Université de Technologie de Compiègne

ISCF

Thème

EXPLOITATION DES PLONGEMENTS DE GRAPHES DE CONNAISSANCE POUR LA RECOMMANDATION TOP-N

Presented by:

Ibrahim Attala
Ermal Belul
Hedil Maatoug
Ambra Zajsi
El Hadj Mohamedou

Supervised by:
Ngoc Luyen LE
Marie-Hélène ABEL

Academic year : 2022-2023

Contents

1 Introduction

All the concepts described below are based on [Grover and Leskovec, 2016] paper about Node2vec.

With the rapid development of the internet, the volume of data has grown exponentially. Because of the overload of information, it is difficult for users to pick out what interests them among a large number of choices. To improve the user experience, recommender systems have been applied for scenarios such as music recommendation, movie recommendation, and online shopping; they are complex models aimed at suggesting relevant items that are on the interest of the users.

In recent years, introducing a knowledge graph (KG) into the recommender system as side information has attracted the attention of researchers. A KG is a heterogeneous graph, where nodes function as entities, and edges represent relations between entities. Items and their attributes can be mapped into the KG to understand the mutual relations between items. Moreover, users and user side information can also be integrated into the KG, which makes relations between users and items, as well as the user preference, can be captured more accurately.

Recent advances in research have demonstrated the effectiveness of knowledge graphs in providing valuable external knowledge to improve recommendation systems. A knowledge graph can encode high-order relations which connect two objects with one or multiple related attributes. Leveraging this wealth of heterogeneous information for top-N item recommendation is a challenging task, as it requires the ability of effectively encoding a diversity of semantic relations and connectivity patterns.

At the same time, it is also beneficial to use a recommendation model whose features have a straight forward interpretation and that can thus be easily adapted to a specific recommendation problem.

In the last decade, research on recommender systems has shown that hybrid systems generally outperform collaborative or content-based systems, addressing problems such as data sparsity, new items or overspecialization. Knowledge graphs are an ideal data structure for hybrid recommender systems, as they allow to easily represent user-item interactions, and item and user properties as typed edges connecting pairs of entities. Recommender systems based on knowledge graphs have shown to generate high quality recommendations that are also easier to interpret and explain. The crucial point to use knowledge graphs to perform item recommendations is to be able to effectively define a measure of user-item relatedness on the graph.

The contributions of the proposed work can be summarized as follows: (a) we use a knowledge graph encompassing both collaborative information from user feedback and item information from Linked Open Data; (b) we learn property-specific vector representations of knowledge graph entities in a completely unsupervised way via node2vec; (c) we compute property-specific relatedness scores between users and items using the obtained vector representations; (d) we combine the property-specific relatedness scores in a global relatedness score using a supervised learning to rank approach optimizing top-N item recommendation; (e) we evaluate the effectiveness of the proposed approach with respect to four baselines; (f) we remark that, thanks to the explicit semantics of the properties of the knowledge graph, the model is easily adaptable to specific recommendation problems, as the features have a clear interpretation.

2 Context and Problem

Our research focus on:

- What data do we have? which format of data? How can we treat the data?
- What kind of information can be added in order to enrich the dataset?
- How can we achieve a graph data with complement information from a knowledge graph?
- How to build a deep learning model?

3 State of the art

3.1 Recommender Systems

Recommender systems have been applied in many domains, including movies, music, POIs, news, education, etc. The recommendation task is to recommend one or a series of unobserved items to a given user, and it can be formulated into the following steps:

- First, the system learns a representation u_i and v_j for the given user u_i and an item v_j.
- Then, it learns a scoring function f : $u_i \times v_j \rightarrow y^{i,j}$, which models the preference of u_i for v_j .
- Finally, the recommendation can be generated by sorting the preference scores for items. To learn the user/item representation and the scoring function.

The recommendation algorithm is the core element of recommender systems, which are mainly categorized into collaborative filtering (CF)-based recommender systems, content-based recommender systems, and hybrid recommender systems:

- **Collaborative Filtering:** CF assumes that users may be interested in items selected by people who share similar interaction records with them. The interaction can either be explicit interaction, like ratings, or implicit interaction, such as click and view.
- **Content-based Filtering:** Compared with the CF-based model, which learns the representation of user and item from global user-item interaction data, content-based methods depict the user and item from the content of items. The assumption of content-based filtering is that users may be interested in items that are similar to their past interacted items. The item representation is obtained by extracting attributes from the item's auxiliary information, including texts, images, etc., while the user representation is based on the features of personal interacted items.

The procedure of comparing candidate items with the user profile is essentially matching them with the user's previous records.
Therefore, this approach tends to recommend items that are similar to items liked by a user in the past.

- **Hybrid Method:** Hybrid method is to leverage multiple recommendation techniques in order to overcome the limitation of using only one type of method. One major issue of CF-based recommendation is the sparsity of user-item interaction data, which makes it difficult to find similar items or users from the perspective of interaction. A special case for this issue is the cold-start problem, which means the recommendation for new user or item is difficult since the user-user and item-item similarity cannot be determined without any interaction records.

By incorporating content information of users and items, also known as user side information and item side information, into the CF-based framework, better recommendation performance can be achieved.

