SPRank: Semantic Path-based Ranking for Top-N Recommendations using Linked Open Data

SPRank: Semantic Path-based Ranking for Top-N Recommendations using Linked Open Data

Tommaso Di Noia, Polytechnic University of Bari
Vito Claudio Ostuni, Polytechnic University of Bari
Paolo Tomeo, Polytechnic University of Bari
Eugenio Di Sciascio, Polytechnic University of Bari

In most real world scenarios the ultimate goal of recommender system (RS) applications is to suggest a short ranked list of items, namely *top-N* recommendations, supposed to be the most appealing for the end user. Often, the problem of computing *top-N* recommendations is mainly tackled with a two steps approach. The system focuses first on predicting the unknown ratings which are eventually used to generated a ranked recommendation list. Actually, the *top-N* recommendation task can be directly seen as a ranking problem where the main goal is not to accurately predict ratings but directly find the best ranked list of items to recommend. In this paper, we present `SPrank`, a novel hybrid recommendation algorithm able to compute *top-N* recommendations exploiting freely available knowledge in the Web of Data. In particular we employ `DBpedia`, a well-known encyclopedic knowledge base in the `Linked Open Data` cloud, to extract semantic *path-based* features and to eventually compute *top-N* recommendations in a *learning to rank* fashion. Experiments with three datasets related to different domains (books, music and movies) prove the effectiveness of our approach compared to state-of-the-art recommendation algorithms.

Categories and Subject Descriptors: H.3.3 [**Information Systems**]: Information Search and Retrieval

General Terms: Recommender Systems, Linked Open Data

Additional Key Words and Phrases: Learning to rank, DBpedia, Hybrid recommender systems

1. INTRODUCTION

Information overload in the current Web challenges users in their decision-making tasks. In such scenario, recommender systems [Ricci et al. 2011] have become essential tools in assisting users to find, in a personalized manner, what is relevant for them in overflowing complex information spaces.
Broadly speaking, recommendation algorithms can be categorized in two main classes: collaborative filtering and content-based. Collaborative filtering techniques recommend items based on liked-mind users' preferences while content-based techniques recommend items sharing similar features to those a user has preferred in past. Hybrid methods [Burke 2002] combine both approaches to achieve better recommendation quality.

An important aspect to consider when building content-based/hybrid systems is that the accuracy of their recommendations is heavily influenced by the quality of content data describing the items. Most approaches describe items in terms of textual features or plain attributes [Lops et al. 2011]. The main drawback of such item representation is that it completely ignores the semantics underlying the item space.

To overcome such limitation, several works on *ontological* and *semantic-aware* recommender systems have been proposed in the past. Most of them exploit item's ontological knowledge to boost collaborative filtering systems [Anand et al. 2007; Cantador et al. 2008; Mobasher et al. 2004] or to build better content-based ones [Semeraro et al. 2009]. Such approaches have been shown to be particularly effective in solving some drawbacks of pure collaborative methods such as cold start and data sparsity.

Authors' addresses: Via Re David 200, Department of Electrical and Information Engineering, Polytechnic University of Bari, 70125 Bari, Italy, email: {tommaso.dinoia,vitoclaudio.ostuni,paolo.tomeo,eugenio.disciascio}@poliba.it.

DOI 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

In the last few years we have assisted to an impressive publication of freely available ontological data on the Web. The *Linking Open Data* project [Bizer et al. 2009a], started as a community effort in 2007, helped to produce billions of RDF[1] (**R**esource **D**escription **F**ramework) statements that are now published on the Web. The result of this initiative is a huge decentralized knowledge base, commonly known as the Linked Open Data (LOD) cloud, wherein each "piece of little knowledge" is enriched by links to related data. The development of a global information space consisting not only of linked documents but also of linked data has resulted in the emergence of the Web of Data as a subset of the World Wide Web. With the increased availability of this freely available knowledge, there is a great interest in taking advantage of such information to boost conventional recommender systems. Most previous work on ontological RSs based on limited domain ontologies and taxonomies. Another important limitation regarding past work on ontological RSs, as well as most of the research on recommender systems, is that they address the rating prediction task where the system goal is to accurately predict all the missing ratings - usually by minimizing the root mean squared error between the predicted an actual ratings - and eventually produce a ranked list based on predicted ratings. Nevertheless, more recently ranking-based recommendation approaches, which directly model the recommendation problem as a ranking problem, have gained increasing interest [Rendle et al. 2009; Shi et al. 2012; Weimer et al. 2007; Liu and Yang 2008; Balakrishnan and Chopra 2012; Volkovs and Zemel 2012] because they seem to better approximate the *top-N* recommendation task [Cremonesi et al. 2010] which focuses on finding a few specific items (*top-N* list) supposed to be the most appealing to the user.

In this paper we present SPrank (***S**emantic **P**ath-based **rank**ing*), a novel hybrid recommendation algorithm able to compute *top-N* item recommendations in a learning to rank setting and combining ontological knowledge belonging to the Web of Data with collaborative user preferences in a unified graph-based data model. While several works have been proposed in the last recent years to address the *top-N* recommendation task as a ranking problem in the literature of collaborative filtering, for ontological and hybrid recommender systems these issues have not been fully investigated yet.
In our evaluation of SPrank, the ontological knowledge describing the items has been extracted from DBpedia[2], a well-known encyclopedic knowledge base belonging to the LOD cloud. From the analysis of the DBpedia semantic graph we extract *path-based* features for describing the interactions between users and items and use *learning to rank* for computing the *top-N* recommendations as a ranking problem.

We want to stress that previous work on semantic-aware and ontological recommender systems do not focus on the ranking nature of the *top-N* recommendation problem. Here we show how SPrank is able to compute more accurate recommendations than other competitive collaborative filtering approaches leveraging the semantic-enabled connections between users and items in a knowledge graph. To the best of our knowledge, this is the first work proposed to address the *top-N* recommendation task as a ranking problem by leveraging the Web Of Data. Main contributions of this paper are:

— formulation of a hybrid recommendation problem in a learning to rank setting for *top-N* recommendations;
— combined usage of semantic item descriptions from the Web of Data and user feedbacks in a unified graph-based feature space;

[1] http://www.w3.org/TR/rdf-concepts/
[2] http://dbpedia.org

— mining of the knowledge graph through path-based features to capture complex and meaningful relationships between items;
— evaluation of the proposed approach in terms of recommendation ranking accuracy on various datasets belonging to three different domains: movie, music and book.

This paper substantially extends our previous work [Ostuni et al. 2013]. First in this paper we present a more generic approach which can deal with both explicit and implicit feedback while our previous work focuses only on the implicit feedback case. Second and more important we discuss and experimentally compare various learning to rank algorithms, while in our previous work we focus exclusively on pointwise methods. Third we give a more detailed and comprehensive description of the framework with related examples and case studies related to three different domains. In addition we perform a more systematic and extensive evaluation on three different datasets together with an analysis about the most significant paths in each of them. We also discuss more in detail the advantage of using Linked Open Data for recommender systems.

The remainder of this work is structured as follows. In Section 2 we discuss related work. We present our approach for *top-N* recommendation in Section 3. The experimental evaluation is carried out in Section 4. Conclusion and future work close the paper. In order to make the paper self-consistent, in Appendix A we briefly recap some notions on RDF and SPARQL and provide a brief description of DBpedia.

2. RELATED WORK

In the following we review previous work on ontological recommendation methods and more recent literature on recommender systems based on LOD and heterogeneous networks. In addition, since we propose a ranking based hybrid recommendation approach we present related work in the area of ranking oriented collaborative filtering methods.

2.1. Ontological and LOD-based recommender systems

In [Middleton et al. 2009] an ontological recommender system is presented that makes use of semantic user profiles to compute collaborative recommendations with the effect of mitigating cold-start and improving overall recommendation accuracy. In [Mobasher et al. 2004] the authors present a *semantically enhanced collaborative filtering* approach, where structured semantic knowledge about items is used in conjunction with user-item ratings to create a combined similarity measure for item comparisons. In [Ziegler et al. 2004] taxonomic information is used to represents the user's interest in categories of products. Consequently, user similarity is determined by common interests in categories and not by common interests in items. In [Anand et al. 2007] the authors present an approach that infers user preferences from rating data using an item ontology. The system collaboratively generates recommendations using the ontology and infers preferences during similarity computation. Another hybrid ontological recommendation system is proposed in [Cantador et al. 2008] where user preferences and item features are described by semantic concepts to obtain users' clusters corresponding to implicit *Communities of Interest*. In all of these works, the experiments prove an accuracy improvement over traditional memory-based collaborative approaches especially in presence of sparse datasets.

Most of the works described so far have been produced before the LOD initiative was officially launched. In the last few years with the availability of such rich data a new class of recommender systems has emerged which can be named as LOD-based recommender systems. This new typology of recommendation methods is attracting increasingly interest in both the communities of Semantic Web and Recommender Systems as proved by the recent Linked Open Data-enabled Recommender Systems challenge

on Book Recommendation [Di Noia et al. 2014].
One of the first approaches that exploits Linked Open Data for building recommender systems is [Heitmann and Hayes 2010]. Here the authors, for the first time propose a recommendere system fed by Linked Open Data and show some experimental results in terms of precision and recall of the obtained results. In [Fernández-Tobías et al. 2011] the authors present a knowledge-based framework leveraging DBpedia for computing cross-domain recommendations. In [Di Noia et al. 2012a; Di Noia et al. 2012b] a model-based approach and a memory-based one to compute content-based recommendations are presented leveraging LOD datasets. Another LOD content-based method is presented in [Ostuni et al. 2014] which defines a neighborhood-based graph kernel for matching graph-based item representations. In [Ostuni et al. 2013] the authors present a mobile application based on DBpedia for context-aware movie recommendations.

Two hybrid approaches have been presented: in [Ostuni et al. 2013] it is shown how to compute top-N recommendations from implicit feedback using linked data sources and in [Khrouf and Troncy 2013] the authors propose an event recommendation system based on linked data and user diversity. In [Rowe 2014a] the authors propose a semantic-aware extension of the SVD++ model, named SemanticSVD++, which incorporates semantic categories of items into the model. The model is able also to consider the evolution over time of user's preferences. In [Rowe 2014b] the authors improve their previous work for dealing with cold-start items by introducing a vertex kernel for getting knowledge about the unrated semantic categories starting from those categories which are known. Finally another interesting direction about the usage of LOD for content-based RSs is explored in [Musto et al. 2014] where the authors present Contextual eVSM, a content-based context-aware recommendation framework that adopts a semantic representation based on distributional models and entity linking techniques. In particular entity linking is used to detect entities in free text and map them to LOD.

2.2. Recommender systems based on heterogeneous networks

Very recently, contextually to the progression of LOD-based recommender systems which rely on RDF graph data, recommendation methods based on generic heterogeneous networks have emerged. In [Yu et al. 2014] a network based entity recommendation method is presented which uses *meta-path* based latent features to represent the connectivity between users and items along different types of paths. Considering users and item latent features along different *meta-path* a global and local matrix factorization model are learnt by using the BPR (**B**ayesian **P**ersonalized **R**anking) [Rendle et al. 2009] approach. Particularly, as defined in [Sun et al. 2012] a *meta-path* is a sequence of edge types. In [Lao and Cohen 2010] the authors present a proximity measure defined as weighted combination of path types, called *path experts*. Such weighted combination is obtained by fitting a logistic regression model. The authors say that when considering only one parameter per edge label the proximity measure is limited because the context in which an edge label appears is ignored. Experiments on several tasks show improvements with respect to a classic Random Walk with Restart approach. They successfully apply such approach also for performing learning and inference in large and imperfect knowledge bases [Lao et al. 2011]. Our approach, SPrank, is similar to these two related work in the way that it bases on the definition of *path-based* features for mining the user-item interactions. However, in our work we exploit LOD data sources for building the knowledge graph exploited for computing the user item path-based features and formalize the recommendation problem in a learning to rank setting which allows very easily to plug in any learning to rank algorithm.

Both LOD-based recommendation approaches and those based on heterogeneous networks share the same underlying data model, that is the multi relational directed graph. The main distinction characterizing LOD knowledge bases with respect plain heterogeneous networks is in the ontological schema associated to the data that is represented in RDFS and OWL and relative inference mechanisms.

2.3. Ranking oriented recommender systems

Referring to the *top-N* recommendation problem in the context of collaborative filtering, several works have been proposed in the last few years. *SLIM* [Ning and Karypis 2011] adopts a *Sparse Linear* method for learning a sparse aggregation coefficient matrix that is used for computing *top-N* recommendations. In [Ning and Karypis 2012] the authors propose an extension of SLIM to incorporate both users and side information about items thus showing an improvement of the performance associated to the usage of such information. In [Weimer et al. 2007] the authors arguing that nDCG (**n**ormalized **D**iscounted **C**umulative **G**ain) is a better training/evaluation metric for *top-N* recommendations, present *CofiRank*. It is a maximum margin matrix factorization (MF) model to minimize a nDCG@N loss function. In [Balakrishnan and Chopra 2012] the authors propose novel models based on matrix factorization which approximately optimize nDCG for *top-N* recommendation. In [Shi et al. 2012] the authors present *CLiMF*, a collaborative-filtering algorithm able to directly optimize the Mean Reciprocal Rank. The authors of [Rendle et al. 2009] introduce the BPR criterion for optimizing a ranking loss. A hybrid extension of BPR is presented in [Gantner et al. 2010] that learns a linear mapping on the user/item features from the factorization and auxiliary user/item-attribute matrix. This extension of BPR is able to compute useful recommendations in cold-start scenarios.

3. SPRANK: SEMANTIC PATH-BASED RANKING

The *Linking Open Data* community project started in 2007 with the goal of extending the current Web with data published according to Semantic Web standards (see Appendix A for a brief description of the RDF and SPARQL). The idea is to use RDF[3] to broadcast various open datasets on the Web, as a vast decentralized knowledge graph. The data model behind RDF is a labeled directed graph where nodes correspond to entities and labeled edges are properties connecting them. The semantics of such properties is explicitly modeled by means of an ontological schema represented in RDFS[4] or OWL[5] (**W**eb **O**ntology **L**anguage).

LOD datasets are natural candidates for feeding content-based and hybrid recommender systems. Depending on the dataset, there is the availability of multi-relational data related to different domains. We can get data about geographic locations, music, movies, art, people, facts, and general common-sense knowledge. If we consider encyclopedic datasets such as DBpedia [Lehmann et al. 2014] or Freebase [Bollacker et al. 2008], we have access to a huge amount of factual knowledge referring to a variety of topics.

