UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

EXPLORACIÓN DE MODELOS TRANSLACIONALES PARA RECOMENDACIÓN DE ÍTEMS.

Jaime Enríquez Ballesteros
Tutor: Fernando Díez Rubio

JUNIO 2021

EXPLORACIÓN DE MODELOS TRANSLACIONALES PARA RECOMENDACIÓN DE ITEMS

AUTOR: Jaime Enríquez Ballesteros
TUTOR: Fernando Díez Rubio

Dpto. Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Junio de 2021

Resumen (castellano)

Los Sistema de Recomendación son herramientas software que desde hace un par de décadas asisten a los usuarios de sistemas informáticos para el consumo de productos y/o servicios. Han ganado una tremenda popularidad debido al auge del comercio electrónico y a la vastísima oferta actual en la web. Por este motivo la investigación básica en este tipo de herramientas es cada vez más popular e importante.

En este trabajo se propone profundizar en una nueva aproximación a los modelos de recomendación que se viene empleando desde hace relativamente poco tiempo. Consiste en emplear algoritmos basados en *Knowledge Graph Embbedings* para representar entidades y relaciones entre ellas y su aplicación a la recomendación de ítems. En particular se desea explorar ciertos modelos denominados translacionales que se han demostrado muy útiles en esta tarea de recomendación.

Con este objetivo, se pretende reproducir la experimentación realizada en un estudio previo en el que se utilizan estos modelos como métodos de recomendación para la base de datos *MovieLens*, comparando su rendimiento frente a algunos modelos base de recomendación. Adicionalmente, se incluye la experimentación con un nuevo conjunto de datos, *LastFM*, con la particularidad de que en este *dataset* no se conocen los ratings explícitos de los usuarios. Como solución, se emplea una técnica que calcula los ratings implícitos de los usuarios del conjunto a partir de estadísticas relacionadas con la escucha de canciones.

Por último, se presenta un posible modelo translacional cuya función de pérdida se basa en la topología del grafo. Esta propuesta se realiza de manera puramente teórica sin poder certificar su validez de forma empírica, que se plantea como trabajo futuro.

Palabras clave (castellano)

Graph Embedding, Grafos de conocimiento, Modelo translacional, Similitud topológica, Recomendación, MovieLens, LastFM

Abstract (inglés)

Recommendation Systems are software tools that for a couple of decades have been assisting users of computer systems for the consumption of products and / or services. These have gained tremendous popularity due to the rise of the e-commerce landscape and the vast current offerings on the web. For this reason, basic research in these types of tools is increasingly popular and important.

This dissertation proposes to delve into a new approach for recommendation models that has been used for a relatively short time. It consists on using algorithms based on *Knowledge Graph Embbedings* to represent entities and relationships between them and their application for the recommendation of items. In particular, we want to explore certain so-called translational models that have proven very useful in this recommendation task.

With this in mind, we intend to reproduce the experimentation carried out in a previous study in which these models were used as recommendation methods for the *MovieLens* database, comparing their performance against some recommendation base models. Additionally, experimentation with a new data set, *LastFM*, is included, with the particularity that in this dataset the explicit ratings of the users are not known. As a solution, a technique is used that calculates the implicit ratings of the users of the set from statistics related to listening to songs.

Finally, a possible translational model is presented whose loss function is based on the topology of the graph. This proposal is made in a purely theoretical way without being able to certify its validity in an empirical way, which is proposed as future work.

Keywords (inglés)

Graph Embedding, Knowledge graphs, Translational model, topological similitude, Recommendation, MovieLens, LastFM

Agradecimientos

A mi tutor Fernando, por su apoyo y paciencia para realizar este proyecto.

A mis padres, por aguantarme durante todos estos años. A mis amigos, por mantenerse a mi lado en los momentos más difíciles. A mi pareja, por estar siempre ahí.

Y especialmente, a mi hermano Miguel, que siempre será mi mejor amigo.

INDICE DE CONTENIDOS

INDICE DE FIGURAS

INDICE DE TABLAS

1 Introducción

Los sistemas de recomendación han cobrado una gran importancia durante las últimas décadas debido al uso multitudinario de plataformas online como redes sociales, aplicaciones multimedia o servicios a demanda. Aplicaciones como *Instagram*, *Spotify* o *Uber* son utilizadas por millones de usuarios cada día. En la sociedad actual existe una gran dependencia de estas plataformas, siendo productos casi indispensables para nuestras vidas. Una de las piezas clave por la que estas aplicaciones son tan esenciales son las recomendaciones.

La constante necesidad por nuevos estímulos y nuevas experiencias es una característica propia de la naturaleza humana, y que puede definirse como principal objetivo de los sistemas de recomendación. Esta necesidad parece haberse acentuado en la época del Internet, donde se obliga a que las recomendaciones sean casi perfectas para recibir una mínima atención por parte del usuario. Por ello, la exploración de nuevos sistemas de recomendación que puedan mejorar los actuales tiene una gran importancia y puede ser de enorme trascendencia. Este trabajo forma parte de este procedimiento como proceso de reproducibilidad.

1.1 Motivación

Esta memoria de TFG tiene como objetivo explicar el proceso de experimentación y análisis de resultados al utilizar distintos modelos translacionales como sistemas de recomendación de ítems.

Los grafos son un conjunto de estructuras que están presentes en una gran variedad de campos de la actividad y el conocimiento, desde las ciencias sociales (red de amistades) hasta la biología (interacciones entre proteínas). El análisis de este tipo de estructuras logra extraer información escondida en el grafo, como por ejemplo reconocer distintas comunidades dentro de una red social, por lo que suele ser un proceso esencial a la hora de entender los datos. Por otro lado, los distintos algoritmos encargados de dicho análisis suelen ser muy costosos y difícilmente escalables debido a la gran cantidad de información a procesar. Con el propósito de reducir el coste de procesamiento surgen distintas técnicas de *graph embedding* (embebimiento de grafos). Estas técnicas representan el grafo como un conjunto de vectores de baja dimensión con el fin de obtener un mejor rendimiento a la hora de procesar los datos.

A lo largo de esta memoria se profundizará en la utilización de distintos modelos de grafos como sistemas de recomendación de ítems. El propósito es estudiar los modelos que vamos a trabajar (que se denominan translacionales), para comprenderlos y explicarlos a través del proceso de experimentación y análisis de resultados.

Los grafos de conocimiento representan redes de entidades del mundo real - objetos, personas o conceptos - e ilustran cómo se relacionan entre sí. Formalmente, se definen como grafos compuestos por una serie de entidades interconectadas por relaciones semánticas, que emplean los modelos denominados translacionales como técnicas de *graph embedding*. Estos modelos asignan una puntuación a cada posible tripleta del grafo <*entidad, relación, entidad*>, tanto existentes como no existentes, con el fin de obtener una puntuación mayor para las tripletas que sí existen. Cada modelo translacional difiere de uno u otro según la

función que se utilice para calcular la puntuación. La representación de los *embeddings* cambiará según la función obtenga una mejora en la puntuación respecto a la representación previa. En la mayoría de casos la función será de pérdida, por lo que se buscará lograr una puntuación mínima. Esta puntuación es análoga al nivel de fidelidad de las representaciones respecto al grafo original.

Hasta hace poco tiempo, los modelos translacionales se limitaban a obtener el conjunto de vectores del grafo y sobre estos vectores se entrenaba un sistema de recomendación. Pero muy recientemente, los propios modelos translacionales han comenzado a ser utilizados como sistemas de recomendación debido a su gran eficiencia y escalabilidad. Se ha demostrado que los resultados obtenidos mejoran a algunos de los modelos clásicos de filtrado colaborativo como pueden ser *SVD* (*Singular Value Descomposition*) o *MostPop* (*Most Popular*).

Al ser una propuesta muy novedosa y sobre la que se ha realizado poca experimentación, hemos creído necesario realizar un proceso de reproducción de la experimentación para comprender y verificar, en la medida de lo posible, la efectividad de este método. Adicionalmente, con el fin de extender el alcance de esta propuesta, creemos que se debe probar a utilizar un nuevo conjunto de datos sobre el que no existan resultados previos.

Debemos hacer énfasis en la importancia que conlleva un proceso de reproducibilidad. Aunque pueda parecer un desarrollo innecesario y repetitivo, es un requisito clave para la validación de cualquier experimento. Esto se debe a la existencia de investigaciones en las que los resultados han resultado ser fraudulentos y que han sido destapadas años más tarde, véase [1]. Concretamente, dentro del campo de los sistemas de recomendación, donde se llevan a cabo procesos muy pesados y los resultados pueden ser fácilmente falsificados, es necesario llevar a cabo este tipo de desarrollos. Asimismo, se han comenzado a organizar talleres dedicados específicamente a las reproducciones de sistemas de recomendación, como [2], donde se discuten y analizan los resultados obtenidos mediante estos procesos.

1.2 Objetivos

En términos generales el trabajo que planteamos tiene por objetivo principal reproducir la experimentación realizada en [3] sobre el conjunto de datos de *MovieLens*[1] y extenderlo, aplicando la misma metodología, sobre un conjunto de datos extraídos de la plataforma de música *LastFM*[2].

Este objetivo principal podemos descomponerlo en los siguientes objetivos más específicos, que son los que guiarán el desarrollo del trabajo:

- O1: Preprocesar los datos mediante dos etapas, de separación y preparación, previas a la etapa de entrenamiento.
- O2: Comprender e implementar los modelos translacionales.
- O3: Analizar los resultados frente a los obtenidos mediante modelos convencionales de sistemas de filtrado colaborativo.
- O4: Proponer un modelo alternativo basado en los modelos translacionales que pueda obtener una mejoría de rendimiento. Concretamente se probará a diseñar una nueva metodología de ranking para el sistema de recomendación utilizando un algoritmo topológico de similitud.

[1] MovieLens, https://grouplens.org/datasets/movielens/1m/

[2] LastFM, http://millionsongdataset.com/lastfm/

1.3 Organización de la memoria

La memoria consta de los siguientes capítulos:

- *Estado del arte*: se hace un breve repaso sobre los usos y aplicaciones de los grafos de conocimiento.
- *Modelos teóricos*: se explican las distintas técnicas de graph embeddings, describiendo cómo se crean los vectores según cada tipo de grafo, profundizando en la metodología que siguen los modelos translacionales y cómo forman los ránkings de recomendación. Además, se describe el algoritmo topológico de homología persistente que utiliza estructuras conocidas como códigos de barras para calcular la similitud entre distintas nubes de puntos.
- *Experimentación*: se detalla la estructura de cada conjunto de datos y la arquitectura de los programas de entrenamiento y prueba, y se describen los procedimientos llevados a cabo para la obtención de los conjuntos de datos, el preprocesamiento previo al entrenamiento, la implementación de los modelos y la descripción de un método alternativo basado en la homología persistente.
- *Resultados*: se analizan los resultados obtenidos durante el periodo de prueba para cada conjunto de datos y cada modelo translacional. Se describen las métricas utilizadas.
- *Conclusiones y trabajo futuro*: se extraen conclusiones de los resultados obtenidos en el apartado anterior y se plantean futuros posibles experimentos para extender el estudio realizado.
- *Anexos*: se añaden fragmentos relevantes de código y tablas de datos que se irán referenciando a lo largo de la memoria sobre las tareas de preprocesamiento de los datos (Anexos A y B), modificaciones en la librería *OpenKE* (Anexos C y D) o resultados de la experimentación (Anexo E).

2 Estado del arte

El uso de técnicas de procesamiento y construcción de grafos de conocimiento ha crecido en los últimos años. El incremento del poder computacional y el avance en el campo de la algoritmia han permitido utilizar de manera empírica modelos propuestos tiempo atrás de manera puramente teórica y el descubrimiento de nuevos métodos. Esto ha facilitado avances en aplicaciones del mundo real dentro de campos como la biología, la medicina, la lingüística o la sociología. Concretamente, las técnicas de *graph embeddings* para los grafos de conocimiento han posibilitado un procesamiento más eficiente de este tipo de grafos, que a su vez ha supuesto una mejora indirecta en una gran variedad de campos.

Por ello, parece necesario realizar un breve repaso sobre algunas de estas aplicaciones, prestando especial atención a los *graph embeddings*. Además, se debe hacer mención de algunas aplicaciones novedosas, propuestas durante los últimos años, que han obtenido buenos resultados para alguno de estos modelos.

Dentro del campo de la biología se incluyen numerosos estudios que se apoyan en grafos de conocimiento para realizar grandes avances en el campo. En [4] se elabora una investigación sobre posibles candidatos como medicamentos para un cáncer de piel muy agresivo conocido como melanoma cutáneo metastásico y facilita una plataforma web pública para su libre uso. Utilizando grafos de conocimiento para definir las relaciones entre proteínas, medicamentos y la propia enfermedad, se define un modelo matemático que obtiene unos resultados muy prometedores. En [5] se propone un método basado en la fusión de grafos de conocimiento obtenidos de distintas fuentes para la detección de Alzheimer a partir de una serie de genes. Utilizando los nuevos conjuntos de datos se obtienen mejores resultados para la detección de Alzheimer en comparación con los conjuntos individuales.

El procesamiento natural de lenguaje es uno de los campos de la lingüística cuya popularidad ha aumentado más en los últimos años. La mayoría de investigaciones relacionadas con este campo se apoyan en grafos de conocimiento y *graph embeddings* para obtener representaciones vectoriales de textos para su consecuente procesamiento. [6] propone un método de obtención de *graph embeddings* a partir del conjunto de tripletas relacionadas al texto y una serie de descripciones textuales asociadas a cada palabra. Por su parte, [7] utiliza modelos translacionales (concretamente *MTransE* y variantes) para fusionar bases de conocimiento en distintos idiomas relacionando cada palabra con su correspondiente traducción.

