

# EL ÁRBOL SOCIAL

Antonio Macías Ferrera  
*Universidad de Sevilla*  
Sevilla, España  
antoniomff@gmail.com

Delfín Santana Rubio  
*Universidad de Sevilla*  
Sevilla, España

—

**Resumen**—El proyecto que nos ocupa consiste en la exploración de nuevos métodos de aprendizaje automático para tareas de clasificación. En concreto, consiste en la obtención de métricas relacionales a partir de un conjunto de datos en forma de grafo que sirvan como “features” de los elementos a clasificar. Usando el lenguaje de programación *Python* y ayudándonos de librerías, hemos obtenido métricas relacionales y con estas hemos evaluado modelos de aprendizaje supervisado para averiguar cuál de ellos es el mejor para la tarea de clasificación de una página de *Facebook* en uno de los siguientes grupos: *políticos, organizaciones gubernamentales, programas de televisión y empresas*. Tras entrenar y evaluar varios modelos con varias combinaciones de métrica e hiperparámetros, se concluyó que un algoritmo de árboles de clasificación (CART) es el más adecuado para esta tarea de aprendizaje automático.

**Palabras clave**—Inteligencia Artificial, aprendizaje automático, clasificación relacional, grafo, nodo, centralidad, políticos, organizaciones, programas, empresas, métrica, comunidades, clustering, árbol, nodo, grafo.

## I. INTRODUCCIÓN

En este trabajo para la asignatura de Inteligencia Artificial, se ha abordado el estudio de un problema de clasificación multiclaso relacional (clasificación relacional) partiendo de un conjunto de datos en forma de grafo. El estudio ha consistido en analizar el conjunto de datos para extraer de ellos una serie de métricas útiles para clasificar elementos según sus relaciones, y así poder elaborar un modelo de aprendizaje automático capaz de clasificar un nodo dentro de la red.

En concreto, se ha empleado un dataset de la red social *Facebook* obtenidos a través del banco de datos de la Universidad de Stanford [1]. Este conjunto de datos contiene información de 22,470 páginas de *Facebook* en forma de grafo no dirigido: los nodos representan las páginas oficiales de *Facebook*, mientras que los enlaces son “me gusta” mutuos entre los sitios. Las características (*features*) iniciales del nodo se extraen de las descripciones los sitios. Estos datos fueron recopilados a través de la API de *Facebook Graph* en noviembre de 2017 y se restringió a las páginas de 4 categorías que están definidas por *Facebook*. Estas categorías son: **políticos, organizaciones gubernamentales, programas de televisión y empresas**.

Para procesar el grafo, se ha hecho uso de librerías de *Python* (principalmente *NetworkX*), con la que se ha representado el grafo de distintas formas. Así, se intentó observar

a simple vista algunas características que pudieran diferenciar a los nodos según su tipo, no obstante, los resultados no fueron concluyentes. Sobre la base de lo anterior se procedió a analizar y evaluar distintas métricas relacionales del grafo. Una vez seleccionadas las mejores métricas, el trabajo consistió en entrenar distintos modelos de aprendizaje automático con cada una de ellas, y evaluar cuáles podían ser las mejores combinaciones de distintos modelos con distintas métricas. Así, se pretende obtener una combinación óptima para entrenar a un modelo de que ayude a determinar a qué tipo de grupo de *Facebook* puede pertenecer una página dada una serie de características.

## II. PRELIMINARES

En esta parte se hace una descripción de las técnicas empleadas para abordar el problema:

### II-A. Métodos empleados

A continuación, se enumerarán los métodos y técnicas empleadas para abordar el problema. Se podrán observar, en orden de utilización, las librerías de *Python* y los métodos principales usados en cada etapa del proyecto.

- *Representación de un grafo*: En primer lugar, para representar el grafo, se ha empleado la librería *NetworkX*, que nos ha permitido visualizar el grafo para poder analizar a simple vista las características que podrían diferenciar a los nodos según su tipo. Algunos de los métodos más usados han sido: *draw\_networkx\_graph*, *random\_layout* y *draw\_networkx\_edges*.
- *Evaluación de métricas*: Para evaluar el conjunto de métricas, primero se han estimado estas métricas sobre el conjunto de datos, y a continuación se ha procedido a estimar si estas son suficientemente diferentes para cada tipo de nodo. Si esto es así, las métricas podrán ser adecuadas ‘a priori’ para clasificar los nodos. Para ello, se ha empleado la librería *scipy* que con el método *ttest\_ind* realiza un análisis de t-student para comparar si dos conjuntos de datos son suficientemente diferentes.
- *Obtención de métricas relacionales*: Se ha empleado de igual forma la librería *NetworkX* de la cual hemos usado los siguientes métodos. En el siguiente apartado se describirán con más detalle cada una de las métricas II-B.

