Top-N Recommendations from Implicit Feedback leveraging Linked Open Data

Vito Claudio Ostuni, Tommaso Di Noia, Eugenio Di Sciascio, Roberto Mirizzi
Polytechnic University of Bari – Via Orabona, 4 – 70125 Bari, Italy
{ostuni,mirizzi}@deemail.poliba.it, {t.dinoia,disciascio}@poliba.it

ABSTRACT

The advent of the `Linked Open Data` (`LOD`) initiative gave birth to a variety of open knowledge bases freely accessible on the Web. They provide a valuable source of information that can improve conventional recommender systems, if properly exploited. In this paper we present `SPrank`, a novel hybrid recommendation algorithm able to compute *top-N* item recommendations from implicit feedback exploiting the information available in the so called Web of Data. We leverage `DBpedia`, a well-known knowledge base in the `LOD` compass, to extract semantic *path-based* features and to eventually compute recommendations using a *learning to rank* algorithm.
Experiments with datasets on two different domains show that the proposed approach outperforms in terms of prediction accuracy several state-of-the-art *top-N* recommendation algorithms for implicit feedback in situations affected by different degrees of data sparsity.

Categories and Subject Descriptors

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

Keywords

Hybrid Recommender System; Top-N Recommendations; Implicit Feedback; Learning to Rank; Linked Data; DBpedia

1. INTRODUCTION

Information overload in the current Web challenges users in their decision-making tasks. Recommender systems have become essential tools in assisting users to find what is relevant for them in increasingly complex information spaces. The *Linking Open Data* project [3] started as a community effort in 2007, and helped to produce billions of `RDF`[1] 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 improve the quality of conventional recommender systems. Although recommender systems leverage well established technologies and tools, new challenges arise when they exploit the huge amount of interlinked data coming from the Web of Data.
In the past, several works on *ontological recommender systems* have been proposed. These works introduced the usage of ontologies and semantic technologies to boost collaborative filtering (CF) systems [2, 5, 16] or to build smarter content-based (CB) systems [21]. In particular they have been shown to be very effective in solving some drawbacks of collaborative methods such as cold start and data sparsity. Most of the works on ontological recommender systems are related to the rating prediction task. However, in the last few years a significant interest has grown towards ranking-oriented approaches as they seem to better approximate the *top-N* recommendation task [7, 23]. Differently from rating prediction, in *top-N* recommendations the goal of the system is to find a few specific items which are supposed to be the most appealing to the user instead of accurately predict all the missing ratings.
Together with the *top-N* recommendation problem, great attention has been gained by implicit feedback scenarios and many recent works have addressed both these issues [20, 22]. Even though the case of explicit feedback where users express their preferences through ratings is the most known in the literature, the implicit feedback case has attracted increasing interest because in many real-world situations explicit ratings are not available and implicit feedback requires no extra feedback on the user side.

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 from implicit feedback that effectively incorporates ontological knowledge belonging to the Web of Data with collaborative user preferences in a graph-based setting. While several works have been proposed in the last recent years to address *top-N* recommendations and implicit feedback in the literature of collaborative filtering, for ontological recommender systems

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

RecSys'13, October 12–16, 2013, Hong Kong, China.
Copyright 2013 ACM 978-1-4503-2409-0/13/09 ...$15.00
http://dx.doi.org/10.1145/2507157.2507172.

these issues mostly have not been investigated yet. In 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 and use a *learning to rank* algorithm for computing the *top-N* recommendations as a ranking problem. We remark that previous work on semantic-aware and ontological recommender systems neither addresses the *top-N* item recommendation task nor deals with implicit feedback datasets. Here we show how SPrank is able to compute accurate recommendations in scenarios where collaborative filtering algorithms notoriously are not very effective, such as the ones affected by data sparsity. To the best of our knowledge, this is the first work proposed to address the *top-N* recommendation task as a ranking problem from implicit feedback by leveraging the Web Of Data. Main contributions of this paper are:

- combination of semantic item descriptions from the Web of Data and implicit feedback for the *top-N* recommendation task;
- formulation of a hybrid recommendation problem in a learning to rank setting;
- mining of the semantic graph of LOD datasets through path-based features to capture complex and not trivial relationships between items;
- evaluation of the proposed approach in terms of accuracy of the *top-N* recommendations on real data from MovieLens and Last.fm.

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 closes the paper.

2. RELATED WORK

Several approaches have been proposed to incorporate ontological knowledge in recommender systems. In the following we review some of them and more recent literature on LOD-based approaches. In addition, since we propose an ontological *top-N* recommendation algorithm for implicit feedback, we present related work in the same area.

Ontological RSs. In [15] *Quickstep* and *Foxtrot* are presented, two ontological recommender systems that make use of semantic user profiles to compute collaborative recommendations. A hybrid recommendation system is proposed in [5] where user preferences and item features are described by semantic concepts to obtain users' clusters corresponding to implicit *Communities of Interest.* In [16] the authors introduce the so called *semantically enhanced collaborative filtering* where structured semantic knowledge about items is used in conjunction with user-item ratings to create a combined similarity measure for item comparisons. In [2] an approach is presented that integrates user rating vectors with an item ontology. In all of these works, the experiments prove an accuracy improvement over collaborative approaches especially in presence of sparse datasets.

LOD-based RSs. Most of the works described so far have been produced before the LOD initiative was officially launched. In [8, 9] a model-based approach and a memory-based one to compute content-based recommendations are presented leveraging LOD datasets. In [10] the authors present a knowledge-based framework leveraging DBpedia for computing cross-domain recommendations.