La rama de la sociología dedicada al estudio de las redes sociales ha experimentado un espectacular aumento en la cantidad y la calidad de la investigación llevada a cabo en los últimos años, debido al interés en profundizar el uso creciente de redes sociales por la población. Asimismo, la posibilidad de procesar grafos de conocimiento de manera sencilla y eficiente ha facilitado dicha investigación. En [8] se alinean usuarios presentes en distintas redes sociales para facilitar el intercambio de información entre redes. El estudio demuestra cómo la predicción de enlaces sociales y recomendación dentro de las redes mejora utilizando este modelo. En [9] se permite inferir patrones de movilidad de la población al relacionar la movilidad de individuos con la de sus amigos. Por su parte, [10] demuestra cómo las técnicas de *graph embedding* son muy útiles para la detección de comunidades en redes y la clasificación de nodos.

Relacionado con las redes sociales disponemos de plataformas de preguntas y respuestas, conocidas como plataformas *cQA* (*Community-based question answering*). Varios estudios ([11] y [12], entre otros) muestran cómo usar las interacciones sociales dentro de una plataforma para relacionar respuestas con preguntas mejoran los modelos actuales.

Como se ha mencionado anteriormente al repasar [6], el estudio de la movilidad es un campo en el que el procesamiento de grafos de conocimiento es muy útil. Otros ejemplos son [13] y [14], donde se obtienen *embeddings* de distintas zonas geoespaciales a partir de datos de movilidad a gran escala, con el objetivo de predecir el flujo de movimiento de personas. Estos estudios han sido oportunos para aplicaciones en el mundo real como el control o predicción de aforo.

Los sistemas de recomendación están muy ligados a los grafos de conocimiento debido a que las entidades, relaciones e ítems del sistema pueden ser representados de manera concisa en un grafo de conocimiento. Algunos se apoyan en los *graph embeddings* para obtener representaciones numéricas del grafo y elaborar las recomendaciones con un procesamiento posterior. En [3], base de este trabajo, se utilizan modelos translacionales directamente como sistemas de recomendación, utilizando una combinación de datos de distintas fuentes. Este método será descrito en más detalle en la siguiente sección. Finalmente, en [15] se emplean grafos de conocimiento y redes neuronales para crear *graph embeddings* para los ítems que dependen de a qué usuarios les interesan dichos ítems. La combinación de *graph embeddings* y redes neuronales es una propuesta cuya popularidad también ha crecido recientemente para sistemas de recomendación.

3 Modelos teóricos

En esta sección explicaremos en qué consiste el proceso de embeber un grafo y las técnicas que existen con este propósito. Profundizaremos en los modelos translacionales y detallaremos el proceso a seguir para ser usados como sistemas de recomendación apoyándonos en [16], [17] y [18]. Por último, describiremos un algoritmo topológico de similitud que utiliza los códigos de barras para calcular una similitud entre conjuntos, representados como nubes de puntos, detallado en [19].

3.1 Graph embeddings.

El *embedding* de grafos consiste en la conversión de un grafo o estructura similar en un conjunto de vectores, de almacenamiento menor, donde se preserva la información del grafo. De esta forma el procesamiento de los datos es mucho más eficiente.

Según el tipo de grafo que se quiera transformar, las entradas a los modelos de *embedding* pueden variar. Por ejemplo, para un grafo homogéneo se esperan unos datos con distinta estructura a los que se tienen con un grafo de conocimiento. La salida también podrá variar según qué elementos sean representados por el *graph embedding*. Por ejemplo, un *embedding* puede tener la forma de un único vector para todo un grafo o estar compuesto por un conjunto de vectores, uno por vértice. La primera opción podría ser útil a la hora de calcular la similitud entre dos grafos desiguales. La segunda podría ser utilizada al estudiar las relaciones entre los componentes de un mismo grafo. En la *Figura 1* se visualizan este tipo de transformaciones.

De este modo es importante distinguir los distintos tipos de entradas y salidas que se pueden tener en el proceso de transformación de grafos. Este paso es clave para poder entender las distintas técnicas de *graph embedding*.

3.1.1 Input en *graph embeddings*.

Como se detalla en [16], se distinguen cuatro tipos de grafos que podrán servir como entrada a un modelo de *graph embedding*:

- *Grafos homogéneos*: se definen como grafos compuestos por nodos de un solo tipo y enlaces de un solo tipo. Los enlaces pueden ser de tipo dirigido o no dirigido y tener un peso asignado. Intuitivamente, tanto la dirección como el peso que se le asigna a cada enlace serán propiedades que aportan información que podrá ayudar a crear los *embeddings*. Por ejemplo, dos nodos conectados por un enlace de gran peso tendrán *embeddings* más próximos en comparación con otros nodos con enlaces más débiles.

- *Grafos heterogéneos*: se definen como grafos compuestos por nodos de más de un tipo y/o enlaces de más de un tipo. Como para los grafos homogéneos, los enlaces pueden ser dirigidos o no dirigidos y tener un peso asignado. Se destaca el subconjunto de *grafos de conocimiento*, donde cada nodo es una entidad y cada enlace una relación entre las entidades cabeza y cola. De este modo los grafos de conocimiento están compuestos por un conjunto de tripletas <*head, relation, tail*>. Un ejemplo para un grafo de conocimiento de gustos musicales podría ser la tripleta <*Alice, Escucha, Bob Dylan*>.

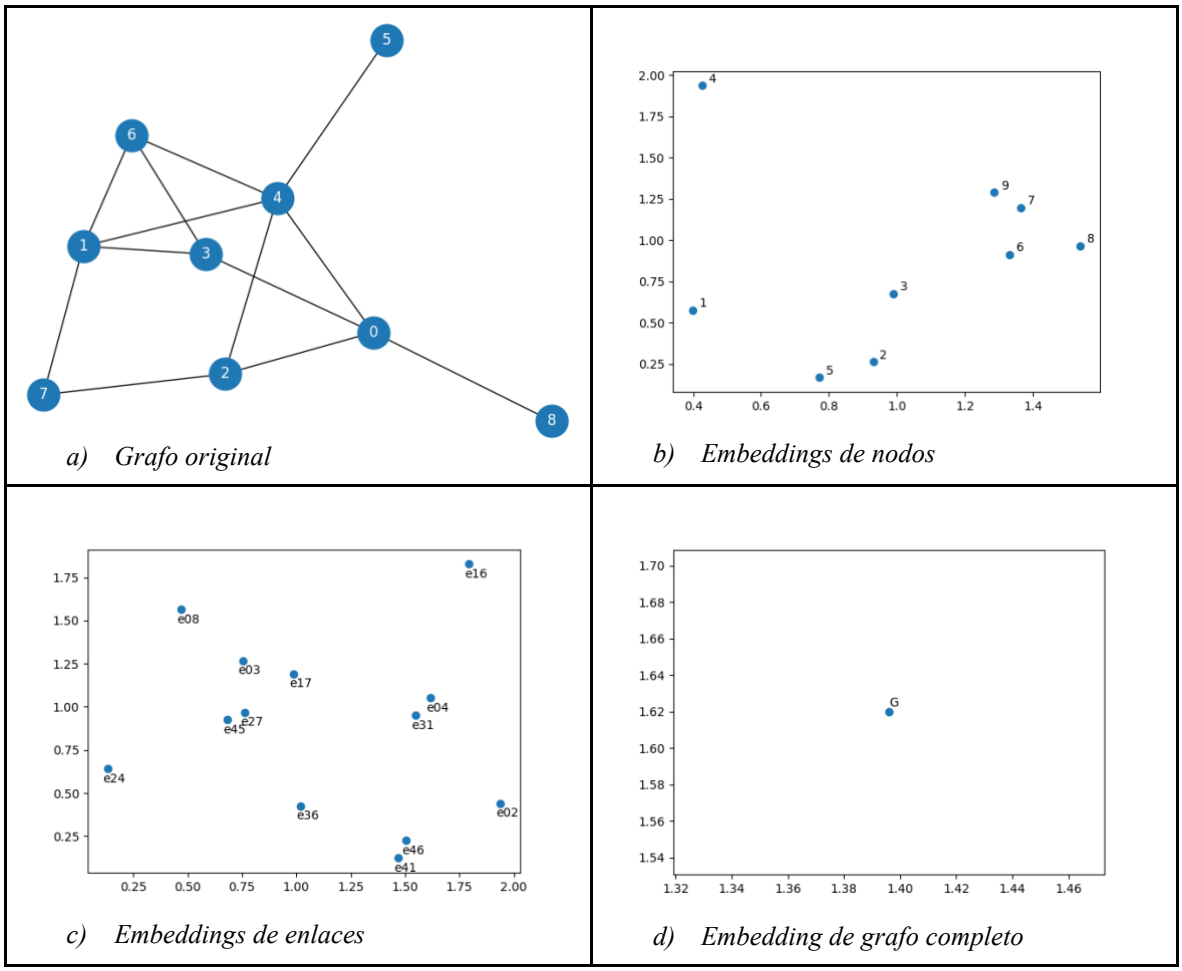


Figura 1: Ejemplos de distintos tipos de embeddings.

- *Grafos con información auxiliar*: se definen como grafos a los que se añade información adicional sobre algún elemento del grafo (nodo o enlace) o sobre el mismo grafo. Se distinguen varios tipos de información adicional, como los atributos o las etiquetas que se añaden a un nodo. Las *bases de conocimiento* son otro tipo de estructuras que añaden información a un grafo. Algunas bases de conocimiento como *YAGO*[3] (*Yet Another Great Onlology*) o *DBpedia*[4] (*Databasepedia*) pueden aumentar la precisión de un *embedding* añadiendo entidades y relaciones extra al grafo original, descubriendo nuevos patrones.

- *Grafos construidos a partir de datos no relacionales*: se definen como grafos creados a partir de matrices, imágenes u otros conjuntos de datos no relacionales. Por tanto, se añade un paso extra al utilizar este tipo de grafos como input para los graph embeddings. Normalmente se interpretan las matrices o imágenes como matrices de adyacencia del grafo, aunque existen técnicas alternativas. Por ejemplo, aplicar KNN (*K-Nearest Neighbors*) sobre la matriz original para formar un grafo y estimar una nueva matriz de adyacencia a partir de este nuevo grafo.

[3] YAGO, https://yago-knowledge.org/
[4] DBpedia, https://www.dbpedia.org/

3.1.2 Output en *graph embeddings.*

Pese a que en la sección anterior se hace énfasis en los *embeddings* sobre los nodos del grafo, se debe tener en cuenta que también existen *embeddings* sobre el resto de subpartes del grafo o sobre el grafo mismo, como muestra la *Figura 1*. Dependiendo de la tarea que quiera llevar a cabo, se utilizará un modelo concreto de *embedding* u otro. Durante esta sección se hará un breve resumen de las diferentes estructuras que se pueden obtener tras un proceso de *embedding*, resumiendo el contenido que se describe en [16] y [17].

- *Embeddings de nodos*: se definen como vectores de baja dimensión para cada nodo del grafo, donde nodos que estén cerca en el grafo deberán estarlo en su representación vectorial. El principal problema de este tipo de *embedding* yace en el tipo de métricas que se utilizan para calcular la similitud entre nodos. Las aproximaciones de primer y segundo orden suelen ser las más utilizadas.

La aproximación de primer orden representa la distancia directa entre un par de nodos, por ejemplo el peso del enlace que los conecta. A su vez, la aproximación de segundo orden describe la aproximación entre dos nodos a partir de sus vecinos próximos, por ejemplo calculando el coseno entre los vectores similitud de cada nodo con sus vecinos. Se pueden emplear aproximaciones de mayor orden, aunque sean menos corrientes.

- *Embeddings de enlaces*: se definen como vectores de baja dimensión para cada enlace del grafo. Se suelen utilizar para la predicción de enlaces y nodos en grafos de conocimiento. Esto se debe a que un enlace involucra a los dos nodos que conecta, además de indicar el direccionamiento, por lo que suelen contener mayor información que los nodos.

- *Embeddings híbridas*: se definen como vectores para representar combinaciones de unidades que componen un grafo. Por ejemplo, la tupla formada por un nodo y un enlace (denominadas *subestructuras*). También se puede extender el *embedding* a subgrafos dentro del grafo original.

Al contrario que los dos métodos anteriores, a esta metodología se le añade el problema de decidir sobre qué parte(s) del grafo se deben realizar los *embeddings* cuando se quieran preparar unos datos para su consecuente procesamiento.

El *embedding híbrido* se puede utilizar para representar una unidad mínima del grafo (por ejemplo un nodo) mediante la combinación de *embeddings* del subgrafo al que pertenece y su propio *embedding*. Este método es muy utilizado en aplicaciones de detección de comunidades como en grafos de redes sociales.

- *Embeddings de grafo completo*: se definen como vectores que representan un grafo completo. Estos vectores suelen representar grafos de pequeño tamaño, como proteínas o moléculas para el consecuente cálculo de similitud con otros grafos.

La obtención de estos *embeddings* tiene por desventajas su gran tiempo de procesamiento y la posible pérdida de información a la que está sujeta. Este último problema también puede suceder para el resto de *embeddings*, pero tendrá menor efecto cuanto mayor sea la granularidad de la salida.

3.1.3 Técnicas de *graph embeddings*.

En esta sección se presentan algunas técnicas de *graph embeddings*, extraídas de [16] y [17]. Se definen los principales grupos de algoritmos de manera resumida con el objetivo de conocer la arquitectura de estos procesos.

La principal diferencia entre los algoritmos de *graph embeddings* yace en qué característica del grafo intentan preservar en los *embeddings*. Independientemente de cómo se aborde este problema, todos estos métodos tienen como objetivo maximizar la similitud entre las unidades del grafo que sean semejantes.

- *Métodos de factorización*: estos métodos representan las conexiones en los grafos como matrices. Esta matriz podrá ser la matriz de adyacencia, la matriz Laplaciana o la matriz de probabilidad de transición de nodos, entre otras. A partir de estas matrices se realiza un proceso de factorización. Este proceso consiste en la minimización de una función de pérdida que utilice la matriz como entrada.

- Métodos basados en *Random Walks*: estos métodos suelen utilizarse con el objetivo de mantener en los *embeddings* propiedades como la centralidad del grafo o similitud entre grafos. Son técnicas especialmente utilizadas sobre grafos de grandes dimensiones ya que su procesamiento es excesivamente costoso. Normalmente consisten en contar el número de veces que se pasa por un nodo o enlace durante una serie de caminos aleatorios sobre el grafo.

