ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/227044312

Data Sparsity Issues in the Collaborative Filtering Framework

Conference Paper · October 2006

DOI: 10.1007/11891321_4

CITATIONS
84

READS
2,335

4 authors, including:

Dunja Mladenić
Jožef Stefan Institute
417 PUBLICATIONS **6,459** CITATIONS

SEE PROFILE

Blaž Fortuna
Jožef Stefan Institute
111 PUBLICATIONS **1,555** CITATIONS

SEE PROFILE

Marko Grobelnik
Jožef Stefan Institute
325 PUBLICATIONS **4,876** CITATIONS

SEE PROFILE

Some of the authors of this publication are also working on these related projects:

Janez Starc PhD View project

Ist-world View project

All content following this page was uploaded by Blaž Fortuna on 28 May 2014.

The user has requested enhancement of the downloaded file.

Data Sparsity Issues in the Collaborative Filtering Framework

Miha Grčar, Dunja Mladenič, Blaž Fortuna, and Marko Grobelnik

Jožef Stefan Institute, Jamova 39, SI-1000 Ljubljana, Slovenia,
miha.grcar@ijs.si

Abstract. With the amount of available information on the Web growing rapidly with each day, the need to automatically filter the information in order to ensure greater user efficiency has emerged. Within the fields of user profiling and Web personalization several popular content filtering techniques have been developed. In this chapter we present one of such techniques – collaborative filtering. Apart from giving an overview of collaborative filtering approaches, we present the experimental results of confronting the k-Nearest Neighbor (kNN) algorithm with Support Vector Machine (SVM) in the collaborative filtering framework using datasets with different properties. While the k-Nearest Neighbor algorithm is usually used for collaborative filtering tasks, Support Vector Machine is considered a state-of-the-art classification algorithm. Since collaborative filtering can also be interpreted as a classification/regression task, virtually any supervised learning algorithm (such as SVM) can also be applied. Experiments were performed on two standard, publicly available datasets and, on the other hand, on a real-life corporate dataset that does not fit the profile of ideal data for collaborative filtering. We conclude that the quality of collaborative filtering recommendations is highly dependent on the sparsity of available data. Furthermore, we show that kNN is dominant on datasets with relatively low sparsity while SVM-based approaches may perform better on highly sparse data.

1 Introduction

Collaborative filtering is based on the assumption that "similar users have similar preferences". In other words, by finding users that are similar to the active user and by examining their preferences, the recommender system can (i) predict the active user's preferences for certain items and (ii) provide a ranked list of items which the active user will most probably like. Collaborative filtering generally ignores the form and the content of the items and can therefore also be applied to non-textual items. Furthermore, collaborative filtering can detect relationships between items that have no content similarities but are linked implicitly through the groups of users accessing them.

This chapter is arranged as follows. In Sect. 2 we first give an overview of collaborative filtering approaches. Later on, we discuss different characteristics of collaborative filtering datasets and present major issues concerning dataset

properties in the context of collaborative filtering (Sect. 3). To understand the influence that highly sparse server-side collected data has on the accuracy of collaborative filtering, we ran a series of experiments in which we used two publicly available datasets and, on the other hand, a real-life corporate dataset that does not fit the profile of ideal data for collaborative filtering. In Sect. 4 we present our evaluation platform and in Sect. 5 we describe the datasets that we used for our experiments. In Sect. 6 and 7 we present the experimental results of confronting the k-Nearest Neighbor (kNN) algorithm with Support Vector Machine (SVM) in the collaborative filtering framework using datasets with different properties. The chapter concludes with a discussion and a short presentation of some ideas for future work (Sect. 8).

2 About Collaborative Filtering

Collaborative filtering compares users according to their preferences. Therefore, a database of users' preferences must be available. The preferences can be collected either explicitly (explicit ratings) or implicitly (implicit preferences). In the first case, the user's participation is required. The user explicitly submits his/her rating of the given item. Such rating can, for example, be given as a value on a rating scale from 1 to 5. The implicit preferences, on the other hand, are derived from monitoring the user's behavior. In the context of the Web, access logs can be examined to determine such implicit preferences. For example, if the user accessed the document, he/she implicitly rated it 1. Otherwise the document is assumed to be rated 0 by the user (i.e. "did not visit"). The collaborative filtering process can be divided into two phases: (i) the model generation phase and (ii) the recommendation phase. Algorithms which tend to skip the first phase are the so called memory-based approaches (also referred to as lazy learning approaches or the nearest neighbors algorithms) (see Sect. 2.1). The preferences database is a huge user-item matrix, $R = [r_{i,j}]$, constructed from the data at hand. A matrix element $r_{i,j}$ represents user i's rating of item j. Memory-based approaches search the matrix to find relationships between users and/or items. Model-based approaches, on the other hand, use the data from R to build a model that enables faster and more accurate recommendations (see Sect. 2.2). The model generation is usually performed offline over several hours or days.

When dealing with collaborative filtering, two fundamental problems of collaborative filtering have to be taken into account: (i) the sparsity of the data and (ii) the scalability problem. The first problem, which we encounter when R is missing many values, can be partially solved by incorporating other data sources (such as the contents of the items) [14], by clustering users and/or items [4, 19], or by reducing the dimensionality of the initial matrix. The last two techniques also counter the scalability problem. This problem arises from the fact that with the growth of the number of users and the number of items, the basic nearest neighbors algorithm fails to scale up its computation. Some of the approaches for countering the two problems are described in Sect. 2.2 (Dimensionality Reduction Techniques).

2.1 Memory-based Approach to Collaborative Filtering

A straightforward algorithmic approach to collaborative filtering involves finding k nearest neighbors (i.e. the most similar users) of the active user and averaging their ratings of the item in question. Even more, we can calculate the weighted average of the ratings – weights being similarity, correlation or distance factors (later on in the text, the term "similarity" is used to denote any of the three measures) between a neighbor user and the active user (e.g. [4]). We can look at a user as being a feature vector. In this aspect, items that are being rated are features and ratings given by the user to these items are feature values. The following formula can be applied to predict user u's rating of item i:

$$p_{u,i} = \overline{v_u} + \kappa \sum_{j \in Users} w(u,j)\,(v_{j,i} - \overline{v_j}) \quad , \tag{1}$$

where $w(u_1, u_2)$ is the weight which is higher for more similar, less distant or more correlated users (feature vectors), $\overline{v_u}$ is the mean rating given by user u, $v_{j,i}$ is the rating of item i given by user j, and κ is merely a normalization factor which depends on our choice of weighting.

