ItemRank: A Random-Walk Based Scoring Algorithm for Recommender Engines

Marco Gori, Augusto Pucci
Dipartimento di Ingegneria dell'Informazione
University of Siena
Via Roma, 56. Siena, Italy
{marco,augusto}@dii.unisi.it

Abstract

Recommender systems are an emerging technology that helps consumers to find interesting products. A recommender system makes personalized product suggestions by extracting knowledge from the previous users interactions. In this paper, we present "ItemRank", a random–walk based scoring algorithm, which can be used to rank products according to expected user preferences, in order to recommend top–rank items to potentially interested users. We tested our algorithm on a standard database, the MovieLens data set, which contains data collected from a popular recommender system on movies, that has been widely exploited as a benchmark for evaluating recently proposed approaches to recommender system (e.g. [Fouss *et al.*, 2005; Sarwar *et al.*, 2002]). We compared ItemRank with other state-of-the-art ranking techniques (in particular the algorithms described in [Fouss *et al.*, 2005]). Our experiments show that ItemRank performs better than the other algorithms we compared to and, at the same time, it is less complex than other proposed algorithms with respect to memory usage and computational cost too.

1 Introduction

A recommender system makes personalized product suggestions by extracting knowledge from the previous user interactions with the system. Such services are particularly useful in the modern electronic marketplace which offers an unprecedented range of products. In fact a recommender system represents an added value both for consumers, who can easily find products they really like, and for sellers, who can focus their offers and advertising efforts. Several recommender systems have been developed that cope with different products, e.g. MovieLens for movies (see [Sarwar *et al.*, 2001]), GroupLens for usenet news [Miller *et al.*, 2002], Ringo for music [Shardanand and Maes, 1995], Jester for jokes [Goldberg *et al.*, 2001] and many other (see e.g. [Schafer *et al.*, 2001] for a review). A recommender system constructs a user profile on the basis of explicit or implicit interactions of the user with the system. The profile is used to find products to recommend to the user. In the simplest approach, the profile is constructed using only features that are related to the user under evaluation and to the products he/she has already considered. In those cases, the profile consists of a parametric model that is adapted according to the customer's behavior. Key issues of collaborative filtering approach are scalability and quality of the results. In fact, real life large–scale E–commerce applications must efficiently cope with hundreds of thousands of users. Moreover, the accuracy of the recommendation is crucial in order to offer a service that is appreciated and used by customers. In this paper, we present "ItemRank", a random–walk based scoring algorithm, which can be used to rank products according to expected user preferences, in order to recommend top–rank items to potentially interested users. We tested our algorithm on a popular database, the MovieLens dataset[1] by the GroupLens Research group at University of Minnesota and we compared ItemRank with other state-of-the-art ranking techniques (in particular the algorithms described in [Fouss *et al.*, 2005]). This database contains data collected from a popular recommender system on movies that has been widely exploited as a benchmark for evaluating recently proposed approaches to recommender system (e.g. [Fouss *et al.*, 2005; Sarwar *et al.*, 2002]). The schema of such archive resembles the structure of the data of many other collaborative filtering applications. Our experiments show that ItemRank performs better than the other algorithms we compared to and, at the same time, it is less complex than other proposed algorithms with respect to memory usage and computational cost too. The paper is organized as follows. In the next subsection (1.1) we review the related literature with a special focus on other graph based similarity measure and scoring algorithms applied to recommender systems. Section 2 describes the MovieLens data set (in subsection 2.1) and illustrates the data model we adopted (in subsection 2.2). Section 3 discusses ItemRank algorithm in details and we address ItemRank algorithm complexity issues in subsection 3.1. Section 4 contains the details of the experimentation, while Section 5 draws some conclusions and addresses future aspects of this research.