For example, we can obtain information about which actors starred in the movie *Pulp Fiction* via the following SPARQL[6] query:

```
PREFIX dbpedia: <http://dbpedia.org/resource/>
PREFIX dbpedia-owl: <http://dbpedia.org/ontology/>
SELECT ?actor WHERE {
```

[3] http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/
[4] http://www.w3.org/TR/rdf-schema/
[5] http://www.w3.org/TR/owl2-overview/
[6] See Appendix A for a very quick overview of SPARQL.

```
   dbpedia:Pulp_Fiction dbpedia-owl:starring ?actor .
}
```

Starting from the previous query we see how to extract rich data related to a specific resource/item as well as to a bunch of them. All the extracted data are an ideal candidate to feed a content-based or hybrid recommender system. As an example, Figure 1 shows a portion of the DBpedia graph related to three different recommendation scenarios. Figure 1(a) refers to the musical artist domain. In this example Adele, Bruno Mars and U2 represent items, while the other nodes such as Pop Music, 19, Bass Guitar, etc. are generic entities belonging to different types such as genre, album, instrument. We note how nodes are connected to other nodes by means of typed links. For example Adele is connected to Pop Music by means of the genre relation[7]. Figure 1(b) shows an example related to the movie domain. Here nodes represent movies, actors, directors, categories, genres. Also in this domain there are different types of relations such as starring, subject, director, etc. Finally, in Figure 1(c) a DBpedia fragment related to the book domain is depicted. In this case, items are books as The Da Vinci Code, The Shining and Angels and Daemons while connected entities such as American mystery writers, Thriller or Edgar Allan Poe describe the items via different relations such as category, genre, writer. Also in this case there are several relations with different semantics. For example the property notableWork indicates that the book is a remarkable writing of the writer.

The main advantages of using LOD for recommender systems can be summarized as:

— availability of a great amount of multi-domain and ontological knowledge freely available to feed the system;
— Semantic Web standards and technologies to retrieve the required data and hence no need for content analysis tasks [Lops et al. 2011] to obtain a structured representation of the items;
— the ontological and relational nature of the data allows the system to analyze item descriptions at a semantic level.

In Figure 2 the main steps to build a system adopting SPrank are depicted. We extract data from the LOD cloud (step 1) and then we build our own knowledge graph (step 2). This latter is augmented with user data (ratings) and is then used to extract relevant paths (step 3) needed to train the SPrank model. Optionally, if the data are not natively coming from the Linked Open Data cloud, a preliminary linking step might be needed.

The main idea behind SPrank is exploring *paths* in a *semantic* graph, such as DBpedia, in order to find items that are related to the ones the user is interested in. Specifically, we define *path-based* features to represent the connectivity between users and items exploiting the multi-relational structure of the graph. Then, on top of such features we apply a learning to rank algorithm for getting a ranking function able to recommend the most relevant items to the user.
In the following we start by detailing the data model and by formulating the recommendation problem. Then we describe the path-based feature and the algorithms for learning the ranking function.

3.1. Data Model

RDF graphs represent the underlying data we use to encode the knowledge about the domain we want to develop our recommendation engine on. Figure 1 shows possible

[7]For the sake of presentation we will refer to RDF properties and resources without their prefix (see Appendix A), i.e., we will write genre instead of dbpedia-owl:genre or broader instead of skos:broader and so on.

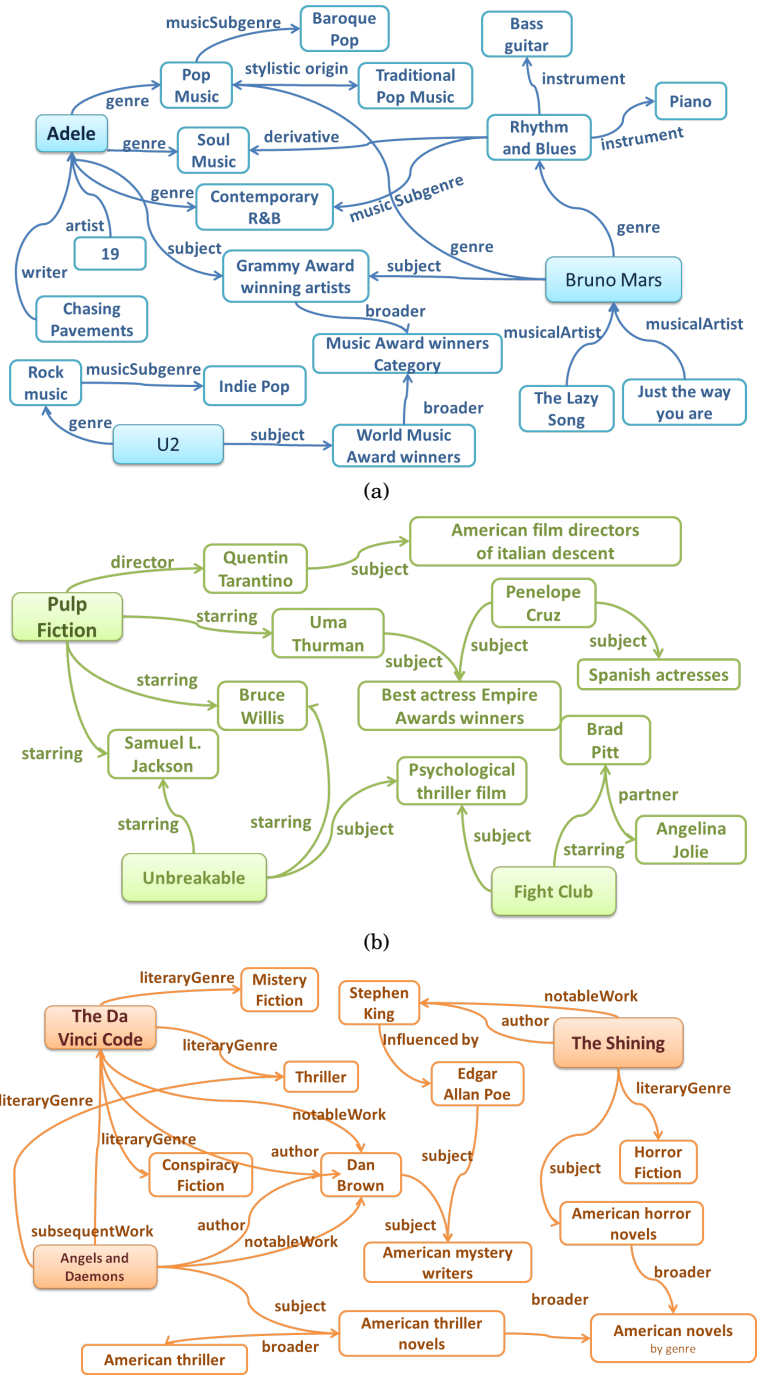

Fig. 1. Examples of DBpedia fragments related to the music (a), movie (b) and book (c) domains.

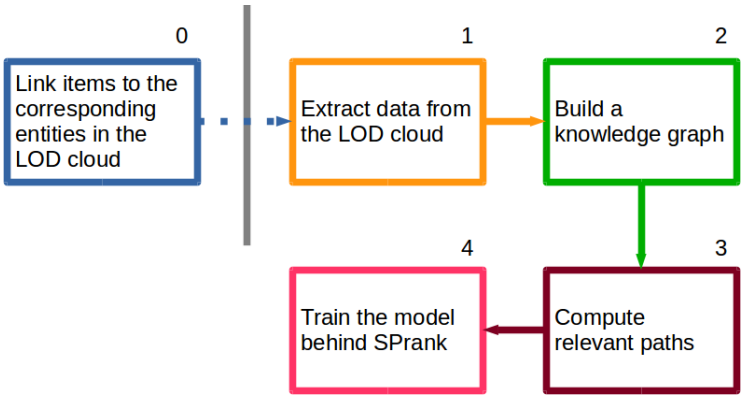

Fig. 2. A schematic representation of the SPrank approach.

examples relative to three different domains (music, movies and books). Such data can be already available as a complete LOD dataset or can be extracted by posing a federated query to different SPARQL endpoints and combining their results in a unique knowledge graph (see Section A.2).

The data model behind the collaborative filtering problem can be represented in a graph data model as well. Let us define the user-item rating matrix $\hat{R}$, where each entry $\hat{r}_{ui}$ represents the rating of user u on item i. We can use $\hat{R}$ for building a bipartite graph where users and items are the nodes and users' feedback are the links. Particularly, we can define a labelling function $\psi_R : \mathbb{R} \rightarrow R$ for mapping the ratings to symbolic user-item relations expressing the item relevance for the user. In this way we are able to extend the original knowledge graph with collaborative information thus representing all the information in an integrated and uniform way as shown in Fig. 3. Consequently the multi-relational graph is defined as $G = \{t \mid t \in V \times \Sigma \times V\}$ where V denotes the set of vertices and Σ indicates the set of edge labels or relations. Due to the nature of the problem we identify three relevant subsets of V: U, I and E representing *users*, *items* and *entities*, respectively. Moreover, in our model the two following relations hold: $V = U \cup E$ and $I \subseteq E$ since we consider items as a particular type of entities. Similarly, Σ contains two categories of relationships. In fact, we have $\Sigma = R \cup P$ where $R = \{r \mid \exists\, (u, r, i) \in G \wedge u \in U \wedge i \in I\}$ and $P = \{p \mid \exists\, (e_j, p, e_k) \in G \wedge e_j, e_k \in E\}$. In other words, a relation $r \in R$ links a user $u \in U$ to an item $i \in I$ while a relation $p \in P$ connects either an item i to another entity $e \in E$ in the graph or an entity $e_j \in E \setminus I$ to another entity $e_k \in E$. In the rest of the paper we will use u, i, e and v to represent a generic node belonging to U, I, E and V, respectively. Analogously, we will denote with σ, p and r generic relations in Σ, P and R. For each $(v_j, \sigma, v_k) \in G$ we assume there always exists (v_k, σ^{-1}, v_j) where σ^{-1} represents the inverse relation of σ. When no confusion arises we write $\sigma = (v_j, v_k)$ as an alternative representation of (v_j, σ, v_k). Figure 3 shows an example of G where we have $U = \{\texttt{Alice}, \texttt{John}, \texttt{Tom}\}$ and $I = \{\texttt{AngelsAndDaemons}, \texttt{TheDaVinciCode}, \texttt{TheShining}\}$. In this example we have two types of relations for modelling the user ratings, $R = \{\texttt{like}, \texttt{dislike}\}$. All the other relations belong to $P = \{\texttt{literaryGenre}, \texttt{subject}, \texttt{author}, \texttt{notableWork}, \texttt{influencedBy}, \texttt{subsequentWork}, \texttt{broader}\}$.

Thanks to this graph-based setting we can consider both collaborative and content aspects in a uniform setting thus leveraging the multi-relational nature of the graph. This will allow us to represent both collaborative and content inputs in a unified feature space.

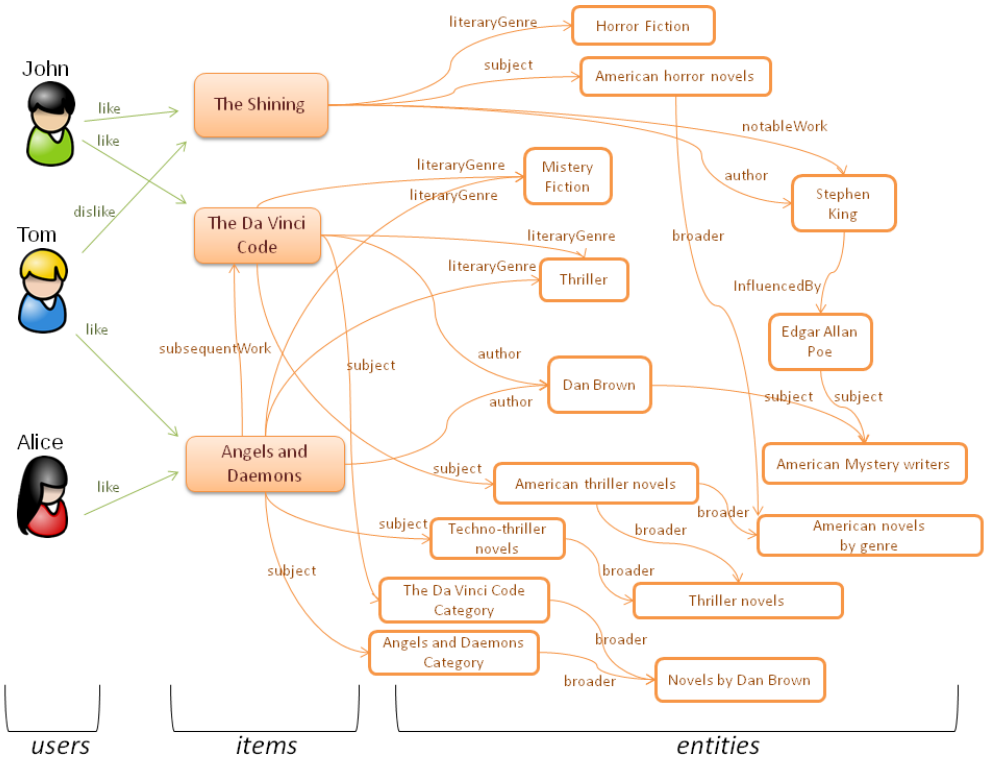

Fig. 3. Example of the graph data model G.

Hereafter, without loss of generality, we refer to the single dataset of DBpedia. Nevertheless, the approach and the algorithms we present in this paper can be used also with any semantic graph built on top of data coming from different datasets.

3.2. Problem Formulation

Given a graph G, our aim is to recommend the most relevant items to each user leveraging the knowledge encoded in the graph. Differently from traditional hybrid/content-based approaches, we directly formulate the problem of computing *top-N* recommendations in the standard learning to rank setting adopted in Web search [Liu 2009] by replacing queries with users and document with items and using user's ratings as relevance scores. In this way we can leverage on well established techniques for learning the ranking function.

For each user-item pair $(u, i) \in U \times I$, we encode the matching between user interests and item content in the feature vector $\mathbf{x_{ui}} \in \mathbb{R}^D$ where D is the dimension of the feature space. Each component in $\mathbf{x_{ui}}$ represents the relevance of i for u with respect to a specific feature[8].

