entity2rec: Property-specific knowledge graph embeddings for item recommendation

entity2rec: Property-specific Knowledge Graph Embeddings for Item Recommendation

Enrico Palumbo[a,b,c,*], Diego Monti[c], Giuseppe Rizzo[a], Raphaël Troncy[b], Elena Baralis[c]

[a] *Istituto Superiore Mario Boella, Via Pier Carlo Boggio 61, 10138 Turin, Italy*
[b] *EURECOM, Campus SophiaTech, 450 Route des Chappes, 06410 Biot, France*
[c] *Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129 Turin, Italy*

Abstract

Knowledge graphs have shown to be highly beneficial to recommender systems, providing an ideal data structure to generate hybrid recommendations using both content-based and collaborative filtering. Most knowledge-aware recommender systems are based on manually engineered features, typically relying on path counting and/or on random walks. Recently, knowledge graph embeddings have proven to be extremely effective at learning features for prediction tasks, reducing the complexity and time required to manually design effective features. In this work, we present entity2rec, which learns user-item relatedness for item recommendation through property-specific knowledge graph embeddings. A key element of entity2rec is the construction of property-specific subgraphs. Through an extensive evaluation on three datasets, we show that: (1) hybrid property-specific subgraphs consistently enhance the quality of recommendations with respect to collaborative and content-based subgraphs; (2) entity2rec generates accurate and non-obvious recommendations, compared to a set of state-of-the-art recommender systems, achieving high accuracy, serendipity and novelty. More in detail, entity2rec is particularly effective when the dataset is sparse and has a low popularity bias; (3) entity2rec is easily interpretable and can thus be

*Corresponding author: palumbo@ismb.it, Via Pier Carlo Boggio, 61, 10138 Turin (Italy), Tel. +39 011 227 62 27

Email addresses: palumbo@ismb.it (Enrico Palumbo), diego.monti@polito.it (Diego Monti), giuseppe.rizzo@ismb.it (Giuseppe Rizzo), raphael.troncy@eurecom.fr (Raphaël Troncy), elena.baralis@polito.it (Elena Baralis)

configured for a particular recommendation problem.

Keywords:
recommender systems, knowledge graph embeddings, knowledge graphs

---

1. Introduction

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 (Adomavicius and Tuzhilin, 2005). 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 (Yu et al., 2014; Di Noia et al., 2016; Catherine and Cohen, 2016; Ostuni et al., 2013; Palumbo et al., 2017) that are also easier to interpret and explain (Kanjirathinkal, 2017). 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. At the present time, most knowledge-aware recommender systems are based on manually engineered features based on path counting and/or random walks (Figueroa et al., 2015). However, feature engineering is a time consuming endeavour and machine learning research has shown that feature learning algorithms generate higher quality features (Bengio et al., 2013). Feature learning algorithms applied to a knowledge graph generate 'knowledge graph embeddings', which have proven to be very effective for prediction tasks such as knowledge graph completion (Bordes et al., 2013; Wang et al., 2014; Lin et al., 2015). Thus, recently, some works have started to use knowledge graph embeddings in recommender systems. Notable examples have used translational models (Zhang et al., 2016; Wang et al., 2018) or random walk-based approaches (Rosati et al., 2016; Ristoski et al., 2018). This paper builds upon this related work and our previous work in the field:

- In Palumbo et al. (2018b) we use the random-walk based approach node2vec (Grover and Leskovec, 2016) to create embeddings of the knowledge graph as a whole;
- In Palumbo et al. (2018c,b) we show how translational models (Bordes et al., 2013) can be used for item recommendation embedding the

knowledge graph as a whole and predicting the 'feedback' property;

- In Palumbo et al. (2017) we introduce the notion of property-specific subgraphs that allows to derive user-item property-specific relatedness scores and to combine them in a global user-item relatedness score used as a ranking function for top-N item recommendation. This 'divide and conquer' approach, based on the aggregation of property-specific embeddings, enables great interpretability and explainability of the recommendations, and produces high quality recommendations.

However, property-specific subgraphs, as built in Palumbo et al. (2017), often exhibit poor connectivity in terms of node degree, i.e. number of connections of a node, due to the fact that many properties typically connect one item to a few or even a single entity. We argue that this may hinder the effectiveness of a random walk-based feature learning algorithm and we propose a new way of building property-specific subgraphs that tackles the connectivity problem, by using users' feedback as a *pivot* property to be combined with all the item properties, one at the time. In this paper, we refer to the approach presented in Palumbo et al. (2017) as 'entity2rec (2017)', while to the approach presented in this work simply as 'entity2rec'.

In this work, we present: (1) a critical analysis of the approach presented in entity2rec (2017) (Palumbo et al., 2017) to build property-specific subgraphs through an analysis of their connectivity in terms of average degree; (2) a new way of building property-specific subgraphs, overcoming the distinction between collaborative and content-based subgraphs and using the 'feedback' property in every graph; (3) a set of different ways to combine property-specific relatedness scores into a single user-item relatedness score used to provide recommendations; (4) a theoretical derivation of the computational complexity of entity2rec in terms of the number of users and items of the system; (5) an extensive empirical evaluation on three standard benchmark datasets comparing entity2rec with its previous version and with a set of state-of-the-art recommender systems. The evaluation protocol is based on standard candidate generation techniques and on metrics that go beyond pure accuracy, as suggested in Herlocker et al. (2004); (6) a comparative analysis of the experimental results across the different datasets and an interpretation of the experimental results using statistical properties of the datasets; (7) an assessment of entity2rec's features importance and an illustration of the interpretability and configurability of entity2rec's rec-

ommendation model. More specifically, we address the following research questions:

RQ1 How do hybrid property-specific subgraphs perform with respect to collaborative-content property-specific subgraphs, namely does entity2rec work better than entity2rec (2017)?

RQ2 How do different aggregation functions perform in generating the user-item relatedness score from property-specific relatedness scores?

RQ3 How does entity2rec perform with respect to collaborative filtering systems in terms of precision, recall, serendipity and novelty of the recommendations?

RQ4 How does entity2rec perform with respect to other knowledge graph embeddings based systems for item recommendation? Is it justified to generate property-specific embeddings to leverage the semantics of the graphs?

RQ5 What is the interpretation of entity2rec features and what is their importance?

RQ6 How can entity2rec be configured to specific user requirements and to explain recommendations?

The structure of the paper is the following. In Sec. 2 we introduce some relevant and recent published work related to the topic of the paper, in Sec. 3 we introduce some definitions that are used throughout the paper, in Sec. 4 we describe the entity2rec approach, in Sec. 5 we provide details on the experimental setup, in Sec. 6 we report and discuss the experimental results and in Sec. 7 we summarize the findings of the paper. In Appendix A we provide additional details on the selection of the properties to build the knowledge graph and in Appendix B we report results presented in Sec. 6 in tabular format.
Software and datasets are publicly available at `https://github.com/D2KLab/entity2rec` to support reproducibility.

2. Related Work

2.1. Knowledge graph embeddings

Feature learning algorithms are applied to networked structure to derive feature vectors effectively encoding the graph structure. Recently, DeepWalk

(Perozzi et al., 2014) has shown that language models such as Word2Vec (Mikolov et al., 2013b) can be extended to graph structures by using random walks to sample sequences of nodes, which can then be treated as 'words' of a document. node2vec (Grover and Leskovec, 2016) has built upon this insight, introducing a 2^{nd} order random walk that can be more easily tailored to a diversity of graph connectivity patterns.

In knowledge graphs, not only the graph structure has to be encoded in the features, but also properties and/or entity types. Thus, knowledge graph embeddings are vector representations of entities and/or relations that attempt to preserve the structure and the semantics of the knowledge graph. They are generated using feature learning algorithms that project entities and/or relations into a vector space by solving a learning problem (unsupervised or supervised). Comprehensive surveys of machine learning algorithms used to learn features from knowledge graphs are Nickel et al. (2016) and Wang et al. (2017). All methods attempt to describe the existing triples in the knowledge graph by learning latent features according to some modeling assumption. RESCAL (Nickel et al., 2011) is a tensor factorization method that explains triples via pairwise interactions of vector representations of entities; NTN (Neural Tensor Network) is an expressive non-linear model that learns representations using neural networks (Socher et al., 2013); distance based models, such as the Structured Embeddings (SE) (Bordes et al., 2011), explain triples using a distance in the vector space. Translational models are a special case of distance-based models that represent relations as translations in the vector space and score triples according to a distance function. These models have shown to be computationally efficient and accurate at the same time (Bordes et al., 2013; Wang et al., 2014; Lin et al., 2015).

2.2. Knowledge-aware recommender systems

Several works have shown the effectiveness of external knowledge resources in enhancing the performance of recommender systems. In Yu et al. (2014) and in Catherine and Cohen (2016), the authors start from a graph-based data model encompassing both user feedback and item relations to generate personalized entity recommendations. Several works leverage Linked Open Data knowledge graphs (Bizer et al., 2009), which represent a wealth of freely available multi-domain ontological knowledge, and have successfully been used in the past to build recommender systems (Figueroa et al., 2015). In Di Noia et al. (2016) and in Ostuni et al. (2013), the authors

adopt a hybrid graph-based data model utilizing Linked Open Data to extract metapath-based features that are fed into a learning to rank framework. In the past couple of years, some works have applied knowledge graph embeddings to recommender systems. In Shi et al. (2018), the authors create metapath-aware embeddings using DeepWalk (Perozzi et al., 2014), which are then combined using an aggregation function and used to initialize a matrix factorization for rating prediction. In Zhang et al. (2016), the authors use translational models to create knowledge graph embeddings that are then combined with embeddings of the items' content (textual and visual knowledge) to initialize a matrix factorization. In Ristoski et al. (2018), knowledge graph embeddings are used as side information for a factorization machine or for a content-based Item-KNN recommender system, like in Rosati et al. (2016). Translational models have also been used to learn knowledge graph embeddings to provide recommendations as a link prediction problem on a knowledge graph (Palumbo et al., 2018c,a). node2vec has been used to generate recommendations using a knowledge graph (Palumbo et al., 2018b). In Sun et al. (2018), the authors use recurrent neural networks to learn representations of different meta-paths connecting users and items that are then aggregated through a pooling and a fully connected layer.

2.3. Beyond accuracy

In their survey dealing with the problem of evaluating a recommender system, Herlocker et al. (2004) review and compare several metrics that could be exploited for conducting a comparison among different algorithms. After discussing the accuracy-based metrics, they argue that, in order to draw a reliable conclusion, it is necessary to also consider other properties of the recommended items. In their opinion, a recommender system should be capable of providing suggestions that are not only accurate but also useful. For example, an extremely popular item may be an accurate but not an interesting suggestion. For this reason, they also discuss other metrics that could be considered beyond the traditional concept of accuracy, such as serendipity, novelty, and diversity. The idea of relying not only on accuracy-based metrics is also supported by Ge et al. (2010). In their work, the authors state that the purpose of an evaluation protocol is to assess the quality of the recommended items and not their accuracy. However, metrics like precision and recall alone are not capable of verifying that the recommendations are actually useful. In fact, only the users of the system can judge their quality in the context of an online experiment. Therefore, their suggestion is to

consider other metrics beyond accuracy when it is necessary to perform an offline study.

3. Preliminaries

In this section, we introduce some definitions that will be used in the remaining of this paper.

Definition 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') Γ, 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 2. *Users are a subset of the entities of the knowledge graph, $u \in U \subset E$.*

Definition 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 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.*

Definition 5. *The property 'feedback' describes an observed positive feedback between a user and an item. Feedback only connects users and items, i.e. $(u, feedback, i)$ where $u \in U$ and $i \in I$.*

A depiction of the knowledge graph model for the specific case of movie recommendation is provided in Fig. 1.

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:

Definition 6. *Given a user $u \in U$, the set of candidate items $I_{candidates}(u) \subset I$ is the set of items that are taken into account as being potential object of recommendation.*

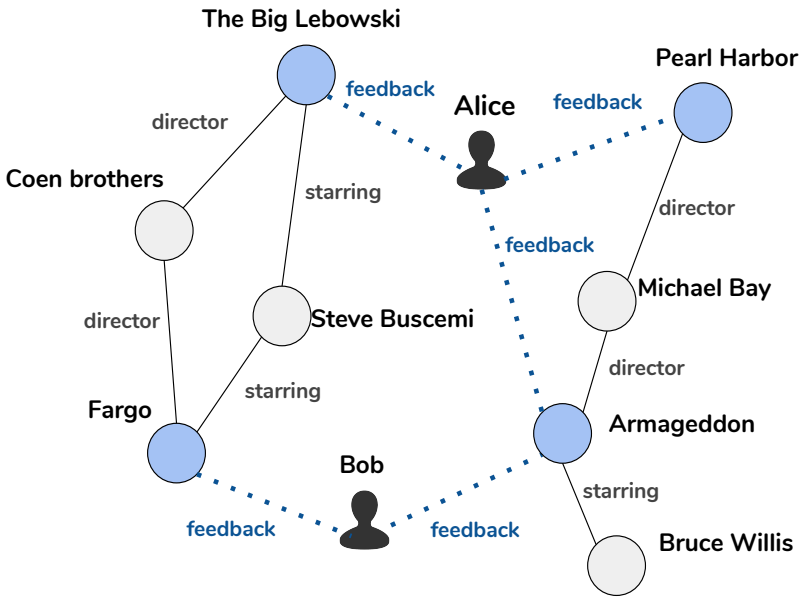


Figure 1: Knowledge graph represents user-item interactions through the special property 'feedback', as well as item properties and relations to other entities. Items are represented in blue, whereas other entities are represented in grey. The knowledge graph allows to model both collaborative and content-based interactions between users and items. In this depiction, 'starring' and 'director' properties are represented as an example, more properties are included in the experiments.