Cuadro I  
MÉTRICAS RELACIONALES

Medidad	Método <i>NetworkX</i>
Centralidad de grado	<i>degree_centrality</i>
Centralidad k-path	(Se ha elaborado un método propio)
Centralidad de cercanía	<i>closeness_centrality</i>
Centralidad de intermejación	<i>betweenness_centrality</i>
Centralidad harmónica	<i>harmonic_centrality</i>
Coeficiente de clusterin	<i>clustering</i>
Coeficiente de clusterin de cuadrados	<i>square_clustering</i>
Identificadores de las comunidades encontradas usando la maximización de la modularidad greedy	<i>greedy_modularity_communities</i>
Identificadores de las comunidades encontradas usando label propagation	<i>label_propagation_communities</i>

- *Entrenamiento y evaluación del modelo:* Se ha utilizado la librería *scikit-learn* usada en las prácticas de la asignatura. Algunos de los métodos para el entrenamiento de los modelos han sido: *GridSearchCV*, *KNeighborsClassifier*, *DecisionTreeClassifier*, *train\_test\_split*, etc.
- *Otras librerías:* Tanto para la representación de grafos como para la obtención de métricas relacionales, se ha requerido de otras librerías como *Pandas*, *Matplotlib*, *itertools*, *SelectFromModel*, *Pipeline*, etc.
- *Node2Vec:* Despues de construir de manera manual las features relacionales, se ha utilizado *Node2Vec* [2]. Este es un método de *node embedding* que utiliza *Random Walks* para construir un corpus que será utilizado por *Word2Vec* [3] para tareas de *machine learning*. Los *Random Walks* son guiados por dos parámetros:  $p$ , probabilidad de revisitar inmediatamente un nodo en el recorrido; y  $q$ , que controla la tendencia que tendrá el camino a la hora de visitar otros nodos más lejanos o cercanos a él. Para utilizarlo, se ha empleado la siguiente librería de *Python*: ↗

## II-B. Trabajo Relacionado

En este apartado se describirán las métricas relacionales que se han usado para abordar la tarea de clasificación. Se comentarán no solo las que se han usado para entrenar a los modelos de aprendizaje automático, sino también las que se evaluaron en primer lugar pero finalmente no se usaron para el entrenamiento. También se describirán brevemente los modelos de aprendizaje automático que se han utilizado.

**II-B1. Métricas de Centralidad:** La centralidad no es un atributo intrínseco de los nodos de una red, sino que es un atributo estructural, por lo que depende de las relaciones con el resto de los nodos de la red. A continuación veremos las medidas de centralidad usadas para clasificar los nodos.

- **Degree centrality:** La centralidad de grado para un nodo  $v$  en un grafo no dirigido se define como el número de adyacencias de ese nodo. Para normalizar esta medida, es usual dividirla por el número de nodos que posee el grafo. [4]

Para cada nodo  $v \in V$ ,  $\delta(v)$  denota el grado de dicho nodo, entonces su centralidad de grado  $C_D(v)$  se define como:

$$C_D(v) = \delta(v)$$

La **complejidad** computacional del cálculo de esta medida toma  $\Theta(V^2)$  para un grafo denso, y  $\Theta(E)$  para un grafo disperso (siendo  $V$  el número total de vértices y  $E$  el número total de aristas). En nuestro caso trabajamos con un grafo disperso, ya que el número de aristas no es cercano al número máximo de aristas que puede tener el grafo. **Densidad del grafo:** **0.00067739872.**

- **K-path centrality:** Es una variante de la centralidad de grado que toma como “grado” al número de caminos de longitud  $k$  (normalmente 1) que conectan a ese nodo  $v$  con otros. En un grafo no dirigido, esta medida equivale a la vecindad considerando una profundidad  $k$ . [4]

- **Closeness centrality:** La centralidad de cercanía para un nodo  $v$  se define como la suma (o promedio) de las longitudes de los caminos más cortos desde ese nodo hacia todos los demás. Como cuanto mayor sea la «distancia» entre dos vértices, menor será la «cercanía» entre estos, la cercanía se define como el inverso de la «lejanía» (de la distancia) entre dos vértices. [5]

Sea  $d(v, \nu)$  la distancia geodésica entre los nodos  $v$  y  $\nu$ , la cercanía  $C_C(v)$  de un nodo  $v \in V$  se define como:

$$C_C(v) = \frac{1}{\sum_{\nu=1}^{|V|} d(v, \nu)}$$

- **Harmonic centrality:** La centralidad harmónica considera la suma del recíproco de las distancias en lugar del recíproco de la suma de las distancias (con la convención de que  $1/\infty = 0$ ). La media armónica se comporta mejor que la media aritmética, que es la utilizada por la centralidad de cercanía tradicional. [6]

Considerando  $v$  y  $\nu$  dos vectores distintos, podemos definir la centralidad harmónica como:

$$C_H(v) = \sum_{\nu=1}^{|V|} \frac{n-1}{d(v, \nu)}$$

- **Betweenness centrality:** La centralidad de intermejación es una medida que cuantifica el número de veces que un nodo se encuentra entre los caminos más cortos de otros nodos. [7]

La complejidad de esta medida para un grafo disperso y tomando el algoritmo de Johnson para calcular el

camino más corto toma:  $\Theta(V^2 \log(V) + V * E)$ .