For each user $u \in U$ we define user profile $I_u = \{i \in I | \hat{r}_{ui} \in \hat{R}\}$ the set of items rated by u. The ratings $\hat{r}_{ui}$ associated to the items in I_u can be used to induce a ranking

[8]In SPrank we rely on *path-based* features as detailed in Section 3.3.

among them. Defining the training set TS and recommendation set RS as follows:

$$TS = \bigcup_u \{\langle u, i, \mathbf{x_{ui}}, \hat{r}_{ui}\rangle | i \in I_u\}$$

$$RS = \bigcup_u \{\langle u, i, \mathbf{x_{ui}}, \hat{r}^*_{ui}\rangle | i \in (I \setminus I_u)\}$$

The goal is then to learn a scoring function $f : U \times I \to \mathbb{R}$ from the training data TS able to replicate for each user its perfect ranking. Once we have $f(u, i)$ we apply it to the recommendation set RS for composing the *top-N* recommendation list by means of the computed scores $\hat{r}^*_{ui}$.
As we will see in Section 3.4, depending on the learning to rank approach there are different ways of learning $f(u, i)$.

Following similar approaches such as item-based similarity, LDA, etc., here we learn a single model over all users. In the former approach feature vectors are representative only of the item content and the user profiling is inherently represented in the predictive model. Instead in our approach the matching between user's and item's characteristics is explicitly represented using the joint user-item feature vector $\mathbf{x_{ui}}$.

3.2.1. Learning with Implicit Feedback Data. The formulation of the learning problem given above relies on the availability of graded or binary relevance values commonly obtained from explicit feedback data. However in many real scenarios explicit user feedback are difficult to obtain. Nevertheless, it is still possible to obtain implicit feedback data by analysing users interactions with the system. Examples of such interactions can be clicks, purchases, video watching, etc. The main problem with implicit feedback is that they reflect only positive user preferences. If a user buys an item it is reasonable to assume that the user likes it. On the contrary the system cannot infer anything about what the user dislikes. The unobserved data are a mixture of actually negative and missing values [Rendle et al. 2009], but the system does not have any information for discriminating between them.
Then, learning the scoring function from such unary data becomes infeasible because all items have same relevance. To overcome this issue for each user we select a portion of unobserved items $I^*_u \subset I \setminus I_u$ to be used as negative data points in the training set which becomes:

$$TS = \bigcup_u \{\langle u, i, \mathbf{x_{ui}}, \hat{r}_{ui}\rangle | i \in (I_u \cup I^*_u)\}$$

where $\hat{r}_{ui} = 0$ with $i \in I^*_u$.

3.3. Path-based features

The main objective of the proposed framework is to recommend items exploiting their underlying properties expressed in the semantic graph. In order to achieve this, `SPrank` extracts features able to characterize the interactions between users, items and entities capturing the complex *paths* between them.

Thanks to the multi-relational nature of the data there are several types of paths. Each particular path has its own semantics and it might have a different relevance for the end user. For example a user might be interested in a movie because of few specific actors and in such case paths involving the `starring` relation would be more discriminative in finding good movies. Another user might not be really interested in few specific actors but rather in those actors belonging to a specific category. In this case considering a sequence of relations such as `(starring, subject)` would be better. Moreover, some paths that involve useless properties can be noisy for the purpose of

Table I. Example of Path Index extracted from the graph in Figure 3.

Path Index	
1:	$(\texttt{like}, \texttt{subsequentWork}^{-1})$
2:	$(\texttt{like}, \texttt{dislike}^{-1}, \texttt{like})$
3:	$(\texttt{like}, \texttt{like}^{-1}, \texttt{dislike})$
4:	$(\texttt{dislike}, \texttt{like}^{-1}, \texttt{like})$
5:	$(\texttt{dislike}, \texttt{like}^{-1}, \texttt{like}, \texttt{subsequentWork}^{-1})$
6:	$(\texttt{like}, \texttt{dislike}^{-1}, \texttt{like}, \texttt{subsequentWork})$
7:	$(\texttt{like}, \texttt{like}^{-1}, \texttt{dislike}, \texttt{like}^{-1}, \texttt{like})$
8:	$(\texttt{like}, \texttt{literaryGenre}, \texttt{literaryGenre}^{-1})$
9:	$(\texttt{like}, \texttt{author}, \texttt{author}^{-1})$
10:	$(\texttt{like}, \texttt{subject}, \texttt{broader}, \texttt{broader}^{-1}, \texttt{subject}^{-1})$
11:	$(\texttt{dislike}, \texttt{subject}, \texttt{broader}, \texttt{broader}^{-1}, \texttt{subject}^{-1})$
12:	$(\texttt{like}, \texttt{notableWork}, \texttt{InfluencedBy}, \texttt{subject}, \texttt{subject}^{-1}, \texttt{author})$
13:	$(\texttt{like}, \texttt{author}, \texttt{InfluencedBy}, \texttt{subject}, \texttt{subject}^{-1}, \texttt{author})$
14:	$(\texttt{like}, \texttt{author}, \texttt{subject}, \texttt{subject}^{-1}, \texttt{InfluencedBy}^{-1}, \texttt{author}^{-1})$
15:	$(\texttt{like}, \texttt{author}, \texttt{subject}, \texttt{subject}^{-1}, \texttt{InfluencedBy}^{-1}, \texttt{notableWork}^{-1})$
16:	$(\texttt{dislike}, \texttt{notableWork}, \texttt{InfluencedBy}, \texttt{subject}, \texttt{subject}^{-1}, \texttt{author})$
17:	$(\texttt{dislike}, \texttt{author}, \texttt{InfluencedBy}, \texttt{subject}, \texttt{subject}^{-1}, \texttt{author})$
18:	$(\texttt{like}, \texttt{subsequentWork}^{-1}, \texttt{subject}, \texttt{broader}, \texttt{broader}^{-1}, \texttt{subject}^{-1})$
19:	$(\texttt{like}, \texttt{subject}, \texttt{broader}, \texttt{broader}^{-1}, \texttt{subject}^{-1}, \texttt{subsequentWork})$
20:	$(\texttt{like}, \texttt{subsequentWork}, \texttt{like}^{-1}, \texttt{like})$

recommendation.
Based on these assumptions, we extract rich features based on sequence of relations for building the feature vector $\mathbf{x_{ui}}$ and delegate to the learning to rank algorithm the task of discerning what paths are the most relevant.

Given the multi-relational graph G as defined in Section 3.1, we define path the sequence of relations of the form $(r_1 \dots \sigma_l \dots \sigma_L)$ such that $r_1 = (u, v_1)$, $\sigma_l = (v_{l-1}, v_l)$ and $\sigma_L = (v_{L-1}, i)$ with $u \in U$, $i \in I$. We refer to the actual sequence of nodes and relations $u \xrightarrow{r_1} v_1 \dots \xrightarrow{\sigma_l} \dots v_{L-1} \xrightarrow{\sigma_L} i$ which generates the particular path as *path instance*. We also define the *length of a path* as the number of relations contained within such path. We only consider paths having length greater than 1 and less or equal than a given L. We collect all the possible paths in G to build an index we call $Path$. We refer to each path in the index as $Path(j)$ to denote the j-th entry. Referring to the graph shown in Figure 3 and setting $L = 6$ we obtain the paths listed in Table I. This table represents the path index $Path$ just introduced. For example with the notation $Path(2)$ we refer to $(\texttt{like}, \texttt{dislike}^{-1}, \texttt{like})$. An example of path instance for $Path(2)$ is $\texttt{John} \xrightarrow{\texttt{like}} \texttt{TheShining} \xrightarrow{\texttt{dislike}^{-1}} \texttt{Tom} \xrightarrow{\texttt{like}} \texttt{AngelsAndDaemon}$.

Considering a user-item pair (u, i), we denote with $\#path_{u,i}(j)$ the number of path instances between u and i corresponding to the specific $Path(j)$ entry in the index. In other words, it represents how many paths of type $Path(j)$ connect u and i. Then, we define the j-th component in the feature vector $\mathbf{x_{ui}}$ as:

$$\mathbf{x_{ui}}(j) = \frac{\#path_{u,i}(j) - \min_{k \in I} \left(\#path_{u,k}(j)\right)}{\max_{k \in I} \left(\#path_{u,k}(j)\right) - \min_{k \in I} \left(\#path_{u,k}(j)\right)} \tag{1}$$

Equation (1) represents the importance of the path $Path(j)$ between u and i in the graph involving all nodes reachable in L hops starting from u. Given the user u and considering the specific path $Path(j)$, we observe how this path is distributed among all items for that user. Since the absolute count values of a path feature for different users might not be comparable, user-based normalization is applied for each feature as proposed by [Liu et al. 2007] for query-document pairs. This formulation allows the

Table II. Path count values computation Example.

John	Tom	Alice
$\#path_{\texttt{John,TheShining}}(14) = 1$	$\#path_{\texttt{Tom,TheShining}}(10) = 1$	$\#path_{\texttt{Alice,TheShining}}(3) = 1$
$\#path_{\texttt{John,TheShining}}(15) = 1$	$\#path_{\texttt{Tom,TheDaVinciCode}}(1) = 1$	$\#path_{\texttt{Alice,TheShining}}(20) = 1$
$\#path_{\texttt{John,TheShining}}(18) = 1$	$\#path_{\texttt{Tom,TheDaVinciCode}}(10) = 2$	$\#path_{\texttt{Alice,TheShining}}(10) = 1$
$\#path_{\texttt{John,TheDaVinciCode}}(6) = 1$	$\#path_{\texttt{Tom,TheDaVinciCode}}(16) = 1$	$\#path_{\texttt{Alice,TheShining}}(15) = 1$
$\#path_{\texttt{John,TheDaVinciCode}}(12) = 1$	$\#path_{\texttt{Tom,TheDaVinciCode}}(8) = 2$	$\#path_{\texttt{Alice,TheShining}}(14) = 1$
$\#path_{\texttt{John,TheDaVinciCode}}(13) = 1$	$\#path_{\texttt{Tom,TheDaVinciCode}}(17) = 1$	$\#path_{\texttt{Alice,TheDaVinciCode}}(9) = 1$
$\#path_{\texttt{John,TheDaVinciCode}}(19) = 1$	$\#path_{\texttt{Tom,TheDaVinciCode}}(4) = 1$	$\#path_{\texttt{Alice,TheDaVinciCode}}(10) = 2$
$\#path_{\texttt{John,AngelsAndDaemons}}(2) = 1$	$\#path_{\texttt{Tom,AngelsAndDaemons}}(16) = 1$	$\#path_{\texttt{Alice,TheDaVinciCode}}(8) = 2$
$\#path_{\texttt{John,AngelsAndDaemons}}(1) = 1$	$\#path_{\texttt{Tom,AngelsAndDaemons}}(11) = 1$	$\#path_{\texttt{Alice,TheDaVinciCode}}(7) = 1$
$\#path_{\texttt{John,AngelsAndDaemons}}(8) = 2$	$\#path_{\texttt{Tom,AngelsAndDaemons}}(17) = 1$	$\#path_{\texttt{Alice,TheDaVinciCode}}(1) = 1$
$\#path_{\texttt{John,AngelsAndDaemons}}(9) = 1$	$\#path_{\texttt{Tom,AngelsAndDaemons}}(5) = 1$	
$\#path_{\texttt{John,AngelsAndDaemons}}(10) = 2$		

system to get better performances compared to the version proposed in [Ostuni et al. 2013].

In order to clarify how to compute the feature vector $\mathbf{x_{ui}}$ and build the training and recommendation sets we compute the path-based features with reference to the graph in Figure 3 and the resulting TS and RS.

Considering the users `John`, `Tom` and `Alice` we have the results shown in Table II. Once we have computed all path count values $\#path_{ui}(j)$ we can compute $\min_{i \in I}\left(\#path_{ui}(j)\right)$ and $\max_{i \in I}\left(\#path_{ui}(j)\right)$ values for each user-path combination. For example, for user `Alice` referring to $Path(10)$ we have $\max_{i \in I}\left(\#path_{\texttt{Alice},i}(10)\right) = 2$ and $\min_{i \in I}\left(\#path_{\texttt{Alice},i}(10)\right) = 0$. Indeed we have $\#path_{\texttt{Alice,TheDaVinciCode}}(j) = 2$ with $j \in \{8, 10\}$ while we have $\#path_{\texttt{Alice,AngelsAndDaemons}}(j) = 0$ for $j \in \{1...20\}$.

Finally we can compute the feature vectors using Formula 1. For example for the pair `Alice`, `TheShining` we have:

$$\mathbf{x}_{\texttt{Alice,TheShining}} = (0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1/2, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1)$$

Tables III and IV show a copy of the training set TS and recommendation set RS built according to the feature values computed above. For better readability purposes we reported in the two tables only paths ranging from $Path(1)$ to $Path(10)$.

3.3.1. Path meaning. Depending on the type of relations composing a path there are different types of paths. In particular, such paths can be: (I) *collaborative* if only links in R are involved as for $(\texttt{like}, \texttt{dislike}^{-1}, \texttt{like})$; (II) *content-based* if there are only $\sigma_l \in P$ with $2 \leq l \leq L$ as for $(\texttt{like}, \texttt{literaryGenre}, \texttt{literaryGenre}^{-1})$; (III) *hybrid* if there is more than one link in R and at least one link in P as for $(\texttt{dislike}, \texttt{like}^{-1}, \texttt{like}, \texttt{subsequentWork}^{-1})$.

From the different paths listed in Table I we can see that each of them has a different meaning. There are basic paths as for example $(\texttt{like}, \texttt{subsequentWork}^{-1})$ which simply recommends the subsequent work of a book the user likes. There are also more complex paths able to catch fine-grained associations between entities. For example $Path(10) = (\texttt{like}, \texttt{subject}, \texttt{broader}, \texttt{broader}^{-1}, \texttt{subject}^{-1})$ it involves the `subject` and `broader` relations. The `subject` property relates an entity to its categories. Categories are very important because are used in Wikipedia for grouping together pages on the same subject. They are organized in `DBpedia` in a taxonomy using the `broader` and `narrower` properties. Specifically, the `broader` property connects specific categories to more generic ones, e.g. `Thriller novels` is the `broader` category of `Techno-thriller novels`. An example of path instance for $Path(10)$ is `Alice` $\xrightarrow{\texttt{like}}$ `AngelsAndDaemon` $\xrightarrow{\texttt{subject}}$ `Techno-thriller novels` $\xrightarrow{\texttt{broader}}$ `Thriller novels` $\xrightarrow{\texttt{broader}^{-1}}$

Table III. Training Set Example.

u	i	$\hat{r}_{ui}$	$\mathbf{x_{ui}}$									
John	TheShining	5	0	0	0	0	0	0	0	0	0	0
John	TheDaVinciCode	4	0	0	0	0	0	**1**	0	0	0	0
Tom	TheShining	1	0	0	0	0	0	0	0	0	0	**0.5**
Tom	AngelsAndDaemons	5	0	0	0	0	**1**	0	0	0	0	0
Alice	AngelsAndDaemons	5	0	0	0	0	0	0	0	0	0	0

Table IV. Recommendation Set Example.

u	i	$\mathbf{x_{ui}}$									
John	AngelsAndDaemons	**1**	**1**	0	0	0	0	0	**1**	**1**	**1**
Tom	TheDaVinciCode	**1**	0	0	**1**	0	0	0	**1**	0	**1**
Alice	TheShining	0	0	**1**	0	0	0	0	0	0	**0.5**
Alice	TheDaVinciCode	**1**	0	0	0	0	0	**1**	**1**	**1**	**1**

`American thriller novels` $\xrightarrow{\texttt{subject}^{-1}}$ `TheDaVinciCode`. An advantage of such complex path is that even if the two items do not belong exactly to the same category, `Techno-thriller novels` and `American thriller novels`, they still belong to categories sharing commonalities between them and hence the two items are somehow related. In this specific case the two categories are connected to the same broader category `Thriller novels`.