When representing a user as a feature vector, many of the features have missing values since not every item was explicitly rated by the user. This fact introduces the sparsity problem which implies that measuring similarity between two feature vectors is not a trivial task. Many times two feature vectors have only a few or no overlapping values at all. When computing similarity over only a few values, the similarity measure is unreliable. Furthermore, when there is no overlapping between two vectors, the degree of similarity can not be determined.

Equation 1 was introduced by [15]. If no ratings of item i are available, the prediction is equal to the average rating given by user u. This is an evident improvement to the equation that simply calculates the weighted average.

Weight Computation. The weights can be defined in many different ways. Some of the possibilities are summarized in the following paragraphs.

Cosine Similarity. The similarity measure can be defined as the cosine of the angle between two feature vectors. This technique is primarily used in information retrieval for calculating similarity between two documents, where documents are usually represented as vectors of word frequencies. In this context, the weights can be defined as:

$$w(u_1, u_2) = \sum_{i \in Items} \frac{v_{u_1,i}\; v_{u_2,i}}{\sqrt{\sum_{k \in I_1} v_{u_1,k}^2}\sqrt{\sum_{k \in I_2} v_{u_2,k}^2}} \quad . \tag{2}$$

Pearson Correlation. The weights can be defined in terms of the Pearson correlation coefficient [15]. Pearson correlation is primarily used in statistics to evaluate the degree of linear relationship between two random variables. It ranges from

-1 (a perfect negative relationship) to $+1$ (a perfect positive relationship), with 0 stating that there is no relationship whatsoever. The formula is as follows:

$$w(u_1, u_2) = \frac{\sum_{j \in Items} \left(v_{u_1,j} - \overline{v_{u_1}}\right)\left(v_{u_2,j} - \overline{v_{u_2}}\right)}{\sqrt{\sum_{j \in Items} \left(v_{u_1,j} - \overline{v_{u_1}}\right)^2 \sum_{j \in Items} \left(v_{u_2,j} - \overline{v_{u_2}}\right)^2}} \ . \qquad (3)$$

2.2 Model-based Approaches to Collaborative Filtering

In contrast to a memory-based method, a model-based method first builds a model out of the user-item matrix. The model enables faster and more accurate recommendations. In the following paragraphs we present two different approaches to model-based collaborative filtering. In the first one, we perceive collaborative filtering as a classification task. We employ a supervised learning algorithm to build a model. In the second one, we are concerned with reducing the dimensionality of the initial user-item matrix and thus build a model that is a lower-dimensional representation of the initial user-item database.

Collaborative Filtering as a Classification Task. The collaborative filtering task can also be interpreted as a classification task, classes being different rating values [3]. Virtually any supervised learning algorithm can be applied to perform classification (i.e. prediction). For each user we train a separate classifier. A training set consists of feature vectors representing items the user already rated, labels being ratings from the user. Clearly the problem occurs if our training algorithm cannot handle the missing values in the sparse feature vectors. It is suggested by [3] to represent each user by several instances (optimally, one instance for each possible rating value). On a 1–5 rating scale, user A would be represented with 5 instances, namely A-rates-1, A-rates-2, ..., A-rates-5. The instance A-rates-3, for example, would hold ones ("1") for each item that user A rated 3 and zeros ("0") for all other items. This way, we fill in the missing values. We can now use such binary feature vectors for training. To predict a rating, we need to classify the item into one of the classes representing rating values. If we wanted to predict ratings on a continuous scale, we would have to use a regression approach instead of classification.

Dimensionality Reduction Techniques. We are initially dealing with a huge user-item matrix. Since there can be millions of users and millions of items, the need to reduce the dimensionality of the matrix emerges. The reduction can be carried out by selecting only relevant users (instance selection) and/or by selecting only relevant items (feature selection). Other forms of dimensionality reduction can also be employed, as described later on in this section.

It is shown by some researchers that feature selection, instance selection and other dimensionality reduction techniques not only counter the scalability problem but also result in more accurate recommendations [19, 12, 18]. Furthermore, the sparsity of the data is consequentially decreased.

When reducing the dimensionality, the first possibility that comes to mind is removing the users that did not rate enough items to participate in collaborative filtering. From the remaining users, we can randomly choose n users to limit our search for the neighborhood of the active user. This method is usually referred to as random sampling. Also, rarely rated items can be removed for better performance. Still, these relatively simple approaches are usually not sufficient for achieving high scalability and maintaining the recommendation accuracy.

Latent Semantic Analysis (LSA). A more sophisticated dimensionality reduction approach is called Latent Semantic Analysis (LSA) [8]. It is based on Singular Value Decomposition (SVD) of the user-item matrix. By using linear algebra, we can decompose a matrix into a triplet, namely $\boldsymbol{M} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$. The diagonal matrix $\boldsymbol{\Sigma}$ holds the singular values of $\boldsymbol{M}$. If we set all but K largest singular values to zero and thus obtain $\boldsymbol{\Sigma}'$, we can approximate $\boldsymbol{M}$ as $\boldsymbol{M}' = \boldsymbol{U}\boldsymbol{\Sigma}'\boldsymbol{V}^T$. By doing so, we transform our initial high-dimensional matrix into a K-dimensional (low-dimensional) space. The neighborhood of the active user can now be determined by transforming the user vector into the low-dimensional space of the approximated matrix and finding k nearest points representing other users. Searching through a low-dimensional space clearly demands less time. Furthermore, dimensionality reduction reduces sparsity and captures transitive relationships among users. This results in higher accuracy.

Probabilistic Latent Semantic Analysis (pLSA). On the basis of LSA, Probabilistic Latent Semantic Analysis (pLSA) was developed [11]. pLSA has its roots in information retrieval but can also be employed for collaborative filtering [12]. In a statistical model, an event like "person u 'clicked on' item i" is presented as an observation pair (u, i) (note that in such case we are dealing with implicit ratings). User u and item i "occur" paired with a certain probability: $P(u, i)$. We are in fact interested in the conditional probability of item i occurring given user u: $P(i|u)$. This conditional form is more suitable for collaborative filtering since we are interested in the active user's interests.