Recommendation systems provide suggestions for items that are most likely of interest to a particular user, cope with information overload, help users making better choices and try to predict items what are the most suitable, based on the user's interest and constraints [Chicaiza and Díaz, 2021].

3.2 Knowledge graphs

The KG is a practical approach to represent large-scale information from multiple domains. A common way to describe a KG is that the nodes represent entities, while edges in the graph function as relations between entities.

Each edge is represented in the form of a triple (head entity, relation, tail entity), also known as a fact in the graph, implying the specific relationship between the head entity and tail entity.

A KG is a heterogeneous network since it contains multiple types of nodes and relations in the graph. Such a graph has strong representation ability since multiple attributes of an entity can be obtained by following different edges in the graph, and high-level relations of entities can be discovered through these relational links [Guo et al., 2022].

Definition 3.1. A knowledge graph is a set K = (E, R,O) where E is the set of entities, $R \subset E \times \Gamma \times E$ is a set of typed relations between entities and O is an ontology.

The ontology O defines the set of relation types ('properties'), the set of entity types Λ, assigns nodes to their type $O : e \in E \rightarrow \Lambda$ and entity types to their related properties $O : \epsilon \in \Lambda \rightarrow \Gamma_\epsilon \subset \Gamma$.

Definition 3.2. Users are a subset of the entities of the knowledge graph,$u \in U \subset E$.

Definition 3.3. Items are a subset of the entities of the knowledge graph, $i \in I \subset E$. Users and items form disjoint sets, $U \cap I = \emptyset$.

Definition 3.4. A triple is an edge of the knowledge graph, i.e. $(i, p, j) \in R$ where $i \in E$ and $j \in E$ are entities and $p \in \Gamma$ is a property.

- A depiction of the knowledge graph model for the specific case of movie recommendation is provided, The problem of top-N item recommendation is that of selecting a set of N items from a set of possible candidate items. Typically, the number of candidates is order of magnitudes higher than N and the recommender system has to be able to identify a short list of very relevant items for the user. More formally:

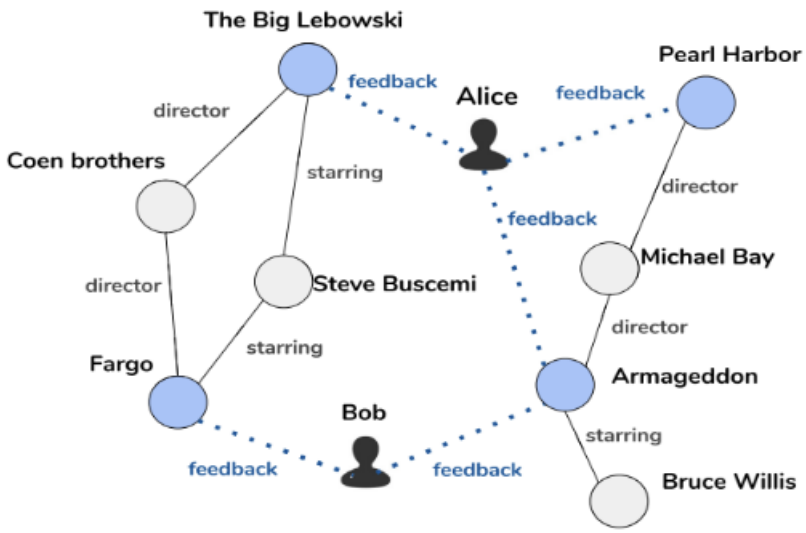

Figure 1: A depiction of the knowledge graph model for the specific case of movie recommendation.

Knowledge graph represents user-item interactions through the special property 'feedback', as well as item properties and relations to other entities.

Items are represented in blue, whereas other entities are represented in grey. The knowledge graph allows to model both collaborative and content-based interactions between users and items.

In this depiction, 'starring' and 'director' properties are represented as an example, more properties are included in the experiments.

3.3 Recommended Systems over knowledge graphs

In 2012, Google proposed the term knowledge graph to refer to the use of semantic knowledge in web searches. Although the term associated with knowledge graph is new, representing pieces of knowledge as interconnected nodes is not a new idea. We can consider current Knowledge graphs as an evolution of semantic networks, an old concept that emerged from the literature on cognitive science and artificial intelligence.

A knowledge graph provides machine-readable data organized as a graph; graph data describes and interconnects entities of an open or a close domain. The data in a knowledge graph is accessible through the web and can be consumed automatically; these characteristics have facilitated the creation of applications such as question-answering, recommender systems, information retrieval, domain-specific applications by knowledge area, and other applications (social networks and geoscience).

The concept of knowledge graphs, or the use of semantic knowledge to improve search and other applications, has evolved from earlier ideas in cognitive science and artificial intelligence such as semantic networks.

Knowledge graphs consist of machine-readable data organized as a graph and can be used in various applications including question answering, recommendation systems, and information retrieval.

In the context of recommendation systems, knowledge graphs can be used to make recommendations by exploiting the connections between entities representing users, items, and their interactions.

Knowledge graph-based recommendation approaches can be classified into three categories: ontology-based, linked open data-based, knowledge graph embeddings-based. There are also path-based and hybrid approaches that combine different methods.

