Top-N Recommendations from Implicit Feedback Leveraging Linked Open Data [⋆]

Vito Claudio Ostuni, Tommaso Di Noia, Roberto Mirizzi, Eugenio Di Sciascio

Polytechnic University of Bari, Italy
{ostuni,mirizzi}@deemail.poliba.it, {t.dinoia,disciascio}@poliba.it

(Extended Abstract)

Abstract. 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 (Linked Open Data) 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.

1 Introduction

We propose SPrank (***S**emantic **P**ath-based **rank**ing*), a hybrid recommendation algorithm [1] to compute *top-N* item recommendations from implicit feedback. SPrank effectively incorporates ontological knowledge belonging to the Web of Data with collaborative user preferences in a graph-based setting. In SPrank, the ontological knowledge describing the items is extracted from DBpedia[1], a well-known encyclopedic knowledge base belonging to the LOD[2] (Linked Open Data) 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. 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 work are:

– combination of semantic item descriptions from the Web of Data and implicit feedback for the *top-N* recommendation task;

[⋆] The full version of this paper, with complete experimental evaluation and related work, has been published in [8].

[1] http://dbpedia.org

[2] http://linkeddata.org

– 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.

2 SPrank: Semantic Path-based Ranking

The common graph-based nature of the data models for both content-based recommender systems exploiting `LOD` and collaborative filtering recommender systems, 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 Fig. 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.

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 Fig. 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 Fig. 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 unobserved items I_u^- 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 [6]. 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. 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 probabil-

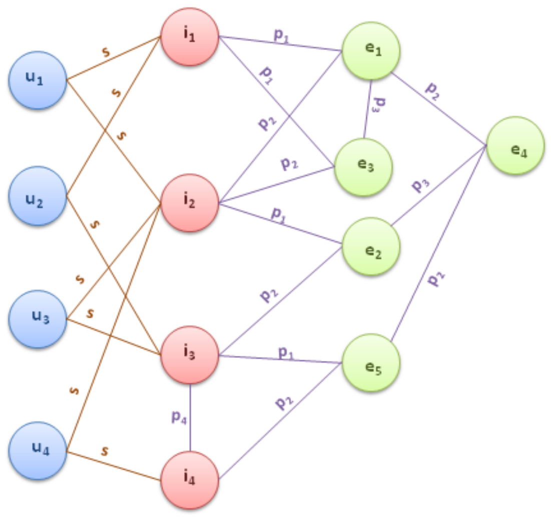

Fig. 1. Graph-based representation of the data model.

ity 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^-.
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. Given a graph G as the one represented in Fig. 1, we want to extract features able to characterize the interactions between users, items and entities capturing the complex relationships between them. 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. Given an undirected graph G as previously defined, we define a path as the acylic sequence of edges of the form $(s, \ldots, r_l, \ldots, r_L)$ where $s = (u, i)$ and $r_L = (v, i')$ with $i \neq i'$. 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:

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

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.
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 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 [6]. There are two point-wise algorithms that have proven particularly successful: *Random Forests* and *Gradient Boosted Regression Trees* (GBRT) [5]. We formulate the learning to rank problem as a combination of both Random Forest and GBRT following the idea of *BagBoo* introduced by [9]. 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.

3 Evaluation

The experiments to evaluate the effectiveness of `SPrank` in terms of accuracy for *top-N* recommendations have been carried out on two datasets belonging to two different domains: `MovieLens` (movies) and `Last.fm` (musical artists). The datasets that have been used are available at: `http://sisinflab.poliba.it/semanticweb/lod/recsys/datasets/`.

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), 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. 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 and 2) 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. For different sizes of the user profile, 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 [2, 7]. The computation of *recall@N* goes through a procedure similar to the one introduced in [2], 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 [2]. For this reason, in our results we do not report it.

Effectiveness of the learning to rank algorithm. To understand whether the chosen learning to rank algorithm is effective, 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 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 1 summarizes the accuracy results obtained on the two datasets. For the `MovieLens` 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 `Last.fm` 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 1. Results for *BagBoo*, *GBRT* and *Sum* given different user profile size.

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*) [10] adopts a *Bayesian Personalized Ranking* (*BPR*) criterion for optimizing a ranking loss. For *BPRLinearMap* we built the item-attribute matrix using the same content data used for `SPrank`. *BPRMF* [3] is a hybrid extension of *BPR* that learns a linear mapping on the user/item features from the factorization and auxiliary user/item-attribute matrix. This extension is able to compute useful recommendations in cold-start scenarios. *SLIM* [7] adopts a *Sparse Linear* method for learning a sparse aggregation coefficient matrix that is used for computing *top-N* recommendations. *SoftMarginRankingMF* (*SMRMF*) is a matrix factorization model for item prediction optimized for a soft margin ranking loss using stochastic gradient descent inspired by [10]. The computation of the recommendations for all these comparative algorithms has been done with the publicly available software library *MyMediaLite* [4].
Results discussion. Table 2 shows the results obtained for `SPrank` and all the other comparative methods on both `MovieLens` and `Last.fm`. We observe that on the `MovieLens` 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 `Last.fm` dataset, which is slightly sparser than `MovieLens`, `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.

References

1. R. Burke. Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, 12(4):331–370, Nov. 2002.
2. 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*, RecSys '10, pages 39–46, New York, NY, USA, 2010. ACM.
3. 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, Washington, DC, USA, 2010. IEEE Computer Society.
4. 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, New York, NY, USA, 2011. ACM.

	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 2. Comparison of `SPrank` with other approaches under different conditions of the user profile size.

5. T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning.* Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001.
6. T.-Y. Liu. Learning to rank for information retrieval. *Found. Trends Inf. Retr.*, 3(3):225–331, Mar. 2009.
7. 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.
8. V. C. Ostuni, T. Di Noia, E. Di Sciascio, and R. Mirizzi. Top-n recommendations from implicit feedback leveraging linked open data. In *7th ACM Conference on Recommender Systems (RecSys 2013).* ACM, ACM Press, 2013.
9. D. Y. Pavlov, A. Gorodilov, and C. A. Brunk. Bagboo: a scalable hybrid bagging-the-boosting model. In *19th ACM CIKM*, 2010.
10. 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, Arlington, Virginia, United States, 2009. AUAI Press.