A Linked Open Data Schema-Driven Approach for Top-N Recommendations

Thomas Wever
thomaswever@gmail.com

Flavius Frasincar
frasincar@ese.eur.nl

Erasmus University Rotterdam
PO Box 1738, NL-3000 DR
Rotterdam, the Netherlands

ABSTRACT

A recent direction on improving the performance of recommender systems is by exploiting data semantics. However, previous research has given little attention to the selection of the most relevant data for recommendations. In this paper we present a schema-driven approach for top-N recommendations. We identify the most promising path types in a structured information network, i.e., `Linked Open Data` (`LOD`), based on variable importance scores. In contrast to previous work, we focus on which information is most important and remove path types that appear unimportant. The methodology is tested on the `MovieLens 1M` dataset, semantically enhanced with data from the `LOD` cloud. The results are threefold. First, we find that the LOD cloud is useful especially for small user profiles. Secondly, we find that the `LOD` cloud is useful for large user profiles only when the most popular items are not considered. Lastly, we find that selecting the most relevant data from the `LOD` cloud improves performance compared to using all extracted `LOD` data.

CCS Concepts

•**Information systems** → **Recommender systems;** Personalization; •**Computing methodologies** → *Semantic networks;*

Keywords

Top-N recommendations; Linked Open Data; information network schema; random forest

1. INTRODUCTION

Today, there is more information available to us than we can ever process [11]. This information overload calls for efficient methods to help users filter out content that is useful to them. A class of algorithms called Recommender Systems (RSs) was specifically designed for this task and has been studied extensively both commercially and academically [1].

SAC 2017, April 03 - 07, 2017, Marrakech, Morocco

ISBN 978-1-4503-4486-9/17/04…$15.00

DOI: http://dx.doi.org/10.1145/3019612.3019843

A RS makes personalized recommendations to guide users in their search for useful content in a potentially enormous space of options [5]. For example, it can advise on which movie to watch on Netflix, which restaurant to go to for dinner, or even who to be friends with on Facebook.

There have been many approaches to address the RS problem. Most recently, researchers have tried to leverage the Semantic Web to increase performance of RSs. The Semantic Web, sometimes referred to as the Web of Data, is an initiative led by the World Wide Web Consortium (W3C) to create a framework in which linked data can be easily shared and reused. With defined standards on formats, the Semantic Web is easily accessible for both humans and machines, and can thus easily be integrated in a variety of applications.

A subset of the Semantic Web which has recently gained much attention in the RS research area is the `Linked Open Data` (`LOD`) cloud. This set originated in 2007 from the Linking Open Data project, which was a community effort primarily lead by researchers and developers [3]. The (ongoing) goal was to identify and convert freely available datasets according to the Linked Data principles, and publish them on the Semantic Web. Thanks to the open nature of the project it was able to grow rapidly, resulting in billions of RDF statements describing and linking data on the `LOD` cloud today.

In this paper, we take a closer look at which information in the `LOD` cloud is most informative. To do this, we build a novel RS similar to `SPrank` [14] exploiting one of the most popular datasets of the `LOD` cloud called `DBPedia`. In contrast to previous research, we aim at improving performance by filtering out uninformative data from the `LOD` cloud. More specifically, we argue that some information is not useful for recommendations, because the information is too generic or simply not important. For example, in a movie RS, the year of release is probably less important information than the director of the movie. To do this, we take a schema-based approach which allows us to find the most relevant information based on variable importance scores. We use the work from [20], by implementing their efficient matrix oriented path finding methodology in a RS environment. Moreover, we evaluate the performance under different scenarios. Specifically, not only do we look at performance for different profile sizes, but we also explore the scenario in which the most popular movies have been removed from the data, similar to [7].

The paper is structured as follows. In section 2 we describe related work. Then, in section 3, we explain the proposed methodology, followed by its evaluation in section 4. Last, in section 5, we give our conclusions and future work.

2. RELATED WORK

A clear overview of the literature regarding the early days of RSs is given by [1]. In general, RSs can be categorized into one of three categories:

- **Content-based** RSs compare the content of previously liked items with new (unseen) items and recommend those items which are most similar.
- **Collaborative** RSs base recommendations on what other users with similar preferences have liked in the past.
- **Hybrid** recommender systems combine content-based and collaborative aspects.

Content-based RSs are built on the assumption that there is a high chance users will like items similar to items they have liked in the past [17]. Collaborative methods, also known as collaborative filtering, rely exclusively on user ratings to make recommendations [19]. Content-based methods cannot take advantage of inter-user relations, while collaborative methods are unable to incorporate item characteristics. Researchers have come up with several hybrid methods to take advantage of the best of both worlds, thereby mitigating some of the problems as cold start and limited content.

Some of the first authors to propose the leverage of the `LOD` cloud in a RS were the authors of [10]. They argued that the use of open data sources in RSs could solve the data acquisition problem, thereby mitigating cold start and sparsity problems.

An early content-based music RS that used the `DBPedia` dataset is *dbrec* [15]. The RS relied on the *Linked Data Semantic Distance* (LDSD) [16] to identify similarities between items, which is based only on the number of links from one item to another. The authors found that *dbrec* provided more novel recommendations, while performing reasonably well in terms of precision compared to more extensive hybrid RSs. Novelty of recommendations was also advocated by the authors of [9], who developed a generic knowledge framework on top of `DBPedia` to make cross-domain recommendations. Specifically, they focused on the scenario where a place of interest acts as input for a music RS. For example, they argued that someone who visited the city of Vienna is likely to enjoy the opera, while someone who visits the O2 stadium in London probably enjoys more popular music.

