Comparison of collaborative filtering algorithms: Limitations of current techniques and proposals for scalable, high-performance recommender systems

4 authors:

Fidel Cacheda
University of A Coruña
102 PUBLICATIONS **1,120** CITATIONS
SEE PROFILE

Victor Carneiro
University of A Coruña
16 PUBLICATIONS **347** CITATIONS
SEE PROFILE

Diego Fernández
University of A Coruña
24 PUBLICATIONS **469** CITATIONS
SEE PROFILE

Vreixo Formoso
University of A Coruña
13 PUBLICATIONS **315** CITATIONS
SEE PROFILE

Comparison of Collaborative Filtering Algorithms: Limitations of Current Techniques and Proposals for Scalable, High-Performance Recommender Systems

FIDEL CACHEDA, VÍCTOR CARNEIRO, DIEGO FERNÁNDEZ,
and VREIXO FORMOSO, University of A Coruña

The technique of collaborative filtering is especially successful in generating personalized recommendations. More than a decade of research has resulted in numerous algorithms, although no comparison of the different strategies has been made. In fact, a universally accepted way of evaluating a collaborative filtering algorithm does not exist yet. In this work, we compare different techniques found in the literature, and we study the characteristics of each one, highlighting their principal strengths and weaknesses. Several experiments have been performed, using the most popular metrics and algorithms. Moreover, two new metrics designed to measure the precision on good items have been proposed.

The results have revealed the weaknesses of many algorithms in extracting information from user profiles especially under sparsity conditions. We have also confirmed the good results of SVD-based techniques already reported by other authors. As an alternative, we present a new approach based on the interpretation of the tendencies or differences between users and items. Despite its extraordinary simplicity, in our experiments, it obtained noticeably better results than more complex algorithms. In fact, in the cases analyzed, its results are at least equivalent to those of the best approaches studied. Under sparsity conditions, there is more than a 20% improvement in accuracy over the traditional user-based algorithms, while maintaining over 90% coverage. Moreover, it is much more efficient computationally than any other algorithm, making it especially adequate for large amounts of data.

Categories and Subject Descriptors: H.3.3 [**Information Search and Retrieval**]: Information filtering

General Terms: Algorithms, Performance

Additional Key Words and Phrases: Collaborative filtering, recommender systems

ACM Reference Format:
Cacheda, F., Carneiro, V., Fernández, D., and Formoso, V. 2011. Comparison of collaborative filtering algorithms: Limitations of current techniques and proposals for scalable, high-performance recommender systems. ACM Trans. Web 5, 1, Article 2 (February 2011), 33 pages.
DOI = 10.1145/1921591.1921593 http://doi.acm.org/10.1145/1921591.1921593

1. INTRODUCTION

The Internet, with hundreds of millions of pages worldwide, has become the greatest source of information that has ever existed. In this context, information retrieval systems are essential tools to guide users to the information they are seeking. Specifically, users demand personalized search systems, not just limited to retrieving the most relevant items, but also more adequate for their particular tastes or interests.

This is the aim of *recommender systems*. They use information about users, user profiles, to predict the utility or relevance of a particular item, thus providing personalized

This work was partly supported by the Spanish government, under project TIN 2009-14203.
Author's address: F. Cacheda, University of A Coruña; email: fidel@udc.es.

DOI 10.1145/1921591.1921593 http://doi.acm.org/10.1145/1921591.1921593

recommendations. Recommender systems have proven to be useful in contexts such as e-commerce, and they surely have a promising future in many other domains, like Web search engines, digital TV program recommenders, etc.

Until now, recommender systems have been used basically in two tasks. First, they have been used to predict the utility of a given item to the user [Shardanand and Maes 1995; Resnick et al. 1994]. In this task, often known as *annotation in context*, the user first selects the item (or items) in which he is interested. This is usually done after performing a search, browsing an online catalog, etc. The recommender system then predicts the rating the user would give to that item.

Second recommender systems have been used, to recommend a list of items to the user. In this case, often called the *find good items* task, the system chooses the items that it considers the most relevant. Actually, recommender systems can also be used for other tasks, such as *find all good items*, *recommended sequence*, *just browsing* or *find credible recommender* [Herlocker et al. 2004], although these have not yet attracted much interest among researchers.

The technique of collaborative filtering [Goldberg et al. 1992], that recommends items based on the opinions of other users, is very popular, especially in e-commerce, given its good results. In recent years, numerous algorithms based on different ideas and concepts have been developed to address this problem. Unfortunately, works that compare these techniques are scarce, making it difficult to select the best algorithm (or algorithms) in a given situation.

In this work, we compare different techniques of collaborative filtering, identifying their main advantages and limitations. We focus on both *annotation in context* and *find good items tasks*. The evaluation was performed following the most common methodology and metrics found in the literature. Moreover, two new metrics that improve the evaluation of the accuracy of algorithms in the task of *find good items* are proposed.

A new collaborative filtering algorithm, based on a novel concept, tendencies or variations of items and users, is also presented. Our experiments show that it is the best algorithm, in general, especially under sparse data conditions thanks to an accuracy equivalent to the best techniques studied and much lower computational complexity. Thus, it is an ideal candidate for online systems or for systems involving many users and/or items.

The article is structured as follows. First, the notation used in this article is presented. Section 3 briefly describes the state of the art, placing special interest on the evaluation of recommender systems. The algorithms evaluated and the metrics applied in this evaluation are also presented. In Section 4, the tendencies-based algorithm proposed in this work is explained. Then, we introduce two new metrics, that measure the algorithm precision on relevant items.

After, the experiments performed are presented and the results discussed comparing the behavior of various algorithms under different situations. Several aspects of each algorithm, such as the effect of the different parameters on the results or the variation in accuracy depending on the rating matrix sparsity, are analyzed. The differences between memory-based and model-based algorithms are highlighted. Moreover, the importance of a good fit between model and data is also considered. The algorithm proposed is especially analyzed confirming that, despite its simplicity, its results are notably better than those obtained by more complex techniques.

Finally, we present the conclusions of this study and discuss what direction to take in future works.

2. NOTATION

Collaborative filtering techniques, as a kind of recommender system, aim to recommend useful *items* to *users*. In the most typical scenario, these techniques deal with a set of users, $U = \{u_1, u_2, \ldots, u_m\}$, and a set of items, $I = \{i_1, i_2, \ldots, i_n\}$. Each user, $u_i \in U$, has an associated profile consisting of the subset of items he has rated, $I_u \subseteq I$, and the corresponding rating for each item. Similarly, the subset of users that have rated a certain item, $U_i \subseteq U$, is defined. The active user, the user for whom a prediction is being obtained, is denoted as u_a.

The ratings usually correspond to integer numbers in a certain range, being R, the set of possible ratings.

Using user profiles, the rating matrix, V, representing the user ratings, is defined. Each element of V, $v_{ui} \in R \cup \varnothing$, denotes the rating given by user $u \in U$ to item $i \in I$, the value $\varnothing$ indicating that the user has not rated the item yet.

In fact, the objective of the Collaborative filtering algorithm is to predict the value of v in these cases. Let us denote as $p_{ui} \in R \cup \varnothing$ the prediction the algorithm makes for the rating of item $i \in I$ by user $u \in U$. If the algorithm is not able to make this prediction, $p_{ui} = \varnothing$.

Finally, we define the subset of user ratings, $v_{u.} = \{v_{ui} \in V / i \in I_u\}$, and the subset of item ratings, $v_{.i} = \{v_{ui} \in V / u \in U_i\}$. We denote the user mean rating as $\overline{v_{u.}}$ and the item mean rating as $\overline{v_{.i}}$.

3. STATE OF THE ART

3.1. Content-Based Filtering vs. Collaborative Filtering

The first approximations to information filtering were based on content [Foltz and Dumais 1992]. These systems select which items to recommend based on their content. Therefore, the user profile is a representation of the content in which the user is interested. This kind of filtering is especially effective when retrieving text documents, where each document is represented by a set of keywords. However, these systems have several limitations [Shardanand and Maes 1995].

First, the items should be analyzable by a machine. This is difficult when retrieving multimedia information where machine perception of the content (colors, textures, etc.) differs greatly from user perception. Although the assignment of attributes by a person (annotated multimedia content) solves this problem, at least in part, content-based filtering is insufficient to deal with much of the information available today.

Another major problem with content-based filtering is its inability to evaluate the quality of an item. For example, it cannot distinguish a good article from a bad one if both articles use similar words. In fact, the quality of an item is a highly subjective feature that depends on the tastes, ideas, culture, etc., of each person and that would be difficult for a machine to analyze.

Finally, content-based filtering does not have a way of finding serendipitous items that are interesting for the user, that is, really good items that are not apparently related to the user profile.

Collaborative filtering systems [Shardanand and Maes 1995] are less sensitive to these problems since they are not based on the content of items but rather on the opinions of other users. The system will recommend items that have received high ratings by other users with similar tastes or interests. In these techniques, the items are actually rated by people. Thus, the system does not need to analyze content (and, therefore, it is valid for any type of item including nonannotated multimedia content), and the quality or subjective evaluation of the items is also considered.

In collaborative filtering–based systems, the user profile is the set of ratings given to different items. These ratings can be captured explicitly, that is, by asking the user, or implicitly by observing his/her interaction with the system. Generally, the rating is represented as a unary value (showing only the relevant items), binary (allowing to distinguish between good and bad items) or, more commonly, as a numerical value on a finite scale.

The user ratings are stored in a table known as the rating matrix. This table is processed in order to generate the recommendations. Depending on how the data of the rating matrix are processed, two types of algorithms, memory-based and model-based, can be differentiated.

Memory-based algorithms use the whole table to compute their prediction. Generally, they use similarity measures to select users (or items) that are similar to the active user. Then, the prediction is calculated from the ratings of these neighbors. (This is why they are also called neighbor-based.) Most of these algorithms can be classified as user-based algorithms or item-based algorithms depending on whether the process of getting neighbors is focused on finding similar users [Resnick et al. 1994; Shardanand 1994] or items [Sarwar et al. 2001].[1]

Model-based algorithms first construct a model to represent the behavior of the users and, therefore, to predict their ratings. The parameters of the model are estimated offline using the data from the rating matrix. In the literature there are different approaches, most related to machine learning [Marlin 2004]: based on linear algebra methods (SVD [Billsus and Pazzani 1998; Sarwar et al. 2000], factor analysis [Canny 2002], PCA [Goldberg et al. 2001], MMMF [Rennie and Srebro 2005], etc.), clustering [Ungar and Foster 1998; Kohrs and Mérialdo 1999], neural networks [Billsus and Pazzani 1998], graphs [Aggarwal et al. 1999], or probabilistic methods, such as Bayes networks [Breese et al. 1998] or latent class models [Si and Jin 2003; Hofmann 2004].

Generally, memory-based algorithms are simpler, but they obtain reasonably accurate results. However, they present serious scalability problems given that the algorithm has to process all the data to compute a single prediction. With a high number of users or items, these algorithms are not appropriate for online systems which recommend items in real time. Furthermore, they are much more sensitive than the model-based to some common problems of recommender systems, among which we highlight the following.

- Sparsity of the rating matrix [Sarwar et al. 2001; Huang et al. 2004]. In most recommender systems, each user rates only a small subset of the available items, so most of the cells in the rating matrix are empty. In such cases, finding similarities among different users or items is challenging.
- Cold-start [Schein et al. 2002]. Related to the previous problem, this one deals with the difficulty in making recommendations for users recently introduced into the system. In such cases, the users have not rated enough items yet, so the recommender system is unable to guess their interests. Some systems overcome this problem by forcing the user first to rate a given set of items. However, these initial ratings could introduce biases into the system. Finally, note that the cold-start problem also affects new items as they will not be recommended until enough users have rated them.
- Shilling [Lam and Riedl 2004; Chirita et al. 2005]. Recommender systems could suffer spam attacks, mainly from users interested in misleading the system to recommend a certain product. Several techniques affecting both neighbor-based [Mobasher et al. 2007] and model-based [Sandvig et al. 2007] algorithms have been studied in the past.

These problems are reduced, although only partly, by the model-based algorithms, thanks to their ability to obtain underlying characteristics in the data and thereby

[1] Given the relations between items tend to be relatively static, item-based algorithms could calculate these relations beforehand. Hence, in the literature, it is not unusual to see them treated as model-based algorithms.

extract more information. Moreover, they tend to be faster in prediction time than the memory-based approaches, although the construction of the model requires a considerable amount of time.

However, model-based algorithms still present several problems. Many models are extremely complex, as they have a multitude of parameters to estimate, and many times they are too sensitive to data changes. Other times, the assumptions of the model do not fit the data, leading to wrong recommendations. In practice, many theoretical models cannot be applied to real data. Moreover, model construction (and update, when new data are added) usually takes a long time.