[1] http://www.movielens.umn.edu

1.1 Related Work

Many different recommending algorithms have been proposed in literature, for example there are techniques based on Bayesian networks [Breese *et al.*, 1998], Support Vector Machines [Grcar *et al.*, 2005] and factor analysis [Canny, 2002]. The most successful and well–known approach to recommender system design is based on collaborative filtering [Sarwar *et al.*, 2001; Shardanand and Maes, 1995]. In collaborative filtering, each user collaborates with others to establish the quality of products by providing his/her opinion on a set of products. Also, a similarity measure between users is defined by comparing the profiles of different users. In order to suggest a product to an "active user", the recommender system selects the items among those scored by similar customers. The similarity measure is often computed using the Pearson–r correlation coefficient between users (e.g. in [Sarwar *et al.*, 2001]). Recently a graph based approach has been proposed in [Fouss *et al.*, 2005]. Fouss et al. compared different scoring algorithm to compute a preference ranking of products (in that case movies) to suggest to a group of users. In this paper the problem has been modeled as a bipartite graph, where nodes are users (**people_node**) and movies (**movie_node**), and there is a link connecting a people_node u_i to a movie_node m_j if and only if u_i watched movie m_j, in this case arcs are undirected and can be weighted according to user preferences expressed about watched movies. Authors tested many different algorithms using a wide range of similarity measures in order to rank movies according to user preferences, some of the most interesting methods are:

Average Commute Time (CT). This is a distance measure between a pair of nodes i and j in a graph, we denote it as $n(i, j)$, it is defined as the average number of steps that a random walker[2] going across a given graph, starting in the state corresponding to node i, will take to enter state j for the first time and go back to i. If we measure this distance between people and movie nodes in the given bipartite graph, we can use this score to perform the movie ranking.

Principal Component Analysis based on Euclidean Commute Time Distance (PCA CT). From the eigenvector decomposition of $\mathbf{L}^+$, that is the pseudoinverse of the Laplacian matrix ($\mathbf{L}$) corresponding to the graph, it is possible to map nodes into a new Euclidean space that preserves the Euclidean Commute Time Distance, it is also possible to project to a m-dimensional subspace by performing a PCA and keeping a given number of principal components. Then distances computed between nodes in the reduced space can be used to rank the movies for each person.

Pseudoinverse of the Laplacian Matrix ($\mathbf{L}^+$). Matrix $\mathbf{L}^+$ is the matrix containing the inner products of the node vectors in the Euclidean space where the nodes are exactly separated by the ECTD, so $\mathbf{l}^+_{i,j}$ can be used as the similarity measure between node i and j, in order to rank movies according to their similarity with the person.

In literature there are many other examples of algorithms using graphical structures in order to discover relationships between items. Chebotarev and Shamis proposed in [Chebotarev and Shamis, 1997] and [Chebotarev and Shamis, 1998] a similarity measure between nodes of a graph integrating indirect paths, based on the matrix-forest theorem. Similarity measures based on random-walk models have been considered in [White and Smyth, 2003], where average first-passage time has been used as a similarity measure between nodes. In collaborative recommendation field is also interesting to consider different metrics described in [Brand, 2005].

[2] see [Norris, 1997] for more details

2 The Problem

Formally, a recommender system deals with a set of users u_i, $i = 1, \ldots, U_n$ and a set of products p_j, $j = 1, \ldots, P_n$, and its goal consists of computing, for each pair: u_i, p_j, a score $\hat{r}_{i,j}$ that measures the expected interest of users u_i for product p_j on the basis of a knowledge base containing a set of preferences expressed by some users about products. So we need a scoring algorithm to rank products/items for every given user according to its expected preferences, then a recommender system will suggest to a user top-ranked items with respect to personalized ordering. In this section we present the data model we adopted and MovieLens data set, that is a widely used benchmark to evaluate scoring algorithms applied to recommender systems. Our choice with respect to the data model and the data set is not restrictive since it reflect a very common scenario while dealing with recommender systems. In the following we will indifferently make use of terms such as *item*, *product* and *movie* depending on the context, but obviously the proposed algorithm is a general purpose scoring algorithm and it does not matter which kind of items we are ranking in a particular scenario, moreover we will also use the notation m_j to refer a product p_j in the particular case of movies to be ranked.

