



UNIVERSIDAD DE BUENOS AIRES
FACULTAD DE CIENCIAS EXACTAS Y NATURALES

Redes neuronales en grafos para predicción de interacciones droga-target

Tesis de Licenciatura en Ciencias de Datos

Nicolas Yanovsky

Director: Ariel Chernomoretz

Buenos Aires, 2024

RESUMEN

La correcta predicción de interacciones entre drogas y sus respectivos targets moleculares (DTIs por sus siglas en ingles) juega un rol muy importante en el reposicionamiento, diseño y descubrimiento de drogas. Sin embargo, debido al gran tamaño del espacio quimio-genómico donde se encuentran estas interacciones y la complejidad de las mismas, la identificación experimental o mediante el uso de simulaciones de DTIs es una tarea de alto costo y consumo de tiempo. Debido a esto, y a la naturaleza no-euclidiana de los espacios donde se representan estas interacciones, se ha visto un interés creciente en los últimos años en la aplicación de redes neuronales en grafos (GNNs por sus siglas en ingles) para la predicción de DTIs.

En este trabajo nos enfocamos en la aplicación de GNNs sobre una red heterogénea que consiste en dos capas: una para drogas y otra para targets. Las conexiones intra-capas denotan algún tipo de similaridad estructural, mientras que las conexiones inter-capas provienen de DTIs documentadas.

Analizamos en profundidad los resultados de la selección del modelo, observando diferencias sustanciales en la performance dados por pequeños cambios en la arquitectura y sus hiperparámetros, confirmando la complejidad del espacio de diseño de estas redes mostrada en trabajos previos. Asimismo, nos enfocamos en el enriquecimiento de las features iniciales tanto para los targets con información externa a la otorgada por la topología de la red, e identificamos su impacto en la performance del modelo. Por ultimo, analizamos sesgos en las predicciones de los modelos respecto a la conectividad de la red.

Palabras claves: Predicción de DTIs, redes heterogéneas, Graph Neural Networks, predicción de aristas, similaridad estructural.

ABSTRACT

The correct prediction of drug-target interactions (DTIs) plays a crucial role in drug repositioning, design, and discovery. However, due to the large size of the chemo-genomic space where these interactions occur and their complexity, the experimental identification or simulation of DTIs is a costly and time-consuming task. Because of this, and due to the non-Euclidean nature of the spaces where these interactions are represented, there has been a growing interest in recent years in the application of Graph Neural Networks (GNNs) for DTI prediction.

In this work, we focus on the application of GNNs on a heterogeneous network consisting of two layers: one for drugs and another for targets. Intra-layer connections denote some form of structural similarity, while inter-layer connections stem from documented DTIs. We thoroughly analyze the results of model selection, observing substantial differences in performance caused by small changes in the architecture and its hyperparameters, confirming the complexity of the design space of these networks as shown in previous studies. Furthermore, we focus on enriching the initial features for targets with external information beyond what is provided by the network’s topology and identify their impact on the model’s performance. Finally, we identify connectivity biases in the model’s predictions.

Keywords: DTI prediction, heterogeneous networks, Graph Neural Networks, link prediction, structural similarity.

Índice general

1.. Motivación	1
2.. Introducción	3
2.1. Interacciones Droga-Target	3
2.2. Grafos y Redes Complejas	4
2.3. Aprendizaje automático en grafos	8
2.3.1. Métodos clásicos	8
2.3.2. Embeddings de nodos	10
2.3.3. Graph Neural Networks	12
3.. Métodos	19
3.1. Dataset	19
3.1.1. construcción de la red	19
3.1.1.1. Integración de datos crudos	19
3.1.1.2. Capa de drogas	20
3.1.1.3. Capa de proteínas	22
3.1.2. Análisis de la red	25
3.2. Implementación	29
3.2.1. Arquitectura	30
3.2.2. Enriquecimiento de features	32
3.2.3. Especificaciones de entrenamiento	35
3.2.4. Selección y validación	36
4.. Resultados y análisis	39
4.1. Mejores modelos	39
4.2. Impacto de los hiperparámetros	40
4.3. Efecto de la incorporación de features externas	43
4.4. Sesgos por conectividad	48
5.. Conclusiones	53
5.1. Mejoras y Trabajos Futuros	54
5.2. Disponibilidad de los datos	55
6.. apéndice	57

1. MOTIVACIÓN

La predicción precisa de interacciones droga-target (DTIs, por sus siglas en inglés) es un desafío crucial en la investigación biomédica y la industria farmacéutica, con aplicaciones significativas en el diseño de nuevos fármacos, reposicionamiento de drogas y descubrimiento de terapias para enfermedades complejas. Estas interacciones son esenciales para comprender el funcionamiento molecular de las drogas y sus efectos en el organismo, permitiendo avances en tratamientos más eficaces y seguros. Sin embargo, el espacio químico-genómico donde se encuentran estas interacciones es vasto y de gran complejidad, haciendo que la identificación experimental de DTIs sea una tarea de alto costo en términos de tiempo y recursos. Debido a esta dificultad, se ha explorado intensamente el desarrollo de métodos computacionales capaces de abordar de manera eficiente esta tarea, permitiendo reducir significativamente el esfuerzo experimental.

Uno de estos abordajes es mediante la teoría de redes ([1], [2]), donde se contemplan grandes cantidades de interacciones entre drogas y targets para hacer uso de esa información global a la hora de realizar las predicciones, en contraposición con enfoques puntuales como las simulaciones de interacción. Este trabajo se enmarca dentro de dicho abordaje, específicamente dentro del aprendizaje automático sobre redes ([3], [4]).

En particular, desarrollamos una red heterogénea que integra espacios químicos y genómicos (donde la información presente en estos espacios es del tipo estructural) junto a DTIs relevadas. La hipótesis de la que partimos es que las interacciones entre drogas y targets son relaciones de tipo estructural entre ambas moléculas, por lo que embeber estas interacciones en un espacio donde distancias entre los objetos que se encuentran allí codifiquen de alguna forma similitud estructural entre estos será de utilidad a la hora de realizar predicciones. Para realizar las predicciones, nos valemos de una novedosa familia de arquitecturas de aprendizaje profundo denominadas redes neuronales en grafos (GNNs por sus siglas en inglés), cuya principal ventaja frente a otros modelos es que permiten tomar como input explícitamente la estructura de las redes, permitiendo explotar patrones en la conectividad de estas a la hora de realizar las predicciones. Esto, sumado a que permiten modelar relaciones heterogéneas entre los objetos de la red e incorporar información externa a la conectividad de estos (como lo puede ser por ejemplo el perfil de expresión de un gen), hacen que estas arquitecturas sean sumamente adecuadas para atacar el problema en cuestión, permitiendo explorar a gran escala el espacio de interacciones entre drogas y targets.

La tesis esta estructurada de la siguiente manera:

En el capítulo 2 se ofrece una introducción teórica necesaria para comprender la metodología utilizada, abarcando los conceptos esenciales de la teoría de grafos y las técnicas de aprendizaje automático sobre redes.

Esta metodología se desarrolla en el capítulo 3, donde se describe el proceso de construcción de una red heterogénea compuesta por drogas y targets y la implementación de una GNN sobre esta para realizar las predicciones de DTIs.

Luego, en el capítulo 4 se presentan los resultados obtenidos y se explora la efectividad de la metodología aplicada junto con diversos análisis de los resultados.

Finalmente, en el capítulo 5 se presentan las conclusiones del trabajo, discutiendo los

aprendizajes alcanzados, las limitaciones enfrentadas, y posibles direcciones a seguir en futuros trabajos en esta línea de investigación.

2. INTRODUCCIÓN

2.1. Interacciones Droga-Target

Al hablar de interacciones entre drogas y sus targets moleculares nos vamos a estar refiriendo a lo largo del trabajo específicamente a interacciones físicas entre compuestos químicos, usualmente pequeñas moléculas, denominadas drogas, y proteínas.

Desde el punto de vista químico, estas interacciones resultan cuando una droga se une a una proteína en un sitio específico de esta, denominado *sitio de unión*, mediante enlaces químicos, como se ilustra en la figura 2.1. La naturaleza de estos enlaces (por ejemplo, si se dan de forma covalente, mediante puentes de hidrógeno, etc), junto a otros factores, determinan finalmente la estabilidad de la interacción, denominada *afinidad de unión*.

La unión de la droga a la proteína es lo que genera de alguna forma el efecto farmacéutico de la primera. Esto se puede dar de varias formas, ya sea desde la inducción de un cambio en la estructura de la proteína, funcionando como competidora hacia otras moléculas que se unen al mismo sitio, etc. El resultado es la modificación, en algún aspecto, de la función de la proteína, generando una secuencia de eventos que finalmente resultan en el efecto de la droga.

A modo de ejemplo, el Finasteride, droga utilizada en el tratamiento de la alopecia (calvicie) androgenica, es un *inhibidor* de la enzima 5- α -reductasa, responsable de catalizar la transformación de testosterona en 5- α -dihidrotestosterona, hormona que causa el achicamiento de los folículos pilosos resultando en la caída de cabello. La *inhibición* de una enzima refiere a la disminución de la actividad de esta, resultando en una menor cantidad de su producto (5- α -dihidrotestosterona en este caso), que en este caso se da al unirse el Finasteride al sitio activo de la enzima, impidiendo la unión de la testosterona.

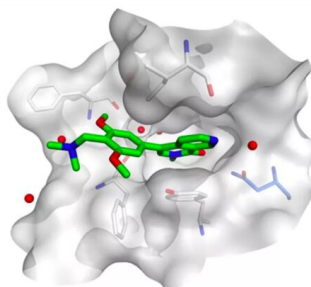


Fig. 2.1: Esquema de interacción entre droga (verde) y una proteína

2.2. Grafos y Redes Complejas

Si bien la descripción microscópica de las DTIs sirve para contextualizar el objeto central de este trabajo, el enfoque que tomamos a lo largo del mismo ignora en buena parte esta escala de resolución, dejando de tener en cuenta interacciones individuales para poder hacerlo a gran escala. Una forma natural de representar un conjunto de DTIs es mediante un **grafo** o una **red**, términos que vamos a intercambiar libremente en el desarrollo del trabajo.

Definición. Un **grafo** o **red** $G = (V, E)$ es un conjunto de **vértices** o **nodos** V , junto con un conjunto de **aristas** o **enlaces** $E \subseteq V \times V$.

Las redes son una representación natural de sistemas que están compuestos por entidades para las cuales está definida algún tipo de relación entre estas, en donde las entidades del sistema vendrían a ser los nodos y tenemos que si G es el grafo que representa al sistema y $v_i, v_j \in V(G)$ dos entidades de este, entonces $(v_i, v_j) \in E(G)$ si están relacionados. En el caso en el que $(v_i, v_j) \in V(G) \iff (v_j, v_i) \in E(G)$, decimos que G es *no dirigido*. La forma de representar mediante un grafo a un sistema es una elección, en el sentido de que el grafo será dirigido o no dependiendo el tipo de interacción que se quiera representar: interacciones asimétricas (por ejemplo, “es amigo de”) son naturalmente representadas mediante grafos dirigidos, mientras que lo contrario pasa para grafos no dirigidos (por ejemplo, la interacción física entre dos moléculas).

En este trabajo, vamos a contar con dos tipos de nodos, uno para las drogas y otro para los targets, mientras que las DTIs van a ser aristas con un extremo en cada tipo de nodo. Además, vamos a contar con aristas entre drogas y aristas entre targets, denotando algún tipo de similitud estructural. Un esquema de dicha red se puede ver en la figura 2.2

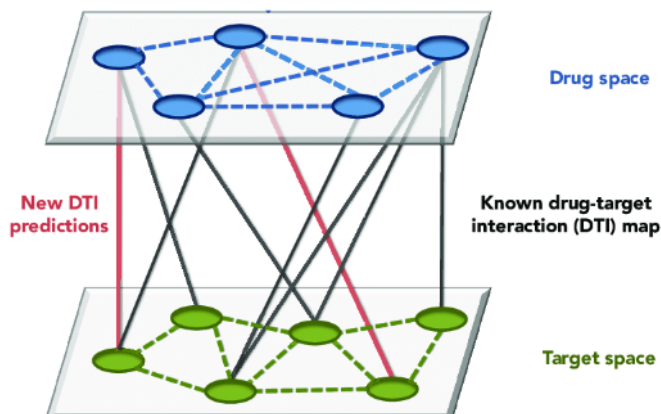


Fig. 2.2: Esquema de la red

La premisa central del trabajo y la razón por la que nos enfocamos en el estudio a una escala macroscópica de estas interacciones es que en la topología de la red, aprovechando la gran cantidad de datos a disposición, se encuentra codificada la información necesaria para poder realizar predicciones de DTIs.

El estudio de redes derivadas a partir de sistemas complejos, denominadas *redes complejas*, donde un sistema que consiste de múltiples entidades que exhiben interacciones diversas y no locales entre sí es representado por un grafo, emergió como un campo científico propio a comienzos de los años 2000, en parte disparado por el acceso a grandes cantidades de datos.

A continuación introducimos algunas propiedades usadas para describir estas redes que serán útiles para lo que sigue:

- Dado un grafo $G = (V, E)$ y un nodo $u \in V$, notamos su vecindario, es decir el conjunto de nodos relacionados con u , como $\mathcal{N}(u) = \{v \in V : (u, v) \in E\}$. El grado de u , es decir el tamaño de su vecindario, se nota como $d(u)$ o k_u y es $|\mathcal{N}(u)|$.
- El *coeficiente de clustering* de un nodo u es la proporción de sus vecinos que son vecinos entre sí y se nota C_u , es decir:

$$C_u = \frac{|\{v, w \in \mathcal{N}(u) : (v, w) \in E\}|}{\binom{k_u}{2}}$$

Notemos que el coeficiente, que cuenta la cantidad de triángulos sobre todos los triángulos posibles (en el vecindario de u , con vértice en u) es una medida local, en el sentido de que habla sobre que tan conectado está el vecindario de un dado nodo. Una versión global del coeficiente cuenta la cantidad de triplas de nodos cerrados (en donde los tres son vecinos entre sí) sobre la cantidad de triplas de nodos conectados (donde al menos dos de tres son vecinos).

- Un *camino* en G es una sucesión de nodos v_1, \dots, v_k tal que $(v_i, v_{i+1}) \in E$. Una *componente conexa* de G es un subconjunto maximal de los nodos de G tal que para cualquier par de nodos dentro de la componente existe un camino que los conecta.
- La distancia entre dos nodos u y v se define como la longitud mínima entre todos los caminos que comienzan en u y finalizan en v , es decir,
 $d(u, v) = \min\{|\mathcal{C}| : \mathcal{C} \text{ es camino entre } u \text{ y } v\}$

En su trabajo pionero [5], Watts & Strogatz muestran que muchos tipos de redes del mundo real exhiben propiedades similares, donde la mayoría de los nodos no son vecinos entre sí, pero la distancia entre cualquier par de ellos es corta, lo que ellos llaman propiedad de *mundo pequeño* (en analogía al fenómeno de mundo pequeño, popularmente conocido como seis grados de separación). Esto se ejemplifica en la figura 2.3.

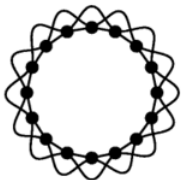
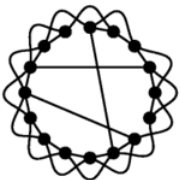

			
Network	Lattice, Ordered	Small World	Random, Disordered
Clustering Coefficient	High	High	Low
Mean Path Length	Long	Short	Short

Fig. 2.3: Diagramas de redes; ordenadas (grilla), small-world, y aleatoria, junto con algunas características. La red de tipo small-world se ubica entre el orden y el desorden. Imagen extraída de [5]

En el mismo período de tiempo, Barabasi & Albert [6] muestran que buena parte de estas redes exhiben una distribución de grado *libre de escala*. Esto es, si k es el grado de un nodo, $P(k) \sim k^{-\gamma}$, y proponen un modelo generativo denominado *preferential attachment* que recupera esta distribución, donde un nuevo nodo se une preferencialmente a nodos de alto grado, resultando en pocos nodos altamente conectados, y muchos nodos de bajo grado, como se ve en la figura 2.4. La razón por la cual estas redes se dicen *libre de escala* es que para muchas de estas redes, el γ que describe su distribución (por lo general $2 < \gamma < 4$) es tal que los momentos mayores o iguales a 2 no están bien definidos, por lo que no hay una descripción promedio de la conectividad de sus nodos.

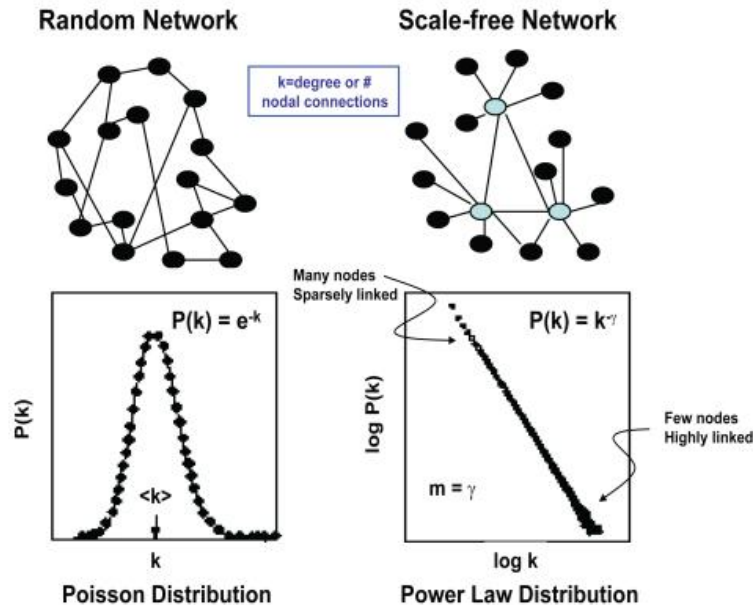


Fig. 2.4: Comparación de la distribución de grado en redes libre de escala y redes aleatorias.

Por último, otra característica importante de este tipo de redes es su estructura *modular* o de *comunidades*. En su artículo del 2002, Girvan & Newman [7] muestran que este tipo de redes exhiben la propiedad de que sus nodos se agrupan en *comunidades*, es decir, grupos de nodos donde la conectividad entre estos es considerablemente mayor que con nodos fuera del grupo, y proponen un algoritmo para detectarlos. Un ejemplo de una red y sus comunidades que estudian se ve en la figura 2.5. En el contexto de muchas redes biológicas, la estructura modular de estas suele resonar con propiedades relevantes del sistema (i.e, las comunidades se organizan respecto a propiedades externas a la conectividad de la red), por lo que su detección revela tanto propiedades de nodos previamente desconocidas como formas en la que el sistema se encuentra organizado. Por ejemplo, Spirin & Mirny [8] analizan una red de interacción de proteínas y muestran que las comunidades en las que se organiza esta revelan tanto estructuras físicas (complejos, factores de transcripción, maquinaria de splicing), como unidades funcionales (cascadas de señales, proteínas involucradas en el ciclo celular, etc).

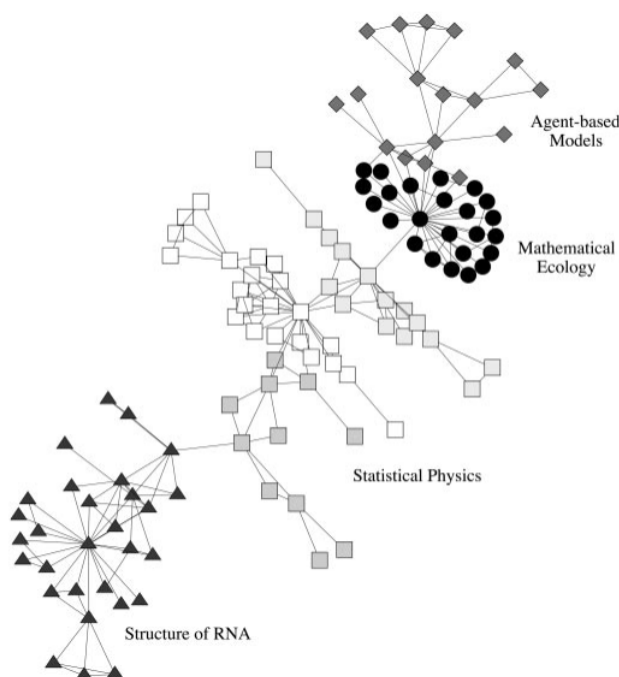


Fig. 2.5: Organización de las comunidades detectadas (Newman) en la red de colaboración del Santa Fe Institute. Se muestra claramente un agrupamiento por áreas de investigación.

La observación de que las propiedades de estas redes diferían de las que uno esperaría en un grafo aleatorio atrajo muchos investigadores al estudio de estas redes, en particular provenientes de la mecánica estadística, impulsando lo que hoy se conoce como *Network Science*.

La validez de tratar el estudio de este tipo de redes como un área de la ciencia por sí misma radica en que el estudio microscópico de las entidades que componen la red (es decir, a nivel individual) no es suficiente para poder describir el sistema completo, ya que las propiedades de este emergen de la complejidad de las interacciones que lo componen [9].

2.3. Aprendizaje automático en grafos

Dada la naturaleza compleja de las interacciones que componen los grafos, el aprendizaje automático (ML, por su traducción al inglés) surge como un enfoque natural hacia tareas donde se requiera predecir o razonar sobre datos donde la estructura de estos este representada mediante un grafo.

Por ejemplo, se podría querer, dada una colección de moléculas (representadas a través de su grafo molecular), predecir propiedades a cerca de estas, como su nivel de solubilidad, o su toxicidad, tarea que se enmarca dentro de la clasificación o regresión de grafos.

Por otro lado, uno podría estar interesado en, dado un conjunto de usuarios en una red social, predecir cuales de estos son *bots*, o dada una red de interacción de proteínas, predecir la función de algunas de estas (dentro de un conjunto acotado de funciones). Estas tareas se enmarcan dentro de lo que se conoce como clasificación de nodos.