To avoid these problems, some authors have developed algorithms that use models that are fast and easy to calculate [Lemire and Maclachlan 2005]. In recent years, algorithms based on a variation of singular value decomposition (SVD), where only known ratings are modeled, have become very popular [Funk 2006; Paterek 2007]. These can be easily trained using a simple gradient descent technique to achieve both good accuracy and efficiency.

Other authors have combined techniques from both model-based and memory-based algorithms [Pennock et al. 2000] to take advantage of the best of both worlds. For example, Koren [2008] obtained good results by integrating a regularized SVD model with a model inspired on neighbor-based algorithms.

Finally, a third type of recommender systems has been proposed. They are the so called hybrids that combine content-based methods with collaborative filtering [Melville et al. 2001; Basilico and Hofmann 2004]. The content-based information is useful to fill in missing ratings, thus, minimizing both sparsity and cold-start problems.

3.2. Evaluation of Collaborative Filtering Algorithms

After almost two decades of research on collaborative filtering algorithms, their evaluation still presents several challenges and problems, summarized in three major questions.

(1) What should we evaluate in a recommender system?

A consensus still has not been reached on the characteristics that should be evaluated. The most common tendency is to evaluate the accuracy or precision of the algorithm. Herlocker et al. [2004] identify three types of metrics to measure the quality of an algorithm.

—*Prediction accuracy*. This measures the difference between the rating the system predicts and the real rating. The most popular of this kind of metric is the mean absolute error (MAE). Other related metrics, such as mean squared error (MSE), root mean squared error (RMSE) or normalized mean absolute error, are also used. RMSE has become extremely popular in recent years, after being used in the Netflix Prize competition [Bennett and Lanning 2007].

—*Classification accuracy*. This measures how well the system differentiates good items from bad ones. Examples of well-known metrics of this type are Precision, Recall and ROC. These metrics are appropriate for the *find good items* task, especially when the preferences of the users are binary. In contrast, if the users express their preferences on a numeric range, these metrics do not evaluate the correct order of items on the recommendation list. They only measure whether recommended items are good without considering which item is better. Of course, this is not always adequate. For example, users usually focus on the first items on the recommendation list, so often we want the best items at such positions. In such cases, this kind of metrics is not the best alternative.

—*Rank accuracy*. This measures the ability of the system to sort the recommended items like the user would have done. In many cases, this kind of metrics is too sensitive given they ask the system to recommend the best items when, in practice, it would suffice to recommend good items and not necessarily the best. Metrics of this type are those that measure the correlation between the prediction and the real classification (Pearson, Spearman's ρ, and Kendall's τ), the half-life utility metric [Breese et al. 1998] or the normalized distance-based performance measure (NDPM) [Balabanović and Shoham 1997].

Measures that are not related to algorithm accuracy can also be found in the literature. A metric commonly used is coverage, which measures the percentage of items for which the system is able to make a prediction. Coverage of an algorithm is especially important when one must find all the good items and not be limited to recommending just a few.

In recent years, some researchers have also evaluated how satisfied the user is with the recommendations. The idea is to evaluate concepts such as the system ability to recommend novel and serendipitous items (a system that only recommends well-known items is not useful for the user), the way a user interacts with the system (for example, how user ratings are introduced), or to what extent a user can trust a system recommendation [Herlocker et al. 2004; McNee et al. 2006].

Other metrics evaluate the efficiency of the algorithm from a computational point of view or its scalability as the number of users or items increases. The behavior of the algorithm with new users or items or its ability to produce good recommendations under sparsity contexts have also been evaluated.

(2) How should an evaluation be performed?

Offline evaluation is the most popular approach in collaborative filtering studies. It is based on a dataset that is divided into two subsets: training and evaluation. The training subset is the data the algorithm knows, that is, the data the algorithm uses to compute the recommendation or rating predictions. These are then compared with the data in the evaluation subset.

However, although this method is widely used, major differences exist among studies, the dataset used, the strategy used to divide the data into training and evaluation subsets, etc., making it difficult to directly compare the results from several works.

The most commonly used datasets are EachMovie, MovieLens [Herlocker et al. 1999], Jester [Goldberg et al. 2001], and, more recently, Netflix [Bennett and Lanning 2007], although there are several works that use proprietary data not available to the general public [Huang et al. 2007]. Synthetic datasets, generated specially to meet a particular property, have also been used.

Regarding the strategies used to build the training subset, there are several alternatives. For example, the method proposed by Breese et al. [1998] is widely used. It consists in constructing the training set taking N ratings from each user.

(3) Which is the best algorithm (or algorithms) in a particular context?

It is difficult to answer this question based on the works found in the literature. Most of them just evaluate the proposed algorithm using the metrics or methodologies that offer the best results, or they just study the most favorable conditions for that particular algorithm. Although the proposal is usually compared with other algorithms, generally, a simple technique, such as user-based, is chosen instead of algorithms that are more recent or related to the context the algorithm was designed for.

Considering the dozens of algorithms developed in the last decade, it is surprising to find the scarcity of works comparing the different techniques. The most outstanding are limited to evaluating a particular family of algorithms, their behavior in specific situations, or simply to comparing a few algorithms with each other.

Breese et al. [1998] performed the first serious attempt to compare different algorithms. They evaluated the behavior of several techniques in three different domains[2] under various training conditions. Fisher et al. [2000] and Calderón-Benavides et al. [2004] are along the same lines, as well as the most recent work by Huang et al. [2007], limited to the evaluation of algorithms based on unary relations[3] widely used in e-commerce. Karypis [2001] and Herlocker et al. [2002] studied item-based and user-based algorithms, respectively, thoroughly comparing the different techniques and their results.

Perhaps the best place to search for a comparison of algorithms is not a research article, but rather the Netflix Prize competition [Bennett and Lanning 2007]. It offers a comparison of the results of the different techniques evaluated with the same dataset and metric (RMSE). However, from a research point of view, the comparison is not that valuable as it is restricted to a given domain and data and, moreover, to a single evaluation subset and metric.

3.3. Metrics

3.3.1. Coverage. This corresponds to the percentage of items the system is able to recommend. Coverage can be used to detect algorithms that, although they present good accuracy, recommend only a small number of items. These are usually very popular items with which the user is already familiar without the help of the system. Therefore, a high coverage value is not only desirable, but it is also helpful to better trust accuracy metrics results.

3.3.2. Prediction Accuracy

—*Mean absolute error* (MAE). The mean absolute error measures the difference, as absolute value, between the prediction of the algorithm and the real rating. It is computed over all the ratings available in the evaluation subset, using the formula:

$$|\overline{E}| = \frac{\sum_i^N |p_i - v_i|}{N}. \tag{1}$$

Despite its limitations when evaluating systems focused on recommending a certain number of items, the simplicity of its calculation and its statistical properties [Herlocker et al. 2004] have made this metric one of the most popular when evaluating recommender systems.

—*Root mean squared error* (RMSE). Related to the previous metric, the root mean squared error, calculated using Equation (2), places greater emphasis on larger errors.

$$|\overline{E}| = \sqrt{\frac{\sum_i^N (p_i - v_i)^2}{N}}. \tag{2}$$

The principal reason for using this metric is that these errors can have the greatest impact on the user decision. For example, on a 5-point scale, a 1-point error may not be perceptible by the user (items rated with either 4 or 5 points are good recommendations), while with a 4-point error, the algorithm could be recommending a very bad item.

[2] Visits to different sections of a website, television programs and movie recommenders.

[3] Where only information about the interest of the users on the items is stored without specifying the rating value.

3.3.3. Classification and Rank Accuracy

- *Precision and recall* [Sarwar et al. 2000]. Precision is defined as the ratio of relevant items to recommended items. Recall is the proportion of relevant items that have been recommended to the total number of relevant items. It is desirable for a system to have high precision and recall values. However, both metrics are inversely related, such that when precision is increased, recall usually diminishes, and vice versa. To consider both precision (P) and recall (R) the measure $F1$ is defined:

$$F1 = \frac{2PR}{P+R}. \tag{3}$$

- *ROC curves, Swets' A Measure*. The metric receiver operating characteristic (ROC), used as an alternative to precision/recall, is especially popular in signal theory. Its use in the evaluation of recommender systems [Herlocker et al. 2004; Huang et al. 2007] allows to interpret graphically the behavior of the system when distinguishing good items (relevant) from bad (nonrelevant). The ROC curve is a graphic representation of recall versus fallout, that is, the relation between misses and hits. The area under the curve, known as Swets' A Measure, offers a measure of the system's ability to distinguish between good and bad items.
- *Half-life utility*. This metric, proposed by Breese et al. [1998], evaluates the usefulness of an ordered list of recommended items. It is based on the fact that items at the beginning of the list have a greater probability of being seen by the user, probability that decreases exponentially as we go down the list. The measure is calculated according to the formula:

$$R_a = \frac{\sum_j \max(v_{aj} - d, 0)}{2^{(j-1)/(\alpha-1)}}, \tag{4}$$

  where d is a neutral vote, slightly negative, and α is the viewing half-life, that is, the position of the item where there is a 50% possibility of it being seen by a user.

3.4. Collaborative Filtering Algorithms

This section presents the collaborative filtering algorithms found in the literature that were evaluated and compared in this work.

3.4.1. User-Based. User-based algorithms, also known as neighborhood-based, are one of the most popular strategies of collaborative filtering. They follow a three-step process.

(1) Calculate the similarity between the active user and the rest of the users.
(2) Select a subset of the users (neighborhood) according to their similarity with the active user.
(3) Compute the prediction using the neighbor ratings.

Since the first proposals of user-based algorithms [Resnick et al. 1994; Shardanand 1994], different techniques to address each of these steps have been studied. Hence, the user-based approach is considered a family of algorithms instead of a single algorithm. Each one combines different strategies for each step. In this work the following variants have been studied.[4]

(1) Computation of similarity between users.
  - Pearson correlation coefficient. This is one of the first techniques proposed [Resnick et al. 1994] and also one of the most popular. Similarity between users

[4] In Herlocker et al. [2002], there is a detailed study of several strategies, including some of the ones analyzed here.

is measured according to their correlation, using the Pearson coefficient:

$$s(a,u) = \frac{\sum_{i \in I_a \cap I_u} (v_{ai} - \overline{v_{a.}})(v_{ui} - \overline{v_{u.}})}{\sqrt{\sum_{i \in I_a \cap I_u} (v_{ai} - \overline{v_{a.}})^2 \sum_{i \in I_a \cap I_u} (v_{ui} - \overline{v_{u.}})^2}}. \tag{5}$$

—Constrained Pearson. Shardanand and Maes [1995] proposed a variant of the Pearson correlation coefficient, consisting of substituting, in Equation (5), the user mean for the central rating (for example, 3 on a scale from 1 to 5). The idea is to take into account the difference between positive (above the central rating) and negative ratings.

$$s(a,u) = \frac{\sum_{i \in I_a \cap I_u} (v_{ai} - 3)(v_{ui} - 3)}{\sqrt{\sum_{i \in I_a \cap I_u} (v_{ai} - 3)^2 \sum_{i \in I_a \cap I_u} (v_{ui} - 3)^2}}. \tag{6}$$

—Vector similarity (Cosine). This is another measure of user similarity that treats users as vectors of item ratings. Then it measures the cosine between the vectors of two users [Breese et al. 1998]. A value close to 1 indicates similarity, while a value close to zero indicates just the opposite.

$$s(a,u) = \sum_{j \in I} \frac{v_{aj}}{\sqrt{\sum_{k \in I_a} v_{ak}^2}} \frac{v_{uj}}{\sqrt{\sum_{k \in I_u} v_{uk}^2}}. \tag{7}$$

—Mean squared difference [Shardanand 1994]. This computes the similarity between users based on the mean difference of the items that both have rated, applying the following formula:

$$msd(a,u) = \frac{\sum_{i \in I_a \cap I_u} (v_{ai} - v_{ui})^2}{|I_a \cap I_u|}. \tag{8}$$

Later, the users whose difference is greater than a certain threshold, L, are discarded and the similarity of the rest is weighted as in the formula:

$$s(a,u) = \frac{L - msd(a,u)}{L}. \tag{9}$$

—Weighted Pearson. This measure is based on the idea of capturing the confidence that can be placed on a neighbor. The problem is that, if two users have few items in common, any of the before mentioned measures can consider them similar simply if, by chance, the ratings of these few items coincide. As the number of items in common increases, this coincidence is more likely to be due to the fact that the users are actually quite similar and not to pure coincidence. So the confidence in the similarity measure increases with the number of items in common. Herlocker et al. [1999] proposed to weigh the Pearson correlation coefficient by the number of items in common, as in the formula:

$$s(a,u) = \begin{cases} s_{pearson}(a,u)\frac{|I_a \cap I_u|}{50} & |I_a \cap I_u| < 50 \\ s_{pearson}(a,u) & otherwise \end{cases} \tag{10}$$

where 50 is a threshold, determined experimentally, beyond which the correlation measure can be trusted.