In [8], the authors used multiple cross-linked datasets of the `LOD` cloud in a content-based movie recommender system. In particular, to determine similarity scores between two movies, they checked how much information was shared between them. For example, two movies might have the same director or the same subject. Roughly speaking, the more information was shared, the higher the degree of similarity.

Lastly, the authors of [14] leveraged `DBpedia` to create an ontology based top-N recommender system called `SPrank` (Semantic Path-based ranking). The authors used a path-based approach to identify features of movies, and implemented a learning to rank algorithm to perform model based recommendations. They evaluated their algorithm using implicit feedback, and were able to make recommendations that outperformed several state-of-the-art top-N recommender systems like BPR (Bayesian Personalized Ranking) [18] and SLIM (Sparse Linear Methods) [13].

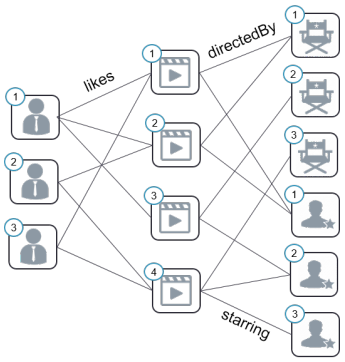


Figure 1: Information network example

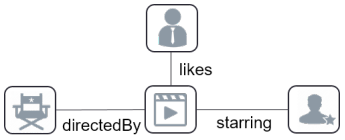


Figure 2: Information schema example

Our research differs from previous research in the sense that we focus on the most important information from an ontology only. Specifically, it differs from [14], because we do not only consider all paths up to a certain length, but rather define network schemas in which only certain path types are allowed. Also, it differs from previous research because of the way we find all path instances. For efficiency reasons, we exploit a matrix oriented path finding methodology proposed by the authors of [20]. Our research differs from [20], because we implement the path finding technique in a RS. In contrast, in [20] the authors implemented their methodology in semantically-enhanced web search. Lastly, our research differs from [7], because they did not consider semantically enhanced RSs, but rather focused on collaborative filtering approaches.

3. METHODOLOGY

We propose a LOD-based RS that depends on finding the number of connections between items following predefined paths in a graph. By defining the structure of the graph we create an information network in which only certain paths can exist. This greatly reduces the time needed to find all paths. Moreover, some paths might add noise instead of semantics. Ignoring these path types could increase accuracy.

3.1 Graphs and Information Networks

This paper uses graph theory to represent and analyze connections between users, items, and entities. In its most basic form, a graph $\mathcal{G} = (\mathcal{N}, \mathcal{L})$ consists of a set of nodes $\mathcal{N} = (n_1, n_2, \ldots, n_N)$ and a set of edges $\mathcal{L} = (l_1, l_2, \ldots, l_L)$ connecting the nodes. All users, items, and entities are represented by nodes, while all connections are represented by edges. A graph can be directed or undirected. In an undirected graph all connections between nodes are symmetric, while in a directed graph, a connection is only specified in one direction. For this paper, we consider solely undirected graphs, as for example movies cannot direct directors, but we do want to be able to associate movies with directors.

More structure can be added to the set of nodes and the set of edges. For example, the set of nodes can be split into three subsets containing only users, items (to be recommended), or entities (related to items). Also, the set of edges can be split in links between users and items, and links between items and entities or other items. This can be specified more formally by using mapping functions as defined by the authors of [20]. Let $\phi : \mathcal{N} \to \mathcal{A}$ be the node mapping function assigning every node $n \in \mathcal{N}$ to a node type $A \in \mathcal{A}$, that is, $\phi(n) \in \mathcal{A}$. Also, let $\xi : \mathcal{L} \to \mathcal{B}$ be the mapping function assigning every edge $l \in \mathcal{L}$ to an edge type $\xi(l) \in \mathcal{B}$. Then, the combination of a graph with mapping functions ϕ and ξ is defined as an *information network*. The structure of an information network is defined in a *network schema*, denoted $T_{\mathcal{G}} = (\mathcal{A}, \mathcal{B})$. The network schema defines which node types exist in the graph, as well as between which node types certain link types are allowed.

Figure 1 shows an example information network. There are 3 users, 4 movies and 6 entities, which in turn consist of 3 directors and 3 actors. Users and movies are linked when the user has liked that particular movie. Similarly, there exists a link between movie and entity when either the movie was directed by that director, or when the actor starred in the movie. The information schema of this network in shown in Figure 2. Here we see all links that are allowed in the information network.

The information network and schema can be used to search for similarities between items and/or users. To do so, we need an efficient way to determine how often two nodes are connected, and with what kind of links. We implement a matrix oriented path-finding approach that has previously been used in semantically-enhanced web search [20]. For this approach, we first introduce the concept of meta paths.

3.2 Meta Paths

In general, two nodes can be similar if there exists one or more connections between them in the information network. Nodes can be connected by a direct link, or a concatenation of multiple links called meta paths. In our example, it can be seen that users can be associated with movies directly if they have liked the movie, or indirectly if they have liked a movie with a similar actor or director.

