Expert Systems With Applications 79 (2017) 8–19

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

SCoR: A Synthetic Coordinate based Recommender system

Harris Papadakis[a,*], Costas Panagiotakis[b,c], Paraskevi Fragopoulou[a,c]

[a] Department of Informatics Engineering, TEI of Crete, 71004 Heraklion, Crete, Greece
[b] Department of Business Administration, TEI of Crete, 72100 Agios Nikolaos, Crete, Greece
[c] Foundation for Research and Technology-Hellas (FORTH), Institute of Computer Science, 70013 Heraklion, Crete, Greece

A R T I C L E   I N F O

Article history:
Received 3 July 2016
Revised 13 February 2017
Accepted 14 February 2017
Available online 17 February 2017

Keywords:
Recommender systems
Synthetic coordinates
Graph
Vivaldi
Matrix factorization
Netflix

A B S T R A C T

Recommender systems try to predict the preferences of users for specific items, based on an analysis of previous consumer preferences. In this paper, we propose SCoR, a Synthetic Coordinate based Recommendation system which is shown to outperform the most popular algorithmic techniques in the field, approaches like matrix factorization and collaborative filtering. SCoR assigns synthetic coordinates to nodes (users and items), so that the distance between a user and an item provides an accurate prediction of the user's preference for that item. The proposed framework has several benefits. It is parameter free, thus requiring no fine tuning to achieve high performance, and is more resistance to the cold-start problem compared to other algorithms. Furthermore, it provides important annotations of the dataset, such as the physical detection of users and items with common and unique characteristics as well as the identification of outliers. SCoR is compared against nine other state-of-the-art recommender systems, sever of them based on the well known matrix factorization and two on collaborative filtering. The comparison is performed against four real datasets, including a brief version of the dataset used in the well known Netflix challenge. The extensive experiments prove that SCoR outperforms previous techniques while demonstrating its improved stability and high performance.

1. Introduction

With the penetration of Internet access all over the globe, consumers' choices have multiplied exponentially. Even though this allows for a multitude of choices and a wider variety of selection, it has become increasingly difficult to match consumers preferences with the most appropriate products, especially given the diversity of needs between them. Recommender systems (Adomavicius & Tuzhilin, 2005; Park, Kim, Choi, & Kim, 2012) try to amend this situation by analyzing consumer preferences and trying to predict the preference of a user for a new item. Recommender systems have a wide range of applications. One of the well known application is in consumer sites with a vast range of products, in order to provide consumers with targeted information about products that might interest them. Another application is in designing marketing strategies, where recommender systems are used to predict the popularity of products. Recommender systems are also used to provide users with recommendations for other entities than consumer products, such as web pages.

The problem of content recommendation can be described as follows. Given a set U of users, a set I of items (e.g. movies) and a set R of ratings (evaluations) of users for items, we need to estimate (predict) the rating for a user-item pair which is not in R. In order to evaluate such predictions, most systems treat a subset of R as unknown in order to calculate predictions and then evaluate their performance based on the Root Mean Squared Error (RMSE) (Adomavicius & Kwon, 2012; Gunawardana & Shani, 2009; Herlocker, Konstan, Terveen, & Riedl, 2004). RMSE is suitable for recommender systems, because it measures inaccuracies on all ratings, either negative or positive. However, it is most suitable for situations where we do not differentiate between errors like in our experiments (Gunawardana & Shani, 2009). In this work, we have used both RMSE and the absolute recommendation error.

Most recommender systems rely on Collaborative Filtering (CF) (Zhou, Wilkinson, Schreiber, & Pan, 2008), which analyzes previous consumer behaviour and preferences in order to make new predictions. Although such approaches suffer from the cold-start problem (how to make predictions for new users with no or very limited preference history), they have the advantage of making use of preexisting information which is automatically generated as the user accesses and possibly evaluates items. Most CF methods rely on some sort of similarity function either between users or between items. Each predicted recommendation for a single user is calcu-

* Corresponding author.
E-mail addresses: adanar@ie.teicrete.gr (H. Papadakis), cpanag@staff.teicrete.gr (C. Panagiotakis), fragopou@ics.forth.gr (P. Fragopoulou).

http://dx.doi.org/10.1016/j.eswa.2017.02.025
0957-4174/© 2017 Elsevier Ltd. All rights reserved.

lated based on the preference for the same item of other users with high similarity with the said user.

Other recommendation methods rely on Dimensionality Reduction Techniques (Goldberg, Roeder, Gupta, & Perkins, 2001), where hidden variables are introduced to explain the co-occurrence of data. Such systems are based on techniques such as Bayesian clustering, Probabilistic Latent Semantic Analysis and Latent Dirichlet Allocation.

One very successful approach is based on Latent Factor models such as Singular Value Decomposition (SVD) (Gorrell, 2006). Such approaches place both users and items on a single space in such a manner that their position in space reflects the preference relationships between them. In this space, all items can be compared and thus predictions can be extracted. Other systems include Diffusion-based algorithms (Ma, King, & Lyu, 2012) which projects input data to object-object networks where nodes preferred by user are sources and Social Filtering methods that take into account trust relationships between users to increase prediction accuracy are employed.

In our approach, all users and items are placed in a single, multi-dimensional Euclidean space. The Vivaldi synthetic coordinates algorithm (Dabek, Cox, Kaashoek, & Morris, 2004) is then employed to properly place them in points in space so that the distance between any user-item pair is directly respective to the preference of that user for the specific item. Indirect relationships between users who have rated the same item, and items rated by the same user are them accurately reflected. The Euclidean distance between a user and an item of unknown rating (for that user) is used to make the prediction (recommendation).

Our proposal exhibits a variety of strengths, compared to existing approaches. Using extensive experiments we demonstrate that our algorithm provides more accurate predictions, compared to several state-of-the-art algorithms, for large, real-world datasets, while requiring no parameterization. As opposed to several recommender systems found in the literature, the proposed method is parameter free. Additionally, the proposed system provides important dataset annotations, such as the detection of users and items with common and unique characteristics. The algorithm's accuracy is only slightly reduced in cases of very sparse datasets (i.e. containing only users with very few ratings).

The structure of this paper is as follows: Section 2 provides a description of the different approaches used in order to address the content recommendation problem, including the state-of-the-art algorithms used for comparison in the experiments. Section 3 describes in detail our contribution, while annotations of a dataset based on the proposed system are described in Section 4. Section 5 describes the setup of the experiments along with the obtained results. Conclusions and discussion are provided in Section 6.

2. Related work

Recommender systems is a popular research field that has been around for years (Adomavicius & Kwon, 2012). Early works in recommender systems date since the early 1990s (Goldberg, Nichols, Oki, & Terry, 1992). Given the wide range of available data on per case basis, many algorithms have been proposed which try to exploit different kinds of available information, or the same kind of information in a different way. In this section, we describe the most important approaches for recommender system design as well as the state-of-the-art algorithms we compare against.

2.1. Types of recommender systems

One of the main recommender system techniques is similarity-based Collaborative Filtering (Adomavicius & Kwon, 2012; Resnick, Iacovou, Suchak, Bergstrom, & Riedl, 1994). Such algorithms are based on a similarity function which takes into account user preferences and decisions and outputs a similarity degree between two users. In order to calculate a recommendation for item i and user u, the top-K users with the highest similarity to user u, who have rated item i, are selected. The resulting prediction is the (weighted) average of those users' ratings for item i. Similarity metrics include, but are not limited to, the cosine similarity, Euclidean similarity, Pearson correlation etc. Another type of similarity functions exploits the structure of the relationships between users and items. Such functions can be based on the number of commonly rated items between users, or the Katz similarity (Wikipedia, 2014).

Other approaches define a similarity function between items instead of users (Linden, Smith, & York, 2003; Sarwar, Karypis, Konstan, & Riedl, 2001a). Such similarity functions are based on a variety of available information such as item metadata or whether the same users like or disliked certain items. A prediction for an item i and user u is the average of the ratings of user u for items highly similar to item i. Again, similarity functions include the cosine similarity, conditional probability, and Pearson correlation, as well as structure-based similarity methods.

Another important approach for recommender systems is Dimensionality Reduction. Each user or item in the system is expressed with a vector. Each user's vector is the set of his ratings for each item in the system (either rated or not rated). Each item's vector is the set of values given to that item by all users in the system. For a large system, the size of the vectors can be prohibitively large. In addition, the sparsity of these datasets makes it more difficult to find correlations between user-items pairs. For these reasons, other methods have been proposed trying to reduce vectors' dimensions. Common dimensionality reduction techniques include Singular Value Decomposition (Gorrell, 2006), Principal Component Analysis, such as the Eigentaste (Goldberg et al., 2001) algorithm used in the Jester site and dataset (Goldberg, 2003), Probabilistic Latent Semantic Analysis, another form of dimensionality reduction like SVD based on statistical probability theory (Mobasher, Burke, & Sandvig, 2006), and Latent Dirichlet Allocation (Mobasher et al., 2006). Dimensionality Reduction techniques are a type of Matrix Factorization method. Matrix factorization approaches have been quite popular in the field of recommender systems. One such algorithm was the big winner of the Netflix Prize competition in 2006 (Bennett, Lanning, & Netflix, 2007).