3.4. Learning the ranking function

`SPrank` learns the ranking function using learning to rank techniques, which consist in automatically building a ranking model from training data using machine learning techniques such that the model can sort new items according to their relevance. Learning to rank techniques can be classified in three main categories: pointwise, pairwise and listwise.

3.4.1. Pointwise methods. The pointwise approach transforms ranking into regression or classification on single instances. Since the input instance in the pointwise approach is the single item, the relative order between items cannot be considered in the learning process, although ranking is more about predicting relative order than the specific relevance value. Hence, this approach can only be considered as a sub-optimal solution to ranking. The advantage of such pointwise models is that these methods deal well with large datasets.

Random Forest (RF) [Breiman 2001] consists in applying bagging to *Classification And Regression Trees* [Breiman et al. 1984]. Bagging – or bootstrap aggregation – is a technique for reducing the variance of an estimated prediction function. It works especially well for high-variance, low-bias procedures, such as trees. The idea behind this technique is to apply multiple times the same learning algorithm to bootstrap sampled versions of the training set. At the end, all the resulting models are averaged to reduce overfitting.

Multiple Additive Regression Trees (MART) [Friedman 2000], also known as Gradient Boosting Regression Trees, is a linear combination of regression trees constructed using the boosting approach. Instead of training many full high variance trees and average them to avoid overfitting, MART sequentially adds small trees (weak learners), each of them with high bias. During each iteration, the new tree to be added focuses explicitly on the data points that are responsible for the current remaining regression error.

3.4.2. Pairwise methods. The pairwise approach transforms ranking into classification on instance pairs. The goal of learning is to minimize the number of miss-ranked instance pairs. It does not concentrate on accurately predicting the relevance of each

instance as the pointwise; instead, it focuses about the relative order between two instances. Each contribution in the loss function is low when the more relevant item in a pair has higher score than the less relevant item, and viceversa.
RankBoost (RankB) [Freund et al. 2003] adopts AdaBoost [Freund and Schapire 1995] for the classification over item pairs. It constructs a linear combination of weak rankers, optimizing a loss function based on the exponential difference between the relevances of pairs of item. Each weak ranker consists of a single feature and a threshold that best distinguishes between relevant and non-relevant data points.

3.4.3. Listwise methods. The drawback of pointwise and pairwise approaches is that they do not model the ranking problem directly. The listwise approach instead tackle the ranking problem directly by optimizing a loss function defined on the ranked list of instances.
AdaRank (ADA) [Xu and Li 2007] is able to directly optimize any Information Retrieval (IR) evaluation metric using boosting. As in AdaBoost (used for classification tasks) the exponential loss is used to update the distribution of input objects and to determine the combination coefficient at each round of iteration. Analogously, in AdaRank, IR evaluation measures are used to update the distribution of queries/users and to compute the combination coefficient.
LambdaMart (LMART) [Wu et al. 2010] is the combination of LambdaRank [Burges et al. 2006] with the boosted tree optimization method MART. LambdaMart is considered a listwise approach in the fact that during the training it uses the derivative estimate of normalized Discounted Cumulative Gain (nDCG) for fitting the parameters of the regression trees.
Coordinate Ascent (CA) [Metzler and Bruce Croft 2007] is a linear model able to optimize any IR measures directly by cycling through each single feature by optimizing over it while the others are kept fixed until no more improvement is observed. The final output score for each user item pair is calculated as linear combination of the values of the features.

4. EXPERIMENTAL EVALUATION

In this section, we detail the settings and results of the experiments accomplished to evaluate the effectiveness of `SPrank` in terms of ranking accuracy for *top-N* recommendations. The evaluation has been carried out on three datasets belonging to three different domains: `MovieLens` (movies), `LibraryThing` (books) and `Last.fm` (music).

4.1. Datasets description

The first dataset is based on the `MovieLens` 1M dataset[9]. The original dataset contains 1,000,209 1-5 stars ratings given by 6,040 users to 3,883 movies. The second dataset is derived from the `LibraryThing`[10] dataset. This dataset is related to the book domain and contains 7,112 users, 37,231 books and 626,000 ratings ranging from 1 to 10. These first two datasets contain explicit feedback data while in the third dataset an implicit feedback dataset consisting of user-artist listening data has been collected. This dataset comes from recent initiatives on information heterogeneity and fusion in recommender systems[11] [Cantador et al. 2011]. It has been built on top of the `Last.fm` music system[12]. This dataset contains 1,892 users, 17,632 artists and 92,834 relations between a user and a listened artist together with their corresponding listening counts.

[9] `http://www.grouplens.org/node/73`
[10] `www.librarything.com`
[11] `http://ir.ii.uam.es/hetrec2011/datasets.html`
[12] `http://www.lastfm.com`

All the previously introduced datasets do not contain any content-based information. So, in order to build our recommendation engine we enhanced their data by mapping items described in each dataset (movies, artists, songs, books) to their corresponding DBpedia URI as detailed in the following.

Please remember that the following step of *item mapping and linking to DBpedia* is optional (we used it here for experimental purposes) and is not a fundamental part of the approach. SPrank has been conceived to work natively on knowledge graphs built from data available in the LOD cloud. As of today there are already many Web portals exposing their data natively in RDF as *The New York Times*[13], *BBC programmes*[14], *Springer*[15] or *Europeana*[16] just to cite a few.

4.1.1. Item Mapping and Linking to DBpedia. The identification of entities in a text is a well known task in the Natural Language Processing community and more recently it has gained momentum thanks to the availability of knowledge bases publicly available on the Web. Since entity linking (EL) with a knowledge base is a tool we use to create our datasets, for the sake of completeness we briefly recall some of its key points and refer the interested reader to the survey by Shen et al. [Shen et al. 2015].

The main task of an EL system is to disambiguate the mention of an entity e belonging to a knowledge base KB within its textual context and eventually map e to each mention within the analyzed text. Entity Linking tools usually execute a pipe of three main procedures [Shen et al. 2015]:

candidate entity generation. Given a string s, the system tries to identify a set $E = \{e_i\}$ of possible entities matching the actual semantics of s. Most of the approaches to candidate entity generation rely on string matching between s and the text/label associated to an entity $e \in KB$. As a preprocessing step, a dictionary can be built from KB in order to speed up the matching process. The dictionary contains a string as key and possible matching entities as value.

candidate entity ranking. In case $|E| > 1$, the system ranks each candidate in E. The final ranked list can be computed by adopting either supervised or unsupervised methods. In both cases a set of features have to be defined. They can base on text comparison and similarity, entity popularity, entity type (when an underlying ontology is available), text surrounding a string, entities already identified within the text and their mutual coherence (they refer to entities which result somehow connected by means of the knowledge base).

unlinkable mention prediction. In this step the system checks if the top-ranked candidates in E are an actual entity representing s. Also in this case we can have either unsupervised or supervised methods to filter out bad matching entities. A very popular approach is based on a threshold that can be set manually or automatically learned from the examples. Other approaches use binary classifiers or feature-based filtering procedures.

Within the Linked Open Data arena, there are many tools available to perform entity linking from text [Gangemi 2013] which refer to DBpedia or Freebase entities. Most of them reach very good results in terms of number of entities identified within a text. The richer the text, the better the results. Unfortunately, this is not the case with MovieLens, LibraryThing and Last.fm datasets as the text to analyze is just the name

[13] http://data.nytimes.com/
[14] http://www.bbc.co.uk/programmes
[15] http://lod.springer.com/
[16] http://data.europeana.eu/

of a movie, a book or a musical artist/band[17]. In such situations, a very interesting alternative to link the names in our datasets to entities in DBpedia is represented by DBpedia Lookup[18] [Bizer et al. 2009b]. With respect to the three procedures described above, DBpedia Lookup behaves as follows:

candidate entity generation. A dictionary is created via a Lucene index. It is built starting from the values of the property rdfs:label associated to a resource. Very interestingly, the dictionary takes into account also the Wikipedia:Redirect[19] links.

candidate entity ranking. Results computed via a lookup in the dictionary are then weighted combining various string similarity metrics and a PageRank-like relevance ranking.

unlinkable mention prediction. The features offered by DBpedia Lookup to filter out resources from the candidate entities are: (i) selection of entities which are instances of a specific class via the QueryClass parameter; (ii) selection of the top N entities via the MaxHits parameter.

As for the last step we used the QueryClass filter to select entities belonging only to the classes representative of the dataset domain: dbpedia-owl:Film for MovieLens, dbpedia-owl:Book, for LibraryThing and dbpedia-owl:Band, dbpedia-owl:MusicalArtist for Last.fm. Besides the built-in features offered by DBpedia Lookup, depending on the domain of the dataset, we implemented other filters to reduce the number of false positives in the final mapping. As an example, we checked if the category dbpedia:Category:XYZT_films was available as the value of dcterms:subject for the matched DBpedia resource, being XYZT the year available in the name of the movie in MovieLens.

The embedding of Wikipedia:Redirect within the dictionary resulted particularly useful in the musical domain where we may have different names referring to the same entity due to the changing over time of the public name of an artist or of a band. This is the case for instance of the resource dbpedia:Alphex_Twin which is matched by the strings "Alphex Twin", "Caustic Window", "Polygon Window", "AFX" and "Bradley Strider". Analogously we were able to match "Black Francis", "Frank Black" and "Frank Black and The Catholics" to dbpedia:Black_Francis.

After the automatic mapping computed by DBpedia Lookup, for each dataset we manually checked all those suspicious entities, if any, referenced by more than one dataset entry. These situations were more frequent particularly in the Last.fm dataset. Many of them were true positives due to the changing over time of the name of an artist or of a band but this was not always the case.

In the end, for MovieLens we found a positive correspondence for 3,300 out of a total of 3,883 movies. For Last.fm we found a match for 11,180 out of a total of 17,632 artists. Finally for LibraryThing we found a match for 11,694 out of a total of 37,231 books. We randomly checked the dataset entries without a match and for them all there was no corresponding DBpedia URI as they were not available in Wikipedia (see for example the band "*There will be fireworks*"). The dump of the resulted mappings is available at http://sisinflab.poliba.it/semanticweb/lod/recsys/datasets/.

At the end of the mapping phase for each dataset we removed the unmapped items and linked the item IDs with the corresponding DBpedia URIs. Hence, all the unmapped items were removed from the evaluation. Then, to avoid popularity biases

[17]As an example the reader may check the results obtained by DBpedia Spotlight, among the state of the art tools, on the string The Matrix 1999 with the query curl http://spodbpedia.org/rest/annotate --data-urlencode "text=the matrix 1999" --data "confidence=0.5" --data "support=20"

[18]https://github.com/dbpedia/lookup

[19]https://en.wikipedia.org/wiki/Wikipedia:Redirect

from the evaluation [Cremonesi et al. 2010] we removed the top 1% most popular items from the three datasets. We also removed from Last.fm and LibraryThing items with less than five ratings and users who rated less than five items. In MovieLens instead we retained users with at least fifty ratings to have at least one dataset characterized by lower sparsity and hence have a fairer comparison with collaborative filtering algorithms used as competitive approaches.
Table VI contains final statistics for the three datasets. We can see that all three datasets have different sparseness values, MovieLens is quite less sparser than the other two that are very sparse (only about the 1% of all user item rating values are available to train the system).

4.1.2. Knowledge graph construction. Once we have the items URIs collection we can build our own knowledge graph shown in the examples depicted in Figure 1. Specifically, for each URI we extracted from DBpedia its local sub-graph composed by all entities reachable in two hops from the item by performing a breadth-first search (via SPARQL queries). In the queries, we consider the RDF properties belonging to the DBpedia ontology[20] plus three more properties: rdf:type, dcterms:subject and skos:broader which encode different kinds of knowledge classification. The rdf:type property allows us to exploit also the highly informative YAGO (**Y**et **A**nother **G**reat **O**ntology) [Suchanek et al. 2007] classes. For example, for the movie *Pulp Fiction* some of its YAGO classes are: *AmericanBlackComedyFilms*, *1994Films*, *AmericanCriminalComedyFilms*, *FilmsAboutOrganizedCrimeInTheUnitedStates*, *1990sCrimeFilms*. We decided to leverage the properties belonging to the DBpedia Ontology, because they represent high-quality, clean and well-structured information. The list of all the properties related to the Film class in the DBpedia ontology is available at http://mappings.dbpedia.org/server/ontology/classes/Film. The list related to the Musical Artist and Band classes is available at http://mappings.dbpedia.org/server/ontology/classes/MusicalArtist and http://mappings.dbpedia.org/server/ontology/classes/Band respectively. The list related to the Book class is available at http://mappings.dbpedia.org/server/ontology/classes/Book.
At the end, each final graph contains: 4,186 users, 3,196 items and 426,245 entities for MovieLens; 7,149 users, 4,541 items and 948,291 entities for LibraryThing; 1,875 users, 2,432 items and 1,642,137 entities for Last.fm.

For MovieLens and LibraryThing instead of converting directly all ratings to relevance relations we set a threshold and associated to those ratings above the threshold the relation like and to the others the relation dislike. In MovieLens we set as threshold the rating 4 and in LibraryThing the rating 8. In Last.fm all implicit feedback are labelled with like.