La intermediación  $C_B(v)$  de un nodo  $i$ , donde  $b_{jk}$  es el número de caminos más cortos desde el nodo  $j$  hasta el nodo  $k$ , y  $b_{ijk}$  el número de caminos más cortos desde  $j$  hasta  $k$  que pasan a través del nodo  $i$ , se define como:

$$C_B(i) = \sum_{j \neq k \in V} \frac{b_{jik}}{b_{jk}}$$

#### II-B2. Coeficientes de clustering:

- **Coeficiente de clustering:** El coeficiente de agrupamiento de un vértice cuantifica qué tanto está de interconectado con sus vecinos. Si el vértice está agrupado como un clique (subgrafo completo), entonces alcanza su valor máximo. [8]

Siendo  $e_{jk}$  la arista que conecta a  $v_j$  con  $v_k$ , y  $k_i$  el grado del vértice, el coeficiente de agrupamiento de un grafo no dirigido se define como:

$$C_i = \frac{2 |e_{jk}|}{k_i * (k_i - 1)} : v_j, v_k \in N_i, e_{jk} \in E$$

- **Coeficiente de clustering de cuadrados:** El squares clustering coefficient cuantifica a cuántos cuadrados es posible que pertenezca el nodo. [9]

Siendo  $q_v(u, w)$  el número de vecinos comunes de  $w$  y  $v$ ,  $\theta_{uv} = 1$  si  $u$  y  $w$  están conectados y  $\theta_{uv} = 0$  en caso contrario:

$$\begin{aligned} a_v(u, w) &= (k_u - (1 + q_v(u, w) + \theta_{uv})) + \\ &+ (k_w - (1 + q_v(u, w) + \theta_{uv})) \end{aligned}$$

$$C_2 = \frac{\sum_{u=1}^{k_v} \sum_{w=u+1}^{k_v} q_v(u, w)}{\sum_{u=1}^{k_v} \sum_{w=u+1}^{k_v} [a_v(u, w) + q_v(u, w)]}$$

#### II-B3. Detección de comunidades:

Se ha utilizado la detección de comunidades como forma alternativa para entrenar a los modelos. Se han detectado las comunidades que conforman el grafo y se les ha asignado un identificador a los nodos que conforman cada comunidad.

- **Greedy Modularity:** Utiliza la función de Clauset-Newman-Moore ‘greedy modularity maximization’ [9] para encontrar las comunidades con la máxima modularidad. Este algoritmo pretende encontrar una modularidad alta para poder establecer unas buenas divisiones en comunidades. Esta es una tarea complicada, por lo que el algoritmo utiliza una optimización.
- **Label Propagation:** Encuentra las comunidades utilizando el método de ‘Label Propagation’ [10]. En

este método cada nodo comienza con una etiqueta (label) única que se va actualizando. En cada iteración, los grupos conectados de nodos llegan a “consenso” sobre qué etiqueta es la predominante, la cual será la nueva etiqueta de los nodos de esa comunidad. Al finalizar, los nodos de cada comunidad terminan con una etiqueta diferente (la misma para todos los nodos de la comunidad), pudiendo así separar las comunidades.

**II-B4. Comparación de las distintas medidas (Test-t de Student):** Para cada una de las medidas vistas en la sección anterior se realiza un *test-t de Student* para evaluar ‘a priori’ cuánto de buenas podrían llegar a ser las métricas a la hora de clasificar los nodos del grafo. Este *test t de Student* compara las medias de dos muestras y determina si son significativamente diferentes. El resultado de la evaluación aporta, entre otros valores, un p-valor en el que nos fijaremos para evaluar las métricas. Un p-valor bajo (generalmente  $< 0.05$ ) indica que hay una diferencia significativa entre las medias de los dos conjuntos.