Definition 7. *A ranking function* $\rho(u, i) : (u, i) \in U \times I_{candidates}(u) \rightarrow \mathbb{R}$ *is a function that takes as inputs a user and a candidate item and assigns a score to them. The higher the score, the more relevant the item is deemed to be for user* u*.*

Definition 8. *A ranking function* $\rho(u, i)$ *induces a permutation of integers corresponding to sorting the list of items* $I_{candidates}(u)$ *according its score:*

$$\pi(u, I_{candidates}(u)) = \{i_1, i_2, \ldots, i_L\} \tag{1}$$

where $L = |I_{candidates}(u)|$ *and* $\rho(u, i_j) > \rho(u, i_{j+1})$ *for any* $j = 1, \ldots, L - 1$*.*

Definition 9. *Top-N item recommendation provides to each user* $u \in U$ *the recommended items* $R(u)$*, i.e. the first* $N \leq L$ *elements of* $\pi(u, I_{candidates}(u))$*:*

$$R(u) = \pi(u, I_{candidates}(u))_1^N = \{i_1, i_2, \ldots, i_N\} \tag{2}$$

where $\rho(u, i_j) > \rho(u, i_{j+1})$ *for any* $j = 1, \ldots, N - 1$*.*

4. entity2rec

In this section, we describe the overall entity2rec approach, which is summarized in Fig. 2.

4.1. node2vec

Before describing the entity2rec approach, we review node2vec (Grover and Leskovec, 2016) on which entity2rec is based. node2vec can be seen as a two stage approach: first, it samples nodes of a graph using a set of biased random walks turning it into a 'document'; then, node embeddings are learned using neural language models such as Word2Vec Mikolov et al. (2013b). A crucial concept introduced by node2vec is that of the neighborhood of a node.

4.1.1. Neighborhood of a node

Networked structures can be organized according to different principles. For example, in certain cases, graphs could be organized according to the principle of homophily, meaning that similar entities will tend to be tightly connected in communities. In other cases, the organization can be based

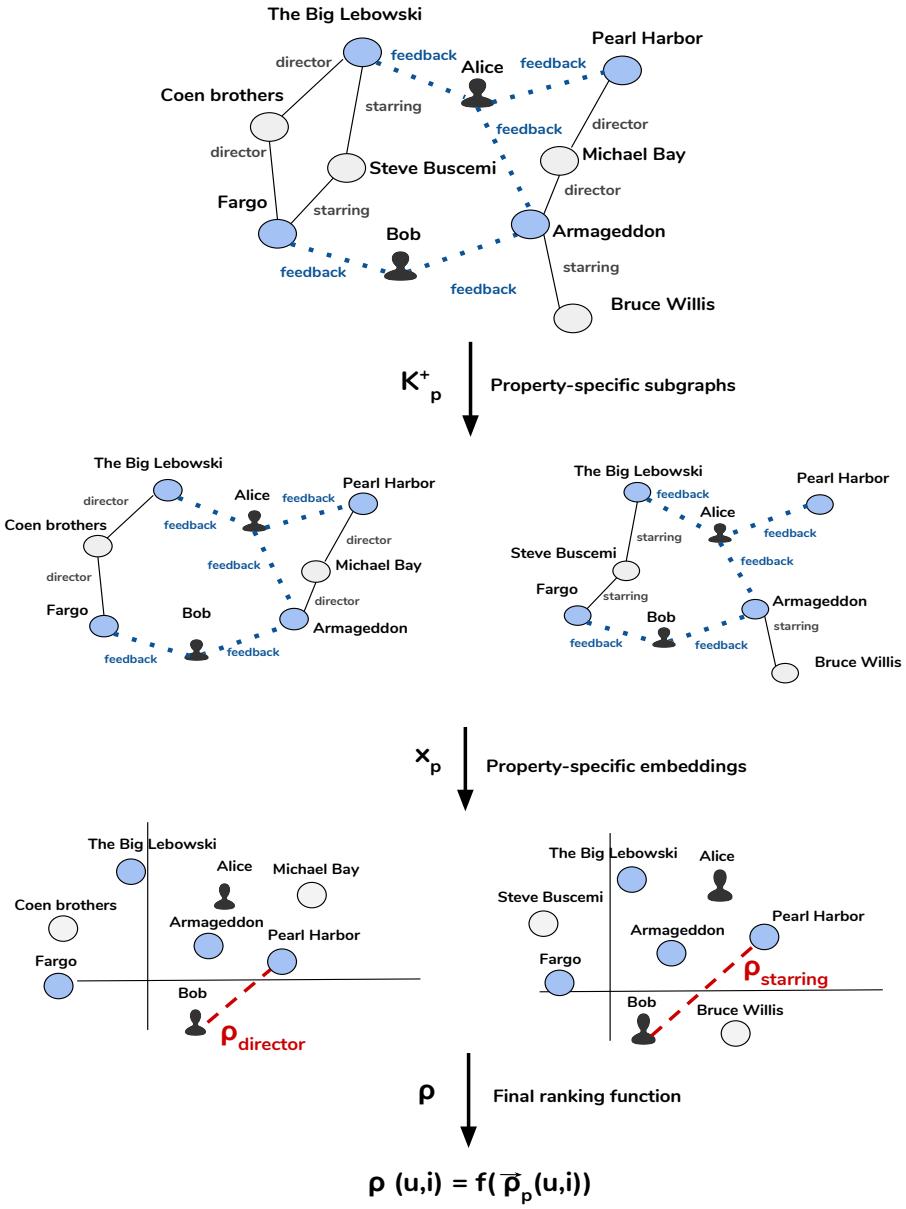


Figure 2: entity2rec starts by creating property-specific subgraphs. Then, on each subgraph, it applies node2vec to compute property-specific embeddings. In the embedding space, property-specific relatedness scores can be computed using vector similarity measures such as cosine similarity. The property-specific relatedness scores are then aggregated to obtain a global relatedness score, which is used as a ranking function for item recommendation. The figure illustrates the case in which hybrid property-specific subgraphs are used, as described in Section 4.3.2, and where only the starring and the director properties are considered.

on the structural roles of nodes in the graph ('structural equivalence') and entities can be considered similar because they have a similar role in the network, e.g. they are both hubs or bridges between different communities. Real-world networks commonly exhibit a mixture of such equivalences (Henderson et al., 2012). Thus, the definition of a node neighborhood must be flexible enough to account for different forms of possible equivalences. The problem of defining neighborhoods can be seen as a question of local search. In general, given a node n, there are two classic and opposite strategy of local search to define a neighborhood $N_s(n)$ of size k: Breadth-First Search (BFS) and Depth-First Search (DFS). One of the shortcomings of prior work is that it fails to offer flexibility in combining these two opposite approaches (Perozzi et al., 2014). node2vec provides a definition of node neighborhood that allows for a mixture of BFS and DFS through a neighborhood sampling strategy based on a biased second order random walk. Parameters p and q control how fast the walk explores and leaves the neighborhood of starting node n. Tuning them allows a flexible sampling strategy and a notion of neighborhood $N_s(n)$, which is able to approximately interpolate between BFS and DFS. p, also called *return parameter*, controls the likelihood of returning to a previously visited node. Low values of p result in a search strategy that lingers around the source node n. q, also called *in-out parameter*, controls the probability of moving further away from the source node n. A high value of q results in a strategy that approximates BFS, while low values encourage outward exploration, approximating DFS.
After samples from the graph are generated using the second order random walk, neural language models are applied to compute node embeddings.

4.1.2. Neural language models

Neural language models have been recently developed to overcome many of the shortcomings of naive one-hot bag of words representations, where word vectors are simply represented as binary feature vectors. While these models have been quite popular for their simplicity and intuitiveness, they suffer from several drawbacks, such as a dimensionality equal to the size of the vocabulary, data sparsity, independence among words. Recent advancements in the field of neural language models have allowed the generation of low-dimensional dense word representations, among which word2vec (Mikolov et al., 2013b) is the most popular one. Word2vec efficiently learns word embeddings from a large amount of raw text, training a two layer neural network to predict the context of a word, defined by a sliding window of am-

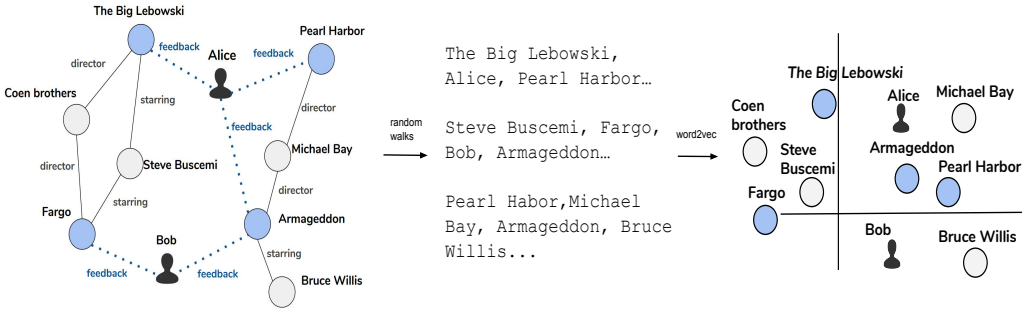


Figure 3: node2vec turns the knowledge graph into a "document" made by a sequences of nodes. Then, neural language models learn embeddings of the graph, preserving structural properties on the graph in the embedding space. This approach has been used in previous work to make recommendations (Palumbo et al., 2018b) and it is included in the results section for comparison.

plitude c. Thus, given a word w_t, the context is defined by the surrounding words $w_{t-c}, w_{t-c+1}...w_{t+c-1}, w_{t+c}$. The most commonly used architecture is the Skip-Gram model. The Skip-Gram model implements a two layer neural network where the input corresponds to the target word w_t and the output to the context words $w_{t-c}, w_{t-c+1}...w_{t+c-1}, w_{t+c}$. Word embeddings are learned so that the average likelihood on the training set of observing a word surrounded by its actual context is maximized.
A full exemplification of the application of node2vec on the KG introduced in Fig. 1 is given in Fig. 3. In this paper, we build upon this work by using property-specific subgraphs and embeddings.

4.2. Property-specific knowledge graph embeddings

From these representations, the relatedness between two nodes can be easily computed using vector similarity measures. However, node2vec cannot account for the diversity of semantic properties of a knowledge graph. Films can be related in terms of starring actors and not in terms of subject, can share the same director but not the same writer. Processing the whole knowledge graph altogether neglecting the semantics of the properties would not allow to account for these variations. 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 $x_p : e \in K_p \rightarrow \mathbb{R}^d$, optimizing the node2vec

objective function (Grover and Leskovec, 2016):

$$\max_{x_p} \sum_{e \in K_p} (-\log Z_e + \sum_{n_i \in N(e)} x_p(n_i) \cdot x_p(e)) \tag{3}$$

where $Z_e = \sum_{v \in K_p} \exp(x_p(e) \cdot x_p(v))$ is the per-node partition function and it is approximated using negative sampling (Mikolov et al., 2013b), 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, i.e. to embed them close together in vector space.

4.3. Property-specific subgraphs

We consider two different strategies to create property-specific subgraphs. The first one is the one presented in Palumbo et al. (2017), 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 (2017) in Section 6. 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 (see Section 6).

4.3.1. 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. (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, property-specific relatedness scores can be defined as follows:

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

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. $\rho_{feedback}(u, i)$ 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 encode 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.

4.3.2. 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. (2017); Grover and Leskovec (2016); 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 $K_p^+ = K_p \bigcup (u, feedback, i)$, where $u \in U$ and $i \in I$. Therefore, we move from an approach where collaborative ($K_{feedback}$) and content-based (K_p with $p \neq feedback$) 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. 4).

In this case, $\Gamma_\epsilon = \Gamma_\epsilon \setminus feedback$. From the subgraphs K_p^+, vector representations x_p can be learned as described in Section 4.2 and property-specific relatedness scores between a user $u \in U$ and an item $i \in I$ can be defined as

follows:

$$\rho_p(u,i) = s(x_p(u), x_p(i)) \tag{4}$$

where s is the cosine similarity. Note that since users are now part of every subgraph K_p^+, it is no longer necessary to distinguish between collaborative features, where the relatedness score can be computed directly as in Eq. 4, 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. (2017). For 'new items', i.e. with no feedback from users, we are able to compute all features.

4.4. Global user-item relatedness

For each user-item pair, we are now able to compute all property-specific relatedness scores $\vec{\rho}(u,i) = \{\rho_p(u,i)\}_{p\in\Gamma}$, either using content/collaborative subgraphs as described in Section 4.3 or using hybrid subgraphs as described in Section 4.3.2. 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.

4.4.1. 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:

$$entity2rec_{avg}(u,i) = avg(\{\rho_p(u,i)\}_{p\in\Gamma}) \tag{5}$$

$$entity2rec_{min}(u,i) = min(\{\rho_p(u,i)\}_{p\in\Gamma}) \tag{6}$$

$$entity2rec_{max}(u,i) = max(\{\rho_p(u,i)\}_{p\in\Gamma}) \tag{7}$$

4.4.2. Supervised approach

In the supervised setting, we define the global user-item relatedness $\rho(u,i;\theta) = f(\vec{\rho}(u,i);\theta)$ as a function f of the property-specific scores $\vec{\rho}(u,i)$ and of a set of parameters θ. The goal is that of finding the parameters θ that optimize top-N item recommendation as a supervised learning to rank problem (Liu et al., 2009).