3.4 Node2Vec

3.4.1 Introduction

Node2Vec is an algorithm that allows the user to map nodes in a graph G to an embedding space. Generally, the embedding space is of lower dimensions than the number of nodes in the original graph G. The algorithm tries to preserve the initial structure within the original graph. Essentially, the nodes which are similar within the graph will yield similar embeddings in the embedding space. These embedding spaces are essentially a vector corresponding to each node in the network. The graph embeddings are commonly used as input features to solve machine learning problems oriented around link prediction, community detection, classification, etc. Generally, when dealing with very large graphs it's quite difficult for scientists to visually represent the data they're working with. A common solution to see how a graph looks is to generate node embeddings associated with that graph and then visualize the embeddings in a lower-dimensional space. This allows you to visually see potential clusters or groups forming in very large networks.

3.4.2 Random walks generation

Having an understanding of what random walks are and how they work is crucial in understanding how node2vec works. We will provide a high-level overview of it, but if you want a more intuitive understanding and implementation of random walks in python you can read the article We have previously written regarding this topic. Having an understanding of what random walks are and how they work is crucial in understanding how node2vec works. We will provide a high-level overview of it.

$$P(c_i = x | c_{i-1} = v) = \begin{cases} \frac{\pi_{vx}}{Z} & \text{if } (v, x) \in E \\ 0 & \text{otherwise} \end{cases}$$

Where z is the normalization constant, and π_{vx} is the unnormalized transition probability between nodes x and v. Clearly, if there is no edge connecting x and v, then the probability will be 0, but if there is an edge, we identify a normalized probability of

going from v to x. The easiest way to introduce a bias to influence the random walks would be if there was a weight associated with each edge. However, that wouldn't work in the case of unweighted networks. To resolve this, we introduce a guided random walk governed by two parameters p and q. p indicates the probability of a random walk getting back to the previous node, and q indicates the probability that a random walk can pass through a previously unseen part of the graph.

$$\alpha_{pq}(t,x) = \begin{cases} \frac{1}{p} & \text{if } d_{tx} = 0 \\ 1 & \text{if } d_{tx} = 1 \\ \frac{1}{q} & \text{if } d_{tx} = 2 \end{cases}$$

Where d_{tx} represents the shortest path between nodes t and x. It can be visually seen in the illustration below.

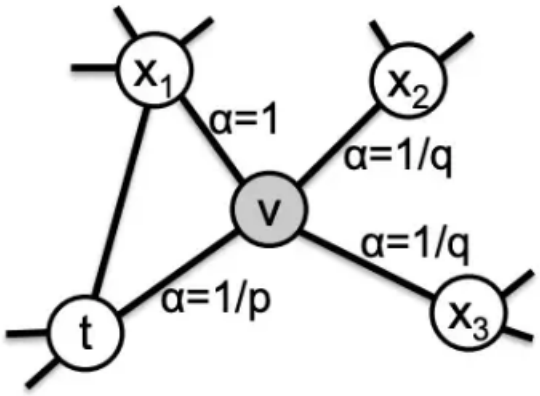

Figure 2: Shortest path representation.

3.4.3 Skip-gram architecture

Having a general knowledge of the word2vec is necessary to understand what's being done in node2vec.For the sake of this rapport, we will provide a high-level overview of the Skip-Gram model. The skip-gram model is a simple neural network with one hidden layer trained in order to predict the probability of a given word being present when an input word is present .

Given some corpus of text, a target word is selected over some rolling window. The training data consists of pairwise combinations of that target word and all other words in the window. This is the resulting training data for the neural network. Once the model is trained, we can essentially yield a probability of a word being a context word for a given target. The following image below represents the architecture of the neural network for the skip-gram model.

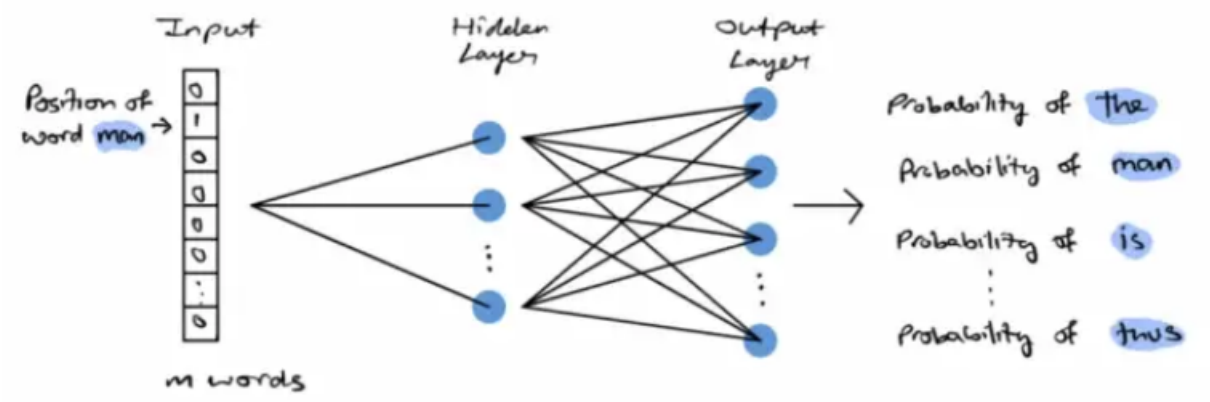

Figure 3: Skip-Gram Model architecture.