Top-N RSs with implicit feedback. The rating prediction task is the main focus of all the ontological approaches presented so far. Instead, referring to the most recent *top-N* recommendation problem in the context of collaborative filtering, several works have been proposed in the last few years. *SLIM* [17] adopts a *Sparse Linear* method for learning a sparse aggregation coefficient matrix that is used for computing *top-N* recommendations. In [18] 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.

Other works represent the *top-N* recommendation task as a ranking problem using learning to rank. In [22] *CLiMF* the authors present a collaborative-filtering algorithm able to directly optimize the Mean Reciprocal Rank. The authors of [20] propose a *Bayesian Personalized Ranking* (BPR) criterion for optimizing a ranking loss. Following this, in [12] a hybrid extension of BPR is presented 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. All these approaches deal with binary relevance data or only positive implicit feedback.

3. SPRANK: SEMANTIC PATH-BASED RANKING

The main idea behind SPrank is exploring *paths* in a *semantic* graph in order to find items that are related to the ones the user is interested in. From the analysis of these paths we extract *path-based* features and apply a learning to rank algorithm for getting a ranking function able to recommend the most relevant items to the user. However, applying learning to rank to hybrid recommender systems is not as straightforward as in Information Retrieval for document ranking since we need to conciliate content-based and collaborative features in the same feature space. In the following we detail the data model and the formulation of the recommendation problem, the path-based feature extraction and the algorithm for learning the ranking function.

3.1 Data Model

The standard language to semantically describe resources in the Web of Data – and in the Semantic Web – is RDF. The RDF data model is a labeled directed graph where nodes correspond to entities and edges are properties connecting them. The semantics of such properties is explicitly modeled by means of an ontological schema represented in RDFS[3] or OWL[4]. Indeed, LOD datasets can be seen as semantic graphs where the knowledge related to an entity, e.g., a movie, a song or an artist, is encoded by linking different nodes/entities with each other via semantic-enabled edges. Semantic datasets can be used as the input for content-based recommender systems. In fact, given a node representing an item, we can use the knowledge associated to the corresponding entity to discover similar items available in the graph.

Also the data model behind collaborative filtering problems

[2]http://dbpedia.org

[3]http://www.w3.org/TR/rdf-schema/

[4]http://www.w3.org/TR/owl2-overview/

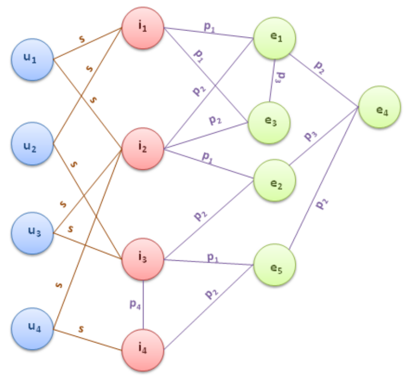

Figure 1: Graph-based representation of the data model.

can be seen as a graph as well – in this case a bipartite graph – where users and items are the nodes and users' feedback are the links.
The common graph-based nature of the data models for the two problems – content-based exploiting `LOD` and collaborative filtering – suggests interesting ways to model a hybrid recommendation engine. We can merge these two graphs obtaining a new graph $G = (V, R)$ as shown in Figure 1, where V denotes the set of vertices and R the set of relationships. 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: $I \subseteq E$ and $V = U \cup E$. Similarly, R contains two categories of relationships. In fact, we have $R = S \cup P$ where $S = U \times I$ and $P \subseteq E \times E$. More precisely, an edge $s \in S$ links a user $u \in U$ to his/her relevant items $i \in I$ while an edge $p \in P$ 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 node belonging to U, I, E and V, respectively. Analogously, we will denote with s, p and r the edges in S, P and R. Thanks to this graph-based formulation of the problem we consider both collaborative and content aspects in a unified representation and hence a unified feature space. The purpose is recommending relevant items i to users u leveraging the knowledge encoded in the graph G. In the rest of the paper, in our data model we will always consider G as undirected.

3.2 Problem Formulation

Following the notation introduced by [20] for implicit feedback scenarios, let $\hat{S}$ be the matrix of implicit feedback, where $\hat{s}_{ui} = 1$ if item i is relevant for user u (i.e., there is an edge of type s between u and i), 0 otherwise. Looking at the graph in Figure 1, for user u_1 we have $\hat{s}_{u_1 i_1} = 1$, $\hat{s}_{u_1 i_2} = 1$ while $\hat{s}_{u_1 i_3} = 0$, $\hat{s}_{u_1 i_4} = 0$.
Starting from $\hat{S}$ we define $I_u^+ = \{i \in I | \hat{s}_{ui} = 1\}$ as the set of relevant items for u and $I_u^- = \{i \in I | \hat{s}_{ui} = 0\}$ as the set of unknown items for u. We call I_u^+ the *user profile* of u. In Figure 1, with reference to user u_1, we have $I_{u_1}^+ = \{i_1, i_2\}$ and $I_{u_1}^- = \{i_3, i_4\}$.
The main assumptions behind our approach are the following two: *(I)* only binary data of the user's interactions can be observed through implicit feedback such as purchases, friendships, clicks, etc.; *(II)* most of the items in I_u^- (items with no observed interactions) are irrelevant, but some of them could actually be relevant. The unobserved items are exactly the items that have to be ranked. The ultimate goal of the system is to rank in the *top-N* positions items likely to be relevant for the user.
We formulate the problem of computing the *top-N* recommendations in a learning to rank fashion similar to the document ranking problem adopted in Web search [14]. In particular we adopt a regression based point-wise method.
For each user-item pair (u, i) , we encode the features able to characterize the interaction between user u and item i in the 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 score between user u and item i with respect to a specific feature[5]. For each user u we assume to have information on the set of relevant items I_u^+, the set of unknown items I_u^-, the implicit binary feedback $\hat{s}_{ui}$ and the feature vector $\mathbf{x_{ui}}$. Finally, we introduce $I_u^{-*} \subseteq I_u^-$ computed by sampling, with a uniform probability distribution, a fixed number of unobserved items from I_u^- equal to K times the size of I_u^+, being K a constant. In other words, I_u^{-*} is the set of the $K \cdot \mid I_u^+ \mid$ uniformly sampled items from I_u^-. In Section 4.3.1 we will motivate the choice of K by means of experimental results.
Now we have all the elements to formally define the training set TR as:

$$TR = \bigcup_u \{\langle \mathbf{x_{ui}}, \hat{s}_{ui} \rangle | i \in (I_u^+ \cup I_u^{-*})\}$$

Given the training set TR, computing the *top-N* recommendations for user u can be formulated as the task of generating a ranking function $f : \mathbb{R}^D \to \mathbb{R}$ such that $f(\mathbf{x_{ui}}) \approx \hat{s}_{ui}$. Eventually, we use $f(\cdot)$ to rank the items in I_u^- and getting the *top-N* recommendation list for u.

3.3 Path-based features

Given a graph G as the one represented in Figure 1, we want to extract features able to characterize the interactions between users, items and entities capturing the complex relationships between them. The aim is to recommend items exploiting their underlying properties and attributes expressed in the semantic graph. The basic idea is to consider all the paths that connect the user to an item in order to have a relevance score for that item. The more paths between user and item, the more the item is relevant for the user. However, in this formulation of the problem there are several types of paths and not all of them have the same relevance. Moreover, some paths that involve useless properties can be noisy for the purpose of recommendation. Based on these assumptions, we leverage a supervised approach and we delegate to the learning to rank algorithm the task of finding what paths are most relevant for computing *top-N* recommendations. In the previous section we introduced the feature vector $\mathbf{x_{ui}}$ encoding the interest of the user u in the item i. In the following we detail how the vector values are computed.
Given an undirected graph G as defined in Section 3.1, we define a path as the acylic sequence of edges of the form $(s, \dots, r_l, \dots, r_L)$ where $s = (u, i)$ and $r_L = (v, i')$ with

[5]In `SPrank` we rely on *path-based* features as we will detail in Section 3.3.

$i \neq i'$. We also define the *length of a path* as the number of edges contained within such path. We consider paths having length greater than 1 and less than or equal than a given L. We collect all the possible paths in G to build a $Path$ index. $Path(j)$ represents the j-th component in the index and it corresponds to a specific sequence of edge labels (i.e., to a path disregarding the actual nodes). Considering a user-item pair (u, i), we denote $\#path_{ui}(j)$ as the number of paths 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. This aggregate information corresponds to the frequency of $Path(j)$ in the sub-graph composed by all paths between u and i. We are now ready to define the path-based features. We define the j-th component in the feature vector $\mathbf{x_{ui}}$ as:

$$\mathbf{x_{ui}}(j) = \frac{\#path_{ui}(j)}{\sum_{d=1}^{D} \#path_{ui}(d)} \qquad (1)$$

Equation (1) represents the importance of the specific sequence of edge labels $Path(j)$ between u and i in the sub-graph constituted by all the associations between these two nodes. Specifically, it is the frequency of the specific path $Path(j)$ normalized with respect to the frequencies of all the existing paths between u and i.
In order to clarify how to compute the feature vector $\mathbf{x_{ui}}$ we show an example with reference to the graph in Figure 1. For $L = 4$ we have the following paths: (s, s, s), (s, p_1, p_2), (s, p_2, p_1), (s, p_1, p_3, p_2), (s, p_2, p_3, p_1), (s, p_4, s, s), (s, p_4), (s, s, s, p_4), (s, p_1, p_2, p_4), (s, p_4, p_2, p_1). We call them, respectively: $path_1, \cdots, path_{10}$ and build the index $Path = \{path_1, \ldots, path_{10}\}$ such that $Path(1)$ corresponds to $path_1$ and so on. Between user u_3 and item i_1, there are only paths $Path(1)$, $Path(3)$ and $Path(5)$. In particular, $\#path_{u_3 i_1}(j) = 0$ for $j = 2, 4, 6, 7, 8, 9, 10$, and $\#path_{u_3 i_1}(1) = 2$, $\#path_{u_3 i_1}(3) = 2$ and $\#path_{u_3 i_1}(5) = 1$. We can now compute the feature vector $\mathbf{x_{u_3 i_1}}$ for the pair (u_3, i_1). Following Equation (1), at the denominator we have the sum of the number of all the paths connecting u_3 and i_1. This sum is equal to 5. Then, $\mathbf{x_{u_3 i_1}} = (2/5, 0, 2/5, 0, 1/5, 0, 0, 0, 0, 0)$.
Path types. Depending on the type of links composing a path there are different types of paths. In particular, such paths can be: (I) *collaborative* if only links in S are involved as for (s, s, s); (II) *content-based* if there are only $r_l \notin S$ with $l = 2, \ldots, L$ as for (s, p_1, p_2) or (s, p_4, p_2, p_1); (III) *hybrid* if there is more than one link in S and at least one link in P as for (s, p_4, s, s) or (s, s, s, p_4). In Figure 2 we represent several examples of path types, referring to the music domain. In particular, musical artists are items (red nodes) to be recommended to users (blue nodes). The green nodes are the entities, coming from `DBpedia`. Artists and entities are connected via `DBpedia` properties. In these paths there are user feedback edges (`likes`) and content-based `RDF` properties (`dcterm:subject`, `skos:broader`, `dbprop:writer`, `dbprop:title`). In Figure 2, the paths connect the user `Andrew` to the artist `Craig David`[6] through the singer `Adele`[7], liked by `Andrew`. Given that `Andrew` has shown interest for `Adele`, the task of the recommender system is to find other artists that are related to `Adele`. In this case we want to know if `Craig David` is a good candidate to be recommended to `Andrew`. A possible way for doing this is by exploiting the tastes of the crowd to find if users who like `Adele` also like `Craig David`. The path *(a)* of Figure 2 is an example of collaborative path because it shows that there is another user (`Alex`) who likes both `Adele` and `Craig David`. We can also have longer chains of people with similar tastes, as shown in path *(b)*. Another way of recommending artists to `Andrew` is to find singers with similar characteristics to `Adele` (e.g., the genre of her songs, the prizes she won). These are content-based paths, as represented in Figure 2 by the paths *(c)*, *(d)*, *(e)*. We note that the usage of a `LOD` knowledge base such as `DBpedia` gives us the advantage of leveraging a valuable source of structured information. In fact, we can catch fine grained and not obvious relationships that in a non-ontological system we would otherwise miss. The last type of path is the hybrid one and it is shown in Figure 2 by path *(f)*. In this case both collaborative and content-based information is combined.