Training data. Given the set of users $U = \{u_1, u_2, \ldots, u_N\}$, each user u_k is associated with a set of items from feedback data $\vec{i_k} = \{i_{k1}, i_{k2}, \ldots, i_{kn(k)}\}$, where $n(k)$ denotes the number of feedback released by the user u_k, and a set of labels $\vec{y_k} = \{y_{k1}, y_{k2}, \ldots, y_{kn(k)}\}$ denoting the ground truth relevance of items $\vec{i_k}$ (e.g. ratings with explicit feedback or boolean values with implicit

feedback). The training set is thus represented as $\tau = \{(u_k, \vec{i_k}, \vec{y_k})_{k=1}^{N}\}$. 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 items $\vec{i_k}$ are sorted according to its score. $\rho(u, i; \theta)$ induces a permutation of integers $\pi(u_k, \vec{i_k}, \theta)$, corresponding to sorting the list of items $\vec{i_k}$ according its score (see Section 3).

Loss. The agreement $A(\pi(u_k, \vec{i_k}, \theta), \vec{y_k})$ between the permutation $\pi(u_k, \vec{i_k}, \theta)$ induced by $\rho(u, i; \theta)$ and the list of ground truth relevance of items $\vec{y_k}$ can be measured by any information retrieval metric that measures ranking accuracy, such as P@k (Powers, 2011). From this score, a loss function can be easily derived as:

$$C(\theta) = \sum_{k=1}^{N} (1 - A(\pi(u_k, \vec{i_k}, \theta)) \tag{8}$$

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:

$$\hat{\theta} = \arg\min_{\theta} C(\theta) \tag{9}$$

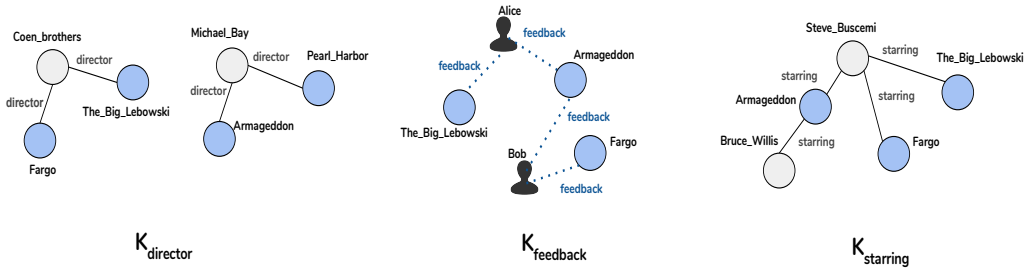
In this work, we use LambdaMart (Burges, 2010), a listwise learning to rank algorithm that has state-of-the-art results in learning to rank and has shown to achieve the best scores in previous work such as Palumbo et al. (2017); Di Noia et al. (2016). We define:

$$entity2rec_{lambda}(u, i) = LambdaMart(\{\rho_p(u, i)\}_{p \in \Gamma}) \tag{10}$$

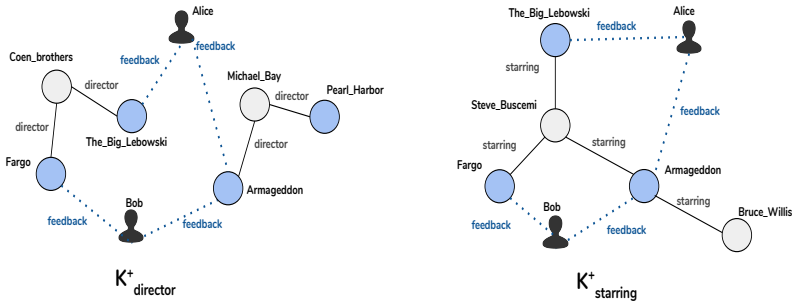
4.5. Computational complexity

In this section, we derive 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:

$$T_{entity2rec} = T^{train}_{entity2rec} + T^{test}_{entity2rec} \tag{11}$$



(a)



(b)

Figure 4: (a): Property-specific subgraphs as defined in Palumbo et al. (2017). Collaborative and content information are separated. User-item relatedness scores can be computed directly only for $K_{feedback}$, 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.

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:

$$T^{train}_{entity2rec} = pT^{train}_{node2vec} + T^{train}_{global} \tag{12}$$

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 and Leskovec, 2016):

$$T^{train}_{node2vec} = T_{preprocessing} + T_{walks} + T_{word2vec} \tag{13}$$

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 $T_{word2vec} = O(Nlog_2N)$ as explained in Mikolov et al. (2013a). Summing the three components, we obtain that:

$$T^{train}_{node2vec}(N, E) = O(E) + O(N) + (N \log_2 N) \tag{14}$$

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

$$T^{train}_{node2vec}(U, I) = O(UI) + O(U + I) + O((U + I) \log_2 (U + I)) \tag{15}$$

Now, in the unsupervised case:

$$T^{train}_{global} = O(1) \tag{16}$$

and the total training complexity:

$$T^{train}_{entity2rec}(p, U, I) = O(pUI)+O(p(U+I))+O(pU \log_2(U+I))+O(pI \log_2(U+I)) \tag{17}$$

The time for testing is:

$$T^{test}_{entity2rec} = O(pUI) \tag{18}$$

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

$$T_{entity2rec}(p, U, I) = O(pUI)+O(p(U+I))+O(pU\log_2(U+I))+O(pI\log_2(U+I)) \tag{19}$$

meaning that for large values of U and I:

$$T_{entity2rec}(p, U, I) \sim O(pUI) \tag{20}$$

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:

$$T^{parallel}_{entity2rec}(U, I) \sim O(UI) \tag{21}$$

5. Experimental setup

In this section, we describe the experimental setup, providing information about the three datasets used in the experiments, how the knowledge graph has been created, and how we have configured and evaluated the systems. The objectives of the experiments reported in this paper are manifold:

- Comparing entity2rec to entity2rec (2017), showing the effectiveness of hybrid property-specific subgraphs;
- Comparing different aggregation functions within entity2rec;
- Comparing entity2rec to other state-of-the-art recommendation algorithms;
- Showing that the use of property-specific subgraphs leads to features that can be easily interpretable and can be configured to a specific recommendation problem.

5.1. Datasets

The first dataset is MovieLens 1M.[1] MovieLens 1M (Harper and Konstan, 2016) is a well known dataset for the evaluation of recommender systems

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

and it contains 1,000,209 anonymous ratings of approximately 3,900 movies made by 6,040 MovieLens users. The second dataset is the LastFM dataset,[2] which contains 92,834 listen counts of 1,892 users of 17,632 musical artists (Cantador et al., 2011). The third dataset is LibraryThing,[3] which contains 7,112 users, 37,231 books and 626,000 book ratings ranging from 1 to 10. For these three datasets, their items have been mapped to the corresponding DBpedia entities (Ostuni et al., 2013) and we make use of these publicly available mappings to create the knowledge graphs using DBpedia data (Auer et al., 2007). We select the most frequently occurring properties p from the DBpedia Ontology[4] (see Appendix A), and for each item property p, we include all the triples (i, p, e) where $i \in I$ and $e \in E$, e.g. (dbr:Pulp_Fiction, dbo:director, dbr:Quentin_Tarantino) in $K_{Movielens1M}$.[5] We finally add the 'feedback' property, modeling user-item interactions. Similarly to what has been done in previous work (Di Noia, 2016; Ristoski et al., 2018), we add a 'feedback' edge for all movie ratings where $r \geq 4$ in X_{train}, all user-item interactions in LastFM as the dataset does not contain explicit feedback, and ratings where $r \geq 8$ for LibraryThing.
We split the data into a training X_{train}, validation X_{val} and test set X_{test} containing, for each user, respectively 70%, 10% and 20% of the ratings. Users with less than 10 ratings are removed from the dataset, as well as items that do not have a corresponding entry in DBpedia. In this process, we lose 674 movies from Movielens1M, 27 users and 7867 musical artists from LastFM, 323 users and 27305 books for LibraryThing. The final datasets statistics after this processing are reported in Table 1. The sparsity of the feedback matrix ρ_f is defined as:

$$\rho_f = \frac{F}{|U||I|} \tag{22}$$

where F is the number of user-item interactions (e.g. ratings or implicit feedback), $|U|$ is the number of users and $|I|$ is the number of items in the dataset. The sparsity measures how many interactions, out of all the possible

[2] `http://files.grouplens.org/datasets/hetrec2011/hetrec2011-lastfm-readme.txt`

[3] `https://www.librarything.com`

[4] `https://wiki.dbpedia.org/services-resources/ontology`

[5] dbo stands for DBpedia Ontology, dct stands for Dublin Core Terms and dbr stands for DBpedia resource.

Dataset	**Domain**	**Feedback**	**Users**	**Items**	$\rho_{\mathbf{f}}$	H
Movielens 1M	Movie	948976	6040	3226	95.13	7.17
LastFM	Music	78633	1865	9765	99.57	7.77
LibraryThing	Book	410199	6789	9926	99.39	8.26

Table 1: Datasets statistics. ρ_r represents the sparsity of the user-item interactions, H is the entropy of the positive feedback distribution.

interactions, have actually taken place between users and items of the system. Given that typically users interact with a very limited fraction of all the items available, the sparsity of datasets for recommender systems is generally very high (Billsus and Pazzani, 1998). We observe that Movielens 1M has a much lower sparsity with respect to LastFM and LibraryThing. The entropy of the dataset is defined in terms of the distribution of positive feedback assigned by users to items:

$$H = -\sum_{i \in I} P^+(i) \log P^+(i) \tag{23}$$

where $P^+(i) : I \to [0, 1]$ is the fraction of positive feedback attributed to the item i. The entropy of the dataset is a useful measure of the popularity bias, i.e. the effect according to which most of the positive feedback is concentrated on a few very popular items (Cremonesi et al., 2010). Being the entropy maximum when the distribution is uniform, a low entropy value indicates a strong concentration of feedback on very popular items, i.e. high popularity bias, whereas a high entropy value points at the contrary effect. In Figure 5, we represent the $P^+(i)$ distributions for the three datasets to visualize this effect. Movielens 1M and LastFM have a strong popularity bias, whereas this effect is relatively weaker for LibraryThing.

Note that the choice of relying on the DBpedia Ontology to build K is not the only possible. For instance, Wikidata (Vrandečić and Krötzsch, 2014) is a more recent project, started in 2012 by the Wikimedia Foundation, collaboratively edited and created by the community. Although it is younger than DBpedia, it is gaining momentum, and currently contains information about 52 million things.[6] The major convenience of using DBpedia is that public mappings have been released to popular RS datasets.

[6] `https://www.wikidata.org/wiki/Wikidata:Statistics`

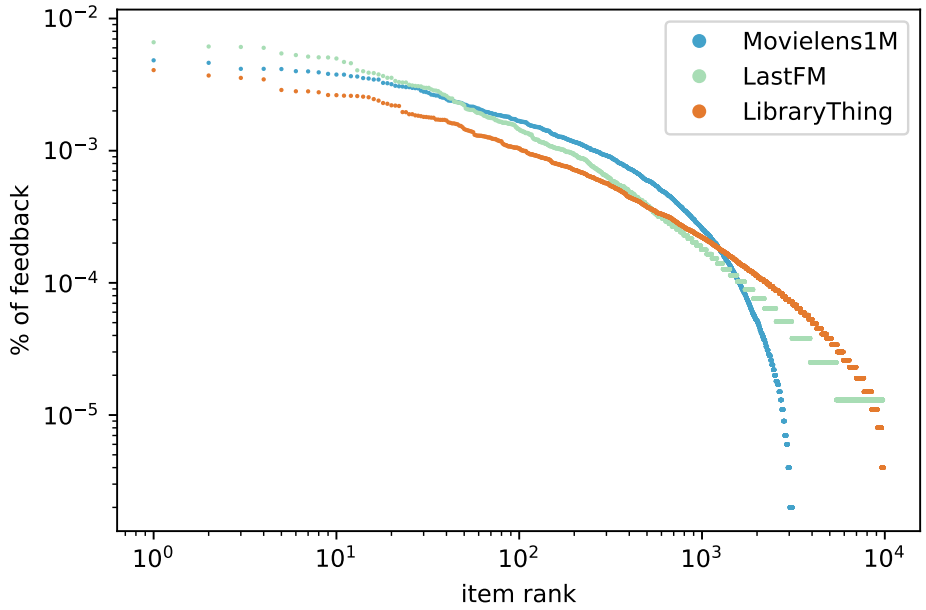


Figure 5: Showing how the positive feedback is distributed among the items of the three datasets, in a log-log scale. LastFM has a strong concentration in the top-100 items, but it has a significant long tail compared to Movielens 1M. In LibraryThing, the popularity bias is weaker and the feedback is more evenly distributed among the items. These considerations are consistent with the values of entropy reported in Table 1.