Given a network schema $T_G = (\mathcal{A}, \mathcal{B})$, a meta path is defined as a sequence of links between two node types and is denoted as

$$\mathcal{P} = A_1 \xrightarrow{R_1} A_2 \xrightarrow{R_2} \dots \xrightarrow{R_{l-1}} A_l. \tag{1}$$

There can be multiple instances of a meta path in an information network. Let $p = (a_1, r_1, a_2, r_2, \dots, r_{l-1}, a_l)$ be a path in an information network. Then, p is defined as an instance of $\mathcal{P}$ if $\forall i, \phi(a_i) = A_i$ and $\forall i, \xi(r_i) = R_i$, where r_i is the link between node a_i and a_{i+1}. Note that a meta path defines a concatenation of links on the schema level, whereas a path denotes an instance of a meta path. We will also refer to meta path as a path type. The length of a meta path is defined as the number of links in $\mathcal{P}$, and a path is symmetric if and only if there exists an inverse path of $\mathcal{P}$, denoted

$$\mathcal{P}^{-1} = A_l \xrightarrow{R_{l-1}} \dots \xrightarrow{R_2} A_2 \xrightarrow{R_1} A_1$$

One can note that the paths considered in this work are symmetric as we are dealing with undirected graphs (i.e., the paths have direction, but the edges do not have orientation).

Two meta paths $\mathcal{P}^{(1)}$ and $\mathcal{P}^{(2)}$ can be concatenated if the last node type of $\mathcal{P}^{(1)}$ is the same as the first node type of $\mathcal{P}^{(2)}$. To clarify, let us consider the singular length path instances from our example in Figure 1:

$$\begin{aligned} \texttt{user 2} &\xrightarrow{likes} \texttt{movie 2} \\ \texttt{movie 2} &\xrightarrow{starring} \texttt{actor 1} \\ \texttt{actor 1} &\xrightarrow{starring} \texttt{movie 1}. \end{aligned}$$

These can be concatenated to form a length 3 path:

$$\texttt{user 2} \xrightarrow{likes} \texttt{movie 2} \xrightarrow{starring} \texttt{actor 1} \xrightarrow{starring} \texttt{movie 1}.$$

Now, we introduce a matrix oriented path finding algorithm to find all path instances in the information network belonging to each defined path type in the information schema.

3.3 Matrix Oriented Path Finding

Given an information network and a relation type R_k, the adjacency matrix $W^{(R_k)}_{A_i A_j}$ is defined as an $n \times m$ matrix indicating which nodes from node type A_i are linked to nodes from node type A_j and by how many links $r \in R_k$. Here, n and m represent the number of nodes belonging to node type A_i and A_j, respectively. Notice that the transpose of $W^{(R_k)}_{A_i A_j}$ represents the adjacency matrix from A_j to A_i.

The commuting matrix $M_{\mathcal{P}}$ contains the number of path instances $p \in \mathcal{P}$, where $\mathcal{P} = A_1 \xrightarrow{R_1} A_2 \xrightarrow{R_2} \dots \xrightarrow{R_{l-1}} A_l$ is a specific meta path, and where $M_{\mathcal{P}}(i, j)$ indicates how many path instances exist between node $x_i \in A_1$ and $y_j \in A_l$, following meta path $\mathcal{P}$. The commuting matrix can be calculated as the product of the adjacency matrices, that is:

$$M_{\mathcal{P}} = W^{R_1}_{A_1 A_2} W^{R_2}_{A_2 A_3} \dots W^{R_{l-1}}_{A_{l-1} A_l}. \tag{2}$$

3.4 Combining Commuting Matrices

Dependent on how many node types and link types have been defined, there are as many commuting matrices as there are unique meta paths in the information network. Here we propose a method for combining commuting matrices. The resulting path count matrix can be used to store the number of paths between users and items.

Let $M_{\mathcal{P}^{(i)}}$ be defined as the $n \times n$ symmetric commuting matrix containing all path counts given meta path $\mathcal{P}^{(i)}$, where n is the total number of items available for recommendation. First, we evaluate all symmetric paths of the type:

$$\texttt{item} \xrightarrow{R_j} * \xrightarrow{R_j} \texttt{item}, \tag{3}$$

where the $*$ can be any node type, including other users. We focus on symmetric paths because these make most sense intuitively. For example, if an actor is liked for his acting skills, this does not necessarily mean he is also a good director. Additionally, due to computational limitation, we restrict the length of these paths to 2. Next, let z be defined as the number of different path types in the information network. We can now define an $(n * n) \times z$ path count matrix as:

$$P = \left[\text{vec}(M_{\mathcal{P}^{(1)}}) \quad \text{vec}(M_{\mathcal{P}^{(2)}}) \quad \cdots \quad \text{vec}(M_{\mathcal{P}^{(z)}})\right], \tag{4}$$

where $\text{vec}(A)$ is the matrix vectorization operator (i.e., stacks the columns of A on top of each other). For this matrix, it holds that element $P(i * n + j, k)$ represents the number of paths from item with index i to item with index j following

meta-path $\mathcal{P}^{(k)}$. Next, let $\mathbf{x}_{u,i}$ be defined as the vector of size z, containing the cumulative path counts from user u to item i. All paths from user to item greater than length 1 are of the form:

$$\texttt{user} \xrightarrow{likes} \texttt{item} \xrightarrow{R_j} * \xrightarrow{R_j} \texttt{item},$$

where the last part is the same as in Equation 3. This results in paths up to length 3. Note that if the length constraint is relaxed, $*$ can also be replaced by multiple instances of Equation 3. Now we can use matrix P to calculate the path counts using the formula:

$$\mathbf{x}_{u,i} = \sum_{j \in \mathcal{I}_u^+} P(ji), \tag{5}$$

where $P(ji)$ is the row vector from P representing the path counts from item j to item i for all path types, and $\mathcal{I}_u^+$ contains all items which are liked by user u.