The main idea of an aspect model (such as pLSA) is to introduce a latent variable z, with a state for every possible occurrence of (u, i). User and item are rendered independent, conditioned on z: $P(u, i) = P(z)P(u|z)P(i|z)$. $P(i|u)$ can be written in the following form:

$$P(i|u) = \sum_z P(i|z)P(z|u) \ . \tag{4}$$

Note that we limit the number of different states of z so that it is much smaller than the number of (u, i) pairs. Let us denote the number of users with N_u, the number of items with N_i, and the number of different states of z with N_z, where $N_z << N_u, N_i$. We can describe the probabilities $P(i|u)$ with $S_1 = N_i \times N_u$ independent parameters. On the other hand, we can summarize the probabilities $P(i|z)$ and $P(z|u)$ with $S_2 = N_i \times N_z + N_u \times N_z$ independent parameters. The dimensionality reduction is evident from the fact that $S_2 < S_1$ (if N_z is

small enough). Such latent class models tend to combine items into groups of similar items, and users into groups of similar users. In contrast to clustering techniques (see Sect. 2.4), pLSA allows partial memberships in clusters (clusters being different states of z).

In (4), the probabilities $P(z|u)$ and $P(i|z)$ can be determined by the Expectation Minimization (EM) algorithm using various mixture models. To support explicit ratings, we extend pLSA by incorporating ratings to our observation pairs and thus observing triplets of the form (u, i, r), where r represents a rating value.

The relation of this method to LSA and SVD can be explained by representing the probabilities $P(u, i)$ in the form of a matrix $\boldsymbol{M_p}$ which can be decomposed into three matrices, namely $\boldsymbol{M_p} = \boldsymbol{U_p \Sigma_p {V_p}^T}$. The elements of these matrices are $u_{i,k} = P(u_i|z_k)$, $\sigma_{k,k} = P(z_k)$, $v_{j,k} = P(i_j|z_k)$ ([2], Chap. 7).

2.3 Item-based Collaborative Filtering

All collaborative filtering approaches that we have discussed so far, are user-centric in the way that they concentrate on determining the user's neighborhood. Some researchers also considered item-based collaborative filtering [17]. The main idea is to compute item-item similarities (according to the users' ratings) offline and make use of them in the online phase. To predict user u's rating of item i, the online algorithm computes a weighted sum of the user u's ratings over k items that are most similar to item i. The main question in this approach is how to evaluate item-item similarities to compute a weighted sum of the ratings. Item-item similarities can be computed by using the techniques for computing user-user similarities, described in Sect. 2.1. The winning technique, according to [17], is the so called adjusted cosine similarity measure. This is a variant of cosine similarity which incorporates the fact that different users may have different rating scales. The similarity measures are then used as weights for calculating a weighted sum of k nearest items.

2.4 Some Other Approaches

Let us briefly summarize some other techniques. Interested reader should consider the appropriate additional reading.

Horting. Horting is a graph-theoretic approach to collaborative filtering [1]. It involves building a directed graph in which vertices represent users and edges denote the degree of similarity between them. If we want to predict user u's rating of item i, we need to find a directed path from user u to a user who rated item i. By using linear transformations assigned to edges along the path, we can predict user u's rating of item i. One characteristic of horting graph is that item i is not rated by any other user along this path. This means that Horting also explores transitive relationships between users.

Clustering Techniques. Bayesian and non-Bayesian clustering techniques can be used to build clusters (or neighborhoods) of similar users [4, 19, 12]. The active user is a member of a certain cluster. To predict his/her rating of item i, we compute the average rating for item i within the cluster that the user belongs to. Some such methods allow partial membership of the user in more than one cluster. In such case, the predicted rating is summed over several clusters, weighted by the user's participation degree. Clustering techniques can also be used as instance selection techniques (instances being users) that are used to reduce the candidate set for the k-Nearest Neighbors algorithm.

Bayesian Networks. Bayesian networks with a decision tree at each node have also been applied to collaborative filtering [4, 6]. Nodes correspond to items, and states of each node correspond to possible rating values. Conditional probabilities at each node are represented as decision trees in which nodes again are items, edges represent preferences, and leaves represent the possible states (i.e. rating values). Bayesian networks are built offline over several hours or even days. This approach is not suitable in systems that need to update rapidly and frequently.

3 Collaborative Filtering Data Characteristics

As already mentioned, the data in the user-item interaction database can be collected either explicitly (explicit ratings) or implicitly (implicit preferences). In the first case, the user's participation is required. The user is asked to explicitly submit his/her rating for the given item. In contrast to this, implicit preferences are inferred from the user's actions in the context of an item (that is why the term "user-item interaction" is used instead of the word "rating" when referring to users' preferences in the following sections). Data can be collected implicitly either on the client side or on the server side. In the first case, the user is bound to use modified client-side software that logs his/her actions. Since we do not want to enforce modified client-side software, this possibility is usually omitted. In the second case, the logging is done by a server. In the context of the Web, implicit preferences can be determined from access logs that are automatically maintained by Web servers.

Collected data is first preprocessed and arranged into a user-item matrix. Rows represent users and columns represent items. Each matrix element is in general a set of actions that a specific user took in the context of a specific item. In most cases a matrix element is a single number representing either an explicit rating or a rating that was inferred from the user's actions.

Since a user usually does not access every item in the repository, the vector (i.e. the matrix row) representing the user is missing some/many values. To emphasize this, we use the terms "sparse vector" and "sparse matrix".

The fact that we are dealing with a sparse matrix can result in the most concerning problem of collaborative filtering – the so called sparsity problem. In order to be able to compare two sparse vectors, similarity measures require some values to overlap. What is more, the lower the amount of overlapping values, the

lower the reliability of these measures. If we are dealing with a high level of sparsity, we are unable to form reliable neighborhoods. Furthermore, in highly sparse data there might be many unrated (unseen) items and many inactive users. Those items/users, unfortunately, cannot participate in the collaborative filtering process.