#### II-B5. Modelos entrenados y evaluados [11]:

- **Árboles de decisión CART (Classification and Regression Trees):** Dividen el espacio de características en regiones mediante reglas de decisión binarias. Cada nodo interno representa una condición sobre una característica, y cada hoja del árbol representa un valor de salida o una clase.
- **Naive Bayes:** Es un modelo basado en el teorema de Bayes con la suposición ‘naive’ de que todas las características son independientes entre sí. Es especialmente útil para tareas de clasificación, que es el caso que nos ocupa.
- **K vecinos más cercanos (KNN):** Es un método de clasificación donde una nueva muestra se clasifica basándose en sus vecinos más cercanos en el espacio de características de la clase. La cercanía generalmente se mide mediante una distancia (como la distancia euclídea).
- **Random Forest:** Es un conjunto de árboles de decisión que se utilizan para mejorar la precisión. Cada árbol se entrena en una muestra diferente del conjunto de datos y se construye utilizando un subconjunto aleatorio de características en cada división. La predicción final del modelo se obtiene mediante el voto mayoritario (para clasificación), o la media (para regresión) de las predicciones individuales de los árboles.
- **Gradient Boosting:** Es un método que construye el modelo en etapas de forma secuencial, y cada nuevo modelo se entrena para corregir los errores cometidos por el anterior. Los modelos se combinan para mejorar la precisión y reducir el error general.

### III. METODOLOGÍA

El desarrollo ha seguido una serie de fases bastante marcadas:

#### *III-A. Selección de métricas*

- **Representación del grafo:** Para seleccionar las métricas relacionales en un primer momento se trató de representar el grafo con diferentes estructuras y tamaños. Sin embargo, debido en parte al gran tamaño del grafo la representación no arrojó ninguna métrica obvia a simple vista. Sobre la base de lo anterior, se tuvo que buscar otra forma de seleccionar las métricas. A continuación, algunas representaciones que se han obtenido:

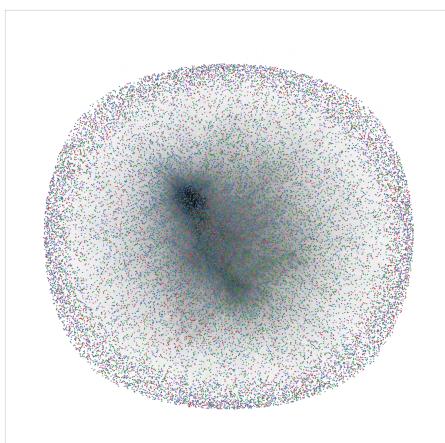


Imagen 1: Grafo completo. TVShow color rojo, Government color azul, Politician color verde y Company color morado

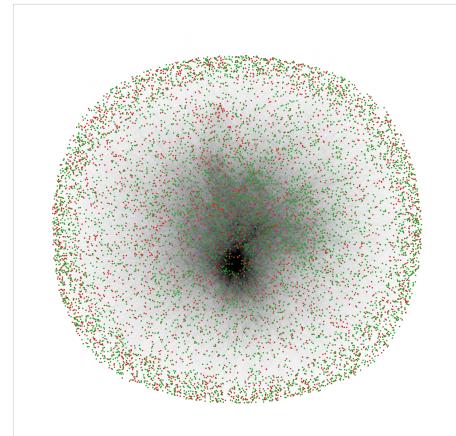


Imagen 3: Grafo solo con Politician (verde) y TVShow (rojo).

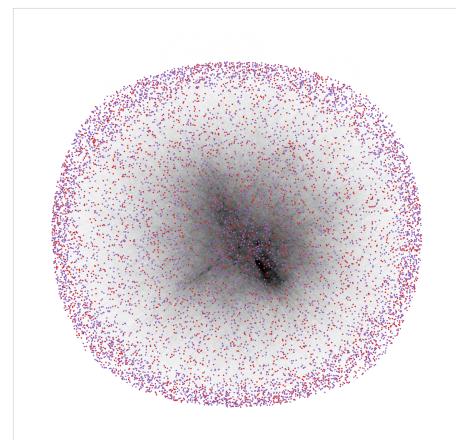


Imagen 4: Grafo solo con TVShow (rojo) y Company (morado).

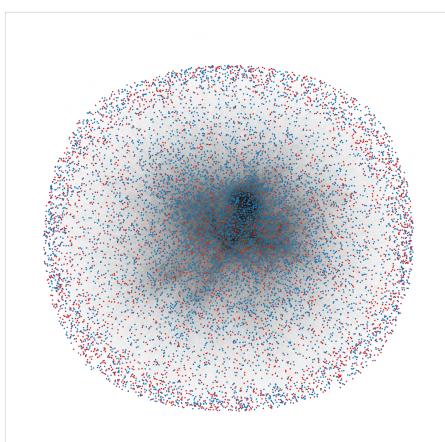


Imagen 2: Grafo solo con Government (azul) y TVShow (rojo).