In Table V we show the execution time of SPrank to compute the paths. The implementation of the algorithm follows a two steps computation. In the first step, it computes and indexes the paths in the RDF graph without taking into account users. Eventually, these paths are used, if needed, to compute the *collaborative*, *content-based* and *hybrid* paths needed by SPrank. The computation has been executed on a machine equipped with 16 CPU Intel(R) Xeon(R) X5570 @ 2.93GHz (4 cores per CPU), 32 GB RAM and Ubuntu 12.04.5 LTS. The implementation we adopted for our experiments is available at https://github.com/sisinflab/lodreclib.

4.1.3. Path analysis. In what follows we describe part of the most informative paths extracted from the three datasets. For all user-item pairs in each dataset we extracted all path instances and collected the relative paths. Overall have been extracted 1144

[20] http://wiki.dbpedia.org/Downloads38#dbpedia-ontology

Table V. Execution time to compute paths.

dataset	time (s)
MovieLens	3,968
LibraryThing	1,432
Last.fm	3,277

Table VI. Datasets Statistics. As for # items we indicate the number of items selected after removing both the top 1% most popular items and the items with less than five ratings for Last.fm and LibraryThing. The value is followed in round brackets by the original value found with the mapping.

dataset	# users	# items	# ratings	spars.(%)	# entities
MovieLens	4,186	3,196 (3,300)	822,597	93.85	426,245
LibraryThing	7,149	4,541 (10,181)	352,123	98.90	948,291
Last.fm	1,875	2,432 (11,180)	44,981	99.01	1,642,137

different paths in MovieLens, 632 in LibraryThing and 1988 in Last.fm. To reduce the number of features we performed feature selection by fitting a Random Forest model on the training data and used correspondent variable importance [Breiman 2001] for selecting only the most significant paths among all extracted ones. Using cross-validation we selected the first 150 paths for MovieLens, and the first 200 ones for LibraryThing and for Last.fm. Tables VII, VIII and IX show the 20 most important paths in MovieLens, LibraryThing and Last.fm, respectively. Rows are sorted by path importance. From the analysis of the first five paths in all three datasets we can see that all of them somehow involve the dcterms:subject, dbpedia-owl:genre and dbpedia-owl:wikiPageWikiLink properties and in some cases also rdf:type. These properties are important because they connect items with other items belonging to topics or categories semantically related. The wikiPageWikiLink property is even more interesting because it maps hypertextual links between Wikipedia pages. Such information can be really precious because it models less intuitive connections. For example in the Wikipedia page of Adele there is a wikiPageWikiLink link to the Wikipedia page related to the *Coachella Valley Music and Arts Festival* which is an annual three-day music and arts festival located in California. We could leverage this particular connection to find other artists who performed at Coachella as well.

From the results reported in the three tables we see that these fine-grained semantic paths are very informative. Moreover, in the first 20 paths there are also pure collaborative paths which are as much important as the other complex content-based paths. Another important thing we can note is that most of the paths in the top 20 are paths of length $L > 3$ meaning that direct connections between items, that is items directly connected at one hop, seem less important. In fact, the most important path in MovieLens is $(\texttt{dislike}, \texttt{distributor}, \texttt{subject}, \texttt{broader}, \texttt{subject}^{-1})$ which has length 5.

4.2. Evaluation protocol and experiment setting

For the evaluation of our approach we adopted the "all unrated items" methodology presented in [Steck 2013]. It consists in creating a *top-N* recommendation list for each user by predicting a score for every item not rated by that particular user, whether the item appears in the user test set or not. Then, performance metrics are computed comparing recommendation lists with test data. This evaluation strategy is equivalent to the AllItems methodology presented in [Bellogin et al. 2011] and similar to the TrainItems one recently presented in [Said and Bellogin 2014]. The main assumption in this methodology is that all the *non-rated items* are considered to be *irrelevant* for the user with the effect of underestimating real recommendation quality. However, as

Table VII. Portion of the Path Index extracted from the `MovieLens` graph referring to the top-20 most important paths.

pathID	path imp. (%)	path
293	0.0087	(dislike, distributor, subject, broader, subject^{-1})
13	0.0078	(like, subject, broader, subject^{-1})
96	0.0067	(like, wikiPageWikiLink^{-1}, wikiPageWikiLink^{-1})
301	0.0063	(like, producer, subject, type^{-1}, type^{-1})
91	0.0062	(like, wikiPageWikiLink, wikiPageWikiLink^{-1}, subject^{-1}, writer^{-1})
15	0.0059	(like, wikiPageWikiLink, subject^{-1})
220	0.0059	(dislike, like^{-1}, dislike)
37	0.0059	(like, wikiPageWikiLink, subject, subject^{-1}, distributor^{-1})
22	0.0058	(like, subject, broader, subject^{-1}, wikiPageWikiLink^{-1})
314	0.0058	(dislike, wikiPageWikiLink^{-1}, starring^{-1})
104	0.0057	(like, subject, wikiPageWikiLink^{-1})
313	0.0056	(dislike, wikiPageWikiLink, starring^{-1})
120	0.0056	(like, producer, subject, subject^{-1}, wikiPageWikiLink^{-1})
111	0.0055	(like, wikiPageWikiLink, subject, wikiPageWikiLink^{-1})
278	0.0054	(dislike, wikiPageWikiLink, subject, wikiPageWikiLink^{-1})
89	0.0054	(like, wikiPageWikiLink^{-1}, subject, subject^{-1}, writer^{-1})
99	0.0053	(like, starring, subject, wikiPageWikiLink^{-1})
100	0.0051	(like, starring, subject, subject^{-1}, wikiPageWikiLink^{-1})
315	0.0051	(dislike, distributor, subject, broader, subject^{-1})
48	0.0049	(like, dislike^{-1}, like)

Table VIII. Portion of the Path Index extracted from the `LibraryThing` graph referring to the top-20 most important paths.

pathID	path imp. (%)	path
162	0.0273	(like, wikiPageWikiLink^{-1}, genre^{-1}, literaryGenre^{-1})
148	0.0224	(dislike, literaryGenre, subject, subject^{-1}, literaryGenre^{-1})
27	0.0216	(like, literaryGenre, type, type^{-1})
56	0.0197	(dislike, wikiPageWikiLink, subject, subject^{-1}, publisher^{-1})
147	0.0188	(dislike, wikiPageWikiLink, subject, subject^{-1}, literaryGenre^{-1})
237	0.0171	(like, previousWork, subject^{-1}, wikiPageWikiLink^{-1})
138	0.0171	(dislike, subject, subject^{-1})
82	0.0168	(like, wikiPageWikiLink, subject, wikiPageWikiLink^{-1}, publisher^{-1})
54	0.0167	(like, subject, subject^{-1}, wikiPageWikiLink^{-1})
69	0.0162	(dislike, wikiPageWikiLink, subject^{-1}, publisher, publisher^{-1})
175	0.0147	(dislike, literaryGenre, genre^{-1}, wikiPageWikiLink^{-1})
83	0.0137	(dislike, like^{-1}, dislike)
102	0.0136	(like, like^{-1}, like)
61	0.0122	(dislike, literaryGenre, wikiPageWikiLink^{-1})
100	0.0115	(dislike, wikiPageWikiLink^{-1}, wikiPageWikiLink)
106	0.0114	(like, publisher, publisher^{-1}, wikiPageWikiLink)
167	0.0113	(dislike, wikiPageWikiLink, subject, wikiPageWikiLink)
166	0.0113	(dislike, wikiPageWikiLink, subject, subject^{-1})
155	0.0112	(like, wikiPageWikiLink^{-1}, subject, subject^{-1}, publisher^{-1})
173	0.0109	(dislike, wikiPageWikiLink^{-1}, wikiPageWikiLink^{-1})

pointed out in [Steck 2013], since the user-experience in *top-N* recommendation applications depends on the ranking of all items, this is a better evaluation methodology than the "rated test-items" one [Steck 2013] where only rated test items are considered for generating the *top-N* list.

The evaluation has been carried out using the holdout method consisting in splitting the data in two disjoint sets: the one for training and the other for testing. We used 80% of ratings in $\hat{R}$ as training ratings for building the training set TS and remaining 20% as test ratings for measuring recommendation accuracy. The tuning of model hyper-parameters in the various learning to rank algorithms we tested was performed

Table IX. Portion of the Path Index extracted from the Last.fm graph referring to the top-20 most important paths.

pathID	path imp. (%)	path
90	0.0170	(like, associatedBand, genre, wikiPageWikiLink^{-1})
79	0.0153	(like, associatedBand^{-1}, genre, genre^{-1}, writer^{-1})
72	0.0149	(like, subject, subject^{-1}, wikiPageWikiLink^{-1})
137	0.0147	(like, musicalArtist^{-1}, genre, genre^{-1})
160	0.0141	(like, associatedMusicalArtist, type, type^{-1}, writer)
20	0.0140	(like, musicalBand, genre, genre^{-1}, artist^{-1})
22	0.0139	(like, associatedMusicalArtist^{-1}, subject, wikiPageWikiLink)
99	0.0133	(like, subject, subject^{-1}, subject, wikiPageWikiLink^{-1})
120	0.0131	(like, associatedBand^{-1}, associatedMusicalArtist^{-1}, genre, genre^{-1})
38	0.0120	(like, musicalBand^{-1}, genre, musicSubgenre^{-1}, genre^{-1})
158	0.0118	(like, genre, stylisticOrigin, wikiPageWikiLink^{-1})
82	0.0111	(like, associatedBand^{-1}, genre, genre^{-1}, musicalBand^{-1})
155	0.0111	(like, associatedMusicalArtist^{-1}, genre, genre^{-1})
169	0.0111	(like, wikiPageWikiLink, subject, subject^{-1}, artist)
75	0.0105	(like, genre, musicSubgenre^{-1}, stylisticOrigin^{-1}, genre^{-1})
43	0.0101	(like, genre, musicFusionGenre^{-1}, stylisticOrigin, genre^{-1})
114	0.0101	(like, musicalArtist^{-1}, subject, subject^{-1}, writer)
113	0.0099	(like, associatedBand, associatedMusicalArtist, genre, genre^{-1})
95	0.0098	(like, associatedBand^{-1}, associatedMusicalArtist^{-1}, subject, subject^{-1})
15	0.0098	(like, genre, stylisticOrigin, wikiPageWikiLink)

through cross-validation on validation data obtained by selecting the 15% of ratings for each user from the training set TS. We repeated the experiment three times for each dataset, and then we averaged the results over the three runs.

For measuring recommendation accuracy we adopted the following standard performance metrics: Precision, Recall and nDCG. The first two are binary metrics in the way that they take into account binary relevance values (relevant/irrelevant), while nDCG can handle graded values.

Precision@N (prec@N) is computed as the fraction of *top-N* recommended items appearing in the test set that are relevant for the user, while Recall@N (rec@N) is computed as the ratio of *top-N* recommended items that are relevant to the total number of relevant items in the test set. We considered as relevant those items rated with four and five stars in MovieLens and more than seven in LibraryThing. For the implicit feedback dataset Last.fm instead, we considered as relevant each artist the user listened to (only-positive implicit feedback scenario).

Differently from precision and recall which consider only the relevance of items, nDCG takes into account both relevance and rank position. Denoting with $\hat{r}_{uk}$ the rating given by user u to the item in position k in the *top-N* list, then nDCG@N for u can be defined as:

$$\text{nDCG@N} = \frac{1}{\text{IDCG@N}} \sum_{k=1}^{N} \frac{2^{\hat{r}_{uk}} - 1}{\log_2(1+k)} \tag{2}$$

where IDCG@N indicates the score obtained by an ideal or perfect *top-N* ranking and acts as normalization factor. For those items with no rating in the test set we assume a fixed default value as suggested in [Steck 2013]. We chose 3 as default value for MovieLens and 5 for LibraryThing because they can be considered neutral ratings in their respective scales. All the reported metrics are computed for each single user and eventually averaged.

4.2.1. Measuring recommendation inaccuracy. When assuming all non-rated items as being irrelevant, those items are treated equally to those which actually are irrelevant (e.g. one and two star ratings in MovieLens). Apparently in this setting the information related to low rated items is completely ignored. Indeed, such valuable information can

be used for measuring the ability of the system of avoiding very bad recommendations. We can reasonably assume that items rated one star on MovieLens are recommendations the user would never receive. We argue that, depending on the domain, this can be an important quality of the system to preserve.

Inspired by the %no metric used in IR for measuring system ineffectiveness [Voorhees 2004] we propose the %bad measure. As the %no metric is the percentage of queries for which no relevant documents are retrieved, the %bad metric is the percentage of users receiving at least one bad recommendation in their *top-N* list. The higher the %bad the worse the general recommendation quality.

In addition we measure also the fall-out (or false positive rate) that in our scenario can be defined as the fraction of bad items in the test set which appear in the *top-N* recommendation list. This metric can be useful to measure the quantity of bad items the system recommends.

We applied these measures to both MovieLens and LibraryThing but not to Last.fm since it contains positive only feedback. In MovieLens we consider bad items those rated one or two stars and in LibraryThing those rated less than five stars.

4.3. Evaluation and results discussion

In this section we detail the experiments we accomplished to evaluate the effectiveness of the proposed approach in terms of *top-N* recommendation accuracy. Specifically, we are interested in analysing the following aspects:

— *effectiveness of considering user rating relations into semantic paths;*
— *comparison of various learning to rank algorithms for computing top-N recommendations;*
— *comparison of the proposed approach with state-of-the-art recommendation algorithms.*

4.3.1. Analysis about the integration of user rating relations into semantic paths. As described in Section 3.3 we define path the sequence of relations of the form $(r_1 \dots \sigma_l \dots \sigma_L)$ where the first relation expresses the user's preference degree in the rated item. Apparently, adding the relevance relation like/dislike to the path may result redundant for the overall framework. Here we analyse whether the usage of such relevance relation in the path connecting the user to the item helps in matching user's preferences with the item description or not. We compare two path variants: (i) *relevance prefix*, (ii) *no relevance prefix*.