Other approaches include Diffusion based methods (Ma et al., 2012), based on projecting the input data to object-object networks. Nodes preferred by user are sources and propagation is used to reach recommended objects.

There also exist several hybrid methods which belong to more than one category. Some of them implement collaborative and content-based methods separately and combine their predictions. Others incorporate content-based characteristics into collaborative approaches or incorporate collaborative characteristics into content-based approaches. Finally, there are approached based on a general unifying model that incorporates both content-based and collaborative characteristics.

Most of the aforementioned approaches rely on a training set of known user preferences in order to be able to make the required predictions. More recent approaches try to enrich this training set with additional metadata, such as the categorization of items into topics, in order to exploit this information and to provide more accurate predictions. An interesting example of such an approach is Carrer-Neto, Hernández-Alcaraz, Valencia-García, and García-Sánchez (2012), which adds semantic information to the training set in order to make better predictions.

2.2. Comparison algorithms

In this section we provide a brief description of the recommender algorithms against which we compare SCoR. Although we included a wide range of algorithms from classic Collaborative Filtering to Matrix Factorization-based, most comparison algorithms are Matrix Factorization, since this is the most recent approach and in general offers more accurate results compared to Collaborative Filtering. In addition, it is more closely related to our approach. These algorithms have all been implemented as part of the Graphchi library.[1] The SGD method (Koren, Bell, & Volinsky, 2009), as all matrix factorization methods, tries to exploit hidden variables and aspects in user selections by expressing the original matrix R of user recommendations as a product of matrices. The training set is used to discover latent aspects in user-item relationships. Then these aspects are used to perform recommendation-predictions for unknown user-item pairs. Each item is decomposed into k aspects (the value of k is set experimentally), with a weight for each aspect, corresponding to its prevalence for that particular item. Similarly, a value for each of the k aspects is computed for each user, indicating its preference for that aspect. This information is then used to compute the desired predictions.

Koren (2008) proposes BIASSGD, a modification of the classic SGD produced by changing the mathematical model to no longer explicitly parameterize the users in the system. This leads to several benefits, such as fewer parameters, faster integration of new users in the system, better explainability of the results and efficient integration of implicit feedback.

SVDPP is essentially the BIASSGD system (Koren, 2008) described above with certain modifications in order to better exploit implicit information in recommender systems data, such as, which items each user has rated (in a true/false fashion instead of a rating-value fashion).

The Alternating Least Squares (ALS) system is presented in Zhou et al. (2008). It tries to alleviate the problem of SVD when there are many missing elements in the rating matrix, i.e. the dataset is sparse. It does so using small random values in place of missing elements. It then employs matrix factorization to express the computed $\widehat{R}$ matrix of user predicted ratings as a product of two matrices.

$$\widehat{R} = U^T \times M \tag{1}$$

M is initialized by assigning the average rating for that item as the first row, and small random numbers for the remaining (missing) entries. Subsequently, it iteratively executes the following two steps: By fixing matrix M, it solves for U, minimizing the sum of squared errors. It then fixes U and solves for M, again minimizing the sum of squared errors. The process terminates when the squared errors sum cannot be further reduced.

The Alternating Least Squares - parallel coordinate descent system (Yu, Hsieh, Si, & Dhillon, 2012) uses ALS combined with parallel coordinate descent in order to minimize the resulting error. This system is similar to the previous one. The main difference is the way equations in the aforementioned iterative two steps are solved. This leads to a faster method for computing the desired predictions. Ultimately, the proposed system has lower time complexity per iteration than ALS and achieves faster and more stable convergence than SGD.

The Restricted Boltzmann machines method (rbm) is analytically described in Hinton (2010). In its conditional form it can be used to model high-dimensional temporal sequences such as video or motion captured data or speech. It is a Markov Chain Monte Carlo method for binary data.

Finally, we included in the experimental evaluation two classic Collaborative Filtering algorithms in order to compare our algorithm against a wide range of recommendation approaches. The first, which is the Personal Mean algorithm (Ekstrand, Riedl, & Konstan, 2011), is based on Eq. (2), where b_u and b_i are user and item baseline predictors, respectively. The second is the User-User algorithm (Sarwar, Karypis, Konstan, & Riedl, 2001b), a classic similarity-based algorithm. In order to provide a recommendation for a certain item i to user u, it takes the average of all ratings for item i by other users, weighted by the similarity of those users to u. Both these algorithms implementations were available in GroupLens' LensKit library.[2]

$$\mu_{u,i} = \mu + b_i + b_u \tag{2}$$

3. The algorithm

3.1. The Vivaldi algorithm

Our proposal is based on the Vivaldi algorithm (Dabek et al., 2004) for Internet latency estimation, which has been modified and extended in order to be used for recommendations. We start with a brief description of the Vivaldi algorithm. Vivaldi is a fully decentralized, light-weight, adaptive synthetic network coordinates algorithm that was initially developed to predict Internet latencies with low error. It uses the Euclidean coordinate system and its associated distance function. Conceptually, Vivaldi simulates a network of physical springs, placing imaginary springs between pairs of network nodes. In a nutshell, each Vivaldi node continuously interacts with only a*small subset* of other nodes, trying to position itself in the Euclidean space so that the Euclidean distance between pairs of nodes matches their measured latency. When the system converges, meaning that each node has obtained its desired position, the Euclidean distance between *any* pair of nodes provides an accurate prediction of their latency.

In more detail, each node x maintains its own coordinates $p(x) \in \Re^n$, the position of node x, a point in the *n-dimensional* Euclidian space. Each node is also aware of a small number of other nodes, with which it directly communicates and makes latency measurements. Vivaldi executes the following steps:

- Initially, all node coordinates are set at random in $\Re^n$.
- Periodically, each node x communicates with another node y (randomly selected from the small set of nodes known to it). Each time node x communicates with node y, it measures its latency $RTT(x, y)$ and learns y's coordinates. Subsequently, node x allows itself to be moved a little by the corresponding imaginary spring connecting it to y. Its position changes a little so as the Euclidean distance of the nodes $d(x, y)$ to better match the measured latency $RTT(x, y)$, according to Eq. (3).

  $$p(x) = p(x) + \delta \cdot (RTT(x, y) - d(x, y)) \cdot \frac{p(x) - p(y)}{d(x, y)} \tag{3}$$

  where $\frac{p(x)-p(y)}{d(x,y)}$ is the unit vector that gives the direction node x should move, and δ controls the method's convergence, since it reflects the fraction of distance node x is allowed to move toward its perfect position. δ can be set proportional to the ratio of the relative error of node x and the sum of relative errors of nodes x and y (Dabek et al., 2004).
- When Vivaldi converges, any two nodes' Euclidean distance matches their latency, even for nodes that did not have any communication during the execution of the algorithm.

Unlike other centralized network coordinate approaches, in Vivaldi each node only maintains knowledge for a handful of other

[1] http://bickson.blogspot.gr/2012/12/collaborative-filtering-with-graphchi.html

[2] http://lenskit.org

nodes, making the algorithm completely distributed in nature. Each node computes and continuously adjusts its coordinates based on measured latencies to a handful of other nodes. At the end of the algorithm, however, the Euclidean distance between *any* pair of nodes, matches their latency. Vivaldi uses a 2+1 dimensional space (the third dimension only taking positive values) to remedy the fact that the triangular inequality does not hold for internet latencies (Zhang & Zhang, 2012). In our recommender system, users, items, and recommendations, form a bipartite graph, thus eliminating the triangular inequality problem since no triangles exist in such a graph.

3.2. Problem formulation

Before we proceed to the description of our algorithm, we first define the recommendation prediction problem in its general form. The input is a list L of triplets in the form (u, i, $r(u, i)$), where:

1. $u \in$ the set $U = \{1, 2, \ldots, N\}$ of unique identifiers of users which have rated items.
2. $i \in$ the set $I = \{1, 2, \ldots, M\}$ of unique identifiers of items that have been rated by users.
3. $r(u, i) \in \mathbb{R}$ denotes the rating (declared preference) of user u for item i.

In real world scenarios, the goal is to calculate unknown recommendations that accurately predict the preference of user u for item i, when u has not yet rated i and thus there is no triplet (u, i, $r(u, i)$) in L, based on the indirect knowledge of exising user-item ratings.

In order to evaluate the prediction accuracy of a recommendation algorithm, the original list L is divided into two parts. The first, which is called the “Training Set” *TS*, is considered to contain the “known” user-item ratings and is used to train the algorithm. The rest of the triplets comprise the, so called, “Validation Set” *VS*, and is used to test the prediction accuracy of the algorithm.