- Métodos basados en *Deep Learning*: los grandes avances en el campo del aprendizaje automático y redes neuronales ha supuesto un incremento en su utilización sobre grafos. La mayoría de modelos utilizados se han trasladado desde otros campos y adaptados con la finalidad de crear *embeddings* de grafos. Estos métodos se pueden dividir en dos grupos:

    - Métodos basados en *random walks*. Al igual que los métodos basados en *random walks*, procesa los datos obtenidos a partir de unos caminos aleatorios sobre el grafo. La principal diferencia es que estos datos son procesados posteriormente utilizando técnicas de *Deep Learning*, normalmente modelos de redes neuronales.
    - Métodos basados en el propio grafo. Este segundo grupo de métodos de *Deep Learning* procesan la estructura del nodo directamente utilizando redes neuronales o *autoencoders*. La principal ventaja de estos métodos sobre el resto es su flexibilidad y la obtención de buenos resultados sin necesidad de modificar sus parámetros por defecto.

- Métodos de optimización basados en reconstrucción de enlaces: estos métodos tratan de minimizar la pérdida de información de los *embeddings* respecto al grafo original optimizando funciones basadas en la reconstrucción del grafo. Un subgrupo son los métodos *Minimized Distance-based Loss*. Consisten en minimizar la pérdida entre la distancia original entre nodos y la distancia entre sus correspondientes *embeddings*. Como representación de la distancia entre los *embeddings* de los nodos se utilizan funciones como la distancia euclídea. Otros subgrupos incluyen los métodos *Maximized Edge Reconstruction Probability* o los métodos *Minimized Margin-based Ranking Loss*.

Este último subgrupo es utilizado a menudo para la creación de los *embeddings* de grafos de conocimiento. Como parte de ellos se encuentran los *modelos translacionales*, cuya popularidad ha incrementado en los últimos años y cuya importancia para entender el presente trabajo es máxima. Estos modelos serán descritos en detalle en la siguiente sección.

3.2 Modelos translacionales.

En esta sección se detalla, de forma resumida, la definición de los modelos translacionales, que extraemos de [3] y [16]. Como se indica en estos artículos, cualquier grafo de conocimiento compuesto por *n* entidades y *m* relaciones puede ser representado por un conjunto de tripletas, cada una de ellas formada, a su vez, por una entidad cabeza *h* (*head*), una entidad cola *t* (*tail*) y una relación *r* (*relation*). Un ejemplo puede ser <*Alicia, Escucha, Bob Dylan*>.

El proceso para representar cada entidad, enlace o tripleta de un grafo de conocimiento en un espacio de dimensión *d* arbitraria, generalmente consiste en tres pasos:

1. *Crear la representación inicial de los vectores*: las entidades se representan con vectores de dimensión *d* y las relaciones como operaciones sobre estos vectores.
2. *Definir una función de puntuación*: se aplica sobre cada posible tripleta del grafo y deberá puntuar imitando el comportamiento del grafo de conocimiento original.
3. *Optimizar las representaciones*: se minimiza la función de pérdida según las funciones de puntuación. Las tripletas que existen (reales) decrementan el valor. Las tripletas no existentes en el grafo lo aumentan. De este modo, la pérdida asociada a cada elemento tenderá a ser pequeña para tripletas existentes y alta para tripletas falsas.

En [16], dependiendo de qué tipo de función de puntuación se utilice, se distinguen dos grupos que utilicen esta técnica de *embeddings*: modelos translacionales y modelos de coincidencia semántica. La principal diferencia entre los dos yace en cómo se calcula el error de una representación. Los modelos translacionales se basan en funciones de distancia, mientras que los modelos de coincidencia semántica hacen uso de funciones de similitud. El primer grupo es la pieza clave de este trabajo por lo que lo describiremos con mayor detalle.

Como se ha mencionado, los modelos translacionales basan sus funciones de pérdida en el cálculo de distancias entre entidades, tras aplicar una translación que dependerá de la relación que une dichas entidades. Cada modelo translacional (de los muchos existentes) depende de dos factores: la forma en la que se aplican las translaciones y la fórmula a utilizar para calcular la distancia entre la entidad cabeza trasladada y la entidad cola. A continuación revisaremos los modelos analizados en este trabajo.

Como indican [3] y [16], el principal representante de los modelos translacionales es el modelo ***TransE*** [20]. Representa las entidades y enlaces como vectores de dimensión *d* (a elección del usuario, como hemos indicado más arriba). A partir de la idea de que una relación sirve como conexión entre dos entidades, se puede obtener la siguiente igualdad para una tripleta <*h, r, t*> existente: $h + r = t$. Siguiendo con el ejemplo anterior <*Alicia, Escucha, Bob Dylan*>, el modelo debe asegurar que *Alicia* + *Escucha* se aproxime lo máximo al valor de la entidad *Bob Dylan*. Del mismo modo, para la tripleta <*Bob, Escucha, Alice Coltrane*> el modelo deberá maximizar *Bob* + *Escucha* = *Alice Coltrane*. Además, se

da a pie a que el valor de la igualdad *Alicia - Bob Dylan = Bob - Alice Coltrane* también se maximice.

Utilizando esta idea, la función de puntuación de tripletas para el modelo *TransE* se define como:

$$f_r(h,t) \;=\; -||\, h \;+\; r \;-\; t \,||_{1/2} \text{ con } \; h,t,r \;\in R^{\mathrm{d}}$$

El objetivo de este modelo consiste en que la pérdida sea menor para las tripletas que existan dentro del grafo. Para ello se lleva a cabo un entrenamiento del modelo, donde se modifican los valores de los vectores utilizando técnicas de *Descenso de Gradiente* (*Gradient Descent*). Es común utilizar *Stochastic Gradient Descent* con *TransE* y sus variantes al tener que tratar con un gran volumen de datos. Con este objetivo, la función de pérdida utilizada para todos los modelos translacionales, denominada *pairwise ranking loss function*, es la siguiente:

$$L \;=\; \sum_{(h,r,t)\in D^{+}} \; \sum_{(h',r,t')\in D^{-}} \; max(\,0, \gamma \;+\; f_r(h,t) \;-\; f_r(h',t')$$

El principal problema de *TransE* se debe a su simpleza. Para relaciones *1* a *N* puede suceder que entidades distintas obtengan representaciones vectoriales muy similares debido a que se creen conflictos del tipo: $h + r = t_1$ y $h + r = t_2$ con $t_1 \neq t_2$. Este mismo problema puede darse para relaciones *N* a *N* o *N* a *1*. Como solución se proponen variantes de *TransE* que permitan tener distintas representaciones para entidades dependiendo de la relación. Las principales variantes consideradas son *TransH* y *TransR*.

Método	***Embeddings***	**Función de puntuación,** $f_r(h,t)$
TransE	$h,t,r \;\in R^{d}$	$-\|\| \, h + r - t \, \|\|_{1/2}$
TransH	$h,t,r,w_r \;\in R^{d}$	$-\|\| \, (h - w_r^{\top} \, h \, w_r) + r - (t - w_{\mathrm{r}}^{\top} h \, w_r) \, \|\|_2^2$
TransR	$h,t \in R^{d}$ $r \in R^{\mathrm{k}};\; M_r \;\in R^{d\times k}$	$-\|\| \, M_r \, h + r - M_r \, t \, \|\|_2^2$
TransD	$h, w_h, t, w_t \in R^{d}$ $r, w_r \;\in R^{k}$	$-\|\| \, (w_{\mathrm{r}} w_{\mathrm{r}}^{\top} + \; I) \, h + r - \; (w_r w_r^{\top} + \; I) \, t \|\|_2^2$
TransM	$h,t,r \;\in R^{d}$	$-\theta_r \|\| \, h + r - t \, \|\|_{1/2}$
TransF	$h,t,r \;\in R^{d}$	$(h + r)^{\top} t \; + (t - r)^{\top} h$

Tabla 1: Modelos translacionales

TransH *[21]* basa su función de puntuación de tripletas en la creación de distintos hiperplanos según la relación utilizada. Para cada relación se crea un vector normal que permita obtener la proyección de los vectores correspondientes a las entidades cabeza y cola. Utilizando las proyecciones de los vectores, el entrenamiento del modelo es idéntico al de *TransE*.

Por su parte, ***TransR*** [22] sigue una idea muy similar a *TransH*. Utiliza distintos espacios para distintas relaciones del grafo. Por lo tanto, las entidades de cada tripleta se proyectan en el nuevo espacio utilizando la matriz de proyección correspondiente a la relación que las une. Como *TransH*, tras obtener la proyección de las entidades, el proceso de entrenamiento es equivalente al utilizado en *TransE*.

Las fórmulas para estos dos últimos modelos, así como para otras variantes populares de *TransE* se incluyen en la Tabla 1, mostrada más arriba.

En la siguiente *Figura 2* podemos visualizar distintos ejemplos para los modelos *TransE*, *TransH* y *TransR* que ilustran, visualmente, en qué consisten las transformaciones en vectores de los grafos de conocimiento:

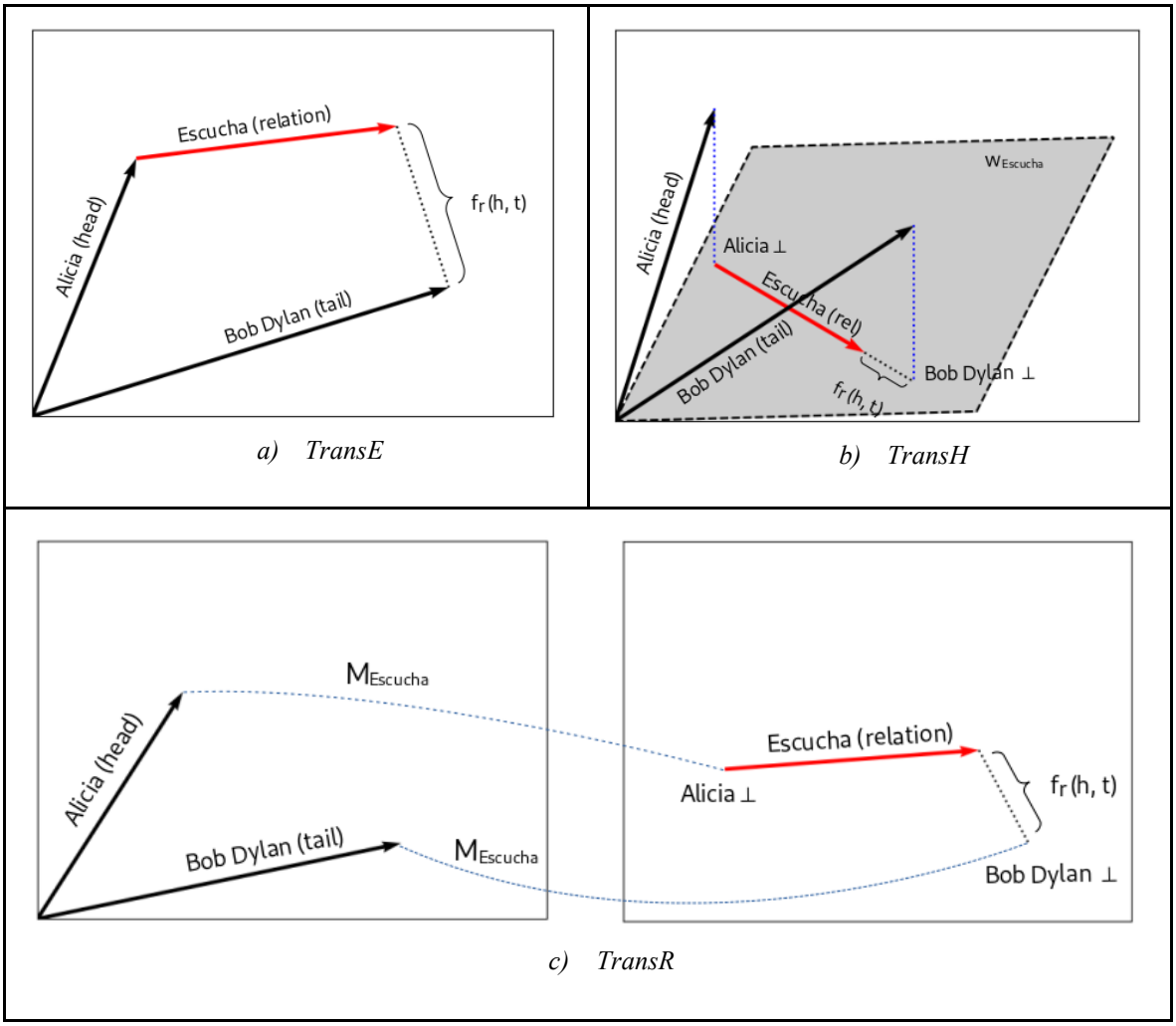


Figura 2: Ejemplos de TransE, TransH y TransR. Para TransE (a), los vectores se sitúan en el mismo espacio. Para TransH (b), los vectores son proyectados sobre el plano $w_{escucha}$ correspondiente a la relación "escucha". Para TransR (c) , los vectores son proyectados sobre un nuevo espacio mediante su multiplicación con la matriz $M_{escucha}$ correspondiente a la relación "escucha". La función de puntuación se obtiene a partir de las nuevas proyecciones de los vectores.

Como ya hemos indicado más arriba, recientemente se ha propuesto utilizar estos modelos como sistemas de recomendación. Utilizando la fusión de grafos que aporten información extra sobre las entidades del grafo principal, se pueden mejorar las predicciones de los ítems que se proponen a los usuarios. A partir de esta fusión se obtienen grafos de conocimiento que se pueden procesar por los modelos translacionales. A partir de estos nuevos grafos, la propuesta utiliza la función de pérdida del modelo translacional como función de ranking del sistema de recomendación.

Una función de ránking debe puntuar cada par usuario-ítem ante la posibilidad de que al usuario le interese el ítem en cuestión. Al utilizar la función de pérdida negada de un modelo translacional como función de ránking, se recomendarán los ítems que tengan una representación vectorial más similar a las ya relevantes para el usuario. Por lo tanto, el entrenamiento del sistema de recomendación consistirá en la asignación de vectores a los usuarios e ítems del conjunto de datos, mientras que para el proceso de prueba se ordenarán los ítems no revisados por cada usuario según la puntuación que obtengan (con la función de pérdida del modelo). Este procedimiento será descrito con más detalle en las siguientes secciones.