Again we refer to our example. Since there are three relation types in the example network schema, we search for three path types:

$$\texttt{Directing:}\ \ \texttt{Movie} \xrightarrow{directedBy} \texttt{Director} \xrightarrow{directed} \texttt{Movie};$$
$$\texttt{Likes:}\ \ \texttt{Movie} \xrightarrow{likedBy} \texttt{User} \xrightarrow{likes} \texttt{Movie};$$
$$\texttt{Starring:}\ \ \texttt{Movie} \xrightarrow{hasStarring} \texttt{Actor} \xrightarrow{starring} \texttt{Movie}.$$

Note that these path types all fit the definition of Equation 3.

3.5 Calculating Top-N Recommendations

Using the same notation as [14], let S be defined as a binary feedback matrix where $s_{u,i} = 1$ if user u liked item i, and 0 otherwise. The user profile of user u is defined as the set of items for which $s_{u,i} = 1$, that is, $\mathcal{I}_u^+ = \{i \in \mathcal{I} | s_{u,i} = 1\}$ (set of items the user likes). Similarly, I_u^- is defined as $I_u^- = \{i \in \mathcal{I} | s_{u,i} = 0\}$ (set of items the user does not like). The user profiles are used to calculate path count vectors $\mathbf{x}_{u,i}$ for every user-item combination in the data using Equation 5. Next, let $I_u^{-*} \subseteq I_u^-$ be defined as a set of size $k * |I_u^+|$ (the number of unliked user items in the training set), where k is a constant. As noted by [14], the size of k has little influence on the results. Therefore, we adopt their setting and set $k = 2$, i.e., we have twice as many unliked items as liked items in the training set. The elements for I_u^{-*} are randomly sampled from I_u^-. The set that can be used to train the RS is then defined as:

$$TR = \bigcup_u \{\langle s_{u,i}, \mathbf{x}_{u,i} \rangle | i \in I_u^+ \cup I_u^{-*}\} \tag{6}$$

Given the set TR, we want to find a ranking function $f : \mathbb{R}^m \to \mathbb{R}$ such that $f(\mathbf{x}_{u,i}) \approx s_{u,i}$.

3.6 Building f Using Random Forests

In order to create a ranking function f, we will make use of random forests [4], which we will briefly discuss here. A random forest is an ensemble method which has shown to be successful on many occasions. As a bagging approach, it is known to reduce variance in comparison to non-bagging solutions such as decision trees. Moreover, they were among the best performing algorithms in the *Yahoo! learning to rank challenge* [6], and they are frequently used successfully in data science competitions on Kaggle (e.g., the Flight Quest competition).

For combining individual trees — we use Classification And Regression Tree (CART) for trees — random forests rely on the concept of bootstrap aggregating, also known as bagging. In this procedure, multiple bootstrap samples with replacements are drawn from the dataset. Each time, a tree model is learned on the bootstrapped sample, and at the end all models are averaged. This reduces overfitting on the train set, because each time the model is learned on a different subset of the data. Additionally, random forests reduce overfitting by limiting the number of possible splits at each split. To do this, each time a split has to be made the model randomly selects a subset of size *mtry* from the independent variables and chooses the optimal split within this subset. This creates an opportunity to find other, maybe less pronounced relations in the data.

Variable Importance.

For each tree in the random forest, a bootstrap sample with replacement is used to train the CART model. This means, that on average one third of the observations in the train set are not used for growing the tree. These observations are defined as "out of bag", or OOB, and provide a good test set for monitoring key statistics.

Variable importance is a difficult concept, because the importance of a variable may be conditional of its interactions with other variables (c.f. [12]). To define variable importance in random forests, let $\mathcal{O}^{(q)}$ be defined as the set of OOB observations for tree q. The variable importance of variable X_m in tree q is then estimated as the increase in the within mean squared error (MSE) after permuting the values in X_j:

$$V^q(X_j) = \frac{\sum_{i \in \mathcal{O}^{(q)}} (y_i - \hat{y}_i^{(q)})^2}{|\mathcal{O}^{(q)}|} - \frac{\sum_{i \in \mathcal{O}^{(q)}} (y_i - \hat{y}_{i,\phi_j}^{(q)})^2}{|\mathcal{O}^{(q)}|}, \tag{7}$$

where $\hat{y}_i^{(q)}$ and $\hat{y}_{i,\phi_j}^{(q)}$ represent the predicted values for observation i before and after permuting the value of $x_{i,j}$, respectively. We permute the values of the path of type j for all observations in $\mathcal{O}^{(q)}$. Lastly, overall variable importance for variable X_j is defined as the increase in within MSE over the whole forest:

$$V(X_j) = \sum_q V^q(X_j) \tag{8}$$

Note that a higher increase in sum of squared errors implicates a higher variable importance, and that $V^q(X_j) = 0$ if variable X_j does not appear in tree q.