5.2. Evaluation

We use the evaluation protocol known as AllUnratedItems (Steck, 2013), i.e. for each user, we select as possible candidate items, all the items present in the training or in the test set that he or she has not rated before in the training set:

$$I_{candidates}(u) = I \setminus \{i \in X_{train}(u)\} \tag{24}$$

Items that are not appearing in test set for user u are considered as negative examples, which is a pessimistic assumption, as users may actually like items that they have not seen yet. Scores are thus to be considered as a worst-case estimate of the real recommendation quality. We measure standard information retrieval metrics such as precision (P@k) and recall (R@k).

$$\mathrm{P}(k) = \frac{1}{|U|} \sum_{u \in U} \sum_{j=1}^{k} \frac{\mathrm{hit}(i_j, u)}{k} \tag{25}$$

$$\mathrm{R}(k) = \frac{1}{|U|} \sum_{u \in U} \sum_{j=1}^{k} \frac{\mathrm{hit}(i_j, u)}{|\mathrm{rel(u)}|} \tag{26}$$

where the value of *hit* is 1 if the recommended item i is relevant to user u, otherwise it is 0, and $rel(u)$ is the set of relevant items for user u in the test set. Differently from $P(k)$, $R(k)$ accounts for the fact that different users can have a different number of relevant items, e.g. for a user who is highly selective and likes fewer items, finding relevant items is harder.

In addition to these accuracy-focused metrics, we also decided to measure the serendipity and the novelty of the recommendations. Serendipity can be defined as the capability of identifying items that are both attractive and unexpected (de Gemmis et al., 2015). Ge et al. (2010) proposed to measure it by considering the precision of the recommended items after having discarded the ones that are too obvious. Eq. 27 details how we compute this metric. *hit_non_pop* is similar to *hit*, but top-k most popular items are always counted as non-relevant, even if they are included in the test set of user u. Popular items can be regarded as obvious because they are usually well-known by most users.

$$\mathrm{SER}(k) = \frac{1}{|U|} \sum_{u \in U} \sum_{j=1}^{k} \frac{\mathrm{hit\_non\_pop}(i_j, u)}{k} \tag{27}$$

In contrast, the metric of novelty is designed to analyze if an algorithm is able to suggest items that have a low probability of being already known by a

user, as they belong to the long-tail of the catalog. This metric was originally proposed by Vargas and Castells (2011) in order to support recommenders capable of helping users to discover new items. We formalize how we computed it in Eq. 28. Note that this metric, differently from the previous ones, does not consider the correctness of the recommended items, but only their novelty.

$$\mathrm{NOV}(k) = -\frac{1}{|U| \times k} \cdot \sum_{u \in U} \sum_{j=1}^{k} \log_2 \mathrm{P}_{\mathrm{train}}(i_j) \tag{28}$$

The function $P_{train} : I \rightarrow [0, 1]$ returns the fraction of feedback attributed to the item i in the training set. This value represents the probability of observing a certain item in the training set, that is the number of ratings related to that item divided by the total number of ratings available. In order to avoid considering as novel items that are not available in the training set, we consider $\log_2(0) \doteq 0$ by definition.

We compare entity2rec to a set of state-of-the-art collaborative filtering recommender systems from the MyMediaLite library (Gantner et al., 2011), which has shown to outperform competing libraries in controlled experiments (Said and Bellogín, 2014):

- BPRMF: a matrix factorization method where the optimization is performed using Bayesian Personalized Ranking (Rendle et al., 2009).
- BPRSLIM: a Sparse LInear Method where the optimization is performed using Bayesian Personalized Ranking (Ning and Karypis, 2011).
- ItemKNN: a K-nearest neighbor recommender based on items (Sarwar et al., 2001).
- LeastSquareSLIM: a Sparse LInear Method optimized for the ranking elastic net loss (Ning and Karypis, 2011).
- MostPop: a simple baseline algorithm where the top-N popular items are recommended to every user.
- WRMF: the Weighted Regularized Matrix Factorization is a matrix factorization method where a weighting matrix is used to account for different confidence levels in the user-item feedback (Hu et al., 2008).

Furthermore, we consider other recommender systems based on translational models for knowledge graph embeddings, where the property to predict is 'feedback', as explained in Palumbo et al. (2018c):

- TransE (Bordes et al., 2013): it learns representations of entities and relations so that $h + l \approx t$ where $(h, l, t) \in R$ is a triple. h is the 'head' entity, l is the relation and t is the 'tail' entity. The score function for a triple is thus $f(h, l, t) = D(h + l, t)$, where D is a distance function such as the L_1 or the L_2 norm.

- TransH (Wang et al., 2014): the first extension of TransE, enables entities to have different representations when involved in different relations by projecting entities on a hyperplane identified by the normal vector w_l. The score function becomes: $f(h, l, t) = D(h_\perp + l, t_\perp)$, where $h_\perp = h - w_l^T h w_l$ and $t_\perp = t - w_l^T t w_l$, where D is a distance function such as the L_1 or the L_2 norm.

- TransR (Lin et al., 2015): it enables entities and relations to be embedded in a separate vector space through a matrix M_l associated to any relation l that performs projections of vectors from entity to relation space. The score function is: $f(h, l, t) = D(h_l + l, t_l)$ where $h_l = hM_l$ and $t_l = tM_l$, where D is a distance function such as the L_1 or the L_2 norm.

Finally, we also analyze the behaviour of knowledge graph embeddings computed using node2vec:

- node2vec (Grover and Leskovec, 2016): it performs random walks through a flexible exploration strategy of the graph and feeds the sampled sequences into a word2vec model that learns node embeddings. node2vec does not take into account the semantics of the properties, we apply it directly on the whole knowledge graph K as done in Palumbo et al. (2018b).

All the baselines have been trained on the user ratings contained in X_{train} in the original matrix format and tested on X_{test}. The implementations of the translational based embeddings[7] and of node2vec[8] are available online.

[7] `https://github.com/thunlp/KB2E`

[8] `https://github.com/aditya-grover/node2vec`

entity2rec is also publicly available on GitHub.[9]

5.3. Configuration

We have configured entity2rec hyper-parameters by assessing the P@5 of the model on the validation set, using grid and manual searches. We have optimized the dimension of the embeddings d, the maximum length of the random walk l, the context size for the optimization c, the return parameter p, the in-out parameter q, the number of random walks per each node of the graph n (see Grover and Leskovec (2016) for more information on these parameters). More in detail, we started to search for hyper-parameters on the Movielens 1M dataset, making a grid search and evaluating the model on the validation set, using the ranges $p \in \{0.25, 1, 4\}$, $q \in \{0.25, 1, 4\}$, $d \in \{200, 500\}$, $l \in \{10, 20, 30, 50, 100\}$, $c \in \{10, 20, 30\}$, $n \in \{10, 50\}$. We found the optimal configuration in this range to be $C1 = \{p : 4, q : 1, d : 200, l : 100, c : 30, n : 50\}$, and we observed that the performance was improving when increasing l and c. Thus, we have run a configuration $C2 = \{p : 4, q : 1, d : 200, l : 100, c : 50, n : 100\}$, which achieved better performance on the validation set. For LastFM, we started from the configuration $C1$, and then explored the ranges: $p \in \{1, 4\}$, $q \in \{1, 4\}$, $c \in \{30, 40, 50, 60\}$, $l \in \{60, 100, 120\}$, $n \in \{50, 100\}$. We found the configuration $C3 = \{p : 4, q : 4, d : 200, l : 100, c : 60, n : 100\}$ to be optimal on the validation set in this range. For LibraryThing, we have explored the ranges $p \in \{1, 4\}$, $q \in \{1, 4\}$, $c \in \{30, 50\}$, keeping $l = 100$, $n = 100$, $d = 200$. The optimal configuration is: $C4 = \{p : 1, q : 1, d : 200, l : 100, c : 50, n : 100\}$. The hyper-parameters of the LambdaMart (Burges, 2010) algorithm have been left to their default values as reported in the implementation that has been used for this work.[10] In general, we can observe that, for all datasets in consideration, using long walks ($l = 100$), many walks per entity ($n = 100$), and a large context size such as $c = 50$ improves the quality of the recommendations. This kind of configuration ought to be used as a starting point in configuring entity2rec with new datasets.

[9] `https://github.com/D2KLab/entity2rec`

[10] `https://github.com/jma127/pyltr`

	$\mathbf{K_p}$			$\mathbf{K_p^+}$		
property	**N**	**M**	κ	**N**	**M**	κ
dbo:cinematography	2757	2136	1.6	9881	382636	77.4
dbo:director	4607	3142	1.4	10881	383642	70.5
dbo:distributor	3180	3180	2.0	9594	383680	80.0
dbo:editing	2292	1809	1.6	9851	382309	77.6
dbo:musicComposer	3682	3031	1.7	10322	383531	74.3
dbo:producer	4358	3817	1.8	11135	384317	69.0
dbo:starring	8969	13990	3.1	15375	394490	51.3
dbo:writer	5044	3836	1.5	11668	384336	65.9
dct:subject	9809	49897	10.2	15967	430397	53.9
feedback	9119	380500	83.5	n/a	n/a	n/a

Table 2: Network stats for Movielens 1 M for K_p and K_p^+. N = n_nodes, M = n_edges, κ = average_degree.

6. Results

6.1. Hybrid property-specific subgraphs

In this section, we compare the new version of entity2rec based on hybrid property specific subgraphs to the one proposed in Palumbo et al. (2017), i.e. entity2rec to entity2rec (2017). In Table 2, Table 3 and Table 4, we report statistics for the property-specific subgraphs K_p and K_p^+ for Movielens 1M, LastFM and LibraryThing respectively. It can be noticed that for many content properties, the average degree is small, indicating that items are connected to few entities, as discussed in Section 1. On the other hand, hybrid property-specific subgraphs always have a better connectivity in terms of average degree of the nodes.

We have compared entity2rec to entity2rec (2017) for the three datasets under analysis. The results are reported in Table 5, Table 6 and Table 7. The first finding is that entity2rec consistently obtains better scores for different configurations of the hyper-parameters for the three datasets, proving the effectiveness of using hybrid property-specific subgraphs as suggested in this work. The second finding is that using learning to rank ($entityrec_{lambda}$) is crucial to obtain good recommendations for entity2rec (2017), but it is no longer useful for entity2rec, especially if hyper-parameters are properly optimized. In order to interpret this result, we remind the reader that, as shown

	$\mathbf{K_p}$			$\mathbf{K_p^+}$		
property	**N**	**M**	κ	**N**	**M**	κ
dbo:associatedBand	15212	19492	2.6	20169	74444	7.4
dbo:associatedMusicalArtist	15211	19491	2.6	20168	74443	7.4
dbo:bandMember	9768	7587	1.6	17113	62539	7.3
dbo:birthPlace	5650	5581	1.9	12992	60533	9.3
dbo:formerBandMember	8870	7481	1.7	16621	62433	7.5
dbo:genre	10258	26064	5.1	13034	81016	12.4
dbo:hometown	9646	12386	2.6	13870	67338	9.7
dbo:instrument	2236	4411	3.9	11137	59363	10.7
dbo:occupation	2211	3246	2.9	10970	58198	10.6
dbo:recordLabel	12159	20279	3.3	15938	75231	9.4
dct:subject	25827	88375	6.8	27946	143327	10.3
feedback	10147	54952	10.8	n/a	n/a	n/a

Table 3: Network stats for LastFM for K_p and K_p^+. N = n_nodes, M = n_edges, κ = average_degree.

	$\mathbf{K_p}$			$\mathbf{K_p^+}$		
property	**N**	**M**	κ	**N**	**M**	κ
dbo:author	13132	9757	1.5	20564	194416	18.9
dbo:country	2015	1921	1.9	16657	186580	22.4
dbo:coverArtist	1990	1438	1.4	17126	186097	21.7
dbo:language	1975	1906	1.9	16630	186565	22.4
dbo:literaryGenre	7211	8526	2.4	17292	193185	22.3
dbo:mediaType	4632	5539	2.4	16676	190198	22.8
dbo:previousWork	4845	3261	1.3	17559	187920	21.4
dbo:publisher	8696	8241	1.9	17878	192900	21.6
dbo:series	3317	2562	1.5	17319	187221	21.6
dbo:subsequentWork	5928	3930	1.3	18245	188589	20.7
dct:subject	18903	51324	5.4	26138	235983	18.1
feedback	16501	184659	22.4	n/a	n/a	n/a

Table 4: Network stats for LibraryThing for K_p and K_p^+. N = n_nodes, M = n_edges, κ = average_degree.