3.3 Similitud topológica.

Como hemos mencionado al comienzo de esta sección, en [19] se explica un método para medir la similitud entre dos estructuras que representan proteínas basado en la topología de las mismas. El trabajo tiene como objetivo combinar medidas de topología y geometría de una molécula, representada como un grafo, para poder compararlas entre sí. La transformación de molécula a grafo consiste en representar cada átomo como un nodo y cada conexión como un enlace. Asimismo, las coordenadas de los nodos del grafo resultante son obtenidas a partir de la estructura original.
Como se explica en [19], la topología es un campo que obtiene muy buenos resultados al comparar una molécula y otra versión modificada (por rotación o traslación) de la misma. Por su parte, el campo de la geometría sólo relaciona estructuras con volumen similar. Es deseable, por lo tanto, poder contar con modelos que combinen ambas propiedades.

Una forma de cuantificar la topología de una estructura, es mediante los números de Betti de dimensión cero (b_0), uno (b_1) y dos (b_2):

- b_0 – representa el número de *componentes*.
- b_1 – representa el número de *huecos* en la estructura.
- b_2 – representa el número de *vacíos* en la estructura.

Para poder obtener cada uno de estos números para un grafo, se debe construir una *nube de puntos*. Esta nube estará compuesta únicamente por los nodos del grafo, prescindiendo de los enlaces. A continuación se crean esferas con centro en cada nodo, cuyo radio va aumentando en cada iteración. A medida que las esferas de nodos distintos intersecan, se crea un enlace entre dichos nodos. Cada una de estas conexiones (entre dos o más nodos) será un *componente* de la estructura. De esta forma, cuando el radio de las esferas sea mínimo, no existirá ningún componente. Mientras que cuando el radio sea máximo, se acabará con un solo componente.

Para poder contar *huecos* y *vacíos* dentro de la estructura se deben definir las piezas que construirán un objeto sólido a partir de las conexiones. En este caso, se toman triángulos y tetraedros. De esta forma, los *huecos* y *vacíos* se formarán a partir de *componentes* cerrados distintos a estas estructuras.

A medida que varía el tamaño de los radios de las esferas, se configuran los valores de los distintos números de Betti. Haciendo uso de este método, se construyen gráficas similares a la que se muestra en la *Figura 3 (*ejemplo extraído directamente de [19]), donde el eje X muestra el tamaño del radio y cada barra denota el tiempo de vida de cada tipo de *componente* (desde que se crea hasta que deja de existir). Estas gráficas se denominan ***códigos de barras*** asociados a la nube de puntos.

Por lo tanto, para cada objeto se obtendrán tres tipos de barras: para componentes (elementos de dimensión 1), huecos (elementos de dimensión 2) y vacíos (elementos de dimensión 3).

A partir de estos conjuntos de barras obtenemos una representación de la geometría del grafo que podremos utilizar para compararlo con otros grafos. Para hacer esto se utiliza una variante de la similitud de Jaccard, mostrada a continuación. Se debe tener en cuenta que la similitud de Jaccard solo se puede calcular para cada par de barras. Además, para toda barra perteneciente a un conjunto, existirá una barra en el otro conjunto donde la similitud sea máxima. Teniendo esto en cuenta, una posible medida de similitud (para *componentes, huecos* y *vacíos*) podría ser la media de estas similitudes máximas, como se indica en la siguiente fórmula:

$$S_{BO}(A,B) = \frac{1}{|A|+|B|}\left[\sum_{a\in A} sup_{b\in B}\frac{a \cap b}{a \cup b} + \sum_{b\in B} sup_{a\in A}\frac{a \cap b}{a \cup b}\right]$$

Teniendo en cuenta que las similitudes de Jaccard devuelven una puntuación perteneciente al intervalo [0,1], la media de las similitudes devolverá una cifra perteneciente al mismo rango.

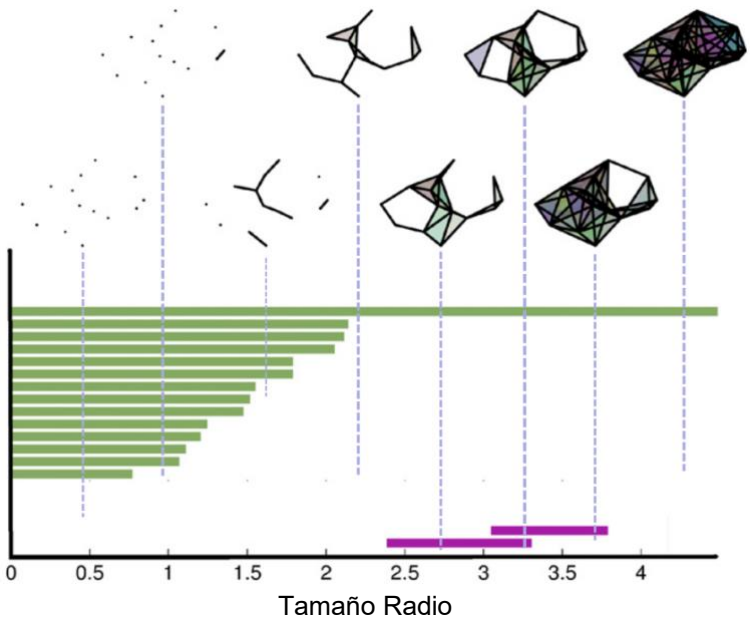


Figura 3: Ejemplo construcción de los códigos de barras. Extraída de [19].

Para los casos límite donde se tengan conjuntos vacíos, la métrica razona como un conjunto vacío sólo podría tener una similitud mayor a cero en el caso de compararlo con otro conjunto vacío, donde además, la similitud sería máxima. La formalización de esta regla se muestra a continuación:

$S_{BOE}(A,B) = S_{BO}(A,B)$	$si\ A \neq \emptyset\ y\ B \neq \emptyset$
$S_{BOE}(A,B) = 1$	$si\ A = \emptyset\ y\ B = \emptyset$
$S_{BOE}(A,B) = 0$	$si\ (A \neq \emptyset\ y\ B = \emptyset)\ o\ (A = \emptyset\ y\ B \neq \emptyset)$

Tras el cálculo de las similitudes para los conjuntos de barras de *componentes*, *huecos y vacíos*, se debe reducir estas tres medidas para obtener la medida de similitud global. Normalmente, como en [19], es común utilizar la media de las tres al ser una solución sencilla y rápida de calcular. Se obtendrá una similitud perteneciente al intervalo [0,1].

4 Experimentación

Distinguimos tres etapas principales en el desarrollo del trabajo. Una primera, donde se seleccionan los datos a utilizar y sus correspondientes fuentes. Seguidamente, se realiza una preparación y filtrado de estos datos, y se construyen los modelos de recomendación, listos para ser entrenados. Por último, se realiza el procesamiento de los datos con los modelos descritos, junto a una etapa de validación. Se describe con detalle cada uno de estos pasos en los siguientes apartados, además de la arquitectura del código del trabajo. Asimismo, se incorpora una propuesta novedosa donde proponemos utilizar la similitud topológica junto a los modelos translacionales como sistema de recomendación.

4.1 Obtención y Preprocesamiento de datos.

Como se explica en [3], durante los últimos años, la inclusión de datos externos junto a los conjuntos de entrenamiento ha mostrado una mejora en los resultados de los modelos de recomendación. Esto supone una fusión de dos o más fuentes de datos con el objetivo de formar un único conjunto sobre el que se pueda entrenar a los modelos.

Para los modelos translacionales, cuyas representaciones de entidades y relaciones se construyen a partir del resto del grafo (del resto de entidades y relaciones), un conjunto de datos con un sólo tipo de relación ("*gustar*" / "*escuchar*" / "*interesar*") como son los conjuntos para modelos de recomendación, es probable que no se obtengan resultados buenos. Por lo tanto, es imperativo enriquecer el conjunto de datos a utilizar por los modelos translacionales, con información de una fuente externa que pueda ajustar las representaciones fielmente a su naturaleza.

Una de las grandes fuentes de información para grafos de conocimiento es el proyecto *DBPedia*. Como su propia web indica, el proyecto recolecta información semántica de *Wikipedia*[5] para obtener un grafo de conocimiento de inmenso tamaño. Dentro de este grafo, encontramos subgrafos relativos a los ítems de nuestros conjuntos de datos. Por ejemplo, en el caso de que el ítem sea un cantante podremos averiguar sus años en activo, lugar y fecha de nacimiento, género musical, etc.

En cuanto a los conjuntos de datos de recomendación, se utilizarán dos. El primero de ellos es *MovieLens 1M*, un conjunto para la evaluación de sistemas de recomendación que contiene más de un millón de reseñas por distintos usuarios de aproximadamente 4000 películas. Este conjunto de datos servirá como punto de referencia al proceso de reproducibilidad de [3]. El segundo conjunto es *LastFM*, que recoge el número de minutos escuchados por distintos usuarios a una serie de artistas. Utilizamos *LastFM* con el objetivo de validar el estudio de modelos translacionales en [3], añadiendo un grado de dificultad al utilizar un conjunto de recomendación basado en el volumen de consumo (*rating implícito*) en vez de las opiniones directas de los usuarios (*rating explícito*).

La siguiente tabla muestra un resumen del contenido de cada uno de estos conjuntos:

[5] Wikipedia. https://en.wikipedia.org/wiki/

Conjunto	Ratings	Usuarios	Items
Movielens 1M	1000209	6040	3226
LastFM	17559530	1883	9518

Tabla 2: Estadísticas básicas de cada conjunto.

4.1.1 *MovieLens*.

La base de datos *MovieLens 1M* descargada[6] contiene tres ficheros: un fichero con las reseñas de los usuarios (*ratings.dat*), un fichero con información sobre usuarios (*users.dat*) y un fichero con información sobre películas (*movies.dat*). Cada uno de estos ficheros contiene registros con la siguiente información:

- *ratings.dat* - ID del usuario, ID de la película, *rating* del usuario (puntuación sobre 5) y *timestamp*.
- *users.dat* - ID, género, edad y código postal del usuario.
- *movies.dat* - ID, título y lista de géneros de la película.

Por otro lado, el repositorio[7] proporcionado en [3] facilita ficheros extraídos de *DBpedia,* que contienen información sobre las distintas películas. Cada fichero relaciona películas de *movies.dat* con atributos como el guionista, director, compositor musical, etc., incluyendo un fichero por atributo. Además, se facilita un fichero (*mappings.tsv*) que permite mapear los IDs de las películas en DBpedia con su equivalente ID en los datos de *Movielens 1M*. Esto simplifica la unión de ambos conjuntos de datos.

Siguiendo con la experimentación del artículo de *Palumbo et al.,* extraemos 7 atributos por película: director, actor, distribuidora, guionista, compositor musical, productor y cinematógrafo. Por lo tanto, se tendrán 8 relaciones entre entidades en el grafo final: las 7 anteriores más la relación de "*relevancia*" de los usuarios sobre con las películas. Debemos tener en cuenta que pueden existir más de una tripleta para una película con un atributo o incluso que *DBpedia* no tenga valor relativo a esa película. Por ejemplo, una película puede estar coproducida por dos directores o puede no tener compositor musical.

Como se detalla en [3], se acepta como película relevante para un usuario toda película con un rating igual o superior a 4. En cualquier otro caso, el ítem se considera irrelevante para el usuario.

Finalmente, realizamos la separación de datos en conjuntos de entrenamiento (*train2id.txt*), prueba (*test2id.txt*) y validación con los siguientes porcentajes de datos sobre el conjunto total:

- Conjunto de entrenamiento: 70%
- Conjunto de prueba: 20%
- Conjunto de validación: 10%

[6] https://grouplens.org/datasets/movielens/1m/

[7] https://github.com/D2KLab/entity2rec

Es importante ver cómo esta separación de datos se realiza sobre el conjunto de relevancia de los usuarios sobre las películas, al ser el tipo de tripleta que se trata de predecir. El resto de tipos de tripletas son información auxiliar que añadimos al conjunto de entrenamiento.

La separación se realiza de manera pseudo-aleatoria mediante un método (*ramdom.shuffle*) de la librería *numpy* de Python. La *Tabla 3* resume el contenido de cada conjunto.

Fichero	**Número de Tripletas**	**Número Tripletas *relevancia***	**Número Tripletas *no relevancia***
train2id.txt	413325	380193	33132
test2id.txt	108627	108627	0
val2id.txt	54313	54313	0

Tabla 3: Estadísticas de los conjuntos de datos de entrenamiento y test MovieLens.

4.1.2 *LastFM*.

La base de datos descargada de *LastFM* contiene dos ficheros: un fichero con información sobre las escuchas mensuales de distintos usuarios para un conjunto de artistas (*usersha1-artmbid-artname-plays.tsv*) y un fichero con información adicional sobre cada usuario (*usersha1-profile.tsv*), aunque este último fichero no lo utilizaremos. La información que contienen los ficheros es la siguiente:

- *usersha1-artmbid-artname-plays.tsv* - ID del usuario, ID del artista, nombre del artista y número de escuchas.
- *usersha1-profile.tsv* - ID del usuario, género, edad, país, fecha de inscripción en *LastFM*.

Como en el caso de MovieLens, el repositorio de [3] vuelve a facilitar ficheros con información sobre los artistas (como género musical, bandas a la que pertenece, etc.). Esta información proviene de DBpedia. Además, se vuelve a contar con un fichero (*mappings.tsv*) que mapea los IDs de los artistas con su correspondiente ID de DBpedia.

En este caso se extraen 9 atributos de DBpedia relacionados a cada artista musical: actos asociados (combinación de bandas asociadas y artistas asociados), banda a la que pertenece, género musical, ciudad natal, lugar de nacimiento, instrumento musical, ocupación, discográficas y sujeto de canciones. Como en el caso anterior, los artistas podrán no tener un valor para cada atributo.

