entity2rec: Learning User-Item Relatedness from Knowledge Graphs for Top-N Item Recommendation

entity2rec: Learning User-Item Relatedness from Knowledge Graphs for Top-N Item Recommendation

Enrico Palumbo
ISMB
Turin, Italy
EURECOM
Sophia Antipolis, France
palumbo@ismb.it

Giuseppe Rizzo
ISMB
Turin, Italy
giuseppe.rizzo@ismb.it

Raphaël Troncy
EURECOM
Sophia Antipolis, France
raphael.troncy@eurecom.fr

ABSTRACT

Knowledge Graphs have proven to be extremely valuable to recommender systems, as they enable hybrid graph-based recommendation models encompassing both collaborative and content information. Leveraging this wealth of heterogeneous information for top-N item recommendation is a challenging task, as it requires the ability of effectively encoding a diversity of semantic relations and connectivity patterns. In this work, we propose entity2rec, a novel approach to learning user-item relatedness from knowledge graphs for top-N item recommendation. We start from a knowledge graph modeling user-item and item-item relations and we learn property-specific vector representations of users and items applying neural language models on the network. These representations are used to create property-specific user-item relatedness features, which are in turn fed into learning to rank algorithms to learn a global relatedness model that optimizes top-N item recommendations. We evaluate the proposed approach in terms of ranking quality on the MovieLens 1M dataset, outperforming a number of state-of-the-art recommender systems, and we assess the importance of property-specific relatedness scores on the overall ranking quality.

KEYWORDS

hybrid recommender system, node2vec, knowledge graph embeddings, knowledge graph, neural language models, word2vec, learning to rank, linked open data

1 INTRODUCTION

In the last years, research on recommender systems has shown that knowledge graphs are beneficial for hybrid recommender systems, achieving high ranking quality and effectively tackling typical problems of collaborative systems such as data sparsity or new items [5, 28]. A line of work focuses on Linked Open Data knowledge graphs [3], which represent a wealth of freely available multi-domain ontological knowledge and have successfully been used in the past to build recommender systems [7–9, 13]. The crucial point to leverage knowledge graphs to perform item recommendations is to be able to effectively model user-item relatedness from this rich heterogeneous network. To this end, it is highly desirable to opt for approaches that are able to automatically learn user and item representations from an optimization problem on this graph of interactions, minimizing the time-consuming endeavour of feature engineering and leading to better performance [2]. At the same time, it is also beneficial to use a recommendation model whose features have a straight forward interpretation and that can thus be easily adapted to a specific recommendation problem. Last year, node2vec, a new approach to effectively perform feature learning from networks, which combines a flexible random walk strategy with the power of neural language models, has been proposed [11]. However, this approach is not tailored to knowledge graphs, as it does not distinguish among types of entities and of relations, and has never been applied to address recommendation problems. Starting from this work, in this paper, we introduce entity2rec[1], a novel approach to measure user-item relatedness for top-N item recommendation. The contributions of the proposed work can be summarized as follows: (a) we use a knowledge graph encompassing both collaborative information from user feedback and item information from Linked Open Data; (b) we learn property-specific vector representations of knowledge graph entities in a completely unsupervised way via node2vec; (c) we compute property-specific relatedness scores between users and items using the obtained vector representations; (d) we combine the property-specific relatedness scores in a global relatedness score using a supervised learning to rank approach optimizing top-N item recommendation; (e) we evaluate the effectiveness of the proposed approach with respect to four baselines; (f) we remark that, thanks to the explicit semantics of the properties of the knowledge graph, the model is easily adaptable to specific recommendation problems, as the features have a clear interpretation.

2 RELATED WORK

Recommendations using knowledge graphs: in the past years, several works have shown the effectiveness of external knowledge resources to enhance the performance of recommender systems. In [5, 28] the authors start from a graph-based data model encompassing both user feedback and item relations to generate personalized entity recommendations. In [19, 20] the authors adopt a hybrid graph-based data model utilizing Linked Open Data to extract metapath-based feature that are fed into a learning to rank framework. In [24] the authors propose a content-based recommender system that automatically learns item representations using a feature learning algorithm on a knowledge graph and show the effectiveness of the learned representations in an Item-based K-Nearest Neighbor method.

