Analysis of Recommender Systems Based on Implicit Datasets

ELISA CAMPOCHIARO
Neptuny, Italy
PAOLO CREMONESI
and
ROBERTO TURRIN
Politecnico di Milano, Italy

Recommender systems are a powerful technique developed to help users searching for interesting items (e.g., books, movies, ...) in domains affected by information overload. A recommender system exploits the information about users and items in order to suggest attractive and personalized products to users. The information about users and items is represented as a dataset and it can be either explicit or implicit. In the first case we know both which products the users purchased and the opinion they expressed. On the other hand, in the case of implicit information, we only know which items the users bought. Since such approach is typically easier, several datasets are implicit rather than explicit, e.g., in the e-commerce settings. In addition, in many cases we deal with mixed datasets, formed by a predominant percentage of implicit information and a very small number of explicit ratings.

In this work we compare the quality of recommendations obtained by using explicit and implicit datasets. We show that, by means of algorithms tuned for implicit datasets, it is possible to reach a recommendation quality very close to the explicit case, with the advantage of not annoying the users with questions about their interests. In order to perform such comparison we convert two famous explicit datasets (Netflix and Movielens) into a binary form, using two different binarization techniques. We test dimensionality reduction-based and item-based algorithms, using an evaluation methodology suited to the top-N recommendation task, i.e., a context where the focus is in finding a very limited number of interesting items to suggest to the user, rather than in accurately predicting the ratings. Moreover, we propose an analysis based on the item popularity, showing different behaviors of dimensionality reduction-based algorithms in recommending popular and unpopular items.

Categories and Subject Descriptors: Knowledge Discovery for Personalized Information Supply []:
General Terms: Recommender Systems, Implicit Datasets, Top-N Recommendation Task, Item Popularity, Dataset Binarization

1. INTRODUCTION

Recommender systems alleviate the problem of searching for personalized resources by proposing a very limited set of items (e.g., movies, music, books, news, web pages,...) that a user is interested in.

Recommender systems are particularly useful in situations where the number of items is continuously growing and the amount of interesting resources is relatively low. Well-known examples include e-commerce applications dealing with books, songs or movies [Resnick et al. 1994; Linden et al. 2003], which offer the capability to exhibit an almost unlimited number of items. As the universe of available resources increases, each item gets less and less visibility. The consequence is that

users are scarcely stimulated in purchasing, not because the desired products are not present in the catalog, rather because they are not able to find them.

The solution could be to provide the users with a fast and efficient keyword-based search tool. Unfortunately, not all domains match with such a solution. For example, IPTV (IP Television) users typically use a remote controller to browse the available television programs. Furthermore, very often customers do not know in advance what they are looking for or they just have a weak idea. In addition, keyword-based search engines make users only discover what they are searching for, preventing, for example, the impulsive shopping, a very powerful leverage of sales. Product promotion might be a solution, but, in most cases, the users are presented general and popular items which may result irrelevant. Promotion pushes some items up, but it does not capture the *long-tail* of potential sales [Anderson 2006].

Several approaches for building recommender systems have been proposed over the past decades [Resnick et al. 1994; Konstan et al. 1997; Schafer et al. 1999; Kitts et al. 2000]. Between the two typical techniques, the content-based [Balabanović and Shoham 1997; Pazzani and Billsus 2006; 1997] and the collaborative [Goldberg et al. 1992], the latter is probably the most successful and widely used.

Collaborative recommender systems are based on a dataset of user ratings, i.e., numerical values which express the preferences of users about items. Such ratings can be either [Nichols 1998]:

(i) *explicit*: the user actively judges an item by assigning a numerical value to it (e.g., the typical 1-to-5 stars rating scale).

(ii) *implicit*: the system implicitly infers the rating by observing the user behavior (e.g., the user buys an item, watches a movie, browses a web page); while in few applications there exists the possibility to infer a value for the user rating by monitoring its behavior (e.g., the time spent reading a news [Konstan et al. 1997; Morita and Shinoda 1994]), in the general case the implicit ratings are usually binary [Deshpande and Karypis 2004].

Since the famous Netflix contest has been advertised [Bennett and Lanning 2007] great effort has been dedicated to recommender systems specifically designed for explicit datasets.

However, explicit ratings are subjected to personal interpretation of the rating scale: for instance, in a 1-to-5 rating scale, a rating equals to 3 could mean a positive rating for a given user, while for another user it could represent an item he does not like. Moreover, in several domains we typically dispose only of an implicit dataset (i.e., binary ratings) or, sometimes, of very few explicit ratings that coexist in a mainly-implicit dataset. This happens because many applications typically leave the users the possibility to rate the item that they have just purchased, but most of the times users simply skip this operation. Therefore, in most of the cases, the only available information is simply whether the user bought or not an item. Indeed, should the user be forced in expressing a judgment about an item, he could get annoyed and he would probably give a hasty rating.

As a consequence of the above discussion, it is common to have a dataset that is composed by a mixture of explicit and implicit ratings, where the latter are usually the dominant percentage. The solution could be either to neglect the implicit ratings, but it would worsen the already critical dataset sparsity, or to convert

the implicit ratings into explicit, by means of machine learning-based techniques [Crammer and Singer 2001; Fung et al. 2006].

In this paper we probe whether all the efforts done for collecting explicit ratings and using algorithms tailored on explicit dataset lead to an actual advantage with respect to using algorithms based on implicit datasets, in particular in terms of achieving the *top-N recommendation task* [Deshpande and Karypis 2004; Herlocker et al. 2004], i.e., the capability of the system in recommending a very limited but accurate list of interesting items.

The contribute of this work is threefold: (i) we show that the quality of recommendations provided by using datasets based on implicit ratings is comparable with the quality provided by using explicit ratings, (ii) we analyze several algorithms implemented for implicit datasets and (iii) we investigate their behavior with different classes of items (e.g., with respect to their popularity).

As for the first result, we take into consideration two well-known explicit datasets: Netflix [Bennett and Lanning 2007] and MovieLens [Resnick et al. 1994]. In order to produce comparable results among the explicit and the implicit cases, we operated the binarization of these datasets converting them from explicit to implicit. On the implicit versions of MovieLens and Netflix we apply different recommender algorithms, item-based and dimensionality reduction-based. Based on a common test methodology, we evaluate the quality of the algorithms by means of classification accuracy metrics: recall, fallout, precision and F-measure. The experiments performed with the explicit ratings are expected to reach higher quality because they exploit the greater quantity of available information. However, we show that with implicit datasets it is possible to obtain results with a quality comparable to the explicit case. In addition, implicit datasets can take advantage of ad-hoc recommender algorithms, tuned in terms of performance and memory requirements. Therefore, we can accept to slightly reduce the quality of recommendations by using an implicit dataset, if this means avoiding to bore the users with questions about the purchased items.