Al contrario que en el caso de *MovieLens*, *LastFM* no contiene ratings explícitos de los usuarios, sino el número de escuchas para cada artista y tema. Por ello es necesario hacer una conversión del número de escuchas a ratings implícitos del usuario. En la sección 3.4.1 de [24], se describen métodos para obtener ratings implícitos de los usuarios a partir de un número de escuchas. Para este trabajo, se utiliza uno de estos procedimientos. Se comienza calculando la función de distribución acumulada complementaria (*CCDF*) del número de

escuchas de cada usuario a partir del número total por ranking de escuchas. Un ejemplo gráfico se muestra en la *Figura 4* para el usuario 2.

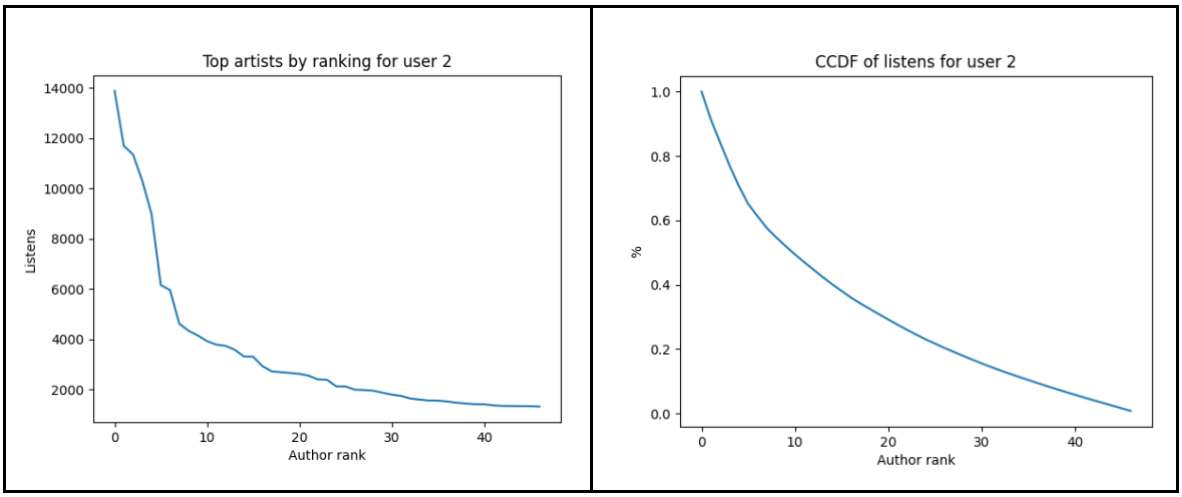


Figura 4: Gráficas escuchas totales y CCDF usuario 2.

A partir del CCDF de cada usuario, se pueden obtener ratings para cada artista de los usuarios según en qué intervalo se encuentre de la distribución. Utilizando los mismos criterios de *ratings* que en *MovieLens*, los *ratings* implícitos pertenecerán al intervalo de 1 a 5. Por lo tanto, los artistas dentro del rango 80%-100% de la CCDF obtendrá un rating de 5, los artistas dentro del 60%-80% obtendrán un rating de 4, etc. Así pues, los artistas dentro del rango 60%-100% serán considerados relevantes para el usuario en cuestión. En el Anexo B de la memoria se incluye el código de la función encargada de este proceso.

Para la evaluación, la separación final del conjunto de datos será equivalente a la empleada para la base de datos *MovieLens*:

- Conjunto de entrenamiento: 70%
- Conjunto de prueba: 20%
- Conjunto de validación: 10%

Como en el conjunto de datos anterior, las tripletas auxiliares serán incluidas exclusivamente en el conjunto de entrenamiento. La separación que se muestra anteriormente se realiza sobre las tripletas que representan la relación de relevancia entre usuarios y artistas.

Fichero	**Número de Tripletas**	**Número Tripletas *relevancia***	**Número Tripletas *no relevancia***
train2id.txt	186052	7709	178343
test2id.txt	2203	2203	0
val2id.txt	1101	1101	0

Tabla 4: Estadísticas de los conjuntos de datos de entrenamiento y test LastFM.

4.2 Arquitectura del código.

Es importante explicar la arquitectura del código desarrollado para este trabajo, ya que permitirá obtener una visión global del funcionamiento de los modelos y facilitará una posible reproducibilidad futura. En los anexos se muestra parte del código desarrollado, como se irá indicando. En el caso de querer inspeccionar con mayor detalle, recomendamos consultar el repositorio[8] del trabajo, dentro del directorio `src`.

Los procedimientos de preprocesamiento explicados en el apartado anterior se encuentran en los ficheros *preprocess_ML.py* (*MovieLens*) y *preprocess_LFM.py* (*LastFM*), que se incluyen en los Anexos A y B respectivamente.

La creación, entrenamiento y validación de los modelos translacionales se hace posible mediante la librería *OpenKE*[9]. Aunque la librería permite entrenar y validar distintos modelos translacionales, hemos visto necesaria la modificación de varios de sus ficheros para adaptar la ejecución a cómo se detalla en [3]. En el Anexo C podemos encontrar los detalles de estas modificaciones.

Antes de explicar estas modificaciones, es importante detallar el proceso de entrenamiento y validación que siguen los modelos translacionales del módulo *OpenKE*. En el Anexo D se incluye el contenido del fichero *transe_main.py*, que contiene el entrenamiento y prueba del modelo translacional *TransE*. Para los modelos *TransR* y *TransH* elaboramos los ficheros (*transr_main.py, transh_main.py*) casi idénticos, que difieren en modificaciones que se indican en el mismo Anexo D. Los ejecutables mostrados en dicho apartado muestran la configuración necesaria para usar el conjunto de *MovieLens*. Para utilizar el conjunto de *LastFM*, solo será necesario cambiar la variable "*dataset*" (con el valor "*LastFM*").

Cada ejecución se divide en cuatro etapas:

1. Creación de los *data loaders*: cargar los ficheros del conjunto de datos de entrenamiento (junto a validación) y prueba.
2. Creación de los modelos translacionales y su correspondiente función de pérdida.
3. Ejecución del entrenamiento de los modelos mediante un objeto *Trainer*.
4. Prueba de los modelos entrenados con el conjunto de prueba mediante un objeto *Tester*.

Por otro lado, es importante indicar los parámetros de entrenamiento y validación que se utilizan para cada uno de los modelos, con el fin de facilitar la reproducibilidad de la experimentación realizada. Al igual que en [3], se entrena cada modelo para dimensiones 10, 20, 30, 50, 100 y 200 con los siguientes parámetros:

- Número de lotes (*batches*) = 100
- Gamma (γ) = 1
- Tasa de aprendizaje = 0.001
- Número de épocas en entrenamiento = 1000.

[8] https://github.com/ebjaime/Trabajo-Final-de-Grado

[9] https://github.com/thunlp/OpenKE

Para el cálculo de las métricas de los modelos base SVD y *MostPop* se utiliza el código propuesto por el respositorio del framework *Cornac*[10], que no requiere ninguna modificación al incluir el cálculo de las métricas mencionadas anteriormente. Esta es solo una de las múltiples opciones que existen que implementen estos modelos.

4.3 Propuesta código de barras.

Como se ha explicado anteriormente, la similitud topológica de los códigos de barras miden la similitud entre dos nubes de puntos. Por otro lado, los sistemas de recomendación obtienen un ranking de ítems para cada usuario. Ante estas dos reflexiones, nos preguntamos si cabe la posibilidad de usar esta similitud para crear un ranking de ítems en el caso de extraer una nube relativa a estos. A continuación presentamos una posible estrategia para aplicar esta vía.

A la hora de probar los modelos, se ha de construir un ranking para cada usuario de todos los ítems presentes en los conjuntos de entrenamiento y test que no hayan sido marcados como relevantes. Por lo tanto, para cada usuario tendremos dos conjuntos de ítems: I_u(ítems relevantes para usuario u) e $\overline{I_u}$(ítems no relevantes o no revisados por usuario u). Además, a partir de estos conjuntos podemos obtener el conjunto de usuarios a los que les ha parecido relevante cada ítem: U_i (usuarios interesados a los que ítem i les parece relevante) e $\overline{U_i}$ (usuarios interesados a los que ítem i no les parece relevante).

Como se ha mencionado anteriormente, los modelos translacionales tienen la capacidad de transformar a las entidades y enlaces que componen un grafo de conocimiento en vectores de dimensión *d*. Por lo tanto, tras una etapa de entrenamiento, cada usuario del grafo podrá representarse con un vector. Esto supone que el conjunto definido anteriormente U_i estará compuesto por una serie de vectores. Dicho de otro modo, a cada ítem del conjunto de datos le corresponde una nube de puntos compuesta por las representaciones de los usuarios a los que dicho ítem les ha parecido relevante.

Como se explica en la sección 3, el método relativo a la similitud S_{BOE} permite calcular la similitud entre dos nubes de puntos, por lo que existe la posibilidad de obtener una similitud entre dos ítems del conjunto. Pese a que cada nube de puntos pueda tener un número desigual de vectores (un ítem puede ser considerado relevante por más usuarios que otro), el método S_{BOE} permite calcular la similitud a pesar de ello.

Volviendo al comienzo de esta sección, el objetivo de los sistemas de recomendación es obtener un ranking de ítems para cada usuario. Por lo que para cada ítem i perteneciente a $\overline{I_u}$ (ítems no relevantes o no revisados por u) una posible puntuación del ranking puede ser:

$$P(i) = \prod_{j \in I_u} S_{BOE}(U_i, U_j) \in [0,1]$$

De este modo, la puntuación que se le asigne a cada ítem vendrá dada por su similitud con el resto de ítems relevantes al usuario. A su vez, estas nubes de puntos se forman a partir de

[10] https://github.com/PreferredAI/cornac

las coordenadas de los usuarios, por lo que la metodología podría ser contextualizada como un método de filtrado colaborativo.

A continuación, se incluye un pseudocódigo que muestra de manera más clara el proceso propuesto en este apartado:

```
def ranking (User u):
      relevant_items = get_related_items(u);
      irrelevant_items = get_all_items(u) - relevant_items;
      rank = []
      for irr_item in irrelevant_items:
            irr_nube_ptos = nube_puntos(irr_item);
            total = 1;
            for r_item in relevant_items:
                  r_nube_ptos = nube_puntos(r_item);
                  total = total * s_boe(r_nube_ptos, irr_nube_ptos);

            rank.add(total);

      return rank;

def nube_puntos(Item i):
      users = [];
      for user in get_all_users();
            if i in get_relevant_items(u):
                  user.add(get_vector(u));

      return users;
```

Por restricciones de tiempo en la realización de este trabajo, este método no ha sido evaluado de forma empírica. Creemos que puede ser una alternativa muy novedosa para recomendación, y así lo dejaremos indicado como trabajo futuro.

5 Resultados

Tras la construcción y entrenamiento de los modelos, vamos a comparar nuestros resultados con los que se obtienen en [3] para el dataset *MovieLens*. Además, verificaremos que también se obtienen buenos resultados para el conjunto de *LastFM*. Primero será necesario resumir las métricas utilizadas para validar nuestros modelos. Después, presentaremos los resultados obtenidos para *MovieLens* y *LastFM* con los modelos translacionales *TransE*, *TransH* y *TransR*, para dimensiones 10, 20, 30, 50, 100 y 200. Además, como en [3], los resultados de los modelos translacionales serán comparados con algunos modelos de recomendación clásicos, como *SVD*, *MostPop* o un modelo aleatorio.

5.1. Evaluación y métricas.

Es importante reiterar que para la evaluación de los modelos, a la hora de crear los rankings de recomendación de ítems, todos los ítems que no han sido considerados relevantes para el usuario (tanto cuando no lo ha consumido como cuando ha recibido una puntuación inferior a 4) formarán parte del ranking. Hay que recordar que para este segundo caso, donde el ítem ha recibido una puntuación negativa (*< 4*) por el usuario, la información no ha sido incluida en el conjunto final de datos ya que las tripletas entre usuarios e ítems donde no existe una relación de relevancia se descartan.

Además, también hay que recordar que en el caso de recomendar un ítem y este no aparezca en el conjunto de test, será considerado como recomendación negativa. Esta evaluación asegura el peor de los casos en el que a un usuario no le gustara ningún ítem fuera de los que le gustan en el conjunto de test. En la realidad, este enfoque es poco realista, pero procura obtener una evaluación prudente y cautelosa.

Las métricas utilizadas para la evaluación de los modelos incluyen algunas métricas clásicas para evaluar sistemas de recomendación, como *Precision@k*, *Recall@k*, *Mean Average Precision* (*MAP*) y *Normalized Discounted Cumulative Gain* (*NDCG*). Aparte de estas, se incluye la métrica *Serendipity@k*, que trata de representar el acierto de las recomendaciones, descartando los ítems más relevantes entre los usuarios. Tiene una fórmula idéntica a *Precision@k*, con la modificación de que los *k* ítems más populares del conjunto pasan a ser no relevantes para el usuario. A continuación, se incluyen las métricas junto con sus correspondientes expresiones:

- ***P@k***

$$\frac{\#\ de\ ítems\ relevantes\ entre\ top\ k\ recomendados}{k}$$

- ***R@k***

$$\frac{\#\ de\ ítems\ entre\ top\ k\ recomendados\ relevantes}{\#\ total\ de\ ítems\ relevantes}$$

- ***SER@k***

  $\boldsymbol{P@k}$ *con top k ítems más populares del conjunto como no relevantes.*

- ***MAP***

$$\frac{1}{\#\ total\ de\ ítems\ relevantes} \sum_{i \in I_r} \frac{\#\ ítems\ relevantes\ hasta\ pos.i\ en\ ranking}{Pos.i\ en\ ranking}$$

- ***NDCG***

$$NDCG = \frac{DCG}{IDCG}$$

siendo

$$DCG = \sum_{i=1}^{I_r} \frac{rel\ (i)}{log\ (i+1)}$$

$$IDCG = \sum_{i=1}^{I_r} \frac{2^{\mathrm{rel(i)}} - 1}{log\ (i+1)}$$

En el Anexo C encontramos las modificaciones necesarias para obtener la implementación de estas métricas dentro de la librería *OpenKE*.