(2) Neighborhood selection. Two alternatives have been studied.

—Correlation threshold [Shardanand and Maes 1995]. This consists of selecting only those users whose similarity with the active user surpasses a given threshold.

—Max number of neighbors [Resnick et al. 1994]. This consists of selecting the N users that are most similar to the active user, where N is a parameter of the algorithm.

(3) Computation of the prediction

—Weighted by correlation. The contribution of each neighbor is weighted by his/her similarity with the active user [Resnick et al. 1994]. The more a user is similar to the active user, the more accurate his/her rating is supposed to be as a prediction.

—Z-score normalization. Before weighting the user rating according to similarity, it is normalized. Normalization assumes that the ratings of each user belong to different distributions: there are users that only give bad ratings to extremely bad items, others that save the good ratings for selected items, others that only rate good items, etc. Therefore, the mean, as well as the standard deviation, of the users are taken into account to normalize their contribution. The details of this normalization, proposed in Herlocker et al. [2002], are shown in Equation (11).

$$p_{ai} = \overline{v_{a.}} + \sigma_a \frac{\sum_{u \in Neigh_a} \left[\left(\frac{v_{ui} - \overline{v_{u.}}}{\sigma_u} \right) s(a, u) \right]}{\sum_{u \in Neigh_a} s(a, u)}. \tag{11}$$

3.4.2. Item-Based. Item-based algorithms are similar to the user-based but, instead of looking for neighbors among users, they look for similar items. Just like user-based algorithms, different strategies can be used as similarity measure, although Sarwar et al. [2001] conclude that the adjusted cosine similarity, Equation (12), obtains by far the best results. Thus, we have only used this measure in our study.

$$s(i, j) = \frac{\sum_{u \in U} (v_{ui} - \overline{v_{u.}}) \left(v_{uj} - \overline{v_{u.}} \right)}{\sqrt{\sum_{u \in U} (v_{ui} - \overline{v_{u.}})^2 \sum_{u \in U} \left(v_{uj} - \overline{v_{u.}} \right)^2}}. \tag{12}$$

After calculating the similarity between the different items, a subset with the N, best neighbors, is selected. To compute the prediction, we add up the ratings the active user has given to such neighbors weighted by its similarity with the item to predict.

$$p_{aj} = \frac{\sum_i \left(s(j, i) v_{ai} \right)}{\sum_i |s(j, i)|}. \tag{13}$$

One of the advantages of this algorithm over the user-based is that the similarity between items tends to be more static than the similarity between users, so the neighborhood can be computed offline.

3.4.3. Similarity Fusion. The problem with both approximations, based on users and on items, is that they only use part of the information available in the rating matrix, that is, the relation between users or items. Given that the amount of information in the matrix is already sparse in most real systems (the users usually rate a small number of items), it would be desirable to use as much information as possible.

Wang et al. [2006] proposed an alternative method which considers both between-users and between-items relations. This aims to obtain better predictions, especially in sparse matrix contexts, that is, when very few ratings are known.

To compute a prediction, the algorithm combines the item ratings of those users that are similar to the active user (SUR), the ratings of the active user on similar items (SIR), and finally, the ratings of similar items by similar users (SUIR). As observed in Equation (14), the weight of each of the contributions will depend on two parameters,

δ and λ, whose values are determined experimentally.

$$\begin{aligned} p_{ui} &= \sum_{r\in R} P(v_{ui} = r|SUR, SIR, SUIR) \\ &= \left(\sum_{r\in R} P(v_{ui} = r|SUIR)\delta\right) + \left(\sum_{r\in R} P(v_{ui} = r|SUR)\delta(1-\lambda)\right) \\ &\quad + \left(\sum_{r\in R} P(v_{ui} = r|SIR)(1-\delta)(1-\lambda)\right). \end{aligned} \tag{14}$$

3.4.4. Personality Diagnosis. This algorithm [Pennock et al. 2000] is based on a simple probabilistic model that approximates the manner in which users rate the items by means of a normal distribution. The idea is that a user, when rating a specific item at different moments, might give a different rating each time. This difference is motivated by diverse causes: mood, other items rated in the same context, etc. Hence, the profile or personality of the user, corresponding to his/her ratings, should be accompanied with this variation that the algorithm models as simple Gaussian noise. So, $P(v_{ij} = x|v_{ij}^{true} = y) \simeq e^{-(x-y)^2/2\sigma^2}$, where v_{ij}^{true} is the real user rating.

Using this model, the algorithm calculates the probability of the active user having the same personality as the other users. Then these probabilities are used to compute the probability distribution of the rating of a particular item. Finally, the prediction would be the most probable rating.

3.4.5. Regression-Based. This is an item-based algorithm, that is, it obtains the rating prediction based on other similar items. The relation between two items is viewed as an expert, modeled as a linear function:

$$f_{ij}(x) = x\alpha_{ij} + \beta_{ij},$$

where parameters α_{ij} and β_{ij} are estimated with a simple linear regression of the ratings of both items. When α_{ij} is close to $+1$ or -1, the expert is a good predictor of the item rating. The relation between the different items is calculated initially, obtaining $n(n-1)$ experts. To obtain the prediction for an item rating, predictions of each expert are combined. In this work, we have used the mean of all the experts whose regression coefficient R^2 is greater than a certain threshold. Vucetic and Obradovic [2000] presented two alternative methods.

3.4.6. Slope One. The slope one algorithms [Lemire and Maclachlan 2005] are based on predictors of the form $f(x) = x + b$, therefore, simpler than those used in the regression-based algorithm.

In the original slope one algorithm, the constant, b, is defined as the mean difference between each item and the item to predict, computed among the users that have rated both items. The final prediction is calculated as in the equation:

$$p_{uj} = \overline{v_{u.}} + \frac{1}{|R_j|}\sum_{j\in R_j}\sum_{x\in S_{ji}} \frac{v_{xi} - v_{xj}}{|S_{ji}|}, \tag{15}$$

where S_{ji} is the set of users that have rated both items j and i, $S_{ji} = U_j \cap U_i$, and R_j is the set of items rated by the user for which $|S_{ji}| > 0$.

This scheme can be improved if the number of ratings available from each user is taken into account, yielding the weighted slope one algorithm:

$$p_{uj} = \frac{\sum_{i \in I_u - i_j} \left(\sum_{x \in S_{ji}} \frac{v_{xi} - v_{xj}}{|S_{ji}|} + v_{ui} \right) |S_{ji}|}{\sum_{i \in I_u - i_j} |S_{ji}|}. \tag{16}$$

Finally, a third variant, bi-polar slope one, has also been studied. It divides the items into those that the user has rated positively and those that have been rated negatively, taking the user mean as threshold.

3.4.7. LSI/SVD. This algorithm, proposed by Sarwar et al. [2000], is based on reducing the dimensionality of the original rating matrix. The new matrices obtained represent latent attributes in the ratings, allowing to find relations among items and eliminating the problems caused by the sparsity of the matrix or anomalous ratings.

Latent semantic indexing (LSI) is based on SVD, a matrix factorization technique that converts a matrix R into three matrices, $R = U \cdot S \cdot V^T$, where U and V are two orthogonal matrices and S a diagonal matrix of size $r \times r$ (where r is the rank of matrix R), formed by the singular values of the rating matrix. This matrix is reduced by discarding the smallest values, to finally have a matrix S_k, with $k < r$. The reconstructed matrix, $R_k = U_k \cdot S_k \cdot V_k^T$, is the best rank-$k$ approximation of the rating matrix. The prediction is calculated from the reduced dimensionality matrices according to the formula:

$$p_{ui} = \overline{v_{u.}} + U_k \cdot \sqrt{S_k^T}(u) \cdot \sqrt{S_k} \cdot V_k^T(i). \tag{17}$$

3.4.8. Regularized SVD. The regularized SVD model, now really popular among collaborative filtering techniques, was originally proposed in this context by Funk [2006]. In this model, each item is represented as a set of features (aspects) and each user as a set of values indicating his/her preference for the various aspects of the items. The number of features to consider, K, is a model parameter. The rating prediction is composed of the sum of both:

$$p_{ij} = x_i^T y_j, \tag{18}$$

where x_i and y_j are K-dimensional vectors representing, respectively, the affinity of each user and item to each one of the K features. The values of such vectors are estimated in the model construction phase, using a variation of SVD where the unknown ratings are ignored. This simplifies the computation of SVD, and it is much faster than the one used in the LSI/SVD algorithm. It involves a gradient descent technique with regularization to minimize the sum of squared residuals:

$$\begin{aligned} e_{ij} &= v_{ij} - p_{ij} \\ x_{ik}+ &= lrate * (e_{ij} y_{jk} - \lambda x_{ik}) \\ y_{jk}+ &= lrate * (e_{ij} x_{ik} - \lambda y_{jk}). \end{aligned}$$

A single feature is estimated at a time.

We have also evaluated several improvements to this model, introduced by Paterek [2007].

—Improved Regularized SVD (RSVD2). It adds one additional parameter per user, c_i, and one per item, d_j, yielding the model:

$$p_{ij} = c_i + d_j + x_i^T y_j. \tag{19}$$

—NSVD2. This is a method that reduces the number of parameters, by modeling x_i as a function of the items rated by the user. In this case, the model is:

$$p_{ij} = c_i + d_j + \sum_{k=1}^{K} y_{jk} \sum_{j_2 \in I_i} y_{j_2 k}. \tag{20}$$

Moreover, we have evaluated the SVD++ algorithm [Koren 2008], a variation of regularized SVD where the implicit feedback (modeled as the items a user has rated) is taken into account:

$$p_{ij} = b_{ij} + y_j^T \left(x_i + |I_u|^{-\frac{1}{2}} \sum_{j_2 \in I_u} w_{j_2} \right), \tag{21}$$

where w is a vector of additional model parameters that weight the contribution of implicit feedback.

3.4.9. Integrated Neighbor-Based – SVD Model. This algorithm, proposed by Koren [2008], combines the SVD++ algorithm described previously with a neighbor-based approach.[5] The details of the model, also trained with gradient descent techniques, can be consulted in Koren [2008].

3.4.10. Cluster-Based Smoothing. This works like user-based algorithms, with the exception that it first groups the users into clusters, to reach two objectives: to increase the density of the rating matrix, approaching the unknown ratings with the cluster mean, and to increase scalability, looking for neighbors only in the clusters that are closest to the active user. The algorithm, whose details are found in Xue et al. [2005], has 5 phases.

(1) In the first phase, offline, the users are grouped into k clusters, using the K-Means algorithm.
(2) The cluster mean is used to fill the user profile, that is, the items that a user has not rated are approached with the mean of the ratings of the other users in the cluster.
(3) The clusters closest to the active user cluster are selected.
(4) Neighbors are chosen among the users belonging to these clusters.
(5) Finally, the prediction is computed using the weighted mean of the neighbors.

4. TENDENCIES-BASED COLLABORATIVE FILTERING

Most, if not all, collaborative filtering algorithms presented so far are based on the similarities among users or items. Although many different techniques have been used to process the data, most focus on finding more or less hidden relationships. The idea is that, if two users show a similar rating pattern, they will probably coincide in the missing ratings, too. However, to find these relations, most techniques require a great amount of information as it is a rather complex task. Therefore, with sparse datasets, these similarity-based algorithms face serious problems.

As an alternative, we have developed an algorithm that, instead of looking for relations between users or items, looks at the differences between them. Users rate items in different ways, variations that are related to their differences in opinions and tastes. However, this is not the only reason. Users with similar preferences might rate items in a different way: some users are more inclined to give positive ratings, leaving negative

[5] A slightly modified version of this algorithm that also takes into account the time in which each rating is given has won the Netflix Prize thanks to its very good accuracy.

ratings for really bad items; other users, however, save their highest ratings for the best items and tend to give negative ratings.

In addition, these variations are not exclusive of systems with explicit ratings. The time a user has to interact with the system or the money the user can spend to buy our products, for example, can influence the ratings extracted by an implicit system. So, it seems clear that user ratings will depend on several factors and not only on the real quality of the item.

These variations had already been observed previously [Resnick et al. 1994; Herlocker et al. 1999], but their incorporation to algorithms had a secondary role, usually limited to a previous normalization of the ratings to prevent them from negatively affecting the final prediction. In other words, the variations used to be discarded to avoid their influence in the calculation of similarities between users or items. However, our algorithm is based on these variations. In our opinion, they are a more accurate indicator of the quality of a particular item and its utility for the user.