For the second step, we evaluate the quality of different algorithms tested on a new implicit, private dataset, made available by one of the most prominent IPTV provider in Europe. The new dataset, from here on referred to as the NewMovies dataset, has been collected by logging the movies purchased by the users during a three-month period in the year 2007. With respect to the Netflix and MovieLens datasets, the NewMovies dataset better reflects the information available to web-based tools about user profiles.

Finally, implicit algorithms are deeply inspected in order to evaluate their behavior with popular and unpopular items. In fact, it is known that collaborative recommender algorithms can be biased toward popular items. More specifically, we show that, by varying the value of a parameter in a dimensionality reduction-based algorithm, we can accordingly tune the capability of the algorithm in recommending popular and unpopular items.

The rest of the work is organized as follows. In Section 2 we present the state of the art of recommender systems, analyzing the characteristics of content-based and collaborative recommender algorithms. Section 3 describes the typical architecture of a recommender system, distinguishing between batch and real time processes.

Section 4 formalizes the collaborative recommender algorithms used within this work, namely item-based and dimensionality reduction-based. In Section 5 we first summarize the dataset partitioning techniques and the evaluation metrics used in the literature. Then we propose a new evaluation methodology, suited to the top-N recommendation task, and we analyze the statistical characteristics of the datasets used in the tests. In Section 6 we present the results of the tests and analyze the complexity of the algorithms, in terms of the time necessary for constructing the model. Finally, Section 7 draws the conclusions and suggests possible further works.

2. STATE OF THE ART

The simplest way to suggest items to users is by means of a *non-personalized* system, which uses popularity information, such as people's ratings or sales data, and recommends items with the highest average rating (e.g., a list of the top-10 most popular items). While this is very easy, the users are presented general items which may result irrelevant.

A solution could be personalizing the recommendations with some *demographic information* (e.g., age, gender, area code, preferred category, ...). The system will recommend items that are preferred by users with the same demographic profile. This is a rough example of the recommender class later referred to as collaborative recommender. By means of such information, new users can get recommendations without providing information about their preferences. On the other hand, highly personalized recommendations can not be achieved because of the exiguity data about user profiles.

In the following we present two approaches, known as *content-based* and *collaborative* filtering, effectively used to generate personalized recommendations based on user preferences.

2.1 Content-based filtering

The content-based approach to recommendation has its roots in the information retrieval field [Salton 1988; Belkin and Croft 1992]. Many current content-based systems focus on recommending items containing textual information, such as documents, web sites, usenet news and mail messages (e.g., anti-spam filtering) [Pazzani and Billsus 2006; 1997; Mooney and Roy 2000; Deerwester et al. 1990]. The improvement over the traditional information retrieval derives by adopting user profiles that contain information about tastes, preferences and needs of users. A content-based system is based on the analysis of the content of the items. The model of an item is so composed by a set of features representing its content. The assumption underlying the content-based techniques is that the meaning and the relevance of items can be captured by such features [Salton 1988].

In a content-based recommender, the user is limited to be recommended items that are extremely similar to those already rated. The point is that content-based systems can not recommend items that are different from anything the user has rated before, but they are limited to the user's past activity: a user who bought, so far, only books written by Isaac Asimov, will never be presented books written by other authors, even if they were interesting for the user.

A final problem related to the content analysis is that, if two different items are represented by the same features, they result equivalent. Thus, for example,

	Memory-based	Model-based
User-based	X	
Item-based		X
Dimensionality reduction-based		X

Table I. Taxonomy of collaborative algorithms.

content-based systems can not distinguish between a valuable movie and a bad one, if they have been shot by the same director or they have the same set of actors.

2.2 Collaborative Filtering

In opposition to content-based methods, collaborative systems try to suggest items to a particular user on the basis of the preferences expressed by other users [Goldberg et al. 1992]. In fact, in everyday life, we often rely on recommendations from other people such as by word of mouth or product reviews [Shardanand and Maes 1995]. Such systems use the opinions of a community of users to help individuals more effectively identify interesting content. Collaborative systems assist and augment this process. They are based on the following two assumptions:

i. groups of users with similar tastes rate the items similarly; for a certain user, we refer to the users similar to him as his *neighborhood*;
ii. correlated items are rated by users similarly; for a certain item, we refer to the items similar to it, according to such definition, as its *neighborhood*.

This latter concept of *correlation* states that whatever the content of an item is, such item is considered somehow similar to another one because the community expresses the same evaluation for both the items.

Starting from the previous two points, we can derive two classes of collaborative filtering, respectively, the (i) user-based and the (ii) item-based collaborative filtering [Wang et al. 2006]. User-based collaborative algorithms are also referred to as *memory-based* in opposition to item-based collaborative algorithms which are *model-based*. In fact, user-based algorithms work on the whole dataset (e.g., users or items), while item-based algorithms produce a representation of the dataset (i.e., the model). Consequently, user-based algorithms do not scale well when the number of users or items increases. Furthermore, item-based algorithms are proven to provide a better quality and with a lower computational cost [Sarwar et al. 2001; Papagelis and Plexousakis 2005]. Within the class of model-based algorithms, we can include the dimensionality reduction-based approaches, which rely on representing the dataset by means of a low-dimensional space. Table I summarizes the taxonomy of collaborative algorithms. Sections 4.1 and 4.2 describe several algorithms which follow, respectively, the item-based and the dimensionality reduction-based approach.

Note that collaborative recommendations do not require anyhow to extract any explicit feature from the items: they are simply based on the community behavior. Thus, such systems do not have the same shortcomings that content-based systems have. In particular, since collaborative systems use other-users' preferences, they can deal with any kind of content. Furthermore they can recommend any items, even the ones with a content which does not correspond to any item previously

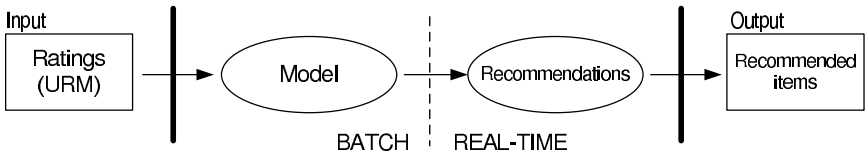


Fig. 1. System architecture

liked. The positive side-effect is that collaborative filtering promotes the serendipity (i.e., the system can recommend an item that the user does not expect, but he likes the item and thus he develops a completely new interest).

However, also collaborative systems have their own limitations, as described in [Adomavicius and Tuzhilin 2005].

The main drawback is that collaborative recommenders are affected by the *new item* (or first rater) problem. Since such systems recommend the items most preferred by the user's neighborhood, a new item will not be recommended to anyone because nobody rated it so far. Therefore, until the new item is rated by a substantial number of users, the recommender system would not be able to recommend it. Note that content-based recommenders do not suffer such a problem because the new items enter the collection with their own features.

A second issue is called the *sparsity* problem. In fact, the effectiveness of collaborative systems depend on the availability of sets of users with similar preferences. Unfortunately, in any recommender system, the number of ratings already collected is usually very small compared to the number of ratings to estimate. As a consequence, it might not be possible to recommend someone with unique tastes, because there will not be anyone enough similar to him.