Por último podríamos querer predecir relaciones de amistad en una red social, interacciones en una red de proteínas, o recomendaciones de películas en una red bipartita del tipo usuario-película. Esta tarea se conoce como *predicción de aristas* y es sobre la que se va a desarrollar este trabajo.

Notemos que, en contraparte con el aprendizaje automático tradicional, las tareas mencionadas difieren sustancialmente en dos aspectos: el primero, la naturaleza no-euclidiana de los datos. Los grafos no tienen una representación natural en \mathbb{R}^n , como las entradas de una tabla (uno podría pensar en las filas de la matriz de adyacencia A como features, pero cualquier permutación de las columnas de esta representa el mismo grafo.). El segundo aspecto radica en que la mayoría de los métodos tradicionales de ML suponen que las muestras con las que se trabaja son *i.i.d.* La tarea de regresión/clasificación de grafos es quizás la más similar al aprendizaje supervisado tradicional, donde cada grafo es una muestra *i.i.d* asociada a una clase o un número real. En cambio, en el caso de la clasificación de nodos o predicción de aristas no tendría sentido pensar que nodos relacionados (o cercanos) son muestras independientes de una distribución objetivo que queremos modelar. Esto imparte una dificultad inicial a la hora de realizar una distinción entre muestras de entrenamiento/validación/testeo. más tarde volveremos sobre estas problemáticas y sus soluciones típicas en el área del ML en grafos.

2.3.1. Métodos clásicos

Previo al auge de las técnicas de aprendizaje profundo, las tareas descritas anteriormente se atacaban mediante herramientas de ML tradicional. Si bien como mencionamos los grafos o los nodos que lo componen no gozan de una representación natural en \mathbb{R}^n , lo usual es extraer features de estos, ya sea mediante heurísticas o conocimiento de dominio, y usar las features como input para algoritmos tradicionales (por ejemplo, un clasificador de Random Forest). Algunas de estas heurísticas son estadísticos simples de los nodos, como su grado o su coeficiente de clustering. Otro tipo de medidas que relevan información algo más global sobre el rol de los nodos en el grafo son las denominadas medidas de *centralidad*. Al igual que las anteriores, estas son medidas que hablan sobre la conectividad de los nodos en el grafo, pero teniendo en cuenta información no solo sobre su vecindario inmediato. Por ejemplo, la centralidad de *intermediatez* caracteriza a los nodos según su frecuencia de aparición en los caminos mínimos entre todos los pares de nodos.

Las features que mencionamos anteriormente sirven como punto de partida para mu-

chos algoritmos clásicos cuando la tarea a resolver suele ser de clasificación de nodos. En la tarea de predicción de aristas no es obvio como hacer esto, pero en espíritu es similar a lo mencionado para nodos individuales. En este caso, extraemos a partir del grafo escalares para cada par de nodos, resultando en una matriz de similitud \mathbf{S} , para la cual esperamos que $\mathbf{S}_{ij} \sim P((i, j) \in E)$. Al igual que en el caso donde extraíamos features de forma individual para los nodos, en este caso también hay un enorme bagaje de estadísticos que se pueden computar para cada par de nodos que reflejen de una manera u otra la similitud entre estos. En su versión más simple y local, podríamos simplemente computar la intersección de los vecindarios, i.e, $\mathbf{S}_{ij} = |\mathcal{N}(i) \cap \mathcal{N}(j)|$, o su versión normalizada denominada índice de Jaccard, que se calcula como

$$\mathbf{S}_{ij}^{\text{Jaccard}} = \frac{|\mathcal{N}(i) \cap \mathcal{N}(j)|}{|\mathcal{N}(i) \cup \mathcal{N}(j)|}$$

De la misma forma que antes, también existen medidas globales que contemplan información más allá de los vecindarios inmediatos, como lo es el índice de Katz que puntúa pares de nodos de manera proporcional a la cantidad de caminos de cualquier longitud entre estos.

2.3.2. Embeddings de nodos

Si bien el enfoque clásico mencionado en la sección anterior es útil, la extracción de features no solo depende de heurísticas cuidadosamente diseñadas para determinadas tareas (razón por la abrumadora cantidad de estas), si no que estas son inflexibles en un proceso de aprendizaje. En esta sección introducimos la idea general para lo que resta de este capítulo, en donde buscamos *aprender*, en vez de diseñar, representaciones para los nodos que codifiquen información estructural del grafo y sus vecindarios locales. Dichas representaciones se conocen como *embeddings*, vectores de baja dimensionalidad en un espacio latente en el que relaciones geométricas en dicho espacio trasladen a relaciones en el grafo original.

Encoder-Decoder

El problema de conseguir dichas representaciones se puede formular de manera abstracta (y es útil hacerlo así) dentro del framework conocido como *encoder-decoder*, donde un modelo denominado *encoder* mapea cada nodo a su representación vectorial y otro modelo denominado *decoder* reconstruye información acerca del vecindario de este a partir de los vectores aprendidos, como se ilustra en la figura 2.6.

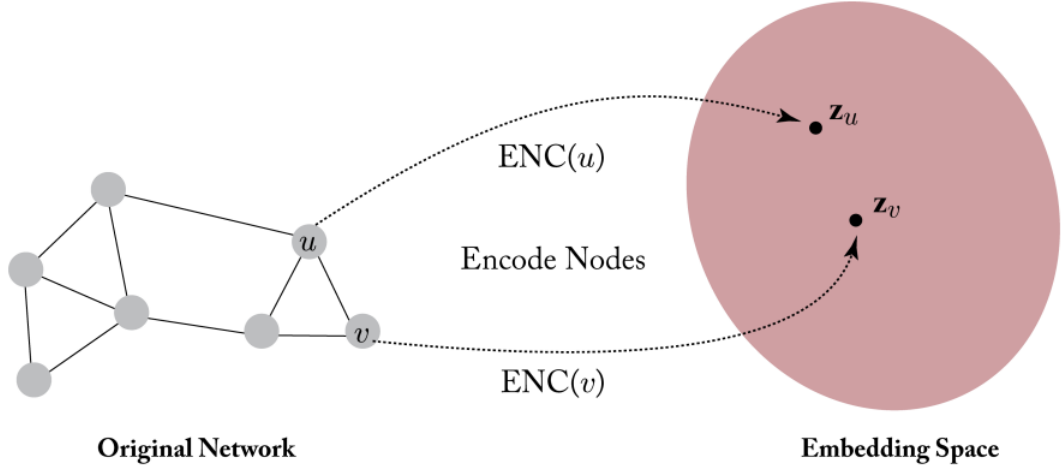


Fig. 2.6: Ilustración del esquema encoder-decoder.

Formalmente, el encoder es una función $enc : V \rightarrow \mathbb{R}^d$ que mapea nodos a sus representaciones de baja dimensionalidad, es decir $enc(v) = z_v$, con $z_v \in \mathbb{R}^d$. Por otro lado, el decoder, en su versión más tradicional, es una función $dec : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}^+$ que de alguna forma reconstruye información acerca de los vecindarios de pares de nodos. El objetivo de este enfoque es optimizar ambas funciones (usualmente en conjunto, y no necesariamente paramétricas) para que la reconstrucción refleje algún tipo de similitud entre los nodos, es decir:

$$dec(enc(u), enc(v)) = dec(z_u, z_v) \simeq \mathbf{S}_{uv}$$

Para lograr esto, se suele minimizar una pérdida de reconstrucción \mathcal{L} sobre un set de

entrenamiento de pares de nodos \mathcal{D} , de la forma

$$\mathcal{L} = \sum_{(u,v) \in \mathcal{D}} \ell(\text{dec}(z_u, z_v), \mathbf{S}_{uv})$$

donde ℓ es una función de pérdida que mide la discrepancia entre la reconstrucción decodificada y el valor real dado por la medida de similitud.

Un tipo particular de métodos para generar *embeddings* que van a ser de nuestro interés son los basados en paseos aleatorios, en los que los embeddings son optimizados de forma tal que dos nodos tengan embeddings similares si tienden a co-ocurrir en paseos aleatorios cortos en el grafo.

Uno de estos métodos es **node2vec** [10]. En este trabajo, se propone optimizar el encoder al maximizar, para cada nodo $u \in V$, la log-probabilidad de observar el vecindario de u $\mathcal{N}_S(u)$ condicionado a su representación $z_u = \text{enc}(u)$, donde S es alguna estrategia de muestreo, es decir, buscamos

$$\min_{\text{enc}} - \sum_{u \in V} \log P(\mathcal{N}_S(u) | z_u) \quad (2.1)$$

Donde $P(\mathcal{N}_S(u) | z_u)$ actúa como decoder en el modelo. Para poder hacer factible el problema de optimización se asume que las probabilidades de observar cualquier nodo $v \in \mathcal{N}_S(u)$ condicionado a z_u son independientes, por lo que

$$P(\mathcal{N}_S(u) | z_u) = \prod_{v \in \mathcal{N}_S(u)} P(v | z_u).$$

Teniendo en cuenta esto, y asumiendo el efecto simétrico que tienen los vecinos de un nodo sobre este en el espacio latente \mathbf{Z} , se modelan las probabilidades como una softmax sobre el grafo

$$P(v | z_u) = \frac{\exp(z_u \cdot z_v)}{\sum_{w \in V} \exp(z_w \cdot z_u)}$$

De esta forma, (2.1) se puede expresar de una manera más tratable y se optimizan las representaciones z_u por descenso de gradiente estocástico.

Como mencionamos, la estrategia S esta basada en paseos aleatorios, con la flexibilidad de que el algoritmo utiliza ciertos hiperparámetros para controlar las caminatas de forma tal que interpolen suavemente a caminatas que se encuentren entre un esquema de BFS y uno de DFS, lo que los autores mencionan y muestran que sirve para contemplar distintos tipos de similitudes entre los nodos, en este caso la homofilia y la equivalencia estructural ¹

Notemos que bajo el formalismo introducido de *encoder-decoder*, el encoder en este caso es simplemente acceder a una fila de una matriz de embeddings, es decir, $\text{enc}(u) = \mathbf{Z}[u]$. De esta forma, los “parámetros” del modelo son los embeddings.

¹ La homofilia refiere a la similitud de nodos en base a cuán conectados están, mientras que la equivalencia estructural refiere a similitudes en base a sus roles estructurales en el grafo.

2.3.3. Graph Neural Networks

El tipo de encoder mencionado en la sección anterior se enmarca dentro de los que se denominan *shallow encoders*. Un problema de que los parámetros del modelo sean los embeddings en sí es que los nodos no comparten parámetros, algo que es ineficiente tanto desde el punto de vista computacional como desde el punto de vista estadístico, ya que se ha visto que compartir parámetros actúa como forma de regularización y mejora la eficiencia del aprendizaje.

Otro problema es que no permiten incorporar información externa de los nodos al modelo como features, trabajando solo con la topología del grafo, algo que podría ser de ayuda en grafos que tienen información rica de sus nodos.

Por último, y quizás el más importante de los problemas que traen este tipo de encoders, es que estos métodos solo pueden generar embeddings para nodos presentes en la etapa de entrenamiento, algo sumamente restrictivo para tareas que involucran generalizar a nuevos nodos.

Teniendo en cuenta lo mencionado, introducimos en esta sección las *Graph Neural Networks* (GNNs), una familia de arquitecturas de aprendizaje profundo específicamente diseñadas para tratar sobre grafos como input, y que permiten aprovechar tanto la topología de la red, convirtiéndola en un sesgo inductivo de los modelos, como incorporar información externa de los nodos.

La necesidad de contar con un modelo que tenga en cuenta la estructura de la red a la hora de generar embeddings para los nodos se debe a que los modelos tradicionales de aprendizaje profundo no aplican, debido a que por ejemplo están diseñados para tratar sobre estructuras de grilla (CNNs), o secuencias (RNNs).

Invarianza y equivarianza por permutaciones

A priori, se podría definir un modelo para grafos que simplemente use la matriz de adyacencia achatada como input a un perceptrón multicapa (MLP)

$$\mathbf{Z}_G = \text{MLP}(A[1] \oplus \dots \oplus A[n])$$

El problema de este enfoque es que el modelo depende totalmente del orden arbitrario de los nodos en A . Idealmente, uno querría que cualquier modelo que tome como input un grafo sea independiente de esto, es decir, que o bien sea invariante por permutaciones (el output no cambia cuando se permuta el input), o equivariante (el output se permuta de la misma forma que el input cuando este es permutado). Por ejemplo, los *shallow encoders* previamente mencionados son equivariantes (si pensamos al output del modelo como la matriz de embeddings \mathbf{Z} .)

La arquitectura básica de una GNN

El modelo básico de una GNN se puede motivar de distintas maneras, dado que por ejemplo puede ser derivada como una generalización de la convolución a grafos [11], o como una versión diferenciable de un kernel usado para un test de isomorfismo [12]. Dado un grafo $G = (V, E)$ junto a features para los nodos $\mathbf{X} \in \mathbb{R}^{|V| \times d}$, estos modelos consiguen embeddings z_u para cada $u \in V$ partiendo de su feature inicial \mathbf{X}_u y actualizándolo iterativamente a partir de su embedding y los embeddings de sus vecinos, en la iteración anterior.

De manera general, el esquema de actualización se puede expresar de la siguiente forma:

$$h_u^{(k+1)} = \text{update}^{(k)} \left(h_u^{(k)}, \text{aggr}^{(k)} \{ h_v^{(k)}, v \in \mathcal{N}(u) \} \right)$$

Es decir, para cada nodo u , en cada iteración tomamos los embeddings de sus vecinos en la iteración anterior, los agregamos en un mensaje conjunto, y usamos ese mensaje junto al embedding propio (también de la iteración anterior) para pasárselo a la función que lo actualiza, donde *aggr* y *update* son funciones arbitrariamente diferenciables (i.e, redes neuronales) específicas de la iteración ². En la figura 2.7 se observa una ilustración del grafo computacional del modelo para un dado nodo.

Notemos que como *aggr* es una función definida sobre conjuntos, el esquema de actualización, que se puede pensar como una función $h^{(k)} : V \rightarrow \mathbb{R}^{d_k}$, resulta invariante ante permutaciones.

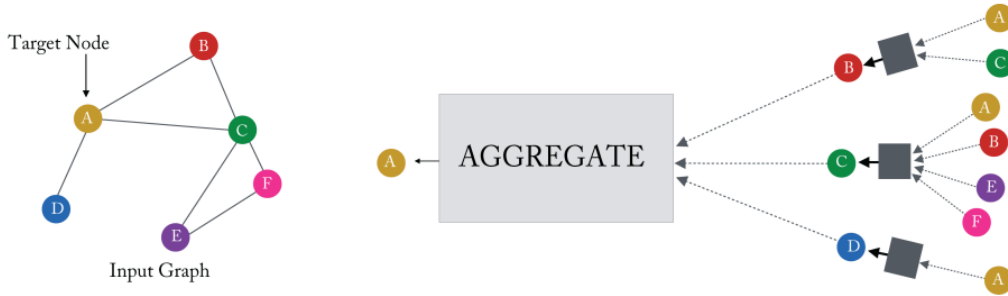


Fig. 2.7: Ilustración del esquema de agregación para el nodo A en dos iteraciones. Notemos que el grafo computacional en cada nodo define un árbol. Imagen extraída de [13]

La intuición básica del funcionamiento de este modelo es relativamente simple: en cada iteración, cada nodo utiliza información de su vecindario para actualizar la suya, por lo que a medida que las iteraciones crecen, un dado nodo contendrá información de sectores del grafo más alejados. Es decir, para la iteración $k = 1$, un dado nodo solo usará información de su vecindario inmediato, mientras que para $k = 2$ habrá usado información también de los vecinos de sus vecinos, etc. Notemos entonces que a medida que las iteraciones se acercan al diámetro del grafo, todos los nodos terminarán usando la misma información para actualizar la suya, lo que supone un problema. Debido a esto se suele trabajar con una cantidad de iteraciones relativamente chica respecto al tamaño de la red.

² Las distintas iteraciones son conocidas como las “capas” del modelo.

Implementación básica

Una implementación básica del esquema de actualización abstracto que se introdujo es el siguiente:

$$h_u^{(k+1)} = \sigma \left(W_{\text{self}}^{(k)} h_u^{(k)} + W_{\text{neigh}}^{(k)} \sum_{v \in \mathcal{N}(u)} h_v^{(k)} + b^{(k)} \right)$$

En donde σ es una función que aporta no linealidad (como ReLu o tanh), $b^{(k)}$ es el bias (que se suele omitir en la escritura por simplicidad) y $W_{\text{self}}^{(k)}, W_{\text{neigh}}^{(k)}$ son matrices de pesos aprendibles de la capa k (compartidas para todos los nodos) que transforman linealmente el embedding del nodo propio y la suma de los de sus vecinos, respectivamente. A partir de esta implementación se desarrollan una gran cantidad de variantes de la arquitectura que tienen que ver con generalizaciones y mejoras del esquema de agregación y actualización.

Introducimos dos implementaciones particulares de modelos que serán los usados más adelante en el trabajo.

GraphSAGE

En Hamilton et al., 2017 [12] se propone GraphSAGE, un modelo que mantiene el esquema de agregación como un hiperparámetro, por lo que su esquema de actualización es

$$h_u^{(k+1)} = \sigma \left(W_{\text{self}}^{(k)} h_u^{(k)} + \text{aggr}^{(k)} \{ h_v^{(k)} : v \in \mathcal{N}(u) \} \right)$$

Entre algunos de los esquemas posibles se encuentran promediar

$$\text{aggr} \{ h_v : v \in \mathcal{N}(u) \} = W_{\text{neigh}} \sum_{v \in \mathcal{N}(u)} \frac{h_v}{|\mathcal{N}(u)|}$$

o tomar máximo (entrada a entrada)

$$\text{aggr} \{ h_v : v \in \mathcal{N}(u) \} = \max \{ W_{\text{neigh}} h_v : v \in \mathcal{N}(u) \}$$

Graph Attention Networks (GAT)

Veličković et al., 2018 [14] introduce, inspirado en mecanismos de atención como el introducido por Bahdanau et al., 2015 [15], un modelo con la flexibilidad de aprender para cada nodo cuanta atención darle a sus vecinos a la hora de actualizar sus embeddings. En el modelo, el esquema de actualización viene dado por

$$h_u^{(k+1)} = W \sum_{v \in \mathcal{N}(u) \cup \{u\}} \alpha_{uv} h_v \quad (2.2)$$

Donde α_{uv} es el parámetro de atención y viene dado por

$$\alpha_{uv} = \frac{\exp \left(\sigma \left(\mathbf{a} [W h_u || W h_v] \right) \right)}{\sum_{w \in \mathcal{N}(u)} \exp \left(\sigma \left(\mathbf{a} [W h_u || W h_w] \right) \right)}$$

Donde \mathbf{a} es un vector de parámetros aprendibles y \parallel simboliza la concatenación.

La ventaja inmediata que tiene este modelo es justamente la flexibilidad de poder pesar el mensaje de los vecinos de un nodo dependiendo de su importancia para este (y notemos que esta puede variar capa a capa), sumada a la mayor interpretabilidad que puede otorgar esto.

Además, el modelo permite tener más de una *attention head* (de manera similar a los transformers introducidos por Vaswani et al., 2017 [16]), en donde (2.2) se calcula independientemente tantas veces como *heads* hayan y finalmente se concatenan o promedian las corridas. Una ilustración de la arquitectura se puede ver en la figura 2.8.

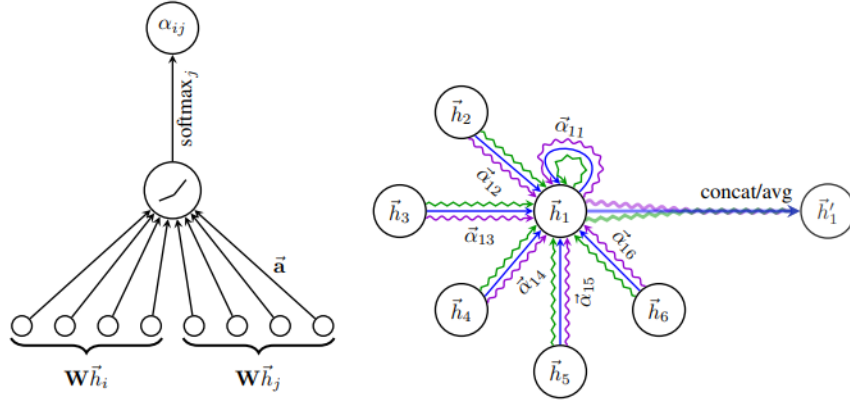


Fig. 2.8: **Izq:** Ilustración del mecanismo de atención $\mathbf{a}[\mathbf{W}h_u || \mathbf{W}h_v]$. **Der:** Ilustración del esquema de atención multi-head, en este caso con 3 cabezas. Cada línea coloreada representa una corrida independiente. (Imagen extraída de [14])

Entrenamiento

Hasta el momento únicamente presentamos la arquitectura básica de una GNN junto a dos implementaciones de esta, pero todavía no mencionamos cómo es el proceso de optimización de estas, que dependerá por lo general de la tarea a realizar.

En lo que resta del trabajo vamos a usar z_u para referirnos al embedding final del nodo u , mientras que usaremos $h_u^{(k)}$ para referirnos al embedding conseguido por el modelo en la capa k (es decir, si nuestro modelo tiene K capas, $z_u = h_u^{(K)}$).

Para ponerlo en concreto, comencemos con el caso de la clasificación de nodos que es relativamente directo. Supongamos que podemos clasificar a nuestros nodos en C clases, y contamos con un subconjunto de los nodos $V' \subset V$ para los cuales contamos con la información de su clase. El procedimiento tradicional para poder realizar la tarea de clasificación sería el siguiente:

1. Separar V' en subconjuntos disjuntos de entrenamiento, validación, y testeo ($V'_{train}, V'_{val}, V'_{test}$)
2. Inicializar features \mathbf{X} para los nodos (ya sea de manera aleatoria o con features externas).
3. Hacer una pasada forward del modelo de GNN (que en la última capa forzará a

los embeddings de los nodos a tener dimensión C y aplicara softmax sobre estos). Una observación importante es que durante la pasada forward del modelo estamos usando las representaciones de todos los nodos en el grafo, independientemente de si son de entrenamiento validación o testeo. A lo que el modelo no tiene acceso es a las clases de los nodos de validación y testeo.