System	P@5	P@10	R@5	R@10	SER@5	SER@10	NOV@5	NOV@10
$entity2rec_{lambda}$ (C2)	0.2125	0.182	0.0967	0.1564	0.1913	0.1526	9.6541	9.6521
entity2rec$_{avg}$ (C2)	**0.2372**	**0.2013**	**0.1045**	**0.1691**	**0.2125**	**0.1676**	9.5770	9.5778
$entity2rec_{min}$ (C2)	0.2198	0.1837	0.0976	0.1547	0.1946	0.1504	9.4661	9.4731
$entity2rec_{max}$ (C2)	0.2206	0.1895	0.0951	0.1556	0.2038	0.1645	10.0462	10.0583
$entity2rec_{lambda}$ (2017) (C2)	0.1836	0.1557	0.0748	0.1208	0.1640	0.1319	9.9477	9.9492
$entity2rec_{avg}$ (2017) (C2)	0.0578	0.0520	0.0234	0.0385	0.0523	0.0408	11.0848	11.0976
$entity2rec_{min}$ (2017) (C2)	0.0166	0.0183	0.0090	0.0164	0.0166	0.0181	11.5409	11.5677
$entity2rec_{max}$ (2017) (C2)	0.0099	0.0087	0.0023	0.0039	0.0095	0.0081	12.1286	12.0962
$entity2rec_{lambda}$ (C1)	0.2221	0.1915	0.0988	0.1592	0.2021	0.1655	9.8909	9.9055
$entity2rec_{avg}$ (C1)	0.2265	0.1936	0.0997	0.1613	0.2051	0.1658	9.8195	9.8331
$entity2rec_{min}$ (C1)	0.2020	0.1726	0.0912	0.1467	0.1787	0.1451	9.7436	9.7550
$entity2rec_{max}$ (C1)	0.1954	0.1672	0.0865	0.1388	0.1825	0.1480	10.2081	10.2742
$entity2rec_{lambda}$ (2017) (C1)	0.1670	0.1413	0.0707	0.1131	0.1521	0.1237	10.3686	10.3843
$entity2rec_{avg}$ (2017) (C1)	0.0100	0.0112	0.0052	0.0101	0.0100	0.0110	**14.5820**	**14.0342**
$entity2rec_{min}$ (2017) (C1)	0.0214	0.0200	0.0107	0.0187	0.0213	0.0197	12.2140	12.1437
$entity2rec_{max}$ (2017) (C1)	0.0069	0.0075	0.0020	0.0043	0.0069	0.0074	12.4162	12.4051

Table 5: entity2rec outperforms entity2rec (2017) for different configurations of hyper-parameters on Movielens 1M ($C1 = \{p : 4, q : 1, d : 200, l : 100, c : 30, n : 50\}$, $C2 = \{p : 4, q : 1, d : 200, l : 100, c : 50, n : 100\}$). In entity2rec as presented in this work, the learning to rank is no longer useful, as the unsupervised approach appears to be more effective. Results can be considered with no error for comparisons as the standard deviation is negligible up to the reported precision.

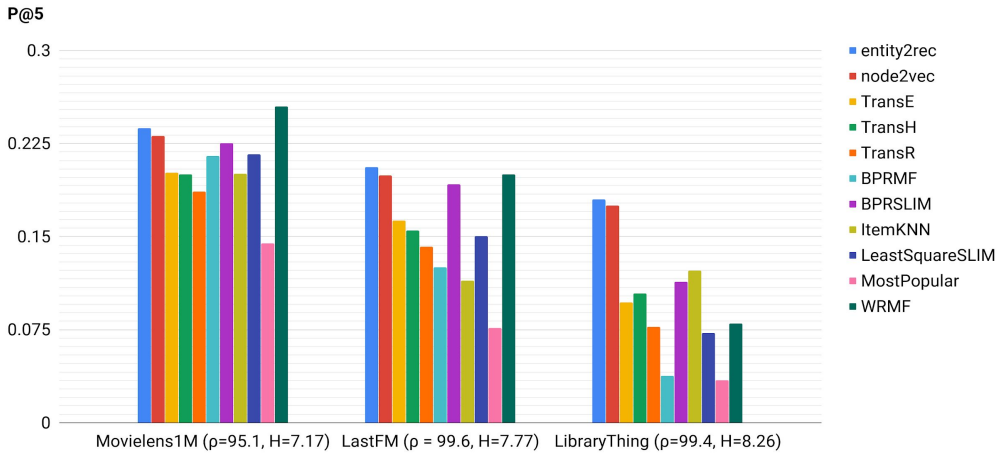
	P@5	P@10	R@5	R@10	SER@5	SER@10	NOV@5	NOV@10
$entity2rec_{lambda}$ (C3)	0.1852	0.1506	0.1066	0.1736	0.1512	0.1526	10.1008	10.2065
entity2rec$_{avg}$ (C3)	**0.2062**	**0.158**	**0.1191**	**0.1823**	**0.1682**	**0.1676**	10.3794	10.514
$entity2rec_{min}$ (C3)	0.2055	0.1571	0.1191	0.1823	0.1664	0.1504	9.8074	9.8965
$entity2rec_{max}$ (C3)	0.1693	0.1366	0.0986	0.1589	0.1423	0.1645	10.2432	10.4455
$entity2rec_{lambda}$ (2017) (C3)	0.1469	0.1227	0.0844	0.1411	0.1194	0.1319	11.0919	11.1020
$entity2rec_{avg}$ (2017) (C3)	0.0597	0.0508	0.0351	0.0598	0.0574	0.0408	13.1432	13.1107
$entity2rec_{min}$ (2017) (C3)	0.0002	0.0001	0.0001	0.0001	0.0002	0.0181	13.0900	11.9789
$entity2rec_{max}$ (2017) (C3)	0.1387	0.1104	0.0801	0.1274	0.1063	0.0081	11.4262	11.4113
$entity2rec_{lambda}$ (C1)	0.1745	0.1372	0.1009	0.1584	0.1405	0.1655	11.2273	11.3843
$entity2rec_{avg}$ (C1)	0.1505	0.1182	0.0870	0.1367	0.1182	0.1658	12.2672	12.3319
$entity2rec_{min}$ (C1)	0.1699	0.1321	0.0981	0.1532	0.1343	0.1451	11.3311	11.4215
$entity2rec_{max}$ (C1)	0.1295	0.1085	0.0753	0.1258	0.1037	0.1480	10.5371	10.8922
$entity2rec_{lambda}$ (2017) (C1)	0.1054	0.0914	0.0611	0.1060	0.0810	0.1237	12.2735	12.2602
$entity2rec_{avg}$ (2017) (C1)	0.0396	0.0353	0.0238	0.0423	0.0380	0.0110	**13.6894**	**13.649**
$entity2rec_{min}$ (2017) (C1)	0.0002	0.0001	0.0001	0.0001	0.0002	0.0197	13.0893	11.9755
$entity2rec_{max}$ (2017) (C1)	0.0952	0.0764	0.0553	0.0889	0.0651	0.0074	12.4498	12.4131

Table 6: entity2rec outperforms entity2rec (2017) for different configurations of hyper-parameters on LastFM ($C1 = \{p : 4, q : 1, d : 200, l : 100, c : 30, n : 50\}$, $C3 = \{p : 4, q : 4, d : 200, l : 100, c : 60, n : 100\}$). In entity2rec as presented in this work, the learning to rank is no longer useful, as the unsupervised approach appears to be more effective. Results can be considered with no error for comparisons as the standard deviation is negligible up to the reported precision.

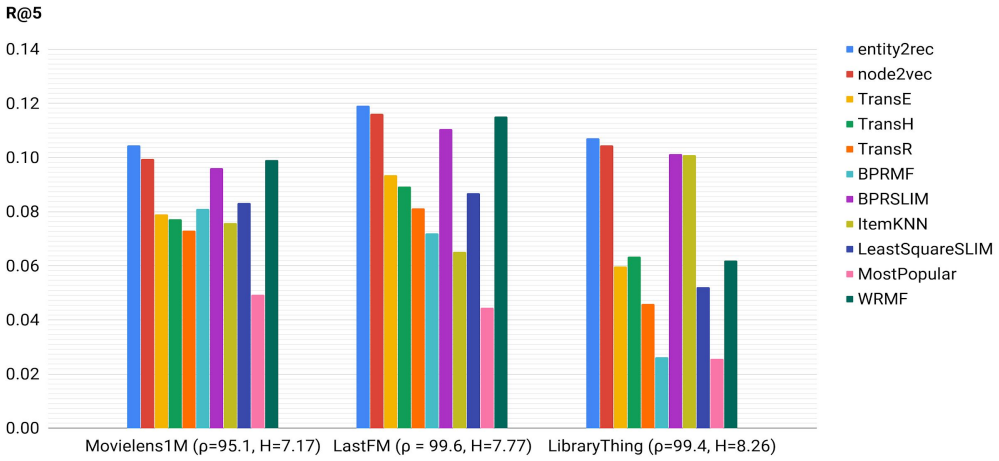
	P@5	P@10	R@5	R@10	SER@5	SER@10	NOV@5	NOV@10
entity2rec$_{lambda}$ (C4)	0.1271	0.1091	0.0803	0.12680	0.1229	0.1025	12.4693	12.7922
entity2rec$_{avg}$ (C4)	0.1800	0.1398	0.1072	0.15640	0.1736	0.1306	12.886	12.0837
entity2rec$_{\mathbf{min}}$ (C4)	**0.1831**	**0.1410**	**0.1084**	**0.15750**	**0.1757**	**0.1309**	11.7088	11.6704
entity2rec_max (C4)	0.1634	0.1304	0.0984	0.14800	0.1591	0.1230	12.7834	12.7289
entity2rec$_{lambda}$ (2017) (C4)	0.1322	0.1026	0.0746	0.11030	0.1285	0.0978	12.9999	13.1216
entity2rec$_{avg}$ (2017) (C4)	0.0720	0.0478	0.0495	0.06290	0.0719	0.0477	13.4809	13.5570
entity2rec_min (2017) (C4)	0.0060	0.0044	0.0027	0.00370	0.0060	0.0044	13.3955	13.5335
entity2rec_max (2017) (C4)	0.0319	0.0196	0.0250	0.03130	0.0316	0.0194	14.5486	14.6138
entity2rec$_{lambda}$ (C1)	0.1396	0.1135	0.0815	0.12630	0.1365	0.1092	13.2614	13.3021
entity2rec$_{avg}$ (C1)	0.1678	0.1298	0.1014	0.14700	0.1639	0.1248	12.7306	12.7946
entity2rec_min (C1)	0.1735	0.1344	0.1041	0.15100	0.1689	0.1283	12.3012	12.3449
entity2rec_max (C1)	0.1411	0.1124	0.0865	0.13020	0.1390	0.1093	13.6293	13.6267
entity2rec$_{lambda}$ (2017) (C1)	0.1160	0.0926	0.0689	0.10330	0.1132	0.0895	13.4436	13.5271
entity2rec$_{avg}$ (2017) (C1)	0.0661	0.0447	0.0453	0.05810	0.0661	0.0446	13.2703	13.4478
entity2rec_min (2017) (C1)	0.0041	0.0032	0.0018	0.00260	0.0041	0.0032	13.5453	13.5686
entity2rec_max (2017) (C1)	0.0129	0.0091	0.0122	0.01680	0.0128	0.0091	**14.339**	**14.3491**

Table 7: entity2rec outperforms entity2rec (2017) for different configurations of hyper-parameters on LibraryThing ($C1 = \{p : 4, q : 1, d : 200, l : 100, c : 30, n : 50\}$, $C4 = \{p : 1, q : 1, d : 200, l : 100, c : 50, n : 100\}$). In entity2rec as presented in this work, the learning to rank is no longer useful, as the unsupervised approach appears to be more effective. Results can be considered with no error for comparisons as the standard deviation is negligible up to the reported precision.

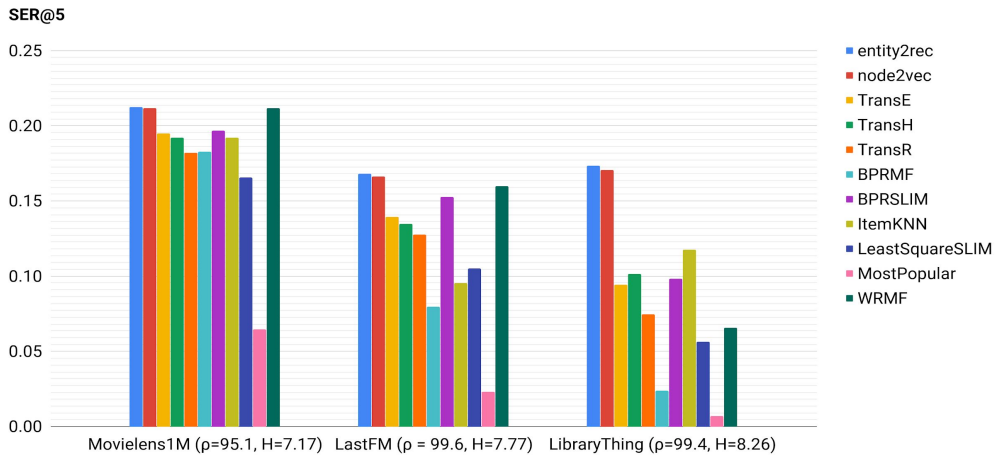
in Palumbo et al. (2017), the most relevant information to make predictions comes from the user-item interaction, i.e. from the 'feedback' property. In entity2rec (2017), the feedback property was contained only in the 'feedback' subgraph, and thus the learning to rank was fundamental to attribute different weights to the properties. On the other hand, for hybrid property-specific subgraphs, user-item interactions are present for all the properties, and the feature learning process, with an appropriate configuration of the hyper-parameters, is able to encode effectively all the necessary information to make recommendations, so that the learning to rank algorithm appears redundant and even damaging, and a simple unsupervised approach such as averaging the features is more effective. Note that the scores of entity2rec (2017) reported in the tables are generally lower than those reported in the original paper Palumbo et al. (2017). This is due to the fact that the evaluation protocol is different. In Palumbo et al. (2017), the set of candidate items did not include all the unrated items, but only a random subset of 100 unrated items per user, making the task easier.