Following from the above two points, at the beginning of its activity, a brand new recommender system will not be able to provide any accurate recommendation; this is referred to as the *cold start* problem.

In addition, since popular items are the most rated, collaborative algorithms are likely to be biased toward the most popular items. For instance, in a book recommender system, there may be many books that have been rated by only few people and these books would be recommended very rarely, even if those few users gave high ratings to them.

3. RECOMMENDER SYSTEM ARCHITECTURE

This section describes the typical architecture of a recommender system. Referring to Figure 1, the input to the system is represented by the user-rating matrix (URM), referred to as $\mathbf{R}$. The URM contains information describing the user interactions with the e–commerce system (e.g., the rating expressed by a user about an item). This is the only information required by collaborative recommender algorithms.

Each row of $\mathbf{R}$ represents a user profile, each column an item. Thus, the element r_{pi} at the p-th row and i-th column is the rating of user profile p about item i. Such rating is null (i.e., zero) if the user did not interact with the item.

Depending on the platform capability, the user rating can be either explicit or implicit. Implicit ratings do not allow to know whether the user is actually interested in the item, while explicit ratings are more confident in representing the real

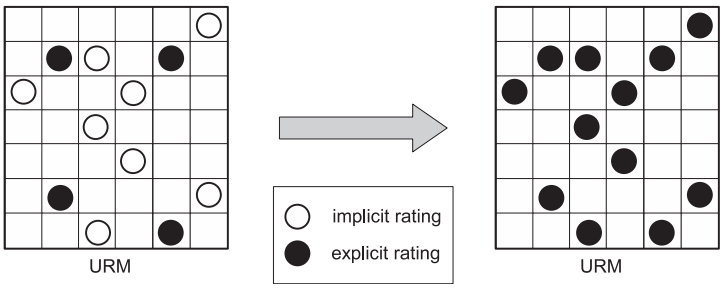


Fig. 2. Coexistence of implicit and explicit ratings. The implicit ratings are transformed into explicit ratings.

user interest toward an item.

The simplest form of implicit rating is represented by a *binary* classification: if a user interacted with an item (e.g., the user purchased a book) the corresponding rating is non-zero.

Systems collecting explicit ratings are expected to reach better results, i.e., to be able to acquire a more precise user profile, because users are clearly expressing their own satisfaction. Note, however, that explicit ratings can be affected by user subjectivity, i.e., personal interpretation that each user gives to the rating scale. We will see in Section 4.1.1 that the user rating bias can be modeled by means of a rating normalization.

The main issue with explicit ratings in e–commerce applications is that the amount of information collected can be dramatically poor. The main reason is that only a small percentage of users voluntary rate the purchased items and any tentative to force them would result in random or hasty judgments. Consequently, the URM is often characterized by a predominant percentage of implicit ratings. A solution to this problem could be to predict the actual value of implicit ratings by using the few explicit information which is available (see Figure 2). In Section 6 we propose to reduce all the explicit information into implicit (i.e., binary) values.

3.1 Batch and Real-Time Processes

The recommender system should guarantee, among other properties:

(i) High quality of recommendations, i.e., the capability to suggest personalized items which are potentially interesting to the user, and to discard items which are disliked by the user. Observe that, when the user pays for an item, the fact that the system suggests a product that he will not like, has a very strong, negative impact on the user trust in the system.

(ii) Capability to recommend any user at any time, e.g., it must properly recommend a user even if his profile has just been updated.

(iii) Scalability and real-time performance. Recommender systems require calculations, which grow both with the number of customers and with the number of items. An algorithm that is efficient when the number of data is limited can turn out to be unable to generate a satisfactory number of recommendations as soon as the amount of data increases. Thus, it is crucial to apply algo-

rithms which are capable of scaling up in a successful manner with real-time response times [Sarwar 2001].

For the aforementioned reasons, the architecture of a recommender system is divided into two asynchronous components, the *batch* (off-line) and the *real-time* (on-line) processes, as shown in Figure 1.

The batch component faces with the most expensive part of the computation, with the highest memory and computing requirements. The purpose is to move as many operations as possible in this part in order to lighten the on-line part. In fact, the on-line component copes with the real-time requests for recommendation coming from the client interface, with rigid time constraints. Thus, on-line phase should consist of few and quick operations.

Model-based algorithms [Herlocker et al. 2004; Sarwar et al. 2001] well fit in this architecture. In fact, they work on a representation of the dataset, i.e., the model, which captures the principal information (e.g., relationships among items), rather than on the whole dataset. The model generation is the most demanding operation, thus it is delegated to the batch part.

Despite such logical division of the recommending process, the model construction in real domains can still result challenging because of the dataset size and the memory and time requirements. Hence, particular attention must be payed to the computation feasibility when facing with real datasets, thus adopting very high scalable solutions.

4. RECOMMENDER ALGORITHMS

Collaborative algorithms are able to recommend users by means of the information collected in the user-rating matrix $\mathbf{R}$. Let us suppose $\mathbf{R}$ is composed by n users and m items: it results in a $n \times m$ matrix. In this work we consider two families of model-based collaborative algorithms: (i) item-based and (ii) dimensionality reduction-based. In the following we describe such algorithms, treating separately the cases of implicit and explicit URM.

4.1 Item-based

Item-based collaborative algorithms capture the fundamental relationships among items. As explained in Section 2.2, two items are similar (from a 'collaborative' point of view) if the community agrees about their ratings. Such similarity can be represented in a $m \times m$ matrix, referred to as $\mathbf{D}$, where the element d_{ij} expresses the similarity between item i and item j. Note that, potentially, $\mathbf{D}$ could be non-symmetric, i.e., $d_{ij} \neq d_{ji}$. That means that, for instance, item i could be very similar to item j (thus if a user likes item i he would like item j), even if item j is not similar to item i.

Item-based algorithms usually represent items in the user-rating space, i.e., an item is a vector whose dimensions are the ratings given by the n users. The coordinate of each dimension is the user rating. As a consequence, item i corresponds to the i-th column of $\mathbf{R}$, and the relationships among items are expressed by means of the similarities among the related vectors. In the following sections we describe several techniques to calculate the similarities among these vectors.

According to the system architecture shown in Figure 1, matrix $\mathbf{D}$ represents

the model of the recommender system and its calculation, being computational intensive, is delegated to the batch part of the recommender system. The real-time part generates a recommendation list by using such model. Given the profile of the target user p to recommend (represented by a vector of ratings), we can predict the rating $\hat{r}_{pi}$ by computing the weighted sum of the ratings given by user p on the items similar to i. Such ratings are weighted by the similarity with item i. Referring to Q_i as the set of items similar to i, the prediction of $\hat{r}_{pi}$ can be formulated as:

$$\hat{r}_{pi} = \frac{\sum_{j \in \mathrm{Q}_i} d_{ji} \cdot r_{pj}}{W} \tag{1}$$

where W is a normalization factor. Such factor could be simply set to 1 or, as discussed in [Sarwar et al. 2001], it can be computed as:

$$W = \sum_{j \in \mathrm{Q}_i} |d_{ji}| \tag{2}$$

thus assuring that $\hat{r}_{pi}$ is within the predefined rating scale. Note that, being a model-based approach, user p can be recommended even tough it is not taken into account during the model construction. This allows, for example, (i) to build the model with a subsample of users (e.g., in order to respect time and memory constraints) and (ii) to recommend a user even if his profile is new or update with respect to the moment the model was calculated.

