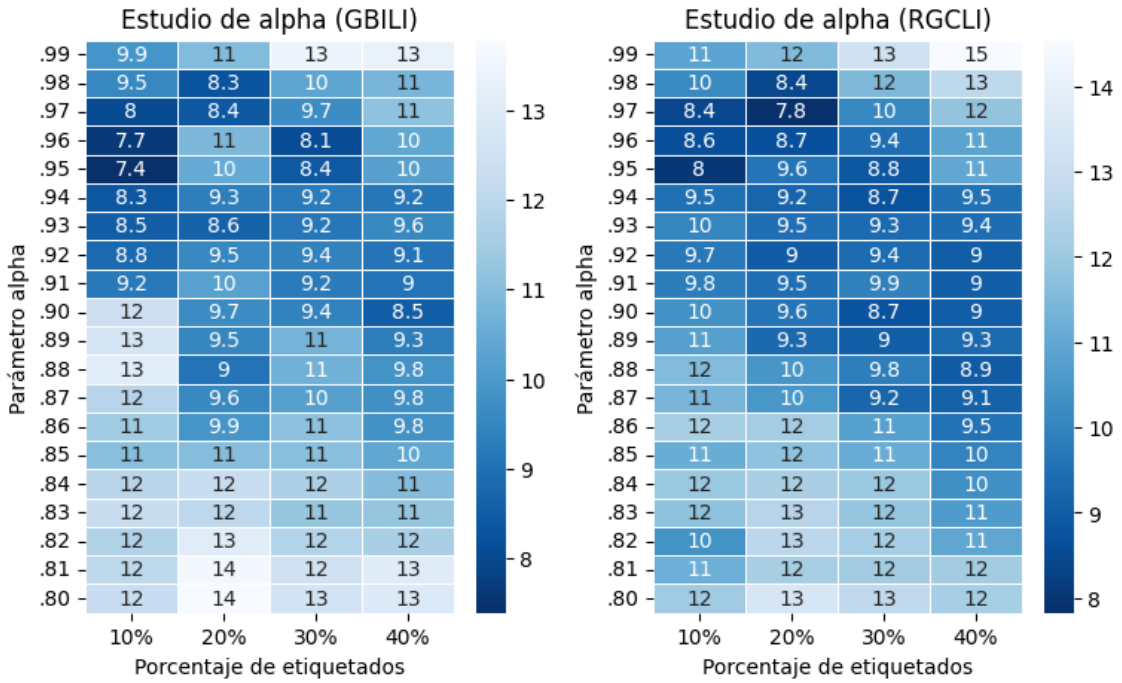

Figura 4.20: ***Mapas de calor de los rankings medios de α entre .8 y .99.***

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

.95	.94	**.93**	.92	.91
9.11	9.01	**8.96**	9.20	9.41

Tabla 4.4: *Ranking promedio de cada valor de α entre .8 y .99 con GBILI*

Según estos nuevos resultados, se va a seleccionar el valor 0.94 para LGC. Esto es porque aunque para GBILI, el mejor era 0.93, su diferencia con 0.94 es muy pequeña. La otra posibilidad era seleccionar 0.93, sin embargo, en RGCLI sí que hay una diferencia muy grande entre el 0.94 y 0.93 como para obviarla como se ha hecho en GBILI.

.96	.95	**.94**	.93	.92
9.40	9.36	**9.23**	9.57	9.27

Tabla 4.5: *Ranking promedio de cada valor de α entre .8 y .99 con RGCLI*

4.3.2.2. Inductivo

Resultados y Discusión

5

Conclusiones

6

1. Conclusión 1.
2. Conclusión 2.
3. Conclusión 3.

Limitaciones y Perspectivas de Futuro

7

Resultados experimentación

Aquí se incluirán todos los resultados *adicionales* que, por completitud, se desean incluir de la experimentación propuesta en 4.