5.2. MovieLens.

A continuación se presentan los resultados obtenidos para el conjunto de datos de *MovieLens*. La Tabla 5 muestra la comparación entre los modelos translacionales (con dimensión 100) y algunos de los modelos de recomendación base como *SVD* y *MostPop*. Además, se añaden los resultados obtenidos con un modelo aleatorio de recomendación.

Modelo	P@5	P@10	R@5	R@10	SER@5	SER@10	MAP	NDCG
TransE	**0.20035**	**0.16702**	**0.07802**	**0.12371**	**0.18953**	**0.15546**	**0.11674**	**0.45521**
TransH	0.19331	0.16212	0.07527	0.12053	0.18548	0.15071	0.10430	0.43532
TransR	0.16889	0.14451	0.06734	0.10744	0.16422	0.14119	0.09953	0.41633
MostPop	0.16122	0.13598	0.06294	0.10125	0.04235	0.03565	0.09760	0.40445
SVD	0.04519	0.03442	0.01765	0.02563	0.03828	0.02960	0.01566	0.28762
Random	0.00851	0.00621	0.00033	0.00458	0.00635	0.00602	0.00732	0.25122

Tabla 5: Comparación entre modelos translacionales y sistemas base de filtrado colaborativo en MovieLens.

Comparando los resultados con el artículo original de *Palumbo et al*, se observan unos resultados muy similares. *TransE* es el modelo que mejores resultados obtiene, seguido de cerca por *TransH*. Para todos los modelos, exceptuando al modelo aleatorio, se obtienen

resultados ligeramente peores a los que se presentan en [3]. Esta ligera diferencia es probable que sea causada por la separación distinta del conjunto de datos al experimento original. Es probable que realizando la separación de datos aleatoria nuevamente, los resultados vuelvan a modificarse levemente.

Dentro de estos pequeños cambios, se puede observar cómo el empeoramiento más notable se da en el modelo *TransR*, aunque el cambio es muy pequeño. Por otro lado, el modelo aleatorio sufre una mejora superior al 50% en algunas métricas, aunque el crecimiento no tiene ninguna base. Son recomendaciones puramente casuales.

En el Anexo E se añade la tabla de resultados que muestra la comparación entre los modelos translacionales para distintas dimensiones en el *embedding*. En ella podemos observar, una vez más, cómo los resultados obtenidos son muy similares a los que se detallan en [3]. Pero, al contrario que el artículo original, el modelo *TransE* obtiene mejores resultados para la dimensión 100, junto a los modelos *TransR* y *TransH*. En algunos casos, como para dimensiones de bajo nivel, según aumenta la dimensión se observa una mejora para los tres modelos. Pero al llegar a dimensiones superiores, donde $d > 50$, los modelos obtienen resultados muy similares.

Los resultados presentados para el conjunto de datos *MovieLens*, permiten validar la experimentación de [3] y confirmar que los modelos translacionales pueden ser utilizados como sistemas de recomendación al obtener resultados superiores a los de algunos modelos base de recomendación de ítems.

5.3. LastFM.

Al contrario que para *MovieLens*, para el conjunto de datos *LastFM* no existen resultados con los que comparar los obtenidos. A pesar de ello, habiendo visto los resultados del apartado anterior y los resultados en [3] para el conjunto de datos *LibraryThing*[11], obtenemos unos valores semejantes. Los resultados se presentan en la Tabla 6, con los modelos translacionales con dimensión 100.

Modelo	**P@5**	**P@10**	**R@5**	**R@10**	**SER@5**	**SER@10**	**MAP**	**NDCG**
TransE	**0.14520**	**0.11339**	**0.40314**	**0.52739**	**0.12430**	**0.10051**	**0.10494**	**0.46231**
TransH	0.12531	0.10641	0.34803	0.47293	0.11092	0.09512	0.09381	0.43789
TransR	0.08214	0.08053	0.25668	0.39283	0.07631	0.06592	0.08004	0.32860
MostPop	0.11844	0.09731	0.33571	0.43738	0.05412	0.04694	0.09148	0.36919
SVD	0.06312	0.05104	0.17412	0.25456	0.05521	0.04219	0.06047	0.19185
Random	0.01215	0.00951	0.03351	0.04518	0.01139	0.00848	0.01407	0.15501

Tabla 6: Comparación entre modelos translacionales y sistemas base de filtrado colaborativo en LastFM.

[11] https://www.librarything.com/

Como se puede observar, los resultados obtenidos coinciden en su mayoría con lo que se esperaba obtener en referencia al dataset anterior. Los modelos *TransE* y *TransH* proporcionan resultados superiores al resto de modelos de recomendación. En este caso, el modelo *TransR* obtiene cifras ligeramente inferiores al sistema *MostPop*. Esto puede explicarse debido a la complejidad de este modelo frente a un conjunto de datos de menor tamaño que *MovieLens*.

En comparación con las cifras obtenidas en *MovieLens*, se observa un deterioro general del rendimiento. Pese a este deterioro, los resultados obtenidos se alinean con lo esperado: los modelos translacionales, generalmente, obtienen mejores recomendaciones en comparación con los modelos base.

En el Anexo E encontramos la tabla de resultados para las distintas dimensiones de los modelos translacionales. Al contrario que para el conjunto de *MovieLens*, los modelos translacionales *TransE* y *TransR* obtienen los mejores resultados para la dimensión 50. En cambio, el modelo *TransH* obtiene unos resultados superiores para la dimensión 100. Generalmente, para las dimensiones altas ($d > 50$) se obtienen unos resultados muy similares para todos los modelos, pero como con *MovieLens* o *LibraryThing* en [3], los resultados son óptimos cuando la dimensión no es superior a 100.

Los resultados sugieren que los modelos translacionales también producen recomendaciones superiores a los modelos base para un conjunto de recomendación no convencional basado en otra métrica distinta a los ratings: número de escuchas. Esto plantea la posibilidad de investigar si, utilizando otra métrica, se obtendrían resultados similares, como el número de visitas a una página o número de clicks.

6 Conclusiones y trabajo futuro

6.1 Conclusiones

En este trabajo hemos estudiado la reproducción, en un caso particular, de algunos algoritmos denominados translacionales y cómo estos modelos tienen un uso válido para la tarea de recomendación de ítems. Con este fin, se ha probado a utilizar dos conjuntos de datos distintos, *Movielens* y *LastFM*, obteniendo resultados satisfactorios para ambos. Seguidamente, se ha propuesto una nueva opción de evaluación para los modelos translacionales, basado en la similitud topológica de grafos.

Los objetivos que nos habíamos fijado se han cumplido en su totalidad. El trabajo realizado ha conseguido reproducir exitosamente la experimentación realizada en [3]. Además, se ha extendido este proceso, al reproducir el mismo experimento sobre otro conjunto de datos basado en una métrica distinta a los ratings de usuarios. En este segundo caso se ha traducido el volumen de escuchas de los usuarios a ratings implícitos, obteniendo resultados acordes con la hipótesis original.

Asimismo, en el trabajo se ha propuesto un modelo translacional alternativo con una función de pérdida basado en una similitud topológica conocida como "código de barras", a falta de probar el modelo empíricamente.

Como idea principal del trabajo, pretendemos hacer entender que un proceso de reproducibilidad de un experimento es un proceso complicado y enrevesado, pero de una gran importancia. La validación de una investigación permite reforzar sus hallazgos e incluso, al revisarla al máximo detalle, se da pie a una ampliación.

6.2 Trabajo futuro

Como se ha mencionado en las secciones anteriores, una de las propuestas como futuro trabajo incluye el desarrollo del modelo translacional basado en la similitud topológica de código de barras, para comprobar su funcionamiento de una manera empírica.

Otra opción incluye la repetición de la experimentación del trabajo utilizando otros modelos translacionales, como *TransG* o *TransM*, con el objetivo de comprobar si estos modelos, de una complejidad superior a los modelos utilizados en el trabajo, también obtienen buenos resultados como sistemas de recomendación.

Por otro lado, como se indica en [3], sería interesante investigar las propiedades de algunos parámetros de los modelos translacionales, como la tasa de aprendizaje o Gamma (γ), de la misma forma que con el parámetro de dimensión (d) en este trabajo.

Incluso, como se indica al final de la sección 5, sería interesante utilizar otros conjuntos de datos e intentar volver a extraer un rating implícito a partir de otras características, como en el caso de *LastFM* con el volumen de escuchas de los usuarios.

Referencias

[1] E. S. Reich, "The rise and fall of a physics fraudster," *Phys. World*, vol. 22, no. 5, pp. 24–29, May 2009.

[2] A. Bellogín, P. Castells, A. Said, and D. Tikk, "Report on the workshop on reproducibility and replication in recommender systems evaluation (RepSys)," *ACM SIGIR Forum*, vol. 48, pp. 29–35, Jun. 2014.

[3] E. Palumbo, G. Rizzo, R. Troncy, E. Baralis, M. Osella, and E. Ferro, "Translational Models for Item Recommendation," in *The Semantic Web: ESWC 2018 Satellite Events*, 2018, pp. 478–490.

[4] J. McCusker, M. Dumontier, R. Yan, S. He, J. Dordick, and D. Mcguinness, "Finding melanoma drugs through a probabilistic knowledge graph," Apr. 2016, doi: 10.7287/PEERJ.PREPRINTS.2007V1.

[5] J. H. Lee and G. H. Gonzalez, "TOWARDS INTEGRATIVE GENE PRIORITIZATION IN ALZHEIMER'S DISEASE," in *Biocomputing 2011*, pp. 4–13.

[6] H. Xiao, M. Huang, and X. Zhu, "SSP: Semantic Space Projection for Knowledge Graph Embedding with Text Descriptions," *arXiv [cs.CL]*. 2016, [Online]. Available: http://arxiv.org/abs/1604.04835.

[7] M. Chen, Y. Tian, M. Yang, and C. Zaniolo, "Multilingual Knowledge Graph Embeddings for Cross-Lingual Knowledge Alignment," in *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, 2017, pp. 1511–1517.

[8] L. Liu, K.-W. Cheung, X. Li, and L. Liao, "Aligning Users across Social Networks Using Network Embedding," Jul. 2016.

[9] B. Alharbi and X. Zhang, "Learning from Your Network of Friends: A Trajectory Representation Learning Model Based on Online Social Ties," Dec. 2016, doi: 10.1109/icdm.2016.0090.

[10] S. Cavallari, V. Zheng, H. Cai, K. Chang, and E. Cambria, "Learning Community Embedding with Community Detection and Node Embedding on Graphs," Nov. 2017, pp. 377–386.

[11] H. Fang, F. Wu, Z. Zhao, X. Duan, Y. Zhuang, and M. Ester, "Community-Based Question Answering via Heterogeneous Social Network Learning," *Proc. Conf. AAAI Artif. Intell.*, vol. 30, no. 1, Feb. 2016, [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/9972.

[12] Z. Zhao, Q. Yang, D. Cai, X. He, and Y. Zhuang, "Expert Finding for Community-Based Question Answering via Ranking Metric Network Learning," in *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, 2016, pp. 3000–3006.

[13] M. Ochi *et al.*, "Representation Learning for Geospatial Areas Using Large-Scale Mobility Data from Smart Card," in *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct*, Heidelberg, Germany, 2016, pp. 1381–1389.

[14] M. Ochi, Y. Nakashio, M. Ruttley, J. Mori, and I. Sakata, "Geospatial Area Embedding Based on the Movement Purpose Hypothesis Using Large-Scale Mobility Data from Smart Card," *Int'l J. of Communications, Network and System Sciences*, vol. 09, pp. 519–534, 2016.

[15] H. Wang *et al.*, "Knowledge-aware Graph Neural Networks with Label Smoothness Regularization for Recommender Systems," *arXiv [cs.LG]*. 2019, [Online]. Available: http://arxiv.org/abs/1905.04413.

[16] H. Cai, V. W. Zheng, and K. C.-C. Chang, "A Comprehensive Survey of Graph Embedding: Problems, Techniques and Applications," *arXiv [cs.AI]*. 2018, [Online]. Available: http://arxiv.org/abs/1709.07604.

[17] P. Goyal and E. Ferrara, "Graph Embedding Techniques, Applications, and Performance: A Survey," *arXiv [cs.SI]*. 2017, [Online]. Available: http://arxiv.org/abs/1705.02801.

[18] S. Wang *et al.*, "Graph Learning Approaches to Recommender Systems: A Review," *arXiv [cs.IR]*. 2020, [Online]. Available: http://arxiv.org/abs/2004.11718.

[19] G. Máté, A. Hofmann, N. Wenzel, and D. W. Heermann, “A topological similarity measure for proteins,” *Biochimica et Biophysica Acta (BBA) - Biomembranes*, vol. 1838, no. 4, pp. 1180–1190, 2014.
[20] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko, “Translating Embeddings for Modeling Multi-relational Data,” in *Advances in Neural Information Processing Systems*, 2013, vol. 26, [Online]. Available: https://proceedings.neurips.cc/paper/2013/file/1cecc7a77928ca8133fa24680a88d2f9-Paper.pdf.
[21] Z. Wang, J. Zhang, J. Feng, and Z. Chen, “Knowledge Graph Embedding by Translating on Hyperplanes,” in *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014, pp. 1112–1119.
[22] Y. Lin, Z. Liu, M. Sun, Y. Liu, and X. Zhu, “Learning entity and relation embeddings for knowledge graph completion,” 2015.

[23] Ò. Celma, ‘Music Recommendation’, in *Music Recommendation and Discovery: The Long Tail, Long Fail, and Long Play in the Digital Music Space*, Ò. Celma, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 43–85.

Glosario

Grafo de conocimiento.	Base de conocimiento que se encarga de enlazar entidades mediante propiedades semánticas.
Graph Embedding.	Técnica de transformación de un grafo en vectores de dimensión inferior a la original, manteniendo las propiedades iniciales.
Modelos translacionales.	Modelos matemáticos que efectúan operaciones de graph embedding sobre grafos de conocimiento mediante la minimización de su función de pérdida.
Descenso de gradiente	Algoritmo de optimización que permite obtener cuándo una función obtiene sus valores más bajos.