4. Calcular la pérdida de clasificación sobre los nodos de entrenamiento (por ejemplo con entropía cruzada) y retropropagar.
5. repetir 3. hasta finalizar el entrenamiento
6. evaluar el modelo sobre los nodos de validación (cuyas clases no fueron observadas por el modelo a la hora de conseguir los embeddings) y ajustar los hiperparámetros.
7. evaluar el modelo final en los nodos de testeo.

Quizás la diferencia fundamental de este pipeline en contraparte con uno tradicional de clasificación es que los conjuntos de validación y testeo forman parte del entrenamiento. En este caso, la información que usamos de estos es simplemente la información de sus adyacencias en el grafo, pero no información de sus clases. Otra diferencia es también que el pipeline descrito es *transductivo*, esto quiere decir que contamos con único “punto” (es decir, un solo grafo) para entrenar y evaluar el modelo. El caso *inductivo* requeriría contar con múltiples grafos y hacer la separación de los datos a ese nivel, algo que en la práctica no ocurre con frecuencia. Una alternativa a esto sería romper el grafo en tres grafos disjuntos (y ahora independientes) para entrenar, validar y testear, pero a costa de perder información de la estructura.

Predicción de aristas

Con esto en mente, a continuación introducimos el framework de predicción de aristas, que es bajo el que se desarrollará este trabajo. El problema de predicción de aristas se puede pensar como un problema de clasificación binaria, donde el modelo estima la probabilidad de que un dado par de nodos (u, v) sea una arista del grafo. Bajo esta lógica, vamos a querer que para pares de nodos que son aristas el modelo los puntúe alto, y lo contrario para pares de nodos no conectados.

Volviendo al esquema de *encoder-decoder*, en este caso el encoder es la GNN, es decir $enc(u) = z_u$, mientras que el decoder puede ser cualquier función $dec : \mathbb{R}^d \times \mathbb{R}^d \rightarrow [0, 1]$, como por ejemplo $dec(u, v) = \sigma(z_u \cdot z_v)$, donde σ es la función logística.

El objetivo de aprendizaje va a ser minimizar entonces algún tipo de pérdida de clasificación binaria, como la entropía cruzada. Para eso, a diferencia del caso de la clasificación de nodos, necesitaremos crear un conjunto de *aristas negativas* E^- (i.e, pares de nodos que no están en el grafo para contar con instancias de la clase negativa).

Dado que ahora las *labels*, son parte de la estructura del grafo (presencia o ausencia de arista para un dado par de nodos), el split de este se hace con algo más de cuidado. En cada instancia del proceso (entrenamiento, validación, testeo), dentro de las aristas positivas E , vamos a contar con dos conjuntos de aristas: Uno al que tendrá acceso el modelo para poder realizar el pasaje de mensajes (sin acceder a sus labels), mientras que el otro será

utilizado únicamente para calcular la pérdida (es decir, virtualmente no forman parte del grafo para el modelo). Para lograr esto, además de partir E en E_{train} , E_{val} , E_{test} , vamos a tener que partir también E_{train} en E_{train}^{msg} y E_{train}^{sup} .

Para ser más precisos, el pipeline se puede describir como sigue:

1. Entrenar al modelo usando las aristas de E_{train}^{msg} , optimizando para predecir E_{train}^{sup}
2. En la etapa de validación, correr el modelo sobre E_{train} en su totalidad, evaluando sobre E_{val}
3. En la etapa de testeo, correr el modelo sobre $E_{train} \cup E_{val}$, evaluándolo en E_{test} .

Por ejemplo, la pérdida de entrenamiento sería ³

$$\mathcal{L}_{train} = - \sum_{(u,v) \in E_{train}^{sup} \cup E_{train}^{-}} A_{uv} \log(\sigma(z_u \cdot z_v)) + (1 - A_{uv}) \log(1 - \sigma(z_u \cdot z_v))$$

Un esquema de este tipo de separación de los datos se puede ver en la figura 2.9

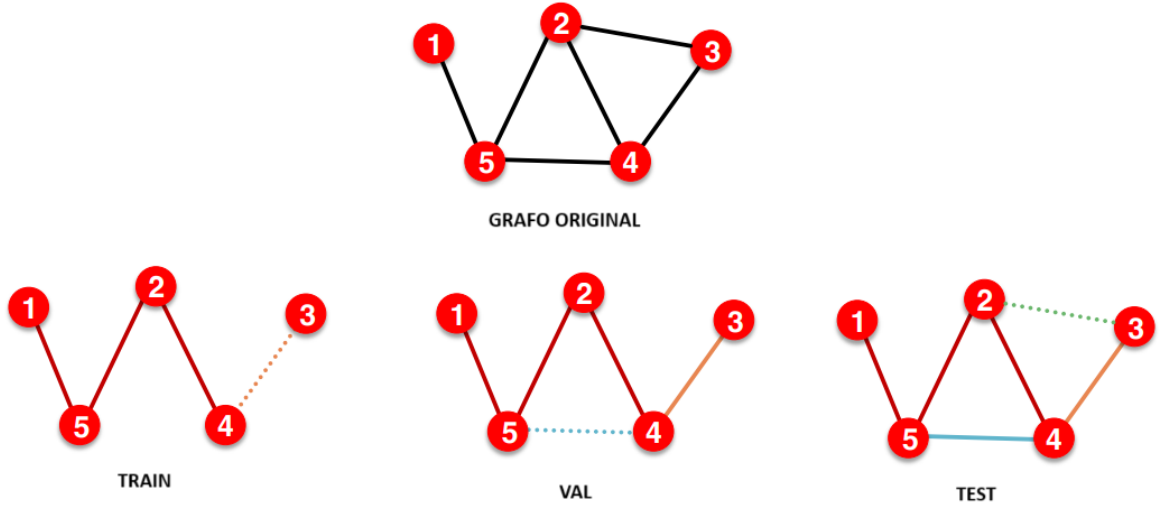


Fig. 2.9: Esquema del split del grafo. En cada etapa, las aristas llenas son sobre las cuales se corre el modelo y las punteadas sobre las que se lo evalúa. Notemos que los conjuntos sobre los cuales se corre el modelo son crecientes.

³ Por cuestiones de eficiencia, se guardan los labels de las aristas, no la matriz de adyacencia.

3. MÉTODOS

A continuación describimos la metodología seguida en el desarrollo del trabajo. Recordemos que este se basa primordialmente en dos pasos:

1. la construcción de una red de dos capas, química y genómica, donde las aristas intercapa son DTIs relevadas de repositorios públicos, mientras que las aristas intra-capas denotan similitud estructural.
2. El montado de una GNN sobre dicha red para predecir las DTIs.

3.1. Dataset

3.1.1. construcción de la red

Para la construcción de la red utilizada, primero integramos datos de DTIs de distintos repositorios públicos. Una vez hecho eso, conectamos la capa de drogas en base a su similitud dada por la distancia de *Tanimoto* entre sus fingerprints, representaciones vectoriales en bits de las moléculas. Por último, conectamos la capa de proteínas al proyectar sobre estas una red bipartita proteína - *dominio pfam*, donde los dominios PFAM son secuencias conservadas dentro de las proteínas que por lo general corresponden a unidades estructurales o funcionales de estas.

3.1.1.1. Integración de datos crudos

Los datos de interacciones entre drogas y targets fueron integrados a partir de tres bases de datos públicas: dos de ellas, MINER y Target-Decagon¹, se encuentran en la [Stanford Biomedical Network Dataset Collection](#), mientras que la última es parte de la Comaprative Toxigenomics Database² (CTD.)

Tanto MINER como Target-Decagon son bases que se encuentran ya curadas y preprocesadas en el formato de tabla de interacciones (Id droga, Id gen) y datan sobre interacciones verificadas en experimentos biológicos o estudios farmacológicos. En ambas bases los genes relevados son del organismo *Homo sapiens sapiens*.

Por otro lado, las interacciones relevadas desde CTD se encuentran sin el pre-procesamiento necesario para poder ser utilizadas. Para eso, filtramos únicamente las interacciones a las que el target es una proteína corresponde a humanos y de estas nos quedamos solo con las interacciones físicas (en CTD hay interacciones relevadas cuyos targets no son estrictamente proteínas, como ácidos nucleicos, y cuyo mecanismo de acción tampoco es necesariamente físico, como puede serlo el efecto de una cadena de señalización). Esto se puede observar en la tabla 3.1.

¹ Se pueden descargar en [1](#) y [2](#) respectivamente.

² Disponible para descargar [aquí](#).

ChemicalID	GeneID	GeneForms	Organism	Interaction
C534883	367	protein	Homo sapiens	affects reaction increases expression
C534883	367	protein	Homo sapiens	decreases reaction increases expression
C095360	5243	protein	Homo sapiens	10-deacetylpaclitaxel binds to ABCB1

Tab. 3.1: Ejemplo de la información contenida en CTD. Notese que para un dado par (droga, target) puede haber más de una interacción relevada dependiendo del *tipo* de esta.

Por último, para integrar las tres bases de datos estandarizamos los Ids de las drogas y genes a sus Ids de PubChem y Entrez Gene del NCBI, respectivamente.

3.1.1.2. Capa de drogas

Para conectar las drogas entre ellas, seguimos una metodología similar a la de Landaburu et al., 2019 [17].

Primero, conseguimos los Ids SMILES (usando el [sistema de mapeo](#) de Ids de PubChem) de todas las drogas a partir de su ID. Un Id SMILES (por Simplified Molecular Input Line Entry System), es una representación de la estructura de una molécula en forma de *string*, conseguida al imprimir los símbolos (átomos y enlaces) de la molécula encontrados en un recorrido DFS de su grafo molecular, como se ilustra en la figura 3.1 (dado que para una misma molécula habrá varias representaciones, los algoritmos usados devuelven una única de estas, de manera sistemática).

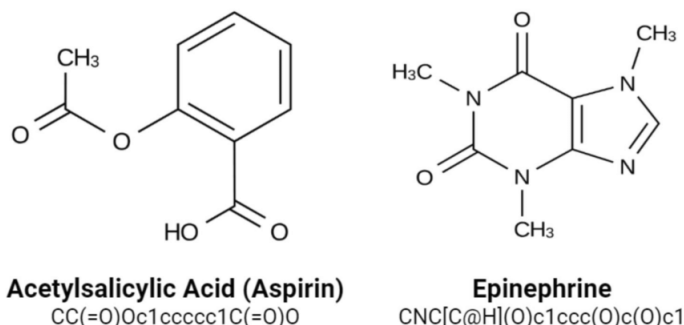


Fig. 3.1: Ejemplo de las representaciones SMILES para las moléculas de la aspirina y la epinefrina.

Con los Ids de SMILES conseguidos para las drogas, estos se parsean para conseguir, para cada droga, lo que se conoce como un Fingerprint molecular. Un Fingerprint molecular es una representación en forma de bit-vector (i.e, un vector de 1 – 0) donde las posiciones en el vector representan la presencia o ausencia de ciertas subestructuras, como se ve en la figura 3.2. Luego, a partir de los fingerprints de las moléculas se puede calcular su similitud de *Tanimoto*, que cuantifica cuan parecidas son dos moléculas a partir de la cantidad de subestructuras que comparten, es decir

$$Tanimoto(A, B) = \frac{A \wedge B}{A \vee B}$$

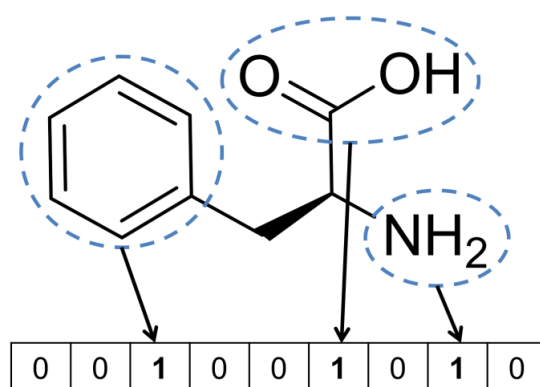


Fig. 3.2: Ilustración de la información que representa un fingerprint molecular.

Distintos tipos de fingerprints se diferencian en la forma en la que se consiguen los vectores. En este trabajo utilizamos fingerprints de tipo circular de radio 3, que para cada átomo en la molécula, se codifica la información contenida en su vecindario (a distancia 3) mediante una función hash que luego es almacenada en una de las posiciones del vector resultante final (en este caso de longitud 2048).

También se experimento con dos tipos de fingerprint más (todos provenientes del paquete RDKit de Python): el de RDKit, que en lugar de codificar la estructura alrededor de un átomo de forma radial lo hace mediante caminos), y el de MACCS, que utiliza subestructuras previamente preestablecidas.

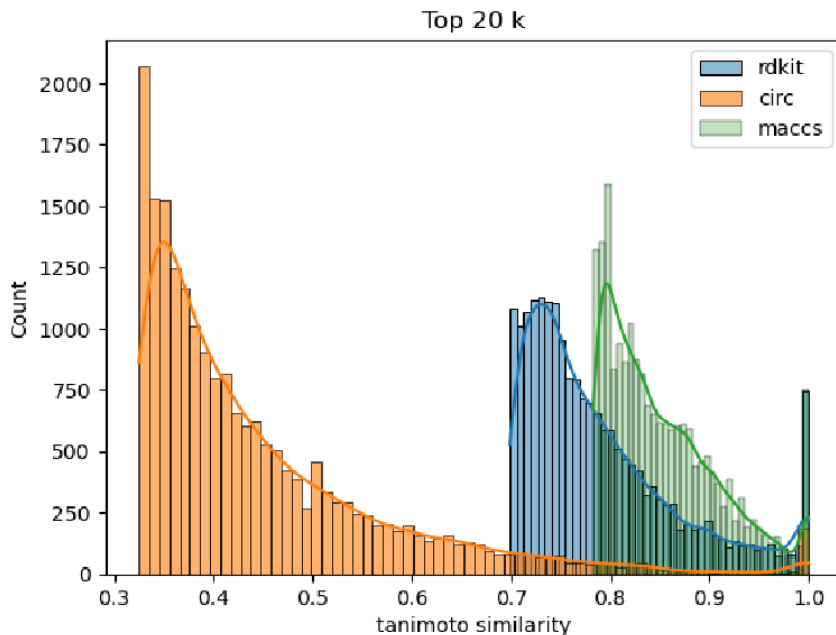


Fig. 3.3: distribución de distancias de Tanimoto de a pares para distintos tipos de fingerprints.

En la figura 3.3 se observa la distribución de scores para el top 20000 de pares para cada tipo de fingerprint. Si bien las alternativas al fingerprint utilizado muestran valores de

similitud mayores, la intersección entre los 20000 pares obtenidos en cada caso es similar en todos los casos (del orden de 10000 pares para los 3 fingerprints). Se decidió utilizar los fingerprints circulares en base a la bibliografía ([18], [19]), que indica que son los más utilizados a la hora de calcular similitud estructural entre pares.

Por último, una vez calculada la similitud de *Tanimoto* para todos los pares de moléculas, se eligieron los 20000 pares de mayor similitud para conectarlos mediante aristas, formando así la red correspondiente a la capa química.

3.1.1.3. Capa de proteínas

Para conectar la capa de proteínas, realizamos lo siguiente:

1. Mapeamos las proteínas a dominios Pfam, generando un grafo bipartito entre proteínas y dominios
2. Realizamos una proyección del grafo a las proteínas, pesada por la cantidad de anotaciones a cada dominio
3. Elegimos un valor de corte en los pesos resultantes entre las proteínas para a partir del cual las conectamos mediante aristas.

Pfam

Pfam (por Protein families), es una base de datos que agrupa proteínas en familias en base a sus secuencias (a partir de alineamientos múltiples y modelos ocultos de Markov (HMMs) sobre estas). Cada proteína en la base de datos puede tener anotaciones sobre, por ejemplo, a que familia pertenece (si lo hace) y también anotaciones específicas sobre regiones de su secuencia. En concreto, muchas proteínas tienen anotadas sobre partes de su secuencia ciertos dominios, regiones específicas asociadas a funciones o unidades estructurales (por ejemplo, sitios de unión al ADN), como se ejemplifica en la figura 3.4. Intuitivamente, se podría esperar que dos proteínas sean similares si comparten muchos de estos dominios, pero pesando los dominios que comparten en cuanto a su especificidad (desde el punto de vista informativo no es lo mismo un dominio que está contenido en muchas proteínas que uno que está contenido en algunas pocas).

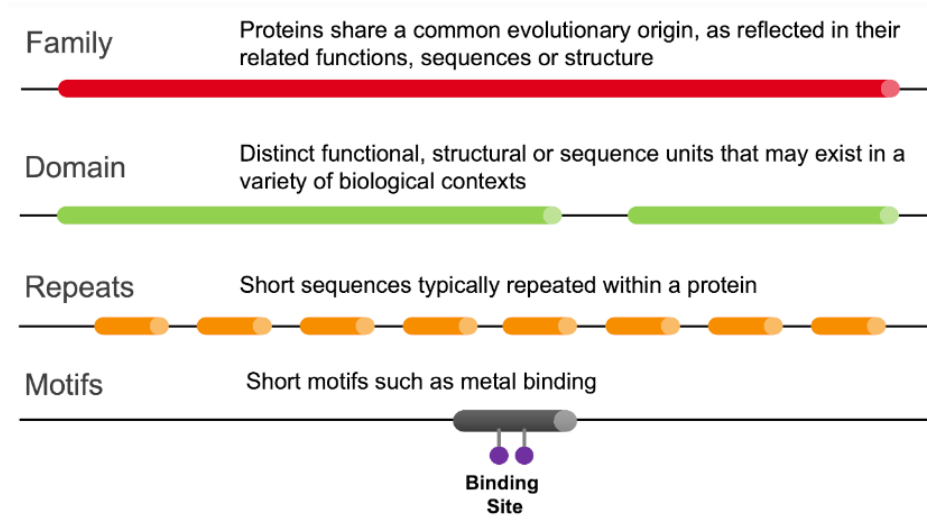


Fig. 3.4: Algunas entradas a las que puede estar anotada una proteína en Pfam. En verde los dominios.

Una vez conseguidas las anotaciones de las proteínas a los dominios, se define naturalmente un grafo bipartito entre estos. Si $A \in \{0, 1\}^{m \times n}$ es la matriz de incidencia del grafo (donde tenemos m dominios y n proteínas, y $A_{ij} = 1$ indica que la proteína número j esta anotada al dominio número i), entonces podemos definir el peso entre dos pares de proteínas j, k como

$$w_{jk} = \sum_{i=1}^m \frac{1}{d_i} A_{ij} A_{ik}$$

Que si D es la matriz diagonal de grados de los dominios, se puede computar eficientemente como

$$W = A^t D^{-1} A$$

Distribucion de pesos

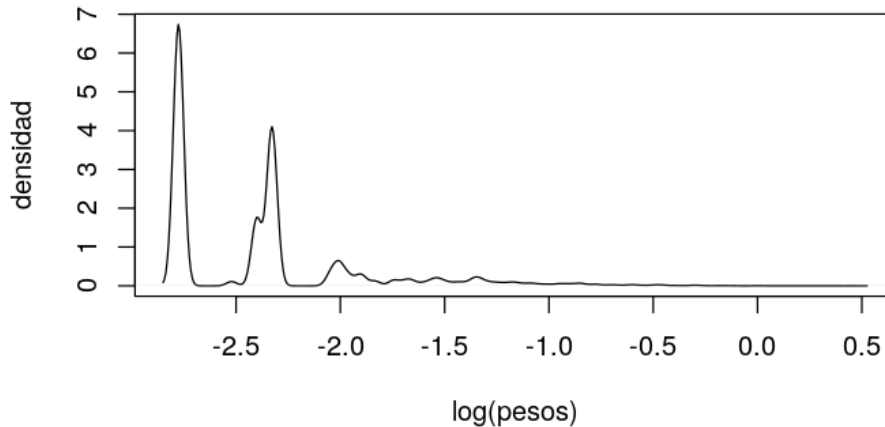


Fig. 3.5: distribución de pesos (únicamente mayores a 0) entre todos los pares de proteínas según la cantidad de dominios compartidos.

En la figura 3.5 se observan en los dos picos el fenómeno descrito donde existen dominios compartidos por muchas proteínas (y ese es uno de los únicos que comparten). Para ejemplificar, en el pico correspondiente a los pesos menores a $\log(-2.5)$ se encuentran pares de proteínas donde las únicas anotaciones que comparten son o bien a un dominio de *Dedo de Zinc C2H2*, o un dominio denominado *KRAB* (presente en muchas proteínas que contienen dedos de Zinc). En particular estos dominios tienen alrededor de 900 proteínas anotadas

Finalmente, para quedarnos únicamente con relaciones significativas (y balancear la cantidad de enlaces de la capa química) decidimos conectar todos los pares de proteínas para los cuales el logaritmo de su peso fuera mayor a -1.5 , formando la red correspondiente a la capa de proteínas.

La red final sobre la cual se montara la GNN para la predicción de DTIs consiste en la unión de ambas capas conectadas mediante las DTIs relevadas.

3.1.2. Análisis de la red

En esta sección realizamos el análisis de la red obtenida. Teniendo en cuenta que por cada capa i de la GNN esta acerca nodos a distancia i , un aspecto importante del análisis será analizar la conectividad de la red.

En la tabla 3.2 vemos un resumen de las propiedades globales de la red entera y los tres subgrafos que la componen.

Red	# nodos	# aristas	% nodos en la CG	~ distancia media	~ diametro
Red completa	11659	80221	0.999	3.9006	11
Droga-Droga	4469	20000	0.597	8.3524	29
Droga-Gen	11659	38391	0.983	4.3902	13
Gen-Gen	3569	21830	0.395	6.0942	15

Tab. 3.2: Resumen de las propiedades de las redes. La distancia media y el diámetro son aproximaciones.