[1] https://github.com/MultimediaSemantics/entity2rec

Feature learning from networks: feature learning from networks is a fundamental step in node and edge classification. Recently, some models have adapted neural language models to networks, resulting in very effective feature learning processes. In general, it has been shown that these methods tend to work better than other dimensionality reduction techniques based on matrix factorization approaches [11]. In Deep Walk [21], the authors learn node representations by simulating random walks on a graph, generating sequences that are successively processed by a neural language model. In [11], the authors propose node2vec, an improvement over Deep Walk, as it adopts a more flexible and sophisticated random walk exploration strategy.

Learning to rank: learning to rank consists in applying machine learning algorithms to learn a ranking function from training data. In this work, we focus on listwise methods, i.e. learning algorithms that are able to directly optimize ranking accuracy metrics from training data. More in detail, we use Adarank [27] and LambdaMart [4]. Adarank is based on boosting, weak learners are iteratively trained to correct the errors of the previous iteration and finally combined together in a linear model. Adarank avoids the problem of the discontinuity of ranking accuracy metrics by optimizing an upper bound of the actual loss function. LambdaMart is the combination of LambdaRank and the boosted tree method MART [10]. The key idea is to optimize the loss function by directly modeling its gradients estimates with respect to the model scores.

3 APPROACH

3.1 Knowledge Graph

In this work, a knowledge graph is defined as a set $K = (E, R, O)$ where E is the set of entities, $R \subset ExΓxE$ 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 entities to their type $O : e \in E \rightarrow \Lambda$ and entity types to their related properties $O : \epsilon \in \Lambda \rightarrow \Gamma_\epsilon \subset \Gamma$. Knowledge graph edges are thus triples $(i, p, j) \in R$ where $i \in E$ and $j \in E$ are entities and $p \in \Gamma$ is a property. In our model, entities include users $u \in U \subset E$ and items $i \in I \subset E \setminus U$. An observed positive feedback between a user and an item (rating $r \geq 4$ in this work) is described by a property not defined by the ontology O, which we name 'feedback'. Thus, $p \in \Gamma_\epsilon^+ = \Gamma_\epsilon \cup `feedback'$. Throughout this work, the ontology O is represented by the DBpedia ontology [1] and we deal with items of the type ϵ ='dbo:Film'. The properties $\Gamma_{dbo:Film}$ related to an entity of the type 'Film' are available online[2].

3.2 Property-specific user-item relatedness

As mentioned in Sec. 1, node2vec [11] has recently shown to be particularly effective at learning vectorial node representations. In a nutshell, node2vec works by simulating a random walk on the graph, generating sequences of nodes, which are then fed into a neural language model (word2vec [18]) as if they were 'sentences' of a document to learn vector representations of the nodes. From these representations, the relatedness between two nodes can easily be computed using vector similarity measures. However, in a knowledge graph different properties have different semantic values and

[2]http://mappings.dbpedia.org/server/ontology/classes/Film

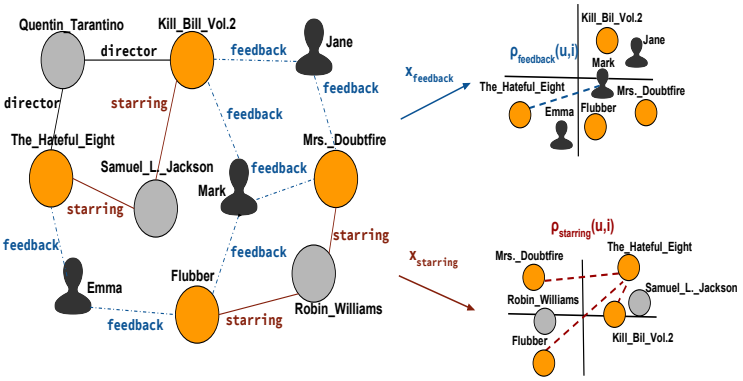

Figure 1: Starting from the knowledge graph K, which contains users (black), items (orange) and other entities (grey), for each property, we generate vector representations x_p with node2vec. The vector representations encode the structure of the graph. For p = 'feedback', we encode collaborative information, users and items are mapped close together in vector space according to their interactions and the relatedness $\rho_{feedback}(u, i)$ can be computed directly. For other properties, such as p = 'starring', we encode content information, items with similar starring actors are embedded close together. The relatedness $\rho_{starring}(u, i)$ has to be computed by averaging the distance from items i' liked by u in the past.