Sparsity is not the only reason for the inaccuracy of recommendations provided by collaborative filtering. If we are dealing with implicit preferences, the ratings are usually inferred from the user-item interactions, as already mentioned earlier in the text. Mapping implicit preferences into explicit ratings is a non-trivial task and can result in false mappings. The latter is even more true for server-side collected data in the context of the Web since Web logs contain very limited information. To determine how much time a user was reading a document, we need to compute the difference in time-stamps of two consecutive requests from that user. This, however, does not tell us weather the user was actually reading the document or he/she, for example, went out to lunch, leaving the browser opened. What is more, the user may be accessing cached information (either from a local cache or from an intermediate proxy server cache) and it is not possible to detect these events on the server side.

Also, if a user is not logged in and he/she does not accept cookies, we are unable to track him/her. In such cases, the only available information that could potentially help us to track the user is his/her IP address. However, many users can share the same IP and, what is more, one user can have many IP addresses even in the same session. The only reliable tracking mechanisms are cookies and requiring users to log in in order to access relevant contents [16].

From this brief description of data problems we can conclude that for applying collaborative filtering, explicitly given data with low sparsity are preferred to implicitly collected data with high sparsity. The worst case scenario is having highly sparse data derived from Web logs. When so, why would we want to apply collaborative filtering to Web logs? The answer is that collecting data in such manner requires no effort from the users and also, the users are not obliged to use any kind of specialized Web browsing software. This "conflict of interests" is illustrated in Fig. 1.

4 Evaluation Platform

To understand the influence of highly sparse server-side collected data on the accuracy of collaborative filtering, we built an evaluation platform. This platform is a set of modules arranged into a pipeline. The pipeline consists of the following four consecutive steps: (i) importing a user-item matrix (in the case of implicit preferences, data needs to be processed prior to entering the pipeline), (ii) splitting the data according to an evaluation protocol, (iii) setting a collaborative filtering algorithm (in the case of the kNN algorithm we also need to specify a similarity measure), (iv) making predictions about users' ratings and collecting evaluation results. The platform is illustrated in Fig. 2.

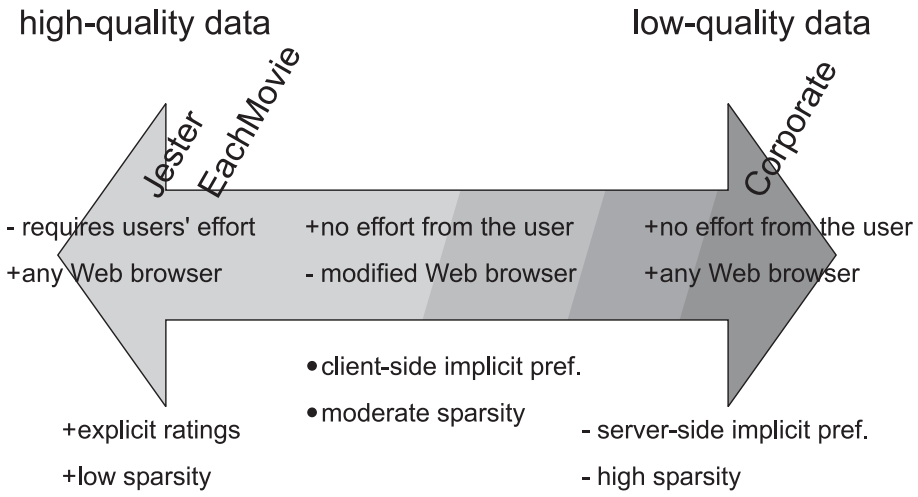


Fig. 1. Data characteristics that influence the data quality and the positioning of the three datasets used in our experiments according to their properties.

Let us briefly discuss some of these stages. Ratings from each user in the test set are further partitioned into "given" and "hidden" ratings, according to the evaluation protocol. For example, 30% of randomly selected ratings from a particular user are hidden, the rest are treated as our sole knowledge about the user (i.e. given ratings). Given ratings are used to form neighborhoods or to build models, while hidden ratings are used to evaluate the accuracy of the selected collaborative filtering algorithm. The algorithm predicts the hidden ratings and since we know their actual values, we can compute the mean absolute error (MAE) or apply some other evaluation metric.

5 Data Description

For our experiments we used three distinct datasets. The first dataset was EachMovie (provided by Digital Equipment Corporation) which contains explicit ratings for movies. The service was available for 18 months. The second dataset with explicit ratings was Jester (provided by [9]) which contains ratings for jokes, collected over a 4-year period. Users were using a scrollbar to express their ratings – they had no notion of actual values. The third dataset was derived from real-life corporate Web logs containing accesses to an internal digital library of a fairly large company. The time-span of acquired Web logs is 920 days. In this third case the users' preferences are implicit and collected on the server side, which implies the lowest data quality for collaborative filtering.

In contrast to EachMovie and Jester, real-life corporate Web logs first needed to be extensively preprocessed. Raw logs contained over 9.3 million requests.

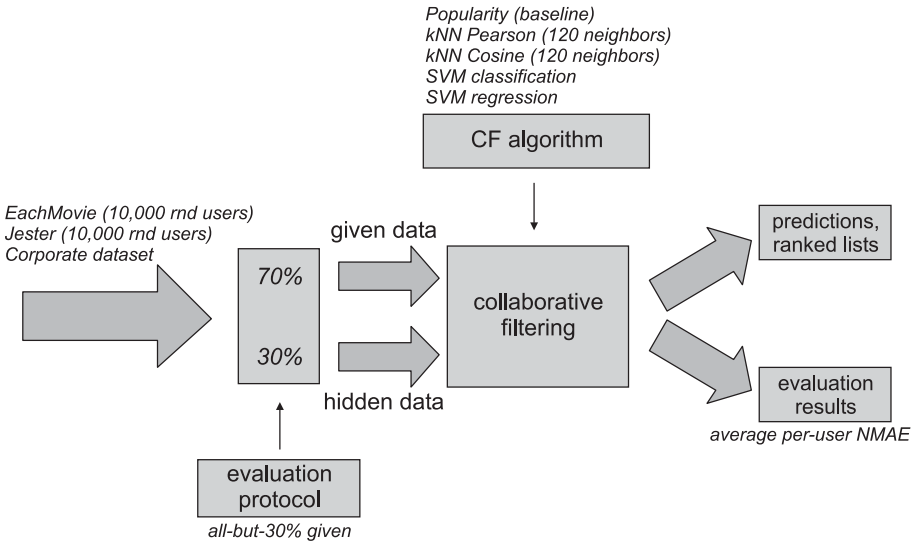


Fig. 2. The evaluation platform. The notes in *italics* illustrate our experimental setting (see Sect. 6).