In fact, such variations have also been modeled in previous work. Jin et al. [2003] have reported the distinction between user ratings and the actual user preferences that were decoupled in their probabilistic model by introducing two hidden variables. More recently, G. Potter has taken a psychology approach, based on behavioral economics science, relating these variations to social or emotional factors of the users [Ellenberg 2008]. For example, if a user watches a good film he rates high, and then he watches a better one, the latter may be rated higher than it would be if the user had only watched this second movie.

Our algorithm, however, interprets those variations in a simpler way: the tendencies of users and items. Instead of using complex calculations, tendencies are easy to compute, and they can be accurately calculated using much fewer data than those needed to find relationships. This is a very important feature as it will allow the development of accurate algorithms with very low computation time and memory requirements.

The concept of tendencies refers to whether a user tends to rate items positively or, on the contrary, negatively. It is important to not confuse this tendency with the value of the user mean rating. For example, a user that only rates good items will probably have a high mean. However, if many users also liked such items, each item may actually have a higher mean rating than the rating given by the user. In this case, it is said that the user tends to rate the items negatively, even though his/her mean is high. Therefore, we define the tendency of a user ($\tau_{u.}$) as the average difference between his/her ratings and the item mean.

$$\tau_{u.} = \frac{\sum\limits_{i \in I_u} (v_{ui} - \overline{v_{.i}})}{|I_u|}. \tag{22}$$

Capturing the tendency of an item ($\tau_{.i}$), that is, whether the users consider it an especially good or especially bad item, is also of our interest. The aim is not to see whether the item rating is high (which could be achieved by determining if the item mean surpasses the global mean rating) but rather to see if it stands out from the items rated by a user. It seems logical to think that, if a user gives an item a higher rating, it is because the user considers the item to be good, or at least better than other items. Once again, we are interested in relative ratings, that is, the rating with respect to the user mean and not the absolute item mean. This value is calculated as:

$$\tau_{.i} = \frac{\sum\limits_{u \in U_i} (v_{ui} - \overline{v_{u.}})}{|U_i|}. \tag{23}$$

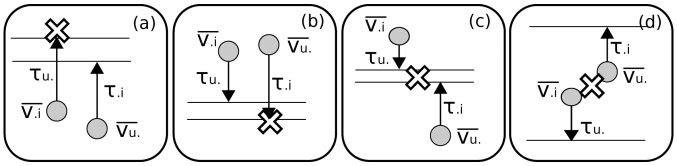

Fig. 1. Possible relations between means (circles) and tendencies (arrows).

Both tendencies take values that vary between $-d$ and d, where d is the difference between the maximum and minimum rating allowed. The algorithm takes into account the user mean and item mean as well as their respective tendencies when computing a prediction. Depending on these values, there can be several cases (shown in Figure 1).

In the first case, Figure 1(a), both the user and the item have a positive tendency, that is, the user tends to give ratings that are above the item mean rating and the item tends to be rated above the user mean. Therefore, it seems a good idea to predict a rating that is above both means. Specifically, we use the formula:

$$p_{ui} = \max(\overline{v_{u.}} + \tau_{.i}, \overline{v_{.i}} + \tau_{u.}), \tag{24}$$

where the reason for using the maximum is to give a better rating to these items whose tendency indicates that they are good.

The second case, Figure 1(b), is just the opposite. Both the user and the item have a negative tendency. In other words, the user usually rates items below their mean and the item tends to be rated below the user mean. In this case, the prediction is calculated as:

$$p_{ui} = \min(\overline{v_{u.}} + \tau_{.i}, \overline{v_{.i}} + \tau_{u.}). \tag{25}$$

The objective of using the minimum is to prevent this item, whose tendency indicates that it is a bad recommendation, from being recommended simply because the user presents a high mean.

The third case, Figure 1(c), occurs when we come across a negative user (his/her tendency is to rate items below their mean), and a good item (its tendency is to be rated above the user mean).[6] If both means corroborate their tendencies (that is, user with low mean and item with high mean), the prediction will be somewhere in the middle between the two, closer to one or another, depending on the value of the different tendencies. The prediction is computed as:

$$p_{ui} = \min(\max(\overline{v_{u.}}, (\overline{v_{.i}} + \tau_{u.})\beta + (\overline{v_{u.}} + \tau_{.i})(1 - \beta)), \overline{v_{.i}}), \tag{26}$$

where β is a parameter that controls the contribution of the item and user mean.

Finally, there could be a situation where the means do not corroborate the tendency (see Figure 1(d). This happens when a user with negative tendency rates an item with low mean (the prediction should be bad) but, at the same time, user mean is high and the tendency of the item is positive (that, on the contrary, would indicate a good evaluation). In this last case, the prediction is computed as:

$$p_{ui} = \overline{v_{.i}}\beta + \overline{v_{u.}}(1 - \beta). \tag{27}$$

As observed, a simple formula is used in the four cases and the calculation will not depend on the number of users or items of the system.

5. JOINING PREDICTION ACCURACY AND CLASSIFICATION ACCURACY METRICS

One of the problems of traditional prediction accuracy metrics, such as MAE, is that they take into account the error committed on all the items. However, in the *find good*

[6] or vice versa, positive user and bad item.

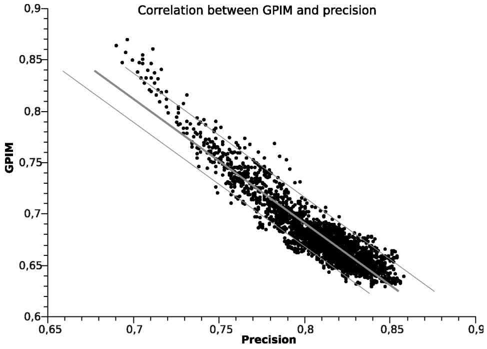

Fig. 2. GPIM and precision metrics show a strong linear correlation.

items task, recommender systems aim to recommend only good items. Thus, metrics like MAE or RMSE, that average the error committed on the whole evaluation set, are not suitable for this task. The reason is that an error in a bad item will only be significant when the system considers it good enough to be included on the recommendation list. In the same way, an error in a good item will only be perceptible by a user if the system excludes it from the list since it considers it a bad item. Thus, giving the same importance to all kinds of errors, as MAE does, is not the best evaluation method for the *find good items* task.

In fact, works focused on this task usually evaluate the results using classification accuracy metrics, such as ROC or precision and recall. However, those metrics only evaluate the quality of the recommendation list and not the error committed on the prediction. In fact, as we have discussed in Section 3.3, one of the limitations of classification accuracy metrics is that they only evaluate whether the recommended items are relevant without taking into account how relevant they actually are. Moreover, rank accuracy metrics, usually applied to avoid this limitation, are too sensitive, as we usually just want to recommend good items and not necessarily the best ones. Actually, prediction accuracy metrics are good candidates to evaluate the *find good items* task, as they evaluate how good the recommendations are but without giving excessive importance to some mistakes. However, in this task, we do not care about the whole evaluation set as prediction accuracy metrics do but only about the items that are part of the recommendation list. In other words, users expect the recommended items to be actually good and, similarly, the items that are good to be recommended.

From this idea, we propose two new metrics, Good Items MAE (GIM) and Good Predicted Items MAE (GPIM), that compute, respectively, the MAE in the prediction of good items and in those items the system predicts as good.[7] So, instead of taking into account all the items in the evaluation subset, they focus only on relevant items. Thus, the first and main advantage of these metrics is that they evaluate the algorithm only in predictions that are relevant to the user, that is, items that either will appear or should appear on the recommendation list. In other words, GPIM and GIM are the equivalent of precision and recall, respectively, but from a prediction accuracy perspective. They combine the best of both worlds, allowing an evaluation based on prediction accuracy for the *find good items* task. In fact, a study of the correlation between GPIM and precision shows a strong linear correlation, as seen in Figure 2, with an R^2 of 86.47% and an ANOVA p-value of 0.000.[8] The correlation is negative because, obviously, a

[7] Of course, we should identify which items are good and which are bad. In this work we have used a rating threshold, considering good items those rated with 4 or 5 points.

[8] This study has involved results on more than 2,500 tests with different algorithms and parameters.

low GPIM value indicates high precision. This strong correlation is an interesting feature as prediction accuracy is usually easier to evaluate than precision using offline datasets.

Moreover, these metrics have an additional advantage that may pass without notice at first sight. If we look at the ratings stored in many recommender systems, we observe that most of these are good. Therefore, the algorithm has fewer data to predict the rating of bad items than the rating of good items, worsening the overall accuracy even when these errors, as we have seen, are not so important. Our metrics avoid these problems by ignoring the errors committed on bad items.

Finally, they have a third and important advantage. They help capture the bias of the algorithm to rate items too optimistically (good results in GIM, bad in GPIM) or pessimistically (good in GPIM, bad in GIM). These biases, that should be avoided, are hard to identify using traditional metrics.

Of course, when the algorithm is used in the *annotation in context* task, where the user can request the evaluation of any item, both metrics are insufficient. For this task, traditional metrics such as MAE or RMSE are better candidates, although GIM and GPIM could also be useful.

6. EXPERIMENTS

6.1. Datasets

We have used two datasets in our evaluation, MovieLens [Herlocker et al. 1999] and Netflix [Bennett and Lanning 2007]. These are the most popular datasets used by researchers and developers in the field of collaborative filtering, along with EachMovie (that, unfortunately, is no longer available). Both datasets can be downloaded from the Internet and have been used in many works, so they are especially interesting for our purposes.

The MovieLens dataset contains real data corresponding to movie ratings captured on the website of the MovieLens movie recommender (http://movielens.umn.edu) during a 7-month period (19-09-1997 to 22-04-1998). From these data, users with less than 20 ratings have been removed, giving a total of 100,000 ratings from 943 users on 1,682 movies. Therefore, the ratio of the number of items to the number of users is 1.78, and the density is 6%. This dataset has been widely used in collaborative filtering research in the last decade.

The Netflix dataset contains over 100 million ratings from 480,189 users on 17,770 movies collected between October 1998 and December 2005. It has become extremely popular in recent years, after having been released to be used by researchers and competitors in the Netflix Prize. In our experiments, however, we have not used the whole dataset. The reason is that many of the algorithms we have evaluated in this work do not scale well enough to such a big amount of data. So in order to evaluate those algorithms, we have randomly selected 30% of the users and items, obtaining a dataset with 9,132,089 ratings by 144,190 users on 5,354 items. We have also compared the results of 100 tests with different algorithms and configurations using both the whole and reduced datasets to check how correlated they are. We have obtained a strong linear correlation, with an R^2 over 99.91% and an ANOVA p-value of 0.000, which shows that we can trust the results of the reduced dataset to estimate results of the whole dataset. Finally, we should note that, differently from MovieLens, the reduced Netflix dataset contains less items than users, 0.04 items per user, and a density of 1.2%.

We can also see that the rating distribution in both datasets is quite similar. First, as seen in Figure 3, most users just rate a low number of items. Moreover, in both cases, the ratings are discrete and go from 1 (low rating) to 5 (high rating). As observed in

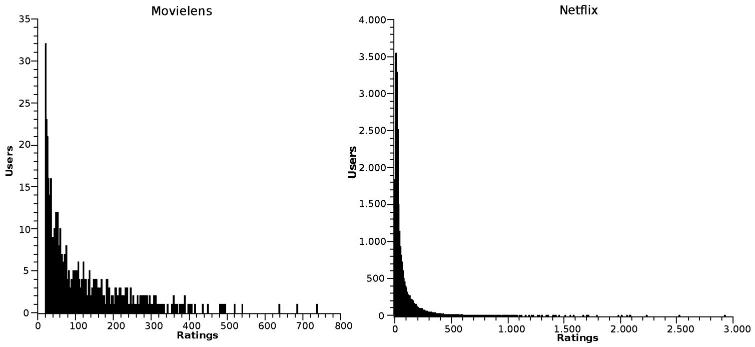

Fig. 3. Ratings per user for MovieLens and Netflix dataset.