The recommendation algorithm calculates a new $\widehat{r}(u, i)$ value for each (u, i, $r(u, i)$) triplet in *VS*. The goal of the algorithm is the $\widehat{r}(u, i)$ values to converge as closely as possible to the corresponding $r(u, i)$ values. The convergence of the method is usually calculated with the Root Mean Square Error Metric (RMSE):

$$RMSE = \sqrt{E\{(R - \widehat{R})^2\}} \tag{4}$$

where R is the set of r values (user declared ratings) and $\widehat{R}$ is the set ratings produced by the recommendation algorithm for *VS*. Each r value is subtracted from its corresponding $\widehat{r}$ value. The lower the calculated RMSE, the better the prediction accuracy of the algorithm.

3.3. The SCoR algorithm

In this section, we describe the main contribution of this paper, the SCoR recommendation algorithm. As already mentioned, at the core of SCoR lies a modified version of Vivaldi (Dabek et al., 2004). The fact that Vivaldi is able to predict unknown latencies between network nodes using a set of measured ones, constitutes it a prime candidate for recommendation prediction.

In our modified version of Vivaldi, network nodes are replaced by user nodes and item nodes, while latencies are replaced by distances calculated by the ratings of users for items. As a result, a bipartite graph is formed, consisting of users nodes on one side and items nodes on the other (instead of just network nodes). For each (u, i, $r(u, i)$) triplet in the Training set (*TS*), an edge is added between nodes u and i. Each edge is assigned a weight $dd(u, i)$, based on rating $r(u, i)$, which reflects the distance of edge (u, i). A (u, i) pair with high r value (high preference of user u for item i) corresponds to an edge with small distance and vice versa. The smallest rating, minR, is assigned a distance of 100, whereas the highest rating is assigned a distance of 0. Given these values, the distance $dd(u, i)$ is calculated as follows:

$$dd(u, i) = 100 \cdot \left(\frac{maxR - r(u, i)}{maxR - minR} \right) \tag{5}$$

where *minR, maxR* denote the minimum (low preference) and maximum (high preference) ratings, respectively. Thus, latency between network nodes in Vivaldi, is replaced with the distance calculated by the declared rating between the user and the item according to the above equation.

Vivaldi then iteratively updates the positions of the nodes to satisfy the desired distances for all edges. Ideally, if a user u has rated item i with value $r(u, i)$, then after Vivaldi has converged, the distance of nodes u and i should be $dd(u, i)$, calculated according to Eq. (5).

The pseudo-code of the proposed method is given in Algorithm 1. The input to the algorithm is the set of users U, the set of items I and the values *minR, maxR*. The training set *TS* and the validation set *VS* consist of the given recommendations (u, i, $r(u, i)$) $\in$ *TS* and the predicted recommendations $\widehat{r}(u, i)$, with (u, i) $\in$ *VS* (produced by SCoR), respectively.

```
   input : $i \in I, u \in U, (u, i, r(u, i)) \in TS$, $minR, maxR$.
   output: $\widehat{r}(u, i), \tilde{}(u, i) \in VS$
 1 foreach $u \in U$ do
 2     $p(u)$ = random position in $\Re^n$
 3 end
 4 foreach $i \in I$ do
 5     $p(i)$ = random position in $\Re^n$
 6 end
 7 repeat
 8     $(u, i) = getRandomSample(TS)[p(u), p(i)] = Vivaldi(p(u), p(i), r(u, i))$
 9 until $\forall x \in I \cup U$ $p(x)$ is stable ;
10 foreach $(u, i) \in TS$ do
11     $W(u, i) = e^{-0.2 \cdot MSE(u)} \cdot (dd(u, i) - d(u, i))^2$
12 end
13 repeat
14     $(u, i) = getWeightedRandomSample(TS, W)$
15     $[p(u), p(i)] = Vivaldi(p(u), p(i), r(u, i))$
16 until $\forall x \in I \cup U$ $p(x)$ is stable ;
17 foreach $(u, i) \in VS$ do
18     $\widehat{r}(u, i) = maxR - (maxR - minR) \cdot \frac{||p(u) - p(i)||_2}{100}$
19     if $\widehat{r}(u, i) < minR$ then
20         $\widehat{r}(u, i) = minR$
21     else if $\widehat{r}(u, i) > maxR$ then
22         $\widehat{r}(u, i) = maxR$
23     end
24 end
```

Algorithm 1: The proposed SCoR algorithm.

Let $G = (V, E)$ denote the given graph with node set V and edge set E. Each node $x \in V$ maintains its own coordinates $p(x) \in \Re^n$ e.g. $n = 40$ (see Section 5.4), the position of node x which is a point in the *n-dimensional* Euclidian space. Similarly to Vivaldi, the algorithm iterates over the following steps:

- Initially, all node coordinates are set at random in $\Re^n$ (see lines 1–6 of Algorithm 1).
- Periodically, each node x randomly selects another node y from a small set of nodes connected to it. Since our graph is bipartite, if x is a user node (u in Algorithm 1), then y is an item node (i in Algorithm 1) and vice versa. Node x, receives the coordinates of node y, calculates its current Euclidian distance to y (Eq. (6)), and compares it against their desired distance (see line 8 of Algorithm 1). Subsequently, node x allows itself to be moved a little by the corresponding imaginary spring connecting it to y (its position changes a little so as the Euclidean distance of the nodes to better match their desired distance).

- This process is repeated until each node's position stabilizes (see line 9 of Algorithm 1).

When the algorithm terminates, a new prediction (recommendation) $\hat{r}(u,i)$ for user u who has never rated item i, is made based on their Euclidean (actual) distance

$$d(u,i) = ||p(u) - p(i)||_2 \tag{6}$$

of the nodes corresponding to u and i (see line 18 of Algorithm 1), as follows:

$$\hat{r}(u,i) = maxR - (maxR - minR) \cdot \frac{||p(u) - p(i)||_2}{100} \tag{7}$$

Values *minR* and *maxR* are used by the algorithm to enforce that the recommendations are valid values in the interval [*minR, maxR*] (see lines 18–23 of Algorithm 1).

We further modified Vivaldi to allow for weighted neighbour selection, for user, item pairs. As already mentioned, in each iteration, each node x updates its position slightly, in order to achieve the desired distance to some node y selected randomly from its neighbouring set. Initially, node y is picked in a uniform fashion from the neighbouring set (line 8 of Algorithm 1). However, as the algorithm progresses, not all user-item pairs are equally successful in achieving the desired distance. In order to help less successful pairs improve their performance, we make a second execution of the algorithm, this time relying more heavily on user-item links that better achieved their desired distance. To implement this step, after Vivaldi converges, we assign a weight to each neighbour, based on the observed error between their desired distance $dd(u, i)$ (see Eq. (5)) and the actual distance $d(u, i)$ (see Eq. (6)) and the Mean Square Error of the node $MSE(u)$ (lines 10–12 of Algorithm 1). Each node u assigns to each of its neighbours i, weight $W(u, i)$ which is calculated according to the following equation:

$$W(u,i) = e^{-0.2 \cdot MSE(u)} \cdot (dd(u,i) - d(u,i))^2 \tag{8}$$

where $MSE(u)$ is the Mean Square Error of node u and its neighbours, reflecting how well the node's position matches its training set ratings, that is defined in the following:

$$MSE(u) = \frac{1}{|NS(u)|} \sum_{i \in NS(u)} (r(u,i) - \hat{r}(u,i))^2 \tag{9}$$

$NS(u)$ and $|NS(u)|$ are the set of neighbours of node u and the number of elements of $NS(u)$, respectively. We then perform another execution of the Vivaldi algorithm using the calculated weights (lines 13–16 of Algorithm 1). During this second execution, nodes with smaller error are picked more often for position updates, implemented by procedure getWeightedRandomSample (see line 14 of Algorithm 1). This procedure selects edge (u, i) with probability $Pr(u, i)$:

$$Pr(u,i) = \frac{W(u,i)}{\sum_{(v,j) \in TS} W(v,j)}. \tag{10}$$

4. Dataset annotations using SCoR

Apart from providing a recommendation system, the proposed framework is able to provide annotations of the user and item datasets based on an analysis of the nodes positions in the *n-dimensional* Euclidean space. SCoR is able to provide the sets of users with similar preferences to each other and the sets of items with similar ratings from users.

Using SCoR, the Euclidean distance $d(u_1, u_2)$ between two user nodes u_1, u_2 is directly related to the absolute difference in their recommendations. When the algorithm converges, it holds that the lower the distance of two nodes, the smaller their recommendation difference $MRD(u_1, u_2)$:

$$MRD(u_1,u_2) = max_{i \in I} |\hat{r}(u_1,i) - \hat{r}(u_2,i)| \tag{11}$$

Table 1
Statistics for the used datasets.