4. EVALUATION

We evaluate the performance of the recommender system using the *One-Plus-Random* methodology [2, 7]. Let T be the set of all user-item ratings. All 5 star ratings are considered to be positive feedback, while all other ratings are regarded as unobserved, i.e., unliked. First, the positive ratings from T are split into a train set and a test set, denoted T_{train} and T_{test}. Specifically, we select 10 positive feedback observations from T to be in T_{test} and put all other observations in T_{train}. The observations in T_{train} are used to create user profiles of a predefined size of at most m. Here, m is the maximum profile size, because there can be users with less than m positive ratings in T_{train}. For every user in T_{train}, we randomly select $k \times m$ irrelevant items for that

user. Together these observation will form the feedback matrix S for the TR set from Equation 6. Since we are only interested in Top-N recommendations, T_{test} only contains positive feedback observations.

After training the model on TR, it is tested by evaluating how good the model can select a relevant item from T_{test} for a specific user when random irrelevant items for that user are added. To do this, we iterate over each user-item combination in T_{test}. Each time, we create a temporary set consisting of the selected item from T_{test}, together with 100 randomly selected irrelevant items for that specific user. These must be items that do not appear in the test set, nor in the user profile of that user. This results in a sample of 101 items, of which we assume that only the test item is relevant (the ratio of relevant to irrelevant items is 1:100). The items are ranked according to their predicted score, and the top-N items are recommended to the user. If the test item is among the recommended items, it is counted as a hit. Performance is measured using recall@N ($R@N$):

$$R@N = \frac{\#relevant\ items\ recommended}{\#relevant\ in\ T_{test}} = \frac{\#hits}{|T_{test}|} \quad (9)$$

where N represents the number of items recommended to the user. This metric measures how much of the relevant items in T_{test} have been recommended to the user. Note that in this test setup, the formula for precision@N can be obtained by dividing Equation 9 by N [7]. Moreover, since there is only one relevant item in every temporary set, precision@N values are not meaningful. Therefore, we chose not to report precision@N values.

Lastly, it is worth mentioning that this methodology tends to underestimate the performance of the recommender system, because some of the unrated items might actually be relevant to the user [7].

4.1 Data

We start with the `MovieLens 1M` dataset[1], which contains 1,000,000 ratings on a scale of 1 (terrible) to 5 (amazing) from 6,040 users on 3,952 movies, where each user has at least 20 ratings.

The second source of data is a subset of the `LOD` cloud called `DBPedia`. To link `DBPedia` with the `MovieLens` dataset we proceed as follows. First, we retrieved all movies with corresponding release years from `DBPedia`. We considered English titles only and filtered out movies of which the release year was unknown. Secondly, for each movie in the `MovieLens` dataset we compared the title and the release year with the movies from `DBPedia`. If there was a match, we replaced the movie ID with the `DBPedia` URI. In total, we were able to successfully match 2,056 movies and disregarded all other movies. Additionally, we only considered users with at least 15 positive ratings. A positive rating is defined as a rating of 5, while all other ratings are considered to be negative (irrelevant) ratings. The rationale behind this approach is that we only want to recommend items of which we think the user will like them a lot. By defining only 5 star ratings as positive we can assume that all these movies were indeed perceived as very good. This left 104,196 positive ratings from 2,731 users, with on average 38 positive ratings per user. For each user, we selected 10 positive ratings to be in the test set T_{test}, and selected m of the other positive ratings for the train set T_{train}. Two profile sizes that we will focus on are $m = 5$ and $m = 50$, representing two situations, one with little information about the users and one with a lot of information about the users. As mentioned before, m specifies the maximum profile size, because there might be users with less than m positive ratings in T_{train}.

Recommending the most popular items should give relatively good results. However, recommending the most popular items is trivial and does not add much value, because trivial recommendations are not interesting to users (as they already were going to see them anyway) nor to content providers (as these movies are popular already and do not need recommendation). Similar to [7], we will therefore not only evaluate performance on the total test set, but also on the test set without the 3.7% most popular movies (i.e., 76 movies). This part is referred to as the long-tail of the data.

After the initial movie matching, we queried `DBPedia` for more RDF triples containing the matched movie URIs as either subject or object. We considered RDF triples that are relevant to the movie domain and all `dcterms:subject` triples linking to the movies in the dataset. The latter link movies to their category, and is also used by Wikipedia to improve its structure, by grouping pages with similar subjects.

In the information schema we defined three node types: User, Movie, and Entity. The former two are self explanatory, the latter encompasses all actors, directors, subjects, writers, producers, music composers, cinematographers, editors, and narrators.