Once computed the predicted ratings for all the items in the dataset that have not been rated by the target user, such ratings are sorted and the N highest-rated items compose the top-N recommendation list.

The set Q_i can be reduced by considering, for instance, only the items with a similarity greater than a certain threshold, or the k most similar items. This latter approach is the classical kNN (k-nearest-neighbors) approach. Section 6 shows that, by varying k, the quality of recommendations varies accordingly.

Alternative ways to compute the real-time prediction are based, for instance, on approximating the ratings by means of regression models [Sarwar et al. 2001].

4.1.1 *Explicit ratings.* When the URM is formed by explicit ratings, the classical similarity metric among vectors is the *cosine* similarity [Deshpande and Karypis 2004]. In fact, the cosine of the angle between two vectors represents their similarity. The element d_{ij} is defined as:

$$d_{ij} = \frac{\sum_{e=1}^{n} r_{ei} \cdot r_{ej}}{\sqrt{\sum_{e=1}^{n} r_{ei}^2} \cdot \sqrt{\sum_{e=1}^{n} r_{ej}^2}} \tag{3}$$

The resulting similarity values are in the range $[0, 1]$.

As pointed out in Section 3, explicit ratings can be biased by user subjectivity. The difference in rating scale between different users is not taken into account by the cosine metric. For instance, in a rating scale between 1 and 5, some user could use the value 3 to indicate an interesting item, someone else could use 3 for a not much interesting item. An alternative is given by the *adjusted cosine* similarity [Sarwar et al. 2001], which alleviates this drawback by subtracting the corresponding user

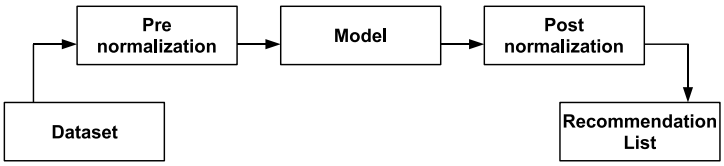


Fig. 3. Pre and post data normalization.

average, i.e., the unbiased rating $\tilde{r}_{pi}$ is given by:

$$\tilde{r}_{pi} = r_{pi} - \bar{r}_p \tag{4}$$

where $\bar{r}_p$ is the user mean rating calculated over all the non-zero ratings of user p. The adjusted cosine metric is define as:

$$d_{ij} = \frac{\sum_{e \in B}(r_{ei} - \bar{r}_e)(r_{ej} - \bar{r}_e)}{\sqrt{\sum_{e \in B}(r_{ei} - \bar{r}_e)^2}\sqrt{\sum_{e \in B}(r_{ej} - \bar{r}_e)^2}}, \tag{5}$$

The set B is composed by users who rated both item i and item j, and $\bar{r}_e$ is the average of ratings given by user e. The values d_{ij} are in the range $[-1, 1]$.

Referring to Figure 3, the user subjectivity is modeled in the adjusted cosine metric by applying a *pre–normalization* (i.e., before the model construction) to the dataset. This is not the only possible rating normalization; in Section 4.2.1 we discuss other pre–normalizations. Furthermore, in the following section we show that we can apply a normalization also on the resulting model: this is referred to as *post–normalization*.

4.1.2 *Implicit ratings.* When using implicit datasets, similarity metric is usually computed using a frequency-based approach, as the one discussed by Deshpande and Karypis in [Deshpande and Karypis 2004]. For instance, when we only dispose of binary values, a high similarity between item i and item j means that when someone buys item i, it is very likely that he will buy also item j.

We can treat implicit datasets in the same way as explicit datasets, i.e., by considering each item as a vector in the user-rating space, where now the coordinates are binary values. Again, the similarity between two items can be computed as the similarity between the correspondent vectors, for example by means of the cosine metric (3). Observe that the adjusted cosine is not applicable because, by subtracting the average of user ratings, we would reset to zero all the user ratings.

With regard to implicit ratings, the cosine similarity is a special case of a more general approach that we refer in the following as direct relations (DR). In its basis formulation, the item-item matrix $\mathbf{D}$ used with the DR is given by:

$$\mathbf{D} = \mathbf{R}^{\mathrm{T}} \cdot \mathbf{R} \tag{6}$$

The elements d_{ii} on the principal diagonal is the total number of ratings for item i, while the other elements d_{ij} represent the number of users that have seen both item i and item j. Referring again to Figure 3, the model (i.e., $\mathbf{D}$) can be post-processed by means of a post-normalization, whose general expression is:

$$d_{ij} \leftarrow \frac{d_{ij}}{{d_{ii}}^{\beta}{d_{jj}}^{\gamma}} \tag{7}$$

where γ and β are constant parameters whose optimal values depend on the dataset. For instance, when $\gamma = \beta = 0.5$, (7) corresponds to the cosine between columns i and j of R, while when $\gamma = 0$ and $\beta = 1$, the element d_{ij} represents $\mathrm{P}(i|j)$, i.e., the conditional probability of buying item i given that item j has been already bought.

Note that the aforementioned approaches are based on counting the co-rated items and they can be efficiently performed by any DBMS (Database Management System) using simple SQL statements without the need of external programs.

4.2 Dimensionality reduction-based

Collaborative algorithms based on dimensionality reduction techniques describe the dataset (i.e., users and items) by means of a limited set of *features*. These features are different in their meaning from the features typically extracted in the case of content-based algorithms. In fact, the latter are characteristics concerning the content of items (e.g., the genre of a movie, the singer of a song,...), while the features used by collaborative algorithms are not based on the content, but on the implicit way the user community interacts with the items.

Let us assume that an item can be described by means of l features, i.e., it is represented as a vector in the l-dimensional feature space. Similarly, a user is represented by a vector in the same space. As a consequence, the correlation between user p and item i (i.e., how much the item matches the user interests) can be computed as the similarity between the correspondent vectors, for instance by means of their inner product:

$$\hat{r}_{pi} = \sum_{e=1}^{l} a_{pe} \cdot b_{ie} \tag{8}$$

where, a_{pe} and b_{ie} are the e-th (unknown) features for user p and item i, respectively.

The point is to compute the l features which minimize the prediction error between the estimated $\hat{r}_{pi}$ and the actual value r_{pi}.