Vemos que si sumamos los nodos en la capa química y en la capa genómica no completamos los nodos en la red entera. En particular, la red entera contiene 5854 drogas y 5805 genes, lo que quiere decir que 2236 genes no contienen conexiones a otros genes (ya sea porque no contaban con anotaciones a dominios Pfam o porque sus pesos con otros genes en la red pesada eran muy bajos) y 1385 drogas no contienen conexiones con otras drogas (por razones análogas).

De todas formas, la gran mayoría de estos nodos aislados en sus respectivas capas tienen conexiones hacia la otra capa en la que se encuentran, como se evidencia en el porcentaje de nodos que contiene la componente gigante de la red completa. Esto asegura que el pasaje de mensajes en la red se puede realizar de manera correcta.

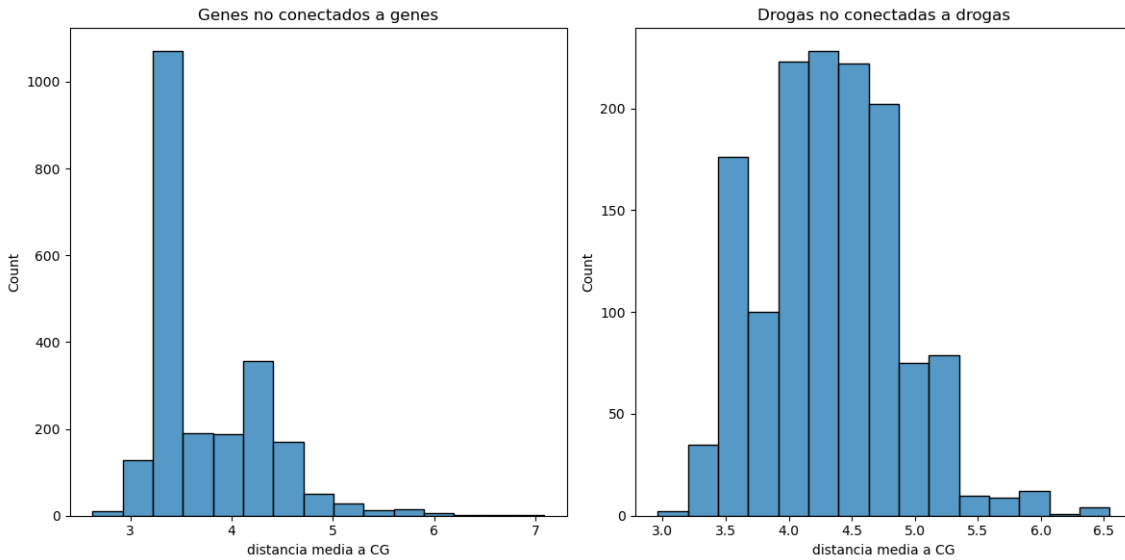


Fig. 3.6: Distribuciones de distancias medias a la componente gigante desde nodos y drogas no conectadas en su capa, respectivamente.

Vemos en la figura 3.6 que tanto los genes y las drogas desconectadas en sus respectivas capas se encuentran bien conectados en la red, teniendo distancias medias a la componente gigante relativamente cortas.

Otra característica importante de la red es su distribución de grado. A continuación lo analizamos para cada capa por separado.

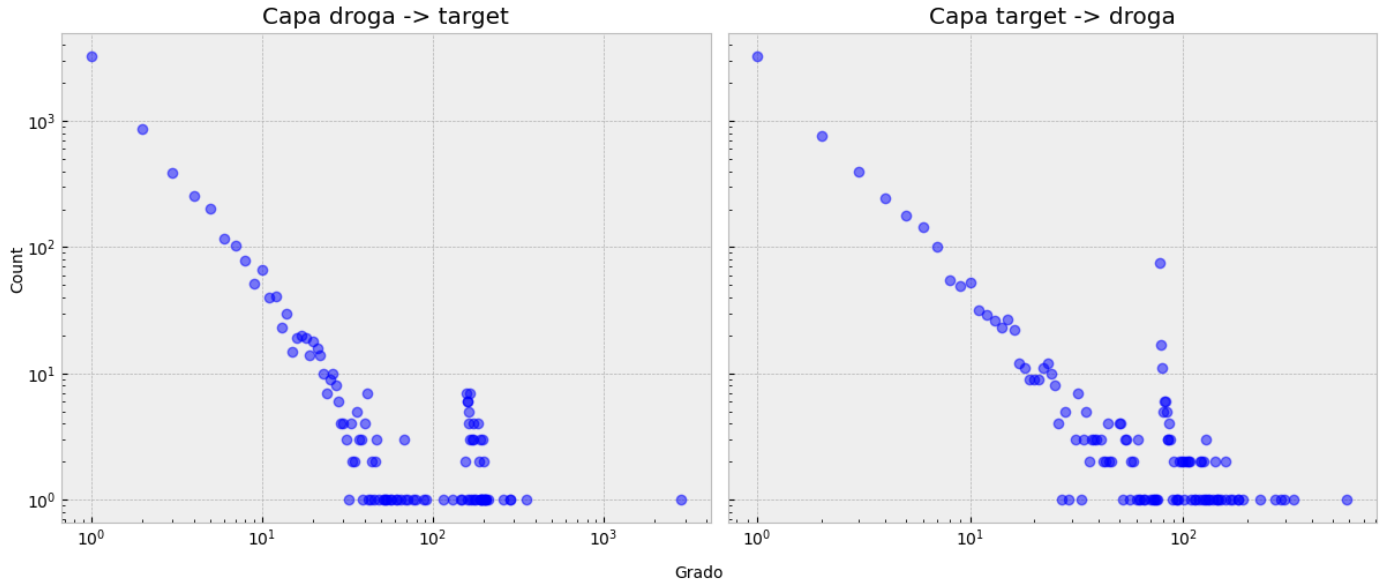


Fig. 3.7: *distribución de grado para la capa droga-target, separados por tipo de nodo. Notemos que las distribuciones se asemeja a las libre de escala ya mencionadas, donde muchos nodos tienen pocas conexiones, y pocos nodos tienen muchas*

En el la figura 3.7 se observan dos picos cerca de ~ 100 para ambos gráficos. El primero se corresponde a drogas utilizadas en tratamientos diversos, como por ejemplo para enfermedades mentales (Haloperidol, Citalopram, Fluoxetina, etc.). El segundo corresponde a conjuntos de genes que codifican proteínas similares (por ejemplo, citocromos) para los cuales se diseña una gran cantidad de drogas distintas cuyo target son esas proteínas. Además, se puede observar que hay un único nodo que presenta altísima conectividad, un orden de magnitud mayor al segundo nodo de mayor grado. Analizándolo, vemos que el nodo corresponde al catión Calcio (Ca^{+2}). Un problema que podría presentar tener este nodo en la red (conocidos como *hubs* o super-conectores), es que acerque demasiado a nodos poco relacionados, permitiendo el intercambio de información entre estos y aportando ruido a la hora de realizar predicciones. Como primer acercamiento al problema decidimos mantener la red intacta, pero una posible solución podría ser quitar el nodo de la red.

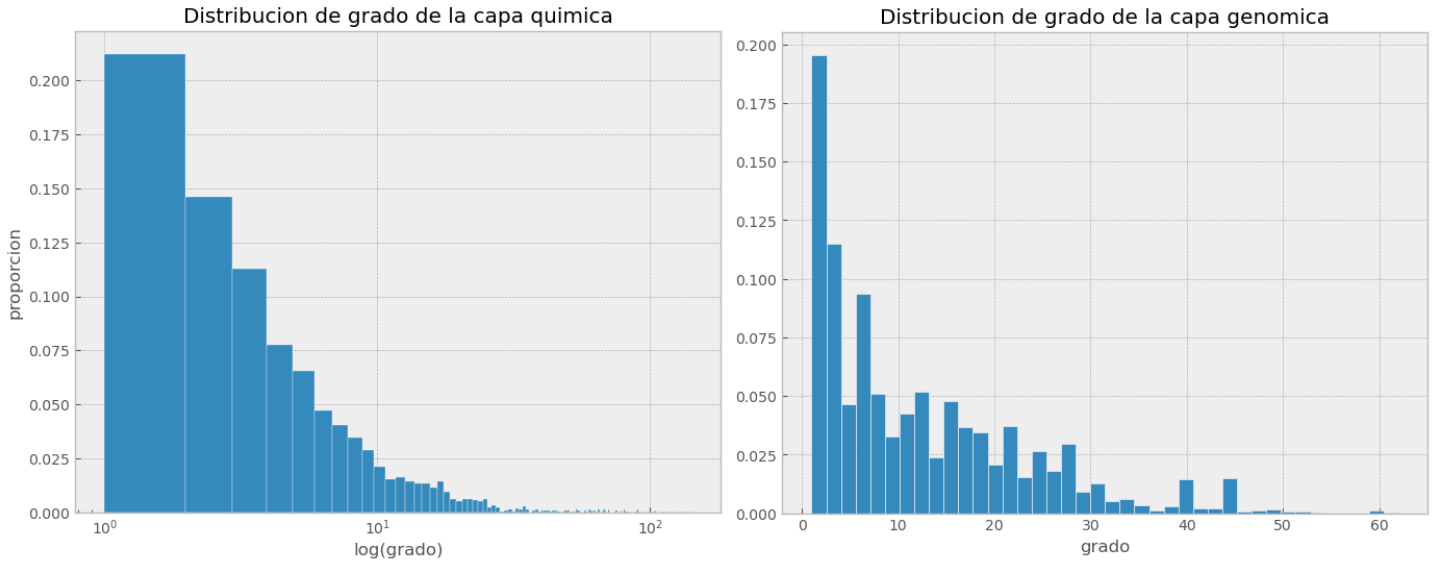


Fig. 3.8: *izq:* distribución de grado para la capa química. *der:* distribución de grado para la capa genómica. Notemos que el grado en la capa química se encuentra en escala logarítmica, debido a que tiene una gran cantidad de nodos espaciados en el rango de grados ~ 30 -100.

Junto con las estadísticas globales de conectividad detalladas anteriormente, las distribuciones de grado (observadas en las figuras 3.7 y 3.8) sirven para tener una idea general de como se puede dar el proceso de intercambio de mensajes entre los nodos en el modelo de GNN.

La presencia de nodos de alta conectividad, como ya se menciono, puede traer ciertos problemas, pero a la vez garantizan que el modelo pueda integrar información global (tanto de la estructura, como a nivel de features) del grafo, algo que podría ayudar en cuanto al poder de generalización (intuitivamente, aprender únicamente patrones de conectividad en subgrafos localizados debería ser más complicado que hacerlo contando con información más “global”). Por otro lado, que el modelo cuente con muchos más ejemplos positivos para ciertos nodos podría generar sesgos a la hora de hacer predicciones, teniendo más dificultad para poder predecir los enlaces de nodos de poco grado. Esto se explorara más adelante.

Por último, realizamos un análisis de transitividad sobre la capa droga-target, que servirá como justificación inicial de que implementar una GNN sobre este grafo podría traer buenos resultados para predecir DTIs.

Debido al mecanismo de pasada de mensajes a través del cual funcionan las GNN, una hipótesis implícita que impone sobre la red es que drogas (genes) conectados tenderán a conectarse con los mismos genes (drogas). Por lo tanto, verificar si esto se cumple en la red es una buena idea. Para hacer esto, vamos a computar un coeficiente de clustering global levemente modificado: como se describió en la sección 2.2, el coeficiente mide la proporción de triángulos cerrados sobre triplas conectadas. En este caso, solo vamos a contar triángulos de la forma (droga, gen, droga) o (gen, droga, gen), mientras que las triplas conectadas serán iguales pero sin la arista en la capa de las drogas (targets) que cierra el triángulo, como se ejemplifica en la figura 3.9.

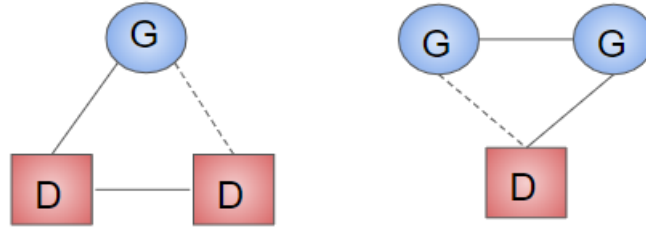


Fig. 3.9: Ilustración de los triángulos contemplados a la hora de computar el coeficiente. Se cuenta, en proporción, cuantas de las líneas puntuadas están llenas en el grafo.

Luego, para determinar si el valor computado en la red es significativo, lo comparamos con el mismo valor para redes aleatorias. En este caso, manteniendo la conectividad de las capas químicas y genómicas, recableamos de manera aleatoria (manteniendo la distribución de grado) las DTIs, y computamos el valor en la nueva red. Si el valor encontrado en esa red recableada es similar al de la red real, eso querría decir que las DTIs reales son iguales que DTIs simuladas a la hora de completar los triángulos mencionados, algo que rechazaría la hipótesis mencionada. Para darle robustez al análisis, simulamos el recableado 1000 veces y graficamos la distribución de los coeficientes calculados.

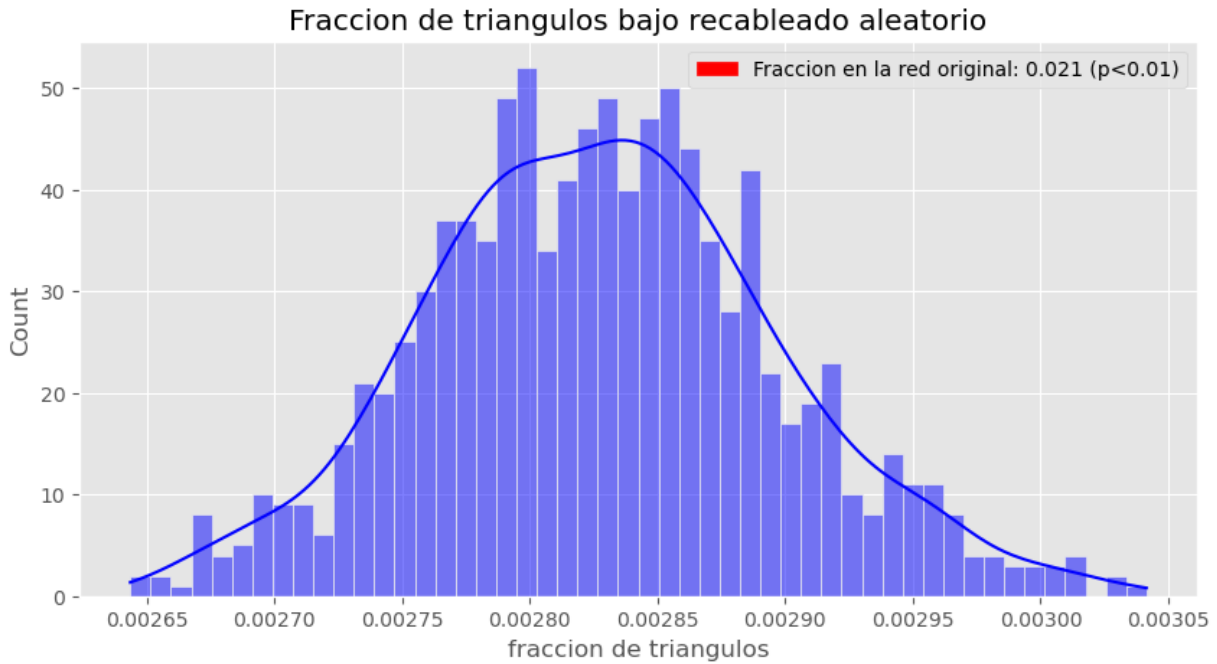


Fig. 3.10: distribución de la fracción de triángulos en 1000 simulaciones de redes recableadas.

Como se observa en la figura 3.10, la fracción computada en la red real es sumamente mayor que en las redes simuladas, por lo que justifica en alguna medida el uso de GNNs para predecir DTIs.

3.2. Implementación

Antes de describir la arquitectura del modelo implementado es necesario remarcar algunas cosas respecto a la naturaleza de la red sobre la que se implementara el modelo. Recordemos la **Implementación básica** de una GNN:

$$h_u^{(k+1)} = \sigma \left(W_{\text{self}}^{(k)} h_u^{(k)} + W_{\text{neigh}}^{(k)} \sum_{v \in \mathcal{N}(u)} h_v^{(k)} + b^{(k)} \right)$$

Por simplicidad notacional, omitimos el bias y usamos la misma matriz de pesos tanto para el nodo y sus vecinos, es decir:

$$h_u^{(k+1)} = \sigma \left(W^{(k)} \sum_{v \in \mathcal{N}(u)} h_v^{(k)} \right)$$

Notemos que en este modelo la misma matriz de pesos es usada para todos los vecinos de un dado nodo, independientemente del tipo de nodo de su vecino y la relación que está definida entre estos.

En particular, la red utilizada tiene dos tipos de nodos (drogas y targets) y tres tipos de relaciones (droga-droga, droga-target, target-target). Las redes de este tipo, que consisten en múltiples tipos de nodos y relaciones se denominan redes *heterogéneas*.

Un dado enlace en una red heterogénea se representa mediante una tripla (s, τ, t) , que dice que el nodo s se relaciona con t mediante la relación τ .

Luego, si \mathcal{R} es el conjunto de relaciones que existen en la red y para una dada $\tau \in \mathcal{R}$, $\mathcal{N}_\tau(u)$ es el conjunto de los vecinos de u que están conectados mediante la relación τ , se puede definir el siguiente esquema de agregación modificado:

$$h_u^{(k+1)} = \sigma \left(\sum_{\tau \in \mathcal{R}} W_\tau^{(k)} \sum_{v \in \mathcal{N}_\tau(u)} h_v^{(k)} \right)$$

Donde ahora para cada relación τ contamos con una matriz de pesos W_τ que transforma los embeddings del vecindario restringido a esa relación. La ventaja de definir este esquema es que ahora permite tratar a los vecinos de manera separada dependiendo de la relación que el nodo tenga con estos, aportándole más expresividad al modelo.

3.2.1. Arquitectura

De forma general, la arquitectura del modelo es una de tipo *encoder-decoder*, donde el encoder se encarga de generar los embeddings finales z_u para cada nodo u y el decoder se encarga de darle un *score* a cada par de nodos en base a sus embeddings finales que será interpretado como la probabilidad de que exista un enlace entre estos.

Siguiendo el diseño presentado por You et al., 2020 [20] (figura 3.11) el *encoder* del modelo consiste en capas de preprocesamiento (MLPs) de las features iniciales, luego alguna arquitectura de GNN, y por ultimo capas de post-procesamiento (MLPs) de los embeddings dados por la GNN.

Como mencionamos en la **Implementación básica**, en este trabajo exploraremos dos arquitecturas de GNN diferentes, SAGE y GAT.

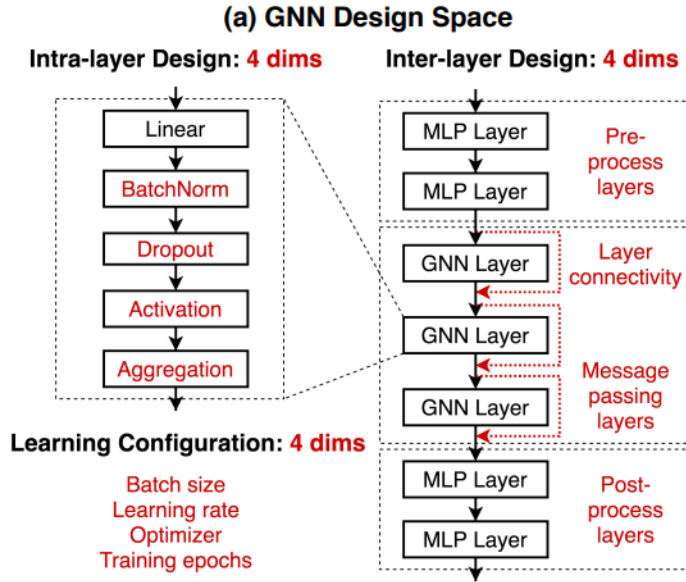


Fig. 3.11: Esquema del espacio de diseño utilizado. Imagen extraída de [20]

Observemos que el diseño se puede separar en tres partes; El diseño inter-capas (Inter-Layer) que controla cuantas de las capas de las mencionadas tendrá el modelo y cómo estarán conectadas, el diseño intra-capas (Intra-Layer) que controla la configuración de cada capa, y la configuración de aprendizaje (Learning configuration), que controla los hiperparámetros relacionados al proceso de entrenamiento. Todos estos hiperparámetros *definen* el espacio de diseño de los modelos posibles, y una especificación de los valores de estos definen un modelo.

A continuación detallamos los hiperparámetros utilizados en el espacio de diseño que definimos para el trabajo.

Diseño global

1. Dimensión de los features

Determina la dimensión de los features iniciales (en ausencia de features externos). Sus valores posibles son 16, 32, 64, 128.

2. Cantidad de capas de pre-procesamiento

Varía entre 0 y 2 y determina la profundidad de la red feed-forward que toma como input las features iniciales

3. Cantidad de capas de GNN

Varía entre 1 y 4 y determina cuantas iteraciones de pasada de mensajes realiza la GNN.

4. Conectividad entre capas

Si es especificada, determina como se incorpora el input en una dada capa de GNN al output de esta. Es decir, si \mathbf{x} es el input en una dada capa, \mathbf{y} es el output y f es la función que realiza una iteración de pasada de mensajes en el grafo, entonces:

Si la conectividad es del tipo “suma” (sum), $\mathbf{y} = f(\mathbf{x}) + \mathbf{x}$.

Si es del tipo “concatenación” (cat), $\mathbf{y} = [f(\mathbf{x}) || \mathbf{x}]$.

Este tipo de conectividad se conoce como *skip-connection*, y en el caso de las GNN permite que la influencia de un propio nodo a la hora de actualizar sus embeddings no se diluya con la cantidad de capas.