4.2 Defining Schemas

We define three different network schemas to test our methodology. First, we define a schema where we ignore the `LOD` cloud, hence we only consider collaborative links between users. This schema is referred to as CF (Collaborative Filtering). The second schema we consider is one where all direct paths are possible, referred to as ADP (All Direct Paths). This is a setup that closely resembles the approach of [14]. However, there are two key differences with their approach. First, we only consider paths up to length 3, whereas they considered all paths up to length 4. We chose to restrict the length to 3 due to computational limitations. Secondly, we only consider direct paths, whereas they also considered cross paths. With direct paths we mean that the link type from Movie to Entity should be the same as the link type from Entity to Movie. In other words, the path from Movie to Movie should be symmetric. An example of a cross path would be the scenario where an entity (e.g., person) starred in one movie, and directed on another movie:

$$\texttt{User} \xrightarrow{likes} \texttt{Movie} \xrightarrow{starring} \texttt{Entity} \xrightarrow{director} \texttt{Movie}.$$

We believe the effect of eliminating cross paths to be minimal, due to the limited presence of cross paths in the graph.

The last information schema we consider is a variant of the ADP schema, where we only consider paths that have a high variable importance score from the random forests trained on the ADP information network. We name this schema as Albatross, referring to the remarkable efficiency of the bird. The rationale behind this approach is that some meta paths are less important than other meta paths. For example, we believe that a user is more likely to like a movie because of its cast, rather than who composed the music for that movie or where the movie was released. In fact, some meta paths

[1] http://grouplens.org/datasets/movielens/

might not add semantics at all, but rather add noise to the data. For example, the release location of the movie tells us very little about the content of the movie and therefore adds little to no semantics to our data. Eliminating these paths based on variable importance scores should therefore be beneficial to the accuracy of the RS. In order to select the most valuable paths from the information network, we evaluate the variable importance scores from the random forest trained with different settings of *mtry*, which defines the number of random features selected by the random forest for each split. We report the variable importance scores for the scenario where *mtry* is equal the square root of the number of features (we have 14 features), as proposed by [4].

4.3 Variable Importance & Path Type Selection

We evaluate the variable importance scores of the random forests trained on small profiles (i.e., $m = 5$) and large profiles (i.e., $m = 50$), respectively. For $m = 5$ (Figure 3(a)), we find that the collaborative path type *likes* is among the most important variables. Also, *director* and *subject* are among the path types with high variable importance scores. The remainder of the path types can be divided into two groups. One group with path types which clearly do not add any semantics to the graph (*basedOn*, *releaseLocation*, *narrator*, *previousWork*, and *subsequentWork*). These variables have very low variable importance scores and are therefore not selected for the Albatross schema (the schema containing only the important paths). The other group consists of path types with all similar variable importance scores (*cinematography*, *producer*, *musicComposer*, *starring*, *writer*, and *editing*). We decided to include these variables in the Albatross schema, as there exists some intuitive explanation of why the information might help recommendations. For the final Albatross schema at $m = 5$, we include *likes*, *director*, *subject*, *cinematography*, *producer*, *musicComposer*, *starring*, *writer*, and *editing*.

For $m = 50$ (Figure 3(b)), we find that some of the same path types show up at the top. Again, *director* shows to be a path type with high variable importance. This makes sense as the director has a significant influence on the outcome of a movie. For example, consider directors as Steven Spielberg, Christopher Nolan, Quentin Tarantino, and Martin Scorsese, who are famous for their talent for creating great movies. Additionally, we find that *producer*, *writer*, and *starring* have high variable importance scores. This can be explained following a similar reasoning as for *director*. Two path types that we believe are in a gray area when it comes to variable importance are *likes* and *cinematography*. It stands out that in contrast to the $m = 5$ scenario, the *likes* path type is not clearly the top variable in terms of variable importance, but rather comes after some of the LOD path types. We decided to still include the variable in the Albatross scenario for $m = 50$, because of the clear intuitive explanation of why this path type is important. The same does not hold for *cinematography*, however, which is why we exclude it from the Albatross schema. For the final Albatross schema at $m = 50$, we include *director*, *writer*, *producer*, *starring*, and *likes*.

4.4 Recall@N : All Items

In this section we report Recall@N scores for the RSs trained on profile sizes $m = 5$ and $m = 50$. As mentioned before, we chose not to report precision@N values, because they differ from recall@N only by a multiplicative term N. The results are shown in Table 1. For $m = 5$, we find that both ADP and Albatross outperform collaborative filtering (CF), where only the users feedback is used as information for the random forest. For N = 5, the CF benchmark has a probability of 26.4% of recommending a relevant item, while the ADP and Albatross RSs have a probability of roughly 36.7% and 36.6%, respectively. Both ADP and Albatross show increasing recall@N after N = 5, while for the benchmark the increase seems to stall at 27.2%. This can be explained by the fact that the collaborative information about the users is very limited at $m = 5$. In this scenario, the LOD cloud provides useful information about movies for recommendations. The difference in performance between ADP and Albatross is negligible, indicating that the unused paths indeed do not provide useful information, but that they also do not add much noise for $m = 5$.