For instance, Paterek in [Paterek 2007] applies an optimization method, referred to as regularized singular value decomposition, already used in the domain of natural language processing [Gorrell 2006]. The l features of users and items are estimated by minimizing the metric RMSE (Root Mean Square Error), one feature at a time, using an optimization technique based on gradient descent. The metric RMSE is defined as:

$$\mathrm{RMSE} = \sqrt{\frac{1}{n} \sum_{p,i} \left(\hat{r}_{pi} - r_{pi}\right)^2} \tag{9}$$

Alternatively, the classical formulation of singular value decomposition (SVD) can be applied directly to the URM, similarly to a process well known in the settings of information retrieval as latent semantic analysis (LSA) [Furnas et al. 1988; Husbands et al. 2000]. In fact, assuming that the URM has rank q, it can be factorized as:

$$\mathbf{R} = \mathbf{U} \cdot \mathbf{S} \cdot \mathbf{V}^{\mathrm{T}} \tag{10}$$

where $\mathbf{U}$ is a $n \times q$ matrix, $\mathbf{V}$ is a $m \times q$ matrix, and $\mathbf{S}$ is a $q \times q$ diagonal matrix containing the singular values, sorted in decreasing order. The columns of $\mathbf{U}$ and

$\mathbf{V}$ are orthonormal, i.e., $\mathbf{U}^T\mathbf{U} = \mathbf{I}$ and $\mathbf{V}^T\mathbf{V} = \mathbf{I}$, being $\mathbf{I}$ the identity matrix with ones on the main diagonal and zeros elsewhere. Note that the SVD is unique except for some linear combinations of rows and columns of the three resulting matrices. Conventionally, the diagonal elements of $\mathbf{S}$ are constructed so to be all positive and sorted by decreasing magnitude.

The actual interest in the SVD derives from the fact that it allows to approximate $\mathbf{R}$ in a low-dimensional space. Indeed, by keeping only the first $l < q$ largest singular values (i.e., the l most significant features), we obtain three new matrices, $\mathbf{U}_l$ $(n \times l)$, $\mathbf{S}_l$ $(l \times l)$ and $\mathbf{V}_l$ $(m \times l)$. In order to simplify the notation, we refer to these new matrices again as $\mathbf{U}$, $\mathbf{S}$ and $\mathbf{V}$, respectively. The product of these matrices is the best rank-l linear approximation of $\mathbf{R}$ in terms of the Frobenius norm [Saad 1992]:

$$\hat{\mathbf{R}} = \mathbf{U} \cdot \mathbf{S} \cdot \mathbf{V}^T \tag{11}$$

Thus, the rating of user p about item i can be predicted as:

$$\hat{r}_{pi} = \sum_{e=1}^{l} u_{pe} \cdot s_{ee} \cdot v_{ie} \tag{12}$$

where u_{pe} is the element in the p-th row and e-th column of $\mathbf{U}$, v_{ie} is the element in the i-th row and e-th column of $\mathbf{V}$, and s_{ee} is the singular value in the e-th row and e-th column of $\mathbf{S}$.

Assuming that $\mathbf{u}_p$ represents the p-row of $\mathbf{U}$ and $\mathbf{v}_i$ the i-row of $\mathbf{V}$, (12) can be rewritten as:

$$\hat{r}_{pi} = \mathbf{u}_p \cdot \mathbf{S} \cdot \mathbf{v}_i^T \tag{13}$$

Reminding that $\mathbf{U}$ and $\mathbf{V}$ have orthonormal columns, by multiplying both terms of (11) by $\mathbf{V}$, we can state that:

$$\mathbf{u}_p \cdot \mathbf{S} = \mathbf{r}_p \cdot \mathbf{V} \tag{14}$$

where $\mathbf{r}_p$ is the p-th row of $\mathbf{R}$ (i.e., the profile vector of user p). Consequently, (13) can be reformulated as:

$$\hat{r}_{pi} = \mathbf{r}_p \cdot \mathbf{V} \cdot \mathbf{v}_i{}^{\mathrm{T}} \tag{15}$$

By means of (15) we are able to recommend any user, even if his profile $\mathbf{r}_p$ is new or it has been updated since the model was created (i.e., since the SVD was performed). This represents a great advantage when compared, for instance, with other dimensionality-reduction techniques (e.g., the regularized SVD), where the features for a certain user are pre–computed and fixed during the model construction.

In order to predict all the ratings for user p, (15) can be straightforwardly extended as:

$$\hat{\mathbf{r}}_p = \mathbf{r}_p \cdot \mathbf{V} \cdot \mathbf{V}^{\mathrm{T}} \tag{16}$$

Note that the product between $\mathbf{V}$ and $\mathbf{V}^{\mathrm{T}}$ results into a $m \times m$ item-item matrix, whose meaning is very similar to the item-item matrix $\mathbf{D}$ discussed in Section 4.1 about item-based algorithms.

There are several advantages in using such SVD-based approach:

—The SVD represents items and users in a low-dimensional space. Once the URM has been factorized, which can result particularly challenging, the system oper-

ates with vectors having only l dimensions, much less than the original space of n users and m items.

—The SVD reduces the noise in the data. In fact, by neglecting the singular values with low magnitude we are discarding the least-informative data, which is typically noisy [Furnas et al. 1988; Deerwester et al. 1990].

—The SVD strengthens the relationships among the data. Thus, if two vectors (either users or items) are similar (because somehow related), they are represented closer in the l-dimensional feature space than in the original space. Observe that the relationship might also be indirect, i.e., by means of the SVD we could discover hidden dependences among users or items.

The major issue with the SVD is its computational complexity, which, in the general case, is $O(mn^2)$. Anyway, in the case of sparse matrices, there exist very efficient and scalable solutions. For instance, the SVD implementation by Lanczos [Berry 1992] is optimized for sparse, large matrices: referring to z as the number of non-zero elements in the URM, the memory requirement is $O(z)$, and the computational complexity is $O(zl)$, i.e., directly proportional to z and to the number of singular values to be computed [Zhang et al. 2001; Sarwar et al. 2000b].

For the above reasons, with regard to the system architecture described in Section 3, the SVD factorization (10) is delegated to the batch part, while the prediction of ratings (15) can be performed at real-time. The real-time process estimates the ratings for all the unrated items of the target user, then such ratings are sorted and the N highest-rated items are selected to form the top-N recommendation list. In our tests, the time spent for computing the top-N recommendation list of a certain user by means of (15) is few milliseconds, fitting any real-time requirements.

In the following we discuss the application of the SVD-based approach in the explicit and implicit cases.

4.2.1 *Explicit ratings.* As well as we discussed in Section 4.1.1, explicit ratings can be biased by user subjectivity, due to arbitrary interpretation of the rating scale. This bias can be modeled by means of normalization techniques, as they are expressed by (4). Once applied the normalization, the model can be constructed by means of (10), where $\mathbf{R}$ is the normalized URM. The real-time recommendations are calculated by using (15), where, instead of the vector $\mathbf{r}_p$, we use the normalized vector $\tilde{\mathbf{r}}_p$ of user p, i.e., :

$$\hat{r}_{pi} = \tilde{\mathbf{r}}_p \cdot \mathbf{V} \cdot {\mathbf{v}_i}^{\mathrm{T}} \tag{17}$$

In order to maintain the values within the predefined rating scale, the ratings in (17) have to be scaled by adding up the user-p mean rating $\bar{r}_p$.