Dataset	Users	Items	Ratings	Ratings/User	σ
smallnetflix	93,705	3561	3,298,163	35.20	69.34
ml	6040	3952	870,607	144.14	282.07
jester	23,499	100	1,486,013	63.24	18.19
jester2	24,937	100	536,694	21.52	5.21

Based on Eq. (7), it holds that the least upper bound (supremum) of *MRD* is given by Eq. (12):

$$MRD(u_1,u_2) \le \frac{(maxR - minR) \cdot d(u_1,u_2)}{100} \tag{12}$$

When the dataset is dense (there are a lot of ratings), then $MRD(u_1, u_2)$ is well approximated by the equality of Eq. (12). Therefore, the distance between two nodes is equal to the corresponding maximum value of their recommendation difference. If the distance between two nodes is low, it means that users u_1, u_2 have similar preferences for any item. On the other hand, if a user is positioned by SCoR away from all other users in the *n-dimensional* Euclidean space, it means that it has unique-peculiar preferences (is an outlier). This can be measured by distance $D_{min}(u_1)$ between node u_1 and its closest user node in the entire dataset.

$$D_{min}(u_1) = min_{u_2 \in U} d(u_1,u_2) \tag{13}$$

Similar analysis can be applied to items. The lower the distance between two items, the lower the difference between their ratings. The SCoR annotations of two datasets are given in the Experimental Results section below (see Section 5.6).

5. Experimental results

In this section, we describe the experiments we conducted using several algorithms and several datasets, in order to evaluate our algorithm.

5.1. Experimental setup

We used four different, well-known, real world datasets in our experiments. The well-known Netflix prize dataset (smallnetflix) (GraphLab, 2012), the MovieLens dataset (ml) (group, 2003), and two versions of the Jester dataset (jester and jester2) (Goldberg, 2003). The Netflix prize dataset is a reduced version of the original one, obtained from the GraphLab website (GraphLab, 2012), and it includes the ratings of several thousand users on several thousand movies, ranging from 1 to 5. The MovieLens dataset, obtained from the GroupLens website (group, 2003) includes one million ratings of users of the MovieLens site, while the Jester datasets contain the ratings of several thousand users on 100 jokes. Each dataset is comprised of two files. A training set which includes the "known" ratings and a validation dataset which is the ground truth against which we test the predictions of the algorithms. The number of users, number of items (e.g. movies, jokes), the number of user ratings, the average number of ratings per user (ratings/user) and the standard deviation of the ratings per user (σ) are shown in Table 1. These statistics are computed on the training set of each respective dataset. The number of ratings per user affects the density of the datasets and it gives the *degree* of the user (node) in the bipartite graph G of the dataset.

Fig. 1 depicts histograms of the user degree for the four used datasets, showing the high variability on the distribution of the degree for the four used datasets. The dataset with the highest degree mean and variance is clearly the ml dataset, where 80% of the users have degree more than 100 and 22% of the users have extremely high degree (more than 500). Its distribution is close to

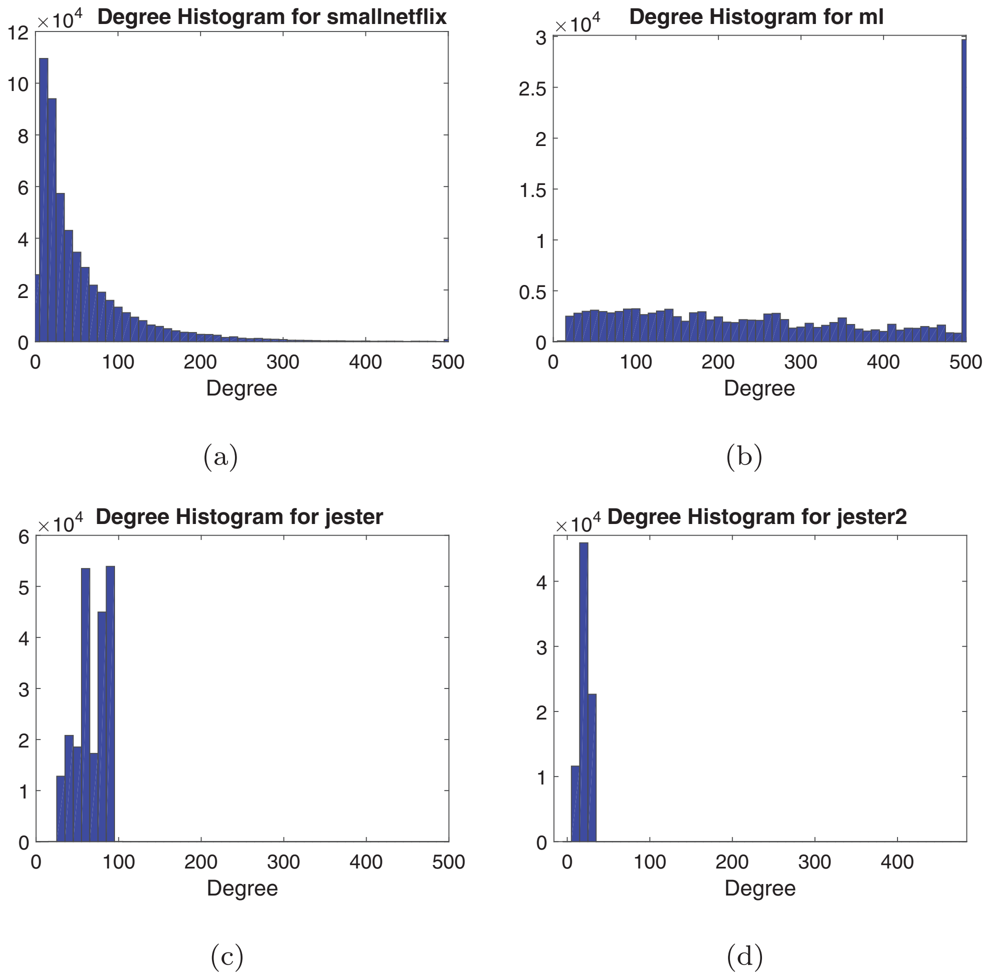


Fig. 1. Histograms of the user degree for the four datasets.

Table 2
Statistics for the modified ml training sets.

Dataset	Users	Items	Ratings	Ratings/User
ml-0.5	6040	3952	435,878	72.16
ml-0.2	6040	3952	174,059	28.81
ml-0.1	6040	3952	86,729	14.35
ml-0.05	6040	3952	43,709	7.23

Table 3
GraphChi parameter values ranges.

Algorithm	lambda	gamma	loss
SGD	10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}	10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}	N/A
BIASSGD	10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}	10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}	N/A
BIASSGD2	10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}	10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}	logistic, abs, square
SVDPP	10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}	10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}	N/A
ALS	0.25, 0.45, 0.65, 0.85	N/A	N/A
ALS_COORD	0.25, 0.45, 0.65, 0.85	N/A	N/A

uniform. The user degree distribution of the smallnetflix dataset is close to gamma. In this dataset, 16% of the users have degree more than 100. Jester and Jester2 have the lowest user degree variation. In both of these datasets, the maximum user degree is less than 100.

In addition, in order to analyze the effect of the dataset density (i.e. Ratings/User) on the algorithm behaviour, we modified MovieLens, which is the most dense dataset, and generated four new datasets with varying densities by randomly keeping a portion (e.g. 50%, 20%, 10% and 5%) of the ratings for the training set. Table 2 shows the corresponding statistics for those new four datasets.

To evaluate its performance SCoR is compared against the following nine recent recommender algorithms: ALS (Zhou et al., 2008), ALS_COORD (Yu et al., 2012), BIASSGD (Koren, 2008), BIASSGD2 (Koren, 2008), RBM (Hinton, 2010), SGD (Koren et al., 2009), SVDPP (Koren, 2008), P_MEAN (Ekstrand et al., 2011) and USER-USER algorithm (Sarwar et al., 2001b). In this work, we have used the implementations of GraphChi library, except the final two, which were provided by the LensKit library. A brief description of these algorithms has been provided in the Related Work section of the paper.

Apart from RBM, the remaining six recommender systems we compare against require a set of input parameters for their execution. We used a small range of values for each parameter, centered around the default values proposed in the GraphChi library, in order to optimize their performance. For every algorithm and dataset, we run experiments for all possible combinations of input values, and selected the configuration that maximized the algorithm's performance for the validation set (although this might not be a fair comparison for the methods that do not receive any parameter). This led to a total of 420 experiments (105 per dataset). The range of parameters used in the experiments is shown in Table 3.