A corpus can be represented as a vector of size N, where each element in N corresponds to a word in the corpus. During the training process, we have a pair of target and context words, the input array will have 0 in all elements except for the target word. The target word will be equal to 1. The hidden layer will learn the embedding representation of each word, yielding a d-dimensional embedding space. The output layer is a dense layer with a softmax activation function. The output layer will essentially yield a vector of the same size as the input, each element in the vector will consist of a probability. This probability indicates the similarity between the target word and the associated word in the corpus.

3.4.4 How it works

The process for node2vec is fairly simple, it begins by inputting a graph and extracting a set of random walks from the input graph. The walks can then be represented as a

directed sequence of words where each node represents a word. The generated random walks are then passed into the skip-gram model. As explained above, the skip-gram model works on words and sentences, each node in the random walk can be represented as a word and the entire walk can be represented as a sentence. The result of the skip-gram model yields an embedding for each node (or word in this analogy). The entire process can be seen below.

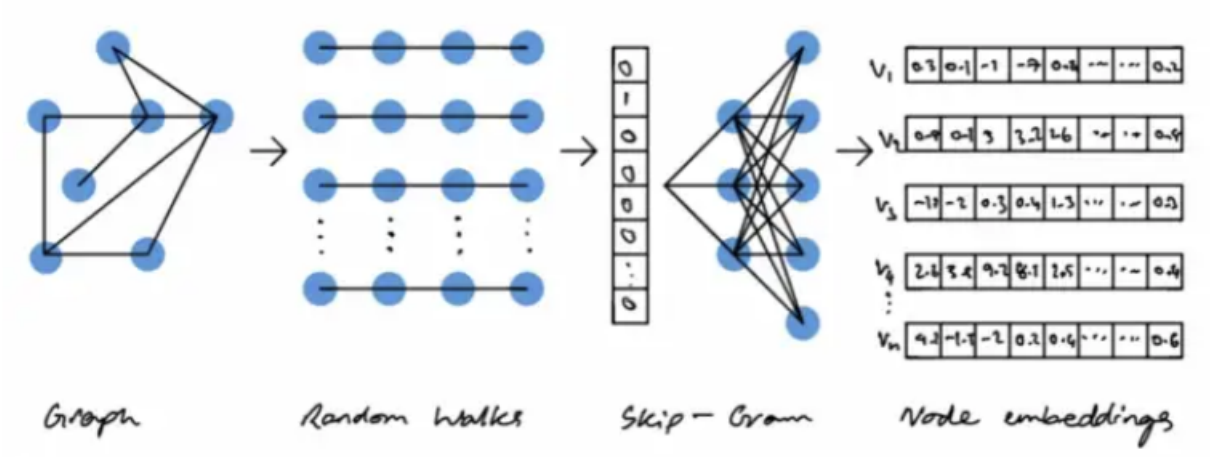

Figure 4: Node2Vec Architecture.

3.4.5 Implementation

For this implementation, we will generate a random graph, apply node2vec to the graph and then visualize the embeddings in a lower dimensional space using PCA. We first Generate Network then Apply Node2Vec, Convert to DataFrame and finally Visualize Embeddings.

3.4.6 Why do we use Node2vec?

- It's scalable and parallelizes easily.
- Open sourced in python.
- Unique approach to learning feature representation via node embeddings.

- The structure of the original network is preserved through the embeddings.
- Node2Vec has many real world applications including and not limited to node classification, community detection, link prediction, etc.

3.4.7 Concluding Remarks

Overall,we think the main takeaways from this article should be that node2vec generates embeddings associated with each node in a given network. These embeddings preserve the original structure of the network, thus similar nodes will have "similar" embeddings. This is an open source algorithm and scales very well to deal with larger networks (as long as you have the computing powers for it).

3.5 Entity2Rec

3.5.1 Property-specific knowledge graph embeddings

The relatedness between two nodes can be easily calculated using vector similarity measures. However, Node2vec cannot represent the semantic properties of a knowledge graph. Films may belong to actors who are not the subject and may share the same director but not the same writer. A complete treatment of all knowledge graphs neglecting the definition of attributes does not allow the computation of these variables. Thus, we start by learning property-specific vector representation of nodes considering one property at the time, i.e. creating property-specific subgraphs K_p. Then,for each K_p independently, we learn a mapping $x_p : e \in K_p \rightarrow R_d$, optimizing the node2vec objective function.

$$max_{x_p} \sum_{e \in K_p} \left(-\log Z_e + \sum_{n_i \in N(e)} x_p(n_i) \cdot x_p(e) \right)$$

Where $Z_e = \sum_{v \in K_p} \exp\left(x_p(e) \cdot x_p(v)\right)$ is the per-node partition function and it is approximated using negative sampling [Mikolov et al., 2013], and $N(e) \in K_p$ is the neighborhood of the entity e defined by the node2vec random walk. The optimization is carried out using stochastic gradient ascent over the parameters defining x_p and it attempts to maximize the dot product between vectors of the same neighborhood, to embed them close together in vector space.

3.5.2 Property-specific subgraphs

We consider two different strategies to create property-specific subgraphs. The first one is the one presented in [Palumbo et al., 2020], which keeps separated collaborative and content-based information. We shall note with K_p the property-specific subgraphs created with this strategy and we will refer to it as entity2rec. The latter is the one presented in this work, which considers hybrid property-specific subgraphs, in the sense that each of them contains both collaborative and content-based information. We refer to the hybrid subgraphs as K_p^+ and to the whole approach as entity2rec, as it has proven to be the most effective one among the two.