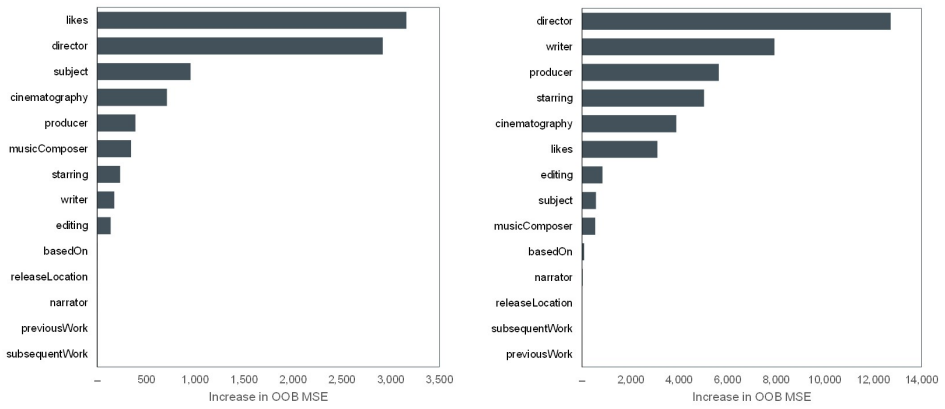
For $m = 50$, we find that the recall@N values are generally higher than for $m = 5$. This makes sense as we now have more information available on the users. One of the effects of a larger user profile is that the CF approach performs really well compared to ADP and Albatross for small N. However, recall that the dataset contains relatively few items with a lot of positive feedback. This probably means that, while the LOD cloud contains useful information on the content of the movies, the popular movies are easily identified by the collaborative information in the dataset, resulting in high recall@N scores. This will be further investigated when we test the RSs on the long tail of the data.

The benefit of path selection is shown by the fact that recall@N is higher for Albatross than for ADP. The difference in performance between ADP and Albatross is highest for N = 10, where recall@N equals 56.3% for ADP and 59.7% for Albatross (difference = 3.4%). This supports our hypothesis that some paths only add noise and should therefore be ignored in the information network.

4.5 Recall@N : Long Tail Items

In order to test the sensitivity of the RSs to the availability of popular items, we test the RSs trained on both profile sizes also on the long-tail of the test set T_{test}. The results are reported in Table 2. As expected, the performance of all RSs are lower compared to the all items test set. This makes sense, because movies which are very popular have a higher chance of being liked by any user. For $m = 5$, we find that the performance of the CF RS is worst, followed by ADP. Albatross performs best in this scenario, but the difference with ADP is again small. Compared to the all items test set, however, the differences seems more pronounced. Specifically, CF sees the largest drop in performance with a decrease in recall@N of roughly 21%. In contrast, for the RSs exploiting the LOD cloud the decrease in recall@N is roughly 18% and 15% for ADP and Albatross, respectively. This strengthens the arguments both for the use of the LOD cloud, as well as the use of path selection. Also, it suggests that the use of path selection becomes more useful when there is not a clear group with very popular items.

For $m = 50$, we find that the Albatross network schema provides the best results. For N = 5 and N = 10, both ADP and Albatross outperform the CF schema. For higher values of N, Albatross also outperforms CF, while ADP is outperformed by CF. This shows the usefulness of path selection.



(a) $m = 5$ (b) $m = 50$

Figure 3: Variable importance scores for all considered path types with mtry = $\sqrt{w}$, where w equals the number of variables (14)

Table 1: Recall@N scores for CF, ADP, and Albatross at different profile sizes; all movies

	$m = 5$			$m = 50$		
N	CF	ADP	Albatross	CF	ADP	Albatross
5	0.264	0.367	0.366	0.518	0.359	0.378
10	0.272	0.511	0.513	0.696	0.563	0.597
15	0.272	0.615	0.616	0.794	0.699	0.731
20	0.272	0.664	0.662	0.854	0.787	0.812
25	0.272	0.682	0.681	0.894	0.851	0.864

Specifically, where ADP is outperformed by CF, Albatross keeps on outperforming CF. These results are also in line with our previous hypothesis that popular movies are easily identified using CF. In this scenario, where the most popular movies have been removed from the test set, the performance of CF is dramatically reduced for low values of N.

4.6 Variable Importance vs. Recall@N

Lastly, we compare the variable importance results with the recall@N scores for both user profile sizes. For $m = 5$, we found that *likes* had the highest variable importance score. It can be seen, however, that for this profile size the CF RS algorithm performed worst, indicating that collaborative information alone is not sufficient for the best recommendations. We believe this can be explained by the fact that there exist more complex relations in the dataset, which are picked up by the random forest. For $m = 5$, for example, this might suggest that *likes* might indeed be an important path type, given that a movie also has the same director or subject.

For $m = 50$, we found that the path types from the `LOD` cloud became more important in terms of variable importance, while the CF performance was good in the all items test set. This can be explained by the fact that the random forests learns from bootstrapped samples, i.e., samples with not necessarily all popular items. In this scenario, the `LOD` path types become more important, which is also confirmed by tests on the long tail of the data. In other words, using only collaborative information tends to result in the recommendation of popular items, while the use of the `LOD` cloud provides more hybrid recommendations (popular and long tail items).

5. CONCLUSIONS

We develop a semantically enhanced, top-N recommender system based on `SPrank` [14], exploiting a subset of the `LOD` cloud called `DBPedia`. In contrast to [14], we define an information network where only predefined paths can be found. By implementing a matrix oriented path finding algorithm [20], we find the number of paths from users to items fol-

Table 2: Recall@N scores for CF, ADP, and Albatross at different profile sizes; long tail

	$m = 5$			$m = 50$		
N	CF	ADP	Albatross	CF	ADP	Albatross
5	0.055	0.155	0.156	0.103	0.205	0.206
10	0.061	0.310	0.316	0.344	0.359	0.395
15	0.061	0.430	0.441	0.544	0.512	0.562
20	0.061	0.488	0.503	0.677	0.634	0.682
25	0.061	0.509	0.528	0.767	0.727	0.768

lowing specific path types and we use this information in a random forest to learn a ranking function. We use variable importance scores to select only the most useful path types, thereby removing path types that only add noise to the data. The RS is evaluated under different scenarios using an empirical dataset consisting of positive feedback on movies. Specifically, we evaluate the results for different profile sizes, as well when the most popular movies have been removed from the test data.