A.1. Estudio del parámetro w

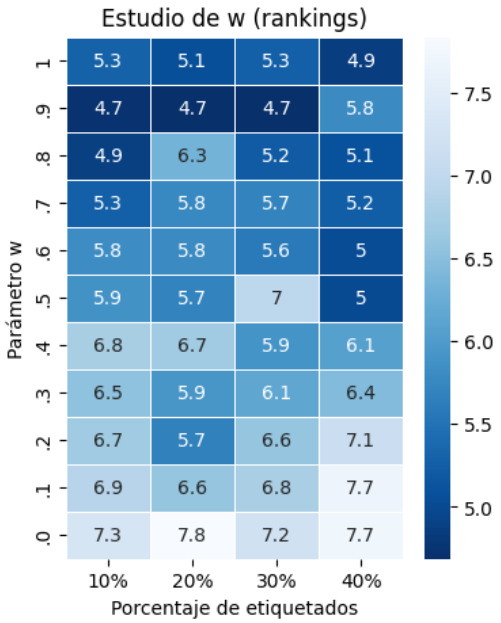

Figura A.1: ***Mapa de calor de los rankings medios (Gini).***

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

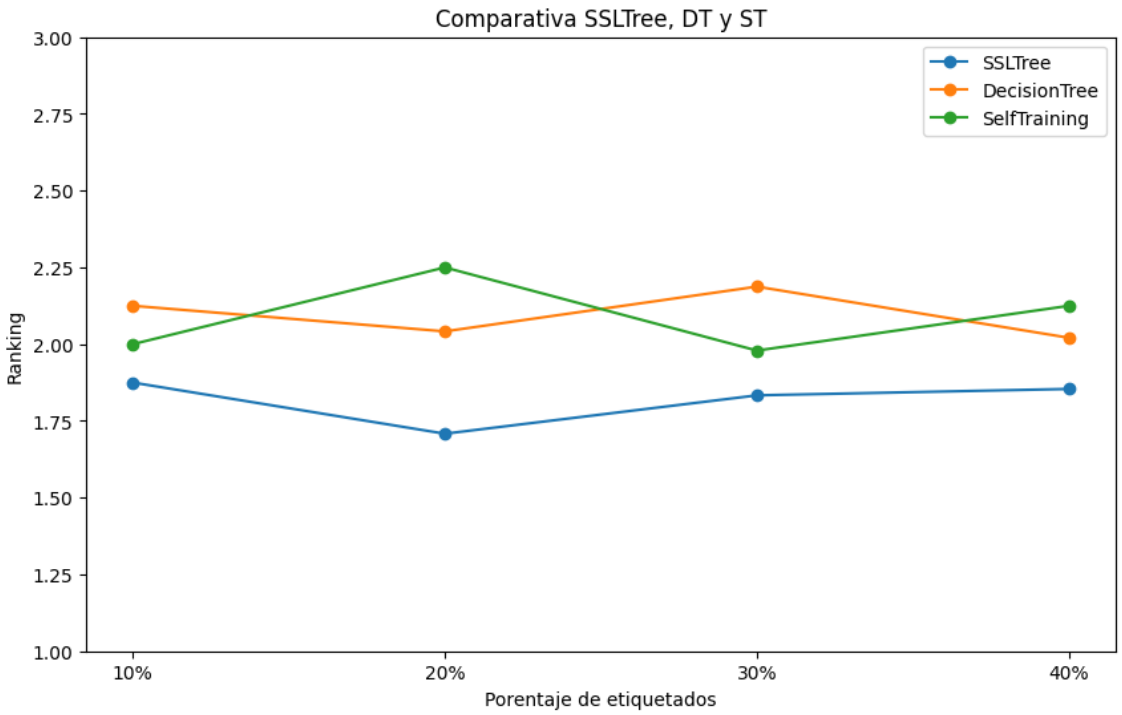

Figura A.2: ***Ranking promedio de cada modelo para todos los datasets (Gini).***

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.