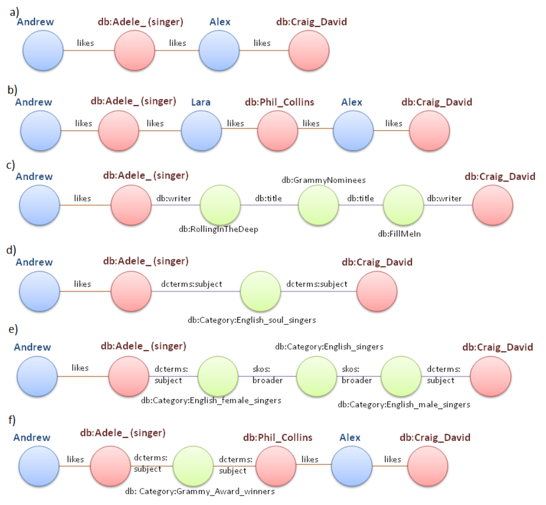

Figure 2: Various examples of content, collaborative and hybrid paths.

3.4 Learning the ranking function

In order to predict the ranking and form the *top-N* recommendation lists we deal with the learning to rank problem by adopting a *point-wise* approach. Point-wise methods have shown to be very effective in Web search ranking [14]. There are two point-wise algorithms that have proven particularly successful: *Random Forests* [4] and *Gradient Boosted Regression Trees* [11]. In fact, all of the top placed teams in the recent *Yahoo! Learning to Rank Challenge* used variants of these two algorithms coping with regression trees [6].
The fundamental concept underlying Random Forests is *bagging*. 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. Random Forest is essentially bagging applied to *Classification And Regression Trees* (*CART*) with full depth, where at each split only a subset of features is uniformly chosen

[6]`http://dbpedia.org/resource/Craig_David`
[7]`http://dbpedia.org/resource/Adele_(singer)`

and evaluated to find the best splitting point.
Similar to Random Forests, *Gradient Boosted Regression Trees* (*GBRT*) is a state-of-the-art regression model, which is a combination of regression trees constructed using the boosting approach. Instead of training many full high variance trees and average them to avoid overfitting, GBRT 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.
We formulate the learning to rank problem as a combination of both Random Forest and GBRT following the idea of *BagBoo* introduced by [19]. BagBoo combines the high accuracy of gradient boosting with resistance to overfitting and variance reduction of random forests. The basic idea is replacing simple tree models that are at the base of random forests with powerful and accurate gradient boosted trees. For a better understanding of BagBoo let us discuss the pseudo-code listed in Algorithm 1. The input of the learning algorithm is the training data TR as defined in Section 3.2, on which learning the ranking function. Q, T, B are the parameters of the algorithm. Q specifies the number of features to be considered for each node in order to find the best split in the CART used in GBRT. T is the number of bags used in Random Forests and B is the number of iterations required by GBRT. The final model f returned by the algorithm is basically a Random forests of $T \cdot B$ trees. We have T GBRTs each one composed by B regression trees. These T GBRTs are learned in the for-loop at line 1. At line 2 TR_t is sub-sampled with replacement from TR. At line 3 the f_t GBRT is learned. At the end all the GBRTs learned in the loop are combined in the forest to form our final ranking function f.

Algorithm 1: BagBoo

```
  Input: training set $TR$
  Output: ranking function $f$
  /* Q is the number of uniformly chosen features, T is
     the number of bags, B is the number of boosting
     iterations                                        */
1 for $t = 1$ to $T$ do
      /* Sample with replacement                       */
2     set $TR_t \subseteq TR$;
      /* Build GBRT with $TR_t$ using $Q$ randomly chosen
         features and $B$ boosting iterations          */
3     $f_t \leftarrow GBRT(TR_t, Q, B)$;
4 end
5 $f = 1/T \sum_{t=1}^{T} f_t$;
```