2.1 MovieLens Data Set

MovieLens site has over $50,000$ users who have expressed opinions on more than $3,000$ different movies. The MovieLens dataset is a standard dataset constructed from the homonym site archive, by considering only users who rated 20 or more movies, in order to achieve a greater reliability for user profiling. The dataset contains over $100,000$ ratings from 943 users for $1,682$ movies. Every opinion is represented using a tuple: $t_{i,j} = (u_i, m_j, r_{i,j})$, where $t_{i,j}$ is the considered tuple, $u_i \in \mathcal{U}$ is an user, $m_j \in \mathcal{M}$ is a movie, and $r_{i,j}$ is a integer score between 1 (bad movie) and 5 (good movie). The database provides a set of features characterizing users and movies which include: the category of the movie, the age, gender, and occupation of the user, and so on. The dataset comes with five predefined splitting, each uses 80% of the ratings for the training set and 20% for the test set (as described in [Sarwar *et al.*, 2002]). For every standard splitting we call $\mathcal{L}$ and $\mathcal{T}$ respectively the set of tuples used for training and for testing, moreover we refer the set of movies in the training set rated by user u_i as $\mathcal{L}_{u_i}$ and we write $\mathcal{T}_{u_i}$ for movies in the test set. More formally: $\mathcal{L}_{u_i} = \{t_{k,j} \in \mathcal{L} : k = i\}$ and $\mathcal{T}_{u_i} = \{t_{k,j} \in \mathcal{T} : k = i\}$.

2.2 Data Model: Correlation Graph

Even from a superficial analysis of the proposed problem, it seems to be clear that there is a different correlation degree between movies, if we could exploit this information from the training set then it would be quite easy to compute user dependent preferences. We define $\mathcal{U}_{i,j} \subseteq \mathcal{U}$ the set of users who watched (according to the training set) both movie m_i and m_j, so:

$$\mathcal{U}_{i,j} = \begin{cases} \{u_k : (t_{k,i} \in \mathcal{L}_{u_k}) \wedge (t_{k,j} \in \mathcal{L}_{u_k})\} & \text{if } i \neq j \\ \emptyset & \text{if } i = j \end{cases}$$

Now we compute the $(|M| \times |M|)$ matrix containing the number of users who watched each pair of movies:

$$\tilde{\mathcal{C}}_{i,j} = |\mathcal{U}_{i,j}|$$

where $|\cdot|$ denotes the cardinality of a set, obviously $\forall i, \tilde{\mathcal{C}}_{i,i} = 0$ and $\tilde{\mathcal{C}}$ is a symmetric matrix. We normalize matrix $\tilde{\mathcal{C}}$ in order to obtain a stochastic matrix $\mathcal{C}_{i,j} = \frac{\tilde{\mathcal{C}}_{i,j}}{\omega_j}$ where ω_j is the sum of entries in $j-th$ column of $\tilde{\mathcal{C}}$. $\mathcal{C}$ is the *Correlation Matrix*, every entry contains the correlation index between movie pairs. The Correlation Matrix can be also considered as a weighted connectivity matrix for the *Correlation Graph* $\mathcal{G}_\mathcal{C}$. Nodes in graph $\mathcal{G}_\mathcal{C}$ correspond to movies in $\mathcal{M}$ and there will be an edge (m_i, m_j) if and only if $\mathcal{C}_{i,j} > 0$. Moreover the weight associated to link (m_i, m_j) will be $\mathcal{C}_{i,j}$, note that while $\tilde{\mathcal{C}}$ is symmetrical, $\mathcal{C}$ is not, so the weight associated to (m_i, m_j) can differ from (m_j, m_i) weight. The Correlation Graph is a valuable graphical model useful to exploit correlation between movies, weights associated to links provide an approximate measure of movie/movie relative correlation, according to information extracted from ratings expressed by users in the training set.

3 ItemRank Algorithm

The idea underlying the ItemRank algorithm is that we can use the model expressed by the Correlation Graph to forecast user preferences. For every user in the training set we know the ratings he assigned to a certain number of movies, that is $\mathcal{L}_{u_i}$, so, thanks to the graph $\mathcal{G}_\mathcal{C}$ we can "spread" user preferences through the Correlation Graph. Obviously we have to properly control the *preference flow* in order to transfer high score values to movies that are strongly related to movies with good ratings. The spreading algorithm we apply has to possess two key properties: propagation and attenuation. These properties reflect two key assumptions. First of all if a movie m_k is related to one or more good movies, with respect to a given user u_i, then movie m_k will also be a good suggestion for user u_i , if we analyse the Correlation Graph we can easily discover relationships between movies and also the strength of these connections, that is the weight associated to every link connecting two movies. The second important factor we have to take into account is attenuation. Good movies have to transfer their positive influence through the Correlation Graph, but this effect decrease its power if we move further and further away from good movies, moreover if a good movie m_i is connected to two or more nodes, these have to share the boosting effect from m_i according to the weights of their connections as computed in matrix $\mathcal{C}$. PageRank algorithm (see [Page *et al.*, 1998]) has both propagation and attenuation properties we need, furthermore thanks to significant research efforts we can compute PageRank in a very efficient way (see [Kamvar *et al.*, 2003a]). Consider a generic graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of nodes connected by directed links in $\mathcal{E}$, the classic PageRank algorithm computes an importance score $PR(n)$ for every node $n \in \mathcal{V}$ according to graph connectivity: a node will be important if it is connected to important nodes with a low out-degree. So the PageRank score for node n is defined as:

$$PR(n) = \alpha \cdot \sum_{q:(q,n)\in\mathcal{E}} \frac{PR(q)}{\omega_q} + (1-\alpha) \cdot \frac{1}{|\mathcal{V}|} \quad (1)$$

where ω_q is the out-degree of node q, α is a decay factor[3]. The equivalent matrix form of equation 1 is:

$$\mathbf{PR} = \alpha \cdot \mathbf{C} \cdot \mathbf{PR} + (1-\alpha) \cdot \frac{1}{|\mathcal{V}|} \cdot \mathbf{1}_{|\mathcal{V}|} \quad (2)$$

where $\mathbf{C}$ is the normalized connectivity matrix for graph $\mathcal{G}$ and $\mathbf{1}_{|\mathcal{V}|}$ is a $|\mathcal{V}|$ long vector of ones. PageRank can also be computed iterating equation 2, for example by applying the Jacobi method [Golub and Loan, 1996], even if iteration should be run until PageRank values convergence, we can also use a fixed number I of iterations. Classic PageRank can be extended by generalizing equation 2:

$$\mathbf{PR} = \alpha \cdot \mathbf{M} \cdot \mathbf{PR} + (1-\alpha) \cdot \mathbf{d} \quad (3)$$

where $\mathbf{M}$ is a stochastic matrix, its non-negative entries has to sum up to 1 for every column, and vector $\mathbf{d}$ has non-negative entries summing up to 1. Vector $\mathbf{d}$ can be tuned in order to bias the PageRank by boosting nodes corresponding to high value entries and matrix $\mathbf{M}$ controls the propagation and attenuation mode. Biased PageRank has been analysed in [Langville and Meyer, 2003] and custom static score distribution vectors $\mathbf{d}$ have been applied to compute topic-sensitive PageRank [Haveliwala, 2002], reputation of a node in a peer-to-peer network [Kamvar *et al.*, 2003b] and for combating web spam [Gyongyi *et al.*, 2004]. We present the *ItemRank* algorithm, that is a biased version of PageRank designed to be applied to a recommender system. ItemRank equation can be easily derived from equation 3. We use graph $\mathcal{G}_\mathcal{C}$ to compute a ItemRank value IR_{u_i} for every movie node and for every user profile. In this case the stochastic matrix $\mathbf{M}$ will be the Correlation Matrix $\mathcal{C}$ and for every user u_i we compute a different IR_{u_i} by simply choosing a different $\mathbf{d}_{u_i}$ static score distribution vector. The resulting equation is:

$$\mathbf{IR}_{u_i} = \alpha \cdot \mathcal{C} \cdot \mathbf{IR}_{u_i} + (1-\alpha) \cdot \mathbf{d}_{u_i} \quad (4)$$

where $\mathbf{d}_{u_i}$ has been built according to user u_i preferences as recorded in training set $\mathcal{L}_{u_i}$. The unnormalized $\tilde{\mathbf{d}}_{u_i}$, with respect to the $j-th$ component, is defined as:

$$\tilde{d}^j_{u_i} = \begin{cases} 0 & \text{if } t_{i,j} \notin \mathcal{L}_{u_i} \\ r_{i,j} & \text{if } t_{i,j} \in \mathcal{L}_{u_i} \wedge t_{i,j} = (u_i, m_j, r_{i,j}) \end{cases}$$

[3]A common choice for α is 0.85

So the normalized $\mathbf{d}_{u_i}$ vector will simply be $\mathbf{d}_{u_i} = \frac{\tilde{\mathbf{d}}_{u_i}}{|\tilde{\mathbf{d}}_{u_i}|}$. ItemRank, as defined in equation 4, can be computed also iteratively in this way:

$$\begin{cases} \mathbf{IR}_{u_i}(0) = \frac{1}{|\mathcal{M}|} \cdot \mathbf{1}_{|\mathcal{M}|} \\ \mathbf{IR}_{u_i}(t+1) = \alpha \cdot \mathcal{C} \cdot \mathbf{IR}_{u_i}(t) + (1-\alpha) \cdot \mathbf{d}_{u_i} \end{cases} \tag{5}$$