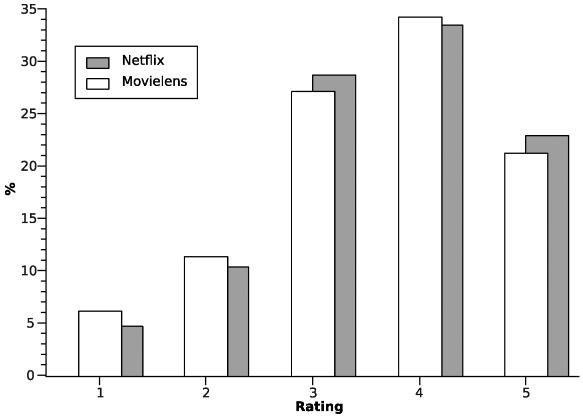

Fig. 4. Percentage of each rating, for both datasets.

the distribution histogram of ratings in Figure 4, users tend to rate movies they liked. This is the reason why ratings 3, 4, and 5 appear more frequently than 1 and 2. Such similarities between both datasets are not surprising given that both pertain to the same domain, the recommendation of movies. We should not forget that algorithms are strongly dependent on the characteristics of the dataset, so the results obtained in this study may not coincide with those obtained with data from other domains. Parameters such as the rating matrix sparsity, the ratio of the number of users to the number of items, variability in the ratings, or their scale can notably influence the accuracy of an algorithm. Although, as mentioned, we have chosen these two datasets because many works have used them, undoubtedly, the application of the evaluation methodology proposed in this article with other datasets can be an especially interesting work.

Finally, we should note that both datasets contain additional information other than user ratings. For example, MovieLens contains demographic data about the users (e.g., sex, age, profession, etc.) and Netflix stores time information about each rating. However, this extra information has not been used in the study. We have focused on evaluating pure collaborative filtering algorithms where only the ratings are considered.

6.2. Methodology

The experiments were performed by dividing the dataset into two groups, a training subset and an evaluation subset. The first set corresponds to data the algorithm already knows, that is, the data used to train the algorithm. With such information, the algorithm computes the recommendation that will be later compared with the original data present in the evaluation subset.

We have followed two different approaches for selecting the training subset. In the first, it is constructed from a percentage of the available ratings, randomly chosen. For our tests we used all the following percentages: 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, and 90%. The evaluation subset was composed by randomly selecting 10% of the dataset.[9] Obviously, ratings that appear in the evaluation subset were never included in the training subset. With high percentages of ratings in the training set, we can evaluate the behavior of the algorithm under relatively high density conditions. In contrast, a small percentage allows to evaluate the algorithm under sparsity conditions, common in the initial phases, or in domains with a large number of users and/or items.

In fact, given that many systems operate under these conditions, we have used a second method to evaluate algorithms under sparsity conditions. It consists in selecting as training set, a fixed number, generally low, of ratings from each user. We tried selecting 1, 2, 3, 4, 5, 7, 10, 12, 15, and 20 ratings. This method, first used by Breese et al. [1998], was named Given-N.

We have evaluated the two most common tasks of recommender systems, the prediction of the item rating (*annotation in context*), and the recommendation of a certain number of items (we tried with 5, 10, 15, 25, 50, 100, and 150 items). In this second case, we confronted a problem inherent to the use of offline datasets, real ratings are not available for most items. In these cases, small errors related to the items rated, like including an item with low rating or leaving out one with a high score, greatly affect the final result [Herlocker et al. 2004]. To minimize this problem, in the evaluation, we force the algorithm to recommend items that have a rating in the evaluation set. Therefore, the final list will consist of N items that were already rated. Although this helps minimize this problem, only the evaluation with real users can satisfactorily solve this question.

In the task of *annotation in context*, this problem is smaller. As long as we have a significant amount of evaluation data, the results can be extrapolated to all the items. However, the biases present in the dataset can influence the results. For example, in the datasets we have used, most of the ratings correspond to good items. Thus, an algorithm that correctly predicts this kind of items will obtain better results than one that performs better on low rated items, simply because errors committed on bad items will have less influence on the final mean. Although in this case this is not a problem, as we are interested in algorithms that correctly predict high rating items, other biases in the datasets could be.

In the evaluation, the most popular metrics in the literature, presented in Section 3.3, were used, coverage; MAE and RMSE; Precision; Recall; ROC and Half-life Utility, along with the new metrics proposed in this work, presented in Section 5. Each test was repeated 5 times, so the results presented in Section 7 refer to their means.

6.3. Algorithms

A set of algorithms representative of the different techniques found in the literature was evaluated, all described in Section 3.4.

- Memory-based
  - User-based (UB)
  - Item-based (IB)
  - Similarity fusion (SF)

[9] We have empirically checked that variations of the evaluation subset have not a great impact on the results.

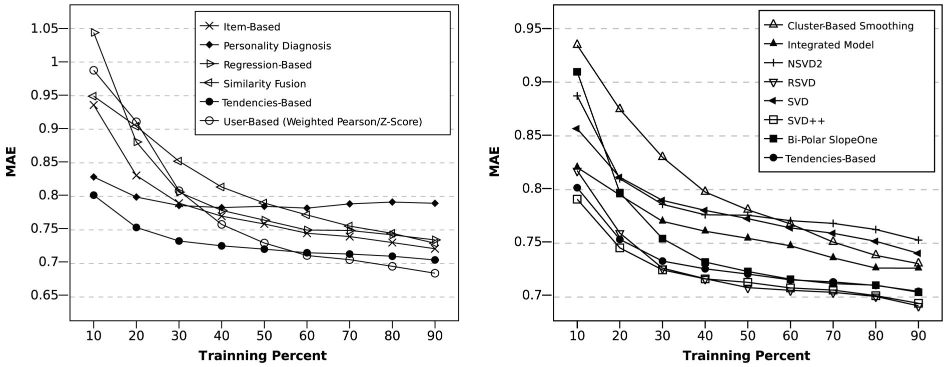

Fig. 5. Evolution of accuracy (MAE) for MovieLens dataset according to matrix density. Note that results of tendencies-based algorithm are shown in both charts.

—Model-based
  —Regression-based (RB)
  —Slope one (SO)
  —LSI/SVD (SVD)
  —Regularized SVD (variants RSVD, RSVD2, NSVD2 and SVD++)
  —Integrated neighbor-based – SVD model (IM)
  —Cluster-based smoothing (CBS)
—Mixed
  —Personality diagnosis (PD)

All have been compared with the tendencies-based (TB) algorithm presented in Section 4.

7. RESULTS

7.1. Accuracy and Lack of Information

First, we studied the evolution of the accuracy and coverage of the different algorithms when the percentage of data used as training set was changed. The results for MovieLens dataset are shown in Figures 5, 6, and 7, while those for Netflix in Figure 8. As observed, all the algorithms improve as this percentage increases. This behavior is just as expected; by increasing the percentage of data in the training set, we are increasing the density of the rating matrix and, therefore, the algorithm has more information to compute the prediction.

Similarly, as the density of the information increases, a slight decrease in the differences among algorithms is observed. Most of them present similar results under relatively high density conditions, while their differences are accentuated as density diminishes. In fact, for MovieLens dataset and with a training set of 80%, there is no statistical significance[10] in the MAE differences among the six best algorithms (UB, RSVD2, SVD++, RSVD, SO and TB), while with 10%, only the three best algorithms (RSVD2, SVD++, TB) present equivalent results. The same conclusion can be reached from results using the Netflix dataset. With 80% training set, no statistically significant differences exist among the best algorithms (RSVD, RSVD2, SVD++, SO and TB), while with 10%, SVD++ presents the best results, followed by RSVD2, NSDV2, RSVD and TB.

[10] At 95% confidence level, using Bonferroni multiple contrast.

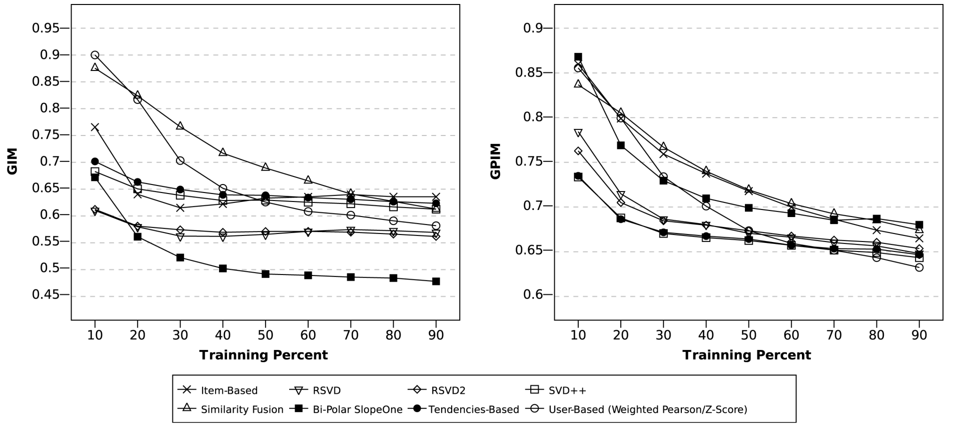

Fig. 6. Evolution of GIM and GPIM according to matrix density, for MovieLens dataset.

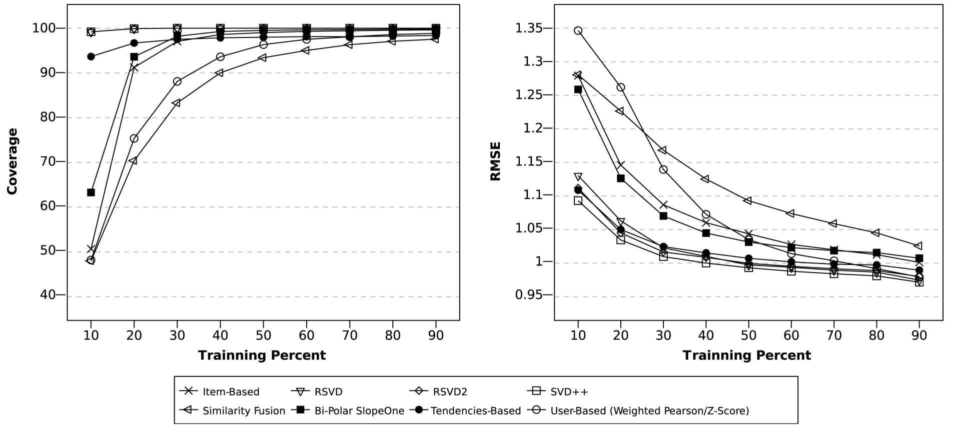

Fig. 7. Evolution of RMSE and coverage according to matrix density, for MovieLens dataset.

In fact, because different techniques present similar results as the density increases, some authors have proposed the existence of a limit in the accuracy the algorithms could reach. This limit would be related to the natural variability of the users; given that a person, asked on different occasions for his/her opinion about an item can give a different rating each time, the prediction of an algorithm is always subjected to this error.

Although we do not deny the existence of this limit, our results show, in contrast, that the greatest limitation an algorithm is subjected to is the lack of information. In other words, the accuracy of a prediction is limited by the amount of information that can be obtained from the rating matrix. And this is further limited by the information that is actually present in the matrix, that is, by its density. In fact, as mentioned previously, the results improve as the amount of information increases (greater proportion of ratings in the training set).

Because the only ratings available are those in the dataset, we cannot increase the amount of data present in the matrix indefinitely and, therefore, it will be difficult to determine whether the accuracy errors of an algorithm are due to the lack of information or to the natural variation of the individuals.

However, what we can report is that the different techniques do not differ as much in their accuracy as in their evolution with matrix density. In other words, they differ

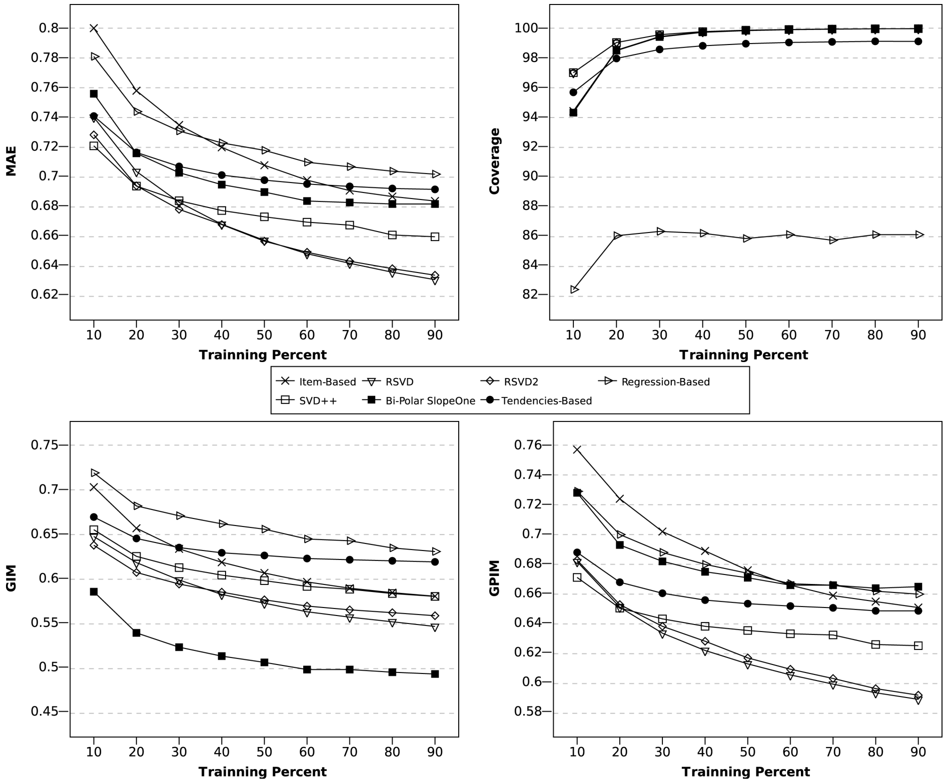

Fig. 8. Evolution of prediction coverage and accuracy according to matrix density, for Netflix dataset.

in their ability to interpret the available information, extract useful knowledge, and obtain predictions with sparse data. As observed, the results under high density conditions are pretty good in all the algorithms, although this changes when the density decreases.