4. EXPERIMENTAL EVALUATION

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

4.1 Datasets

The first dataset is a subset of the MovieLens 1M dataset[8]. The original dataset contains 1,000,209 ratings for 3,883 movies by 6,040 users. Since our recommendation approach is based on positive implicit feedback, we chose 5-star ratings as relevant for the user and we ignored all the other ratings. Hence, for each user u we set $\hat{s}_{ui} = 1$ iff $rating_{ui} = 5$. In this way we can reasonably assert that I_u^+ contains only items relevant to u [7]. The second dataset comes from recent initiatives on information heterogeneity and fusion in recommender systems[9] [1]. In particular, we used the dataset collected from the Last.fm music system[10]. This is an implicit feedback dataset consisting of user-artist listening data, indicating the frequency a user listened to an artist's song. 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. In this case we chose the users' average listening count as threshold to identify the relevant artists for each user. Then, for each user u we set $\hat{s}_{ui} = 1$ iff $count_{ui} > avg_u$, with the obvious meaning for $count_{ui}$ and avg_u. For the purpose of our experiments in both datasets we removed users with less than twenty relevant items. In the following we will denote the two datasets MovieLens and Last.fm with ML and LF, respectively. Being our approach based on LOD knowledge bases, we need to map items in ML and LF to resources in LOD. In this work, the LOD knowledge base we leverage is DBpedia. It is one of the main projects in the Linked Open Data cloud. It currently describes more than 3.77 million things and all this information is stored in RDF triples. We can extract sub-graphs related to the movie and the music domain by querying its SPARQL endpoint[11]. In the current 3.8 release there are about 71,715 movies and 34,186 artists. In the following we detail the procedure we used to map items in ML and LF to the correspondent DBpedia URIs and extract the semantic sub-graphs of interest.

4.1.1 *Mapping and Linking datasets with DBpedia*

As the initial step for the mapping, we extracted from DBpedia the title and the year of production for all the movies in ML, by the following SPARQL query:

```
SELECT DISTINCT ?movie ?title ?year WHERE {
  ?movie rdf:type dbpedia-owl:Film.
  ?movie rdfs:label ?title.
  ?movie dcterms:subject ?cat .
  ?cat rdfs:label ?year .
  FILTER langMatches(lang(?title), "EN") .
  FILTER regex(?year, "^[0-9]{4} film", "i")
}
```

We used a similar query to extract music artists from DBpedia. Then, we performed a one-to-one mapping between the obtained sets of labels and the name of the movies and artists respectively in ML and LF by using the Levenshtein distance and checking also the year of production in case of ML. For ML we found a positive correspondence for 3,148 out of a total of 3,883 movies. For LF we found a match for 9,490 out of a total of 17,632 artists. The dump of the mappings is available here[12]. At the end of this mapping phase we removed the unmapped items from the two datasets and we replaced the item IDs in the two datasets with the corresponding DBpedia URIs.

[8]http://www.grouplens.org/node/73

[9]http://ir.ii.uam.es/hetrec2011/datasets.html

[10]http://www.lastfm.com

[11]http://dbpedia.org/sparql

[12]http://sisinflab.poliba.it/mappingdatasets2dbpedia.zip

4.1.2 *Entity extraction from DBpedia*

For each URI we extracted from DBpedia all the RDF triples containing it. For example the following SPARQL query returns all the triples corresponding to the songs written by Adele:

```
SELECT ?s WHERE
{ ?s dbpedia-owl:writer dbpedia:Adele_(singer). }
```

An example of a returned triple is:
`dbpedia:Chasing_Pavements dbpedia-owl:writer dbpedia:Adele_(singer).`
This triple states that Chasing Pavements has been written by Adele. At the end of this step we have an initial version of our graph of interest, consisting of entities directly linked to items. Starting from this set of entities we further query DBpedia to find other entities linked to them. Iteratively, we obtain a graph of increasing size incrementing the distance from the original items. In the queries, we consider the RDF properties belonging to the DBpedia Ontology[13] plus two more properties: dcterms:subject and skos:broader. The former relates a resource to its categories. They are used in Wikipedia to organize the entire project, and to help users to give a structure to the knowledge base, by grouping together pages on the same subject. The latter models the hierarchical structure of the categories. An example of these two properties is the path *(e)* in Figure 2. 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 class is available at http://mappings.dbpedia.org/server/ontology/classes/MusicalArtist. At the end of the process, the final graph for ML contains 3,792 users, 2,795 movies and 104,351 entities (actors, directors, categories, etc.) while the one for LF it contains 852 users, 6,256 artists and 150,925 entities (songs, genres, categories, etc.). We considered entities within paths of length $L = 4$.

4.2 Evaluation Protocol

For the evaluation of our approach we adopted a method similar to the one described in [7]. From the original matrix $\hat{S}$, we built two new matrices: $\hat{S}_{train}$ and $\hat{S}_{test}$. We used the former for producing the sets I_u^+, I_u^- and TR (used for training the model), as detailed in Section 3.2, the latter for evaluating the trained model. To create $\hat{S}_{test}$ we randomly selected ten positive feedback for each user from the initial matrix $\hat{S}$. In order to assess the performance of SPrank in situations affected by different level of data sparsity, we evaluated our algorithm considering different sizes of user profiles as done by [22]. Specifically, for each user we randomly considered at most m positive feedback from the original matrix $\hat{S}$ (denoted with "given m" in Tables 1, 2 and 4) to form $\hat{S}_{train}$, and we discarded all the others. We say *at most* m because some users can actually have less than m positive feedback. Different values of m correspond to different matrices $\hat{S}_{train}$, different sets I_u^+, I_u^- and then different training sets TR. In Table 1 we show some statistics about the number of non-zero entries in $\hat{S}_{train}$ (nnz), the average number of non-zero entries for each user (avg nnz) and the sparseness (spars) of $\hat{S}_{train}$ for the different conditions (given 5, given 10, etc.). For each condition we learned the model on the training set TR and we applied the learned model to predict the unknown values (0-entries) in $\hat{S}_{train}$. We got the full recommendation lists by sorting the predicted values for each user and then we evaluated the accuracy for the *top-N* items. We repeated the procedure 5 times for each condition by randomly drawing new training/test sets in each round and at the end we averaged the results. To evaluate the accuracy of the system we measured the *recall@N*, widely used for evaluating *top-N* recommender systems [7, 17]. The computation of *recall@N* goes through a procedure similar to the one introduced in [7], as detailed in the following.
For each $\hat{s}_{ui}$ in $\hat{S}_{test}$, from the full recommendation list computed for u, we randomly selected 100 items appearing neither in the test set related to that user nor in the user profile. We got a ranked list consisting of these 101 items. The *top-N* recommendation list is obtained by considering just the first N items ($N = 5, 10, 20$) in this ranked list. Being *pos* the position of the test item i within the ranked list, we have a hit if $pos \leq N$, otherwise we have a miss. We note that for any single test case, we have just one relevant item (i.e., the tested item i). The recall for a single test case is either 0 (in case of a miss) or 1 (in case of a hit). The overall recall on all users is defined by averaging over all test cases:

$$recall@N = \frac{\#hits}{\mid \hat{S}_{test} \mid} \tag{2}$$

The formula for *precision@N* differs from Equation 2 just by a multiplicative term N appearing at the denominator [7]. For this reason, in our results we do not report it.

	MovieLens			**LastFM**		
given	**nnz**	**avg-nnz**	**spars**	**nnz**	**avg-nnz**	**spars**
5	18960	5.00	99.82%	4260	5.00	99.92%
10	36650	9.66	99.65%	8146	9.56	99.84%
20	64137	16.91	99.39%	13542	15.89	99.74%
30	84301	22.23	99.2%	-	-	-
50	111001	29.27	98.95%	-	-	-
All	157194	41.45	98.51%	17783	20.87	99.66%

Table 1: Statistics of the ML and LF training matrices under different conditions of user profile size.

4.3 Evaluation and results discussion

In the following, we present the results of the experiments that have been carried out to answer to the following questions:
1. *Is the chosen learning to rank algorithm effective?*
2. *Does the proposed approach improve, in situations affected by different degree of data sparsity, the recommendation accuracy with respect to existing state-of-the-art ranking-oriented approaches for positive implicit feedback?*

4.3.1 *Effectiveness of the learning to rank algorithm*

To answer the first question we compared *BagBoo*, the learning to rank algorithm used in SPrank, with two other algorithms: *Sum* and *GBRT*. In *Sum*, the ranking function $f(\mathbf{x_{ui}})$ is the arithmetic sum of all the path-based feature values. On the one side, the comparison with *Sum* gives us a baseline for semantic-based recommendation algorithm where all associations are equally considered.This allows us to evaluate our learning to rank algorithm against a basic

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

not ranking-oriented semantic recommender. On the other side, the comparison with *GBRT* is useful to prove the effectiveness of combining bagging and boosting. We recall that *BagBoo* is an extension of *GBRT* as it is an ensemble of gradient boosting regression trees.
Table 2 summarizes the accuracy results obtained on the two datasets. For the ML dataset we observe that *BagBoo* outperforms both *GBRT* and *Sum* in all the different conditions of sparsity degree, even when there are few positive examples for each user. For example, if we analyze the *recall@5*, the improvement of *BagBoo* over *Sum* goes from +6.6% to +11% respectively for the two limit conditions (`given 5` and `given All`). We also see that for the condition `given 5` the improvement with respect to *GBRT* is very marked (+15.8%), but it decreases with the increasing of positive examples (+2% for `given 30` and +3.2% for `given All`). For this dataset we observe that when there are only a few positive training examples for each user, *GBRT* is not able to learn an effective ranking function. This is not the case for the bagging of several *GBRT*s. Looking at the results for LF we observe again *BagBoo* outperforms the other algorithms but for the condition `given 5`.
From the results on the two datasets we can draw the following conclusions: `SPrank` benefits from learning to rank, both *GBRT* and *BagBoo* show substantial improvements with respect to the *Sum* baseline, particularly when the number of positive examples, and then the training data, increases; due to bagging, *BagBoo* outperforms *GBRT* in all the analyzed situations. Hence, *BagBoo* is a valid candidate for learning the ranking function in `SPrank`.

	MovieLens			LastFM		
Alg.	**r@5**	**r@10**	**r@20**	**r@5**	**r@10**	**r@20**
			given 5			
BagBoo	**0.420**	**0.578**	**0.745**	**0.349**	0.457	0.551
GBRT	0.262	0.405	0.572	0.323	0.442	0.572
Sum	0.354	0.560	0.541	0.319	**0.482**	**0.593**
			given 10			
BagBoo	**0.462**	**0.623**	**0.786**	**0.423**	**0.541**	0.636
GBRT	0.427	0.603	0.771	0.371	0.510	0.615
Sum	0.382	0.581	0.565	0.349	0.517	**0.668**
			given 20			
BagBoo	**0.496**	**0.661**	**0.816**	**0.496**	**0.618**	**0.721**
GBRT	0.475	0.653	0.810	0.452	0.592	0.689
Sum	0.396	0.599	0.587	0.385	0.533	0.693
			given 30			
BagBoo	**0.515**	**0.679**	**0.831**	-	-	-
GBRT	0.495	0.669	0.824	-	-	-
Sum	0.417	0.610	0.595	-	-	-
			given 50			
BagBoo	**0.524**	**0.691**	**0.841**	-	-	-
GBRT	0.497	0.668	0.825	-	-	-
Sum	0.423	0.618	0.613	-	-	-
			given All			
BagBoo	**0.539**	**0.699**	**0.846**	**0.543**	**0.657**	**0.752**
GBRT	0.507	0.679	0.837	0.448	0.587	0.708
Sum	0.429	0.633	0.632	0.399	0.546	0.702

Table 2: Results for *BagBoo*, *GBRT* and *Sum* given different user profile size.