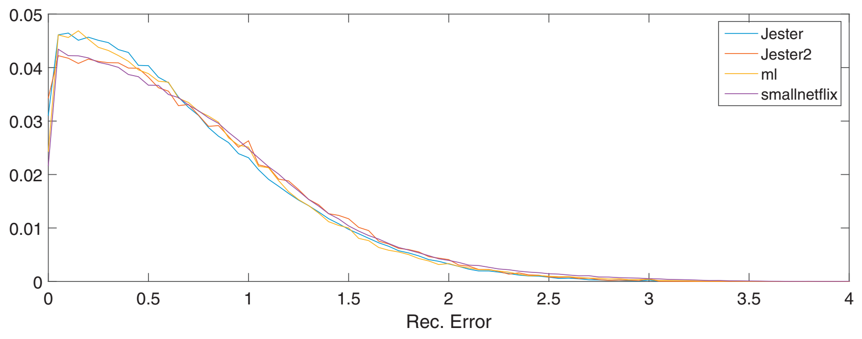


Fig. 2. The PDF of absolute recommendation error computed for each of the four datasets for the SCoR system.

Table 4
RMSE of the ten recommender systems for the four datasets.

Dataset:	smallnetflix	ml	jester	jester2
ALS	1.16	0.964	0.943	1.38
ALS_COORD	1.14	0.932	0.917	1.30
BIASSGD	0.958	0.897	0.872	0.909
BIASSGD2	0.967	0.888	0.861	0.923
SCoR	**0.940**	**0.875**	**0.854**	**0.894**
RBM	0.941	0.900	0.880	0.912
SGD	0.961	0.898	0.872	0.906
SVDPP	0.989	0.944	0.910	0.953
P-MEAN	0.950	0.913	0.886	1.025
USER-USER	0.96	0.905	0.869	1.072

Our algorithm requires no parameterization apart from the number of dimensions of the Euclidean space, which we set to $n = 40$ for all datasets. Additional experiments showed that optimal results are obtained using any number of dimensions above 40, without any noticeable effect on the RMSE, demonstrating the robustness of the proposed scheme (see Section 5.4).

5.2. Performance evaluation

Table 4 presents the performance of the ten recommender systems for the four datasets. According to this table, SCoR gives the highest performance for all datasets. This table summarizes the main results of our experiments. For the rest of this section, we will provide a more in-depth analysis of the results.

Fig. 2 presents the probability density function (PDF) of the absolute recommendation error computed for SCoR and each of the four datasets. It holds that as the error increases, the probability of getting this error rapidly reduces for all datasets in a similar manner. Fig. 3 illustrates the four histograms of the absolute recommendation error computed on each of the datasets, for all algorithms. In order to achieve a fair comparison, we selected the same intervals for each histogram with a step of 0.5.

$$intervals = \{[0, 0.5), [0.5, 1), [1, 1.5), [1.5, 2), [2, \infty]\} \tag{14}$$

One can see the high performance of our algorithm, since it gets high values on the first two bins of the histogram which correspond to low error values. In addition, for all datasets it gives very low values in the last bin of the histogram which correspond to the cases where the absolute error is more than two, meaning it has the lowest probability to fail. This demonstrates that SCoR is very robust compared to the other methods. High performance and robustness results are also obtained for SGD, BIASSGD2, BIASSGD and RBM. Concerning the ALS and ALS_COORD algorithms, although they usually give high values in the first bin of the histograms, they yield quite high values in the last bin, meaning that these systems have some probability of failure.

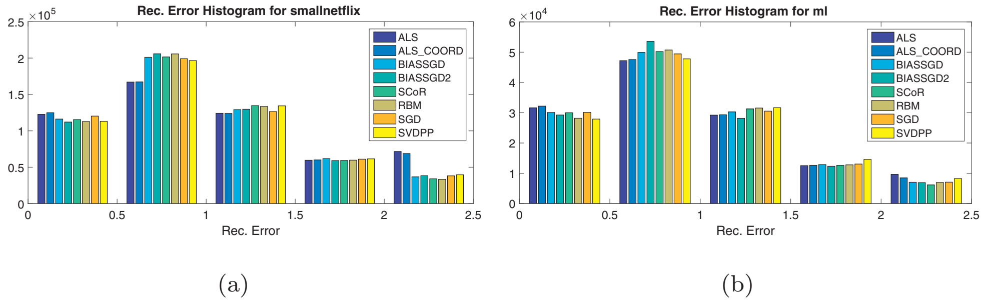
In order to evaluate how the performance of each system is affected by the user degree, the users of each dataset are classified into three equally sized classes according to their degree. The first class of the smallnetflix dataset contains 33.33% of all users with the lowest degree. Then, the RMSE is computed for each class and for each dataset. Fig. 4, depicts the performance of the average user degree for the class. As expected, for most systems the recommendation error slightly decreases as the user degree increases, meaning that, on average, it is more difficult to give a prediction for a user with low degree. Under any dataset and class of users, SCoR yields the lowest or the second lowest RMSE, showing it outperforms the other algorithms. The high performance of the proposed system compared to the rest of the systems is more clear for users of the first class, meaning that SCoR is able to give high performances under the most difficult cases of each dataset. Finally, most of the systems have similar performance for users belonging to the third class. This phenomenon is stronger for the jester dataset (see Fig. 4(c)), where the RMSE for the third class is between 0.85 and 0.87 for most of the systems.

The computation of the RANK (Haindl & Mikeš, 2016) metric per algorithm, which is defined as the average order in performance of each system over the four datasets, can be used as a metric to summarize the algorithms' performance.

$$RANK(s) = \frac{\sum_{dat \in Datasets} Order(s, dat)}{|Datasets|} \tag{15}$$

where $Order(s, dat)$ is the order in performance of system s on the dataset dat, e.g. $Datasets = \{smallnetflix, ml, jester, jester2\}$ and $|Datasets|$ is the number of datasets (e.g. $|Datasets| = 4$). Since we have ten algorithms, the RANK of each one is between one and ten. According to the RANK definition it holds that the lower the RANK, the better the algorithm (Haindl & Mikeš, 2016). Fig. 9 depicts, the RANK of the ten algorithms in ascending order, that is computed for the data of Table 4. According to this experiment we can classify the ten systems into three classes. SCoR belongs in the first class (top performance system), since it clearly outperforms the rest of the systems with RANK = 1. In the second class, systems with good performance are included, like SGD with RANK 3.25 and BIASSGD,BIASSGD2,RBM with RANK 4. In the last class, P-MEAN, SVDPP, ALS_COORD, USER-USER and ALS systems having RANK above 6 are included.

Fig. 5 depicts the PDF of high values (more than two) of the absolute recommendation error computed for each of the four datasets for the **(a)** SCoR, **(b)** SGD, **(c)** BIASSGD and the **(d)** BIASSGD2 systems. SGD, BIASSGD and BIASSGD2 have been selected, since they are the first, second, third, and fourth highest performance systems according to the RANK metric. The PDF of high absolute recommendation errors is selected, since it determines how often the system fails. According to this experiment, it holds that the probability of getting high recommendation errors is usually lower for SCoR than the other three systems under any dataset,



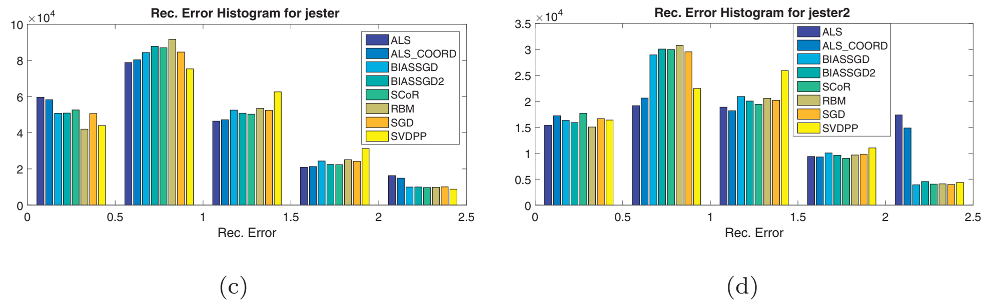
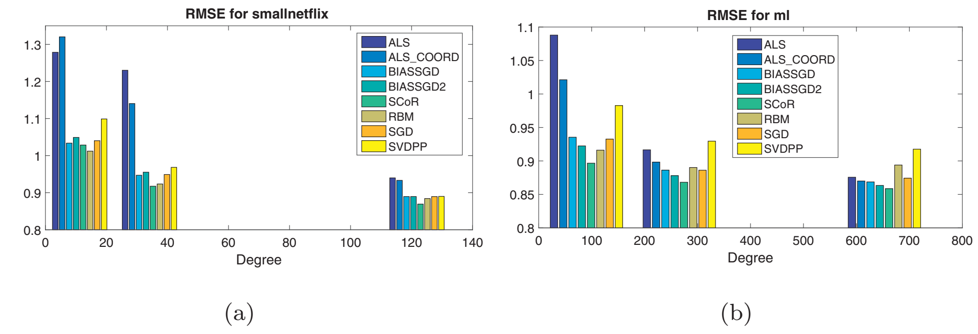


Fig. 3. The histogram of absolute recommendation error computed for each of the four datasets.