To understand better the difference between the two variants let us use an example with reference to the graph in Figure 3 and paths in Table I. We want to establish whether TheDaVinciCode might be a good book to recommend to Tom who expressed a positive preference for the book AngelsAndDaemon and a negative one for TheShining. Let us consider the following two path instances $\texttt{Tom} \xrightarrow{\texttt{like}} \texttt{AngelsAndDaemon} \xrightarrow{\texttt{subject}} \texttt{Techno-thriller novels} \xrightarrow{\texttt{broader}} \texttt{Thriller novels} \xrightarrow{\texttt{broader}^{-1}} \texttt{American thriller novels} \xrightarrow{\texttt{subject}^{-1}} \texttt{TheDaVinciCode}$ and $\texttt{Tom} \xrightarrow{\texttt{dislike}} \texttt{TheShining} \xrightarrow{\texttt{subject}} \texttt{American horror novels} \xrightarrow{\texttt{broader}} \texttt{American novels by genre} \xrightarrow{\texttt{broader}^{-1}} \texttt{American thriller novels} \xrightarrow{\texttt{subject}^{-1}} \texttt{TheDaVinciCode}$ belonging to $Path(10)$ and $Path(11)$, respectively. We can note that they differ only for the first relation. If we ignore the relevance relation in the path-based feature (*no relevance prefix* option) then both path instances would fall into the same path $(\texttt{subject}, \texttt{broader}, \texttt{broader}^{-1}, \texttt{subject}^{-1})$. Thus, the recommender would be unable to discern between path instances connecting the target item to what

Table X. Path-based feature variants comparison.

	MovieLens			LibraryThing		
	nDCG@10	prec@10	rec@10	nDCG@10	prec@10	rec@10
no relevance pref.	0.3765	0.1159	0.0528	0.2312	0.0587	0.1237
relevance prefix	0.4056	0.1613	0.0785	0.3068	0.1019	0.2232

Table XI. Comparative results of the various learning to rank algorithms tested on MovieLens.

Alg	nDCG@10	prec@10	rec@10	%bad@10	fall-out@10
ADA	0.4228	0.1893	0.0802	0.0798	0.0180
LMART	0.4056	0.1613	0.0785	0.0542	0.0150
CA	0.4042	0.1472	0.0741	0.0331	0.0082
RANKB	0.3827	0.1389	0.0667	0.0515	0.0136
MART	0.3291	0.0595	0.0242	0.0197	0.0048
RF	0.3273	0.0572	0.0240	0.0212	0.0031

Table XII. Comparative results on LibraryThing.

Alg	nDCG@10	prec@10	rec@10	%bad@10	fall-out@10
ADA	0.2429	0.0591	0.1354	0.0198	0.0521
LMART	0.3068	0.1019	0.2232	0.0378	0.0171
CA	0.2456	0.0631	0.1383	0.0136	0.0476
RANKB	0.2194	0.0503	0.1221	0.0153	0.0356
MART	0.1639	0.0152	0.0312	0.0017	0.0052
RF	0.1471	0.0241	0.0204	0.0012	0.0023

the user likes and what she dislike because both path instances are encountered into the same feature.

In order to establish which path variant gives us the better accuracy we selected AdaRank as learning algorithm and trained and evaluated it on two different versions of the datasets, one built following the *no relevance prefix* option and the other following the *relevance prefix* option. The results obtained with the two different options are reported in Table X. As expected, we achieve better accuracy on both MovieLens and LibraryThing when the relevance relation is taken into account in the path (*relevance prefix* option). The difference is more marked on MovieLens probably because of the different ratings distribution. In LibraryThing the proportion of like relations is much bigger than dislike because most of the ratings are above the threshold 8. Instead in MovieLens the amount of like and dislike is more balanced, hence considering the relevance as prefix of the path in this case seems to have more influence. We did not perform the same experiment on Last.fm because it contains only positive feedback and in such case there is no difference between the two options.

4.3.2. Comparison of learning to rank algorithms. Another important part of the study we conducted regards the comparison of various learning to rank algorithms for computing *top-N* hybrid recommendations. Particularly we experimentally compared the various learning to rank methods described in Section 3.4. For the implementation of those algorithms we used the open source library *RankLib*[21]. For all listwise algorithms we used nDCG@10 as training and validation metric. Tables XI, XII and XIII show the results obtained for all the tested algorithms on the three datasets. As we can see there is no global algorithm which performs always better than the others. In MovieLens the best approach is ADARank while on LibraryThing and Last.fm is LambdaMART. The main important insight we can draw is that in all three cases listwise methods outperform both the pairwise and pointwise ones and also that the only pairwise method tested outperforms the other two pointwise methods. We can note also that in Last.fm

[21] http://people.cs.umass.edu/~vdang/ranklib.html

Table XIII. Comparative results on Last.fm.

Alg	nDCG@10	prec@10	rec@10
ADA	0.1688	0.0773	0.1915
LMART	0.1918	0.0841	0.2043
CA	0.1686	0.0787	0.1914
RANKB	0.1666	0.0778	0.1869
MART	0.1639	0.0779	0.1912
RF	0.1632	0.0775	0.1888

the differences between all different methods but LambdaMART are very limited. A reason for this could be the limited size of the training set with respect to the other two datasets.
As expected, from the obtained results we can confirm that listwise approaches, which directly learn the ranking model by optimizing a ranking measure on the top results, outperform both pointwise and pairwise methods for learning the ranking function in SPrank. These results are in accordance with the research done in the past few years about learning to rank in collaborative filtering [Rendle et al. 2009; Shi et al. 2012; Weimer et al. 2007; Liu and Yang 2008; Balakrishnan and Chopra 2012; Volkovs and Zemel 2012] and Web search [Liu 2009].

4.3.3. Comparison with other methods. In order to evaluate the effectiveness in terms of ranking accuracy of SPrank we compared it with several state of the art recommendation algorithms:

— PopRank is a popularity-based baseline approach which provides the same recommendation to all users based on the global popularity of items. Recent studies have shown that recommending the most popular items could already result in a high performance [Cremonesi et al. 2010].
— Bayesian personalized ranking (BPR-MF) uses a matrix factorization as the learning model with Bayesian Personalized Ranking optimization criterion [Rendle et al. 2009].
— BPRLinearMap (BPRLin) is a hybrid matrix factorization method for positive implicit feedback able to work in cold-start scenarios [Gantner et al. 2010]. It learns a linear mapping on the user/item features from the factorization and the auxiliary user/item-attribute matrix. In the experiments we built the item-attribute matrix using all entities directly connected to the items.
— SLIM [Ning and Karypis 2011] uses a Sparse Linear method for learning a sparse aggregation coefficient matrix W to be used for computing the *top-N* recommendations.
— Soft Margin Ranking Matrix Factorization (RankingMF) is a matrix factorization model for item ranking which uses ordinal regression score as loss function [Weimer et al. 2008].
— Biased Matrix Factorization (MF) is a matrix factorization model that minimizes RMSE using stochastic gradient descent [Koren et al. 2009]. It consider also explicit user and item biases in the final rating prediction. MF represents the state-of-the-art for rating prediction tasks.
— Markov Chain Monte Carlo (MCMC-FM) is a factorization machine model using Markov Chain Monto Carlo inference [Salakhutdinov and Mnih 2008].
— KNN-FM is an item-based K-Nearest Neighbours algorithm using factorization machine for modeling the item features [Rendle 2010].
— Ranking-FM is a ranking-based matrix factorization and factorization machine algorithm.

— HeteRec is a matrix factorization algorithm that uses meta-path based latent features to represent the connectivity between users and items along different types of paths in a heterogeneous information network [Yu et al. 2014].
— PathRank is random-walk based node ranking measure for heterogeneous network, extending the Personalized PageRank [Lee et al. 2012].
— BPR-SSLIM is the extended version of SLIM using also the item side information [Ning and Karypis 2012] and the BPR optimization criterion.

The computation of the recommendations for MF, SLIM, BPR-MF, BPR-SSLIM, PopRank, BPRLin and RankMF has been done with the publicly available software library *MyMediaLite*[22] [Gantner et al. 2011], while for MCMC-FM we have used the *LibFM*[23] [Rendle 2012] library and for KNN-FM and Ranking-FM we used the *GraphLab*[24] [Low et al. 2012] implementation. HeteRec and PathRank have been reimplemented from scratch. For SLIM, RankingMF and BPR-MF since they base on implicit feedback or binary data we considered only relevant items during the training and removed irrelevant training items from output recommendation lists. In this way all the recommenders consider for each user the same candidate items to compute the *top-N* recommendation list. For SPrank we chose as learning algorithm the best performing, on average, in each dataset: LambdaMART[25].

Results discussion. Table XIV shows the results obtained on the MovieLens dataset. Looking at the accuracy metrics nDCG, precision and recall SPrank outperforms the popularity baseline and other popular approaches as the matrix factorization model for rating prediction and the hybrid learning to rank algorithm BPRLin. Furthermore, SPrank outperforms also the other graph-based approaches - HeteRec and PathRank - in terms of accuracy. Nevertheless, SPrank does not beat the other hybrid learning to rank methods SLIM, BPR-MF, Ranking-FM and BPR-SSLIM. However we argue that another important quality of recommendation list to preserve is avoiding very bad recommendations. Then, if we look at the %bad@10 and fall-out@10 results we see that both SLIM and BPR-MF are the two approaches giving the worst performances, their fall-out is more than twice worse the SPrank value. On the other hand, BPR-SSLIM obtains a better %bad@10 than SPrank, but a worst fall-out@10. With reference to Ranking-FM, SPrank performs worse for each metrics. Summarizing on MovieLens, SPrank outperforms some of the other algorithms in terms of ranking accuracy, but it gets worse performances compared to most of the hybrid learning to rank approaches. However it is much more conservative in avoiding defeating recommendations compared to most of them.

Table XV shows the results for the LibraryThing dataset. In this case SPrank outperforms widely all the other approaches in terms of ranking accuracy, except for BPR-SSLIM, that obtains better recall and %bad@10, but worst precision and fall-out values (there is no statistically significant difference in terms of nDCG). Another close approach is the other graph-based method HeteRec, which obtains anyway still lower values than SPrank. The most conservative approach in terms of %bad@10 and fall-out@10 is MF.

Finally, Table XVI reports the results obtained on the Last.fm data. On this dataset SPrank beats all the other approaches. Only BPR-SSLIM provides a better recall, but SPrank obtains strongly better values for nDCG and precision. Even if this dataset is

[22] http://www.mymedialite.net
[23] http://www.libfm.org
[24] http://graphlab.com
[25] For MovieLens, we have ADARank performing better than LambdaMart. Hence, ADARank can be used in place of LambdaMART to improve SPrank performances in MovieLens.

Table XIV. Comparative results on MovieLens. The differences between SPRank and the comparative approaches are statistically significant according to the Student's paired t-test [Smucker et al. 2007] with $p < 0.0001$ (for Ranking-FM the differences are still significant but with $p < 0.01$).

Alg	nDCG@10	prec@10	rec@10	%bad@10	fall-out@10
SPrank	0.4056	0.1613	0.0785	0.0542	0.0150
MF	0.3618	0.0945	0.0422	0.0113	0.0016
SLIM	0.4597	0.2631	**0.1469**	0.1876	0.0471
BPR-MF	0.4359	0.2572	0.1354	0.1874	0.0459
PopRank	0.3915	0.1663	0.0753	0.0943	0.0205
BPRLin	0.3752	0.1437	0.0731	0.1729	0.0352
RankMF	0.3695	0.1428	0.0739	0.1870	0.0312
MCMC-FM	0.3576	0.0902	0.0414	**0.0012**	**0.0014**
KNN-FM	0.3521	0.0870	0.0388	0.0024	0.0040
Ranking-FM	0.4179	0.1850	0.0846	0.0478	0.0111
HeteRec	0.3404	0.0884	0.0478	0.0060	0.0105
PathRank	0.3477	0.0881	0.0488	0.0034	0.0084
BPR-SSLIM	**0.4674**	**0.2636**	0.1439	0.0124	0.0247

sparse and pure CF methods are a little penalized in such case, SPrank extensively outperforms the other hybrid approaches - BPRLinear, factorization machines based algorithms, HeteRec and PathRank.

Summing up, in all three datasets SPrank outperforms the popularity baseline and the well established biased matrix factorization approach for rating prediction.It represents the best algorithm on Last.fm and the second best on LibraryThing in terms of accuracy. As for LibraryThing, it obtains accuracy results very close to the ones of BPR-SSLIM. The main difference with this latter is on the %bad and fall-out values. SPrank tends to recommend less bad items (lower fall-out than BPR-SSLIM) and it distributes them on a higher number of users. Conversely, BPR-SSLIM suggest a higher number of bad items and it concentrate them on a smaller set of users.

SPrank is not able to outperform the other hybrid algorithms on MovieLens, but it still substantially recommends much less bad items. Considering the statistical differences among the three datasets shown in Table VI, it is worthy to note that SPrank would be the best choice on dataset with fewer ratings and more content information, as Last.fm in this case that is the most sparse and enriched dataset. Instead, considering MovieLens, the less sparse and enriched dataset in this work, SPrank is not as able as the other hybrid learning to rank approaches where the collaborative component plays an important role. This is confirmed by the results on LibraryThing, that has intermediate statistical values and where SPrank strongly overcomes almost all the comparative algorithms.

All in all, we may say that SPrank could be an interesting alternative for those datasets where the rating matrix is sparse but the content-based part may benefit from rich LOD datasets.

5. CONCLUSION

We have presented SPrank, a novel hybrid *top-N* item recommendation algorithm able to effectively exploit the knowlege encoded in LOD datasets for computing accurate *top-N* recommendations. In SPrank, user feedback data and semantic item descriptions extracted from LOD are merged in a unique graph-based representation that allows us to extract *path-based* features describing complex relationships between users and items. Then, the recommendation problem is formulated in a learning to rank setting similar to the one adopted in Web search. We have conducted various experiments on three different datasets, MovieLens, Last.fm and LibraryThing, to: (i) study and compare the accuracy performances of various learning to rank algorithms within our

Table XV. Comparative results on LibraryThing. The differences between SPRank and the comparative approaches are statistically significant according to the Student's paired t-test with p $<$ 0.0001 (except with respect to BPR-SSLIM in terms of nDCG@10 with p-value $>$ 0.1).

Alg	nDCG@10	prec@10	rec@10	%bad@10	fall-out@10
SPrank	**0.3068**	**0.1019**	0.2232	0.0378	0.0171
MF	0.1423	0.017	0.0209	**0.0022**	**0.0025**
SLIM	0.2317	0.0543	0.0988	0.0139	0.0281
BPRMF	0.1858	0.0449	0.0751	0.0112	0.026
PopRank	0.1598	0.0397	0.0452	0.0121	0.0266
BPRLin	0.275	0.0794	0.1723	0.0192	0.041
RankMF	0.1714	0.0369	0.0459	0.0094	0.0152
MCMC-FM	0.1421	0.0362	0.1299	0.0024	0.0125
KNN-FM	0.1723	0.0249	0.0655	0.0018	0.0121
Ranking-FM	0.1958	0.0431	0.0798	0.0027	0.0136
HeteRec	0.2894	0.0930	0.2125	0.0515	0.0377
PathRank	0.2568	0.0521	0.1116	0.0089	0.0237
BPR-SSLIM	0.3051	0.0962	**0.2328**	0.0085	0.0494

Table XVI. Comparative results on Last.fm. The differences between SPRank and the comparative approaches are statistically significant according to the Student's paired t-test with p $<$ 0.0001 (except with respect to BPRMF in terms of prec@10 with p $>$ 0.1).