This dynamic system has to be run for every user, luckily it only needs on average about 20 iterations to converge. The interpretation of $\mathbf{IR}_{u_i}$ score vector for user u_i is straightforward, ItemRank scores induce a sorting of movies according to their expected liking for a given user. The higher is the ItemRank for a movie, the higher is the probability that a given user will prefer it to a lower score movie.

3.1 Complexity Issues

ItemRank algorithm results to be very efficient both from computational and memory resource usage point of view. We need to store a $|M|$ nodes graph with a limited number of edges. The interesting fact is that graph $\mathcal{G}_C$ contains edges (m_i, m_j) and (m_j, m_i) if and only if $\exists u_k : t_{k,i} \in L_{u_k} \wedge t_{k,j} \in L_{u_k}$, so no matter the number of users satisfying the previous condition, ratings information will be compressed in just a couple of links anyway. It is interesting to note that the data structure we use scale very well with the increase of the number of users, in fact $\mathcal{G}_C$ node set cardinality is independent from $|\mathcal{U}|$ and also the number of edges tend to increase very slowly after $|\mathcal{U}|$ has exceeded a certain threshold $\bar{U}$. That is a very useful property, because in a real applicative scenario the number of users for a certain e-commerce service and the number of expressed preferences about products will rise much faster than the total amount of offered products. Moreover ItemRank computation is very efficient, thanks to its strong relationship with PageRank algorithm, and we only need about 20 iterations of system 5 for every user in order to rank every movie according to every user taste, so if we have $|\mathcal{U}|$ users we have to run the algorithm $|\mathcal{U}|$ different times. ItemRank is more efficient than similar Random-Walk based approach such as CT and L^+ (already introduced in subsection 1.1, see [Fouss *et al.*, 2005] for details), in fact both CT and L^+ require to handle a graph containing nodes representing users and products and edges referred to user preferences. So in this graph there are $|\mathcal{U}| + |\mathcal{M}|$ nodes and two edges (u_i, m_j),(m_j, u_i) for every opinion $(u_i, m_j, r_{i,j})$, while in the case of ItemRank you have only $|\mathcal{M}|$ nodes and ratings information is compressed. CT is used to rank every movie with respect to every system user, so the average commute time (CT) $n(u_i, m_j)$ referred to any user-movie couple u_i, m_j has to be computed, but $n(u_i, m_j) = m(u_i|m_j) + m(m_j|u_i)$ where $m(u_i|m_j)$ denotes the average first-passage time from node u_i to node m_j. So CT needs $2 \cdot |\mathcal{U}| \cdot |\mathcal{M}|$ average first-passage time computations, while ItemRank has to be applied only $|\mathcal{U}|$ times to rank every movie with respect to its similarity to every user. The situation is similar also if we consider L^+ algorithm, in this case, as stated in [Fouss *et al.*, 2005], the direct computation of the pseudoinverse of the Laplacian matrix L becomes intractable if the number of nodes becomes large (that could easy happen while the number of users increase), some optimized methods to partially overcome these limitations has been proposed in [Brand, 2005].

4 Experimental Results

To evaluate the performances of the ItemRank algorithm, we ran a set of experiments on the MovieLens data set, described in subsection 2.1. The choice of this particular data set is not restrictive, since it is a widely used standard benchmark for recommender system techniques and its structure is typical of the most common applicative scenarios. In fact we can apply ItemRank every time we have a set of users ($\mathcal{U}$) rating a set of items or products ($\mathcal{I}$ that is the generic notation for $\mathcal{M}$), if we can model our recommendation problem this way (or in any equivalent form) it will be possible to use ItemRank to rank items according to user preferences. We chose an experimental setup and performance index that is the same as used in [Fouss *et al.*, 2005], this way we can directly compare our algorithm with some of the most promising scoring algorithms we found in related literature (CT, L^+ and so on), having many points of contact with ItemRank "philosophy". We split MovieLens data set as described in [Sarwar *et al.*, 2002], in order to obtain 5 different subsets, then we applied ItemRank 5 times (5-fold cross validation). Each time, one of the 5 subsets is used as the test set and the remaining 4 sub sets have been merged to form a training set. At the end we computed the average result across all 5 trials. So we have 5 splittings, each uses 80% of the ratings for the training set (that is $80,000$ ratings) and 20% for the test set (the remaining $20,000$ ratings), that is exactly the same way tests have been performed in [Fouss *et al.*, 2005]. The performance index we used is the *degree of agreement* (DOA), which is a variant of Somers'D (see [Siegel and Castellan, 1988] for further details). DOA is a way of measuring how good is an item ranking (movie ranking in MovieLens case) for any given user. To compute DOA for a single user u_i we need to define a set of movies $\mathcal{NW}_{u_i} \subset \mathcal{M}$ that is the set of movies that are not in the training set, nor in the test set for user u_i, so:

$$\mathcal{NW}_{u_i} = \mathcal{M} \setminus (\mathcal{L}_{u_i} \cup \mathcal{T}_{u_i})$$

Now we define the boolean function *check_order* as:

$$\textit{check\_order}_{u_i}(m_j, m_k) = \begin{cases} 1 & \text{if } IR_{u_i}^{m_j} \geq IR_{u_i}^{m_k} \\ 0 & \text{if } IR_{u_i}^{m_j} < IR_{u_i}^{m_k} \end{cases}$$

where $IR_{u_i}^{m_j}$ is the score assigned to movie m_j with respect to user u_i preferences, by the algorithm we are testing. Then we can compute individual DOA for user u_i, that is:

$$\text{DOA}_{u_i} = \frac{\sum_{(j \in \mathcal{T}_{u_i},\, k \in \mathcal{NW}_{u_i})} \textit{check\_order}_{u_i}(m_j, m_k)}{|\mathcal{T}_{u_i}| \cdot |\mathcal{NW}_{u_i}|}$$

So DOA_{u_i} measures for user u_i the percentage of movie pairs ranked in the correct order with respect to the total number of pairs, in fact a good scoring algorithm should rank the movies that have indeed been watched in higher positions than movies that have not been watched. A random ranking produces a degree of agreement of 50%, half of all the pairs are in correct order and the other half in bad order. An ideal ranking correspond to a 100% DOA. Two different global degree of agreement can be computed considering ranking for individual users: Macro-averaged DOA and micro-averaged DOA. The Macro-averaged DOA (or shortly Macro DOA) will be the average of individual degree of agreement for every user, so:

$$\text{Macro DOA} = \frac{\sum_{u_i \in U} \text{DOA}_{u_i}}{|\mathcal{U}|}$$

The micro-averaged DOA (or shortly micro DOA) is the ratio between the number of movie pairs in the right order (for every user) and the total number of movie pairs checked (for every user), so it can be computed as:

$$\text{micro DOA} = \frac{\sum_{u_i \in \mathcal{U}} \left(\sum_{(j \in \mathcal{T}_{u_i},\, k \in \mathcal{NW}_{u_i})} \textit{check\_order}_{u_i}(m_j, m_k) \right)}{\sum_{u_i \in \mathcal{U}} \left(|\mathcal{T}_{u_i}| \cdot |\mathcal{NW}_{u_i}| \right)}$$

Then micro DOA is something like a weighted averaging of individual DOA values. In fact the bigger is set $\mathcal{T}_{u_i}$ for a given user u_i, the more important is the individual DOA_{u_i} contribution to micro DOA global computation. Macro DOA and micro DOA have been evaluated for every experiment we ran. We summarize experimental results in table 1 and 2. In table 1 we compare ItemRank performances to a simplified version of the same algorithm, in order to highlight the importance of the information hidden in the Correlation Matrix $\mathcal{C}$. ItemRank with the binary graph is identical to classical ItemRank (described in section 3) but there is a key difference in the way we build matrix $\mathcal{C}$ (we denote the simplified version as $\mathcal{C}^{bin}$), in this case it is obtained by normalizing a binary version of $\tilde{\mathcal{C}}$ ($\tilde{\mathcal{C}}^{bin}$), so we have: $\mathcal{C}^{bin}_{i,j} = \frac{\tilde{\mathcal{C}}^{bin}_{i,j}}{\omega_j}$ where $\tilde{\mathcal{C}}^{bin}_{i,j}$ can be computed as:

$$\tilde{\mathcal{C}}^{bin}_{i,j} = \begin{cases} 1 & \text{if } \mathcal{U}_{i,j} > 0 \\ 0 & \text{if } \mathcal{U}_{i,j} = 0 \end{cases}$$

In other words if we compute ItemRank with binary graph, we are weighting every correlation edge connecting two items in the same way, no matter the number of co-occurrences in user preference lists for these items, since $\mathcal{C}^{bin}_{i,j}$ correspond to the weight of edge (m_i, m_j) in the Correlation Graph $\mathcal{G}_{\mathcal{C}}$ we use for information propagation.