The tests under sparsity environments are good indicators of the ability of algorithms to extract more information from the rating matrix. As stated previously, we used the given-n strategy to evaluate the algorithms under such conditions. Results for MovieLens and Netflix datasets are presented in Figures 9 and 10, respectively. For the sake of clarity, we only show the results for the best algorithms.

Comparing Figures 5, 8, 9, and 10, we see how the evolution of accuracy as the matrix density increases is different for each algorithm. This evolution characterizes a technique better than any accuracy measure under specific conditions. So, more than their accuracy, what differentiates the algorithms is their ability to extract information. For example, as discussed in the next section, the evolution of accuracy in memory-based approaches is different than in model-based techniques.

7.2. Model-Based vs. Memory-Based Algorithms

In memory-based algorithms, the results improve rapidly as the density increases.[11] These algorithms present good results under relatively high density conditions, but their accuracy decreases considerably under sparsity conditions given that they only

[11] Or from another point of view, they quickly worsen as the matrix density decreases.

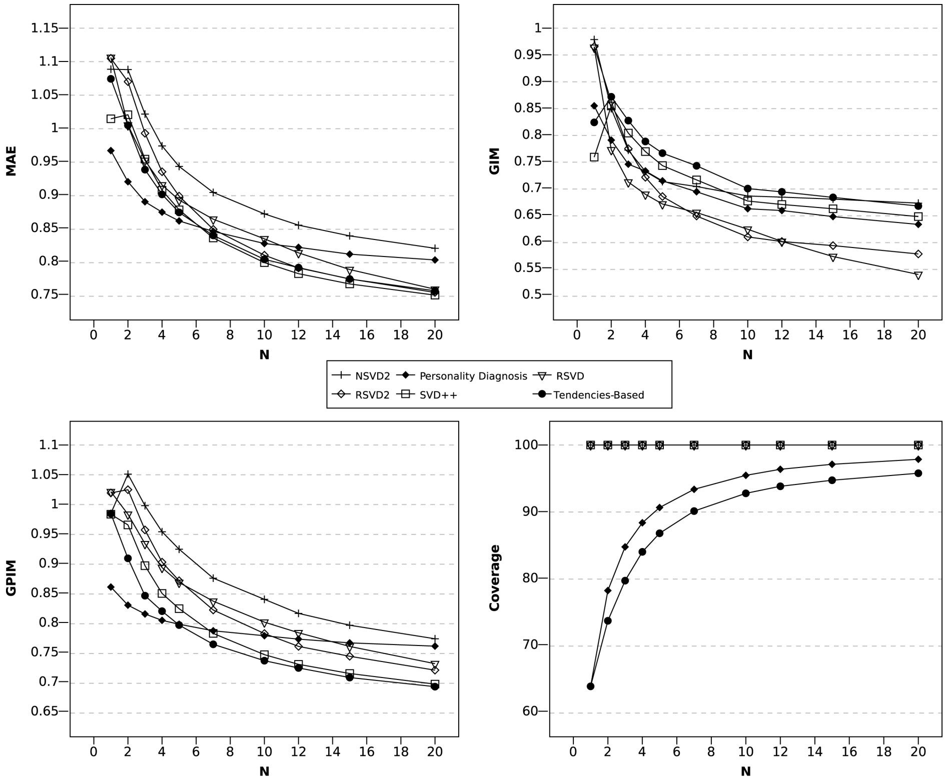

Fig. 9. Accuracy and coverage of the different algorithms under sparsity conditions, by given-n strategy, for the MovieLens dataset.

use a small part of the available information, the relation between users or items, depending on the case. As the density of the matrix diminishes, the difficulty in calculating these relations increases and, thus, the results worsen notably. In fact, observing the important decrease in coverage of this type of algorithms, it is not difficult to relate the bad results in accuracy with the lack of information. The similarity fusion algorithm, using both the relations between users and between items, makes better use of the available information, so it obtains better results for accuracy and coverage under sparse situations and, moreover, it maintains good accuracy under more dense contexts.

On the other hand, model-based algorithms are less sensitive to density changes, presenting a much smoother decline. The explanation for this behavior is found in the model construction phase when a large amount of information is extracted from the rating matrix. The models tend to capture information that is hidden, difficult to extract using the traditional memory-based methods. This is why they work relatively well in sparse contexts. However, the actual construction of the model discards part of the information, making these algorithms less accurate than memory-based under high density conditions. The fact of discarding information can be performed consciously, with the objective of having a scalable algorithm that is computationally efficient. It can also be due to the fact that certain aspects of the information cannot be explained by the model. In fact, as one might expect, the accuracy of these algorithms depends on how well the model fits the real data. Techniques that work well in one dataset may not be adequate in another context.

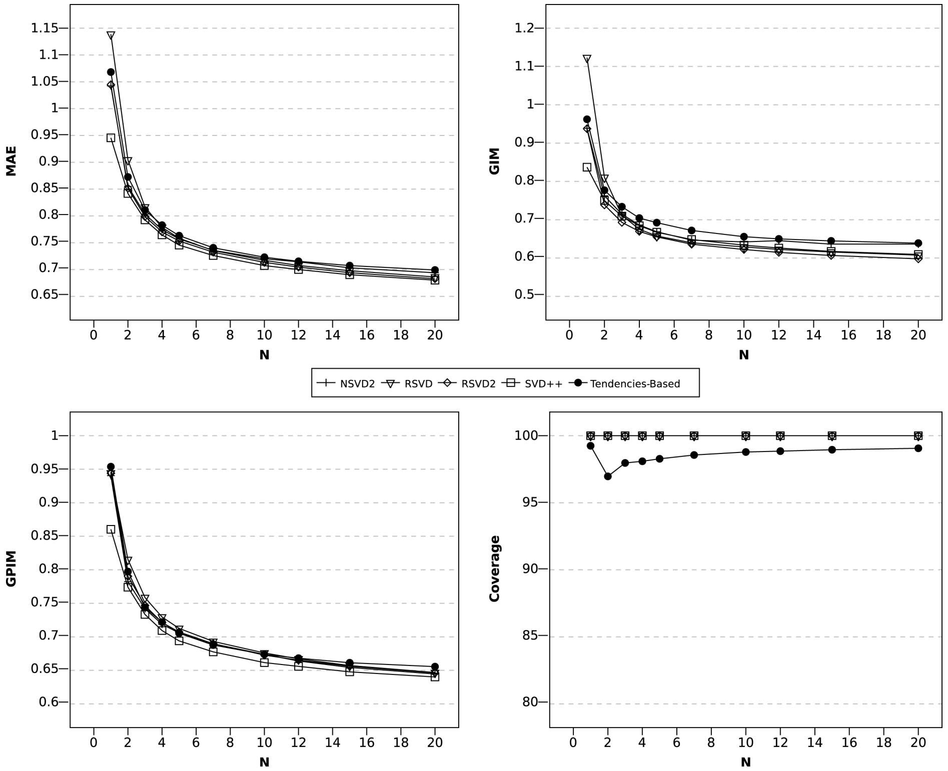

Fig. 10. Accuracy and coverage of the algorithms under sparsity conditions, by given-n strategy, for the Netflix dataset.

Some of the model-based strategies showing the best results in our experiments have been all SVD-based algorithms. In fact, regularized SVD techniques stand out as the best algorithms in many cases, with good behavior in both MovieLens and Netflix. Actually, only the tendencies-based algorithm can compete with them in all cases, although regarding accuracy, some memory-based techniques achieve better results with high data density as expected.

Furthermore, the results in the classification metrics show that they present precise recommendations, as observed in Figure 11. All the algorithms achieve more accurate results with the Netflix dataset, as expected, given that dataset contains more rating data. Overall, the algorithms that present the best results with these metrics are SVD techniques, tendencies-based and slope one (although its precision is not outstanding). With enough training percent, some neighbor-based approaches, such as regression-based, also obtain good results.

In contrast, clustering has been one of the worst strategies in our experiments, the cluster-based smoothing algorithm showing mediocre results. Contrary to what might seem intuitive, the classification of users into clusters does not add any worth to the simple user-based approach. In fact, the results show that the overall mean is a better prediction than the cluster mean with the closest users. Moreover, as seen in Figure 12, as we increase the number of clusters, both accuracy and coverage decrease.

The grouping of users into clusters does not seem to be a good strategy. Although this result may seem surprising, if we study the nature of the problem we can see that it is not. Clustering is based on the idea that classes of users exist, that is, we can

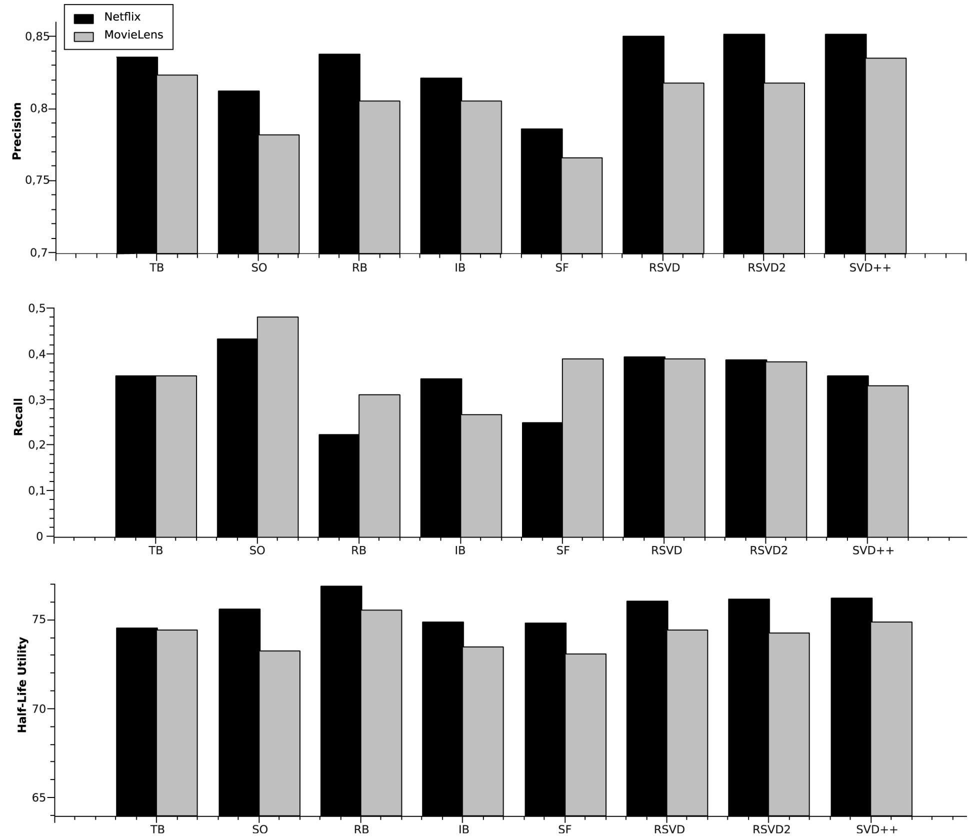

Fig. 11. Classification metrics results for both MovieLens and Netflix datasets, with a training set of 50%.