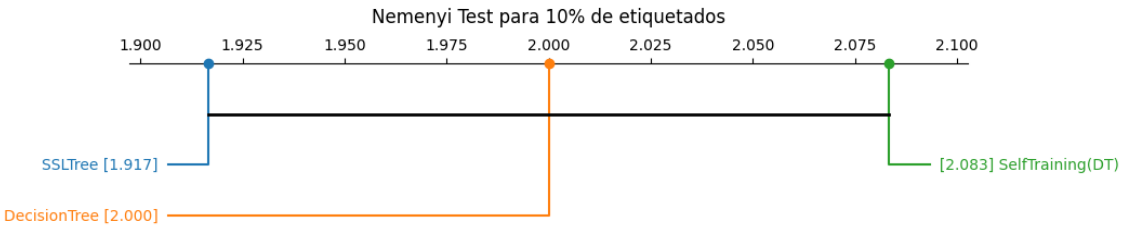

Figura A.3: ***Comparativa básica: Nemenyi Test para 10 % de etiquetados (Gini).***

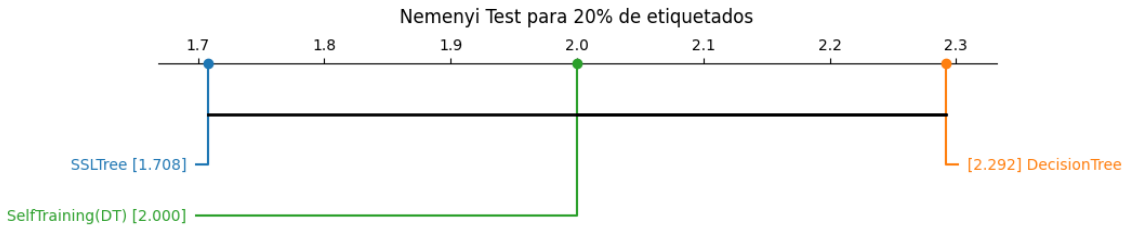

Figura A.4: ***Comparativa básica: Nemenyi Test para 20 % de etiquetados (Gini).***

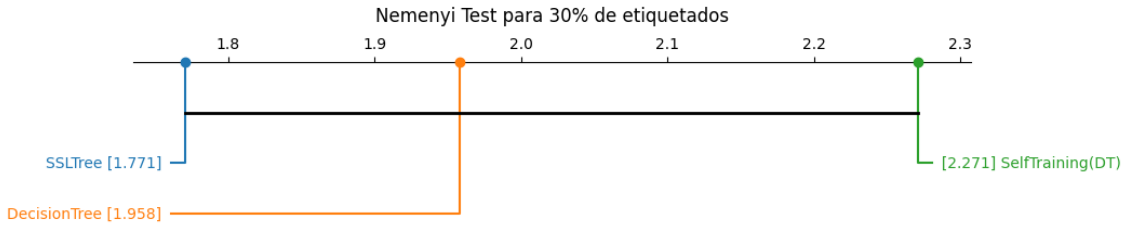

Figura A.5: ***Comparativa básica: Nemenyi Test para 30 % de etiquetados (Gini).***

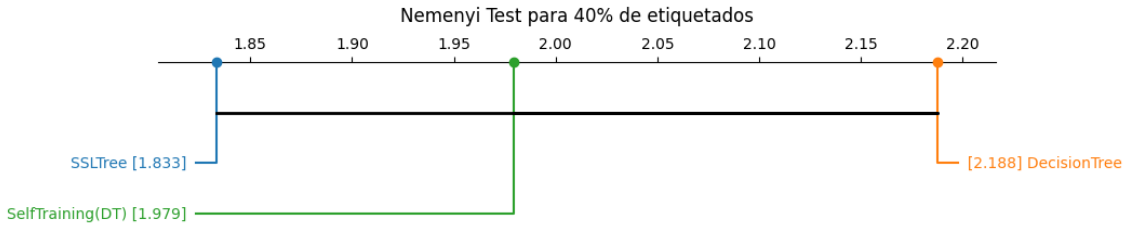

Figura A.6: ***Comparativa básica: Nemenyi Test para 40 % de etiquetados (Gini).***

Apéndize B

B

Bibliografía

Alexander S. Gillis, D. P. (2021). Supervised learning. https://www.techtarget.com/searchenterpriseai/definition/supervised-learning. [Internet; descargado 18-abril-2024].