Alg	nDCG@10	prec@10	rec@10
SPrank	**0.1918**	**0.0841**	0.2043
SLIM	0.1395	0.0723	0.1548
BPRMF	0.1496	0.0785	0.1523
PopRank	0.0684	0.0537	0.0756
BPRLin	0.1058	0.0521	0.1235
RankMF	0.0689	0.0487	0.0876
MCMC-FM	0.0317	0.0102	0.0574
KNN-FM	0.0248	0.0081	0.0445
Ranking-FM	0.0460	0.0160	0.0902
HeteRec	0.0669	0.0207	0.1199
PathRank	0.1442	0.0387	0.1878
BPR-SSLIM	0.1737	0.0467	**0.2605**

framework; (ii) validate the effectiveness of our approach with respect to other state-of-the-art approaches. Regarding the comparison of the various learning to rank algorithms, as expected, listwise approaches outperform in terms of ranking accuracy the pointwise and pairwise ones. The results we obtained validate the hypothesis that ranking based approaches better address the *top-N* recommendation task with respect to pointwise methods which are usually better suited for the rating prediction task. Among the three listwise methods we tested, LambdaMART is, on average, best performing algorithm.

Regarding the comparison of our approach with other competitive approaches, the results show that SPrank shows its effectiveness for sparse datasets while remaining competitive with dense ones. With reference to the MovieLens dataset, it shows interesting ranking accuracy with respect to other state-of-the-art learning to rank collaborative filtering algorithms with the advantage of being more conservative in making bad recommendations. Instead in both Last.fm and LibraryThing the proposed framework outperforms the other learning to rank CF algorithms in terms of ranking accuracy. Summing up, SPrank is able to achieve high ranking accuracy as CF ranking based algorithms which however recommend a high percentage of bad items and, at the same time, recommend a low ratio of bad items as rating prediction methods such as matrix factorization. With SPrank, we have also shown that a new breed of recom-

mendation engines may benefit from the availability of publicly available Linked Open Data datasets by exploiting both the knowledge they encode and their graph-based structure.

Acknowledgements. We would like to thank Alejandro Bellogin for his valuable comments on the evaluation section and Francesco Paolo Albano for his help with the creation of the mappings via DBpedia Lookup. The authors acknowledge partial support of PON01_00850 ASK-Health and PON02_00563_3470993 VINCENTE.

REFERENCES

Sarabjot Singh Anand, Patricia Kearney, and Mary Shapcott. 2007. Generating semantically enriched user profiles for Web personalization. *ACM Trans. Internet Technol.* 7, 4, Article 22 (Oct. 2007).

Suhrid Balakrishnan and Sumit Chopra. 2012. Collaborative Ranking. In *Proceedings of the Fifth ACM International Conference on Web Search and Data Mining (WSDM '12)*. ACM, New York, NY, USA, 143–152.

Alejandro Bellogin, Pablo Castells, and Ivan Cantador. 2011. Precision-oriented Evaluation of Recommender Systems: An Algorithmic Comparison. In *Proceedings of the Fifth ACM Conference on Recommender Systems (RecSys '11)*. ACM, New York, NY, USA, 333–336.

Christian Bizer, Tom Heath, and Tim Berners-Lee. 2009a. Linked Data - The Story So Far. *Int. J. Semantic Web Inf. Syst* 5, 3 (2009), 1–22.

Christian Bizer, Jens Lehmann, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak, and Sebastian Hellmann. 2009b. {DBpedia} - A crystallization point for the Web of Data. *Web Semantics: Science, Services and Agents on the World Wide Web* 7, 3 (2009), 154 – 165. The Web of Data.

Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: A Collaboratively Created Graph Database for Structuring Human Knowledge. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data (SIGMOD '08)*. ACM, New York, NY, USA, 1247–1250.

Leo Breiman. 2001. Random Forests. *Machine Learning* 45, 1 (2001), 5–32.

Leo Breiman, Jerome Friedman, Charles J. Stone, and R. A. Olshen. 1984. *Classification and Regression Trees* (1 ed.). Chapman and Hall/CRC.

Christopher J. C. Burges, Robert Ragno, and Quoc V. Le. 2006. Learning to Rank with Nonsmooth Cost Functions. In *NIPS*, Bernhard Schölkopf, John C. Platt, Thomas Hoffman, Bernhard Schölkopf, John C. Platt, and Thomas Hoffman (Eds.). MIT Press, 193–200.

Robin Burke. 2002. Hybrid Recommender Systems: Survey and Experiments. *User Modeling and User-Adapted Interaction* 12, 4 (Nov. 2002), 331–370.

Iván Cantador, Alejandro Bellogín, and Pablo Castells. 2008. A multilayer ontology-based hybrid recommendation model. *AI Commun. Special Issue on Rec. Sys.* 21, 2-3 (April 2008), 203–210.

Iván Cantador, Peter Brusilovsky, and Tsvi Kuflik. 2011. 2nd Workshop on Information Heterogeneity and Fusion in Recommender Systems (HetRec 2011). In *Proceedings of the 5th ACM conference on Recommender systems (RecSys 2011)*. ACM, New York, NY, USA.

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 (RecSys '10)*. ACM, New York, NY, USA, 39–46. DOI:http://dx.doi.org/10.1145/1864708.1864721

Tommaso Di Noia, Ivàn Cantador, and Vito Claudio Ostuni. 2014. Linked Open Data-enabled Recommender Systems: ESWC 2014 Challenge on Book Recommendation. In *Semantic Web Evaluation Challenges 2014*.

Tommaso Di Noia, Roberto Mirizzi, Vito Claudio Ostuni, and Davide Romito. 2012a. Exploiting the web of data in model-based recommender systems. In *Proceedings of the sixth ACM conference on Recommender systems (RecSys '12)*. ACM, New York, NY, USA, 253–256.

Tommaso Di Noia, Roberto Mirizzi, Vito Claudio Ostuni, Davide Romito, and Markus Zanker. 2012b. Linked open data to support content-based recommender systems. In *Proceedings of the 8th International Conference on Semantic Systems (I-SEMANTICS '12)*. ACM, New York, NY, USA, 1–8.

Frank Manola, Eric Miller. 2004. RDF Primer. http://www.w3.org/TR/rdf-primer. (2004).

Ignacio Fernández-Tobías, Iván Cantador, Marius Kaminskas, and Francesco Ricci. 2011. A generic semantic-based framework for cross-domain recommendation. In *Proceedings of the 2nd International Workshop on Information Heterogeneity and Fusion in Recommender Systems (HetRec '11)*. ACM, New York, NY, USA, 25–32.

Yoav Freund, Raj Iyer, Robert E. Schapire, and Yoram Singer. 2003. An Efficient Boosting Algorithm for Combining Preferences. *J. Mach. Learn. Res.* 4 (Dec. 2003), 933–969.

Yoav Freund and Robert E. Schapire. 1995. A Decision-theoretic Generalization of On-line Learning and an Application to Boosting. In *Proceedings of the Second European Conference on Computational Learning Theory (EuroCOLT '95)*. Springer-Verlag, London, UK, UK, 23–37.

Jerome H. Friedman. 2000. Greedy Function Approximation: A Gradient Boosting Machine. *Annals of Statistics* 29 (2000), 1189–1232.

Aldo Gangemi. 2013. A Comparison of Knowledge Extraction Tools for the Semantic Web. In *The Semantic Web: Semantics and Big Data*, Philipp Cimiano, Oscar Corcho, Valentina Presutti, Laura Hollink, and Sebastian Rudolph (Eds.). Lecture Notes in Computer Science, Vol. 7882. Springer Berlin Heidelberg, 351–366. DOI:http://dx.doi.org/10.1007/978-3-642-38288-8_24

Zeno Gantner, Lucas Drumond, Christoph Freudenthaler, Steffen Rendle, and Lars Schmidt-Thieme. 2010. Learning Attribute-to-Feature Mappings for Cold-Start Recommendations. In *Proceedings of the 2010 IEEE International Conference on Data Mining (ICDM '10)*. IEEE Computer Society, Washington, DC, USA, 176–185.

Zeno Gantner, Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. 2011. MyMediaLite: a free recommender system library. In *Proceedings of the fifth ACM conference on Recommender systems (RecSys '11)*. ACM, New York, NY, USA, 305–308.

Benjamin Heitmann and Conor Hayes. 2010. Using Linked Data to Build Open, Collaborative Recommender Systems. In *AAAI Spring Symposium: Linked Data Meets AI*.

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 (RecSys '13)*. ACM, New York, NY, USA, 185–192.

Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix Factorization Techniques for Recommender Systems. *Computer* 42, 8 (Aug. 2009), 30–37.

Ni Lao and William W. Cohen. 2010. Relational retrieval using a combination of path-constrained random walks. *Machine Learning* 81, 1 (2010), 53–67.

Ni Lao, Tom Mitchell, and William W. Cohen. 2011. Random Walk Inference and Learning in a Large Scale Knowledge Base. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP '11)*. Association for Computational Linguistics, Stroudsburg, PA, USA, 529–539.

Sangkeun Lee, Sungchan Park, Minsuk Kahng, and Sang-goo Lee. 2012. PathRank: A Novel Node Ranking Measure on a Heterogeneous Graph for Recommender Systems. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management (CIKM '12)*. ACM, New York, NY, USA, 1637–1641. DOI:http://dx.doi.org/10.1145/2396761.2398488

Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N. Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick van Kleef, Sören Auer, and Christian Bizer. 2014. DBpedia - A Large-scale, Multilingual Knowledge Base Extracted from Wikipedia. *Semantic Web Journal* (2014).

Nathan N. Liu and Qiang Yang. 2008. EigenRank: a ranking-oriented approach to collaborative filtering. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval (SIGIR '08)*. 83–90.

Tie-Yan Liu. 2009. Learning to Rank for Information Retrieval. *Found. Trends Inf. Retr.* 3, 3 (March 2009), 225–331.

Tie Y. Liu, Jun Xu, Tao Qin, Wenying Xiong, and Hang Li. 2007. LETOR: Benchmark Dataset for Research on Learning to Rank for Information Retrieval. In *SIGIR '07: Proceedings of the Learning to Rank workshop in the 30th annual international ACM SIGIR conference on Research and development in information retrieval*.

Pasquale Lops, Marco de Gemmis, and Giovanni Semeraro. 2011. Content-based Recommender Systems: State of the Art and Trends. In *Recommender Systems Handbook*. 73–105.

Yucheng Low, Danny Bickson, Joseph Gonzalez, Carlos Guestrin, Aapo Kyrola, and Joseph M. Hellerstein. 2012. Distributed GraphLab: A Framework for Machine Learning and Data Mining in the Cloud. *Proc. VLDB Endow.* 5, 8 (April 2012), 716–727. DOI:http://dx.doi.org/10.14778/2212351.2212354

Donald Metzler and W. Bruce Croft. 2007. Linear Feature-based Models for Information Retrieval. *Inf. Retr.* 10, 3 (June 2007), 257–274.

Stuart E Middleton, David De Roure, and Nigel R Shadbolt. 2009. Ontology-Based Recommender Systems. *Handbook on Ontologies* 32, 6 (2009), 779–796.

Bamshad Mobasher, Xin Jin, and Yanzan Zhou. 2004. Semantically Enhanced Collaborative Filtering on the Web. In *Web Mining: From Web to Semantic Web*, Bettina Berendt, Andreas Hotho, Dunja Mladenič, Maarten Someren, Myra Spiliopoulou, and Gerd Stumme (Eds.). Lecture Notes in Computer Science, Vol. 3209. Springer Berlin Heidelberg, 57–76.

Cataldo Musto, Giovanni Semeraro, Pasquale Lops, and Marco de Gemmis. 2014. Combining Distributional Semantics and Entity Linking for Context-Aware Content-Based Recommendation. In *User Modeling, Adaptation, and Personalization - 22nd International Conference, UMAP 2014, Aalborg, Denmark, July 7-11, 2014. Proceedings*. 381–392.

Xia Ning and George Karypis. 2011. SLIM: Sparse Linear Methods for Top-N Recommender Systems. In *Proceedings of the 2011 IEEE 11th International Conference on Data Mining (ICDM '11)*. IEEE Computer Society, Washington, DC, USA, 497–506.

Xia Ning and George Karypis. 2012. Sparse linear methods with side information for top-n recommendations. In *Proceedings of the sixth ACM conference on Recommender systems (RecSys '12)*. ACM, New York, NY, USA, 155–162.

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 (RecSys '13)*. ACM, New York, NY, USA, 85–92.

Vito Claudio Ostuni, Tommaso Di Noia, Roberto Mirizzi, and Eugenio Di Sciascio. 2014. A Linked Data Recommender System using a Neighborhood-based Graph Kernel. In *The 15th International Conference on Electronic Commerce and Web Technologies (Lecture Notes in Business Information Processing)*. Springer-Verlag.

Vito Claudio Ostuni, Giosia Gentile, Tommaso Di Noia, Roberto Mirizzi, Davide Romito, and Eugenio Di Sciascio. 2013. Mobile Movie Recommendations with Linked Data. In *Human-Computer Interaction & Knowledge Discovery @ CD-ARES'13 (IFIP International Cross Domain Conference and Workshop on Availability, Reliability and Security, CD-ARES 2013)*. Springer.

Eric Prud'hommeaux and Andy Seaborne. 2008. SPARQL Query Language for RDF. `http://www.w3.org/TR/rdf-sparql-query`. (2008).

Steffen Rendle. 2010. Factorization Machines.. In *ICDM*. IEEE Computer Society, 995–1000. `http://dblp.uni-trier.de/db/conf/icdm/icdm2010.html#Rendle10`

Steffen Rendle. 2012. Factorization Machines with libFM. *ACM Trans. Intell. Syst. Technol.* 3, 3, Article 57 (May 2012), 22 pages.

Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian personalized ranking from implicit feedback. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence (UAI '09)*. AUAI Press, Arlington, Virginia, United States, 452–461.

Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor (Eds.). 2011. *Recommender Systems Handbook*. Springer.