First, failed requests, redirections, posts, and requests by anonymous users were removed. We were left with slightly over 1.2 million requests (14% of all the requests). These requests, however, still contained images, non-content pages (such as index pages), and other irrelevant pages. What is more, there were several different collections of documents in the corporate digital library. It turned out that only one of the collections was suitable for the application of collaborative filtering. Thus, the amount of potentially relevant requests dropped drastically. At the end we were left with only slightly over 20,500 useful requests, which represents 0.22% of the initial database size.

The next problem emerged from the fact that we needed to map implicit preferences contained in log files into explicit ratings. As already explained, this is not a trivial task. The easiest way to do this is to label items as 1 (accessed) or 0 (not accessed) as also discussed in [4]. The downside of this kind of mapping is that it does not give any notion of likes and dislikes. [7] have shown linear correlations between the time spent reading a document and the explicit rating given to that same document by the same user (this was already published by [13]). However, their test-users were using specialized client-side software, which made the collected data more reliable (hence, in their case, we talk about client-side implicit preferences). Despite this fact we decided to take reading times into account when preprocessing Web logs.

We plotted reading times inferred from consecutive requests onto a scatter plot shown in Fig. 3. The x-axis shows requests ordered by their time-stamps, and the y-axis shows the inferred reading time on a logarithmic scale. We can see that

the area around 24 hours is very dense. These are the last accesses of a day. People went home and logged in again the next day, which resulted in an approximately 24-hour "reading" time. Below the 24-hour line, at an approximately 10-hour reading time, a gap is evident. We decided to use this gap to define outliers – accesses above the gap are clearly outliers. We decided to map reading times onto a discrete 3-value scale (ratings being 1="not interesting", 2="interesting", and 3="very interesting"). Somewhat ad-hoc (intuitively) we defined two more boundaries: one at 20 seconds and another at 10 minutes. Since items were research papers and 20 seconds is merely enough to browse through the abstract, we decided to label documents with reading times below 20 seconds as "not interesting". Documents with reading times between 20 seconds and 10 minutes were labelled as "interesting" and documents with reading times from 10 minutes to 10 hours were labelled as "very interesting". We decided to keep the outliers due to the lack of data. They were labelled as "interesting" which is the most "neutral" value on our rating scale. Since we had no reliable knowledge about the outliers, this should have minimized the error we made by taking them into account.

Table 1 shows the comparison between the three datasets. It is evident that a low number of requests and somewhat ad-hoc mapping onto a discrete scale are not the biggest issues with our corporate dataset. The concerning fact is that the average number of ratings per item is only 1.22, which indicates extremely poor overlapping. Sparsity is consequently very high (99.93%). The other two datasets are much more promising. The most appropriate is the Jester dataset with very low sparsity, followed by EachMovie with higher sparsity but still relatively high average number of ratings per item. Also, the latter two contain explicit ratings, which means that they are more reliable than the corporate dataset (see also Fig. 1).

Table 1. The comparison between the three datasets.

	Ratings		Size			Sparsity		
	Explicit/ implicit	Scale	Num of users	Num of items	Num of ratings	%**	Avg # of r'tings/usr	Avg # of ratings/item
EachMovie	Explicit	Discrete 0–5	61,131	1,622	2,558,871	97.42	41.86	1,577.60
Jester	Explicit	Continuous –10 – +10	73,421	100	4,136,360	43.66	56.34	41,363.60
Corporate	Implicit	Discrete 1–3*	1,850	16,941	20,669	99.93	11.17	1.22

*after preprocessing
**computed as the number of missing values divided by the user-item matrix size (i.e. the number of rows times the number of columns)

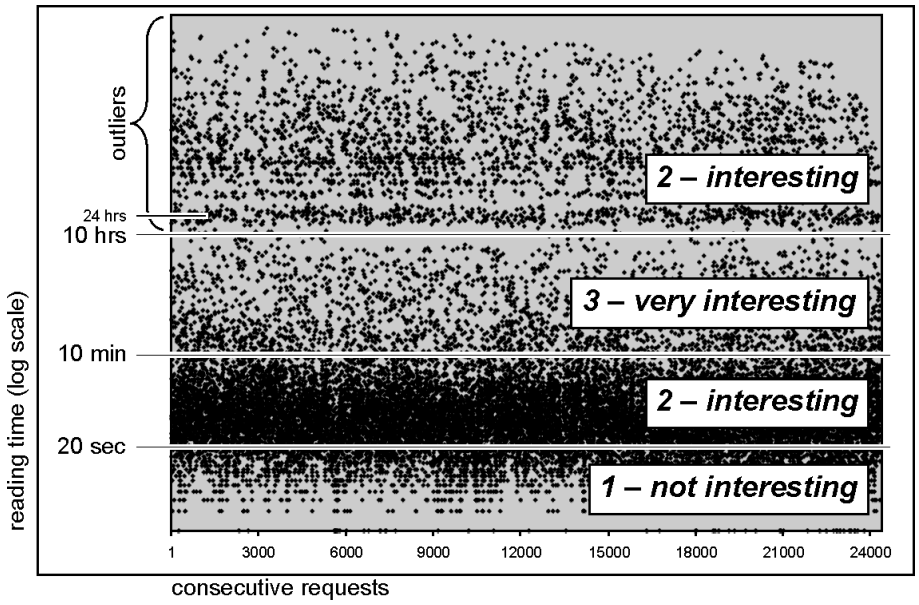


Fig. 3. Mapping implicit preferences contained in the corporate Web logs onto a discrete 3-value scale.

6 Experimental Setting

We ran a series of experiments to see how the accuracy of collaborative filtering recommendations differs between several different approaches and the three different datasets (from EachMovie and Jester we considered only 10,000 randomly selected users to speed up the evaluation process). Ratings from each user were partitioned into "given" and "hidden" according to the "all-but-30%" evaluation protocol. The name of the protocol implies that 30% of all the ratings were hidden and the remaining 70% were used to form neighborhoods.