5. Cantidad de capas de post-procesamiento

Varía entre 0 y 2 y determina la profundidad de la red feed-forward que toma como input los embeddings generados por las capas de la GNN.

Diseño intra-capas

1. Dimensión de la capa oculta

Determina la dimensión de salida de todas las capas del modelo. Su rango de valores es el mismo que para las features iniciales.

2. Batch normalization

Si es especificada, se aprende una estandarización de los embeddings después de cada capa de pasada de mensajes. Si x es el embedding correspondiente a un nodo, entonces para cada nodo se computa

$$y = \frac{x - E[x]}{\sqrt{Var[x] + \epsilon}} \gamma + \beta$$

Donde ϵ es un escalar (por default 10^{-5}) y γ, β son vectores de pesos aprendibles de la longitud de x .

3. Dropout

parámetro que determina con probabilidad $p \in [0, 0.3, 0.5]$ si un dado elemento de la matriz de embeddings \mathbf{X} se convierte en 0 (de manera forzada).

4. Norma L_2

Solo aplicable para las capas de GNN, que de aplicarse, normaliza los embeddings de las capas de GNN por su norma L_2

5. Agregación macro

Define si los embeddings de las distintas relaciones τ se suman o se promedian.

6. Agregación micro

Solo aplica para SAGE. Determina el tipo de agregación de vecindario usada (suma, promedio o máximo).

7. Cabezas y concatenación

Solo aplican para GAT, con valores en $[1, 2, 3]$. Determina la cantidad de cabezas de atención utilizadas y si son concatenadas o promediadas.

Configuración de aprendizaje

1. Cantidad de épocas

Controla la cantidad de pasadas forward del modelo en la etapa de entrenamiento.

2. Tasa de aprendizaje

Varia entre $[0.01, 0.001, 0.0001]$. Controla el tamaño de paso del optimizador.

En cuanto al *decoder* del modelo, utilizamos un decoder simple de producto interno, es decir:

$$dec(u, v) = \sigma(z_u \cdot z_v)$$

Donde σ es la función logística aplicada entrada a entrada, $\sigma(x) = \frac{1}{1 + e^{-x}}$

3.2.2. Enriquecimiento de features

Otra variable de nuestro modelo son las features iniciales utilizadas. Si bien en [21] se demuestra y comprueba empíricamente que la inicialización aleatoria le aporta expresividad a las GNNs (en contraparte con la limitación dada por el test de isomorfismo de Weisfeiler-Lehman, demostrada por Xu et al., 2019 [22]), una gran ventaja que tienen estos modelos es el poder incorporar features externas para los nodos cuando estas están disponibles.

En este caso, conseguimos features iniciales para un subconjunto de los genes, siguiendo una metodología similar a la de Go2vec desarrollada Zhong et al., 2019 [23].

Las features corresponden a embeddings de los genes en un grafo donde se puede definir una distancia semántica respecto a su función molecular, como se detalla a continuación.

Gene Ontology

El Gene Ontology (GO), es un vocabulario ordenado diseñado para estandarizar y catalogar anotaciones de genes y productos génicos, facilitando el análisis de datos provenientes de estos. Organiza su conocimiento en tres grandes áreas; procesos biológicos, funciones moleculares y, componentes celulares. Cada área se compone de un conjunto de **términos**, los cuales describen algún aspecto del área y se organizan de manera jerárquica en un grafo acíclico dirigido (DAG), donde si un término es hijo de otro, esto quiere decir que el primero se encuentra dentro de la “categoría” del segundo. Esto se puede ver en la figura 3.12.

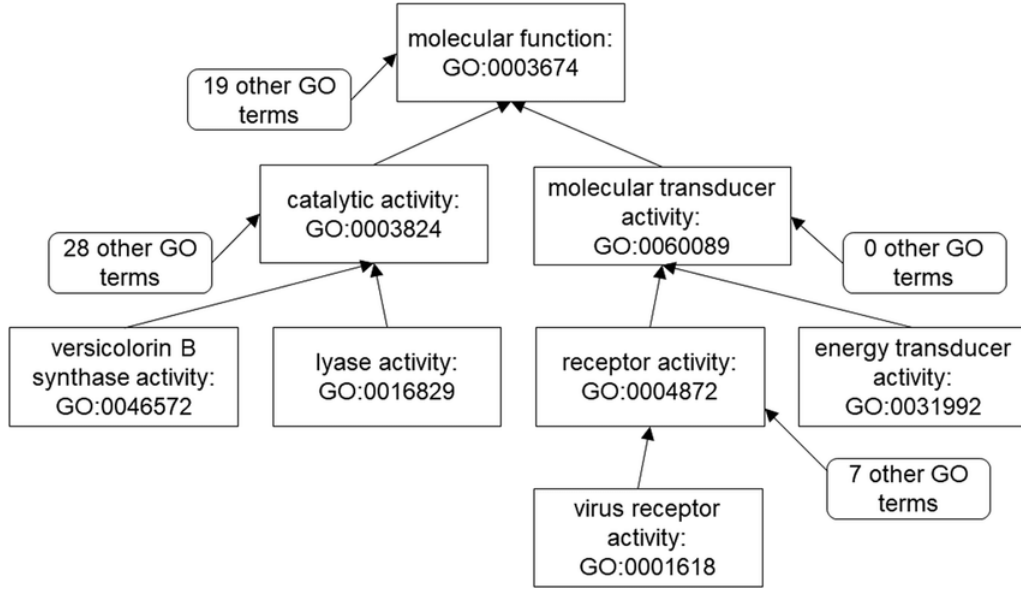


Fig. 3.12: Esquema del DAG que define GO (en este caso para funciones moleculares), donde cada término está asociado a un Id. Notemos la estructura jerárquica del grafo de menos a más específico, yendo de arriba hacia abajo.

Dependiendo de su función, una dada proteína va a estar anotada a ciertos términos en el grafo (se puede pensar también que si una proteína está anotada a un término, también está anotada a todos sus ancestros) generando un grafo ampliado que consiste en el mostrado junto a las proteínas y sus anotaciones, que denominaremos GOA. Luego, dada la naturaleza jerárquica del grafo, se puede pensar que medir distancias entre proteínas allí puede correlacionar con similitud entre ellas a nivel de su función molecular, que a su vez correlaciona con similitud a nivel estructural.

Es importante remarcar que en el grafo no hay aristas entre proteínas, que son últimamente las que vamos a querer predecir para determinar si los embeddings conseguidos son informativos. Para eso, vamos a definir una distancia entre pares de proteínas a partir de los embeddings de los términos a los que están asociadas, y usar esa distancia como un predictor de enlace entre las proteínas, algo que comparamos contra las aristas de la capa genómica de nuestra red.

Concretamente, realizamos lo siguiente:

1. corremos sobre el GOA **node2vec**, un método que produce embeddings para todos los nodos del grafo introducido en la sección de **Encoder-Decoder**.
2. para dos proteínas p_1, p_2 , si V_1 y V_2 son los conjuntos de términos a las que están asociadas, podemos calcular su similitud como

$$s(p_1, p_2) = \min \left\{ \frac{1}{|V_1|} \sum_{v_i \in V_1} \max_{v_j \in V_2} \cos(v_i, v_j), \frac{1}{|V_2|} \sum_{v_j \in V_2} \max_{v_i \in V_1} \cos(v_i, v_j) \right\}$$

$$\text{Donde } \cos(v_i, v_j) = \frac{v_i \cdot v_j}{\|v_i\| \|v_j\|}.$$

3. Generamos la misma cantidad de aristas negativas que con las que contamos (las aristas de la capa genómica).

4. Para cada arista (tanto positiva como negativa) calculamos su probabilidad de enlace como la similitud entre los nodos que la componen.
5. reportamos el AUC como medida de performance.

En una primera iteración realizamos este procedimiento sobre el grafo original, obteniendo un AUC de 0.7. Analizando la distribución de scores obtenida, vimos que esta era relativamente alta pero no difería mucho entre las aristas positivas y las negativas (ver figura 3.13), lo que indica que se estaba puntuando alto pares de nodos que no estaban conectados.

Teniendo en cuenta la función de similitud, algo que podría estar ocurriendo es que, en genes con pocas anotaciones, estas sean hacia términos muy altos en la jerarquía. Por ejemplo, si dos nodos están anotados únicamente a unos pocos términos de alta jerarquía, la similitud entre estos será cercana a 1, algo que aporta ruido en el calculo de la performance.

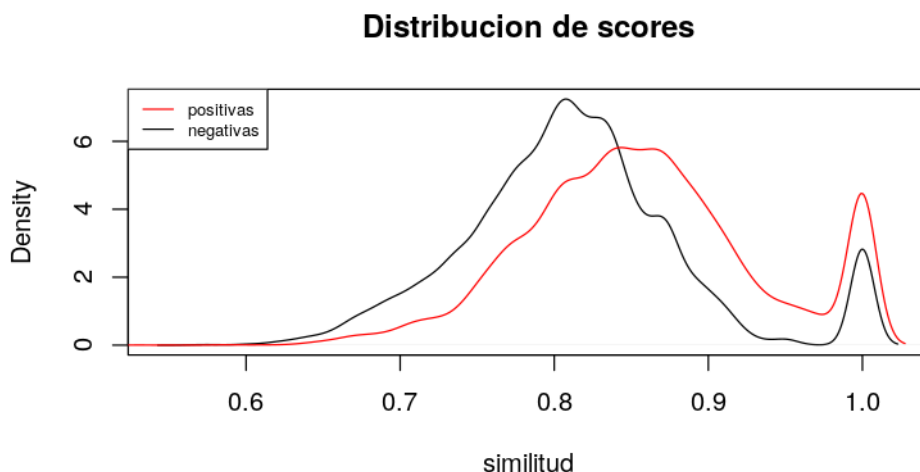


Fig. 3.13: *Distribución de scores. Se observa un pico de las aristas negativas en 1 también, un caso extremo del problema mencionado.*

Para sortear esta dificultad, decidimos quitar del GOA las anotaciones hacia nodos “poco informativos”. Dado un término t del GO, se puede definir su *contenido de información* (IC) como $-\log_2(p_t)$, donde p_t es la probabilidad de encontrar una instancia del término en la red. Esto es, si hay N anotaciones totales y el término t tiene anotadas (donde para este cálculo incluimos las anotaciones de sus descendientes), entonces $p_t = \frac{m}{N}$.

Decidimos quitar anotaciones a todos los términos cuyo IC sea menor a $-\log_2(0.05)$, es decir, términos con probabilidad mayor a 0.05 de ser “encontrados” en una anotación aleatoria (Ver distribución de IC en la figura 6.1 del apéndice).

Repetimos el procedimiento mencionado en este nuevo grafo (además realizando un barriado de los hiperparámetros de **node2vec**), consiguiendo un AUC de 0.88.

En total, conseguimos features de dimensión 64 para 3291 genes (de un total de 5805). Como se observa en la figura 3.14 la distancia en la red de genes sin features a genes con features es muy baja, esto garantiza que la información de los features de los genes se podrá difundir de manera rápida sobre todos.

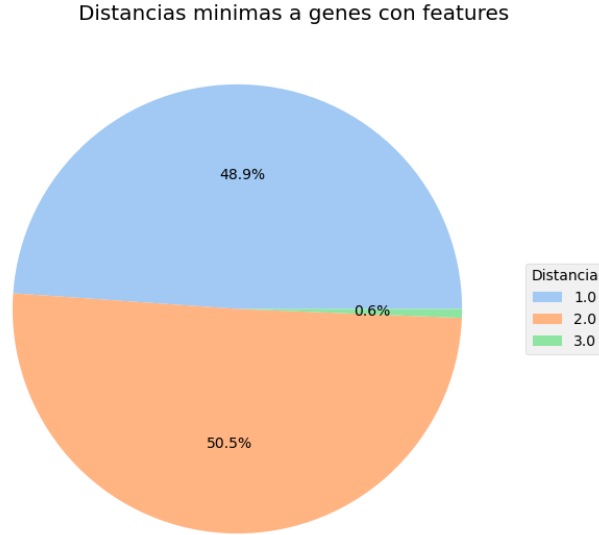


Fig. 3.14: Distancias de genes sin features a genes con features.

3.2.3. Especificaciones de entrenamiento

Recordemos que, como se mencionó en la sección de **Predicción de aristas**, para poder entrenar este tipo de modelos en esta tarea es necesario crear aristas negativas tanto para los conjuntos de entrenamiento (únicamente para el de supervisión), y para las aristas de supervisión de los conjuntos de validación y testeo. En este caso, creamos cantidades iguales de aristas negativas para sus respectivos conjuntos, de modo que las clases se encuentren balanceadas a lo largo de los *splits*. Siguiendo el trabajo de Yang et al., 2020 [24], conseguimos aristas negativas donde los nodos se muestrean con probabilidad proporcional a su grado^{0.75}.

Dado que nuestro objetivo es predecir DTIs, el split de las aristas lo hacemos únicamente sobre las de la capa droga-target, en un 80 % – 10 % – 10 %. A su vez, separamos el 20 % de las aristas de entrenamiento como aristas de supervisión para optimizar los pesos del modelo. En la tabla 3.3 mostramos las especificaciones del split.

Conjunto	Aristas de pasaje de mensajes	Aristas de supervisión	Total
Entrenamiento	24570	6142	30713
Validación	30713	3839	34552
Testeo	34552	3839	38391

Tab. 3.3: especificación del split. Para cada conjunto de supervisión, hay la misma cantidad de aristas negativas.

Por ultimo, utilizamos como función de pérdida la entropía binaria cruzada que minimizamos utilizando el optimizador ADAM [25].

3.2.4. Selección y validación

Debido al tamaño del espacio de hiperparámetros y el tiempo de entrenamiento (30 segundos por modelo, para $\sim 1.49 \times 10^6$ configuraciones posibles para SAGE, $\sim 4.47 \times 10^6$ para GAT) se hace imposible una optimización de estos al hacer una búsqueda en grilla. En vez de realizar una búsqueda completamente aleatoria, para cada modelo optimizamos sus hiperparámetros adaptando el proceso de *Simulated Annealing* [26] para este caso en particular.

Para esto, partimos de alguna configuración como semilla, y en cada paso del algoritmo nos movemos hacia una configuración vecina con igual probabilidad. Luego, aceptamos dicha configuración tanto si mejora el AUC, como si lo empeora, pero en tal caso con probabilidad inversamente proporcional a la disminución (ver figura 3.15). Además, permitimos que la penalización por disminución sea más estricta a medida que se avanza en la búsqueda, permitiendo una mayor exploración del espacio en iteraciones tempranas.

A continuación vemos el pseudocódigo para el algoritmo que realiza la búsqueda en el espacio de hiperparámetros. La función “próximo” toma como input una especificación de los hiperparámetros del modelo, elige de manera aleatoria alguno de esos hiperparámetros, y luego elige nuevamente de manera aleatoria un valor de ese hiperparámetro adyacente al valor actual de ese (por ejemplo, si el hiperparámetro elegido fue la cantidad de capas de GNN y su valor dentro de la configuración actual es 2, entonces la nueva configuración, si es aceptada, podrá tener 1 o 3 capas).

Algorithm 1 Simulated Annealing para optimización de hiperparámetros

```

 $k \leftarrow 0$ 
while  $k < k_{\max}$  do
   $T \leftarrow 1 - \frac{k}{k_{\max}}$  ▷ Controla la estrictez al aceptar un paso
   $params \leftarrow \text{proximo}(params\_actuales)$ 
   $auc \leftarrow \text{calcular\_auc}(modelo, params)$ 
   $\Delta \leftarrow auc\_actual - auc$ 
   $aceptar \leftarrow \min(1, \exp(-\Delta/T))$ 
  if  $aceptar > \text{generar\_random}(0, 1)$  then
     $params\_actuales \leftarrow params$ 
     $auc\_actual \leftarrow auc$ 
  end if
   $k \leftarrow k + 1$ 
end while

```

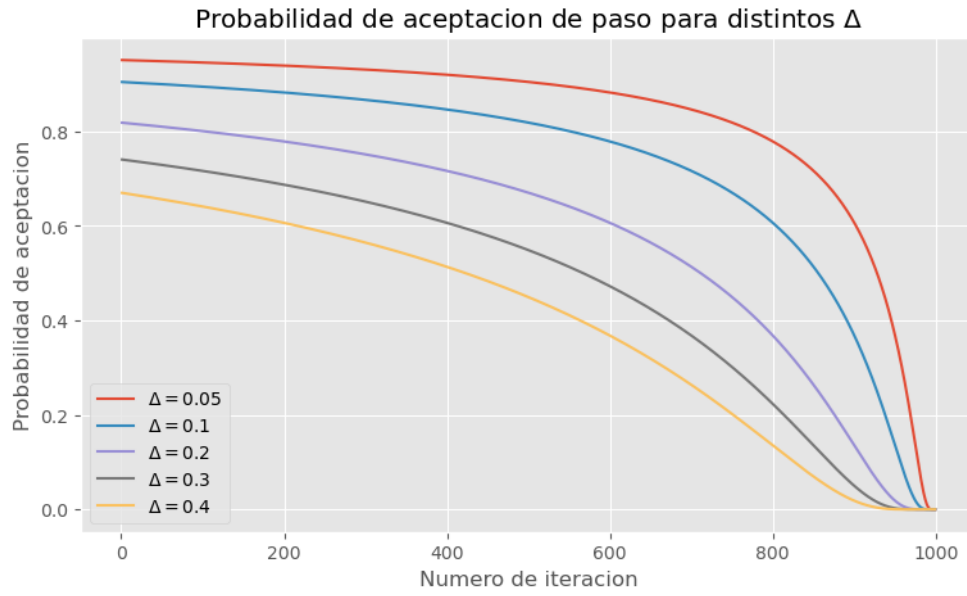


Fig. 3.15: *Probabilidad de aceptar un paso no beneficioso para distintos Δ con el pasar de las iteraciones.*

Finalmente, corrimos 1000 iteraciones de la caminata al azar para ambos modelos (SAGE y GAT).

4. RESULTADOS Y ANÁLISIS

4.1. Mejores modelos

A continuación detallamos las especificaciones de los dos mejores modelos (tabla 4.2) encontrados en las caminatas (uno para SAGE y otro para GAT). Ambos se entrenaron sobre 10 splits aleatorios y reportamos distintas métricas de performance promediadas sobre los 10 conjuntos de testeo, junto a su desvío estándar (tabla 4.1), y curvas de aprendizaje (figura 4.1). Comparamos contra una MLP de 4 capas.

Model	AUC-ROC	Accuracy	AUPRC	Precision	Recall
SAGE	0.951 ± 0.003	0.874 ± 0.005	0.952 ± 0.003	0.898 ± 0.016	0.845 ± 0.023
GAT	0.957 ± 0.003	0.890 ± 0.004	0.959 ± 0.003	0.895 ± 0.007	0.884 ± 0.006
MLP	0.576 ± 0.009	0.554 ± 0.007	0.575 ± 0.007	0.561 ± 0.009	0.503 ± 0.05

Tab. 4.1: Métricas de performance para los dos mejores modelos, promediadas sobre 10 splits aleatorios, junto con su desvío estándar.

Hiperparámetro	SAGE	GAT
Learning Rate	0.01	0.0001
Epochs	500	500
dimensión de features	128	64
Features	go2vec	go2vec
Capas de Pre-Process	0	0
Capas de Post-Process	0	1
Conectividad entre capas	False	cat
dimensión de la capa oculta	32	64
Batch Normalization	False	True
Dropout	0.0	0.3
agregación macro	sum	sum
Normalizacion L2	False	False
Capas de GNN	4	4

Tab. 4.2: Hiperparámetros generales de los dos mejores modelos

En cuanto a los hiperparámetros específicos de SAGE, la agregación micro resulto del tipo “máximo”, mientras que para GAT la cantidad de cabezas de atención fueron 2, las cuales se concatenan.

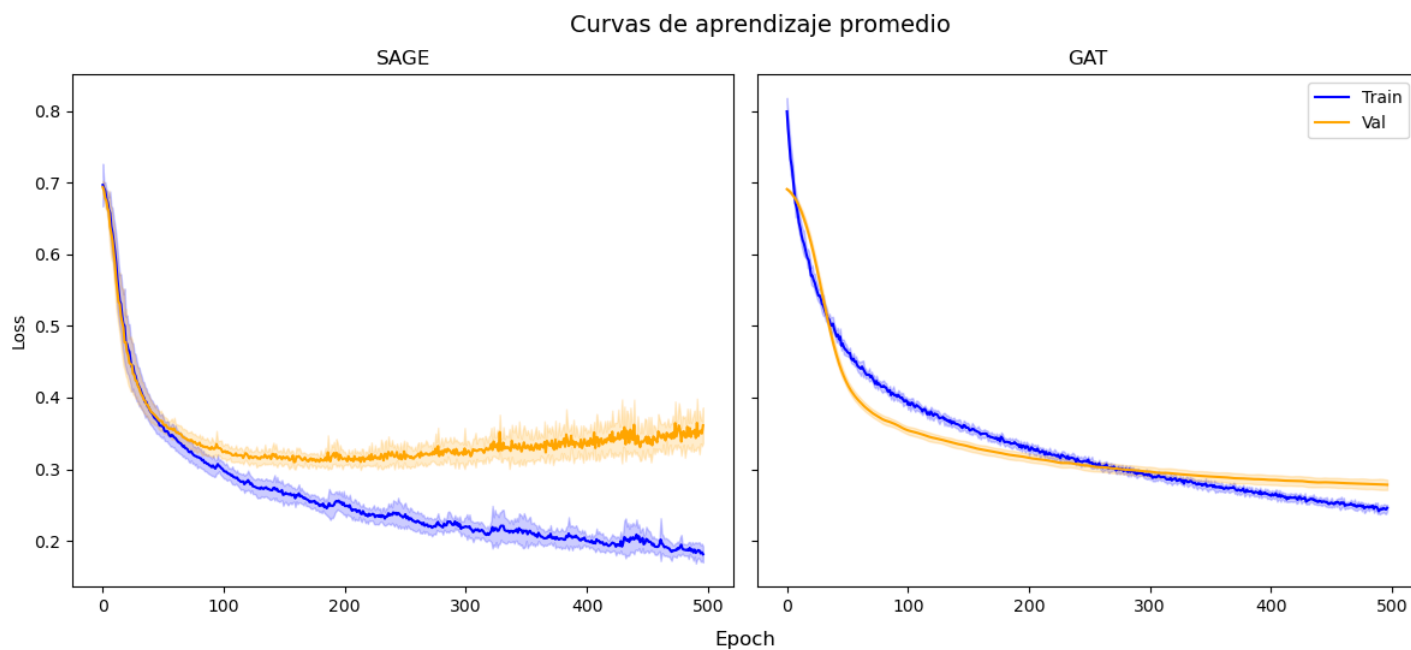


Fig. 4.1: *Curvas de aprendizaje promedio para ambos modelos. Se observa que GAT es muy robusto al dataset, y parece ajustarse menos al conjunto de entrenamiento.*

4.2. Impacto de los hiperparámetros

En esta sección analizamos el impacto de los hiperparámetros para ambas arquitecturas.