Anexos

A preprocess_ML.py

```
"""
Created on Fri Jan  29 11:26:45 2021

@author: Jaime Enriquez Ballesteros, @ebjaime
"""

"""
@inproceedings{han2018openke,
  title={OpenKE: An Open Toolkit for Knowledge Embedding},
  author={Han, Xu and Cao, Shulin and Lv Xin and Lin, Yankai and Liu, Zhiyuan and Sun, Maosong and Li, Juanzi},
  booktitle={Proceedings of EMNLP},
  year={2018}
}
"""

import os
import pandas as pd
import numpy as np

dataset = "Movielens1M"
train_pct = 0.7
val_pct = 0.1
test_pct = 0.2


# Lectura de atributos
dbo_directors = pd.read_csv("../data/%s/graphs/dbo:director.edgelist" % dataset,
                            header=None,
                            sep=" ")
dbo_directors.columns = ["Movie", "Director"]
dbo_directors = dbo_directors[["Director", "Movie"]]
```

```
dbo_starring = pd.read_csv("../data/%s/graphs/dbo:starring.edgelist"% dataset,
                               header=None,
                               sep=" ")
dbo_starring.columns = ["Movie", "Actor"]
dbo_starring = dbo_starring[["Actor", "Movie"]]

dbo_distributor = pd.read_csv("../data/%s/graphs/dbo:distributor.edgelist"% dataset,
                               header=None,
                               sep=" ")
dbo_distributor.columns = ["Movie", "Distributor"]
dbo_distributor = dbo_distributor[["Distributor","Movie"]]

dbo_writer = pd.read_csv("../data/%s/graphs/dbo:writer.edgelist"% dataset,
                               header=None,
                               sep=" ")
dbo_writer.columns = ["Movie", "Writer"]
dbo_writer = dbo_writer[["Writer", "Movie"]]

dbo_musicComposer = pd.read_csv("../data/%s/graphs/dbo:musicComposer.edgelist"% dataset,
                               header=None,
                               sep=" ")
dbo_musicComposer.columns = ["Movie", "MusicComposer"]
dbo_musicComposer= dbo_musicComposer[["MusicComposer", "Movie"]]

dbo_producer = pd.read_csv("../data/%s/graphs/dbo:producer.edgelist"% dataset,
                               header=None,
                               sep=" ")
dbo_producer.columns = ["Movie", "Producer"]
dbo_producer = dbo_producer[["Producer", "Movie"]]

dbo_cinematography = pd.read_csv("../data/%s/graphs/dbo:cinematography.edgelist"% dataset,
                               header=None,
                               sep=" ")
```

```
dbo_cinematography.columns = ["Movie", "Cinematography"]
dbo_cinematography = dbo_cinematography[["Cinematography", "Movie"]]


# Lectura de reseñas
feedback = pd.read_csv("../data/%s/original/ratings.dat" % dataset,
                       header=None,
                       sep="::")
feedback.columns = ["UserID", "MovieID", "Rating", "Timestamp"]

# Traduccion MovieID a Movie en reseñas
mappings = pd.read_csv("../data/%s/original/mappings.tsv" % dataset,
                       header=None,
                       sep="\t")
mappings.columns = ["MovieID", "MovieTitle", "Movie"]

feedback = feedback.merge(mappings[["MovieID", "Movie"]], on="MovieID")

# Filtracion de solo ratings>=4
feedback_relevant = feedback.drop(feedback.loc[feedback.Rating<4].index)
feedback_relevant.drop(["Rating", "Timestamp", "MovieID"], axis=1, inplace=True)

# Creacion entity2id.txt y relation2id.txt
directors = dbo_directors["Director"].unique()
actors = dbo_starring["Actor"].unique()
distributors = dbo_distributor["Distributor"].unique()
writers = dbo_writer["Writer"].unique()
musicComposers = dbo_musicComposer["MusicComposer"].unique()
producers = dbo_producer["Producer"].unique()
cinematographies = dbo_cinematography["Cinematography"].unique()

users = feedback["UserID"].unique()
movies = {*feedback["Movie"].unique(), *dbo_directors["Movie"].unique(), *dbo_starring["Movie"].unique(),
          *dbo_distributor["Movie"].unique(),*dbo_writer["Movie"].unique(), *dbo_musicComposer["Movie"].unique(),
          *dbo_producer["Movie"].unique(), *dbo_cinematography["Movie"].unique()}


relations_list = ["feedback", "dbo:director", "dbo:starring", "dbo:distributor", "dbo:writer", "dbo:musicComposer",
                  "dbo:producer", "dbo:cinematography"]
relations = {}
with open("../data/%s/relation2id.txt" % dataset, "w") as f:
    f.write(str(len(relations_list))+"\n")
    id=0
    for relation in relations_list:
        f.write(str(relation)+"\t"+str(id)+"\n")
        relations[str(relation)] = id
        id+=1

entities_list = {*users, *directors, *actors, *distributors, *writers, *musicComposers,
                 *producers, *cinematographies, *movies}
entities = {}
with open("../data/%s/entity2id.txt" % dataset, "w") as f:
    f.write(str(len(entities_list))+"\n")
    id=0
    for entity in entities_list:
        f.write(str(entity)+"\t"+str(id)+"\n")
        entities[str(entity)] = id
        id+=1


# Creacion train2id.txt, test2id.txt y val2id.txt
feedback_np = feedback_relevant.values
np.random.shuffle(feedback_np)
```

```
train = {}
train["feedback"] = feedback_np[:int(train_pct*len(feedback_np))]
val = {}
val["feedback"] = feedback_np[int(train_pct*len(feedback_np)) : int(train_pct*len(feedback_np))+
                                                                int(val_pct*len(feedback_np))]
test = {}
test["feedback"] = feedback_np[int(train_pct*len(feedback_np)) + int(val_pct*len(feedback_np)):]

# Solo el conjunto de entrenamiento contendra las triplas no feedback
for rel,vals in zip(relations_list[1:], [dbo_directors.values, dbo_starring.values, dbo_distributor.values,
                                         dbo_writer.values, dbo_musicComposer.values, dbo_producer.values,
                                         dbo_cinematography.values]):
    train[rel] = vals

# Creacion de cada conjunto de datos
sets = [] # Train, test, val

for set in [train, test, val]:
    set2id = []
    for rel in set:
        for tripla in set[rel]:
            head = str(tripla[0])
            tail = str(tripla[1])
            set2id.append((str(entities[head]), str(entities[tail]), str(relations[rel])))
    sets.append(set2id)

# Inserción de cada conjunto en su fichero correspondiente
train2id =  open("../data/%s/train2id.txt" % dataset, "w")
test2id = open("../data/%s/test2id.txt" % dataset, "w")
val2id = open("../data/%s/valid2id.txt" % dataset, "w")

for i, set in enumerate([train2id, test2id, val2id]):
    set.write(str(len(sets[i]))+"\n")
    for triple in sets[i]:
        set.write(triple[0]+"\t"+triple[1]+"\t"+triple[2]+"\n")

    set.close()

# Ejecutar n_n.py para crear 1-1.txt, 1-n.txt ....
os.system("cd ../data/%s; python n-n.py" % dataset)
```

B preprocess_LFM.py

Código equivalente a *preprocess_ML.py* a excepción del cálculo de los ratings de los usuarios:

```
def plot_user_listens(user_id=2):
    plt.plot(range(len(feedback[feedback.UserID == user_id])),
             feedback[feedback.UserID == user_id].sort_values(by="Listens", ascending=False)["Listens"].values)
    plt.title("Top artists by ranking for user " + str(user_id))
    plt.ylabel("Listens")
    plt.xlabel("Author rank")
    plt.show()

# Calculo de Funcion de Distribucion Acumulada Complementaria (CCDF) para cada usuario
def ccdf_listens(feedback=feedback, user_id=2, plot=False, cutoff=0.6):
    if user_id is None:  # Caso en el que feedback ya viene como DF de un solo usuario
        listens_user = feedback.sort_values(by="Listens")["Listens"].values
    else:
        listens_user = feedback[feedback.UserID == user_id].sort_values(by="Listens")[
            "Listens"].values

    cdf = np.cumsum(listens_user / listens_user.sum())[::-1]
    if plot:
        plt.plot(range(len(cdf)), cdf)
        plt.title("CCDF of listens for user " + str(user_id))
        plt.ylabel("%")
        plt.xlabel("Author rank")
        plt.show()
    return listens_user[::-1][cdf > cutoff]


# Filtrar artistas que están dentro del mejor 40% para cada usuario
feedback_relevant_idx = feedback.groupby(["UserID"]).apply(lambda x:
                                                           x.loc[x["Listens"].isin(ccdf_listens(x, user_id=None))])

feedback_relevant = feedback.loc[feedback_relevant_idx.reset_index(level=["UserID"], drop=True).index]
feedback_relevant.drop(["Listens", "AuthorID"], axis=1, inplace=True)
```

C Modificaciones OpenKE.

Todas las modificaciones de la librería *OpenKE* son relativas a los procesos de validación y prueba de los modelos, con el objetivo de obtener una serie de métricas que la librería *OpenKE* no proporciona y que deben de ser calculadas.

C.1 *openke/config/Tester.py*

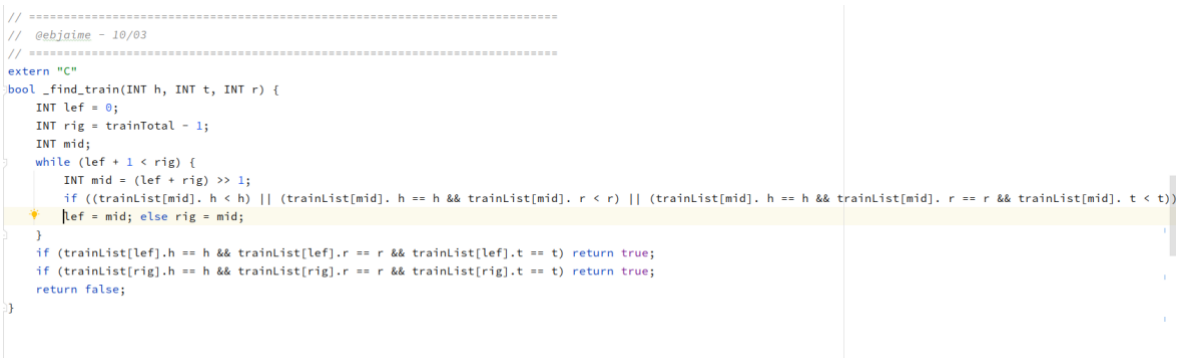
```
def run_link_prediction(self, type_constrain = False):
    self.lib.initTest()
    self.data_loader.set_sampling_mode('link')
    if type_constrain:
        type_constrain = 1
    else:
        type_constrain = 0

    training_range = tqdm(self.data_loader)

    num_relevant_documents_in_test = self.lib.getTestTotal()

    h_at_10 = 0
    h_at_5 = 0

    mean_avg_prec = 0

    # Obtener cinco y diez productos mas populares para calcular SER@n
    ten_most_popular, item_ids = self.k_most_popular(10)
    five_most_popular = ten_most_popular[:5]

    ser_at_5 = 0
    ser_at_10 = 0

    ndcg = 0

    for index, [data_head, data_tail] in enumerate(training_range):

        # Se seleccionan solo items que son peliculas
        data_tail["batch_t"] = item_ids

        # score = self.test_one_step(data_head)
        # self.lib.testHead(score.__array_interface__["data"][0], index, type_constrain)
        score = self.test_one_step(data_tail)
        # self.lib.testTail(score.__array_interface__["data"][0], index, type_constrain)

        # Menor score es mejor recomendacion
        score_sorted = item_ids[np.argsort(score)]


        count_unrated = 0 # Contador de items unrated
        count_relevant = 0 # Contador de items relevantes

        map_aux = 0 # Calculo de MAP intermedio

        dcg = 0 # discounted cumulative gain

        for score in score_sorted:
            # Si es unrated (no pertenece al conjunto de entrenamiento ni validación) segun AllUnratedItems
            unrated = not self.lib._find_train(int(data_head['batch_h'][0]), int(score), int(data_head['batch_r'][0])) \
                        and not self.lib._find_val(int(data_head['batch_h'][0]), int(score), int(data_head['batch_r'][0]))

            if unrated:
                count_unrated+=1
                # Si es relevante pertenece al conjunto de test
                relevant = self.lib._find_test(int(data_head['batch_h'][0]), int(score), int(data_head['batch_r'][0]))
                if relevant:
                    count_relevant+=1
                    dcg += 1/np.log2(count_unrated + 1)

                    if count_unrated <= 5:
                        h_at_5 += 1
                        if score not in five_most_popular:
                            ser_at_5 += 1

                    if count_unrated <= 10:
```

```
                    h_at_10 += 1
                    if score not in ten_most_popular:
                        ser_at_10 += 1

                map_aux += count_relevant/(count_unrated+1)

        if count_relevant != 0:
            mean_avg_prec += map_aux / count_relevant

        idcg = sum([1/np.log2(pos + 1) for pos in range(count_relevant)])
        ndcg += dcg/idcg


    p_at_5 = h_at_5 / (index * 5)
    p_at_10 = h_at_10 / (index * 10)

    r_at_5 = h_at_5 / num_relevant_documents_in_test
    r_at_10 = h_at_10 /num_relevant_documents_in_test

    mean_avg_prec /= index

    ser_at_5 /= (index * 5)
    ser_at_10 /= (index * 10)

    ndcg /= len(training_range)

    print("\n\nP@5: ",p_at_5)
    print("P@10: ",p_at_10)
    print("R@5: ",r_at_5)
    print("R@10: ",r_at_10)
    print("MAP: ", mean_avg_prec)
    print("SER@5: ",ser_at_5)
    print("SER@10: ",ser_at_10)
    print("NDCG: ", ndcg)

    return p_at_5, p_at_10, r_at_5, r_at_10, mean_avg_prec, ser_at_5, ser_at_10, ndcg
```

Se añaden el cálculo de algunas métricas típicas de sistemas de recomendación como *P@5* (*Precision at 5*), *P@10* (*Precision at 10*), *R@5* (*Recall at 5*), *R@10* (*Recall at 10*), *MAP* (*Mean Average Precision*), *SER@5* (*Serendipity at 5*), *SER@10* (*Serindipity at 10*) y *NDGC* (*Normalized Discounted Cumulative Gain*).