3.5.3 Collaborative-content subgraphs

For each property $p \in \Gamma_\epsilon$ we define a subgraph K_p as the set of entities connected by the property p, i.e. the triples (i, p, j) For example, if $p =$ 'starring', we have edges connecting movies to their starring actors, e.g.e.g. (Fargo, starring, Steve Buscemi), if $p =$ 'subject' we have edges connecting movies to their category, e.g. (Fargo, subject, American crime drama films). The only subgraph K_p containing users is that corresponding to $p =$ 'feedback', where triples represent user-item interactions, e.g. (user201, feedback, dbr:Fargo). From the vector representations x_p, propertyspecific relatedness scores can be defined as follows:

$$\rho_p(u,i) = \begin{cases} S(x_p(u), x_p(i)) & \text{if } p = \text{'feedback'} \\ \frac{1}{|R+(u)|} \sum_{i' \in R_+(u)} S(x_p(i), x_p(i')) & \text{otherwise} \end{cases}$$

Where $R_+(u)$ denotes a set of items liked by the user u in the past and s denotes a measure of vector similarity. In this work, we consider s as the cosine similarity. The features include both collaborative and content information and have a straight-forward interpretation. When considering $p =$ 'feedback', K is reduced to the graph of user-item interactions and thus $\rho_{feedback}(u,i)$ models collaborative filtering. $P_{feedback}(u,i)$ will be high when $x_{feedback}(u)$ is close to the item $x_{feedback}(i)$ in vector space, i.e. when i has been liked by users who have liked the same items of u in the past and are thus tightly connected in the $K_{feedback}$ graph. On the other hand, when p corresponds to other properties of the ontology O, the features en-code content information. For instance, if p is 'starring', $\rho_{starring}(u,i)$ will be high if $x_{starring}(i)$ is close to items $x_{starring}(i)$ i.e. when i shares starring actors with items that the user u has liked in the past. For 'new items', i.e. with no feedback from users, we are still able to compute all the content-based features.

Figure 5: (a): Property-specific subgraphs as defined in [Palumbo et al., 2020]. Collaborative and content information are separated. User-item relatedness scores can be computed directly only for Kfeedback, whereas for content properties it is necessary to average the distance with respect to the items that a user has rated in the past. For properties such as "director", the connectivity is poor. (b): Property-specific subgraphs as defined in this work. Feedback from user to items is included in every graph, improving connectivity and allowing to measure directly user-item relatedness for all properties.

3.5.4 Hybrid subgraphs

The largest issue with the use of property-specific subgraphs that consider collaborative and content-based information as separated is that often these graphs have poor connectivity, as a consequence of the fact that many properties connect one item to a few or even a single entity. This is clearly undesirable for feature learning algorithms based on random walks such as [Palumbo et al., 2020], [Grover and Leskovec, 2016], and [Perozzi et al., 2014]. Consider the example of the property p = 'director'. Since most films have only one director, $K_{director}$ is similar to a set of disconnected star graphs,

where each director is connected to his/her movies. In order to overcome this limitation and maintain the interpretability and explainability of the approach of using one property at the time to create features, we propose to use the 'feedback' property as a pivot property to create bridges between different parts of the graph, i.e. to replace K_p with $K_p^+ = K_p \cup (u, feedback, i)$ where $u \in U$ and $i \in I$. Therefore, we move from an approach where collaborative ($K_{feedback}$) and content-based (k_p with p = feedback) information is well distinguished to a set of property-specific graphs that are hybrid, as they contain both the 'feedback' information and one specific item property (Fig.6).In this case, $\Gamma_\epsilon = \Gamma_\epsilon$ \feedback. From the subgraphs, K_p^+ vector representations x_p can be learned and property-specific relatedness scores between a user $u \in U$ and an item $i \in I$ can be defined as follows:

$$\rho_p(u, i) = S(x_p(u), x_p(i))$$

where s is the cosine similarity. Note that since users are now part of every K_p^+ subgraph, it is no longer necessary to distinguish between collaborative features, where the relatedness score can be computed directly, and content-based features that require to look at the items that the user u has rated in the past, as done in [Palumbo et al., 2020]. For 'new items', i.e. with no feedback from users, we are able to compute all features.

3.5.5 Global user-item relatedness

For each user-item pair, we are now able to compute all property-specific relatedness scores, $\rho^{\rightarrow}(u, i) = \{\rho_p(u, i)\}_{p \in \Gamma}$ either using content/collaborative subgraphs or using hybrid subgraphs as described. We aim to consider these scores as features of a global user-item relatedness model that can be used to provide item recommendation. To this end, we experiment both an unsupervised and a supervised approach.