should have different weights in judging the relatedness between two entities. Films can be related in terms of starring actors and not in terms of subject, share the same director but not the same writer, and processing the whole knowledge graph altogether neglecting the semantics of the properties would not allow to account for this. Thus, we start by learning property-specific vector representation of nodes considering one property at the time. 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). Note that for properties where each entity is connected to more than one entity (e.g. 'starring' or 'feedback'), K_p can exhibit complex connectivity patterns. 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 [11]:

$$\max_{x_p} \sum_{e \in K_p} (-\log Z_e + \sum_{n_i \in N(e)} x_p(n_i) \cdot x_p(e)) \qquad (1)$$

where $Z_e = \sum_{v \in K_p} \exp(x_p(e) \cdot x_p(v))$ is the per-node partition function and is approximated using negative sampling [18], $N(e) \in K_p$ is the neighborhood of the entity e defined by the node2vec random walk and the optimization is carried out using stochastic gradient ascent over the parameters defining x_p. The optimization attempts to maximize the dot product between vectors of the same neighborhood, i.e. to embed them close together in vector space. Thus, from the vector representations $x_p(e)$, property-specific relatedness scores can be defined as follows:

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

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 s = cosine similarity). The features include both collaborative and content information and have a straight-forward interpretation (Fig. 1). 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. The hybrid approach is able to tackle the 'new item' issue, as for a new item with no feedback from users, we are still able to compute all the content-based features. For a new user u, the scores $\rho_p(u,i)$ can be computed by averaging the $\rho_p(u',i)$ over all the other users $u' \in U$.

3.3 Global user-item relatedness

For each user-item pair, we are now able to compute all property-specific relatedness scores $\vec{\rho}(u,i) = \{\rho_p(u,i)\}_{p \in \Gamma_\epsilon^+}$. We aim to use these scores as features of a global user-item relatedness model that can be used to provide item recommendation. To this end, 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 key idea is that of finding the parameters θ that optimize top-N item recommendation as a supervised learning to rank problem [16].

Training data: given a set of users $U = \{u_1, u_2..u_N\}$, each user u_k is associated with a set of items from feedback data $\vec{i_k} = \{i_{k1}, i_{k2}..i_{kn(k)}\}$ where $n(k)$ denotes the number of feedback released by the user u_k and a set of labels $\vec{y_k} = \{y_{k1}, y_{k2}..y_{kn(k)}\}$ 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 $T = \{(u_k, \vec{i_k}, \vec{y_k})_{k=1}^N\}$.

Ranking function: $\rho(u,i;\theta)$ is a ranking function, meaning that, for each user u_k, the corresponding items $\vec{i_k}$ are sorted according to its score. More formally, $\rho(u,i;\theta)$ 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.

Loss: the agreement $M(\pi(u_k, \vec{i_k}, \theta), \vec{y_k})$ 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 $P@N$, Mean Average Precision (MAP), NDCG [22]. From this score, a loss function can be easily derived as:

$$C(\theta) = \sum_{k=1}^{N} (1 - M(\pi(u_k, \vec{i_k}, \theta)) \quad (2)$$

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:

$$\hat{\theta} = \arg\min_{\theta} C(\theta) \quad (3)$$

The exact form of f, of the loss function C, the set of parameters θ to optimize and the approach used to optimize the loss function depend on the learning to rank algorithm and on the metrics that are optimized. In this work, we use Adarank [27] and LambdaMart [27] as ranking algorithms and we measure $P@N$ and MAP [22].

4 EXPERIMENTAL SETUP

Dataset: the dataset used for the evaluation of the proposed approach is MovieLens 1M[3]. MovieLens 1M [12] is a popular dataset for the evaluation of recommender systems and it contains 1,000,209 anonymous ratings of approximately 3,900 movies made by 6,040 MovieLens users. In a previous work [20], MovieLens 1M items have been mapped to the corresponding DBpedia entities. We leverage these publicly available mappings to make use of Linked Open Data in building the knowledge graph K. Since not every item in the Movielens data has a corresponding DBpedia entity, after this mapping we have 948978 ratings, from 6040 users on 3226 items.