P@5
0.3
0.225
0.15
0.075
0
Movielens1M (ρ=95.1, H=7.17)
LastFM (ρ = 99.6, H=7.77)
LibraryThing (ρ=99.4, H=8.26)
entity2rec
node2vec
TransE
TransH
TransR
BPRMF
BPRSLIM
ItemKNN
LeastSquareSLIM
MostPopular
WRMF

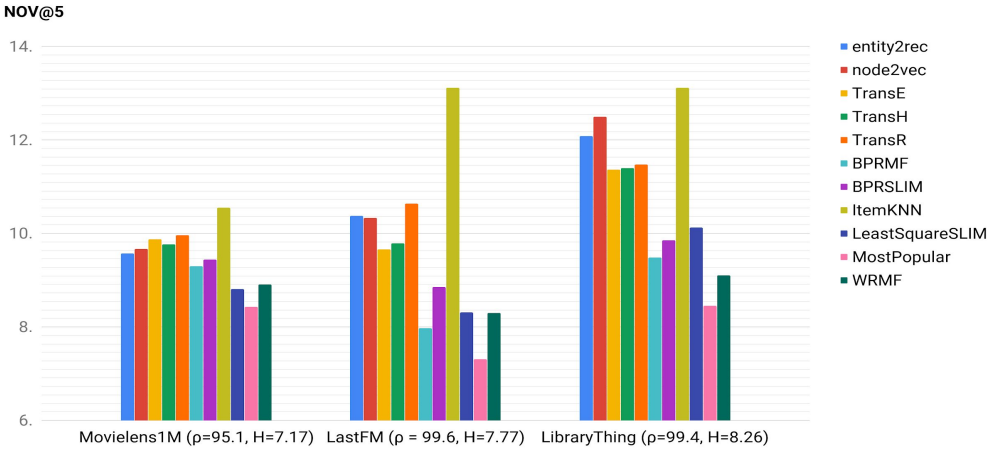
(a)


R@5
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
Movielens1M (ρ=95.1, H=7.17)
LastFM (ρ = 99.6, H=7.77)
LibraryThing (ρ=99.4, H=8.26)
entity2rec
node2vec
TransE
TransH
TransR
BPRMF
BPRSLIM
ItemKNN
LeastSquareSLIM
MostPopular
WRMF

(b)



(c)



(d)

Figure 6: Results on Movielens 1M, LastFM, and LibraryThing datasets for P@5, R@5, SER@5, and NOV@5. entity2rec performs well for all datasets, but it is especially effective for LibraryThing, where sparsity and entropy are high. Scores are reported in tabular form in Appendix B. entity2rec refers to $entity2rec_{avg}(C1)$, $entity2rec_{avg}(C2)$, and $entity2rec_{avg}(C3)$ for Movielens 1M, LastFM, and LibraryThing respectively. node2vec refers to $node2vec(C1)$, $node2vec(C2)$ and $node2vec(C3)$ for Movielens 1M, LastFM, and LibraryThing respectively. Results can be considered with no error for comparisons as the standard deviation is negligible up to the reported precision.

6.2. Comparison with state-of-the-art recommender systems

We have measured precision, recall, serendipity and novelty for all the recommender systems and the three datasets under analysis. We can see from Figure 6a, Figure 6b, and Figure 6c that entity2rec outperforms competing systems for all datasets for P@5, R@5, and SER@5, except for P@5 in Movielens 1M where WRMF is performing better. As we can see from Figure 6d, WRMF is characterized by low novelty of the recommendations, i.e. it tends to recommend very popular items. This proves to be effective in Movielens 1M and LastFM that have a high popularity bias as shown by the entropy value (Table 1), the distribution of items (Figure 5), previous literature (Cremonesi et al., 2010) and the effectiveness of the MostPopular baseline. Looking at SER@5, i.e. considering the top-5 items as non relevant, entity2rec has a slightly better score than WRMF. Movielens 1M is also characterized by a lower sparsity with respect to the other datasets, and this favors collaborative filtering systems, which are known to suffer from data sparsity (Adomavicius and Tuzhilin, 2005). In fact, all of collaborative filtering systems, even ItemKNN that does not perform dimensionality reduction, achieve scores above 20%. LastFM is much sparser than Movielens 1M, and this affects the performance of ItemKNN, whereas matrix factorization based techniques maintain good scores. In LibraryThing, where the popularity bias is weaker (Figure 5), entity2rec is significantly more effective than competing systems and matrix factorization techniques are much less well performing with respect to the other two datasets. ItemKNN, on the other hand, is outperforming matrix factorization techniques. Looking at R@5 (Figure 6b) we can see that entity2rec outperforms all competing systems, also in the Movielens 1M dataset, where WRMF has a better precision. Since R@5 is weighting the number of hits by the number of relevant items for the user, this shows that entity2rec generally works better than other systems with users having fewer relevant items.

In terms of novelty, ItemKNN is the best performing system (Figure 6d), but it is not comparable to competing systems in terms of P@5, R@5 and SER@5. Recommender systems based on knowledge graph embeddings (entity2rec, node2vec, translational models) have better novelty with respect to collaborative filtering systems. We also observe that entity2rec outperforms node2vec for all the datasets and for P@5, R@5, and SER@5, justifying the creation of property-specific embeddings that are aggregated in a later stage, rather than embedding the whole knowledge graph. Furthermore, we observe that using knowledge graph embeddings approaches based on neural

language models such as entity2rec and node2vec is more effective than using translational models.

In general, we can say that entity2rec generates accurate (high precision and recall) and non-obvious (high serendipity, good novelty) recommendations and it is particularly effective with respect to state-of-the-art systems when the sparsity and the entropy of the datasets are high (e.g. LibraryThing). A tabular representation of the scores on the three datasets is reported in Appendix B.

6.3. Feature interpretability

In this section, we address the question of the interpretation of the entity2rec model. We focus on $entity2rec_{avg}$, as it proved to be more effective on the three datasets. $entity2rec_{avg}$, as described in Section 4, is the average of property-specific relatedness scores:

$$entity2rec_{avg}(u, i) = avg(\{\rho_p(u, i)\}_{p \in \Gamma}) \tag{29}$$

where the property-specific relatedness scores are obtained from the embeddings of the property-specific subgraphs, containing user-item interactions and item relations p to other entities. With respect to most knowledge-aware recommender systems based on metapaths, the interpretation is thus easier, as it considers one property at the time. We report in Table 8, Table 9, and Table 10 the scores of using a single property ρ_p as a ranking function. We can see that for all datasets, the information coming from "dct:subject" is quite relevant. Then, for movies, the starring actors and the director appear to be strong features, for musical artists the record label and the genre, and for books previous and subsequent work. In general, none of the features individually outperforms the average of the features.

The simplicity of the final ranking function and the ability to interpret the model in an easy way has several positive consequences. An in-depth discussion of this point is matter of future work, but we mention two big advantages. The first is that entity2rec can be easily used in an interactive interface, as a sort of multicriteria recommender system, replacing Eq. 29 with a weighted average of the property-specific relatedness score. For example, the user might assign more weight to the "director" property and recommendations could change accordingly. The second is the possibility of explaining recommendations along different dimensions. Explanations can be generated in terms of related items, looking at what items of the users'

property	P@5	P@10	R@5	R@10	SER@5	SER@10	NOV@5	NOV@10
feedback_dbo:cinematography	0.1847	0.1599	0.0813	0.1323	0.1675	0.1341	9.835	9.813
feedback_dbo:director	0.1913	0.1615	0.0842	0.1344	0.1741	0.1354	9.859	9.859
feedback_dbo:distributor	0.1846	0.1581	0.0805	0.1309	0.1673	0.1337	9.894	9.903
feedback_dbo:editing	0.1829	0.1565	0.0810	0.1299	0.1668	0.1320	9.855	9.829
feedback_dbo:musicComposer	0.1861	0.1598	0.0817	0.1310	0.1691	0.1350	9.891	9.882
feedback_dbo:producer	0.1777	0.1500	0.0826	0.1322	0.1603	0.1267	10.349	10.457
feedback_dbo:starring	0.2113	0.1794	0.0937	0.1507	0.1965	0.1559	9.957	10.028
feedback_dbo:writer	0.1808	0.1483	0.0822	0.1285	0.1652	0.1269	**10.393**	**10.599**
feedback_dct:subject	**0.2249**	**0.1932**	**0.0958**	**0.1555**	**0.2044**	**0.1658**	9.831	9.822

Table 8: Feature evaluation for Movielens 1M dataset. The most effective features appear to be the subject, the starring actors and the director of the movie. The writer and the producer introduce more novelty. Results can be considered with no error for comparisons as the standard deviation is negligible up to the reported precision.

property	P@5	P@10	R@5	R@10	SER@5	SER@10	NOV@5	NOV@10
feedback_dbo:associatedBand	0.1539	0.1260	0.0894	0.1459	0.1253	0.0949	11.050	11.102
feedback_dbo:associatedMusicalArtist	0.1575	0.1284	0.0915	0.1486	0.1299	0.0970	10.950	11.081
feedback_dbo:bandMember	0.1511	0.1231	0.0873	0.1416	0.1217	0.0910	***11.60***	***11.66***
feedback_dbo:birthPlace	0.1612	0.1289	0.0925	0.1482	0.1287	0.0942	11.080	11.137
feedback_dbo:formerBandMember	0.1580	0.1264	0.0909	0.1454	0.1274	0.0929	11.480	11.498
feedback_dbo:genre	0.1801	0.1428	0.1042	0.1652	0.1466	0.1083	10.330	10.451
feedback_dbo:hometown	0.1708	0.1374	0.0979	0.1573	0.1371	0.1034	10.360	10.414
feedback_dbo:instrument	0.1601	0.1281	0.0919	0.1475	0.1270	0.0925	11.250	11.285
feedback_dbo:occupation	0.1457	0.1154	0.0844	0.1336	0.1103	0.0809	10.960	10.932
feedback_dbo:recordLabel	0.1856	0.1499	0.1076	0.1735	0.1532	0.115	10.420	10.463
feedback_dct:subject	**0.1954**	**0.1550**	**0.1131**	**0.1799**	**0.1655**	**0.1239**	10.190	10.271

Table 9: Feature evaluation for LastFM dataset. The most effective features appear to be the subject, the record label and the genre. The instruments and the former band players introduce more novelty. Results can be considered with no error for comparisons as the standard deviation is negligible up to the reported precision.

profile are more similar to the recommended ones using the global $\rho(u, i)$ (e.g. "Because you liked Titanic"); in terms of properties, by comparing the property-specific relatedness scores $\rho_p(u, i)$ and using the highest scores to unravel specific properties of the item to which the user is more related (e.g. "Because you may like the cast"); in terms of item content, since property-specific relatedness scores $\rho_p(u, e)$ can be measured between a user u and any entity of the knowledge graph $e \in E$, not just for items $i \in I$ (e.g. "Because you may like Steve Buscemi").

7. Conclusions and future work

In this paper, we have presented entity2rec, a recommender system based on property-specific knowledge graph embeddings. We have introduced a new way of creating property-specific subgraphs and to aggregate the relatedness

property	P@5	P@10	R@5	R@10	SER@5	SER@10	NOV@5	NOV@10
feedback_dbo:author	0.1603	0.1279	0.0972	0.1463	0.1556	0.1208	**12.736**	**12.645**
feedback_dbo:country	0.1629	0.1254	0.0976	0.1407	0.1572	0.1171	12.360	12.372
feedback_dbo:coverArtist	0.1625	0.1250	0.0973	0.1406	0.1571	0.1170	12.362	12.366
feedback_dbo:language	0.1619	0.1258	0.0971	0.1407	0.1558	0.1172	12.350	12.363
feedback_dbo:literaryGenre	0.1633	0.1273	0.0978	0.1430	0.1583	0.1199	12.353	12.361
feedback_dbo:mediaType	0.1610	0.1241	0.0956	0.1399	0.1559	0.1167	12.411	12.444
feedback_dbo:previousWork	0.1688	0.1321	0.1001	0.1484	0.1630	0.1241	12.523	12.458
feedback_dbo:publisher	0.1643	0.1273	0.0979	0.1430	0.1588	0.1196	12.326	12.319
feedback_dbo:series	0.1635	0.1280	0.0976	0.1438	0.1581	0.1204	12.460	12.433
feedback_dbo:subsequentWork	0.1687	0.1307	0.1007	0.1465	0.1634	0.1227	12.520	12.458
feedback_dct:subject	**0.1733**	**0.1362**	**0.1037**	**0.1537**	**0.1684**	**0.1275**	12.031	11.997

Table 10: Feature evaluation for LibraryThing dataset. The most effective features appear to be the subject, the previous and following works. The author introduces more novelty. Results can be considered with no error for comparisons as the standard deviation is negligible up to the reported precision.

scores into the final ranking function, and conducted several experiments to address our research questions:

RQ1 *How do hybrid property-specific subgraph perform with respect to collaborative-content property-specific subgraphs, namely does entity2rec work better than entity2rec (2017)?* Experiments reported in Section 6.1 show that hybrid property-specific subgraphs significantly enhance the recommendation quality for all the datasets.

RQ2 *How do different aggregation functions perform in generating the user-item relatedness score from property-specific relatedness scores?* Experiments in Section 6.1 show that, when building content-collaborative property-specific subgraphs as in entity2rec (2017), a learning to rank algorithm to combine the property-specific relatedness scores is fundamental to obtain good recommendations. However, when using hybrid property-specific subgraphs and a proper configuration of the hyper-parameters (long walks, many walks per entity, large context size), the learning to rank is no longer beneficial.