3.5.6 Unsupervised approach

In the unsupervised approach, user-item property-specific relatedness scores are combined into a single user-item relatedness score used as ranking function $\rho(u, i)$ through different possible functions such as:

- $entity2rec_{avg}(u,i) = avg(\{\rho_p(u,i)\}_{p\in\Gamma})$
- $entity2rec_{min}(u,i) = min(\{\rho_p(u,i)\}_{p\in\Gamma})$
- $entity2rec_{max}(u,i) = max(\{\rho_p(u,i)\}_{p\in\Gamma})$

3.5.7 Supervised approach

In the supervised setting, we define the global user-item relatedness $\rho(u, f(\rho^{\rightarrow}(u,i);\theta)i;\theta) =$ as a function f of the property-specific $\rho^{\rightarrow}(u,i)$ scores and of a set of parameters θ. The goal is that of finding the parameters that optimize top-N item recommendation as a supervised learning to rank problem.

Training data. Given the set of users $U = \{u_1, u_2, ..., u_N\}$ each user u_k is associated with a set of items from feedback data $i_k^{\rightarrow} = \{i_{k1}, i_{k2}, ..., i_{kn}(k)\}$ where n(k) denotes the number of feedback released by the user u_k, and a set of labels $y_k^{\rightarrow} = \{y_{k1}, y_{k2}, ..., y_{kn}(k)\}$ denoting the ground truth relevance of items $i_k^{\rightarrow}$ (e.g. ratings with explicit feedback or boolean values with implicit feedback). $(u_k, i_k^{\rightarrow}, y_k^{\rightarrow})_{k=1}^{N}$ The training set is thus represented as $\tau = \{\}$. In the case of implicit feedback, negative examples are obtained by random sampling, i.e. by randomly selecting items that the user has not interacted with. Sorting. $\rho(u,i;\theta)$ is a ranking function, meaning that, for each user u_k the corresponding $i_k^{\rightarrow}$ items are sorted according to its score. $\rho(u,i;\theta)$ induces a permutation of integers $\pi(u_k,\theta)$, corresponding to sorting the list of items $i_k^{\rightarrow}$ according its score.

Loss.The $A(\pi(u_k, i_k^{\rightarrow}, \theta), y_k^{\rightarrow})$ agreement between the $i_k^{\rightarrow}$ permutation $\pi(u_k,\theta)$ induced by $\rho(u,i;\theta)$ and the list ground $y_k^{\rightarrow}$ truth relevance of items can be measured by any information retrieval metric that measures ranking accuracy. From this score, a loss function can be easily derived as:

$$C(\theta) = \sum_{k=1}^{N}(1 - A(\pi(u_k, i_k^{\rightarrow}, \theta))$$

Optimization. The learning process has thus the objective of finding the set of parameters θ that minimize the loss function C over the training data:

$$\hat{\theta} = argminC(\theta)$$

In this work, we use LambdaMart (Burges, 2010), a listwise learning to rank algorithm that has state-of-the-art results in learning to rank and has shown to achieve the best scores in previous work such as [Palumbo et al., 2020] and [Di Noia, 2016]. We define:

$$entity2rec_{lambda}(u,i) = LambdaMart(\{\rho_p(u,i)\}_{p\in\Gamma})$$

3.5.8 Computational Complexity

The computational complexity of entity2rec in terms of the number of users U and the number of items I. The computational complexity of entity2rec can be divided into a component required for training and one required for testing:

$T_{entity2rec} = T^{train}_{node2rec} + T^{train}_{global}$

The training of entity2rec can be in turn expressed as the training time of node2vec repeated for p times, where p is the number of distinct properties in the graph, which is the time required to generate the embeddings and the time required to learn the global relatedness score :

$T^{train}_{node2rec} = pT^{train}_{node2vec} + T^{train}_{global}$

node2vec is mainly composed of three steps: one in which transition probabilities are pre-computed for each edge, one in which nodes are sampled through random walks, and one in which the word2vec model is applied to learn the embeddings (Grover & Leskovec, 2016):

$T^{train}_{node2vec} = T_{preprocessing} + T_{walks} + T_{word2vec}$

The time for pre-processing transition probabilities depends on the number of edges $T_preprocessing = O(E)$. Since a fixed number of random walks is performed for each node and a single random walk is done in unitary time, $T_walks = O(N)$. Learning the embeddings with word2vec using the Skip-gram model can be done in $T_word2vec = O(Nlog2N)$ as explained in Mikolov, Chen, Corrado, and Dean (2013a). Summing the three components, we obtain that :

$T^{train}_{node2vec}(N, E) = O(E) + O(N) + (Nlog_2 N)$

Given that the number of edges $E \sim UI$ and the number of nodes $N \sim U + I$, we have:

$T^{train}_{node2vec}(U, I) = O(UI) + O(U + I) + O((U + I)log_2(U + I))$

Now in the unsupervised case :

$T^{train}_{global} = O(1)$

And the total training complexity :

$T^{train}_{entity2rec}(p, U, I) = O(pUI) + O(p(U + I)) + O(pUlog_2(U + I)) + O(pl\ log2(U + I))$

The time for testing is :

$T^{test}_{tentity2vec} = O(pUI)$

As for each pair of user and items we need to compute p scores. The total complexity for entity2rec becomes:

$T_{entity2rec}(p, U, I) = O(pUI) + O(p(U + I)) + O(pUlog_2(U + I)) + O(pl\ log2(U + I))$

meaning that for large values of U and I :

$T_{entity2rec}(p, U, I) \sim O(pUI)$

Entity2rec is linear in the number of users, linear in the number of items and linear in the number of properties in the graph. Note that the dependence on p can be removed using an embarrassingly parallel implementation that distributes the generation of the embeddings on different machines, obtaining:

$T^{parallel}_{entity2rec}(U, I) \sim O(UI)$

3.6 SAPRQL and DBpedia

3.6.1 SPARQL

Before introducing SPARQK, we first introduce a concept related to it, that is *Resource Description Framework* (RDF).