Configuration: first, we define the set of properties p for which the features have to be computed. We start from the properties associated to the class 'Film' by the DBpedia ontology $\Gamma_{dbo:Film}$ and to this set we add 'dct:subject', which relates an entity to a set of possible categories, such as 'American Films'. For each of these properties, we obtain the subgraph K_p through SPARQL [23] queries. The properties for which we obtain a sufficient number of triples (> 15) are those reported in Tab. 3. The subgraph $K_{feedback}$ is derived from the positive ($r \geq 4$) ratings of the MovieLens 1M dataset. The second part of the configuration consists in defining the hyper-parameters of the model, i.e. internal parameters that are not explicitly set by the learning process. For what concerns node2vec, we optimize the parameters P, Q on validation data, experimenting the pairs $(P,Q) \in \{(1,1),(1,4),(4,1)\}$. P, also called *return parameter*, controls the likelihood of returning to a previously visited node. Q, also called *in-out parameter*, controls the probability of moving further away from the source node e. By playing with (P,Q), node2vec achieves a random walk strategy able to approximate a Breadth-First or a Depth-First search. For other parameters, we use the configuration obtained in a previous work, i.e. a walk length $l = 10$, a dimension of the embeddings vector $d = 500$, a number of walks per entity $n = 500$, a context size $k = 10$ and a number of epochs $\eta = 5$. Another hyper-parameter of the proposed approach is the learning to rank model. We experiment two listwise algorithms, AdaRank [27] and LambdaMart [4], on validation data. The implementation of these models is that provided by the software library RankLib[4].

Evaluation protocol: we adopt an evaluation protocol that is similar to RelPlusN [25]. We split the ratings in temporal order, using 80% of data to create a training set and 20% to create a probe set. In the training set, we consider ratings $r \geq 4$ as relevant items and the rest as not relevant. From the probe set, we sample a 4% of data to create a probe validation set and 16% to create a probe test set. Then, from each of these two datasets, items with rating $r = 5$ are selected and considered as relevant and for each user a number $N = 100$ of other items not rated by the user are randomly sampled as non-relevant examples, creating the validation and the test set. This

[3] https://grouplens.org/datasets/movielens/1m/
[4] https://sourceforge.net/p/lemur/wiki/RankLib/

P,Q	Ranking model	P@5	P@10	MAP
1,4	**LambdaMart**	**0.0791**	**0.0293**	**0.1717**
4,1	LambdaMart	0.0182	0.0193	0.0570
1,1	LambdaMart	0.0174	0.0188	0.0565
1,4	AdaRank	0.0134	0.0098	0.0278
4,1	AdaRank	0.0078	0.0083	0.0286
1,1	AdaRank	0.0109	0.0098	0.0358

Table 1: Results on the validation set of entity2rec for different combinations of hyper-parameters

Model	P@5	P@10	MAP
entity2rec	**0.2814**	**0.2127**	**0.4232**
MostPop	0.2154	0.1815	0.2907
NMF	0.1208	0.1150	0.1758
SVD	0.0543	0.0469	0.0888
ItemKNN	0.0463	0.0232	0.0990

Table 2: Results on the test set of entity2rec against the baselines

evaluation protocol allows to avoid the limitations of the TestItems protocol, in which only rated items are ranked [26] and that is thus very different from the context of a real recommender system. Note that the effect of this evaluation protocol is that of underestimating the quality of the recommendations, as the assumption that all unrated items are not relevant is pessimistic. The set of elements liked by a user $R_+(u)$ is defined as the items in the training set with ratings $r = 5$. We measure P@5, P@10 and MAP. As baselines, we use state-of-the-art collaborative filtering algorithms based on Non-Negative Matrix Factorization (NMF) [17], on Singular Value Decomposition [15], ItemKNN with baselines [14] and the Most Popular Items recommendation strategy, which is known to be quite effective on MovieLens due to the power-law distribution of user feedback data [6] (see also Tab. XIV in [19]). The baselines are implemented using the *surprise* python library[5].

5 RESULTS

Comparison with state-of-the-art: the hyper-parameter optimization on validation data shows that LambdaMart performs better than Adarank and that the pair $(P, Q) = (1, 4)$, which corresponds to a walk lingering in the neighborhood of the starting node, approximating Breadth-First-Search, works better than the other pairs. The best score, for every metric, is obtained by the pair $(P, Q) = (1, 4)$ and the model LambdaMart (Tab. 1[6]). Thus, we evaluate this configuration on the test set, comparing it to the NMF, SVD, ItemKNN and Most Popular Items baselines (Sec. 4), outperforming all of them for all the metrics under consideration (Tab. 2).