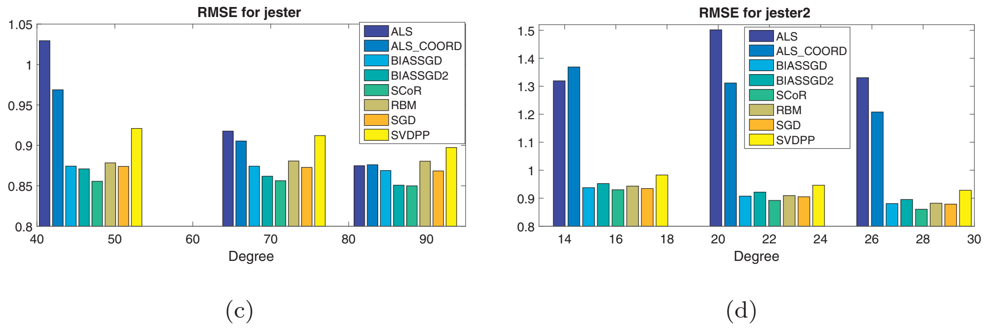


Fig. 4. The RMSE computed for each of the four datasets using three classes according to the degree distribution of each dataset.

showing the stability and robustness of the proposed system. The high performance of SCoR system is more clear under smallnetflix and ml datasets.

5.3. The SCoR convergence

This experiment examines the convergence and the stability of SCoR. Fig. 6 depicts the evolution of RMSE for SCoR on the Training and Validation sets of the ml dataset, during a single execution of the algorithm. The RMSE of the first 50 iterations and of the next 4950 iterations of the SCoR system are shown on the left and right figures, respectively. These iterations include both the unweighted and the weighted execution of the Vivaldi algorithm. As expected, the two time series of RMSE (on Training and Validation sets) are highly correlated converging at similar times. The Pearson correlation coefficient ρ (see Eq. (16)) between these two time series $z_1(t)$ and $z_2(t)$ is 0.9987.

$$\rho(z_1, z_2) = \frac{\sum_t (z_1(t) - \overline{z_1}) \cdot (z_2(t) - \overline{z_2})}{\sqrt{\sum_t (z_1(t) - \overline{z_1})^2} \cdot \sqrt{\sum_t (z_2(t) - \overline{z_2})^2}}, \tag{16}$$

where $\overline{z_1}$ and $\overline{z_2}$ denote the mean values of $z_1(t)$ and $z_2(t)$, respectively.

In addition, this figure shows the stability of the proposed system, meaning that the more iterations the better the results obtained, as well as the faster the convergence of the SCoR system. It holds that after 50 iterations of SCoR, the RMSE on the Vali-

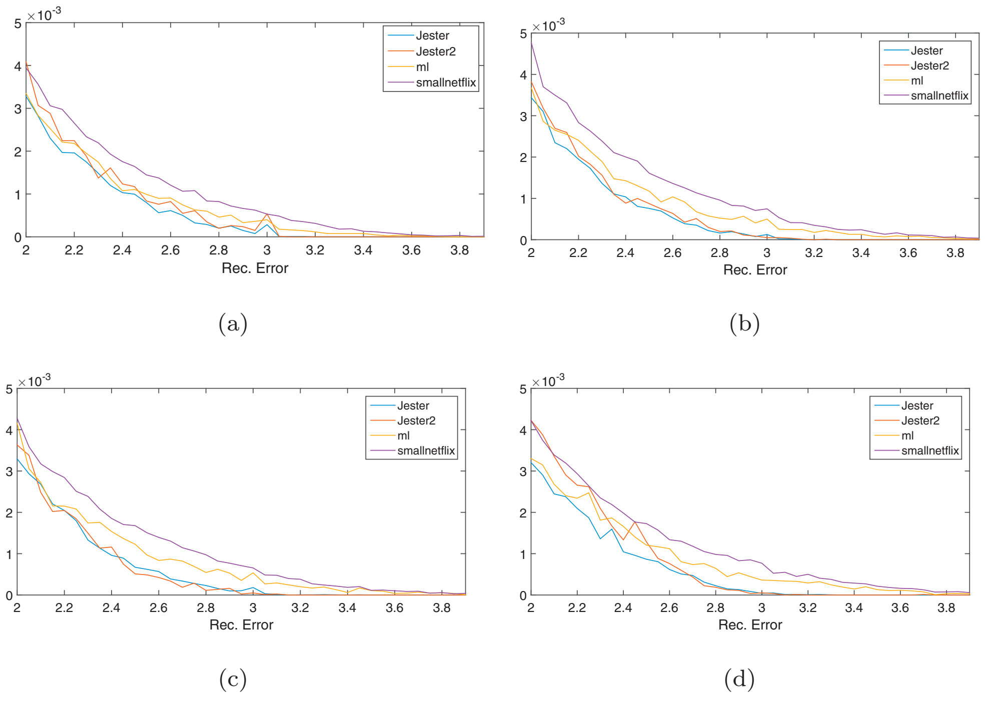


Fig. 5. The PDF for high values of the absolute recommendation error computed for each of the four datasets for **(a)** SCoR, **(b)** SGD, **(c)** BIASSGD and **(d)** BIASSGD2.

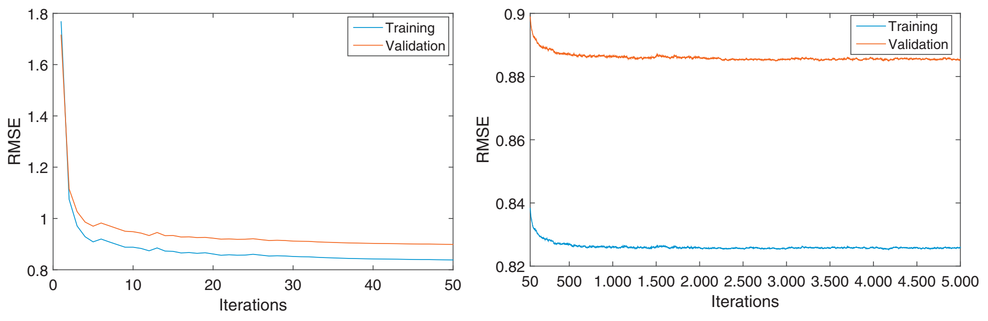


Fig. 6. The evolution of RMSE for SCoR on the Training and Validation sets of the ml dataset.

dation set achieves 98.5% of its minimum value, while after 700 iterations, when the system has converged, the RMSE on the Validation set achieves 99.92% of its minimum value. According to our experiments, the behaviour of SCoR over time is quite similar under any dataset.

5.4. The behaviour of SCoR with respect to the number of dimensions

This experiment examines the performance of SCoR with respect to the different values of the number of dimensions n of the Euclidean space that are used in the Vivaldi method and how it can be determined automatically. The RMSE of the Training and Validation sets of SCoR for each of the four datasets under different number of dimensions of the Euclidean space n is shown in Fig. 7. As expected, the RMSE of the Training set is lower than the corresponding RMSE of the Validation set. Both curves are decreasing with n, converging in a similar way, meaning that n can be selected by an analysis of the RMSE of the Training set, an observation which is quite important for applications. In our experimental setup, we have selected $n = 40$ for any dataset, since even if we select a higher n we get almost the same results, which demonstrates the robustness of the proposed scheme with respect to n.

Finally, regarding the effect of dimensionality increase in computational complexity, we performed several experiments on the same dataset with increasing number of dimensions, measuring execution time. The increase in the execution time proved to be consistently linear to the number of dimensions, with a 40% increase in execution time, when having 40 dimensions, compared to only 5.

5.5. The behaviour of SCoR with respect to density

The goal of this experiment is to examine the behaviour of SCoR with respect to the dataset density (Ratings/User). To do so, we used the ml dataset and its four modifications as described in Section 5.1, getting five datasets ml, ml-0.5, ml-0.2, ml-0.1 and ml-0.05 with

$$Ratings/User = \{144, 72, 28.8, 14.4, 7.2\} \tag{17}$$

respectively. We compared SCoR against the three systems with the highest performance, namely the SGD, the BIASSGD and the BIASSGD2. Fig. 10 illustrates the RMSE of SCoR, SGD, BIASSGD and BIASSGD2 under different ratings/user (the modifications of the ml dataset). As expected, RMSE decreases with density. It holds that SCoR works well when the density is high and medium. The performance of SCoR increases as we move from high to medium density values. However, when the density is very low, which correspond to very sparse datasets (e.g. ratings/user < 15),