Similarly, explicit ratings can be affected by a systematic tendency for some items to receive higher ratings than others [Bell and Koren 2007]. This bias can be modeled through the following pre–normalization, which subtracts the item average rating from each rating, i.e., :

$$\tilde{r}_{pi} = r_{pi} - \bar{r}_i^{'} \tag{18}$$

where $\bar{r}_i^{'}$ indicates the average rating for item i. Again, the model is computed by factorizing the normalized URM by means of (10), while the real-time recommendations are obtained by means of (17), then summing up the item-i mean rating

$\bar{r}_i^{'}$.

Moreover, a global rating pre-normalization can be applied in order to overcome global rating attitudes. This normalization is performed by subtracting a constant value G from the ratings:

$$\tilde{r}_{pi} = r_{pi} - \mathrm{G} \tag{19}$$

Similarly to the previous approaches, the SVD factorization (10) is applied to the unbiased URM, and the real-time recommendations are given by (17), whose values are summed up G.

For instance, consider G equals to the average value $\bar{\mathrm{G}}$ computed over all the ratings in the dataset: in Netflix $\bar{\mathrm{G}}$ is approximately 3.6. The interpretation of the global rating normalization is that a rating greater than $\bar{\mathrm{G}}$ represents user interest (since it is greater than the average rating given by the community), while a rating lower than $\bar{\mathrm{G}}$ indicates negative opinion. Observe that the average global rating $\bar{\mathrm{G}}$ is the simplest rating estimator which minimizes the metric RMSE defined in (9), where, for each pair (p, i), $\hat{r}_{pi} = \bar{\mathrm{G}}$.

The presence of biases in the user ratings is considered by Bell and Koren in [Bell and Koren 2007], where they propose to alleviate what they call *global effects* by applying the aforementioned pre–normalizations in a specific sequence: they first remove the average global rating, then the item mean rating and finally the user mean rating.

4.2.2 *Implicit ratings.* In the case of implicit URM, the model can be directly computed by factorizing the binary URM by means of (10), and the real-time top-N recommendation list can be calculated by using (15).

5. EVALUATION METHODOLOGY

In this section, we first describe the possible methodologies and metrics used, respectively, to partition datasets and to evaluate recommender algorithms. Then, we propose a new evaluation methodology which is used within this work, and, finally, we describe the statistical characteristics of the datasets used in the tests.

5.1 Dataset partitioning

In order to evaluate the capability of a recommender system in suggesting items to users, we have to partition the URM into two different sets: the training set and the test set. The *training set* is used to generate the model of the system and it can be further divided into a model and a validation set for tuning some parameters of the algorithm. The *test set* is used to test the model generated in the training step.

There are different techniques that can be used to partition the URM: M-fold cross validation, leave-one-out and holdout.

The M-*fold cross validation* divides the users of the URM into M distinct partitions: at each step $\mathrm{M} - 1$ partitions are used to generate the model and the remaining partition is used for the tests. In literature, ([Duda et al. 2000] and [Witten and Frank 2005]) the number of partitions suggested to have a robust test is 10. In Figure 4 we can see an example of this partitioning technique. By means of this method, the tested users are unknown to the system, because they are not used to build the model. This aspect is very important, especially in the evaluation of model-based algorithms.

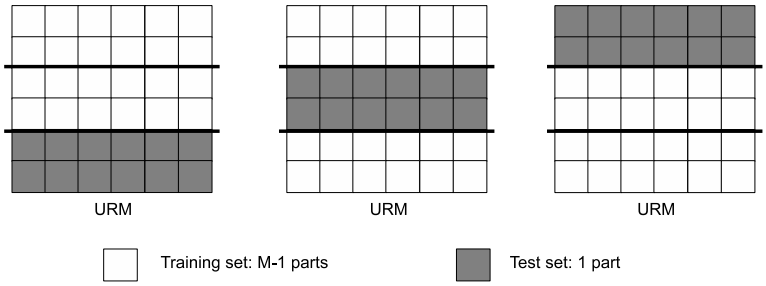


Fig. 4. Example of M-fold partitioning.

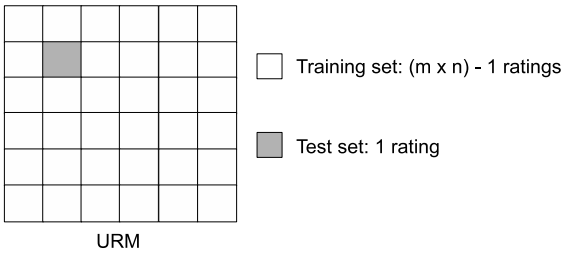


Fig. 5. Example of leave-one-out partitioning.

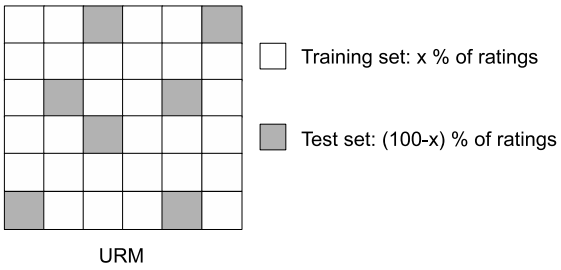


Fig. 6. Example of holdout partitioning.

With *leave-one-out*, during the construction of the model we withhold one non-null user rating of the URM (i.e., we set it to 0), while the remaining ratings are used to generate the model. The withheld rating represents the test set (Figure 5). This procedure is repeated for all (or a sample of) the ratings. Leave-one-out is attractive because almost all the ratings are available to build the model. However it suffers from overfitting problems and it has a high computational complexity.

With the *holdout* technique a random set of ratings is withheld from the URM and it is used as test set. The remaining part of the ratings are used as training set. Figure 6 shows an example of holdout. Holdout can suffer from the same overfitting problems as leave-one-out, because the tested users are not totally unknown to the model. This is true especially if the test set is too small. Moreover, the holdout technique modifies the user profiles and this can lead to erroneous results. This

is particularly perceptible in the case of datasets with short user profiles (i.e., few ratings per user).

5.2 Evaluation metrics

There are two distinct classes of metrics which can be used to evaluate a recommender system: error metrics and classification metrics.

Error metrics evaluate the error made in the prediction of the rating given by user p for item i. Let r_{pi} be the true rating given by user p for item i and $\hat{r}_{pi}$ be the value predicted by the recommender algorithm. An error metric estimates the distance between r_{pi} and $\hat{r}_{pi}$.

For these reasons, error metrics can be applied only to explicit datasets, since in the implicit URM we do not have any information about the opinion of the users about the purchased items.