Feature selection: one of the strengths of the proposed approach is that of using features that have a clear interpretation. In this experiment, we try to evaluate the importance of the selected features $\rho_p(u, i)$ on the overall ranking quality. To this end, we select one feature at the time and we compute the ranking accuracy (Tab. 3), observing that the feature 'feedback' is by far the best performing one. However, including content-based features improves the overall performance of the system, as can be seen by comparing 'feedback' with 'entity2rec', which includes all the features. We expect this difference to be even more relevant on other datasets affected by a higher sparsity as observed in similar experiments with hybrid recommender systems [19, 28]. It is also worth noting that $\rho_{feedback}$ outperforms all the collaborative filtering baselines of Tab. 2, showing the effectiveness of the proposed approach in learning a measure of user-item relatedness even without content information.

Features	P@5	P@10	MAP
$\rho_{dbo:based\_on}$	0.0529	0.0247	0.0925
$\rho_{dbo:cinematography}$	0.0386	0.0290	0.0794
$\boldsymbol{\rho_{feedback}}$	**0.2317**	**0.1708**	**0.3550**
$\rho_{dbo:director}$	0.0219	0.0211	0.0949
$\rho_{dbo:editing}$	0.0294	0.0305	0.0724
$\rho_{dbo:music\_composer}$	0.0077	0.0040	0.0493
$\rho_{dbo:narrator}$	0.0572	0.0389	0.0834
$\rho_{dbo:producer}$	0.0119	0.0241	0.0498
$\rho_{dbo:starring}$	0.0128	0.0372	0.0728
$\rho_{dct:subject}$	0.0285	0.0326	0.0688
$\rho_{dbo:writer}$	0.0061	0.0216	0.0472
entity2rec	**0.2814**	**0.2127**	**0.4232**

Table 3: Performance of the proposed approach considering one feature at the time

6 CONCLUSIONS

In this work, we propose entity2rec, a new measure of user-item relatedness for top-N item recommendation. Starting from a knowledge graph, property-specific relatedness scores are obtained using a feature learning approach based on neural language models on property-specific subgraphs and then combined through a learning to rank approach to generate a global user-item relatedness measure optimizing top-N item recommendations. This two-stage approach has two major benefits: it allows to configure the system to take into account a specific property when providing recommendations (e.g. recommend movies with similar actors) and it allows to let the learning to rank algorithm implicitly weight the properties to provide recommendations. The results on the MovieLens 1M dataset show that the proposed approach outperforms commonly used collaborative filtering techniques based on matrix factorization, nearest neighbors and the Most Popular items strategy. From the data to the ranking, the feature engineering effort is minimal and the manual intervention is limited to setting and configuring hyper-parameters of the learning process. Future work will involve a more comprehensive evaluation, comparing the model to other hybrid state-of-the-art recommender systems on different domains and datasets, and in a general improvement of the feasibility of the proposed approach in a real recommendation scenario (such as computational time, online updates, context of the recommendations).

[5]http://surprise.readthedocs.io/en/v1.0.2/matrix_factorization.html

[6]Note that the scores obtained on the test set tend to be higher than those on the validation set because the ratio of relevant/non-relevant items is higher in the latter.

REFERENCES

[1] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. 2007. Dbpedia: A nucleus for a web of open data. In *The semantic web*. Springer, 722–735.

[2] Yoshua Bengio, Aaron Courville, and Pascal Vincent. 2013. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence* 35, 8 (2013), 1798–1828.

[3] Christian Bizer, Tom Heath, and Tim Berners-Lee. 2009. Linked data-the story so far. *Semantic Services, Interoperability and Web Applications: Emerging Concepts* (2009), 205–227.

[4] Christopher JC Burges. 2010. From ranknet to lambdarank to lambdamart: An overview. *Learning* 11, 23-581 (2010), 81.

[5] Rose Catherine and William Cohen. 2016. Personalized Recommendations using Knowledge Graphs: A Probabilistic Logic Programming Approach. In *Proceedings of the 10th ACM Conference on Recommender Systems*. ACM, 325–332.