Impact of irrelevant item sampling. In Table 3 we show how different values of K, that is the factor that influences the size of I_u^{-*} (cf. Section 3.2), affects the accuracy of `SPrank`. We can observe that increasing K and hence the number of unknown examples in the training data, the accuracy does not increase. An evident advantage in selecting a small K is the reduction of the training set size and consequently of the time required to build the model. Based on these observations, we chose $K = 2$ in all our experiments.

	MovieLens			LastFM		
K	**r@5**	**r@10**	**r@20**	**r@5**	**r@10**	**r@20**
1	0.529	0.691	0.84	0.541	0.643	0.737
2	0.539	0.699	0.846	0.543	0.657	0.752
5	0.531	0.690	0.841	0.544	0.658	0.749
10	0.525	0.691	0.839	0.530	0.639	0.736
25	0.525	0.689	0.838	0.537	0.642	0.737

Table 3: Accuracy results for different values of K (used to create I_u^{-*}).

4.3.2 *Comparison with other Algorithms*

In order to evaluate `SPrank` we compared it both with a hybrid algorithm proposed to address cold-start scenarios and with ranking oriented collaborative filtering algorithms. All of these approaches are state-of-the-art methods for positive implicit feedback scenarios. *BPRLinearMap* (*BPRLin*), *BPRMF* and *SLIM* have been presented in Section 2. For *BPRLinearMap* we built the item-attribute matrix using the same content data used for `SPrank`. *SoftMarginRankingMF* (*SMRMF*) is a matrix factorization model for item prediction optimized for a soft margin ranking loss using stochastic gradient descent inspired by [23] [20].The computation of the recommendations for all these comparative algorithms has been done with the publicly available software library *MyMediaLite* [13].
Results discussion. Table 4 shows the results obtained for `SPrank` and all the other comparative methods on both ML and LF. We observe that on the ML dataset our algorithm outperforms the others under all the different user profile conditions but `given All` where *BPRMF* achieves the best recall values, and `SPrank` gets the second best place. We can note that `SPrank` outperforms significantly the other methods when $\hat{S}$ is very sparse. Referring to the *SLIM* method, for the conditions `given 5` and `given 10`, the improvements in terms of *recall@5* are respectively +20.2% and +23.3%.
For the LF dataset, which is slightly sparser than ML, `SPrank` is always the best performing algorithm. In this case, *BPRLin* – the most suited approach for dealing with data sparsity – is the second best performing. Also in this case the improvements are substantial especially for the conditions of higher sparseness.
From these experimental results, we can conclude that `SPrank` is able to compute accurate *top-N* recommendations even when there are few positive feedback where instead collaborative filtering algorithms have showed lower accuracy. On both datasets `SPrank` has also outperformed *BPRLin*, the other hybrid recommendation method proposed to address cold start and sparsity problems.

5. CONCLUSION

In this paper we have presented `SPrank`, a novel hybrid *top-N* item recommendation algorithm from implicit feedback able to exploit `LOD` knowledge bases to compute accurate recommendations. In `SPrank`, positive implicit feedback and semantic item descriptions are merged in a unique graph-based representation that allows us to extract *path-based* features describing complex relationships between items preferred by the user and items to be recommended. Then,

	MovieLens			LastFM		
Alg.	**r@5**	**r@10**	**r@20**	**r@5**	**r@10**	**r@20**
given 5						
`SPrank`	**0.420**	**0.578**	**0.745**	**0.349**	**0.457**	**0.551**
BPRMF	0.353	0.502	0.672	0.213	0.308	0.413
SLIM	0.218	0.363	0.552	0.077	0.152	0.287
BPRLin	0.218	0.314	0.442	0.289	0.381	0.440
SMRMF	0.216	0.354	0.526	0.111	0.181	0.280
given 10						
`SPrank`	**0.462**	**0.623**	**0.786**	**0.423**	**0.541**	**0.636**
BPRMF	0.383	0.533	0.704	0.278	0.386	0.503
SLIM	0.229	0.377	0.569	0.128	0.217	0.350
BPRLin	0.279	0.384	0.510	0.407	0.495	0.567
SMRMF	0.263	0.423	0.614	0.178	0.261	0.372
given 20						
`SPrank`	**0.496**	**0.661**	**0.816**	**0.496**	**0.618**	**0.721**
BPRMF	0.429	0.589	0.756	0.335	0.451	0.570
SLIM	0.320	0.437	0.614	0.182	0.278	0.427
BPRLin	0.348	0.464	0.592	0.486	0.577	0.643
SMRMF	0.360	0.505	0.698	0.236	0.343	0.470
given 30						
`SPrank`	**0.515**	**0.679**	**0.831**	-	-	-
BPRMF	0.486	0.648	0.802	-	-	-
SLIM	0.388	0.495	0.652	-	-	-
BPRLin	0.377	0.497	0.621	-	-	-
SMRMF	0.387	0.563	0.752	-	-	-
given 50						
`SPrank`	**0.524**	**0.691**	**0.841**	-	-	-
BPRMF	0.519	0.682	0.834	-	-	-
SLIM	0.453	0.566	0.688	-	-	-
BPRLin	0.392	0.513	0.639	-	-	-
SMRMF	0.416	0.596	0.783	-	-	-
given All						
`SPrank`	0.539	0.699	0.846	**0.543**	**0.657**	**0.752**
BPRMF	**0.564**	**0.727**	**0.865**	0.373	0.495	0.615
SLIM	0.513	0.651	0.761	0.228	0.323	0.470
BPRLin	0.414	0.535	0.658	0.527	0.616	0.679
SMRMF	0.471	0.651	0.818	0.266	0.383	0.510

Table 4: Comparison of `SPrank` with other approaches under different conditions of the user profile size.

the recommendation problem is cast in a learning to rank fashion to compute *top-N* recommendations.
We have demonstrated in our experiments that `SPrank` outperforms several state-of-the-art approaches for implicit feedback in two datasets belonging to two different domains: `MovieLens` for movie and `Last.fm` for music. The experiments have been conducted varying the sparseness of the implicit feedback matrix for analyzing how `SPrank` deals with sparse data with respect to other competitive approaches. We have observed significant improvements over collaborative filtering methods especially in case of high sparsity.