C.2. *openke/base/Corrupt.h*

```
// ================================================================================
//  @ebjaime - 10/03
// ================================================================================
extern "C"
bool _find_test(INT h, INT t, INT r) {
    INT lef = 0;
    INT rig = testTotal - 1;
    INT mid;
    while (lef + 1 < rig) {
        INT mid = (lef + rig) >> 1;
        if ((testList[mid]. h < h) || (testList[mid]. h == h && testList[mid]. r < r) || (testList[mid]. h == h && testList[mid]. r == r && testList[mid]. t < t))
            lef = mid; else rig = mid;
    }
    if (testList[lef].h == h && testList[lef].r == r && testList[lef].t == t) return true;
    if (testList[rig].h == h && testList[rig].r == r && testList[rig].t == t) return true;
    return false;
}

// ================================================================================
//  @ebjaime - 10/03
// ================================================================================
extern "C"
bool _find_val(INT h, INT t, INT r) {
    INT lef = 0;
    INT rig = validTotal - 1;
    INT mid;
    while (lef + 1 < rig) {
        INT mid = (lef + rig) >> 1;
        if ((validList[mid]. h < h) || (validList[mid]. h == h && validList[mid]. r < r) || (validList[mid]. h == h && validList[mid]. r == r && validList[mid]. t < t))
         lef = mid; else rig = mid;
    }
    if (validList[lef].h == h && validList[lef].r == r && validList[lef].t == t) return true;
    if (validList[rig].h == h && validList[rig].r == r && validList[rig].t == t) return true;
    return false;
```



```
// ========================================================================
//  @ebjaime - 10/03
// ========================================================================
extern "C"
bool _find_train(INT h, INT t, INT r) {
    INT lef = 0;
    INT rig = trainTotal - 1;
    INT mid;
    while (lef + 1 < rig) {
        INT mid = (lef + rig) >> 1;
        if ((trainList[mid]. h < h) || (trainList[mid]. h == h && trainList[mid]. r < r) || (trainList[mid]. h == h && trainList[mid]. r == r && trainList[mid]. t < t))
        lef = mid; else rig = mid;
    }
    if (trainList[lef].h == h && trainList[lef].r == r && trainList[lef].t == t) return true;
    if (trainList[rig].h == h && trainList[rig].r == r && trainList[rig].t == t) return true;
    return false;
}
```

Se añaden tres métodos necesarios para la búsqueda de triplas en cada conjunto de datos.

D transe_main.py, transh_main.py y transh_main.py

D.1. transe_main.py

```
"""
@inproceedings{han2018openke,
  title={OpenKE: An Open Toolkit for Knowledge Embedding},
  author={Han, Xu and Cao, Shulin and Lv Xin and Lin, Yankai and Liu, Zhiyuan and Sun, Maosong and Li, Juanzi},
  booktitle={Proceedings of EMNLP},
  year={2018}
}
"""

import os
import pandas

## OpenKE ##
from OpenKE.openke.config import Trainer, Tester
from OpenKE.openke.module.model import TransE
from OpenKE.openke.module.loss import MarginLoss
from OpenKE.openke.module.strategy import NegativeSampling
from OpenKE.openke.data import TrainDataLoader, TestDataLoader

# ================================================================================
#  PARAMETERS
# ================================================================================

# Sampling parameters (Train)
nbatches = 100 # Number of batches when sampling during training
threads = 4 #8
bern = 1
filter = 1
negative_relations = 0 # Corrupt relations are not needed

# Sampling parameters (Test)
sampling_mode = "link"
```

```
# Training parameters
dataset = "Movielens1M"
ds = [10,20,30,50, 100, 200] # Dimension
gamma = 1
lr = 0.001 # Learning rate
epochs = 1000

use_gpu=False

#Normalization parameters
p_norm = 1 # order of the norm for score
norm_flag = True # Flag for triple nomralization



# ================================================================================
#  DATA LOADERS
# ================================================================================
# Calculate number of  corrupt entities
with open("../../data/" + dataset + "/train2id.txt", "r") as f:
    num_corrupt_entities = int(f.readline())

# Dataloader for training
train_dataloader = TrainDataLoader(
    in_path = "../../data/%s/" % dataset,
    nbatches = nbatches,
    threads = threads,
    sampling_mode = "normal",
    bern_flag = bern,
    filter_flag = filter,
    # neg_ent = 25,
    neg_ent = int(num_corrupt_entities/threads), # Set number of corrupt entities equal to batch size.
                                                 # Each thread will create (batch size / threads)
```

```
    neg_rel = negative_relations)


# Dataloader for test
test_dataloader = TestDataLoader("../../data/%s/" % dataset, sampling_mode)

# ========================================================================
#  MODEL
# ========================================================================

for d in ds:
    transe = TransE(
        ent_tot = train_dataloader.get_ent_tot(),
        rel_tot = train_dataloader.get_rel_tot(),
        dim = d,
        p_norm = p_norm,
        norm_flag = norm_flag)

# ========================================================================
#  LOSS FUNCTION
# ========================================================================

    # define the loss function
    model = NegativeSampling(
        model = transe,
        loss = MarginLoss(margin = gamma),
        batch_size = train_dataloader.get_batch_size()
    )

    # ========================================================================
    #  TRAINING
    # ========================================================================
    # If the model does not yet exist
    if os.path.isfile('../checkpoint/transe_%s_d%d_e%d_lr%f.ckpt' % (dataset, d, epochs, lr)) is False:

        trainer = Trainer(model = model,
                          data_loader = train_dataloader,
                          train_times = epochs,
                          alpha = lr, # Learning rate
                          opt_method="sgd",
                          use_gpu = use_gpu)

        # Train the model
        trainer.run()
        try:
            transe.save_checkpoint('../checkpoint/transe_%s_d%d_e%d_lr%f.ckpt' % (dataset, d, epochs, lr))
        except FileNotFoundError:
            os.system("mkdir ../checkpoint")
            transe.save_checkpoint('../checkpoint/transe_%s_d%d_e%d_lr%f.ckpt' % (dataset, d, epochs, lr))


    # ========================================================================
    #  TESTING
    # ========================================================================
    # Test the model
    transe.load_checkpoint('../checkpoint/transe_%s_d%d_e%d_lr%f.ckpt' % (dataset, d, epochs, lr))
    tester = Tester(model = transe, data_loader = test_dataloader, use_gpu = True)
    tester.run_link_prediction(type_constrain = False)
    p_at_5, p_at_10, r_at_5, r_at_10, \
        mean_avg_prec, ser_at_5, ser_at_10, ndcg = tester.run_link_prediction(type_constrain = False)
    try:
        with open("results.txt", "a") as f:
            f.write("TransE\t"+str(d)+"\t"+str(p_at_5)+"\t"+str(p_at_10)+"\t"+str(r_at_5)+"\t"+str(r_at_10)+"\t"+
                    str(mean_avg_prec) + "\t" + str(ser_at_5) + "\t" + str(ser_at_10) + "\t" + str(ndcg) + "\t")
    except FileNotFoundError:
        with open("results.txt", "w") as f:
```

```
            f.write("TransE\t"+str(d)+"\t"+str(p_at_5)+"\t"+str(p_at_10)+"\t"+str(r_at_5)+"\t"+str(r_at_10)+"\t"+
                    str(mean_avg_prec) + "\t" + str(ser_at_5) + "\t" + str(ser_at_10) + "\t" + str(ndcg) + "\t")
    except FileNotFoundError:
        with open("results.txt", "w") as f:
            f.write("Model\tDimension\tp@5\tp@10\tr@5\tr@10\tmap\tser@5\tser@10\tndcg")
            f.write("TransE\t" + str(d)+"\t"+str(p_at_5)+"\t"+str(p_at_10)+"\t"+str(r_at_5)+"\t"+str(r_at_10)+"\t"+
                        str(mean_avg_prec)+"\t"+str(ser_at_5)+"\t"+str(ser_at_10)+"\t"+str(ndcg)+"\t")
```

D.2. transh_main.py

El contenido es idéntico a *transe_main.py* a excepción del cambio de modelo translacional utilizado:

```
    # define the model
    transh = TransH(
        ent_tot = train_dataloader.get_ent_tot(),
        rel_tot = train_dataloader.get_rel_tot(),
        dim = d,
        p_norm = p_norm,
        norm_flag = norm_flag)


    # define the loss function
    model = NegativeSampling(
        model = transh,
        loss = MarginLoss(margin = gamma),
        batch_size = train_dataloader.get_batch_size()
    )
```

D.3. transr_main.py

El contenido es casi idéntico a *transe_main.py*. Se añaden variables extra al comienzo del fichero y cambia el modelo utilizado para entrenar sobre los datos. Estas diferencias se muestran a continuación:

```
# ================================================================================
#  PARAMETERS
# ================================================================================

# Sampling parameters (Train)
nbatches = 100 # Number of batches when sampling during training
threads = 4 #8
bern = 1
filter = 1
negative_relations = 0 # Corrupt relations are not needed

# Sampling parameters (Test)
sampling_mode = "link"

dataset = "Movielens1M"
gamma = 1
lr = 0.001 # Learning rate
ds_e = [10,20,30,50,100,200] # Dimension entity
ds_r = [10,20,30,50,100,200] # Dimension relation
epochs = 1000
```

[…]

```
# ================================================================================
#  MODEL
# ================================================================================
for d_e, d_r in zip(ds_e, ds_r):

    # define the model
    transr = TransR(
        ent_tot = train_dataloader.get_ent_tot(),
        rel_tot = train_dataloader.get_rel_tot(),
        dim_e = d_e,
        dim_r = d_r,
        p_norm = p_norm,
        norm_flag = norm_flag,
        rand_init=rand_init)
```

E Comparación entre modelos translacionales.

E.1. MovieLens.

Modelo	d	P@5	P@10	R@5	R@10	SER@5	SER@10	MAP	NDCG
TransE	10	0.09720	0.09590	0.02236	0.04336	0.09027	0.08033	0.04236	0.37702
TransH	10	0.00753	0.01387	0.00167	0.00294	0.00515	0.00691	0.01654	0.32334
TransR	10	0.00966	0.01943	0.00335	0.00973	0.00866	0.01731	0.02651	0.30041
TransE	20	0.15885	0.15443	0.06779	0.09833	0.14964	0.13772	0.11478	0.41997
TransH	20	0.14815	0.14002	0.05339	0.09421	0.14389	0.13871	0.10030	0.42150
TransR	20	0.14684	0.13917	0.05210	0.09906	0.13997	0.12903	0.09812	0.41688
TransE	30	0.19743	0.16609	0.06858	0.12001	0.17941	0.15414	0.11563	0.45228
TransH	30	0.18941	0.16098	0.06676	0.11149	0.18642	0.15511	0.10489	0.43133
TransR	30	0.18860	0.15597	0.05997	0.10870	0.17430	0.15177	0.10366	0.43051
TransE	50	0.19956	**0.17185**	0.07743	0.12161	0.18863	0.15533	0.11633	0.45412
TransH	50	0.19224	0.16441	0.07106	0.12290	0.18466	0.15444	0.10970	0.43390
TransR	50	0.17563	0.14376	0.06684	0.11653	0.16604	0.14178	0.10091	0.41066
TransE	**100**	**0.20035**	0.16702	**0.07802**	**0.12371**	**0.18953**	**0.15546**	**0.11674**	**0.45521**
TransH	100	0.19331	0.16212	0.07527	0.12053	0.18548	0.15071	0.10930	0.43532
TransR	100	0.16889	0.14451	0.06734	0.10744	0.16422	0.14119	0.09953	0.41633
TransE	200	0.17014	0.13157	0.07201	0.11359	0.16883	0.14661	0.10845	0.44561
TransH	200	0.17365	0.14570	0.07214	0.11035	0.16904	0.13455	0.10664	0.42189
TransR	200	0.15443	0.11785	0.06552	0.09851	0.15002	0.13871	0.09931	0.42500

E.2. LastFM.

Modelo	d	P@5	P@10	R@5	R@10	SER@5	SER@10	MAP	NDCG
TransE	10	0.08112	0.07710	0.10711	0.12771	0.07553	0.07241	0.08551	0.33874
TransH	10	0.00318	0.00751	0.07192	0.08464	0.00292	0.00443	0.01712	0.30538
TransR	10	0.00151	0.00567	0.00661	0.00815	0.00100	0.00313	0.01532	0.29057
TransE	20	0.10011	0.11261	0.18168	0.20512	0.09847	0.08923	0.10023	0.37993
TransH	20	0.00914	0.01836	0.09561	0.10048	0.00894	0.01928	0.03500	0.36954
TransR	20	0.00451	0.01530	0.03617	0.58210	0.00320	0.00624	0.03054	0.32906
TransE	30	0.12510	0.11051	0.27116	0.29844	0.10347	0.09361	0.10342	0.42561
TransH	30	0.10279	0.10511	0.14667	0.18023	0.09963	0.09832	0.04801	0.41578
TransR	30	0.03191	0.07051	0.10614	0.11693	0.02987	0.03612	0.03612	0.34007
TransE	50	**0.15099**	**0.14821**	**0.41588**	**0.53599**	**0.14223**	**0.13699**	**0.10867**	**0.46903**
TransH	50	0.12114	0.11694	0.23115	0.30517	0.11954	0.10542	0.06912	0.44532
TransR	50	0.09141	0.10046	0.28811	0.36190	0.07573	0.08094	0.05730	0.32853
TransE	100	0.14520	0.11339	0.40314	0.52739	0.12430	0.10051	0.10494	0.46231
TransH	100	0.12531	0.10641	0.34803	0.47293	0.11092	0.09512	0.09381	0.43789
TransR	100	0.08214	0.08053	0.25668	0.39283	0.07631	0.06592	0.08004	0.32860
TransE	200	0.12477	0.10891	0.30167	0.43952	0.11363	0.11280	0.10466	0.36287
TransH	200	0.10667	0.10057	0.22679	0.31583	0.09731	0.09682	0.09548	0.35906
TransR	200	0.08681	0.09581	0.27631	0.38550	0.07932	0.08223	0.08029	0.30956