Table 1 clearly shows the usefulness of a properly weighted Correlation Matrix $\mathcal{C}$ compared to $\mathcal{C}^{bin}$. This table provides both Macro and micro DOA for every split and for ItemRank and its simplified version with binary graph: ItemRank clearly works much better when we use a proper Correlation Matrix. For example, if we look at Macro DOA mean values, ItemRank with Correlation Matrix $\mathcal{C}$ obtain +15.43 points (in %) with respect to $\mathcal{C}^{bin}$ version. These are interesting results because they confirm our main hypothesis: ItemRank algorithm ranks items according to the information extracted from the Correlation Matrix (that is equivalent to the weighted Correlation Graph) and the way we compute $\mathcal{C}$ entries is really able to properly model relationships among evaluated items. Finally table 2 shows a performance comparison among different scoring algorithm applied to MovieLens data set. We briefly described some of these algorithms in subsection 1.1, for further details see [Fouss *et al.*, 2005]. For every tested algorithm we provide Macro DOA index, that has been computed for every technique as the average result across all 5 trials of 5-fold cross-validation. Moreover we provide the difference (in %) with performance obtained by the trivial MaxF algorithm and the standard deviation (STD) of this quantity. MaxF is our baseline for the task, it is a user independent scoring algorithm, it simply ranks the movies by the number of persons who watched them, movies are suggested to each person in order of decreasing popularity. So MaxF produces the same ranking for all the users. ItemRank performs better than any other considered technique obtaining **+3.69** with respect to the baseline and a very good standard deviation (**0.31**). In this test ItemRank also perform better than L^+ algorithm by obtaining a Macro DOA value of **87.76** versus 87.23 for L^+ and also a better standard deviation. In addition it is worth to note that ItemRank is less complex than other proposed algorithms with respect to memory usage and computational cost too, as already argued in subsection 3.1.

5 Conclusions

In this paper, we present a random–walk based scoring algorithm, which can be used to recommend products according to user preferences. We compared our algorithm with other state-of-the-art ranking techniques on MovieLens data set. ItemRank performs better than the other algorithms we compared to and, at the same time, it is less complex than other proposed algorithms with respect to memory usage and computational cost too. Future research topics include the experimentation of the algorithm on different applications. We are now working on a extension of ItemRank. The version presented so far is able to handle the recommendation task as a item scoring/ranking problem. But we can face the problem from the regression point of view too. So we expect ItemRank 2.0 will also be able to produce expected satisfaction prediction for a given recommendation, other than product ranking.

Acknowledgments

Many thanks to Giuliano Testa for running the experiments.

References

[Brand, 2005] M. Brand. A random walks perspective on maximizing satisfaction and profit. In *2005 SIAM International Conference on Data Mining*, 2005.

[Breese *et al.*, 1998] J. S. Breese, D. Heckerman, and C. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *14th Conference on Uncertainty in Artificial Intelligence (UAI-98)*, pages 43–52, July 1998.

[Canny, 2002] J. Canny. Collaborative filtering with privacy via factor analysis. In *IEEE Conference on Security and Privacy*, May 2002.

[Chebotarev and Shamis, 1997] P. Chebotarev and E. Shamis. The matrix-forest theorem and measuring relations in small social groups. *Automation and Remote Control*, 58(9):15051514, 1997.

	ItemRank		ItemRank (binary graph)	
	micro DOA	Macro DOA	micro DOA	Macro DOA
SPLIT 1	87.14	87.73	71.00	72.48
SPLIT 2	86.98	87.61	70.94	72.91
SPLIT 3	87.20	87.69	71.17	72.98
SPLIT 4	87.08	87.47	70.05	71.51
SPLIT 5	86.91	88.28	70.00	71.78
Mean	**87.06**	**87.76**	**70.63**	**72.33**

Table 1: Performance comparison between ItemRank and its simplified version with binary Correlation Graph.