Matthew Rowe. 2014a. SemanticSVD++: incorporating semantic taste evolution for predicting ratings. In *2014 IEEE/WIC/ACM International Conferences on Web Intelligence, WI 2014*.

Matthew Rowe. 2014b. Transferring semantic categories with vertex kernels: recommendations with SemanticSVD++. In *The Semantic Web - ISWC 2014 - 13th International Semantic Web Conference*.

Alan Said and Alejandro Bellogin. 2014. Comparative Recommender System Evaluation: Benchmarking Recommendation Frameworks. In *Proceedings of the 8th ACM Conference on Recommender Systems (RecSys '14)*. ACM, New York, NY, USA, 129–136. `DOI:http://dx.doi.org/10.1145/2645710.2645746`

Ruslan Salakhutdinov and Andriy Mnih. 2008. Bayesian Probabilistic Matrix Factorization Using Markov Chain Monte Carlo. In *Proceedings of the 25th International Conference on Machine Learning (ICML '08)*. ACM, New York, NY, USA, 880–887. `DOI:http://dx.doi.org/10.1145/1390156.1390267`

Giovanni Semeraro, Pasquale Lops, Pierpaolo Basile, and Marco de Gemmis. 2009. Knowledge infusion into content-based recommender systems. In *Proceedings of the third ACM conference on Recommender systems (RecSys '09)*. ACM, New York, NY, USA, 301–304.

Wei Shen, Jianyong Wang, and Jiawei Han. 2015. Entity Linking with a Knowledge Base: Issues, Techniques, and Solutions. *Knowledge and Data Engineering, IEEE Transactions on* 27, 2 (Feb 2015), 443–460.

Yue Shi, Alexandros Karatzoglou, Linas Baltrunas, Martha Larson, Nuria Oliver, and Alan Hanjalic. 2012. CLiMF: learning to maximize reciprocal rank with collaborative less-is-more filtering. In *Proceedings of the sixth ACM conference on Recommender systems (RecSys '12)*. ACM, New York, NY, USA, 139–146.

Mark D. Smucker, James Allan, and Ben Carterette. 2007. A Comparison of Statistical Significance Tests for Information Retrieval Evaluation. In *Proceedings of the Sixteenth ACM Conference on Conference on Information and Knowledge Management (CIKM '07)*. ACM, New York, NY, USA, 623–632. `DOI:http://dx.doi.org/10.1145/1321440.1321528`

Harald Steck. 2013. Evaluation of Recommendations: Rating-prediction and Ranking. In *Proceedings of the 7th ACM Conference on Recommender Systems (RecSys '13)*. ACM, New York, NY, USA, 213–220.

Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. 2007. Yago: A Core of Semantic Knowledge. In *Proceedings of the 16th International Conference on World Wide Web (WWW '07)*. ACM, New York, NY, USA, 697–706.

Jiankai Sun, Shuaiqiang Wang, Byron J. Gao, and Jun Ma. 2012. Learning to rank for hybrid recommendation. In *Proceedings of the 21st ACM international conference on Information and knowledge management (CIKM '12)*. ACM, New York, NY, USA, 2239–2242.

Maksims Volkovs and Richard S. Zemel. 2012. Collaborative Ranking With 17 Parameters. In *Advances in Neural Information Processing Systems 25*, P. Bartlett, F.c.n. Pereira, C.j.c. Burges, L. Bottou, and K.q. Weinberger (Eds.). 2303–2311.

Ellen M. Voorhees. 2004. Measuring Ineffectiveness. In *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '04)*. ACM, New York, NY, USA, 562–563.

Markus Weimer, Alexandros Karatzoglou, Quoc V. Le, and Alex J. Smola. 2007. COFI RANK - Maximum Margin Matrix Factorization for Collaborative Ranking. In *Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems*.

Markus Weimer, Alexandros Karatzoglou, and Alex J. Smola. 2008. Improving Maximum Margin Matrix Factorization. In *Machine Learning and Knowledge Discovery in Databases, European Conference, ECML/PKDD 2008, Antwerp, Belgium, September 15-19, 2008, Proceedings, Part I (Lecture Notes in Computer Science)*, Walter Daelemans, Bart Goethals, and Katharina Morik (Eds.), Vol. 5211. Springer, 14.

Qiang Wu, Christopher J. Burges, Krysta M. Svore, and Jianfeng Gao. 2010. Adapting Boosting for Information Retrieval Measures. *Inf. Retr.* 13, 3 (June 2010), 254–270.

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 (SIGIR '07)*. ACM, New York, NY, USA, 391–398.

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 (WSDM '14)*. ACM, New York, NY, USA, 283–292.

Cai-Nicolas Ziegler, Georg Lausen, and Lars Schmidt-Thieme. 2004. Taxonomy-driven computation of product recommendations. In *Proceedings of the thirteenth ACM international conference on Information and knowledge management (CIKM '04)*. ACM, New York, NY, USA, 406–415.

A. BACKGROUND TECHNOLOGIES

In this appendix we briefly recap the basic notions of two Semantic Web standardized technologies at the base of Linked Data, i.e., RDF [Eric Miller 2004] and SPARQL [Prud'hommeaux and Seaborne 2008] and we provide a quick description of DBpedia, the most relevant dataset available in the LOD cloud.

A.1. Resource Description Framework - RDF

The Resource Description Framework (RDF) is a general model for describing information about resources on the Web. It has been developed by the World Wide Web Consortium (W3C) in 1998 as the building block for the Semantic Web. It allows to represent Web entities and their relations as well as to attach to them a machine understandable and processable meaning (semantics) that can be further exploited to perform automatic reasoning tasks able to infer new knowledge from the explicitly stated one. Thanks to RDF, resources are made available on the Web, enabling applications to exploit them by taking into account their meaning. Each statement about resources is modeled in the form of a triple: *subject-predicate-object*. *Subjects* and *predicates* are represented by URIs, while *objects* can be identified either by URIs or by literals (data values). As an example, the two following triples are valid RDF statements about the movie *Pulp Fiction*:

```
<http://dbpedia.org/resource/Pulp_Fiction>
   <http://www.w3.org/2000/01/rdf-schema#label>
      "Pulp Fiction"@en

<http://dbpedia.org/resource/Pulp_Fiction>
   <http://dbpedia.org/ontology/director>
      <http://dbpedia.org/resource/Quentin_Tarantino>
```

where it is stated that we may refer to the the Pulp Fiction resource with the string "Pulp Fiction" and that it was directed by Quentin Tarantino. RDF information representation can be formally modeled through a labeled directed graph. In fact, if we consider all the RDF statements (triples) as a whole, what we get is a graph, where nodes are resources connected to other resources or to literal values through predicates (the edges of the graph).
RDF can be serialized by means of different syntaxes. The most compact is the so called turtle syntax that allows us to use prefixes to shorten the URIs and represent them ad CURIEs. The turtle version of the two triples above is:

```
@prefix dbpedia: <http://dbpedia.org/resource/>
@prefix dbpedia-owl: <http://dbpedia.org/ontology/>
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>

dbpedia:Pulp_Fiction  rdfs:label  "Pulp Fiction"@en .
dbpedia:Pulp_Fiction  dbpedia-owl:director  dbpedia:Quentin_Tarantino .
```

From an ontological point of view, an interesting built-in RDF predicate is http://www.w3.org/1999/02/22-rdf-syntax-ns#type. It states that a resource is an instance of a class.

```
@prefix dbpedia: <http://dbpedia.org/resource/>
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
@prefix dbpedia-owl: <http://dbpedia.org/ontology/>
```

```
dbpedia:Pulp_Fiction  rdf:type  dbpedia-owl:Film .
```

The previous triple asserts that *Pulp Fiction* is an instance of the class *Film*.

DBpedia. Among the RDF datasets available in the LOD cloud, DBpedia is one of the main projects. It is an effort to extract structured information from Wikipedia and make it freely accessible as RDF triples. This knowledge base currently (release 3.9) describes 4 million resources, out of which 3.22 million are classified in a consistent ontology[26]. Labels and abstracts of DBpedia resources are stored in up to 97 languages. In addition, it is highly connected to other RDF datasets of the Linked Open Data cloud thus making DBpedia a cornerstone for the entire LOD project. Compared to other hierarchies and taxonomies, DBpedia has the advantage that each entity/resource is endowed with a rich description including text-based abstracts. Another advantage compared to static hierarchies is that DBpedia evolves as Wikipedia changes. Hence, problems such as domain coverage, content freshness, machine-understandability can be addressed more easily when considering DBpedia.
Each LOD dataset, including DBpedia, can be queried by means of its SPARQL endpoint. For DBpedia, it allows anyone to ask complex queries about any topic available in Wikipedia.
Noteworthy is that most DBpedia URIs have owl:sameAs links to other resources in other LOD knowledge bases[27]. Linking pieces of data across different datasets distributed on the Web is the main aim of the Linked Data initiative. Hence, we could also leverage such links for merging the data extracted from DBpedia with other data from Freebase, LinkedMDB or Yago for example as we show at the end of the next section.

A.2. Simple Protocol and RDF Query Language - SPARQL

SPARQL is the de facto query language for RDF datasets. The language reflects the graph-based nature of the underlying data model. Indeed, graph-matching is its query mechanism. A basic SPARQL query is of the form:

```
PREFIX dbpedia: <http://dbpedia.org/resource/>
PREFIX dcterms: <http://purl.org/dc/terms/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT ?c ?l
WHERE {
   <http://dbpedia.org/resource/Pulp_Fiction>  dcterms:subject  ?c .
   ?c  rdfs:label ?l.
}
```

where we ask for all possible values that can be assigned to the variables ?l and ?c in order to match the graph pattern expressed in the WHERE clause. The graph pattern is represented as a set of triples where, usually, at least one of the three elements is a variable. In the previous example we have ?c as a variable for the first triple and ?c and ?l for the second one. The syntax to represent the triples follows the same rules as for RDF. SPARQL has other syntactic elements in common with RDF, such as the prefix declaration. Indeed, URIs can be represented either explicitly or in their CURIE form. Analogously to SQL, SPARQL offers different aggregation functions such as COUNT, SUM, MIN, MAX, AVG, GROUP CONCAT, SAMPLE, nested queries, negation, etc..

[26] http://wiki.dbpedia.org/Ontology
[27] The owl:sameAs property is the ontological property specifically used to link a resource in a specific knowledge base to the equivalent resources in other knowledge bases of the LOD cloud.

An interesting feature of SPARQL is the availability of filtering constraints in the representation of a graph pattern. As a way of example, the previous query can be modified by filtering only labels represented in English.

```
PREFIX dbpedia: <http://dbpedia.org/resource/>
PREFIX dcterms: <http://purl.org/dc/terms/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT ?c ?l
WHERE {
   <http://dbpedia.org/resource/Pulp_Fiction>  dcterms:subject  ?c .
   ?c  rdfs:label ?l.
   FILTER (langMatches(lang(?etichetta), "en"))
}
```

Other filtering constraints can be defined by comparing the value a variable is bound to, e.g., `FILTER (?var1 != ?var2)` or by applying string functions as in `FILTER (regex(?label,"ne.","i"))` where we filter the labels matching the regular expression `ne.` regardless of the letter case (case insensitive). Besides the SELECT query form, SPARQL allows the user to pose other kind of queries via DESCRIBE, ASK and CONSTRUCT. By DESCRIBE the system returns all the triples involving a particular entity. For instance, if we want to retrieve all the triples related to actors starring in both The Matrix Revolutions and Memento:

```
PREFIX dbpedia: <http://dbpedia.org/resource/>
PREFIX dbpedia-owl: <http://dbpedia.org/ontology/>

DESCRIBE ?actor{
    dbpedia:The_Matrix_Revolutions dbpedia-owl:starring ?actor .
    <http://dbpedia.org/resource/Memento_(film)>
                                   dbpedia-owl:starring ?actor .
}
```

The ASK query form returns a Boolean answer stating if the requested sub-graph is represented within the dataset.

```
PREFIX dbpedia: <http://dbpedia.org/resource/>
PREFIX dbpedia-owl: <http://dbpedia.org/ontology/>

ASK{
    dbpedia:The_Matrix_Revolutions dbpedia-owl:starring
                                     dbpedia:Carrie-Anne_Moss .
}
```

Finally, we may use the CONSTRUCT query form to return a set of triples built on-the-fly starting from the results of the pattern matching.

```
PREFIX dbpedia: <http://dbpedia.org/resource/>
PREFIX dbpedia-owl: <http://dbpedia.org/ontology/>
PREFIX dbpprop: <http://dbpedia.org/property/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX vcard: <http://www.w3.org/2006/vcard/ns#>

CONSTRUCT{
    ?actor1 vcard:bday ?date .
```

```
    ?actor1 foaf:knows ?actor2 .
}
WHERE{
    dbpedia:The_Matrix_Revolutions dbpedia-owl:starring ?actor1 .
    ?actor1 dbpprop:birthDate ?date .
    dbpedia:The_Matrix_Revolutions dbpedia-owl:starring ?actor2 .
    FILTER(?actor1 != ?actor2) .
}
```

With the previous query we build new triples involving the VCARD[28] and the FOAF[29] ontologies starting from the data in DBpedia.

Starting for the version 1.1 of the language, SPARQL makes possible to execute federated queries through different remote endpoints. Via the SERVICE keyword, the user is allowed to explicitly state the endpoints to be queried. As an example, via the following query, we may ask to build a semantic graph containing triples containing joint information coming from DBpedia and Yago[30] datasets in the domain of books:

```
PREFIX yago: <http://yago-knowledge.org/resource/>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX dbpedia-owl: <http://dbpedia.org/ontology/>

CONSTRUCT{
   ?book ?p ?o .
   ?book yago:linksTo ?yagolink .
}
WHERE{
    SERVICE <http://live.dbpedia.org/sparql> {
       ?book rdf:type dbpedia-owl:Book .
       ?book ?p ?o .
       ?book owl:sameAs ?yago .
       FILTER(regex(str(?yago),"http://yago-knowledge.org/resource/")) .
    }
    SERVICE <http://lod2.openlinksw.com/sparql> {
       ?yago yago:linksTo ?yagolink .
    }
}
```

Here we ask DBpedia for all the triples relates to entities of type Book and for each of them, if there is a owl:sameAs link with the Yago dataset, we look for the triples involving the linksTo property of the corresponding Yago resource. Once we get the resources bound to the variable ?yagolink we connect them to the initial DBpedia entity.

[28] http://www.w3.org/TR/vcard-rdf/
[29] http://xmlns.com/foaf/spec/
[30] www.mpi-inf.mpg.de/yago/