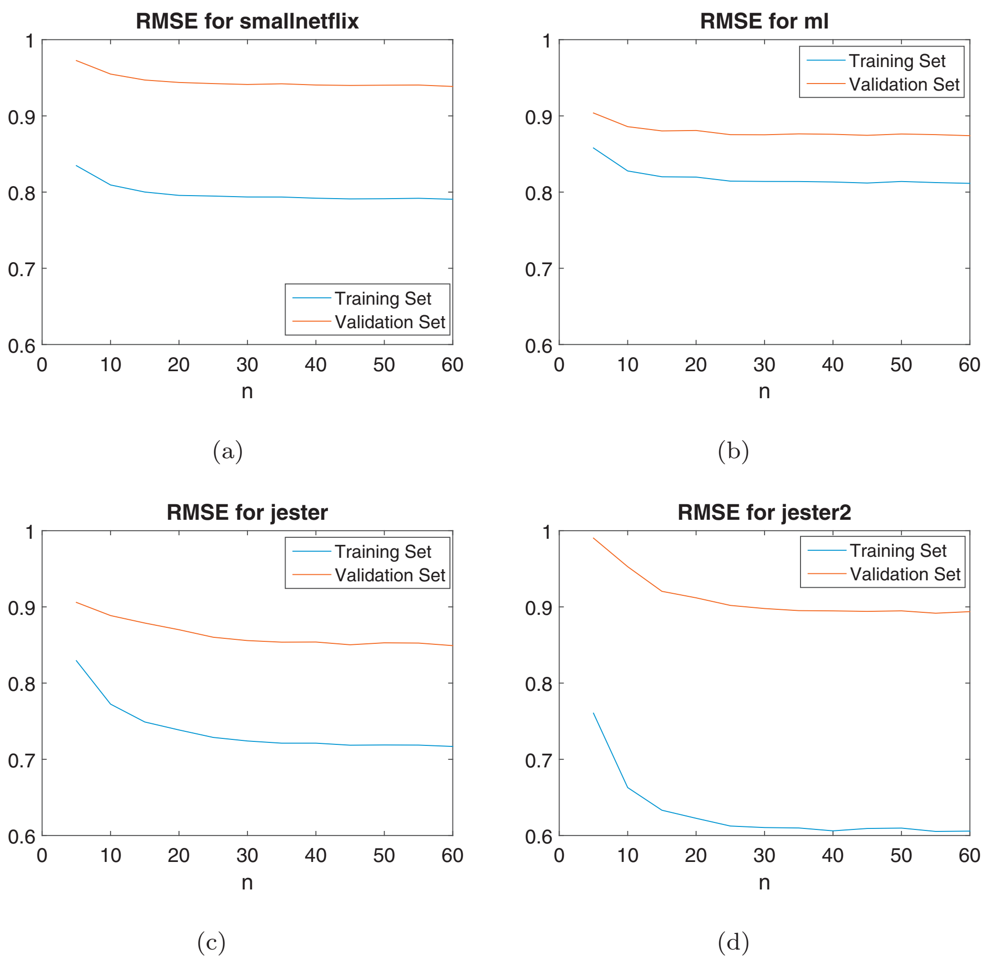


Fig. 7. The RMSE of SCoR for each of the four datasets under different number of Euclidean space dimensions *n*.

SGD outperforms SCoR, BIASSGD and BIASSGD2. This means that SCoR works well for medium to high density datasets but it is possible to fail for very sparse datasets. This behaviour is somewhat expected, since the Vivaldi synthetic coordinates algorithm works better when the number of physical springs per node is high enough (more than the given space-dimensionality) in order to be able to place the nodes in proper positions in the multi-dimensional Euclidean space.

5.6. The annotations of SCoR

The SCoR based annotations of smallnetflix and ml datasets (see Section 5.1) are given hereafter. Fig. 8(a) depicts the cumulative distribution functions (CDFs) of D_{min} for users of ml (blue curve) and smallnetflix (red curve) datasets, when $D_{min} < 10$. It holds that 11% of smallnetflix users have very similar preferences ($D_{min} < 10$), while the same holds for only 5% of ml users, since the red and blue curves passes from points (10,0.11) and (10,0.05), respectively. This can also be explained by the fact that smallnetflix is larger than the ml dataset. Fig. 8(b) depicts the CDFs of D_{min} for users of ml (blue curve) and smallnetflix (red curve) datasets when $D_{min} > 50$ in order to detect outliers. According to this figure, it holds that 1% and 2% of the ml and smallnetflix users have $D_{min} > 50$ and can be classified as outliers, respectively, since the two CDFs passes from points (50,0.99) and (50,0.98), respectively. Thus, we can conclude that smallnetflix has more outliers than the ml dataset. This is not generally expected, since in a large dataset, the users' density is higher meaning that it is more difficult to detect outliers.

In order to validate the phenomenon of outliers, we inserted on the ml training set 100 users with 100 random ratings each, called "noisy users". Noisy users have, by construction, unique-peculiar preferences. Fig. 8(c) depicts the probability density function (PDF) of all pairs of distances between users of the ml dataset (blue curve), between noisy users and all users of the ml dataset (red curve), and between all item pairs of the ml dataset (black curve). As expected, the average value of the red distribution is higher (about 50%) than the average value of the blue one. Additionally, the average distance between items is higher than the average distance between users, indicating that items, as expected, are less related to each other. This means that it is more rare for two items to receive similar ratings by all common users which have rated them than two users that grade common items in a similar way. Moreover, there exist some pairs of items with very high distance between them, e.g. about 5% of the distances between pair of items is higher than 100.

Fig. 8(d) depicts the number of clusters of users of ml dataset (blue curve) and movies of ml dataset (red curve) as a function of cluster diameter. The clusters are constructed by the agglomerative hierarchical clustering method with stopping criterion the given maximum cluster diameter. A cluster diameter is the maximum distance between any two points of the cluster. This experiment shows that the users create more easily clusters than the movies. Even when the diameter is 100 there exists 124 clusters of movies, while the number of clusters of users is only 19. So, the users can be grouped in larger clusters which inevitably will be less in number. On the other hand, the disparity in space of the

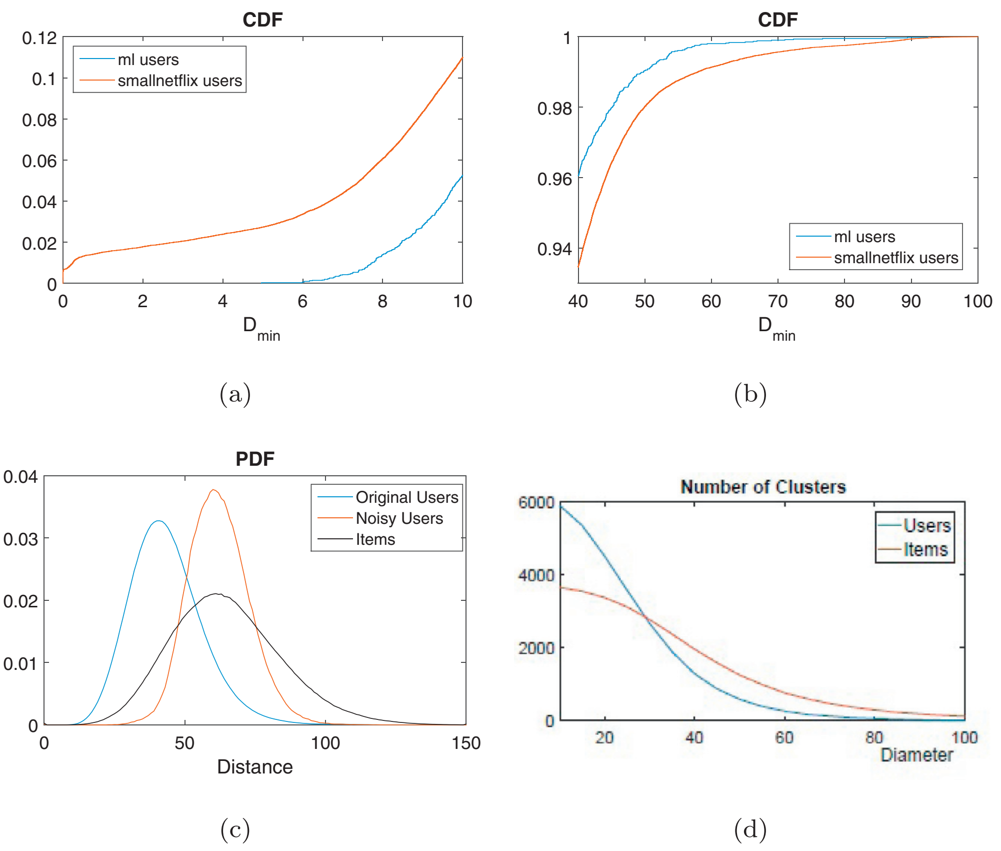


Fig. 8. **(a)** The CDFs of D_{min} for the users of ml (blue curve) and smallnetflix (red curve) datasets, when $D_{min} < 10$. **(b)** The CDFs of D_{min} for the users of ml (blue curve) and smallnetflix (red curve) datasets, when $D_{min} > 40$. **(c)** The PDFs for all pairs of distances between users of ml dataset (blue curve), between noisy users and all users (red curve) and between items of ml dataset (black curve). **(d)** The number of clusters of users of the ml dataset (blue curve) and items of the ml dataset (red curve) as a function of cluster diameter. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

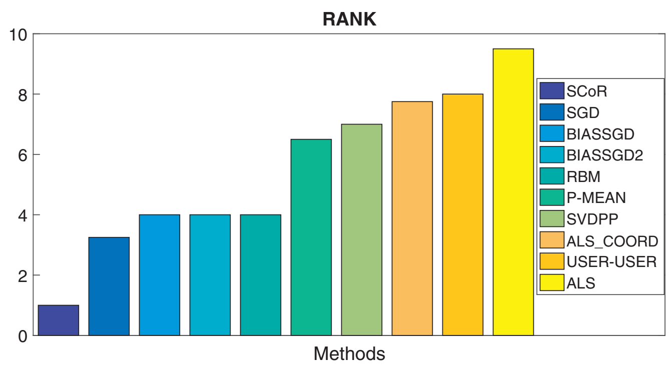


Fig. 9. The RANK of the ten systems computed for the four datasets.