Berton, L. y Lopes, A. D. A. (2014). Graph construction based on labeled instances for semi-supervised learning. In *2014 22nd international conference on pattern recognition*, pages 2477–2482. IEEE.

Breiman, L. (2017). *Classification and regression trees*. Routledge.

Deo, N. (2017). *Graph theory with applications to engineering and computer science*. Courier Dover Publications.

Derrac, J., Garcia, S., Sanchez, L., y Herrera, F. (2015). Keel data-mining software tool: Data set repository, integration of algorithms and experimental analysis framework. *J. Mult. Valued Logic Soft Comput*, 17:255–287.

Dorador, A. (2024). Improving the accuracy and interpretability of random forests via forest pruning. *arXiv preprint arXiv:2401.05535*.

Dridi, S. (2021a). Supervised learning - a systematic literature review. *ResearchGate*.

Dridi, S. (2021b). Unsupervised learning - a systematic literature review. *ResearchGate*.

Gordon, A., Breiman, L., Friedman, J., Olshen, R., y Stone, C. J. (1984). Classification and regression trees. *Biometrics*, 40(3):874.

javaTpoint. Unsupervised machine learning. https://www.javatpoint.com/unsupervised-machine-learning. [Online; accessed 18-April-2024].

Kingsford, C. y Salzberg, S. L. (2008). What are decision trees? *Nature biotechnology*, 26(9):1011–1013.

Levatić, J., Ceci, M., Kocev, D., y Džeroski, S. (2017). Semi-supervised classification trees. *Journal of Intelligent Information Systems*, 49:461–486.

Lewis, R. J. (2000). An introduction to classification and regression tree (cart) analysis. In *Annual meeting of the society for academic emergency medicine in San Francisco, California*, volume 14. Citeseer.

Li, M. y Zhou, Z.-H. (2007). Improve computer-aided diagnosis with machine learning techniques using undiagnosed samples. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 37(6):1088–1098.

Lukas Huber (2022). A friendly intro to semi-supervised learning. https://towardsdatascience.com/a-friendly-intro-to-semi-supervised-learning-3783c0146744. [Online; accessed 18-April-2024].

Martel, J. (2020). Machine learning: qué es y cómo funciona. https://itelligent.es/es/machine-learning-que-es-como-funciona/. [Internet; descargado 18-abril-2024].

Mingers, J. (1989). An empirical comparison of pruning methods for decision tree induction. *Machine learning*, 4:227–243.

Quinlan, J. R. (1986). Induction of decision trees. *Machine learning*, 1:81–106.

Quinlan, J. R. (2004). C5. 0. *http://www. rulequest. com/see5-info. html*.

Quinlan, J. R. (2014). *C4. 5: programs for machine learning*. Elsevier.

Rokach, L. (2010). Ensemble-based classifiers. *Artificial intelligence review*, 33:1–39.

Salles, T., Rocha, L., y Gonçalves, M. (2021). A bias-variance analysis of state-of-the-art random forest text classifiers. *Advances in Data Analysis and Classification*, 15:379–405.

Solutions, N. T. (2018). Machine learning algorithms: Beginners guide part 1. https://www.neovasolutions.com/2018/06/06/machine-learning-algorithms-beginners-guide-part-1/. [Internet; descargado 18-abril-2024].

Song, Z., Yang, X., Xu, Z., y King, I. (2022). Graph-based semi-supervised learning: A comprehensive review. *IEEE Transactions on Neural Networks and Learning Systems*, 34(11):8174–8194.

van Engelen, J. E. y Hoos, H. H. (2020). A survey on semi-supervised learning. *Machine Learning*, 109(2):373–440.

Wikipedia contributors (2024). Training, validation, and test data sets — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Training,_validation,_and_test_data_sets&oldid=1218746717. [Online; accessed 18-April-2024].