Resource Description Framework (RDF) is a standard format for representing and describing data on the web. It is used to describe the relationships between resources, such as the relationship between a webpage and its author, or the relationship between a book and its publisher like MovieLens.

RDF is based on the idea of a triple, which consists of a subject, a predicate, and an object. The subject is the resource being described, the predicate is the relationship between the subject and the object, and the object is the resource or value that is related to the subject.

For instance, there is a book called Deep Learning with Python written by François Chollet, we can say that "François Chollet is the author of Deep Learning with Python", here the subject here is "Françcois Chollet", the object is the book "Deep Learning with Python" and the predicate is the action "is the author of".

RDF data is stored in an RDF graph and can be represented in various formats,such as RDF/XML, JSON-LD ....

We can use SPARQL to query data from RDF data.

SAPRQL is a query language that is designed to retrieve and manipulate data stored in RDF format, it is similar to SQL language but instead of querying data from relational data we use it to query RDF data, it can also be used to construct RDF data, but in our work we used it just to get a specific data related to the MovieLens and Vehicules datasets.

SPARQL allows us to perform several types of queries :

- Select : Retrieve specific data from an RDF dataset ;

- Describe : Retrieve a description of the ressources ;
- Insert : Add data to a dataset ;
- Delete : Remove data from the dataset ;
- Ask : Check if a given query pattern has a solution in the dataset, this query is similar to a SELECT statement in SQL with a LIMIT 1 clause ;
- CONSTRUCT : Create a new RDF data based on existing data.

We can retrieve data with SPARQL in various formats, such as XML, JSON, CSV.

3.6.2 DBpedia

DBpedia is a project that extracts structured information from Wikipedia and makes it available on the web as RDF data. It contains informations about entities such as people, books or vehicules as well as the relationships between them.

We can use *SPARQL* to extract data from DBpedia. DBPedia also provides an API and a web interface that allows us perform queries on its datasets with SPARQL without the need to download the dataset.

3.6.3 Datasets and DBpedia

We have 2 datasets on *Movies* and *Vehicules* that are ready to use but we also use *DBpedia* to extract data on the same datasets, why ?

MovieLens dataset includes three tables : User, Movie and Rating, it is a data collected from the evaluation of the users, so it contains the interactions/feedback of user about movies that we can not find in DBpedia.

DBpedia is a knowledge graph , it contains knowledge informations about the movies, we can find the categoris, actors, directors, descriptions and other informations about the movies, tha we can not find in the *MoviesLens* dataset.

To perform some recommendation algorithms where we need the content of the item like *Content-based Filtering* we have to use *DBpedia* to get informations about the items and not just the interactions.

4 Implementation

4.1 Datasets

4.1.1 Movielens dataset

This dataset contains a set of movie ratings from the MovieLens website, a movie recommendation service. This dataset was collected and maintained by GroupLens, a research group at the University of Minnesota. There are 5 versions included: ”25m”, ”latest-small”, ”100k”, ”1m”, ”20m”.

4.1.2 Vehicle dataset

This dataset contains information about used cars. The columns in the given dataset are as follows:

- Name
- Year
- Selling price
- KM driven
- Fuel
- Transmission
- Mileage
- Engine
- Max power
- Torque
- Seats

4.2 Graph construction

We create an edge between two movie nodes in the graph if both movies are rated by the same user, and for the vehicles we create an edge between two vehicle nodes in the graph if both vehicles have the same fuel, price or engine, depends on the criteria of recommendation that our recommender system is based on.

4.3 Implementing the biased random walk

A random walk starts from a given node, and randomly picks a neighbour node to move to. If the edges are weighted, the neighbour is selected probabilistically with respect to weights of the edges between the current node and its neighbours. This procedure is repeated for number of steps to generate a sequence of related nodes.

The biased random walk balances between breadth-first sampling (where only local neighbours are visited) and depth-first sampling (where distant neighbours are visited) by introducing the following two parameters:

- **Return parameter** (p): Controls the likelihood of immediately revisiting a node in the walk. Setting it to a high value encourages moderate exploration, while setting it to a low value would keep the walk local.
- **In-out parameter** (q): Allows the search to differentiate between inward and outward nodes. Setting it to a high value biases the random walk towards local nodes, while setting it to a low value biases the walk to visit nodes which are further away.

4.4 Training the skip-gram model

Our skip-gram is a simple binary classification model that looks as follows.