We applied three variants of memory-based collaborative filtering algorithms: (i) k-Nearest Neighbor using the Pearson correlation coefficient (kNN Pearson), (ii) k-Nearest Neighbor using the cosine similarity measure (kNN Cosine), and (iii) the popularity predictor (Popularity). The latter predicts the user's ratings by simply averaging all the available ratings for the given item. It does not form neighborhoods or build models and it provides each user with the same recommendations. It serves merely as a baseline when evaluating collaborative filtering algorithms (termed "POP" in [4]). For the kNN variants we used a neighborhood of 120 users (i.e. k=120), as suggested in [9].

In addition to the variants of a memory-based approach we also applied two variants of a model-based approach: SVM classifier and SVM regression. In general, SVM classifier can classify a new example into one of two classes: positive or negative. If we want to predict ratings, we need a multi-class variant

of SVM classifier, classes being different rating values. The problem also occurs when dealing with a continuous rating scale such as the one of the Jester dataset. To avoid this problem, we simply sampled the scale interval and thus transformed the continuous scale into a discrete one (in our setting we used $0.\overline{3}$ precision to sample the Jester's rating scale).

Although the work of [3] suggests using items as examples, the task of collaborative filtering can equivalently be redefined to view users as examples. Our preliminary results showed that it is best to choose between these two representations with respect to the dataset properties. If the dataset is more sparse "horizontally" (i.e. the average number of ratings per user is lower than the average number of ratings per item), it is best to take users as examples. Otherwise it is best to take items as examples. Intuitively, this gives more training examples for building models which are consequently more reliable. With respect to the latter, we used users as examples when dealing with EachMovie (having on average 41.86 ratings per user vs. 1,577.60 ratings per item) and Jester datasets (having on average 56.34 ratings per user vs. 41,363.60 ratings per item) and items as examples when dealing with the corporate dataset (having on average 11.17 ratings per user vs. 1.22 ratings per item).

We combined several binary SVM classifiers in order to perform multi-class classification. Let us explain the method that was used on an example. We first transform the problem into a typical machine learning scenario with ordered class values as explained earlier in the text. Now, let us consider a discrete rating scale with values from 1 to 5. We need to train 4 SVMs to be able to classify examples into 5 different classes (one SVM can only decide between positive and negative examples). We train the first SVM to be able to decide weather an example belongs to class 1 (positive) or to any of the classes 2–5 (negative). The second SVM is trained to distinguish between classes 1–2 (positive) and classes 3–5 (negative). The third SVM distinguishes between classes 1–3 (positive) and classes 4–5 (negative), and the last SVM distinguishes between classes 1–4 (positive) and class 5 (negative). In order to classify an example into one of the 5 classes, we query these SVMs in the given order. If the first one proves positive, we increase the number of votes for class 1 otherwise for classes 2–5, if the second one proves positive, we increase the number of votes for classes 1 and 2 otherwise for classes 3–5, and so on in that same manner. The class with the maximal number of votes is the final outcome of our multi-class SVM classifier. If there are several classes with the same number of votes, the average class value is computed (in the case of a discrete scale, the average value is rounded to the nearest integer value). We refer to this multi-class classifier as "head vs. tail voting". We used binary SVM classifier as implemented in TextGarden (`http://www.textmining.net`) with a linear kernel and $cost = 0.1$. We built a model only if there were at least 7 positive and at least 7 negative examples available (because our preliminary experiments showed that this is a reasonable value to avoid building unreliable models).

Another variant of a model-based approach is SVM regression which is much more suitable for the application to collaborative filtering than SVM classifier.

It can directly handle continuous and thus also ordered discrete class values. This means we only need to train one model as opposed to SVM classifier where several models need to be trained. We used SVM regression as implemented in LibSvm [5] with a linear kernel and $cost = 0.1$, $\epsilon = 0.1$. As in the case of SVM classifier, we built a model only if there were at least 14 examples available.

Altogether we ran 5 experiments for each dataset-algorithm pair, each time with a different random seed (each time we also selected a different set of 10,000 users from EachMovie and Jester). When applying collaborative filtering to the corporate dataset, we made 10 repetitions (instead of 5) since this dataset is smaller and highly sparse, which resulted in less reliable evaluation results. Thus, we ran 100 experiments altogether.

We decided to use normalized mean absolute error (NMAE) as the accuracy evaluation metric. We first computed NMAE for each user, then we averaged it over all the users (termed "per-user NMAE") [10]. MAE is extensively used for evaluating collaborative filtering accuracy and was normalized in our experiments to enable us to compare evaluation results over different datasets.

7 Evaluation Results

We present the results of experiments performed on the three datasets using the described experimental setting (see Sect. 6). We used two-tailed paired Student's t-Test with significance level $\alpha = 0.005$ to determine if the differences in results are statistically significant.

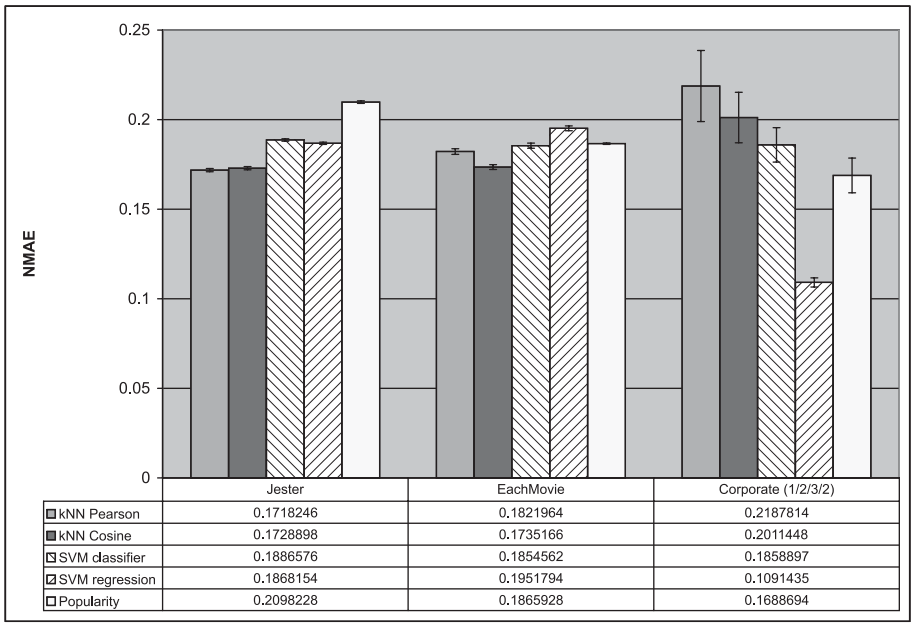
We need to point out that in some cases the algorithms are unable to predict the ratings as the given ratings do not provide enough information for the prediction. For instance, Popularity is not able to predict the rating for a particular item if there are no ratings in the given data for that item. When calculating the overall performance we exclude such cases from the evaluation as we are mainly interested in the quality of prediction when possible, even if the percentage of available predictions is low. We prefer the system to provide no recommendation if there is not enough data for a reasonably reliable prediction.