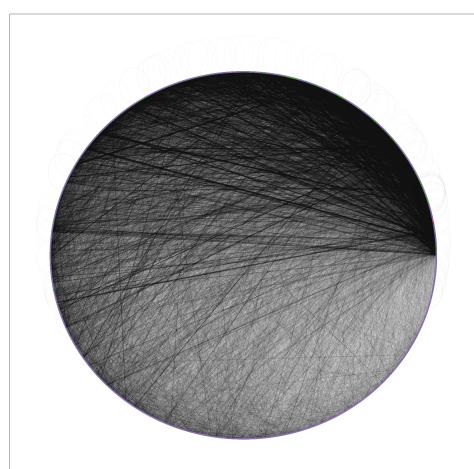


Imagen 5: Grafo representado en forma circular.

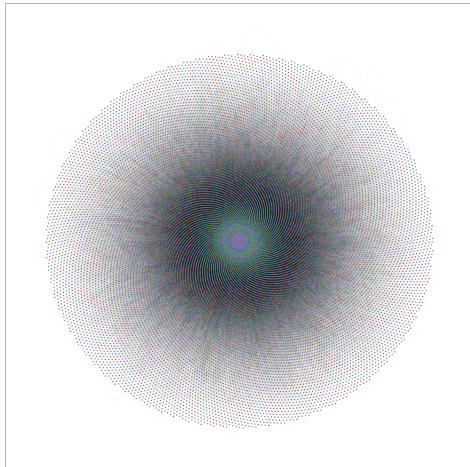


Imagen 6: Grafo representado en forma espiral.

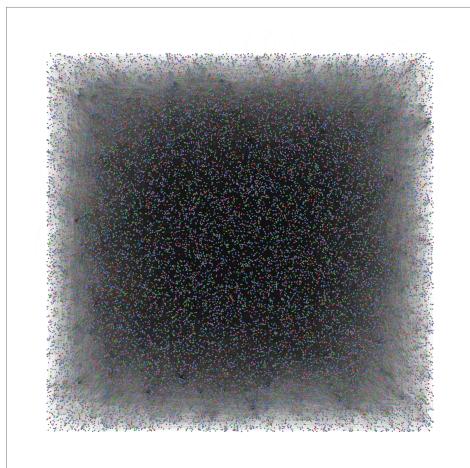


Imagen 7: Grafo representado en forma random.

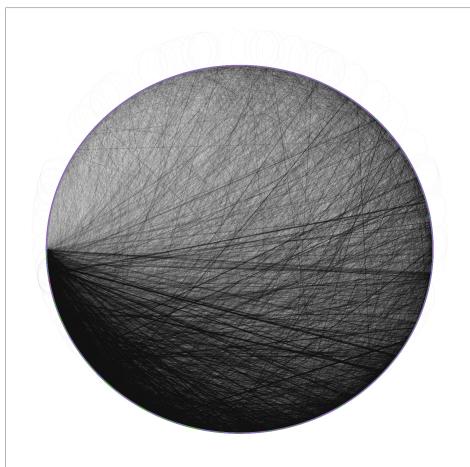


Imagen 8: Grafo representado en forma 'shell'.

las características relacionales del grafo. Se empleó un t-test para obtener las métricas que pudieran diferenciar mejor a priori los nodos por tipo. La única métrica que obtuvo unos malos resultados de forma clara fue la ‘Betweenness centrality’, por tanto podría haber sido excluida a priori a la hora de evaluar los modelos.

- **Validación del tutor de las métricas:** Una vez obtenidas las métricas, se consultó con el tutor la viabilidad de estas. Al exponerle cómo habíamos seleccionado las métricas (con el t-test) nos recomendó entrenar y evaluar los modelos también con las métricas que habían arrojado malos resultados. El t-test evalúa las métricas de manera aislada, no obstante, en los modelos se evalúan en conjunto, por lo que resulta interesante estudiarlas también sin tomar demasiado en cuenta el resultado del t-test.

### III-B. Selección de modelos

Los modelos seleccionados deberán hacer tareas de clasificación multiclas. Teniendo en cuenta esta premisa, se seleccionaron modelos ya implementados en librerías de *Python* (principalmente [scikit-learn](#)). Los modelos seleccionados fueron: **árbol de decisión CART, Naive Bayes, KNN, Random Forest y Gradient Boosting** [11].

### III-C. Entrenamiento y evaluación de los modelos

Se entrenaron y evaluaron los modelos seleccionados con las métricas seleccionadas. A continuación, se presentan varias tablas con los resultados de dicha evaluación.

- **Accuracy:** Se muestra una tabla con la exactitud (*accuracy*) de cada modelo con todas las métricas y con solo las mejores métricas. También se presenta otra tabla con los mejores hiperparámetros tras aplicar la búsqueda en rejilla para cada modelo.