Variable importance scores from the random forest models suggest that *likes*, *director*, and *subject* are among the most important path types when user profiles were small. For larger user profiles, the variable importance scores indicate that *director*, *producer*, *writer*, and *starring* are important path types.

We find two scenarios for which the `LOD` cloud is most beneficial to the RS's accuracy. First, we find that accuracy is improved by exploiting the `LOD` cloud in the scenario of small user profiles. Secondly, we find that when there is a lack of very popular items, exploiting the `LOD` cloud also improves the RS's accuracy.

When the most popular items are removed, the Albatross schema gives the best performance in terms of recall@N for both large and small user profiles. When tested on the all items test set, we find that for small user profiles, using the `LOD` cloud is useful, yet we cannot show the importance of path type selection. For large user profiles, pure collaborative filtering performs similar to the `LOD` based approaches, but Albatross outperforms ADP, suggesting the usefulness of path selection. We argue that the collaborative filtering approach performs well in this case, because there are a lot of popular items which are easily identified via the *likes* paths. This puts the `LOD` based methods at a disadvantage, because they tend to recommend more long tail items.

The results from this paper can be used to improve other state-of-the-art RSs. For example, the concept of only searching for predefined paths could be implemented for other domains than the movie domain. Also, the proposed methodology could improve recommender systems by providing users with insights into why certain items are recommended.

6. REFERENCES

[1] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17(6):734–749, 2005.

[2] A. Bellogin, P. Castells, and I. Cantador. Precision-oriented evaluation of recommender systems: an algorithmic comparison. In *Fifth ACM Conference on Recommender systems (RecSys 2011)*, pages 333–336. ACM, 2011.

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

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

[5] R. Burke. Hybrid recommender systems: Survey and experiments. *User modeling and user-adapted interaction*, 12(4):331–370, 2002.

[6] O. Chapelle and Y. Chang. Yahoo! learning to rank challenge overview. In *Yahoo! Learning to Rank Challenge*, pages 1–24, 2011.

[7] P. Cremonesi, Y. Koren, and R. Turrin. Performance of recommender algorithms on top-n recommendation tasks. In *Fourth ACM Conference on Recommender systems (RecSys 2010)*, pages 39–46. ACM, 2010.

[8] T. Di Noia, R. Mirizzi, V. C. Ostuni, D. Romito, and M. Zanker. Linked open data to support content-based recommender systems. In *8th International Conference on Semantic Systems (I-SEMANTICS 2012)*, pages 1–8. ACM, 2012.

[9] I. Fernández-Tobías, I. Cantador, M. Kaminskas, and F. Ricci. A generic semantic-based framework for cross-domain recommendation. In *2nd International Workshop on Information Heterogeneity and Fusion in Recommender Systems (HetRec 2011)*, pages 25–32. ACM, 2011.

[10] B. Heitmann and C. Hayes. Using linked data to build open, collaborative recommender systems. In *AAAI Spring Symposium: Linked Data Meets Artificial Intelligence*, pages 76–81. AAAI, 2010.

[11] L. Kim. Here's what happens in 60 seconds on the internet. https://smallbiztrends.com/2015/12/60-seconds-on-the-internet.html, 2015.

[12] A. Liaw and M. Wiener. Classification and regression by randomforest. *R news*, 2(3):18–22, 2002.

[13] X. Ning and G. Karypis. Slim: Sparse linear methods for top-n recommender systems. In *IEEE 11th International Conference on Data Mining (ICDM 2011)*, pages 497–506. IEEE, 2011.

[14] 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)*, pages 85–92. ACM, 2013.

[15] A. Passant. dbrec – music recommendations using dbpedia. In *9th International Semantic Web Conference (ISWC 2010)*, pages 209–224. Springer, 2010.

[16] A. Passant. Measuring semantic distance on linking data and using it for resources recommendations. In *AAAI Spring Symposium: Linked Data Meets Artificial Intelligence*, pages 93–98. AAAI, 2010.

[17] M. J. Pazzani and D. Billsus. Learning and revising user profiles: The identification of interesting web sites. *Machine Learning*, 27(3):313–331, 1997.

[18] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *Twenty-fifth Conference on Uncertainty in Artificial Intelligence (UAI 2009)*, pages 452–461. AUAI Press, 2009.

[19] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl. Grouplens: an open architecture for collaborative filtering of netnews. In *1994 ACM Conference on Computer Supported Cooperative Work (CSCW 1994)*, pages 175–186. ACM, 1994.

[20] Y. Sun, J. Han, X. Yan, P. S. Yu, and T. Wu. Pathsim: Meta path-based top-k similarity search in heterogeneous information networks. *37th International Conference on Very Large Data Bases (VLDB 2011)*, 2011.