As mentioned earlier in Sect. 3, the three datasets we used have different characteristics that influence the accuracy of predictions. Jester is the dataset with the lowest sparsity and is thus in our case the most suitable for the application of collaborative filtering. We see that the kNN methods significantly outperform the other three methods. kNN Pearson slightly yet significantly outperforms kNN Cosine. Both SVM approaches clearly outperform Popularity. Of the two, SVM regression performs better, significantly outperforming SVM classifier.

EachMovie is sparser than Jester yet much more suitable for the application of collaborative filtering than the corporate dataset. As in the previous case, the kNN approaches perform best (of the two, kNN Cosine performs significantly better than kNN Pearson). However, kNN Pearson does not outperform SVM classifier significantly (it does, however, significantly outperform SVM regression

and Popularity). Interestingly, SVM regression is outperformed by all other four algorithms (including Popularity).

The corporate dataset is the worst of the three – it is extremely sparse and collected implicitly on the server side. It reveals the weakness of the kNN approach – lack of overlapping values results in unreliable neighborhoods. Popularity significantly outperforms the two kNN approaches and it also significantly outperforms SVM classifier. SVM classifier significantly outperforms kNN Pearson while the difference between SVM classifier and kNN Cosine, on the other hand, is insignificant. The winning approach on this dataset is by far SVM regression – not only it outperforms the other four approaches, it is also more stable (this is evident from the error bars in Fig. 4) and is able to predict the highest number of ratings. In this paper we are not concerned with the inability to predict but it is still worth mentioning that SVM regression can predict 73% of the hidden ratings, Popularity 24%, and the kNN approaches only around 8% of the hidden ratings.



	Jester	EachMovie	Corporate (1/2/3/2)
kNN Pearson	0.1718246	0.1821964	0.2187814
kNN Cosine	0.1728898	0.1735166	0.2011448
SVM classifier	0.1886576	0.1854562	0.1858897
SVM regression	0.1868154	0.1951794	0.1091435
Popularity	0.2098228	0.1865928	0.1688694

Fig. 4. The results of the experiments.

To gain a better understanding of how the sparsity level (computed as the number of missing values in the user-item matrix divided by the matrix size, i.e. the number of rows times the number of columns) influences the prediction accuracy of the evaluated algorithms, we performed an additional experiment in which we gradually increased sparsity of a subset of the Jester dataset (1000

randomly selected users). We confronted the two SVM approaches with kNN Cosine ($k = 120$) and Popularity at 50%, 60%, 70%, 80%, 90%, 95%, and 98% sparsity. This time, we did not impose any example count restrictions when building SVM models. The results are depicted in Fig. 5. We used two-tailed paired t-Test with significance level $\alpha = 0.05$ to determine if the differences in results are statistically significant. At 98% sparsity, kNN is significantly outperformed by the other three algorithms. Popularity significantly outperforms SVM classifier while the difference between Popularity and SVM regression is statistically insignificant. Also, the difference between the two SVM approaches is statistically insignificant.

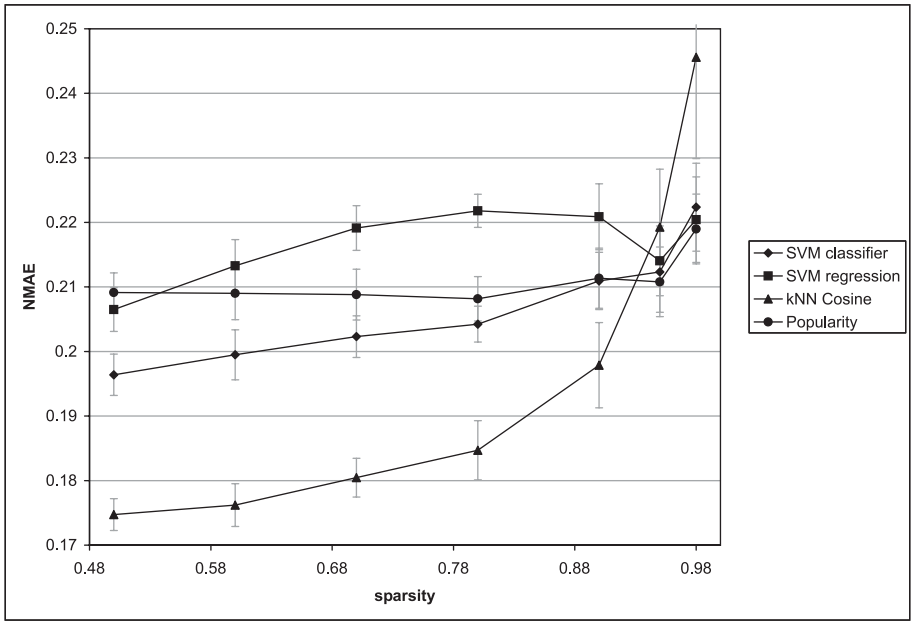


Fig. 5. The results of the additional experiment in which we gradually increased sparsity of the Jester dataset to gain a better understanding of how the sparsity level influences the prediction accuracy of the evaluated algorithms.

8 Conclusions and Future Work

In our experimental setting we confronted the k-Nearest Neighbor algorithm with Support Vector Machine in the collaborative filtering framework. Our main experiment showed that kNN is dominant on the datasets with relatively low sparsity (i.e. EachMovie and Jester) and that it fails on the corporate dataset as it is unable to form reliable neighborhoods (see Fig. 4). On the corporate

dataset, SVM regression is by far the best performing algorithm. We cannot conclude that SVM regression performs best on *any* highly sparse dataset as we would also need to take other data characteristics (such as distribution of rating values) into account in order to perform more complete analyses. We can, however, say that SVM regression handles sparsity and also uneven distribution of rating values in our corporate dataset better than the other four algorithms.