	MaxF	CT	PCA CT	One-way	Return	L^+	ItemRank	Katz	Dijkstra
Macro DOA	84.07	84.09	84.04	84.08	72.63	87.23	**87.76**	85.83	49.96
difference with MaxF (in %)	0	+0.02	-0.03	+0.01	-11.43	+3.16	**+3.69**	+1.76	-34.11
STD of the difference with MaxF	0	0.01	0.76	0.01	1.06	0.84	**0.31**	0.24	1.52

Table 2: Comparison among different scoring algorithm applied to MovieLens data set.

[Chebotarev and Shamis, 1998] P. Chebotarev and E. Shamis. On proximity measures for graph vertices. *Automation and Remote Control*, 59(10):14431459, 1998.

[Fouss *et al.*, 2005] F. Fouss, A. Pirotte, J. M. Renders, and M. Sarens. A novel way of computing dissimilarities between nodes of a graph, with application to collaborative filtering. In *IEEE / WIC / ACM International Joint Conference on Web Intelligence*, pages 550–556, 2005.

[Goldberg *et al.*, 2001] K. Goldberg, T. Roeder, D. Gupta, and C. Perkins. Eigentaste: A constant time collaborative filtering algorithm. *Information Retrieval*, 4(2):133151, 2001.

[Golub and Loan, 1996] G. Golub and C. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, third edition, 1996.

[Grcar *et al.*, 2005] M. Grcar, B. Fortuna, D. Mladenic, and M. Grobelnik. Knn versus svm in the collaborative filtering framework. In *ACM WebKDD 2005 Taming evolving, Expanding and Multi-faceted Web Clickstreams Workshop*, 2005.

[Gyongyi *et al.*, 2004] Z. Gyongyi, H. Garcia-Molina, and J. Pedersen. Combating web spam with trustrank. Technical report, Stanford University, 2004.

[Haveliwala, 2002] T. Haveliwala. Topic-sensitive pagerank. In *Eleventh International Conference on World Wide Web*, 2002.

[Kamvar *et al.*, 2003a] S. Kamvar, T. Haveliwala, C. Manning, and G. Golub. Extrapolation methods for accelerating pagerank computations. In *Twelfth International Conference on World Wide Web*, 2003.

[Kamvar *et al.*, 2003b] S. Kamvar, M. Schlosser, and H. Garcia-Molina. The eigentrust algorithm for reputation management in p2p networks. In *Twelfth International Conference on World Wide Web*, 2003.

[Langville and Meyer, 2003] A. Langville and C. Meyer. Deeper inside pagerank. *Internet Mathematics*, 1(3):335–380, 2003.

[Miller *et al.*, 2002] B. Miller, J. Riedl, and J. Konstan. Grouplens for usenet: Experiences in applying collaborative filtering to a social information system. In C. Leug and D. Fisher, editors, *From Usenet to CoWebs: Interacting with Social Information Spaces*. Springer-Verlag, 2002.

[Norris, 1997] J. Norris. *Markov Chains*. Cambridge University Press, 1997.

[Page *et al.*, 1998] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. *Technical report, Stanford University*, 1998.

[Sarwar *et al.*, 2001] B. M. Sarwar, G. Karypis, J. A. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *10th International World Wide Web Conference (WWW10)*, May 2001.

[Sarwar *et al.*, 2002] B. M. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Recommender systems for large-scale e-commerce: Scalable neighborhood formation using clustering. In *Fifth International Conference on Computer and Information Technology*, 2002.

[Schafer *et al.*, 2001] J. Schafer, J. Konstan, and J. Riedl. Electronic commerce recommender applications. *Journal of Data Mining and Knowledge Discovery*, January 2001.

[Shardanand and Maes, 1995] U. Shardanand and P. Maes. Social information filtering: Algorithms for automating "word of mouth". In *CHI 95*, 1995.

[Siegel and Castellan, 1988] S. Siegel and J. Castellan. *Nonparametric Statistics for the Behavioral Sciences*. McGraw-Hill, 1988.

[White and Smyth, 2003] S. White and P. Smyth. Algorithms for estimating relative importance in networks. In *Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data mining*, page 26627, 2003.