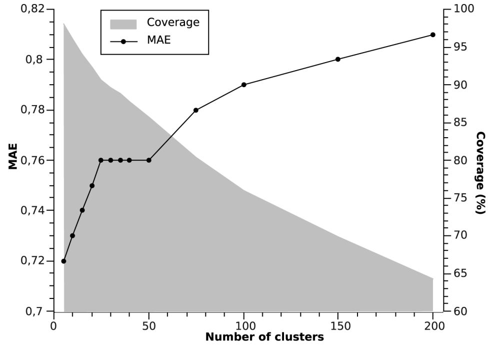

Fig. 12. Evolution of accuracy and coverage according to number of clusters.

group the users according to their tastes or interests, which may seem intuitive. In the context studied in this work, one can think that a cluster is a set of users who like the same genre, actor, director, etc. However, in practice it is difficult for these groups to exist. The fact that we like action movies does not imply that we do not like comedies, or that if we like a director, this does not mean that we like all his movies. In the end, the tastes or preferences of each user are too diverse, making it difficult to place them

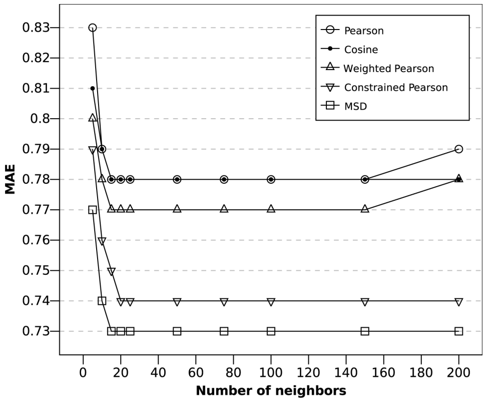

Fig. 13. Accuracy according to the number of neighbors with a training subset of 80% for MovieLens dataset. Observe that with few neighbors the error increases considerably.

into a particular category. Therefore, it is not surprising for methods that base their recommendations on the existence of these categories to not obtain good results.[12]

7.3. Study of User-Based Techniques

User-based algorithms and cluster-based methods have a similar problem. As commented previously, these algorithms have a step in which the users from whom the prediction will be calculated are selected. The idea is to take into account users with tastes and interests that are most similar to those of the active user, his/her neighbors.

Traditionally two strategies have been used, correlation threshold and maximum number of neighbors. Herlocker et al. [2002] reported that the best strategy is the latter.[13] It involves selecting up to N users, according to their similarity with the active user. It seems logical to think that the lower the value of N is, the better the result obtained, since the prediction is based on users more similar to the active user. However, as observed in Figure 13, when fewer than 20 neighbors are used, accuracy worsens considerably. The results based on 50 or 100 neighbors are much better than those obtained based on the 5 or 10 that best resemble the active user, independently of the measure of similarity selected. This behavior is clearly related to the difficulty that two users have the same tastes. Users with coincident opinions in a good number of items and, therefore, considered similar, can totally disagree in others.

This problem is also observed in the second strategy, where users, whose similarity with the active user surpasses a certain threshold, are selected as neighbors. As seen in Figure 14, as we increase the threshold, and, therefore, the prediction is based on the most similar users, accuracy worsens.[14] Furthermore, the difficulty in finding closely correlated users is the reason for the pronounced decrease in coverage as the correlation threshold increases. This result reinforces the idea that it is hard to find users with similar tastes.

[12] Overlapping clustering-based algorithms (a user could belong to several clusters) or co-clustering (grouping both users and items) [George and Merugu 2005; Shafiei and Milios 2006] offer an interpretation that is closer to reality. Extending the study performed in this work to these algorithms could be interesting.

[13] The same conclusion is also reached from our experiments.

[14] Note that the apparent improvement in accuracy in certain cases is actually produced with practically zero coverage and corresponds to easy-to-predict items. Herlocker et al. [2002] showed that, in fact, there is not such improvement, since if we only measure the error in those items, accuracy diminishes when the correlation threshold is increased.

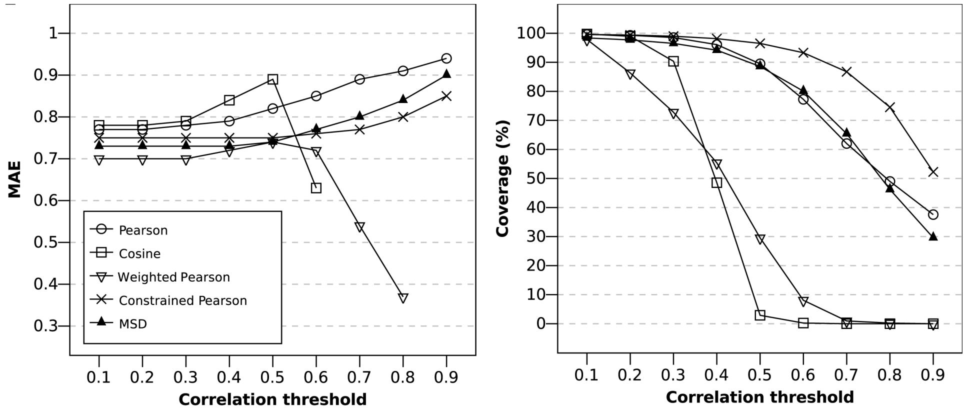

Fig. 14. Accuracy and coverage according to correlation threshold with a training set of 80%.

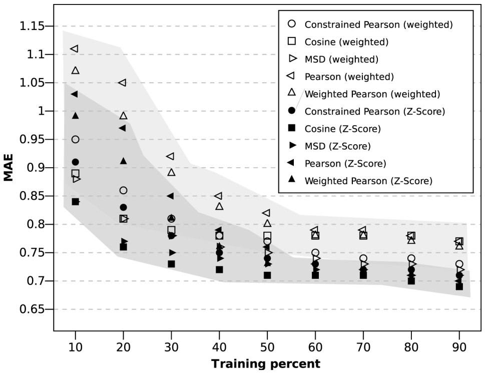

Fig. 15. Effect of normalization (z-score) on the accuracy of user-based algorithms.

It is also interesting to observe the effect of the normalization (z-score) of the ratings before computing the prediction. In all cases, normalization substantially improved the results, as observed in Figure 15. In fact, especially under high density conditions, algorithms that use normalization are clearly better than those that do not, independently of the similarity measure used [Herlocker et al. 2002]. With enough data density, the worst algorithm with normalization presents better results than the best algorithm without normalization. This observation is especially interesting: in an algorithm such as user-based, strongly centered on capturing similarities between users, the factor with the greatest impact on accuracy is not the measure of similarity used, but rather the use or not of a normalization process based on the differences between users.

This fact confirms the importance of the tendencies of the users when rating items. The results of our experiments reveal that the differences between users and/or items are more useful than their similarities to predict the utility of a particular item. Two factors stand out as the reason for this behavior. First, as stated before, it is difficult to find users with the same tastes or interests. Second, the sparsity conditions recommender systems normally deal with make the calculation of relations between users (already complex by itself) difficult and, what is worse, many times it can lead to erroneous interpretations.[15]

7.4. Tendencies-Based Algorithm: A Simple but Accurate Approach

In fact, as seen in Figures 5, 8, 9, 10, and 11, the tendencies-based algorithm is, together with some SVD variants, the one that obtains the best results.

Under high density conditions, it is as accurate as the best algorithms, hardly being surpassed in MAE by some memory-based or SVD-based approaches. In fact, in most cases there are no statistically significant differences in accuracy between the tendencies-based algorithm and those that obtain slightly better results. Moreover, the results of the metrics GPIM and GIM show that its errors are less visible to the user than those committed by other algorithms, as it presents high accuracy for the relevant items. For both MovieLens and Netflix dataset, it achieves the best results in GPIM, together with SVD++, IM, and some SO variants. This indicates it has a great precision, that is, the items it recommends are really good. In GIM it also obtains good results for both datasets (statistically equivalent to SVD++, too), although for MovieLens, some algorithms are better (such as bi-polar SO, RSVD and RSVD2). For Netflix, both RSVD and RSVD2 achieve statistically equivalent results. Furthermore, as it obtains good results in both GPIM and GIM, it is not biased towards excessively optimistic or pessimistic predictions, unlike memory-based algorithms, for example. Only RSVD variants are unbiased, too.

These results are also confirmed in the classification accuracy metrics, with similar results for both datasets. In precision, it is statistically equivalent to RSVD and RSVD2 techniques, and only SVD++ obtains slightly better results. In ROC metric, it is the best together with SVD and SVD++ (there are no statistically significant differences among them). The results for half-life utility are also good, only slightly worst than SVD, regression-based, slope one or integrated model (the last two are statistically equivalent). On the other side, results in recall are modest when compared with PD or bi-polar SO; although it is equivalent to algorithms with similar precision such as those based on SVD.

Under sparsity conditions, it also presents the best results overall, together with some SVD variants. With less than 50% of the data as training set, it clearly outperforms the accuracy and precision of all memory-based approaches. With 10%, high sparsity conditions, it improves the best user-based algorithm by 20%, and by 10% algorithms such as slope one or item-based. Only RSVD, RSVD2 and SVD++ present equivalent results under sparse conditions. Furthermore, along with personality diagnosis and SVD-based strategies, it is the only one whose coverage practically does not diminish when decreasing matrix density. With a training set containing only 5 ratings per user, it is still able to make predictions for more than 90% of items, while the user-based barely covers 12%, slope one less than 9% and item-based does not even reach 6%.

7.5. Algorithm Efficiency

It is with computational efficiency where the tendencies-based algorithm obtains the best results. Although previous works in the field have not paid too much attention to this issue, it is actually very important, as in production environments, most recommender systems should compute predictions in real time. In Table I the comparison of the computational complexity of the different algorithms is shown.

[15] A paradigmatic case is the Pearson correlation coefficient, two users that have evaluated only one item in common are always considered perfectly correlated.

Table I. Computational Efficiency of the Different Algorithms Studied

Algorithm	Training	Prediction
User-based	-	$O(mn)$
Item-based	$O(mn^2)$	$O(n)$
Similarity fusion	$O(n^2m + m^2n)$	$O(mn)$
Personality diagnosis	$O(m^2n)$	$O(n)$
Regression-based	$O(mn^2)$	$O(n)$
Slope one	$O(mn^2)$	$O(n)$
LSI/SVD	$O((m+n)^3)$	$O(1)$
RSVD	$O(mnk)$	$O(1)$
RSVD2	$O(mnk)$	$O(1)$
NSVD2	$O(mnk)$	$O(1)$
SVD++	$O(mn^2k)$	$O(1)$
Integrated model	$O(mn^2k)$	$O(1)$
Cluster-based smoothing	$O(mn\alpha + m^2n)$	$O(mn)$
Tendencies-based	$O(mn)$	$O(1)$

m is the number of users and n the number of items.

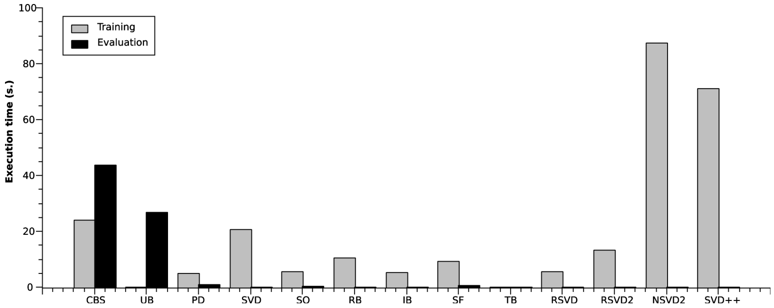

Fig. 16. Training and evaluation times for different algorithms for the MovieLens dataset.

Complexity has been separated into two parts, training and prediction, corresponding to the construction of the model using the training data and making a single prediction, respectively. Training only needs to be performed once,[16] while generally, a large number of predictions will be made. Thus, a low complexity when making a prediction can compensate an elevated complexity in training.

In general, model-based algorithms are more efficient when computing a prediction, although the construction of the model has a considerable complexity. Our tendencies-based algorithm stands out by far as the most efficient, with a complexity of $O(mn)$ in training and of $O(1)$ in the prediction.

These theoretical results are confirmed by studying the execution times. Figure 16 shows both the training and evaluation times of several algorithms for the MovieLens dataset, with a training set of 80%.[17] Evaluation time refers to the time invested in computing the rating prediction for all items in the evaluation set (around 10,000 predictions). As expected, most model-based algorithms have a low prediction time, although the model construction is rather complex in some cases. Techniques such as SVD++ that have obtained good results with accuracy metrics present a high model construction time. On the other hand, user-based techniques have no training phase, but they need more time to compute a prediction. Among the algorithms studied, our

[16] Actually, the algorithm must be retrained often, to take into account the new items, users and/or ratings.

[17] Time was measured on a PC with Quad Core Xeon processor at 2.66 GHz, 2 × 6 MB cache and 1333 FSB, and 8 GB RAM. Algorithms were implemented in Java. Note that different implementations may achieve different execution times.

tendencies-based proposal shows, by far, the best results. Its training time, 13.2ms. in 80% of the MovieLens dataset, is several orders of magnitude better than algorithms with similar accuracy, such as RSVD (5,433ms.), RSVD2 (13,198ms.) or SVD++ (71,025ms.). At the same time, its prediction time (12ms.) is similar to that of other model-based approaches and much better than that of algorithms such as user-based. For Netflix (80% training set), the same great results are obtained: less than 4s. of training time, much better than RSVD (640s.), RSVD2 (860s.) or SVD++ (9,700s.).