MODELO	ACCURACY (todas las métricas)	ACCURACY (mejores métricas)
Arbol de decisión	0.7992	<b>0.8401</b>
Naive bayes	0.46895	<b>0.4831</b>
KNN	<b>0.5596</b>	0.5389
Random Forest	<b>0.5967</b>	0.5696
Gradient Boosting	<b>0.5910</b>	0.5509

Tabla 1: Resultados del entrenamiento de los modelos.

- **T-test de Student:** En este caso, se decidió obviar las features predeterminadas del grafo y utilizar en su lugar las distintas métricas que se pudieron obtener a partir de

MODELO	MEJORES HIPER-PARÁMETROS
Arbol de decisión	criterion: entropy, max depth: None, min samples leaf: 1, min samples split: 2
Naive bayes	alpha: 1.0
KNN	knn metric: euclidean, knn N neighbors: 7
Random Forest	criterion: gini, max depth: 12, min samples leaf: 1, min samples split: 10
Gradient Boosting	learning rate: 0.1, max depth: 5, N estimators: 150, subsample: 0.8

Tabla 2: Resultados de las búsquedas en rejilla para cada modelo.

■ *Evaluación métricas para árbol de decisión:*

Atributo	Importancia
<b>label propagation communities id</b>	<b>0.370096</b>
<b>greedy modularity communities id</b>	<b>0.250201</b>
square clustering	0.819221
closeness centrality	0.072177
clustering	0.066479
harmonic centrality	0.070663
betweenness centrality	0.051013
k path centrality	0.021066
degree centrality	0.024901

Tabla 3: Importancias de las métricas para el modelo de Arbol de Decisión.

■ *Evaluación métricas para Naive Bayes:*

Atributo	Importancia
<b>label propagation communities id</b>	<b>0.025590</b>
greedy modularity communities id	0.087450
square clustering	0.067423
<b>closeness centrality</b>	<b>0.037836</b>
clustering	0.0716551
<b>harmonic centrality</b>	<b>0.035381</b>
betweenness centrality	0.013217
k path centrality	0.059858
degree centrality	0.060080

Tabla 4: Importancias de las métricas para el modelo de Naive Bayes.

■ *Evaluación métricas para KNN:*

Atributo	Importancia
label propagation communities id	0.228082
greedy modularity communities id	0.310191
square clustering	0.155986
closeness centrality	0.126836
clustering	0.154428
harmonic centrality	0.191144
betweenness centrality	0.024278
k path centrality	0.253672
degree centrality	0.184023

Tabla 5: Importancias de las métricas para el modelo de KNN.

■ *Evaluación métricas para Random Forest:*

Atributo	Importancia
<b>label propagation communities id</b>	<b>0.217267</b>
<b>greedy modularity communities id</b>	<b>0.120824</b>
<b>square clustering</b>	<b>0.127351</b>
<b>closeness centrality</b>	<b>0.117354</b>
clustering	0.091710
<b>harmonic centrality</b>	<b>0.132949</b>
betweenness centrality	0.09019
k path centrality	0.045026
degree centrality	0.054188

Tabla 6: Importancias de las métricas para el modelo de Random Forest.

■ *Evaluación métricas para Gradient Boosting:*

Atributo	Importancia
<b>label propagation communities id</b>	<b>0.260853</b>
<b>greedy modularity communities id</b>	<b>0.152577</b>
<b>square clustering</b>	<b>0.145382</b>
closeness centrality	0.056970
clustering	0.085312
<b>harmonic centrality</b>	<b>0.207524</b>
betweenness centrality	0.090234
k path centrality	0.039429
degree centrality	0.051953

Tabla 7: Importancias de las métricas para el modelo de Gradient Boosting.

- **Accuracy sólo con las métricas relacionadas con detección de comunidades y con todas menos ellas:**

MODELO	ACCURACY (sólo comunidades)	ACCURACY (sin comunidades)
Arbol de decisión	<b>0.5897</b>	0.5111
Naive bayes	0.4439	<b>0.4591</b>
KNN	<b>0.4326</b>	0.4813
Random Forest	<b>0.4662</b>	0.5194
Gradient Boosting	<b>0.4662</b>	0.5125

Tabla 8: Resultados del entrenamiento de los modelos con y sin métricas relacionadas con detección de comunidades.

- **Accuracy con node2vec según dimensiones:** Se muestra la tabla con la exactitud de cada modelo con todas las métricas utilizando los datos generados al utilizar node2vec según las dimensiones. Por cuestiones de tiempo de ejecución no se han podido probar los modelos Gradient Boosting y Random Forest.