6. REFERENCES

[1] Hetrec '11: Proceedings of the 2nd international workshop on information heterogeneity and fusion in recommender systems. ACM, 2011.

[2] S. S. Anand, P. Kearney, and M. Shapcott. Generating semantically enriched user profiles for web personalization. *ACM Trans. Internet Technol.*, 7(4), 2007.

[3] C. Bizer, T. Heath, and T. Berners-Lee. Linked data - the story so far. *Int. J. Sem. Web Inf. Syst*, 5(3):1–22, 2009.

[4] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[5] I. Cantador, A. Bellogín, and P. Castells. A multilayer ontology-based hybrid recommendation model. *AI Commun.*, 21(2-3):203–210, 2008.

[6] O. Chapelle and Y. Chang. Yahoo! learning to rank challenge overview. *Journal of Machine Learning Research - Proceedings Track*, 14:1–24, 2011.

[7] P. Cremonesi, Y. Koren, and R. Turrin. Performance of recommender algorithms on top-n recommendation tasks. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 39–46. ACM, 2010.

[8] T. Di Noia, R. Mirizzi, V. C. Ostuni, and D. Romito. Exploiting the web of data in model-based recommender systems. In *Proceedings of the sixth ACM conference on Recommender systems*, pages 253–256. ACM, 2012.

[9] T. Di Noia, R. Mirizzi, V. C. Ostuni, D. Romito, and M. Zanker. Linked open data to support content-based recommender systems. In *Proceedings of the 8th International Conference on Semantic Systems*, I-SEMANTICS '12, pages 1–8. ACM, 2012.

[10] I. Fernández-Tobías, I. Cantador, M. Kaminskas, and F. Ricci. A generic semantic-based framework for cross-domain recommendation. In *Proc. of the 2nd Int. Workshop on Information Heterogeneity and Fusion in Recommender Systems*, HetRec '11, pages 25–32. ACM, 2011.

[11] J. H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29:1189–1232, 2000.

[12] Z. Gantner, L. Drumond, C. Freudenthaler, S. Rendle, and L. Schmidt-Thieme. Learning attribute-to-feature mappings for cold-start recommendations. In *Proceedings of the 2010 IEEE International Conference on Data Mining*, ICDM '10, pages 176–185. IEEE Computer Society, 2010.

[13] Z. Gantner, S. Rendle, C. Freudenthaler, and L. Schmidt-Thieme. Mymedialite: a free recommender system library. In *Proceedings of the fifth ACM conference on Recommender systems*, RecSys '11, pages 305–308. ACM, 2011.

[14] T.-Y. Liu. Learning to rank for information retrieval. *Found. Trends Inf. Retr.*, 3(3):225–331, 2009.

[15] S. E. Middleton, D. D. Roure, and N. R. Shadbolt. Ontology-based recommender systems. *Handbook on Ontologies*, 32(6):779–796, 2009.

[16] B. Mobasher, X. Jin, and Y. Zhou. Semantically enhanced collaborative filtering on the web. In B. Berendt, A. Hotho, D. Mladenic, M. Someren, M. Spiliopoulou, and G. Stumme, editors, *Web Mining: From Web to Semantic Web*, volume 3209 of *LNCS*, pages 57–76. Springer Berlin Heidelberg, 2004.

[17] X. Ning and G. Karypis. Slim: Sparse linear methods for top-n recommender systems. In *Proceedings of the 2011 IEEE 11th International Conference on Data Mining*, ICDM '11, pages 497–506. IEEE Computer Society, 2011.

[18] X. Ning and G. Karypis. Sparse linear methods with side information for top-n recommendations. In *Proceedings of the sixth ACM conference on Recommender systems*, RecSys '12, pages 155–162. ACM, 2012.

[19] D. Y. Pavlov, A. Gorodilov, and C. A. Brunk. Bagboo: a scalable hybrid bagging-the-boosting model. In *Proc. of the 19th ACM int. conference on Information and knowledge management*, CIKM '10, pages 1897–1900. ACM, 2010.

[20] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, UAI '09, pages 452–461. AUAI Press, 2009.

[21] G. Semeraro, P. Lops, P. Basile, and M. de Gemmis. Knowledge infusion into content-based recommender systems. In *Proceedings of the third ACM conference on Recommender systems*, pages 301–304. ACM, 2009.

[22] Y. Shi, A. Karatzoglou, L. Baltrunas, M. Larson, N. Oliver, and A. Hanjalic. Climf: learning to maximize reciprocal rank with collaborative less-is-more filtering. In *Proceedings of the sixth ACM conference on Recommender systems*, RecSys '12, pages 139–146. ACM, 2012.

[23] M. Weimer, A. Karatzoglou, Q. V. Le, and A. J. Smola. Cofi rank - maximum margin matrix factorization for collaborative ranking. In *Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems*, 2007.