En la figura 4.2 vemos el resultado de las caminatas para ambos.

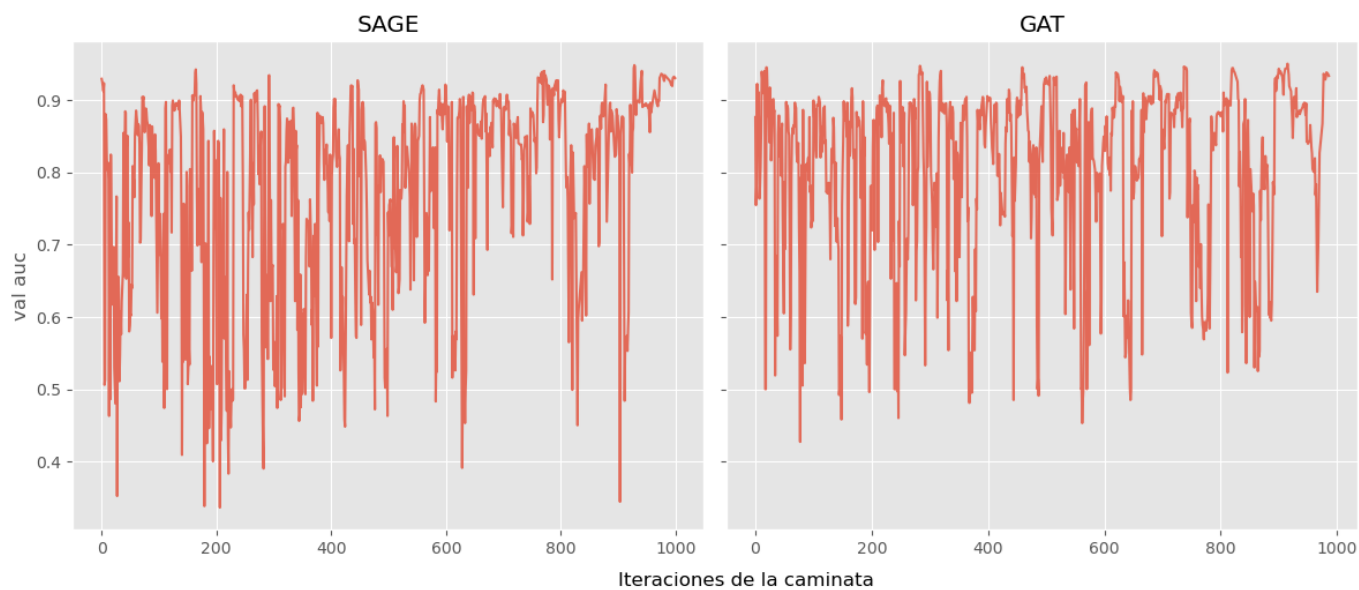


Fig. 4.2: *Evolución de performance sobre el conjunto de validación para los pasos de la caminata (únicamente pasos aceptados).*

Por cuestiones de eficiencia y recursos disponibles (una única GPU NVIDIA GeForce RTX 30390 24gb VRAM), el entrenamiento y la validación de los modelos se realizó sobre uno solo de los 10 splits.

En la figura 4.2 se puede notar que para ambos modelos la Evolución de AUC en el conjunto de validación no solo es errática si no que hay pasos donde la disminución de la performance es drástica, lo que implica la existencia de configuraciones que únicamente difieren en unos pocos hiperparámetros, pero que lo hacen en gran medida en cuanto a performance. Para ver en que medida se da esta sensibilidad a los hiperparámetros, vemos en la figura 4.3 diferencias de AUC entre todos los pares de configuraciones a una dada distancia, donde para cada par de configuraciones definimos su distancia como la cantidad de hiperparámetros en la que difieren.

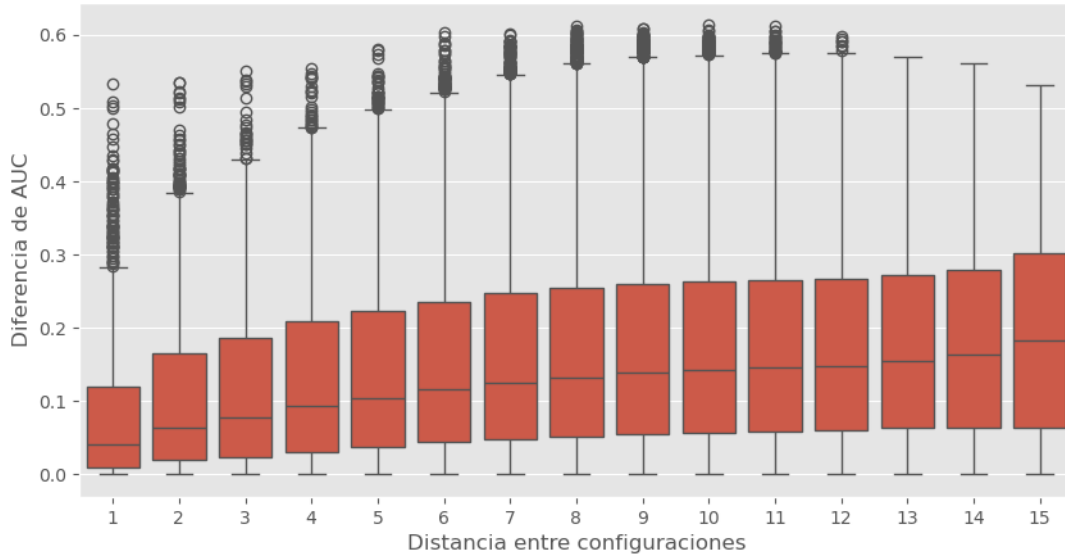


Fig. 4.3: *Diferencias de AUC entre pares de configuraciones de SAGE a una dada distancia. Observamos que, si bien hay una leve tendencia creciente, existen pares de configuraciones que difieren en un único hiperparámetro y aún así presentan diferencias de AUC mayores a 0.3.*

Explorar en detalle cuales son específicamente los hiperparámetros (y sus valores) que resultan en cambios drásticos de performance es complicado debido a efectos combinatorios que se dan entre estos. Para dar algunos ejemplos, si filtramos los pasos de la caminata para los cuales la diferencia en AUC entre un paso y el siguiente es mayor a 0.3 (recordemos que un paso y el siguiente solo difieren en el valor de un hiperparámetro), conseguimos pasos en los que se aplica o se deja de aplicar batch normalization, pasos en los que se pasa de 0 a 2 capas de pre-procesamiento (y viceversa), pasos en los que la agregación micro pasa de “suma” a “máximo” (y viceversa), etc. Esto dificulta aislar de manera puntual el impacto de los hiperparámetros (es decir, atribuirle cierto efecto al mirar el cambio entre pasos consecutivos), por lo que nos contentamos con ver diferencias en distribución.

Una diferencia importante en cuanto a las caminatas de ambos modelos es la cantidad de configuraciones las cuales utilizan las features de go2vec conseguidas para un subconjunto de los genes (versus inicializar todas las features de manera aleatoria). En ese sentido, la caminata de SAGE esta más balanceada, con 551 de 1000 instancias que utilizan las features, en contraparte con la caminata de de GAT para la cual 919 las utilizan. Para analizar la diferencia entre las instancias que usan las features y las que no, vemos en la figura 4.4 un violin-plot (similar a un boxplot, donde la anchura es proporcional a la cantidad de instancias) de la performance en los resultados para SAGE según las features usadas.

Para todo lo que sigue del trabajo, realizaremos los análisis y experimentos correspondientes en modelos (ya sea el mejor o no) de SAGE. Los resultados para GAT son similares.

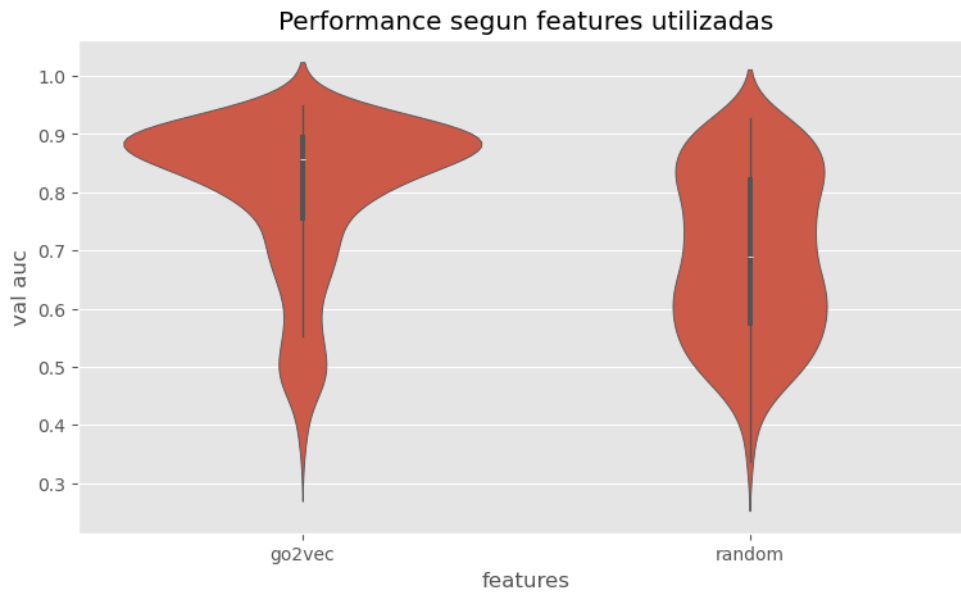


Fig. 4.4: *Diferencias en performance de los modelos según el tipo de features utilizadas. Se ve un claro aumento en performance al utilizar las features conseguidas para los genes.*

Teniendo en cuenta esta diferencia en performance según las features utilizadas, distinguimos entre las instancias según el tipo de feature utilizado al ver las distribuciones de performance por hiperparámetro. A continuación vemos las distribuciones para un conjunto de hiperparámetros que exhiben diferencias respecto a sus valores.

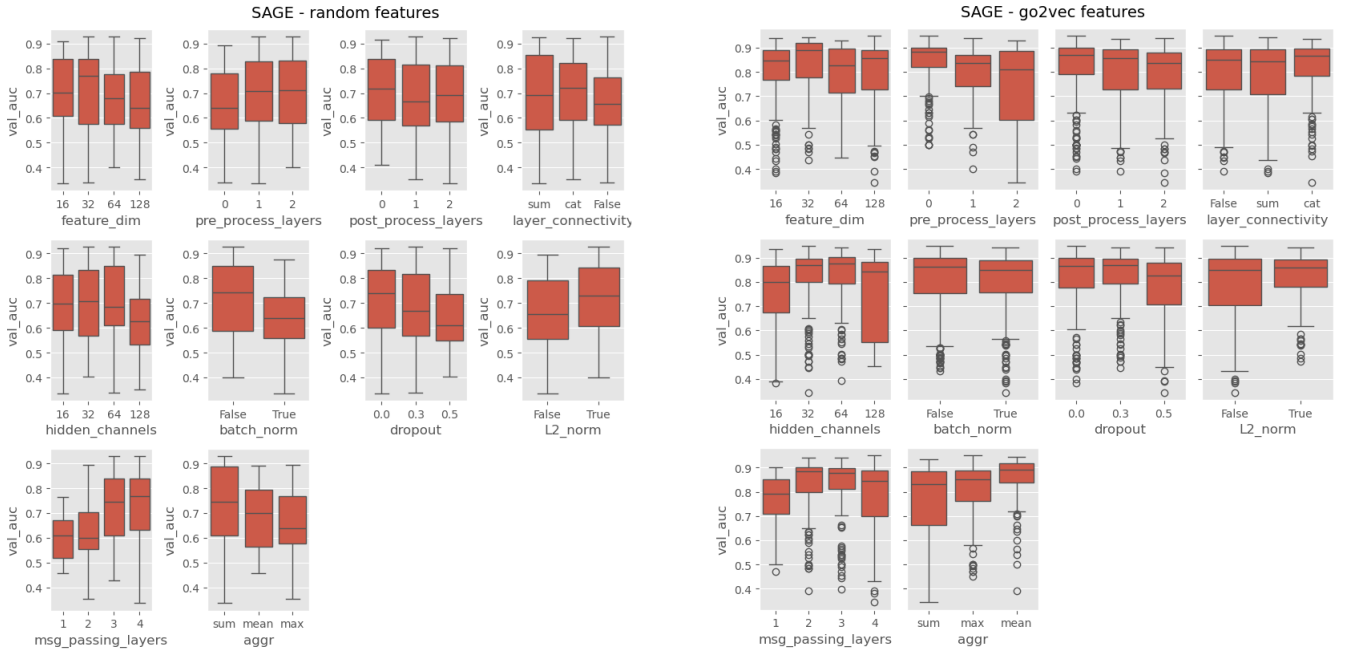


Fig. 4.5: *Distribuciones de performance según hiperparámetros para SAGE diferenciando según el tipo de feature inicial utilizada.*

En la figura 4.5 se puede observar que para las configuraciones con features random, utilizar más capas de pre-procesamiento parece ser necesario para obtener una buena performance, mientras que para las que utilizan las features de go2vec se ve exactamente lo contrario. Otra diferencia se da en la cantidad de capas de pasada de mensajes, donde para las primeras se observa la necesidad de más capas para obtener mejor performance, mientras que para las segundas la mejor performance se da al utilizar entre 2 y 3 capas. La agregación micro es otro hiperparámetro donde las tendencias se dan de forma invertida: para las configuraciones random, utilizar la suma es superior a tomar el promedio o el máximo, mientras que para las configuraciones de go2vec utilizar el promedio supera a tomar máximo o usar la suma.

La presencia de diferencias significativas dentro de un mismo modelo, tanto en términos absolutos de performance como en las tendencias dentro de los valores de los hiperparámetros respecto a esta, nos indica que el impacto de la utilización de las features conseguidas mediante go2vec requiere un análisis más profundo para poder comprenderlo.

4.3. Efecto de la incorporación de features externas

Para poder analizar de forma justa la manera en la que la incorporación de features externas para algunos genes mejora la performance del modelo, tomamos una caminata de 1000 pasos para SAGE en la que únicamente utilizamos features random. Luego, para cada una de las mejores 400 configuraciones dentro de esa caminata, entrenamos esos modelos únicamente reemplazando las features aleatorias por las features conseguidas. De esta forma, para cada modelo contamos con un modelo idéntico que solo difiere en las features iniciales, permitiendo una comparación justa.

En la figura 4.6 vemos la performance de todos los modelos. Para esto, ordenamos las

400 configuraciones iniciales de menor a mayor valor de AUC (siempre en el conjunto de validación) y sobre cada una graficamos el AUC correspondiente a su modelo equivalente con features externas.

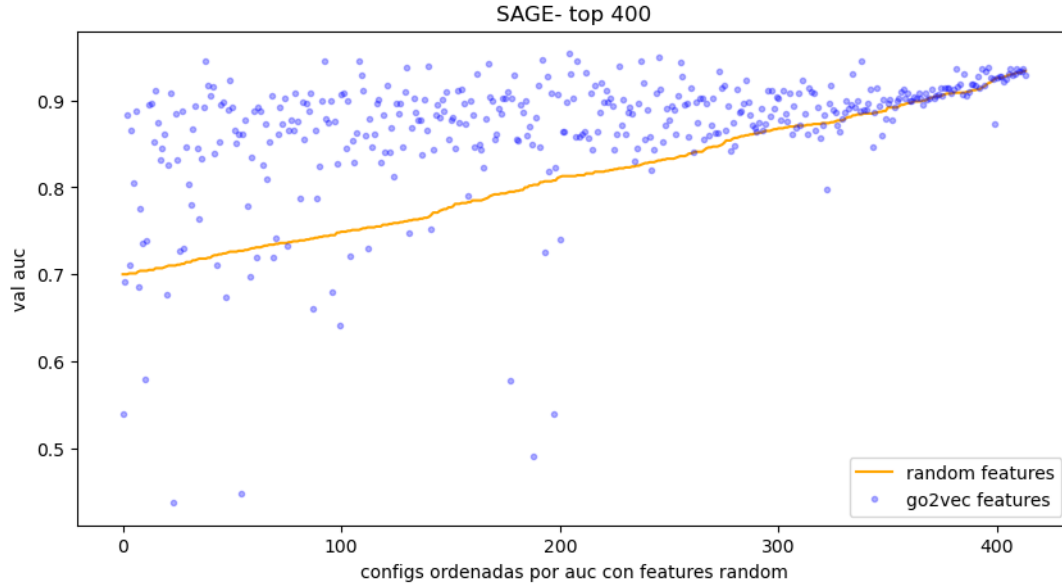


Fig. 4.6: Comparación en la performance para las configuraciones.

Se observa de manera clara que para la gran mayoría de las configuraciones, el uso de las features conseguidas mejora la performance. Notemos que por lo general, esta mejoría se da en el rango de valores de 0.85 a ≈ 0.92 , y se va concentrando a medida que la performance de la configuración correspondiente a las features random aumenta.

Para entrar en detalle, seleccionamos una configuración donde el AUC se incremento de 0.7 a 0.9 al incorporar features. La configuración fue elegida de forma que no contenga capas de pre-procesamiento para que el modelo reciba las features sin este pre-procesamiento, que de hacerlo agregaría complejidad al análisis.

En particular, comparamos tres modelos para la configuración mencionada; uno entrenado utilizando únicamente features aleatorias, otro entrenado con la incorporación de las features conseguidas para los genes, y por último un modelo entrenado con la incorporación de las features pero asignándoselas a nodos de manera aleatoria, los cuales denominaremos *modelo R*, *modelo F*, y *modelo FR₀* (por features randomizadas nulo), respectivamente. El resultado más sorprendente de los análisis realizados tiene que ver con la especificidad de las fetures a nivel nodo. Notamos que la performance en el modelo FR sigue siendo considerablemente mejor que en el modelo R, y presenta casi nula diferencia con el modelo F. Es decir, parece ser que la incorporación de las features al modelo mejora la performance de este, pero la información agregada para que se de dicha mejora no proviene de la especificidad de las features, que cabe recordar que provienen de embeddings del GOA generados mediante node2vec como se introdujo en la sección de [Gene Ontology](#).

Esta observación se da después de realizar 10 corridas independientes de los 3 modelos para la configuración mencionada. Es decir, para el modelo FR, en cada una de las 10 repeticiones se renuevan al azar los genes a los cuales se les asignan las features. Se obtuvieron valores medios de AUC de 0.686 ± 0.11 , 0.912 ± 0.002 y 0.905 ± 0.005 respectivamente. En la figura 4.7 se muestran las curvas de aprendizaje para los tres modelos.

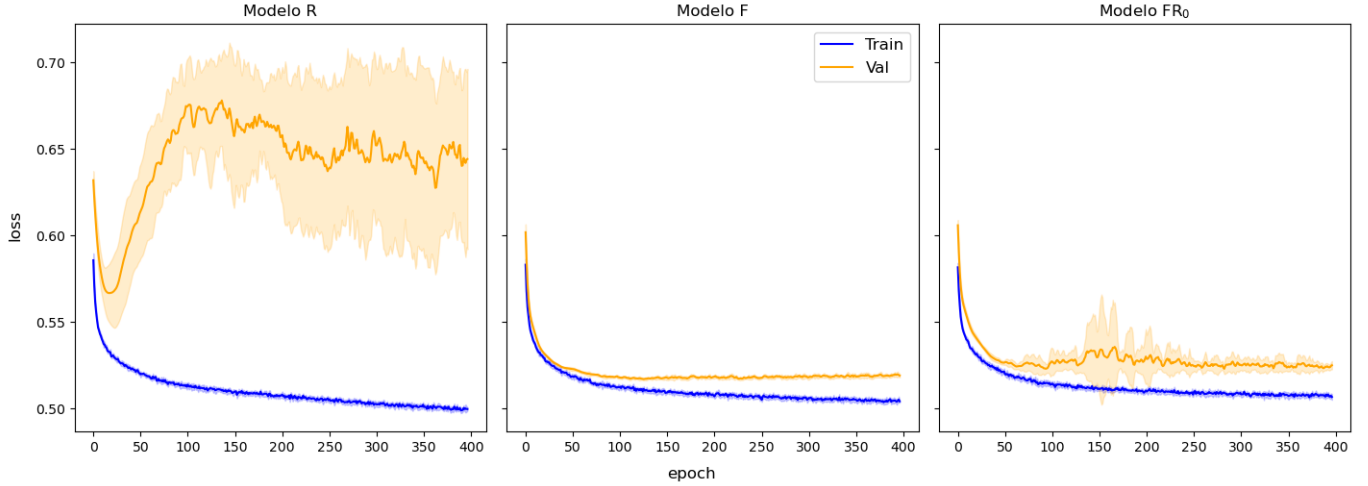


Fig. 4.7: Curvas de aprendizaje para los tres modelos. Los modelos R son los que mayor varianza presentan en las pérdidas de validación, seguidos por los modelos FR_0

Descartamos que este efecto se de por redundancia de las features al analizar las distancias coseno entre ellas, que están en su mayoría lejos de 0 (donde la distancia coseno entre dos vectores es $1 - \cos(v_i, v_j)$, ver figura 6.2 del apéndice). más aún, lo que “aprenden” el modelo F y el modelo FR_0 es significativamente distinto. Esto se desprende al observar, para cada nodo en ambos modelos, sus 10 vecinos más cercanos en el espacio de embeddings y ver la intersección entre ambos vecindarios, donde para la gran mayoría de los nodos esta intersección es muy baja (ver figura 6.3 del apéndice).