MODELO	32 dimensiones	64 dimensiones	128 dimensiones
Arbol de decisión	0.834668	0.776146	0.752336
Naive bayes	0.805741	0.810191	0.829996
KNN	0.932799	0.937472	0.936805

Tabla 9: Resultados del entrenamiento de los modelos con los datos generados al utilizar node2vec según las dimensiones.

- **Accuracy con node2vec según longitud del camino:** Se muestra la tabla con la exactitud de cada modelo con todas las métricas utilizando los datos generados al utilizar node2vec según la longitud del camino. Por cuestiones de tiempo de ejecución no se han podido probar los modelos Gradient Boosting y Random Forest.

MODELO	5 nodos	30 nodos	50 nodos
Arbol de decisión	0.784824	0.834668	0.8424566
Naive bayes	0.795728	0.805741	0.802181
KNN	0.929462	0.932799	0.935024

Tabla 10: Resultados del entrenamiento de los modelos con los datos generados al utilizar node2vec según la longitud del camino.

- **Accuracy con node2vec según número total de caminos:** A continuación, se muestra una tabla con

la exactitud de cada modelo con todas las métricas utilizando los datos generados al utilizar node2vec según el número total de caminos. Por cuestiones de tiempo de ejecución no se han podido probar los modelos Gradient Boosting y Random Forest.

MODELO	20 caminos	200 caminos
Arbol de decisión	0.820650	0.834668
Naive bayes	0.802626	0.805741
KNN	0.930574	0.932799

Tabla 11: Resultados del entrenamiento de los modelos con los datos generados al utilizar node2vec según el número total de caminos.

#### IV. RESULTADOS

En este apartado se explicarán los resultados conseguidos tras evaluar y entrenar los modelos. Se tomarán como referencias los resultados resumidos en las tablas de la sección III-C.

1. En las tabla 1 y 8 se observa que el modelo que presenta mejor precisión tras su entrenamiento es el modelo de **Árbol de Decisión** entrenado con comunidades (label propagation y greedy modularity). Este es **mejor** cuando se usan solo las métricas de comunidades, llegando a obtener una precisión superior a 0.8, mucho mayor que el resto de modelos que rondan el 0.5.
2. El modelo de **Naive Bayes** es el que presenta el **peor** resultado, siendo el único con una precisión menor a 0.5 tanto siendo entrenado con todas las *features*, como sólo con las más importantes.
3. El resto de modelos (**KNN, Random Forest y Gradient Boosting**) presentan una precisión ligeramente superior a 0.5 siendo entrenados con todas las *features*, aunque sus resultados empeoran al ser entrenados con cualquier otra combinación de ellas.
4. Respecto a la métrica de centralidad de intermediación (*Betweenness centrality*), cabe mencionar que fue la única que se pudo descartar a priori tras evaluar todas las métricas con el t-test de Student. Tras entrenar todos los modelos, se ha podido corroborar que la métrica no sirve para mejorar la precisión de los modelos respecto a otras como la de comunidades, clustering o harmonic centrality; aunque aún así se ha usado para intentar mejorar el resultado de los modelos que necesitaban hacer uso de todas las *features*.

5. Finalmente, respecto a los resultados obtenidos al utilizar **node2vec**, destaca que se han obtenido mejores resultados que utilizando las features relacionales construidas manualmente. Se ha conseguido una mayor precisión usando **KNN**. Por otro lado, se puede observar que la elección de parámetros no afecta demasiado a la hora de clasificar los nodos al usar node2vec.

## V. CONCLUSIONES

En este apartado se expondrán las conclusiones a las que se han llegado una vez analizados los resultados del apartado anterior:

- Observando los resultados tras entrenar los modelos con las métricas seleccionadas podemos determinar que el **modelo que ha obtenido mejores resultados es el Árbol de Decisión**. Sin embargo, en general los resultados han sido pobres, tanto con las métricas originales como con las que ofrecían mejores resultados para el modelo. Sobre la base de lo anteriormente expuesto, se podría llegar a la conclusión de que quizás no se han seleccionado las mejores métricas para entrenar a los modelos, sin embargo se han utilizado una variedad de métricas considerable. Por esta razón, los malos resultados de los modelos son causa de la técnica utilizada, es decir, de la construcción manual de features relaciones.
- En otro orden de cosas, se destaca la importancia que han tenido las métricas relacionadas con la detección de comunidades. Esto nos indica que los nodos del grafo se agrupan en comunidades, pero los resultados al entrenar los modelos solo con estas métricas nos indican que por lo general es necesario utilizar el resto de métricas para hacer predicciones. Además, debido a que afortunadamente el modelo con mejor puntuación es el Árbol de Decisiones, podemos aprender sobre cómo el modelo clasifica los nodos, y así saber acerca de la estructura interna del grafo.