Among error metrics, the most used in the evaluation of recommender systems are *root mean squared error (RMSE)* [Bennett and Lanning 2007; Herlocker et al. 2004], already defined in (9), *mean squared error (MSE)* [Buczak et al. 2002] and *mean absolute error (MAE)* [Herlocker et al. 1999; Sarwar et al. 2000b; Sarwar et al. 2001], which are defined as follows:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{r}_{ij} - r_{ij})^2 \tag{20}$$

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|\hat{r}_{ij} - r_{ij}| \tag{21}$$

these metrics are easy to compute, but they can suffer from the *outlier problem* (especially MSE and RMSE), i.e., they are sensible to ratings with large prediction errors. With respect to MSE and RMSE, MAE has a more intuitive interpretation, since it represents the average distance between the true rating and the predicted one.

Even if these metrics are often adopted in the evaluation of recommender algorithms on explicit datasets, we do not use them in this paper. In fact, in order to have comparable results among the different datasets, we need to use a common evaluation metric for all the experiments. Since part of the tests are performed on implicit datasets, we can not use error metrics. Moreover, [Herlocker et al. 2004] suggests that error metrics are not meaningful in evaluating the top-N recommendation task, i.e., a context where we are interested in giving an ordered list of N attractive items to the users, rather than in predicting ratings with precision.

For these reasons we opt for information-retrieval classification metrics. In fact, classification accuracy metrics allow to measure the capacity of recommender systems in suggesting interesting or uninteresting items to the user [Herlocker et al. 1999; Sarwar et al. 2000b]. These metrics suit our task because we are not interested in predicting the exact value of a rating, but, rather, we want to understand if a suggested item will be interesting for the user.

According to information retrieval, items in the URM can be classified either as relevant or irrelevant:

—an item is marked as relevant (or interesting) to the user if we know for certain

that the user likes that item (e.g., if the user rated the item with 5-out-of-5 stars);
—an item is marked as irrelevant (or uninteresting) to the user if we know for certain that the user does not like that item (e.g., if the user rated the item with 1-out-of-5 star).

According to classification accuracy metrics, a recommended item can be classified as:

—*true positive (TP)*: the system suggests a relevant item to the user.
—*true negative (TN)*: the system does not suggest an item that the user dislikes.
—*false positive (FP)*: the system suggests an item that the user dislikes.
—*false negative (FN)*: the system does not suggest an item interesting for the user.

The most popular classification accuracy metrics [Sarwar et al. 2000a; 2000b; Papagelis and Plexousakis 2005] are *recall* (T), *fallout* (F) and *precision* (P). They are generally defined as:

$$\mathrm{T} = \frac{\mathrm{TP}}{\mathrm{TP+FN}} \tag{22}$$

$$\mathrm{F} = \frac{\mathrm{FP}}{\mathrm{TN+FP}} \tag{23}$$

$$\mathrm{P} = \frac{\mathrm{TP}}{\mathrm{N}} \tag{24}$$

where N is the number of recommended items. The recall, also referred to as *true positive rate*, evaluates the capability of the system in recommending interesting items to the user. On the other side, fallout, also referred to as *false positive rate*, is a metric complementary to recall and measures how frequently the recommender system suggests non-interesting items to the user. In the evaluation of the global behavior of a recommender algorithm, both these metrics should be used.

Generally speaking, both precision and recall heavily depend on the number of rated items per user and, thus, their values should not be interpreted as absolute measures, but only to compare different algorithms on the same dataset. Measuring recall and precision is difficult because it is often unknown, for a certain user, how many relevant items there exist in the catalogue. As earlier noticed, items must be considered either relevant or irrelevant when calculating precision and recall for a user. Obviously, there exist items which can not be properly defined neither as relevant nor irrelevant because the user has not rated them.

Any catalogue probably contains a large number of items that meet a specific user taste. If that user has rated only a small percentage of such items, a recommender algorithm might appear to have a low recall, since the system may recommend unrated, relevant items (which are considered as irrelevant).

Moreover, consider one of the dataset partitioning methodology illustrated in Section 5.1. If the number of items withheld from the user profile (i.e., the test set) is much smaller than the number of recommended items, the precision for that user can not achieve high values. In the extreme case of leave-one-out applied to the top-N recommendation task, the maximum value of precision is bounded by 1/N, where N is again the number of recommended items.

This bound arises because the traditional definition of precision, applied in the settings of recommender algorithms, considers unrated items as non-interesting items. We can better understand the problem by looking at the details of the evaluation process. Let us imagine we are testing a recommender algorithm. We select one user from the URM and we withhold from his profile some of the ratings. We use the algorithm to recommend N items to that user. Suppose that some of the N recommended items are not in the set of withheld ratings. The question we face with is how to estimate the relevance of these items. Of course, that user is the only person who could determine if an item with no ratings meets his interests.

Notice that the previous definition of precision assumes that all recommended items for which we have no rating information are to be considered as irrelevant (e.g., we assume that the user would not like such items) and they count negatively in computing the precision. Indeed, such definition provides a too pessimist estimation of the real precision achieved by the algorithm.

One way to reduce the error in the estimate of the precision is to increase the number of ratings withheld from the user profile. However, this implies reducing the number of ratings used to describe a user, which negatively affects the quality of the recommender algorithms.

A better way to compute the precision in the top-N recommendation task is by considering only the items rated by the users. Given a user, we partition all the items in the catalogue into two sets, A and B, representing the relevant and irrelevant items for that user, respectively. The set A of relevant items can be further partitioned into two subsets:

—the set A_R of items that are clearly relevant to the user (e.g., items rated with 5-out-of-5 stars);

—the set A_0 of relevant items that we do not know to be interesting for the user (e.g., because the user has not rated them).

Similarly, the set B of non-relevant items can be partitioned into:

—the set B_R of items that are clearly irrelevant to the user (e.g., items rated with 1-out-of-5 star);

—the set B_0 of non-relevant items, that we do not know to be uninteresting for the user.

Of course, the following conditions are valid:

$$A = A_R \cup A_0$$

$$B = B_R \cup B_0$$

$$A_R \cap A_0 = \emptyset$$

$$B_R \cap B_0 = \emptyset$$

Figure 7 graphically shows the sets A, B, A_R, B_R, A_0 and B_0, while Figure 8 represents TP, FP, TN and FN. We will propose in Section 5.3 a methodology to better identify items in categories A_R and B_R.

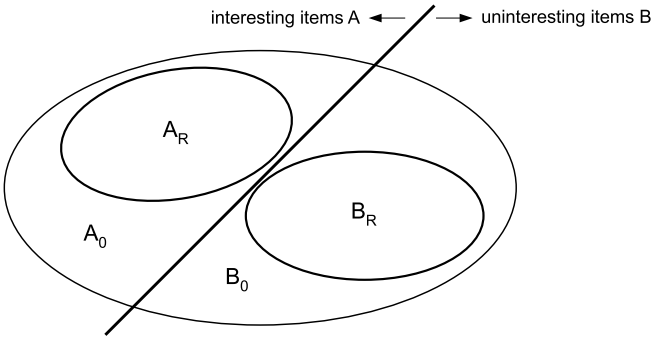


Fig. 7. A_R is the set of items that we know to be relevant for the users. B_R is the set of items that we know to be non-relevant for the users. A_0 and B_0 are, respectively, the sets of relevant and non-relevant items that we do not know to be interesting or non-interesting for the users.