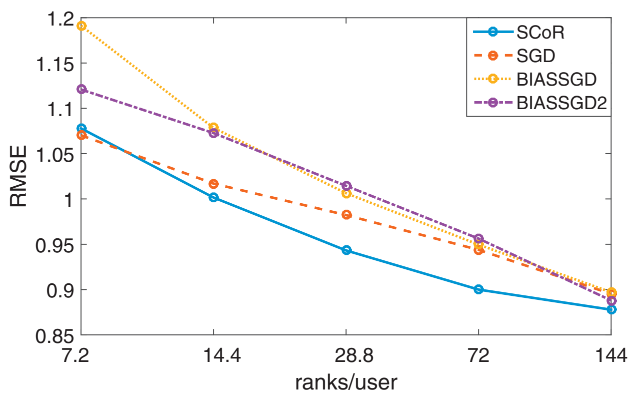


Fig. 10. The RMSE of SCoR, SGD, BIASSGD and BIASSGD2 for different ratings/user (modifications of ml dataset).

movies means that they can only be grouped in many clusters of small cardinality.

6. Conclusions

We presented SCoR, a recommender system which is based on the Vivaldi synthetic network coordinates system. The proposed algorithm has been tested on several real datasets with high variability in density and size. The proposed system is compared against seven state-of-the-art recommender systems, proving its effectiveness, stability and higher performance. Under any dataset, SCoR is the first in performance. Apart from the high performance and stability of SCoR, other advantages of the proposed system compared to the other state-of-the-art algorithms are the fact that it does not require any parameter for execution. Although we have selected the configurations of the state-of-the-art algorithms that maximize their performance for each dataset, the SCoR system outperforms them under the most difficult datasets, even for users with low degree. Additionally, the proposed framework is able to provide important annotations for the datasets, by easily detecting users and items with common and unique preferences-ratings.

We plan to extend the system to better handle sparse datasets as well as cold-start users. An important axis for future work includes exploring and demonstrating the performance of SCoR under dynamic changes in the dataset. The convergence ability of the algorithm as new users and/or items arrive into the system will be tested under different scenarios. The fact that items tent to arrive in a much faster pace compared to users will be evaluated. Another important research direction would be to explore the behaviour of the system under temporal patterns, as items tend to become very popular for a period of time and to fade away subsequently, and users tend to re-rate the same items differently. To study these phenomena the temporal characteristics of the datasets should be taken into consideration in the performed experiments. Finally, incorporating metadata and semantic information in the algorithm could enhance significantly its performance.Finally, a working demo showcasing our algorithm can be found at https://sites.google.com/site/netdilab/research/scor.

References

Adomavicius, G., & Kwon, Y. (2012). Improving aggregate recommendation diversity using ranking-based techniques. *IEEE Transactions on Knowledge and Data Engineering, 24*(5), 896–911.

Adomavicius, G., & Tuzhilin, A. (2005). 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.

Bennett, J., Lanning, S., & Netflix, N. (2007). The netflix prize. *In kdd cup and workshop in conjunction with kdd*.

Carrer-Neto, W., Hernández-Alcaraz, M. L., Valencia-García, R., & García-Sánchez, F. (2012). Social knowledge-based recommender system. application to the movies domain. *Expert Systems with applications, 39*(12), 10990–11000.

Dabek, F., Cox, R., Kaashoek, F., & Morris, R. (2004). Vivaldi: A decentralized network coordinate system. In *Proceedings of the 2004 conference on applications, technologies, architectures, and protocols for computer communications*. In *SIGCOMM '04* (pp. 15–26). New York, NY, USA: ACM.

Ekstrand, M. D., Riedl, J. T., & Konstan, J. A. (2011). Collaborative filtering recommender systems. *Foundations and Trends in Human-Computer Interaction, 4*(2), 81–173. doi:10.1561/1100000009.

Goldberg, D., Nichols, D., Oki, B. M., & Terry, D. (1992). Using collaborative filtering to weave an information tapestry. *Communications of the ACM, 35*(12), 61–70. doi:10.1145/138859.138867.

Goldberg, K. (2003). The jester recommender systems dataset. http://www.ieor.berkeley.edu/~goldberg/jester-data/.

Goldberg, K., Roeder, T., Gupta, D., & Perkins, C. (2001). Eigentaste: A constant time collaborative filtering algorithm. *Information Retrieval, 4*(2), 133–151. doi:10.1023/A:1011419012209.

Gorrell, G. (2006). Generalized hebbian algorithm for incremental singular value decomposition in natural language processing. *EACL 2006, 11st conference of the european chapter of the association for computational linguistics, proceedings of the conference, april 3–7, 2006, trento, italy*.

GraphLab (2012). The smallnetflix recommender systems dataset. http://www.select.cs.cmu.edu/code/graphlab/datasets/.

group, G. (2003). The movielens recommender systems dataset. http://http://grouplens.org/datasets/movielens/.

Gunawardana, A., & Shani, G. (2009). A survey of accuracy evaluation metrics of recommendation tasks. *Journal of Machine Learning Research, 10*(Dec), 2935–2962.

Haindl, M., & Mikeš, S. (2016). A competition in unsupervised color image segmentation. *Pattern Recognition, 57*, 136–151.

Herlocker, J. L., Konstan, J. A., Terveen, L. G., & Riedl, J. T. (2004). Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems, 22*(1), 5–53.

Hinton, G. (2010). A practical guide to training restricted Boltzmann machines. *Technical Report*. University of Toronto.

Koren, Y. (2008). Factorization meets the neighborhood: A multifaceted collaborative filtering model. In *Proceedings of the 14th acm sigkdd international conference on knowledge discovery and data mining*. In *KDD '08* (pp. 426–434).

Koren, Y., Bell, R., & Volinsky, C. (2009). Matrix factorization techniques for recommender systems. *Computer, 42*(8), 30–37.

Linden, G., Smith, B., & York, J. (2003). Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing, 7*(1), 76–80.

Ma, H., King, I., & Lyu, M. R. (2012). Mining web graphs for recommendations. *IEEE Transactions on Knowledge and Data Engineering, 24*(6), 1051–1064. doi:10.1109/TKDE.2011.18.

Mobasher, B., Burke, R. D., & Sandvig, J. J. (2006). Model-based collaborative filtering as a defense against profile injection attacks. In *Proceedings, the twenty-first national conference on artificial intelligence and the eighteenth innovative applications of artificial intelligence conference, july 16–20, 2006, boston, massachusetts, USA* (pp. 1388–1393).

Park, D. H., Kim, H. K., Choi, I. Y., & Kim, J. K. (2012). A literature review and classification of recommender systems research. *Expert Systems with Applications, 39*(11), 10059–10072.

Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., & Riedl, J. (1994). Grouplens: An open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 acm conference on computer supported cooperative work*. In *CSCW '94* (pp. 175–186). New York, NY, USA: ACM.

Sarwar, B., Karypis, G., Konstan, J., & Riedl, J. (2001a). Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on world wide web*. In *WWW '01* (pp. 285–295). New York, NY, USA: ACM.

Sarwar, B., Karypis, G., Konstan, J., & Riedl, J. (2001b). Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on world wide web*. In *WWW '01* (pp. 285–295). New York, NY, USA: ACM. doi:10.1145/371920.372071.

Wikipedia (2014). Katz centrality. http://en.wikipedia.org/wiki/Katz_centrality.

Yu, H.-F., Hsieh, C.-J., Si, S., & Dhillon, I. (2012). Scalable coordinate descent approaches to parallel matrix factorization for recommender systems. In *Proceedings of the 2012 ieee 12th international conference on data mining*. In *ICDM '12* (pp. 765–774). Washington, DC, USA: IEEE Computer Society.

Zhang, Y., & Zhang, H. (2012). Triangulation inequality violation in internet delay space. In *Advances in computer science and information engineering* (pp. 331–337). Springer.

Zhou, Y., Wilkinson, D., Schreiber, R., & Pan, R. (2008). Large-scale parallel collaborative filtering for the netflix prize. In *Proc. 4th intl conf. algorithmic aspects in information and management, lncs 5034* (pp. 337–348). Springer.