RQ3 *How does entity2rec perform with respect to collaborative filtering systems in terms of accuracy, serendipity and novelty of the recommendations?* Experiments in Section 6.2 show that entity2rec outperforms all collaborative filtering systems for accuracy-based metrics for all datasets, but P@5 in Movielens 1M, where WRMF obtains the best score. However, WRMF shows to consistently have a little level of

novelty, which highlights the fact that predictions tend to be concentrated on highly popular items. In particular, entity2rec appears to be particularly effective for the LibraryThing dataset. This dataset is characterized by a high sparsity, which is similar to a real world scenario, and a lower popularity bias with respect to the other datasets.

RQ4 *How does entity2rec performs with respect to other knowledge graph embeddings based systems for item recommendation? Is it justified to generate property-specific embeddings to leverage the semantics of the graphs, or is node2vec enough?* Experiments in Section 6.1 show translational models appear to be less effective than node2vec-based models for generating recommendations. entity2rec appears the most effective way to generate recommendations using knowledge graph embeddings among the systems under analysis. The fact that entity2rec outperforms node2vec on all the datasets shows that it is justified to generate property-specific embeddings and aggregate the scores, rather than creating embeddings of the whole graph without taking into account the semantics of the properties.

RQ5 *What is the interpretation of entity2rec features and what is their importance?* The recommender model of entity2rec has a straight forward interpretation, as shown in Section 6.3. $entity2rec_{avg}$ aggregates several property-specific relatedness scores, whose interpretation is a measure of the relatedness between the user and the item with respect to specific aspects of the item content. "dct:subject" appears to be effective for all datasets. Then, for movies, the starring actors and the director appear to be strong features, for musical artists the record label and the genre, and for books previous and subsequent work.

RQ6 *How can entity2rec be configured for specific user requirements and to explain recommendations?* In Section 6.3, we show that, since $entity2rec_{avg}$ averages property-specific relatedness scores, the aggregation function could be easily modified according to weights entered by a user in an interactive interface. Moreover, property-specific relatedness scores can be leveraged to generate rich explanations of the recommendations, in terms of related past items, item properties and item content.

As a summary, we believe that the strengths of entity2rec that make it outperform competing approaches are:

- its ability of leveraging a very rich and diverse source of data (KG containing both content-based and collaborative information).
- extending a state-of-the-art graph embedding approach (node2vec), which has proven to be very effective for prediction tasks. Also, the use of property-specific embeddings that are then aggregated in a "divide and conquer" approach can be seen as a sort of "ensemble method", which are known to boost accuracy in machine learning tasks.

entity2rec is domain-agnostic, as we have applied it to movie, book, and musical artist recommendations. However, should it be applied to a new domain, it would require an ontology that models the item properties.

As a future work, we will perform an online experiment testing the effectiveness of the richer explanations generated by entity2rec, as well as the users' appreciation of the configurability of the recommender interface. We will also deal with the problems of having a recommender system working online, i.e. how to generate recommendations for new users (e.g. asking to rate a set of seed items and using item-to-item relatedness scores), and of updating entity2rec when new data (users, items, and/or feedback) comes in without retraining the model from scratch.

8. Acknowledgements

This work was partially supported by EIT Digital within the innovation activity PasTime (grant number 17164).

9. Declaration of interests

Declarations of interest: none

References

Adomavicius, G., Tuzhilin, A., 2005. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. IEEE transactions on knowledge and data engineering 17, 734–749.

Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., Ives, Z., 2007. Dbpedia: A nucleus for a web of open data, in: The semantic web. Springer, pp. 722–735.

Bengio, Y., Courville, A., Vincent, P., 2013. Representation learning: A review and new perspectives. IEEE transactions on pattern analysis and machine intelligence 35, 1798–1828.

Billsus, D., Pazzani, M.J., 1998. Learning collaborative information filters., in: Icml, pp. 46–54.

Bizer, C., Heath, T., Berners-Lee, T., 2009. Linked data-the story so far. Semantic Services, Interoperability and Web Applications: Emerging Concepts , 205–227.

Bordes, A., Usunier, N., Garcia-Duran, A., Weston, J., Yakhnenko, O., 2013. Translating embeddings for modeling multi-relational data, in: Advances in neural information processing systems, pp. 2787–2795.

Bordes, A., Weston, J., Collobert, R., Bengio, Y., et al., 2011. Learning structured embeddings of knowledge bases., in: AAAI, p. 6.

Burges, C.J., 2010. From ranknet to lambdarank to lambdamart: An overview. Learning 11, 81.

Cantador, I., Brusilovsky, P., Kuflik, T., 2011. 2nd workshop on information heterogeneity and fusion in recommender systems (hetrec 2011), in: Proceedings of the 5th ACM conference on Recommender systems, ACM, New York, NY, USA.

Catherine, R., Cohen, W., 2016. Personalized recommendations using knowledge graphs: A probabilistic logic programming approach, in: Proceedings of the 10th ACM Conference on Recommender Systems, ACM. pp. 325–332.

Cremonesi, P., Koren, Y., Turrin, R., 2010. Performance of recommender algorithms on top-n recommendation tasks, in: Proceedings of the fourth ACM conference on Recommender systems, ACM. pp. 39–46.

Di Noia, T., 2016. Recommender systems meet linked open data, in: International Conference on Web Engineering, Springer. pp. 620–623.

Di Noia, T., Magarelli, C., Maurino, A., Palmonari, M., Rula, A., 2018. Using ontology-based data summarization to develop semantics-aware recommender systems, in: Gangemi, A., Navigli, R., Vidal, M.E., Hitzler, P., Troncy, R., Hollink, L., Tordai, A., Alam, M. (Eds.), The Semantic Web, Springer International Publishing, Cham. pp. 128–144.

Di Noia, T., Ostuni, V.C., Tomeo, P., Di Sciascio, E., 2016. Sprank: Semantic path-based ranking for top-n recommendations using linked open data. ACM Transactions on Intelligent Systems and Technology (TIST) 8, 9.

Figueroa, C., Vagliano, I., Rocha, O.R., Morisio, M., 2015. A systematic literature review of linked data-based recommender systems. Concurrency and Computation: Practice and Experience 27, 4659–4684.

Gantner, Z., Drumond, L., Freudenthaler, C., Schmidt-Thieme, L., 2012. Personalized ranking for non-uniformly sampled items, in: Proceedings of KDD Cup 2011, pp. 231–247.

Gantner, Z., Rendle, S., Freudenthaler, C., Schmidt-Thieme, L., 2011. MyMediaLite: A free recommender system library, in: Proceedings of the 5th ACM Conference on Recommender Systems (RecSys 2011).

Ge, M., Delgado-Battenfeld, C., Jannach, D., 2010. Beyond accuracy: Evaluating recommender systems by coverage and serendipity, in: Proceedings of the fourth ACM conference on Recommender Systems, ACM Press. pp. 257–260. doi:10.1145/1864708.1864761.

de Gemmis, M., Lops, P., Semeraro, G., Musto, C., 2015. An investigation on the serendipity problem in recommender systems. Information Processing & Management 51, 695–717. doi:10.1016/j.ipm.2015.06.008.

Grover, A., Leskovec, J., 2016. node2vec: Scalable feature learning for networks, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM. pp. 855–864.

Harper, F.M., Konstan, J.A., 2016. The movielens datasets: History and context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 19.

Henderson, K., Gallagher, B., Eliassi-Rad, T., Tong, H., Basu, S., Akoglu, L., Koutra, D., Faloutsos, C., Li, L., 2012. Rolx: structural role extraction & mining in large graphs, in: 18th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 1231–1239.

Herlocker, J.L., Konstan, J.A., Terveen, L.G., Riedl, J.T., 2004. Evaluating collaborative filtering recommender systems. ACM Transactions on Information Systems 22, 5–53. doi:10.1145/963770.963772.

Hu, Y., Koren, Y., Volinsky, C., 2008. Collaborative filtering for implicit feedback datasets, in: Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on, Ieee. pp. 263–272.

Kanjirathinkal, R.C., 2017. Explainable Recommendations. Ph.D. thesis. Stanford University.

Lin, Y., Liu, Z., Sun, M., Liu, Y., Zhu, X., 2015. Learning entity and relation embeddings for knowledge graph completion., in: AAAI, pp. 2181–2187.

Liu, T.Y., et al., 2009. Learning to rank for information retrieval. Foundations and Trends® in Information Retrieval 3, 225–331.

Mikolov, T., Chen, K., Corrado, G., Dean, J., 2013a. Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781 .

Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J., 2013b. Distributed representations of words and phrases and their compositionality, in: Advances in neural information processing systems, pp. 3111–3119.

Nickel, M., Murphy, K., Tresp, V., Gabrilovich, E., 2016. A review of relational machine learning for knowledge graphs. Proceedings of the IEEE 104, 11–33.

Nickel, M., Tresp, V., Kriegel, H.P., 2011. A three-way model for collective learning on multi-relational data., in: ICML, pp. 809–816.

Ning, X., Karypis, G., 2011. Slim: Sparse linear methods for top-n recommender systems, in: Data Mining (ICDM), 2011 IEEE 11th International Conference on, IEEE. pp. 497–506.

Ostuni, V.C., Di Noia, T., Di Sciascio, E., Mirizzi, R., 2013. Top-n recommendations from implicit feedback leveraging linked open data, in: Proceedings of the 7th ACM conference on Recommender systems, ACM. pp. 85–92.

Palumbo, E., Rizzo, G., Troncy, R., 2017. Entity2rec: Learning user-item relatedness from knowledge graphs for top-n item recommendation, in: Proceedings of the Eleventh ACM Conference on Recommender Systems, ACM. pp. 32–36.

Palumbo, E., Rizzo, G., Troncy, R., Baralis, E., Osella, M., Ferro, E., 2018a. An empirical comparison of knowledge graph embeddings for item recommendation., in: DL4KGS@ ESWC, pp. 14–20.

Palumbo, E., Rizzo, G., Troncy, R., Baralis, E., Osella, M., Ferro, E., 2018b. Knowledge graph embeddings with node2vec for item recommendation, in: European Semantic Web Conference, Springer. pp. 117–120.

Palumbo, E., Rizzo, G., Troncy, R., Baralis, E., Osella, M., Ferro, E., 2018c. Translational models for item recommendation, in: European Semantic Web Conference, Springer. pp. 478–490.

Perozzi, B., Al-Rfou, R., Skiena, S., 2014. Deepwalk: Online learning of social representations, in: Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM. pp. 701–710.

Powers, D.M., 2011. Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation. Journal of Machine Learning Technologies .

Rendle, S., 2011. Context-aware ranking with factorization models. Springer.

Rendle, S., Freudenthaler, C., Gantner, Z., Schmidt-Thieme, L., 2009. Bpr: Bayesian personalized ranking from implicit feedback, in: Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence, AUAI Press. pp. 452–461.

Ristoski, P., Rosati, J., Di Noia, T., De Leone, R., Paulheim, H., 2018. Rdf2vec: Rdf graph embeddings and their applications. Semantic Web , 1–32.

Rosati, J., Ristoski, P., Di Noia, T., Leone, R.d., Paulheim, H., 2016. Rdf graph embeddings for content-based recommender systems, in: CEUR workshop proceedings, RWTH. pp. 23–30.

Said, A., Bellogín, A., 2014. Comparative recommender system evaluation: benchmarking recommendation frameworks, in: Proceedings of the 8th ACM Conference on Recommender systems, ACM. pp. 129–136.

Sarwar, B., Karypis, G., Konstan, J., Riedl, J., 2001. Item-based collaborative filtering recommendation algorithms, in: Proceedings of the 10th International Conference on World Wide Web, ACM, New York, NY, USA. pp. 285–295. URL: http://doi.acm.org/10.1145/371920.372071, doi:10.1145/371920.372071.

Shi, C., Hu, B., Zhao, X., Yu, P., 2018. Heterogeneous information network embedding for recommendation. IEEE Transactions on Knowledge and Data Engineering .

Socher, R., Chen, D., Manning, C.D., Ng, A., 2013. Reasoning with neural tensor networks for knowledge base completion, in: Advances in neural information processing systems, pp. 926–934.

Steck, H., 2013. Evaluation of recommendations: rating-prediction and ranking, in: Proceedings of the 7th ACM conference on Recommender systems, ACM. pp. 213–220.

Sun, Z., Yang, J., Zhang, J., Bozzon, A., Huang, L.K., Xu, C., 2018. Recurrent knowledge graph embedding for effective recommendation, in: Proceedings of the 12th ACM Conference on Recommender Systems, ACM, New York, NY, USA. pp. 297–305. URL: http://doi.acm.org/10.1145/3240323.3240361, doi:10.1145/3240323.3240361.

Vargas, S., Castells, P., 2011. Rank and relevance in novelty and diversity metrics for recommender systems, in: Proceedings of the fifth ACM conference on Recommender Systems, ACM Press. pp. 109–116. doi:10.1145/2043932.2043955.

Vrandečić, D., Krötzsch, M., 2014. Wikidata: a free collaborative knowledgebase. Communications of the ACM 57, 78–85.

Wang, H., Zhang, F., Xie, X., Guo, M., 2018. Dkn: Deep knowledge-aware network for news recommendation. arXiv preprint arXiv:1801.08284 .

Wang, Q., Mao, Z., Wang, B., Guo, L., 2017. Knowledge graph embedding: A survey of approaches and applications. IEEE Transactions on Knowledge and Data Engineering 29, 2724–2743.

Wang, Z., Zhang, J., Feng, J., Chen, Z., 2014. Knowledge graph embedding by translating on hyperplanes., in: AAAI, pp. 1112–1119.