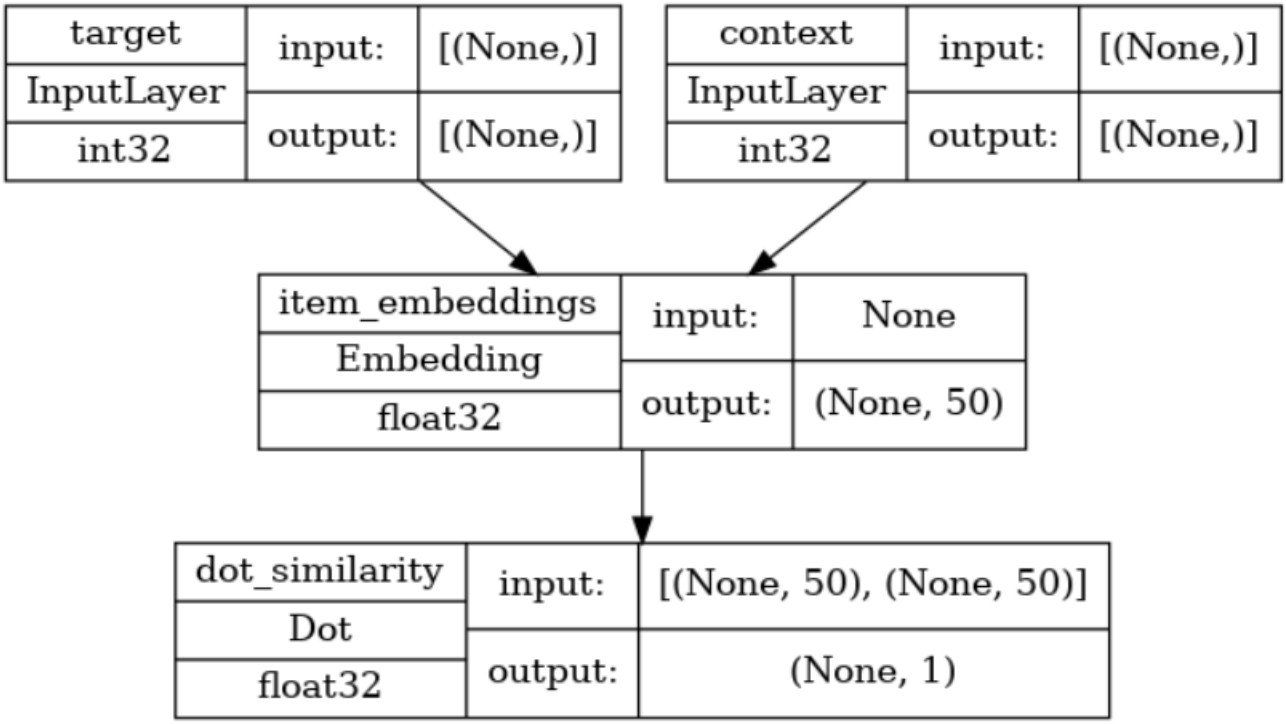

Figure 6: The Skip-gram model.

4.5 Top-n recommendations

After training the model, our recommender system is able to give the top 5 recommendations that correspond to the user request.

```
Matrix, The (1999)
------------------
- Fight Club (1999)
- Matrix, The (1999)
- Star Wars: Episode IV - A New Hope (1977)
- Raiders of the Lost Ark (Indiana Jones and the Raiders of the Lost Ark) (1981)
- Lord of the Rings: The Fellowship of the Ring, The (2001)
```

Figure 7: Movie recommendations.

```
Audi A6 2.0 TDI  Design Edition
-------------------------------
- Audi A6 2.0 TDI  Design Edition
- Jeep Compass 2.0 Longitude Option BSIV
- MG Hector Smart DCT
- Skoda Octavia Elegance 1.8 TSI AT
- Toyota Fortuner 2.8 2WD MT
```

Figure 8: Vehicle recommendations.

5 Conclusion

In this project we talked about recommender systems and knowledge graphs.

We have explained the Entity2Rec which is a recommender system based on property-specific knowledge graph embeddings which use the DBpedia Ontology to define properties of the classes. This method is a new way of creating property-specific subgraphs and to aggregate the relatedness scores into the final ranking function.
We have developed a personalized recommender system based on embedding knowledge graphs, allowing more information to be extracted and added. We also used Node2Vec, which aims to capture the structural relationships between entities in the graph, and create graph embeddings from the knowledge graphs.
We trained our deep learning model on MovieLens and vehicle datasets and evaluated the performances for movie and vehicle recommendations .

We hope this project can help others gain a better understanding of work in this area.

References

[Chicaiza and Díaz, 2021] Chicaiza, J. and Díaz, P. V. (2021). A comprehensive survey of knowledge graph-based recommender systems: Technologies, development, and contributions. *Inf.*, 12:232.

[Di Noia, 2016] Di Noia, T. (2016). Recommender systems meet linked open data. In Bozzon, A., Cudre-Maroux, P., and Pautasso, C., editors, *Web Engineering*, pages 620–623, Cham. Springer International Publishing.

[Grover and Leskovec, 2016] Grover, A. and Leskovec, J. (2016). node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864.

[Guo et al., 2022] Guo, Q., Zhuang, F., Qin, C., Zhu, H., Xie, X., Xiong, H., and He, Q. (2022). A survey on knowledge graph-based recommender systems. *IEEE Transactions on Knowledge amp; Data Engineering*, 34(08):3549–3568.

[Mikolov et al., 2013] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26.

[Palumbo et al., 2020] Palumbo, E., Monti, D., Rizzo, G., Troncy, R., and Baralis, E. (2020). entity2rec: Property-specific knowledge graph embeddings for item recommendation. *Expert Systems with Applications*, 151:113235.

[Perozzi et al., 2014] Perozzi, B., Al-Rfou, R., and Skiena, S. (2014). Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710.