Por otro lado, vemos de dos formas distintas que las features si codifican información relevante a cerca de la estructura de la red armada, y asignar esas features a genes de manera aleatoria destruye esa información. Primero, analizamos como correlacionan con la conectividad dentro de la capa genómica, que si bien mostramos que reflejaban la similitud en dicha capa, esto lo hacían mediante los embeddings de los términos del Gene Ontology a los que estaban asociados. Para ver esto, de todos los enlaces en la capa genómica, filtramos aquellos en los que ambas puntas del enlace cuentan con features. Luego, generamos esa misma cantidad de enlaces negativos (donde ambos vértices del enlace negativo también cuentan con features), y graficamos las distribuciones de distancias coseno por enlace para ambos conjuntos, realizando el mismo procedimiento pero permutando de forma aleatoria las features dentro de los nodos seleccionados. Esto se ve en la figura 4.8.

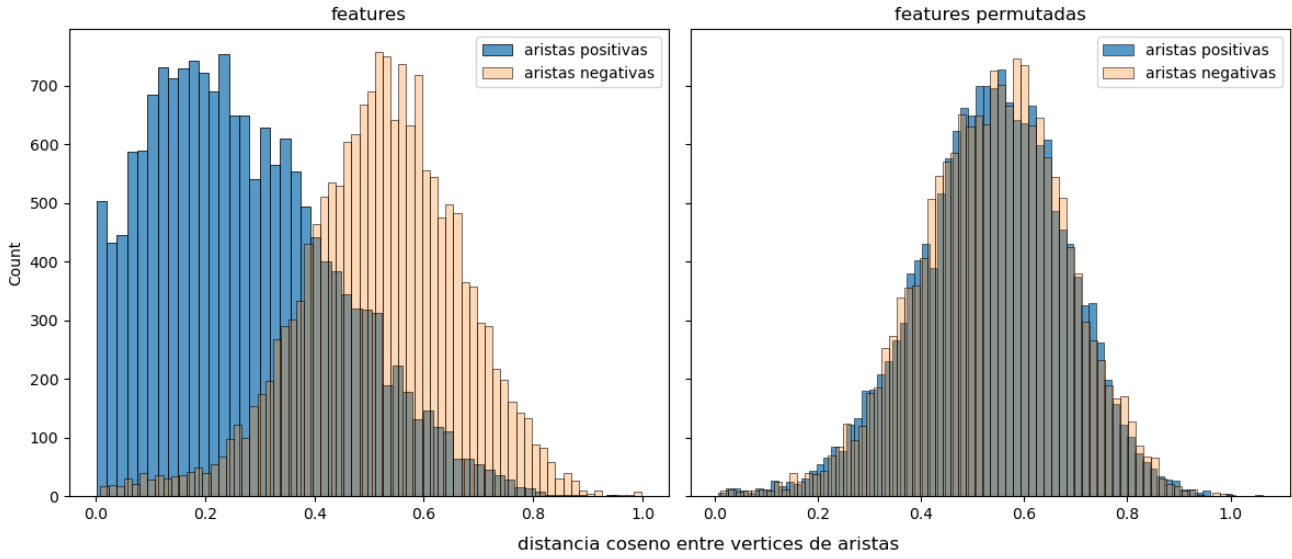


Fig. 4.8: Distribución de distancias coseno entre puntas de enlaces. Notemos la diferencia tanto en magnitud como en distribución de las distancias para las aristas positivas con sus features correspondientes con las distancias para aristas negativas y aristas con sus features permutadas.

Realizamos también un análisis similar al anterior pero poniendo énfasis en que el objetivo del modelo es la predicción de DTIs. Para esto, agrupamos para cada droga los genes a los que esta asociada (es decir, su vecindario en la capa DTI). Luego, calculamos para antes y después del entrenamiento la distancia (coseno) promedio en los vecindarios. Intuitivamente, dado que el objetivo del modelo es predecir DTIs, los genes conectados a una misma droga deberían agruparse cerca después del entrenamiento del modelo. En la figura 4.9 graficamos esto para los tres modelos.

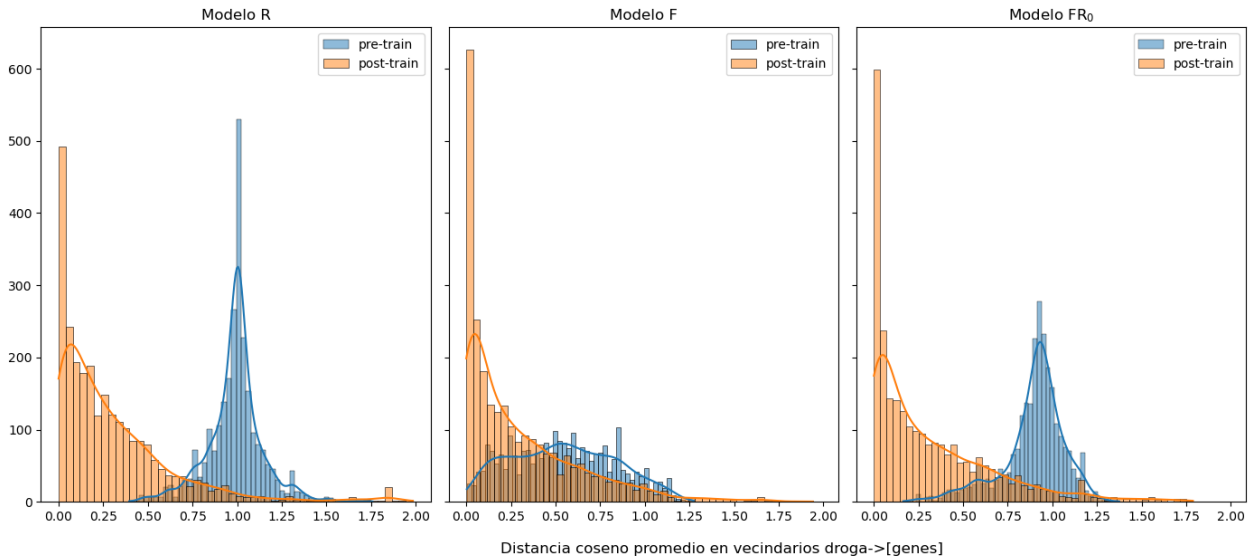


Fig. 4.9: Distancias medias en vecindarios droga→gen.

De la figura 4.9 se pueden desprender dos cosas; por un lado, vemos que al considerar los genes con sus features correspondientes (gráfico para el modelo F, en azul), estas muestran una distancia considerablemente menor a lo que se esperaría si fueran aleatorias (el pico en 1 para las features aleatorias se debe a un fenómeno de concentración de la medida en altas dimensiones, donde para dos vectores $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$, $\lim_{d \rightarrow \infty} \mathbb{P}(\mathbf{x} \cdot \mathbf{y} = 0) = 1$). Además, comparando los gráficos para el modelo F y el modelo FR_0 , podemos concluir que genes asociados a una misma droga comparten features similares, y asignar las features de manera aleatoria destruye esta información dentro de esos vecindarios. De todas formas, esto tampoco logra afectar la performance para este último modelo.

Por último, realizamos un análisis de ablación de las features, tanto en el modelo donde estas se corresponden con sus genes, como en el modelo donde fueron asignadas de manera aleatoria. En este sentido, quitamos paulatinamente y de manera creciente porciones de estas, y reportamos el AUC para cada porcentaje quitado, lo que se observa en la figura 4.10.

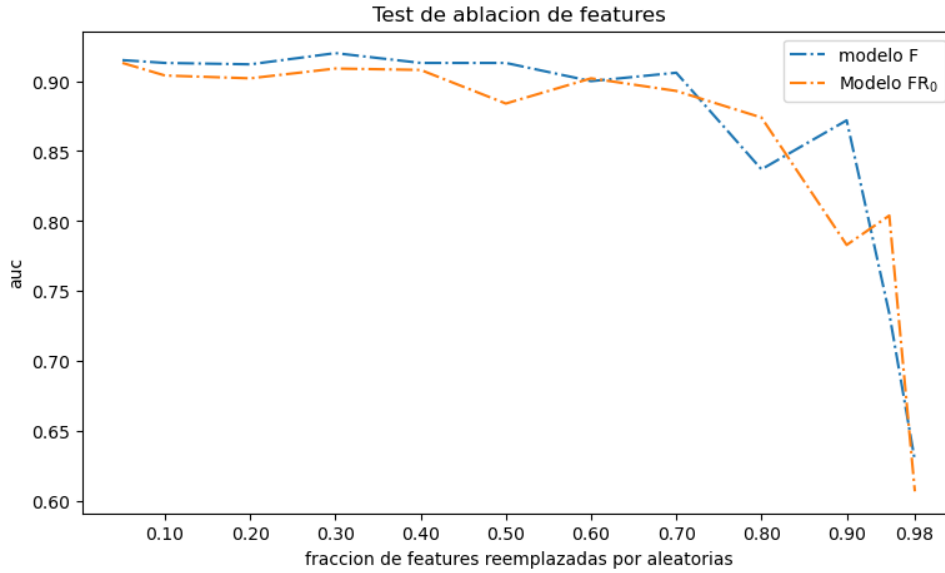


Fig. 4.10: *Test de ablación de features.*

En la figura 4.10 podemos observar tanto para el modelo F y el modelo FR_0 , parece resultar equivalente en términos de performance utilizar el 100 % de las features o el 30 %, a partir de donde recién se ven caídas importantes en términos de performance.

En resumen, mostramos que la inclusión de las features conseguidas mejoran de manera sistemática la performance de los modelos, pero que la razón de la mejora no parece deberse a la especificidad de estas al nivel de los genes a las que corresponden. Además, descartamos que esto se deba a redundancia entre las features y vimos que la similitud entre estas se encuentra correlacionada con la estructura de la red tanto a nivel de la capa genómica como a nivel de la capa de DTIs. Por último, notamos que la performance al incluir las features se mantiene estable aun cuando reemplazamos hasta el 70 % de estas por features aleatorias. Esto, en conjunto al análisis realizado de la red donde mostramos que esta presenta alta conectividad promedio (i.e, las distancias entre cualquier par de

nodos dentro de la red son cortas), nos da la pauta para pensar que el modelo utiliza esta información de modo distribucional y logra integrarla rápidamente en pocas iteraciones de pasaje de mensajes. Dejamos como trabajo a futuro formalizar esta hipótesis en términos de la conectividad dentro de la red para poder testearla e intentar dilucidar este fenómeno.

4.4. Sesgos por conectividad

En esta sección analizamos sesgos del modelo a la hora de realizar las predicciones en base a la estructura del grafo. Como vimos en el [Análisis de la red](#), la distribución de grado en la capa DTI se asemeja a una distribución libre de escala, donde muchos nodos cuentan con pocos vecinos, y pocos nodos cuentan con una gran cantidad de estos. Como lo mencionamos, esto puede ser beneficioso para el modelo en el sentido que permite integrar información global de la red a la hora de realizar el pasaje de mensajes (que es una operación de vecindarios locales), pero a su vez puede traer consigo algunos problemas. Por ejemplo, los nodos de grado alto estarán sobre representados en el modelo, en el sentido de que para el entrenamiento este contara con muchas más instancias de la clase positiva para aristas de las que este nodo es parte en comparación con nodos de grado bajo, facilitando en alguna forma las predicciones para las que el nodo de alto grado forma parte (que también estarán sobre representadas a nivel del nodo en el conjunto de test). A continuación vemos, para cada enlace en el conjunto de test, su grado combinado en la capa DTI (i.e, la suma de los grados de los nodos que componen el enlace en esa capa), junto al score con el que lo puntuó el mejor modelo conseguido para SAGE.

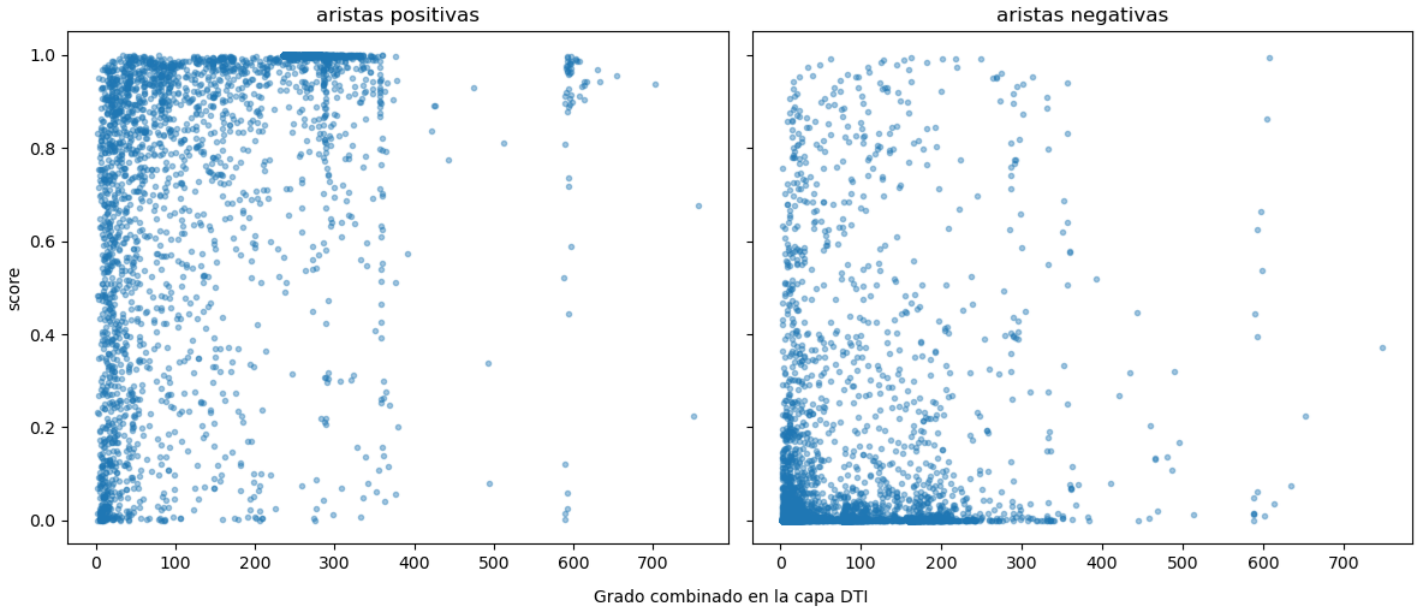


Fig. 4.11: *Score por enlace según su grado combinado en la capa DTI para aristas positivas y negativas. Se excluyeron algunas aristas de grado combinado muy alto por temas de escala.*

En la figura 4.11 podemos notar que se da un fenómeno cercano al sesgo que mencionábamos. Es decir, mirando a las aristas positivas, aristas de grado bajo combinado

presentan scores muy variados, mientras que a medida que crece el grado de las aristas los scores se concentran más cerca de 1. Lo contrario se ve para las aristas negativas.

Esto no es algo malo per se, dado que son sesgos implícitos de los datos más que de el modelo, pero podría ser indicio de ciertos problemas a la hora de realizar evaluaciones nodo a nodo.

Recordemos que la principal métrica utilizada en este trabajo es el AUC, dado que estamos pensando al problema de predicción de aristas como uno de clasificación binaria. Es conveniente pensar para lo siguiente al AUC como una métrica donde lo único que importa es el ordenamiento relativo de los rankings, no el valor absoluto de los scores. Es decir, mientras más alto puntuemos en general las instancias positivas por sobre las negativas, mayor el AUC, independientemente del valor absoluto de estos puntajes. Ahora bien, que nuestros mejores modelos alcancen valores altos de AUC quiere decir que por lo general, la mayoría de las aristas positivas están siendo puntuadas más alto que las aristas negativas, pero esto no nos dice mucho en general sobre como se están dando estos puntajes nodo a nodo.

Pensando al problema de la predicción de DTIs no como uno de clasificación binaria global, si no como uno de recomendación local (es decir, para una dada droga/target quiero recomendarle posibles targets/drogas con las que puede interactuar), se podría de alguna forma relajar el requerimiento de puntuar las interacciones positivas más altas que las negativas de forma global, y únicamente esperar que para cada droga/target se puntúe más alto sus interacciones positivas que sus interacciones negativas.

Este enfoque proveniente del área de los sistemas de recomendación parece alinearse más con las posibles aplicaciones que podría tener la predicción de DTIs. Por ejemplo, una farmacéutica podría estar interesada en el tratamiento de una enfermedad causada por cierto gen, para lo que querrían buscar drogas que puedan atacar la proteína codificada por el gen. Trasladando esto a nuestro trabajo, uno esperaría que para el gen en cuestión, las predicciones positivas de este puntúen más alto que las negativas, independientemente de si sus interacciones positivas puntúan más bajo que interacciones negativas correspondientes a genes diferentes.

Esto se puede formalizar utilizando una métrica denominada *Precision@K*: Dado un K , para cada nodo ordenamos las predicciones en las que esta involucrado según su score y contemplamos únicamente las primeras K . Luego, si de esas K , m son interacciones positivas, su *Precision@K* sera de $\frac{m}{K}$.

Realizamos una extensión al set de testeo y consideramos una métrica levemente modificada para resolver algunos problemas de esta al aplicarlas a nuestra tarea. Por ejemplo, un dado nodo no cuenta con interacciones negativas, entonces para cualquier K su *Precision@K* sera 1. Ademas, la elección de K impone demasiada restricción dependiendo de la cantidad total de interacciones para un nodo dado. Si esta cantidad es p , y tenemos que $K \geq p$, entonces su *Precision@K* estará acotada por su cantidad de interacciones positivas. Contemplando esto, balanceamos el set de testeo agregando nuevas aristas negativas de forma tal que para cada nodo con más aristas positivas que negativas ahora tenga la misma cantidad de ambas. Luego, dado K , definimos para cada nodo $K' = \min(K, m)$, donde m es la cantidad de interacciones positivas de ese nodo, y calculamos para cada nodo su *Precision@K'*. Con esta métrica, que un nodo tenga *Precision@K'* = 1 quiere decir que el 100 % de las aristas positivas en las que está involucrado fueron puntuadas más alto que sus negativas.

Con esto en mente, vemos a continuación la $Precision@K'$ promediada sobre todos los nodos en el set de testeo para distintos valores de K .

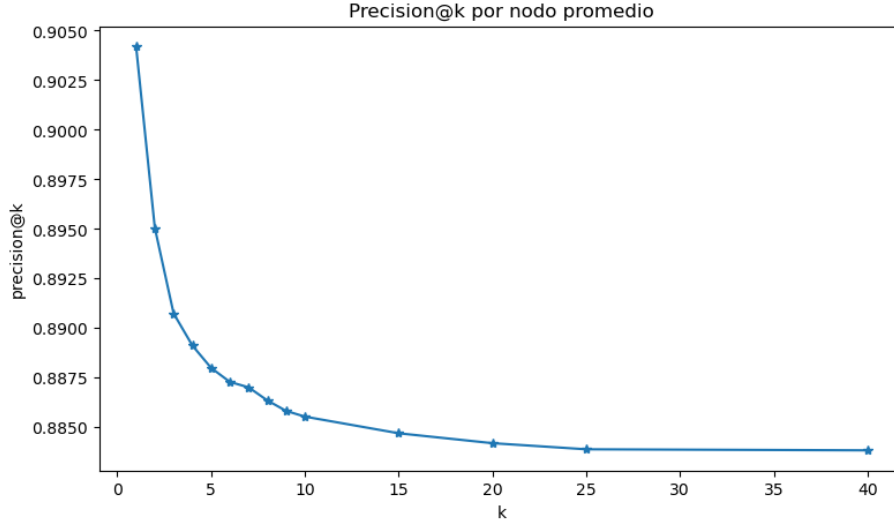


Fig. 4.12: $Precision@K$ promedio sobre los nodos de testeo para distintos K . Notemos que a medida que se agranda k , los únicos nodos cuya $Precision@K$ cambia son para los cuales $K < m$, estabilizándose en ~ 0.88 .

En la figura 4.12 se puede observar que en promedio, un dado nodo puntúa más del 88 % de sus predicciones positivas más alto que las negativas. Si bien en promedio esto es bueno, nuevamente tenemos el problema de que hay cierto sesgo en esta métrica según el grado de los nodos. Tomamos $K' = 5$ para ejemplificarlo, como se observa en la figura 4.13.

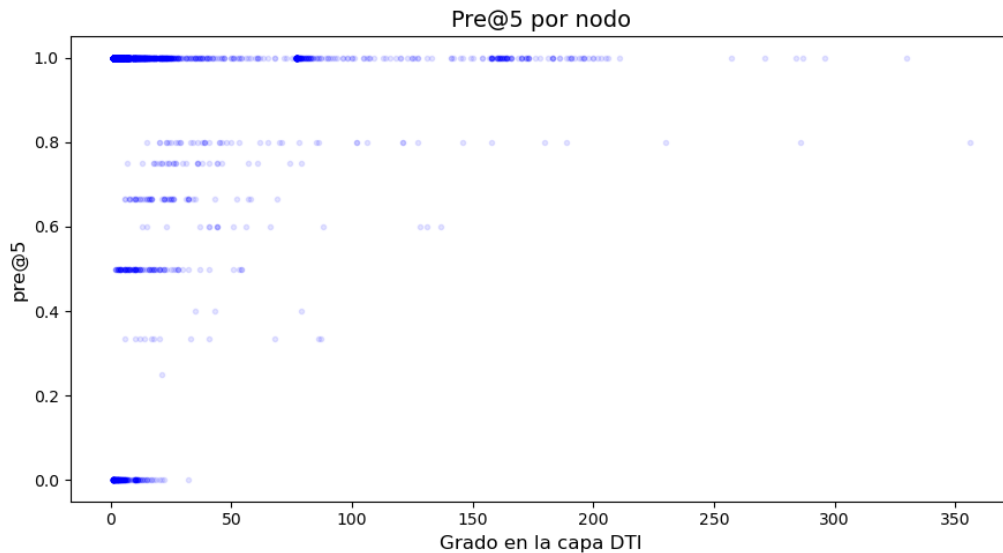


Fig. 4.13: $Precision@5$ por nodo, ordenados según su grado en la capa DTI.