8. CONCLUSIONS

In this article, we have performed an ambitious comparison of different collaborative filtering algorithms to observe the behavior of the algorithms under diverse situations, and not only under the most favorable conditions. Similarly, the use of a clearly defined methodology will allow the comparison of other techniques in the future, resolving one of the biggest problems in the evaluation of collaborative filtering algorithms.

We have also highlighted the characteristics of the methodology and metrics used, pointing out the limitations of the offline evaluation to determine the quality of the recommendations, that is, their utility for the user. As a contribution to the evaluation of collaborative filtering systems, we have proposed two new metrics, GPIM and GIM, that focus on measuring the quality of a recommendation list using prediction accuracy techniques. They simplify the evaluation using offline datasets, and at the same time, they detect undesirable biases in the predictions.

Furthermore, we have proposed a novel strategy of collaborative filtering, based on the tendencies or differences between users and items, instead of on their similarities. Its good results, along with its high computational efficiency, make it a great alternative, especially in contexts with large number of users and items and few available ratings. Our approach is as accurate as most modern methods, and its computational efficiency is much better.

The results demonstrate the great influence of matrix density on the accuracy of the algorithms and how this influence depends on the type of algorithm. We have seen how memory-based algorithms offer good behavior with relatively dense matrices, worsening significantly in the absence of information. We have also observed how model-based algorithms, although less accurate than memory-based techniques under optimum conditions, behave better with sparse data. The different experiments performed let us relate this behavior with the ability of the different algorithms to interpret the available data and extract useful information. Most algorithms behave similarly for both MovieLens and Netflix datasets, which is not surprising since both contain data from the same domain, movie recommenders.

We have found that most of the difficulties algorithms have in extracting information are related to the limitations of the techniques based on capturing similarities between users and/or items. However, most current algorithms still use these kinds of techniques. Two possible explanations for these limitations are the great diversity in opinions and tastes and the difficulty in finding such similarities under sparsity conditions.

It would be interesting for future experiments to include new algorithms and other datasets from different domains. The application of the tendencies-based algorithm to contexts such as information retrieval on the Web may also be of special interest. This technique seems appropriate for the large amounts of data found in such domain. Another possibility is the study of other alternatives to the traditional techniques based on the similarities between users or items. The good results obtained by the tendencies-based algorithm in our experiments are a good indication of what direction to take in future works.

Finally, in a recent paper, Marlin and Zemel [2009] have considered the nonrandom nature of missing ratings, achieving significant improvements in prediction accuracy by taking this fact into account. It would be really interesting to study the impact of such fact in the tendencies-based algorithm and how this technique could be improved to address it.

REFERENCES

AGGARWAL, C. C., WOLF, J. L., WU, K.-L., AND YU, P. S. 1999. 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 (KDD'99)*. ACM, New York, NY, 201–212.

BALABANOVIĆ, M. AND SHOHAM, Y. 1997. Fab: content-based, collaborative recommendation. *Comm. ACM 40,* 3, 66–72.

BASILICO, J. AND HOFMANN, T. 2004. Unifying collaborative and content-based filtering. In *Proceedings of the 21st International Conference on Machine Learning (ICML'04)*. ACM, New York, NY, 9.

BENNETT, J. AND LANNING, S. 2007. The netflix prixe. In *Proceedings of KDD Cup and Workshop (KDDCup'07)*. ACM, 4.

BILLSUS, D. AND PAZZANI, M. J. 1998. Learning collaborative information filters. In *Proceedings of the 15th International Conference on Machine Learning*. Morgan Kaufmann, San Francisco, CA, 46–54.

BREESE, J. S., HECKERMAN, D., AND KADIE, C. 1998. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the 14th Annual Conference on Uncertainty in Artificial Intelligence*. 43–52.

CALDERÓN-BENAVIDES, M. L., GONZÁLEZ-CARO, C. N., DE J. PÉREZ-ALCÁZAR, J., GARCÍA-DÍAZ, J. C., AND DELGADO, J. 2004. A comparison of several predictive algorithms for collaborative filtering on multi-valued ratings. In *Proceedings of the ACM Symposium on Applied Computing (SAC'04)*. ACM, New York, NY, 1033–1039.

CANNY, J. 2002. Collaborative filtering with privacy via factor analysis. In *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'02)*. ACM, New York, NY, 238–245.

CHIRITA, P.-A., NEJDL, W., AND ZAMFIR, C. 2005. Preventing shilling attacks in online recommender systems. In *Proceedings of the 7th Annual ACM International Workshop on Web Information and Data Management (WIDM'05)*. ACM, New York, NY, 67–74.

ELLENBERG, J. 2008. This psychologist might outsmart the math brains competing for the Netflix prize. *Wired Maga. 16,* 3.

FISHER, D., HILDRUM, K., HONG, J., NEWMAN, M., THOMAS, M., AND VUDUC, R. 2000. Swami (poster session): A framework for collaborative filtering algorithm development and evaluation. In *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'00)*. ACM, New York, NY, 366–368.

FOLTZ, P. W. AND DUMAIS, S. T. 1992. Personalized information delivery: an analysis of information filtering methods. *Comm. ACM 35,* 12, 51–60.

FUNK, S. 2006. Netflix update: Try this at home. http://sifter.org/ simon/journal/20061211.html.

GEORGE, T. AND MERUGU, S. 2005. A scalable collaborative filtering framework based on co-clustering. In *Proceedings of the 5th IEEE International Conference on Data Mining (ICDM'05)*. IEEE Computer Society, Los Alamitos, CA, 625–628.

GOLDBERG, D., NICHOLS, D., OKI, B. M., AND TERRY, D. 1992. Using collaborative filtering to weave an information tapestry. *Comm. ACM 35,* 12, 61–70.

GOLDBERG, K., ROEDER, T., GUPTA, D., AND PERKINS, C. 2001. Eigentaste: A constant time collaborative filtering algorithm. *Inform. Retr. 4,* 2, 133–151.

HERLOCKER, J., KONSTAN, J. A., AND RIEDL, J. 2002. An empirical analysis of design choices in neighborhood-based collaborative filtering algorithms. *Inform. Retr. 5,* 4, 287–310.

HERLOCKER, J. L., KONSTAN, J. A., BORCHERS, A., AND RIEDL, J. 1999. An algorithmic framework for performing collaborative filtering. In *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'99)*. ACM, New York, NY, 230–237.

HERLOCKER, J. L., KONSTAN, J. A., TERVEEN, L. G., AND RIEDL, J. T. 2004. Evaluating collaborative filtering recommender systems. *ACM Trans. Inform. Syst. 22,* 1, 5–53.

HOFMANN, T. 2004. Latent semantic models for collaborative filtering. *ACM Trans. Inform. Syst. 22,* 1, 89–115.

HUANG, Z., CHEN, H., AND ZENG, D. 2004. Applying associative retrieval techniques to alleviate the sparsity problem in collaborative filtering. *ACM Trans. Inform. Syst. 22,* 1, 116–142.

HUANG, Z., ZENG, D., AND CHEN, H. 2007. A comparison of collaborative-filtering recommendation algorithms for e-commerce. *IEEE Intell. Syst. 22,* 5, 68–78.

JIN, R., SI, L., AND ZHAI, C. 2003. Preference-based graphic models for collaborative filtering. In *Proceedings of the 19th Conference in Uncertainty in Artificial Intelligence*. 329–336.

KARYPIS, G. 2001. Evaluation of item-based top-n recommendation algorithms. In *Proceedings of the 10th International Conference on Information and Knowledge Management (CIKM'01)*. ACM, New York, NY, 247–254.

KOHRS, A. AND MÉRIALDO, B. 1999. Clustering for collaborative filtering applications. In *Proceedings of the International Conference on Computational Intelligence for Modeling, Control and Automation (CIMCA'99)*.

KOREN, Y. 2008. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceeding of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'08)*. ACM, New York, NY, 426–434.

LAM, S. K. AND RIEDL, J. 2004. Shilling recommender systems for fun and profit. In *Proceedings of the 13th International Conference on World Wide Web (WWW'04)*. ACM, New York, NY, 393–402.

LEMIRE, D. AND MACLACHLAN, A. 2005. Slope one predictors for online rating-based collaborative filtering. In *Proceedings of SIAM Data Mining (SDM'05)*.

MARLIN, B. 2004. Collaborative filtering: A machine learning perspective. M.S. thesis, University of Toronto.

MARLIN, B. M. AND ZEMEL, R. S. 2009. Collaborative prediction and ranking with non-random missing data. In *Proceedings of the 3rd ACM Conference on Recommender Systems (RecSys'09)*. ACM, New York, NY, 5–12.

MCNEE, S. M., RIEDL, J., AND KONSTAN, J. A. 2006. Being accurate is not enough: how accuracy metrics have hurt recommender systems. In *Extended Abstracts on Human Factors in Computing Systems (CHI'06)*. ACM, New York, NY, 1097–1101.

MELVILLE, P., MOONEY, R., AND NAGARAJAN, R. 2001. Content-boosted collaborative filtering. In *Proceedings of the ACM SIGIR Workshop on Recommender Systems*.

MOBASHER, B., BURKE, R., BHAUMIK, R., AND WILLIAMS, C. 2007. Toward trustworthy recommender systems: An analysis of attack models and algorithm robustness. *ACM Trans. Internet Technol. 7,* 4, 23.

PATEREK, A. 2007. Improving regularized singular value decomposition for collaborative filtering. In *Proceedings of the KDD Cup Workshop at the 13th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD'07)*. 39–42.

PENNOCK, D., HORVITZ, E., LAWRENCE, S., AND GILES, C. L. 2000. Collaborative filtering by personality diagnosis: A hybrid memory- and model-based approach. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence (UAI'2000)*. 473–480.

RENNIE, J. D. M. AND SREBRO, N. 2005. Fast maximum margin matrix factorization for collaborative prediction. In *Proceedings of the 22nd International Conference on Machine Learning (ICML'05)*. ACM, New York, NY, 713–719.

RESNICK, P., IACOVOU, N., SUCHAK, M., BERGSTROM, P., AND RIEDL, J. 1994. Grouplens: an open architecture for collaborative filtering of netnews. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW'94)*. ACM, New York, NY, 175–186.

SANDVIG, J. J., MOBASHER, B., AND BURKE, R. 2007. Robustness of collaborative recommendation based on association rule mining. In *Proceedings of the ACM Conference on Recommender Systems (RecSys'07)*. ACM, New York, NY, 105–112.

SARWAR, B., KARYPIS, G., KONSTAN, J., AND REIDL, J. 2001. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th International Conference on World Wide Web (WWW'01)*. ACM, New York, NY, 285–295.

SARWAR, B., KARYPIS, G., KONSTAN, J., AND RIEDL, J. 2000. Application of dimensionality reduction in recommender systems–a case study. In *Proceedings of the ACM WebKDD Workshop*.

SCHEIN, A. I., POPESCUL, A., UNGAR, L. H., AND PENNOCK, D. M. 2002. Methods and metrics for cold-start recommendations. In *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'02)*. ACM, New York, NY, 253–260.

SHAFIEI, M. AND MILIOS, E. 2006. Model-based Overlapping Co-Clustering. In *Proceedings of the 4th Workshop on Text Mining at the 6th SIAM International Conference on Data Mining*.

SHARDANAND, U. 1994. Social information filtering for music recommendation. M.S. thesis, Massachussets Institute of Technology.

SHARDANAND, U. AND MAES, P. 1995. Social information filtering: algorithms for automating "word of mouth". In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'95)*. ACM Press/Addison-Wesley Publishing Co., New York, NY, 210–217.

SI, L. AND JIN, R. 2003. A flexible mixture model for collaborative filtering. In *Proceedings of the 20th International Conference on Machine Learning*.

UNGAR, L. AND FOSTER, D. 1998. Clustering methods for collaborative filtering. In *Proceedings of the Workshop on Recommendation Systems*. AAAI Press, Menlo Park, CA.

VUCETIC, S. AND OBRADOVIC, Z. 2000. A regression-based approach for scaling-up personalized recommender systems in e-commerce. In *Proceedings of the ACM WebKDD Workshop*.

WANG, J., DE VRIES, A. P., AND REINDERS, M. J. T. 2006. Unifying user-based and item-based collaborative filtering approaches by similarity fusion. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'06)*. ACM, New York, NY, 501–508.

XUE, G.-R., LIN, C., YANG, Q., XI, W., ZENG, H.-J., YU, Y., AND CHEN, Z. 2005. Scalable collaborative filtering using cluster-based smoothing. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'05)*. ACM, New York, NY, 114–121.

Received September 2008; revised May 2009, November 2009; accepted July 2010