[6] Paolo Cremonesi, Yehuda Koren, and Roberto Turrin. 2010. Performance of recommender algorithms on top-n recommendation tasks. In *Proceedings of the fourth ACM conference on Recommender systems*. ACM, 39–46.

[7] Tommaso Di Noia. 2016. Recommender Systems Meet Linked Open Data. In *International Conference on Web Engineering*. Springer, 620–623.

[8] Tommaso Di Noia, Roberto Mirizzi, Vito Claudio Ostuni, Davide Romito, and Markus Zanker. 2012. Linked open data to support content-based recommender systems. In *Proceedings of the 8th International Conference on Semantic Systems*. ACM, 1–8.

[9] Cristhian Figueroa, Iacopo Vagliano, Oscar Rodríguez Rocha, and Maurizio Morisio. 2015. A systematic literature review of Linked Data-based recommender systems. *Concurrency and Computation: Practice and Experience* 27, 17 (2015), 4659–4684.

[10] Jerome H Friedman. 2001. Greedy function approximation: a gradient boosting machine. *Annals of statistics* (2001), 1189–1232.

[11] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 855–864.

[12] F Maxwell Harper and Joseph A Konstan. 2016. The movielens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems (TiiS)* 5, 4 (2016), 19.

[13] Houda Khrouf and Raphaël Troncy. 2013. Hybrid event recommendation using linked data and user diversity. In *Proceedings of the 7th ACM conference on Recommender systems*. ACM, 185–192.

[14] Yehuda Koren. 2010. Factor in the neighbors: Scalable and accurate collaborative filtering. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 4, 1 (2010), 1.

[15] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 42, 8 (2009).

[16] Tie-Yan Liu and others. 2009. Learning to rank for information retrieval. *Foundations and Trends® in Information Retrieval* 3, 3 (2009), 225–331.

[17] Xin Luo, Mengchu Zhou, Yunni Xia, and Qingsheng Zhu. 2014. An efficient non-negative matrix-factorization-based approach to collaborative filtering for recommender systems. *IEEE Transactions on Industrial Informatics* 10, 2 (2014), 1273–1284.

[18] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*. 3111–3119.

[19] Tommaso Di Noia, Vito Claudio Ostuni, Paolo Tomeo, and Eugenio Di Sciascio. 2016. Sprank: Semantic path-based ranking for top-n recommendations using linked open data. *ACM Transactions on Intelligent Systems and Technology (TIST)* 8, 1 (2016), 9.

[20] Vito Claudio Ostuni, Tommaso Di Noia, Eugenio Di Sciascio, and Roberto Mirizzi. 2013. Top-n recommendations from implicit feedback leveraging linked open data. In *Proceedings of the 7th ACM conference on Recommender systems*. ACM, 85–92.

[21] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 701–710.

[22] David Martin Powers. 2011. Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation. *Journal of Machine Learning Technologies* (2011).

[23] Bastian Quilitz and Ulf Leser. 2008. Querying distributed RDF data sources with SPARQL. In *European Semantic Web Conference*. Springer, 524–538.

[24] Jessica Rosati, Petar Ristoski, Tommaso Di Noia, Renato de Leone, and Heiko Paulheim. 2016. RDF graph embeddings for content-based recommender systems. In *CEUR workshop proceedings*, Vol. 1673. RWTH, 23–30.

[25] Alan Said and Alejandro Bellogín. 2014. Comparative recommender system evaluation: benchmarking recommendation frameworks. In *Proceedings of the 8th ACM Conference on Recommender systems*. ACM, 129–136.

[26] Harald Steck. 2013. Evaluation of recommendations: rating-prediction and ranking. In *Proceedings of the 7th ACM conference on Recommender systems*. ACM, 213–220.

[27] Jun Xu and Hang Li. 2007. Adarank: a boosting algorithm for information retrieval. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 391–398.

[28] Xiao Yu, Xiang Ren, Yizhou Sun, Quanquan Gu, Bradley Sturt, Urvashi Khandelwal, Brandon Norick, and Jiawei Han. 2014. Personalized entity recommendation: A heterogeneous information network approach. In *Proceedings of the 7th ACM international conference on Web search and data mining*. ACM, 283–292.