Realizamos una discusión en la sección de [Mejoras y Trabajos Futuros](#) sobre una posible

solución a este fenómeno.

5. CONCLUSIONES

Durante el desarrollo del trabajo presentamos una metodología para realizar predicciones de DTIs a partir de una GNN montada sobre una red heterogénea de la cual las DTI formaban parte.

La construcción de la red, donde partiendo de DTIs relevadas desde bases de datos públicas las integramos en una red heterogénea de dos capas (una para genes y otra para drogas), presento ciertas dificultades. La construcción de las conexiones intra-capas se realizó teniendo en mente el tipo de relación a predecir, por lo que decidimos reflejar en estas conexiones similitud estructural entre los genes y entre las drogas, respectivamente. Un problema fue que la similitud estructural no es una métrica bien definida, por lo que quedo en nosotros definir métricas tanto para drogas como para genes. Por otro lado, una vez definidas las métricas estas fueron calculadas entre todos los pares, por lo que se tuvo que tener cuidado en el criterio de corte para definir los enlaces.

Luego, implementamos una GNN sobre la red armada, consiguiendo buenas métricas de performance para la predicción de DTIs. Esto aporta evidencia a la hipótesis original del trabajo, en donde afirmamos que la topología de la red conseguida codificaba en ella la información necesaria para poder realizar las predicciones. Además, conseguimos features externas a la topología del grafo para algunos genes a partir de anotaciones funcionales de sus proteínas, y mostramos que la incorporación de estas features mejoraba de manera sistemática la performance del modelo.

Por último, al analizar en profundidad los resultados obtenidos, notamos algunas cosas importantes.

Durante el análisis de las caminatas al azar mediante las que optimizamos los hiperparámetros, vimos que estos modelos presentan alta sensibilidad respecto a estos, y que el tamaño del espacio de diseño junto con efectos combinatorios que se dan entre los hiperparámetros impone ciertas restricciones a la hora de analizar que cambios puntuales en estos resultan en cambios sistemáticos de la performance.

Al analizar el efecto de la incorporación de features a los modelos, notamos que la manera en la que estos integraban esta información no dependía en buena medida de la especificidad de los features (i.e, la correspondencia entre un gen y el feature que lo representa), algo que nos resultó muy extraño. Esto sucede aún cuando mostramos que las features (y su especificidad) codifican información relevante sobre la estructura de la red. Además, realizamos un estudio de ablación sobre las features y mostramos también que el modelo únicamente necesita una porción muy reducida de estas para realizar mejoras en la performance.

Finalmente, analizamos las predicciones en base a la conectividad de la red y mostramos que existe un sesgo hacia el grado, donde el modelo se vuelve más confiado a la hora de realizar predicciones sobre una arista a medida que el grado combinado en la capa DTI de los nodos que la componen aumenta.

5.1. Mejoras y Trabajos Futuros

Dedicamos esta sección a discusiones sobre posibles mejoras y trabajos futuros que se desprendieron durante el desarrollo de esta tesis.

Desde el punto de vista del armado y análisis de la red, estos se podrían completar con dos cosas. Por un lado, prestar especial atención al tipo de fingerprint utilizado a la hora de calcular distancias a pares entre las drogas. Analizar con más profundidad la razón de la distribución de scores vista en la figura 3.3 es necesario para poder hacer una elección más segura del fingerprint. Se podría contemplar también, siguiendo el espíritu de este trabajo, *aprender* las representaciones de las drogas con una GNN al hacer uso de sus grafos moleculares, lo que dejaría de lado el uso de fingerprints (más aún, la arquitectura de una GNN se puede pensar como la forma diferenciable del algoritmo que obtiene fingerprints circulares).

Por otro lado, para completar el análisis de la red y de alguna forma “garantizar” que las redes obtenidas tienen sentido, se podría realizar un análisis de las comunidades de estas y asegurarse que dentro de estas se respetan propiedades fisicoquímicas de las moléculas que las componen, o propiedades estructurales de las proteínas, respectivamente.

En cuanto a la implementación, existen modelos de GNNs más novedosos específicamente diseñados para la tarea de predicción de aristas [27], por lo que probar implementarlos podría llegar a traer mejoras generales.

Respecto a los análisis realizados, esbozar y testear una hipótesis sobre como estos modelos usan la información externa de las features incorporadas seria un gran avance para esclarecer el fenómeno mencionado. Algo que intentamos fue analizar la evolución de los modelos capa a capa, para lograr entender el flujo de la información inicial a través del modelo. Sin embargo, notamos que el modelo “aprende” poco durante las representaciones intermedias, en el sentido de que si midiéramos la performance en cualquiera de las capas previas a la ultima, esta no solo seria mala si no que seria similar a lo largo de todas estas (ver figura 6.4 del apéndice). Para esclarecer esto, podríamos incorporar una técnica denominada *deep supervision*, donde tenemos un decoder por cada capa en vez de únicamente después de la ultima. Luego, la pérdida de entrenamiento tendría en cuenta las representaciones intermedias del modelo, lo que podría llegar a solucionar este problema. En cuanto a los sesgos por conectividad mencionados, una posible solución al problema es la siguiente:

Recordemos que el AUC, la principal métrica de evaluación utilizada, nos habla de cuan bien podemos ordenar de forma global las predicciones del modelo. Es decir, un buen valor de AUC sera aquel en el que la mayoría de las predicciones positivas puntúen más alto que las negativas. Esto se alinea con la función de pérdida utilizada (entropía cruzada binaria), que se define como:

$$\mathcal{L} = - \sum_{(u,v) \in \mathcal{D}} A_{uv} \log(\sigma(z_u \cdot z_v)) + (1 - A_{uv}) \log(1 - \sigma(z_u \cdot z_v))$$

Donde \mathcal{D} es algún conjunto de pares de nodos. Es decir, \mathcal{L} se minimiza empujando hacia arriba el puntaje de las aristas positivas (donde $A_{uv} = 1$) respecto a las negativas ($A_{uv} = 0$), sin tener en cuenta si una dada arista negativa comparte un extremo con una positiva.

Esto puede suponer un problema, como mostramos a continuación:

Supongamos que tenemos tres nodos A, B, C donde A se conecta con B , B con C , y $A - -C$ es una arista negativa del modelo. Desde el punto de vista de la pérdida mencionada, el modelo intentara puntuar $A - B$ y $B - C$, más alto que $A - C$, pero esto es de alguna manera incompatible, dado que tanto A y B como B y C necesitaran estar cerca en el espacio de embeddings, pero A y C lejos, por lo que alguna de las predicciones se vera perjudicada. Intuitivamente, si A es un nodo de alto grado mientras B y C son de bajo grado, al modelo le sera más fácil puntuar alto $A - B$ y bajo la arista negativa $A - -C$, por lo que o $B - C$ tendrá puntaje bajo.

Ahora bien, si volvemos a la métrica de evaluación centrada en los nodos ($Precision@K'$), esto no tiene porque ser así. Es decir, para conseguir un buen valor de dicha métrica, $A - B$ y $B - C$ necesitar puntuar más alto que $A - -C$, pero de manera local, sin considerar las demás predicciones del modelo. Notemos entonces que alcanza con que la distancia entre A y B sea mayor a la distancia entre B y C independientemente del valor de esta distancia. Una función de pérdida que actúa como sustituta para la $Precision@K'$ (que no es diferenciable), es la *Bayesian Personalized Ranking*. Si para cada nodo u^* del modelo, consideramos sus interacciones positivas $E(u^*) = \{v \in V : (u^*, v) \in E\}$ y algún conjunto de interacciones negativas $E^-(u^*) = \{v \in V : (u^*, v) \notin E\}$, la pérdida se define como:

$$\mathcal{L}(u^*) = - \sum_{(u^*, v_{pos}) \in E(u^*)} \sum_{(u^*, v_{neg}) \in E^-(u^*)} \log(\sigma(z_{u^*} \cdot z_{v_{pos}} - z_{u^*} \cdot z_{v_{neg}}))$$

Y la pérdida global resulta del promedio de las perdidas por cada nodo. Ahora, el modelo aprende a optimizar localmente para cada nodo u^* los parámetros de forma que las interacciones positivas de este puntúen más alto que sus negativas (para lo cual $(z_{u^*} \cdot z_{v_{pos}} - z_{u^*} \cdot z_{v_{neg}}) > 0$), ignorando de cierta manera el puntaje de las demás predicciones del modelo.

Queda a futuro implementar los modelos con esta función de pérdida para ver si el sesgo por conectividad mejora.

Finalmente, seria ideal realizar algunos casos de estudio para las predicciones novedosas del modelo (i.e, pares de nodos cuyo score es alto y no son aristas positivas ni negativas del modelo), para entender si los resultados tienen sentido bioquímico (realizar esto con algún experto en el área seria lo mejor).

5.2. Disponibilidad de los datos

La implementación de la metodología del trabajo se encuentra en este [repositorio](#), aunque a día de hoy no esta estructurado con la interpretabilidad y reproducibilidad fácil en mente.

6. APÉNDICE

Figuras del apéndice

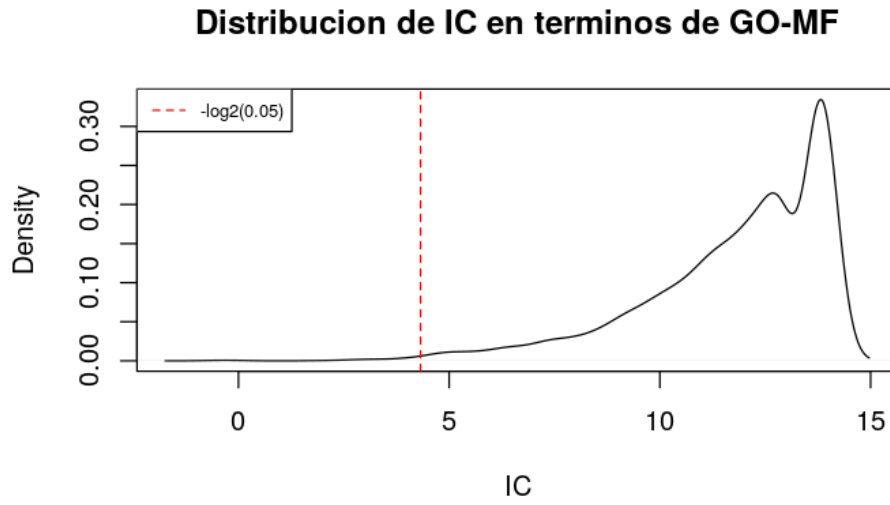


Fig. 6.1: *distribución de IC para los términos de GO.*

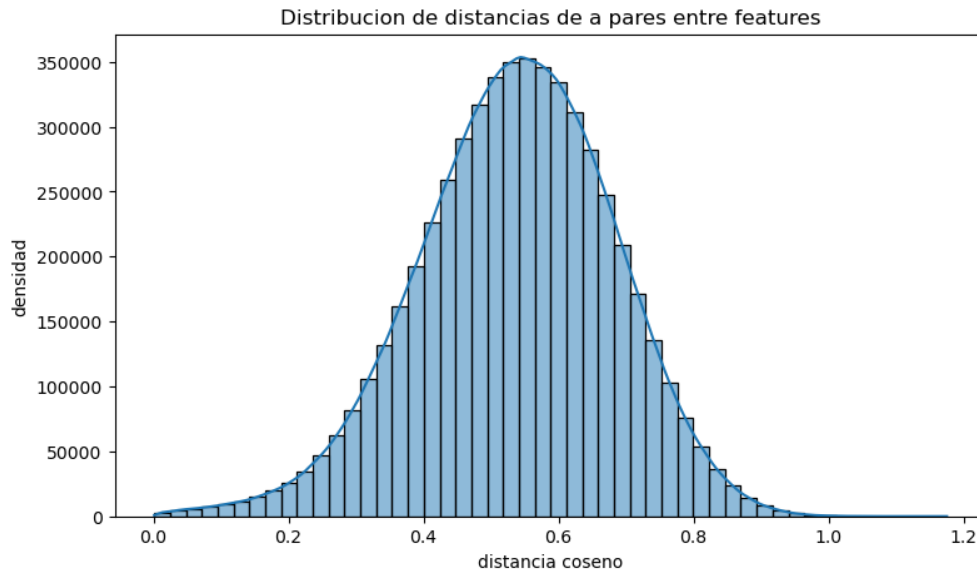


Fig. 6.2: *distribución de distancias coseno de a pares entre los features conseguidos para los genes.*

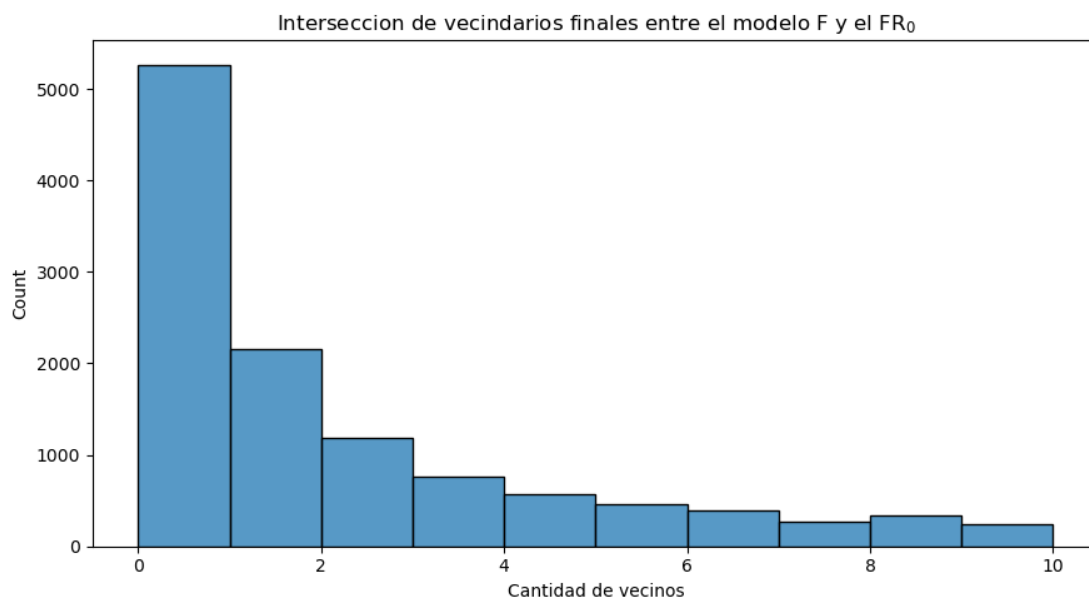


Fig. 6.3: *distribución de la cantidad de vecinos compartidos entre los 10 más cercanos para un nodo en el modelo con features en los genes correspondientes y el mismo nodo en el modelo con features asignadas de forma aleatoria.*

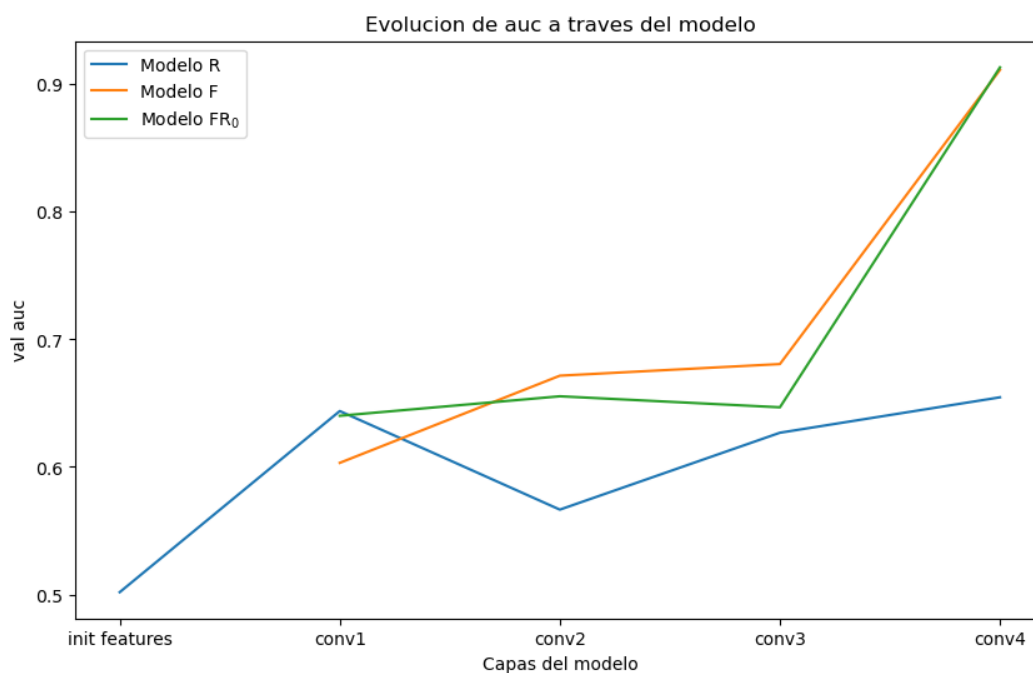


Fig. 6.4: *Evolución de AUC para los distintos modelos a través de sus capas. La performance para los modelos F y FR₀ no se puede calcular directamente de las features iniciales dado que estas difieren en dimensión para las drogas y los genes.*

Bibliografía

- [1] Feixiong Cheng, Chuang Liu, Jing Jiang, Weiqiang Lu, Weihua Li, Guixia Liu, Weixing Zhou, Jin Huang, and Yun Tang. Prediction of drug-target interactions and drug repositioning via network-based inference. *PLOS Computational Biology*, 8(5):1–12, 05 2012.
- [2] Yoshihiro Yamanishi, Michihiro Araki, Alex Gutteridge, Wataru Honda, and Minoru Kanehisa. Prediction of drug–target interaction networks from the integration of chemical and genomic spaces. *Bioinformatics*, 24(13):i232–i240, 07 2008.
- [3] Kanghao Shao, Yunhao Zhang, Yuqi Wen, Zhongnan Zhang, Song He, and Xiaochen Bo. DTI-HETA: prediction of drug–target interactions based on GCN and GAT on heterogeneous graph. *Briefings in Bioinformatics*, 23(3):bbac109, 04 2022.
- [4] Qing Ye, Chang-Yu Hsieh, Ziyi Yang, Yu Kang, Jiming Chen, Dongsheng Cao, Shibbo He, and Tingjun Hou. A unified drug–target interaction prediction framework based on knowledge graph and recommendation system. *Nature communications*, 12(1):6775–12, 2021.
- [5] Duncan J. Watts and Steven H. Strogatz. Collective dynamics of ‘small-world’ networks. *Nature*, 393(6684):440–442, 1998.
- [6] Albert-Laszlo Barabasi and Reka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, October 1999.
- [7] M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99(12):7821–7826, June 2002.
- [8] Victor Spirin and Leonid A. Mirny. Protein complexes and functional modules in molecular networks. *Proceedings of the National Academy of Sciences*, 100(21):12123–12128, 2003.
- [9] P. W. Anderson. More is different. *Science*, 177(4047):393–396, 1972.
- [10] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. KDD ’16, page 855–864, New York, NY, USA, 2016. Association for Computing Machinery.
- [11] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral Networks and Locally Connected Networks on Graphs. *arXiv:1312.6203 [cs]*, May 2014. arXiv: 1312.6203.
- [12] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS’17, page 1025–1035, Red Hook, NY, USA, 2017. Curran Associates Inc.

- [13] William L. Hamilton. Graph representation learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 14(3):1–159, 2020.
- [14] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2018.
- [15] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2015.
- [16] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.
- [17] Lionel Urán Landaburu, Ariel J Berenstein, Santiago Videla, Parag Maru, Dhanasekaran Shanmugam, Ariel Chernomoretz, and Fernán Agüero. TDR Targets 6: driving drug discovery for human pathogens through intensive chemogenomic data integration. *Nucleic Acids Research*, 48(D1):D992–D1005, 11 2019.
- [18] Stefi Nouleho Ilemo, Dominique Barth, Olivier David, Franck Quessette, Marc-Antoine Weisser, and Dimitri Watel. Improving graphs of cycles approach to structural similarity of molecules. *PLOS ONE*, 14(12):1–25, 12 2019.
- [19] Jérôme Hert, Peter Willett, David J. Wilton, Pierre Acklin, Kamal Azzaoui, Edgar Jacoby, and Ansgar Schuffenhauer. Comparison of topological descriptors for similarity-based virtual screening using multiple bioactive reference structures. *Org. Biomol. Chem.*, 2:3256–3266, 2004.
- [20] Jiaxuan You, Rex Ying, and Jure Leskovec. Design space for graph neural networks, 2021.
- [21] Ralph Abboud, İsmail İlkan Ceylan, Martin Grohe, and Thomas Lukasiewicz. The surprising power of graph neural networks with random node initialization, 2021.
- [22] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks?, 2019.
- [23] Xiaoshi Zhong, Rama Kaalia, and Jagath Rajapakse. Go2vec: transforming go terms and proteins to vector representations via graph embeddings. *BMC Genomics*, 20, 2019.
- [24] Zhen Yang, Ming Ding, Chang Zhou, Hongxia Yang, Jingren Zhou, and Jie Tang. Understanding negative sampling in graph representation learning, 2020.
- [25] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.
- [26] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.
- [27] Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks, 2018.