Yu, X., Ren, X., Sun, Y., Gu, Q., Sturt, B., Khandelwal, U., Norick, B., Han, J., 2014. Personalized entity recommendation: A heterogeneous information network approach, in: Proceedings of the 7th ACM international conference on Web search and data mining, ACM. pp. 283–292.

Zhang, F., Yuan, N.J., Lian, D., Xie, X., Ma, W.Y., 2016. Collaborative knowledge base embedding for recommender systems, in: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining, ACM. pp. 353–362.

Appendix A. Item modeling

In order to create the knowledge graph, we need an item model, i.e. we need to define the properties that describe an item type for the datasets used in this work. To this end, we start from the DBpedia Ontology, that defines properties for different item types[11]. However, a property selection strategy purely based on the schema of the data does not provide any guarantee on how frequently used are those properties in the data. Thus, we opt for an empirical approach where we count what are the DBpedia properties ("dbo:") most frequently used in DBpedia data to describe the items in the datasets. More specifically, for each item i in the dataset, we retrieve all triples (i, p, o) in DBpedia and we count the most frequently occurring properties p. Then, we sort the properties according to their frequency of occurrence and we

[11] `http://mappings.dbpedia.org/server/ontology/classes/`

select the first N so that the frequency of the N+1-th property is less than 50% of the previous one. In this way, we avoid to select a fixed number of properties and we rely on the actual frequency of occurrence to determine the cut-off. Finally, we add "dct:subject" to the set of properties, as it provides an extremely rich categorization of items, as done in previous work (Palumbo et al., 2017; Di Noia et al., 2016). We obtain: ["dbo:director", "dbo:starring", "dbo:distributor", "dbo:writer","dbo:musicComposer", "dbo:producer", "dbo:cinematography", "dbo:editing", "dct:subject"] for Movielens 1M, ["dbo:genre", "dbo:recordLabel", "dbo:hometown", "dbo:associatedBand", "dbo:associatedMusicalArtist", "dbo:birthPlace", "dbo:bandMember", "dbo:formerBandMember", "dbo:occupation", "dbo:instrument", "dct:subject"] for LastFM and ["dbo:author", "dbo:publisher", "dbo:literaryGenre", "dbo:mediaType", "dbo:subsequentWork", "dbo:previousWork", "dbo:series", "dbo:country", "dbo:language", "dbo:coverArtist", "dct:subject"].
Recently, some works such as ABSTAT (Di Noia et al., 2018) have dealt with the problem of finding a selection of properties to create a knowledge graph that optimizes recommender systems accuracy. For example, for the datasets under analysis, ABSTAT property selection in the configuration: $k = 10$, *norep.AbsOccAvgS* is: ["dbo:director", "dbo:starring", "dbo:distributor", "dbo:writer", "dbo:musicComposer", "dbo:producer", "dbo:cinematography", "dbo:music", "dbo:language", "dct:subject"] for Movielens 1M, ["dbo:genre", "dbo:recordLabel", "dbo:hometown","dbo:birthPlace","dbp:placeOfBirth", "dbo:deathPlace", "dbo:field", "dbo:nationality", "dbp:placeOfDeath", "dct:subject"] for LastFM and ["dct:subject", "dbo:author", "dbo:publisher", "dbo:literaryGenre", "dbo:mediaType", "dbo:country", "dbo:language", "dbo:series", "dbo:nonFictionSubject", "dbo:coverArtist"]. We have run an experiment comparing entity2rec on a KG built using our property selection and the ABSTAT selection for the LastFM dataset, which is the one where the properties are differing more, but the scores did not show a consistent improvement of the recommendation quality (Tab. A.11).

Appendix B. Scores

In this section, we report the extended results of the comparison of entity2rec with the state-of-the-art in tabular form. For completeness, we include also:

Property selection	System	P@5	P@10	R@5	R@10	SER@5	SER@10	NOV@5	NOV@10
dbo + frequency (C3)	$entity2rec_{lambda}$	0.1852	0.1506	0.1066	0.1736	0.1512	0.1126	10.101	10.206
dbo + frequency (C3)	**entity2rec$_{avg}$**	**0.2062**	**0.158**	**0.1191**	**0.1823**	**0.1682**	**0.1193**	**10.379**	**10.514**
dbo + frequency (C3)	$entity2rec_{min}$	0.2055	0.1571	0.1191	0.1823	0.1664	0.1171	9.807	9.897
dbo + frequency (C3)	$entity2rec_{max}$	0.1693	0.1366	0.0986	0.1589	0.1423	0.1063	10.243	10.446
ABSTAT (C3)	$entity2rec_{lambda}$	0.1799	0.1410	0.1039	0.1628	0.1419	0.1028	10.343	10.444
ABSTAT (C3)	$entity2rec_{avg}$	0.1915	0.1463	0.1102	0.1683	0.1528	0.1082	10.681	10.782
ABSTAT (C3)	$entity2rec_{min}$	0.2010	0.1537	0.1162	0.1778	0.1604	0.1143	9.830	9.873
ABSTAT (C3)	$entity2rec_{max}$	0.1731	0.1370	0.1003	0.1585	0.1422	0.1047	10.280	10.462
dbo + frequency (C1)	$entity2rec_{lambda}$	0.1745	0.1372	0.1009	0.1584	0.1405	0.1054	11.227	11.384
dbo + frequency (C1)	$entity2rec_{avg}$	0.1505	0.1182	0.0870	0.1367	0.1182	0.0881	12.267	12.332
dbo + frequency (C1)	$entity2rec_{min}$	0.1699	0.1321	0.0981	0.1532	0.1343	0.1001	11.331	11.421
dbo + frequency (C1)	$entity2rec_{max}$	0.1295	0.1085	0.0753	0.1258	0.1037	0.0825	10.537	10.892
ABSTAT (C1)	$entity2rec_{lambda}$	0.1750	0.1386	0.1011	0.1605	0.1433	0.1076	11.251	11.363
ABSTAT (C1)	$entity2rec_{avg}$	0.1390	0.1090	0.0808	0.1263	0.1043	0.0788	12.039	12.087
ABSTAT (C1)	$entity2rec_{min}$	0.1604	0.1284	0.0926	0.1485	0.1264	0.0971	11.320	11.401
ABSTAT (C1)	$entity2rec_{max}$	0.1382	0.1118	0.0800	0.1291	0.1110	0.0848	10.968	11.237

Table A.11: ABSTAT property selection and the heuristics used in this paper ("dbo + frequency") are compared on the LastFM dataset. The heuristics work best for $C3 = \{p : 4, q : 4, d : 200, l : 100, c : 60, n : 100\}$, ABSTAT selection works best for $C1 = \{p : 4, q : 1, d : 200, l : 100, c : 30, n : 50\}$. Scores can be considered with no error for comparisons as the standard deviation is negligible up to the reported precision.

- SoftMarginRankingMF: a matrix factorization model for item prediction optimized for a soft margin (hinge) ranking loss (Rendle, 2011).
- WeightedBPRMF: a weighted version of BPRMF with frequency adjusted sampling (Gantner et al., 2012).

All scores can be considered as without error up to the digit reported in the tables, as the standard deviation is negligible.

System	P@5	P@10	R@5	R@10	SER@5	SER@10	NOV@5	NOV@10
$entity2rec_{lambda}$ (C2)	0.2125	0.1820	0.0967	0.1564	0.1913	0.1526	9.654	9.652
entity2rec$_{\mathbf{avg}}$ (C2)	0.2372	0.2013	**0.1045**	**0.1691**	**0.2125**	0.1676	9.577	9.578
$entity2rec_{min}$ (C2)	0.2198	0.1837	0.0976	0.1547	0.1946	0.1504	9.466	9.473
$entity2rec_{max}$ (C2)	0.2206	0.1895	0.0951	0.1556	0.2038	0.1645	10.046	10.058
node2vec (C2)	0.2313	0.2010	0.0994	0.1649	0.2119	0.1720	9.675	9.656
TransE	0.2014	0.1751	0.0791	0.1304	0.1951	0.1638	9.882	9.948
TransH	0.2001	0.1735	0.0772	0.1290	0.1920	0.1592	9.768	9.841
TransR	0.1864	0.1613	0.0731	0.1231	0.1822	0.1513	9.960	10.059
BPRMF	0.2150	0.1900	0.0809	0.1391	0.1829	0.1431	9.303	9.392
BPRSLIM	0.2252	0.1929	0.0961	0.1562	0.1969	0.1593	9.438	9.564
ItemKNN	0.2004	0.1820	0.0758	0.1325	0.1920	0.1671	10.546	10.378
LeastSquareSLIM	0.2162	0.1808	0.0832	0.1325	0.1656	0.1254	8.811	8.922
MostPopular	0.1446	0.1292	0.0492	0.0849	0.0647	0.0537	8.429	8.524
SoftMarginRankingMF	0.1222	0.1122	0.0405	0.0737	0.1206	0.096	9.721	9.805
WeightedBPRMF	0.0945	0.0865	0.0364	0.0660	0.0900	0.0804	**11.334**	**11.357**
WRMF	**0.255**	**0.2179**	0.0991	0.1605	0.2117	**0.1704**	8.910	9.038

Table B.12: Results for the Movielens 1M dataset. Scores can be considered with no error for comparisons as the standard deviation is negligible up to the reported precision.

System	P@5	P@10	R@5	R@10	SER@5	SER@10	NOV@5	NOV@10
$entity2rec_{lambda}$ (C2)	0.1852	0.1506	0.1066	0.1736	0.1512	0.1126	10.101	10.206
entity2rec$_{\mathbf{avg}}$ (C2)	**0.2062**	**0.158**	**0.1191**	**0.1823**	**0.1682**	**0.1193**	10.379	10.514
$entity2rec_{min}$ (C2)	0.2055	0.1571	0.1191	0.1823	0.1664	0.1171	9.807	9.897
$entity2rec_{max}$ (C2)	0.1693	0.1366	0.0986	0.1589	0.1423	0.1063	10.243	10.446
node2vec (C3)	0.1994	0.1562	0.1161	0.1822	0.1665	0.1234	10.337	10.470
TransE	0.1628	0.1328	0.0935	0.1538	0.1393	0.1084	9.662	9.861
TransH	0.1549	0.1227	0.0892	0.1410	0.1347	0.1005	9.791	9.984
TransR	0.1417	0.1158	0.0812	0.1336	0.1276	0.0986	10.643	10.852
BPRMF	0.1254	0.0968	0.0720	0.1109	0.0798	0.0547	7.976	8.171
BPRSLIM	0.1921	0.1469	0.1106	0.1700	0.1526	0.1085	8.860	9.101
ItemKNN	0.1144	0.1001	0.0652	0.1149	0.0956	0.0747	**13.116**	**12.937**
LeastSquareSLIM	0.1502	0.1129	0.0868	0.1303	0.1050	0.0718	8.307	8.481
MostPopular	0.0764	0.0648	0.0444	0.0750	0.0231	0.0177	7.307	7.480
SoftMarginRankingMF	0.0426	0.0408	0.0244	0.0466	0.0387	0.0342	9.876	9.925
WeightedBPRMF	0.1067	0.0857	0.0611	0.0980	0.0958	0.0744	10.452	10.682
WRMF	0.2003	0.1491	0.1152	0.1711	0.1598	0.1095	8.305	8.561

Table B.13: Results for the LastFM dataset. Scores can be considered with no error for comparisons as the standard deviation is negligible up to the reported precision.

System	P@5	P@10	R@5	R@10	SER@5	SER@10	NOV@5	NOV@10
$entity2rec_{lambda}$ (C2)	0.1271	0.1091	0.0803	0.1268	0.1229	0.1025	12.469	12.792
$entity2rec_{avg}$ (C2)	0.1800	0.1398	0.1072	0.1564	0.1736	0.1306	12.089	12.084
entity2rec$_{\mathbf{min}}$ (C2)	**0.1831**	**0.1410**	**0.1084**	**0.1575**	**0.1757**	0.1309	11.709	11.670
$entity2rec_{max}$ (C2)	0.1634	0.1304	0.0984	0.1480	0.1591	0.1230	12.783	12.729
node2vec (C4)	0.1749	0.1379	0.1046	0.1551	0.1706	**0.1311**	12.495	12.458
TransE	0.0972	0.0791	0.0598	0.0919	0.0944	0.0754	11.370	11.486
TransH	0.1041	0.0828	0.0634	0.0951	0.1016	0.0787	11.397	11.522
TransR	0.0774	0.0648	0.0459	0.0721	0.0748	0.0612	11.474	11.572
BPRMF	0.0377	0.0336	0.0262	0.0463	0.0240	0.0192	9.487	9.546
BPRSLIM	0.1136	0.0866	0.1013	0.1439	0.0982	0.0709	9.853	10.008
ItemKNN	0.1225	0.0995	0.1009	0.1556	0.1177	0.0912	**13.114**	**12.773**
LeastSquareSLIM	0.0722	0.0517	0.0522	0.0700	0.0565	0.0354	10.131	10.580
MostPopular	0.0343	0.0299	0.0256	0.0430	0.0070	0.0056	8.453	8.646
SoftMarginRankingMF	0.0204	0.0191	0.0141	0.0271	0.0183	0.0138	10.313	10.423
WeightedBPRMF	0.0381	0.0324	0.0326	0.0529	0.0375	0.0307	11.830	11.963
WRMF	0.0802	0.0645	0.0620	0.0969	0.0658	0.0487	9.101	9.288

Table B.14: Results for the LibraryThing dataset. Scores can be considered with no error for comparisons as the standard deviation is negligible up to the reported precision.