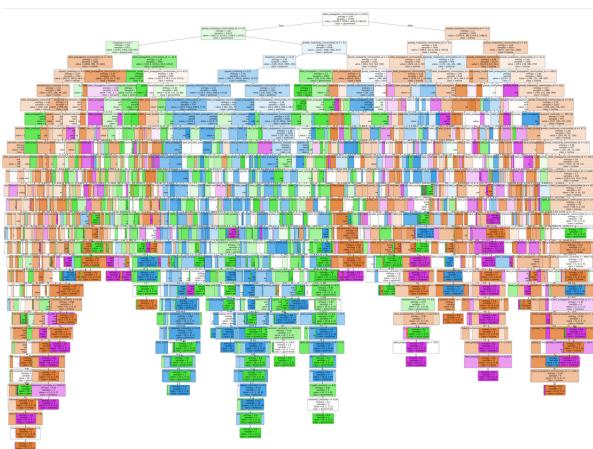


Imagen 9: Árbol completo.

Como se puede apreciar, el árbol es demasiado grande como para poder analizarlo con un solo vistazo, por lo que habría que hacer zoom en él. Podemos resaltar que las primeras decisiones evalúan a qué comunidad pertenece el nodo a clasificar.

```
label_propagation_communities_id <= 229.5
    entropy = 1.95
    samples = 17976
    value = [5196.0, 5504.0, 4614.0, 2662.0]
    class = government
```

Imagen 6: Primera decisión del árbol.

```
greedy_modularity_communities_id <= 0.5
    entropy = 1.83
    samples = 12617
    value = [2521.0, 5124.0, 3798.0, 1174.0]
    class = government
```

Imagen 7: Segunda decisión del árbol.

```
greedy_modularity_communities_id <= 3.5
    entropy = 1.7
    samples = 5359
    value = [2675.0, 380.0, 816.0, 1488.0]
    class = tvshow
```

Imagen 8: Tercera decisión del árbol.

- Cabe recalcar que uso de **node2vec ha dado muy buenos resultados**, destacando el modelo KNN. Estos buenos resultados nos indican que es posible hacer tareas de clasificación sobre los nodos del grafo a partir de la propia estructura del grafo. No obstante, aunque aporte mejores resultados y no requiera prácticamente ningún esfuerzo utilizar node2vec, los resultados obtenidos tras su uso no nos arrojan información relevante sobre el grafo más que los nodos se pueden clasificar.

Para concluir, se expondrá lo aprendido sobre el trabajo y se valorará su importancia: Este proyecto es el primero desde que comenzamos la carrera que podemos categorizar realmente como de investigación. Lo aprendido durante la realización de este proyecto nos servirá para el resto de nuestra vida tanto técnica como profesionalmente. Por un lado, hemos mejorado nuestras habilidades de análisis, resolución de problemas y hemos aprendido a usar formatos científicos de documentación. Por otro lado, hemos adquirido nuevos conocimientos técnicos acerca de temas tan importantes como los grafos, hemos ampliado nuestro conocimiento sobre el aprendizaje automático, además de aprender nuevas técnicas de análisis.

## REFERENCIAS

- [1] B. Rozemberczki, C. Allen, and R. Sarkar, “Multi-scale attributed node embedding”, [Online] ↗
- [2] Aditya Grover and Jure Leskovec, “node2vec: Scalable Feature Learning for Networks”, [Online] ↗
- [3] Tomas Mikolov, Kai Chen, Greg Corrado and Jeffrey Dean, “Efficient Estimation of Word Representations in Vector Space”, [Online] ↗
- [4] J. Sun and J. Tang, “A survey of models and algorithms for social influence analysis”, in *Social Network Data Analytics*, C. C. Aggarwal, Ed. New York: Springer, 2011, pp. 177–214. [Online] ↗
- [5] G. Sabidussi, “The centrality index of a graph”, *Psychometrika*, vol. 31, no. 4, pp. 581–603, 1966. [Online] ↗
- [6] M. A. Beauchamp, “An improved index of centrality”, *Systems Research and Behavioral Science*, vol. 10, no. 2, pp. 161–163, 1965. [Online] ↗
- [7] U. Brandes, “A faster algorithm for betweenness centrality”, *Journal of Mathematical Sociology*, vol. 25, pp. 163–177, 2001. [Online] ↗
- [8] D. J. Watts and S. Strogatz, “Collective dynamics of small-world networks”, [Online] ↗
- [9] Clauset, A., Newman, M. E. and Moore, C. “Finding community structure in very large networks.”, [Online] ↗
- [10] Gennaro Cordasco, Luisa Gargano, “Community Detection via Semi-Synchronous Label Propagation Algorithms”, [Online] ↗
- [11] “Supervised learning”, scikit-learn: Machine Learning in Python. [Online] ↗