We believe that in general kNN is dominant on datasets with relatively low sparsity. As the sparsity increases, kNN starts failing as it is unable to form reliable neighborhoods. Our second experiment shows that kNN fails (in comparison to Popularity and the SVM approaches) at around 95% sparsity (see Fig. 5). In the case of such highly sparse data it is therefore better to use another approach. We propose the use of SVM regression as it has several advantages over SVM classifier and Popularity: (i) only one model needs to be built for each item (in contrast to building several binary models in the case of SVM classifier), (ii) the prediction is faster than with SVM classifier as only one model needs to be queried, (iii) SVM regression is able to predict more ratings than Popularity (as already explained, in this work we are not concerned with the inability to predict, nevertheless, this is still one of the strong points of SVM regression), and last but not least, (iv) SVM regression works better with eccentric users than Popularity (the analyses with eccentric users are beyond the scope of this work; intuitively Popularity only satisfies the "average" user while SVM regression is more sophisticated and gives good recommendations also to eccentric users).

What is also evident from the evaluation results is that an inadequate number of values in the corporate dataset represents our biggest problem. We will not be able to evaluate collaborative filtering algorithms on the given corporate dataset properly, until we reduce its sparsity. One idea is to apply LSI (Latent Semantic Indexing) [8] or to use pLSI (Probabilistic Latent Semantic Indexing) [11] to reduce the dimensionality of the user-item matrix, which consequently reduces sparsity. Another idea, which we believe is even more promising in this context, is to incorporate textual contents of the items. There were already some researches done on how to use textual contents to reduce sparsity and improve the accuracy of collaborative filtering [14]. Luckily, we are able to obtain textual contents for the given corporate dataset.

Also evident is the fact that the cosine similarity measure works just as well (or even better) as the Pearson correlation measure on EachMovie and Jester. Early researches show much poorer performance of the cosine similarity measure [4].

As a side-product we noticed that the true value of collaborative filtering (in general) is shown yet when computing NMAE over some top percentage of eccentric users. We defined eccentricity intuitively as MAE (Mean Absolute Error) over the overlapping ratings between "the average user" and the user in question (greater MAE represents greater eccentricity). The average user was defined by averaging ratings for each particular item. This is based on the intuition that the ideal average user would rate every item with the item's average rating. The incorporation of the notion of eccentricity can give the more sophisticated algo-

rithms a fairer trial. We computed average per-user NMAE only over the top 5% of eccentric users. The supremacy of the kNN algorithms over Popularity became even more evident. In near future, we will define an accuracy measure that will weight per-user NMAE according to the user's eccentricity, and include it into our evaluation platform.

Acknowledgements

This work was supported by the Slovenian Research Agency and the IST Programme of the European Community under SEKT, Semantically Enabled Knowledge Technologies (IST-1-506826-IP) and PASCAL Network of Excellence (IST-2002-506778). (This publication only reflects the authors' views.) The authors would also like to thank Tanja Brajnik for her help.

References

1. C. C. Aggarwal, J. L. Wolf, K.-L. Wu, and P. S. Yu. Horting hatches an egg: A new graph-theoretic approach to collaborative filtering. In *Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1999.
2. P. Baldi, P. Frasconi, and P. Smyth. *Modeling the Internet and the Web: Probabilistic Methods and Algorithms*. Wiley, New York, 2003.
3. D. Billsus and M. J. Pazzani. Learning collaborative information filers. In *Proceedings of the 15th International Conference on Machine Learning*, 1998.
4. J. S. Breese, D. Heckerman, and C. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence*, 1998.
5. C.-C. Chang and C.-J. Lin. *LibSvm: A Library for Support Vector Machines*, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.
6. D. M. Chickering, D. Heckerman, and C. Meek. A bayesian approach to learning bayesian networks with local structure. In *Proceedings of the 13th Conference on Uncertainty in Artificial Intelligence*, 1997.
7. M. Claypool, P. Le, M. Wased, and D. Brown. Implicit interest indicators. In *Proceedings of ACM 2001 Intelligent User Interfaces Conference*, 2001.
8. S. Deerwester, S. T. Dumais, and R. Harshman. Indexing by latent semantic analysis. *Journal of the Society for Information Science*, 41(6):391–407, 1990.
9. K. Goldberg, T. Roeder, D. Gupta, and C. Perkins. Eigentaste: A constant time collaborative filtering algorithm. *Information Retrieval*, (4):133–151, 2001.
10. J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems*, 22(1):5–53, 2004.
11. T. Hofmann. Probabilistic latent semantic analysis. In *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence*, 1999.
12. T. Hofmann. Latent semantic models for collaborative filtering. *ACM Transactions on Information Systems*, 22(1):89–115, 2004.
13. J. A. Konstan, B. N. Miller, D. Maltz, J. L. Herlocker, L. R. Gordon, and J. Riedl. Grouplens: Applying collaborative filtering to usenet news. *Communications of the ACM*, 40(3):77–87, 1997.

14. P. Melville, R. J. Mooney, and R. Nagarajan. Content-boosted collaborative filtering for improved recommendations. In *Proceedings of the 18th National Conference on Artificial Intelligence*, 2002.
15. P. Resnick, N. Iaocvou, M. Suchak, P. Bergstrom, and J. Riedl. Grouplens: An open architecture for collaborative filtering for netnews. In *Proceedings of ACM 1994 Conference on Computer Supported Cooperative Work*, pages 175–186, 1994.
16. M. Rosenstein and C. Lochbaum. What is actually taking place on web sites: E-commerce lessons from web server logs. In *Proceedings of ACM 2000 Conference on Electronic Commerce*, 2000.
17. B. Sarwar, G. Karypis, J. Konstan, and J. Reidl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th International Conference on World Wide Web*, 2001.
18. K. Yu, X. Xu, M. Ester, and H.-P. Kriegel. Selecting relevant instances for efficient and accurate collaborative filtering. In *Proceedings of the 10th International Conference on Information and Knowledge Management*, 2001.
19. C. Zeng, C.-X. Xing, and L.-Z. Zhou. Similarity measure and instance selection for collaborative filtering. In *Proceedings of the 12th International World Wide Web Conference*, 2003.

View publication stats