Let us now go back to our scenario. The recommender algorithm generates a list of N items to recommend to the target user. Some of the recommended items are in the set A_R, i.e., we know they are relevant for the user. We classify these items as true-positive (TP). Some of the recommended items may be in the list B_R of items, i.e., we know they are uninteresting for the user. We classify these items as false-positive (FP). The remaining recommendations concern the set of items for which we are not able to state their relevance for the user: the number of such recommendations is $N - TP - FP$. The classical definition of precision can be rewritten as a function of the number of recommended items N:

$$P(N) = \frac{TP + (N - TP - FP) \cdot P(N - TP - FP)}{N} \tag{25}$$

where the supplementary term $(N - TP - FP) \cdot P(N - TP - FP)$ is meant to estimate the relevance of the unrated recommended items by using the precision of the algorithm when recommending a list of $N - TP - FP$ items (by definition, if an algorithm has a measured precision equals to P, then the user can expect that, on average, the P percent of the recommended items will be relevant). If we assume that P does not vary with the length of the recommended list, e.g., if we assume $P(N) = P(N - TP - FP) = P$, we obtain

$$P = \frac{TP}{TP + FP}. \tag{26}$$

The newly defined precision is a much closer approximation of the actual precision, because its definition considers also the unrated items.

We can see the effectiveness of the new definition of precision by using a simple example: a recommender system that randomly suggests one single item (i.e., $N = 1$) to the current user. In such a system, the probability that the recommender suggests an interesting items to the user is

$$P_{TRUE} = \frac{|A|}{|A| + |B|}$$

where $|\cdot|$ denote the cardinality of a set. P_{TRUE} represents the actual, unknown, precision of the recommender system. The measured precision, according to the

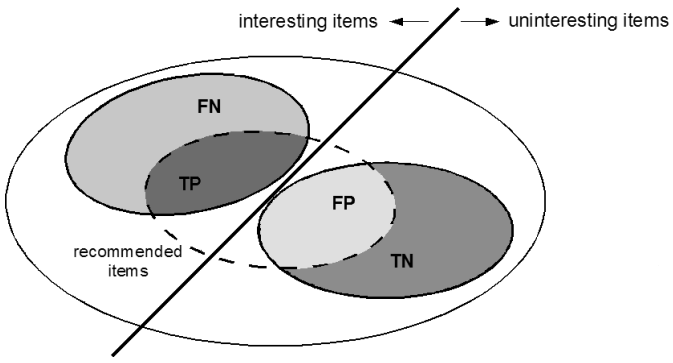


Fig. 8. Graphic representation of TP, FP, TN and FN; the dashed line encloses the recommended items.

standard definition (24), is

$$\mathrm{P}' = \frac{|\mathrm{A_R}|}{|\mathrm{A}| + |\mathrm{B}|}.$$

which now represents the probability of recommending an item that we know it is interesting for that user. According to the new definition (26), the same probability should be measured as:

$$\mathrm{P}'' = \frac{|\mathrm{A_R}|}{|\mathrm{A_R}| + |\mathrm{B_R}|}.$$

If we assume that the distribution of relevant and irrelevant items within the user's test set is the same as the true distribution for the user across all items[1], we have that:

$$\frac{|\mathrm{A_R}|}{\mathrm{A}} = \frac{|\mathrm{B_R}|}{\mathrm{B}}.$$

For this reason, by comparing P'' and $\mathrm{P_{TRUE}}$ we have that:

$$\mathrm{P}'' = \mathrm{P_{TRUE}}.$$

On the contrary, we have that:

$$\mathrm{P}' << \mathrm{P_{TRUE}}$$

because usually the number of positively-rated items is much smaller with respect to the number of interesting items in the catalogue, i.e.,

$$\mathrm{A_R} << \mathrm{A}.$$

In order to use only one metric which synthesizes the global behavior of the system, we can compute the F-measure$_a$ (F_α), as defined in [Powers 2000; Flach 2003]:

$$\mathrm{F}_\alpha = \frac{\mathrm{P} \cdot \mathrm{R} \cdot (1+\alpha)}{\alpha \cdot \mathrm{P} + \mathrm{R}}. \tag{27}$$

[1]This assumption does not take into account that users commonly avoid rating items that do not interest them.

The computation of (27) requires to compute recall and precision and to set the parameter α. Such parameter weights the importance of recall and precision in the evaluation of the global behavior of the recommender algorithm. Typically $\alpha = 1$, so that the F-measure is the harmonic mean of precision and recall. The precision defined in (26) can be calculated from recall and fallout as follows [Raghavan et al. 1989]:

$$\mathrm{P} = \frac{\mathrm{H} \cdot \mathrm{R}}{\mathrm{H} \cdot \mathrm{R} + (1 - \mathrm{H}) \cdot \mathrm{F}}, \tag{28}$$

where H is a constant value which depends on the dataset and it is defined as:

$$\mathrm{H} = \frac{|\mathrm{A_R}|}{|\mathrm{A_R} \cup \mathrm{B_R}|}. \tag{29}$$

5.3 Top-N evaluation methodology

As a consequence of the considerations highlighted in the previous section, in the following we propose an evaluation methodology suited to the top-N recommendation task. This methodology exploits the M-fold and leave-one-out partitioning techniques together.

According to the M-fold partitioning, users are split into M parts: $\mathrm{M} - 1$ parts represent the training set and are used to generate the model, while the remaining part represents the test set. This kind of partitioning guarantees that the tested users are kept out of the model. We evaluate the quality of recommendations using recall, fallout and F-measure.

As already seen, while recall measures the behavior of the system in recommending interesting items to the user, fallout measures the behavior on uninteresting items. We state that item i is interesting for user p, and it belongs to the set $\mathrm{A_R}$, if the following condition is satisfied:

$$A_{\mathrm{R}} = \{(p, i) | r_{pi} \geq 3 \wedge r_{pi} > \bar{r}_p\} \tag{30}$$

where $\bar{r}_p$ is again the mean of all ratings given by user p. The first constraint in (30) concerns the rating scale: a rating is positive if it is greater or equals to 3, i.e., the median value (assuming a 1-to-5 rating scale). The second constraint regards the user subjectivity bias, i.e., the tendency of users to personally interpret the rating scale: a rating is positive if greater than the user average value. Similarly, item i is classified as uninteresting for user p, and it belongs to the set $\mathrm{B_R}$, if r_{pi} satisfies the condition:

$$B_{\mathrm{R}} = \{(p, i) | r_{pi} < 3 \wedge r_{pi} \leq \bar{r}_p\} \tag{31}$$

Once created the model, the remaining partition is tested using a method similar to the leave-one-out approach (see Figure 9):

Step 1. We select one user in the test set, referred to as the *active user*.

Step 2. We remove one rated item, referred to as the *test item*, from the active user profile.

Step 3. Exploiting the model constructed in the training step, we generate an ordered list of recommended items.

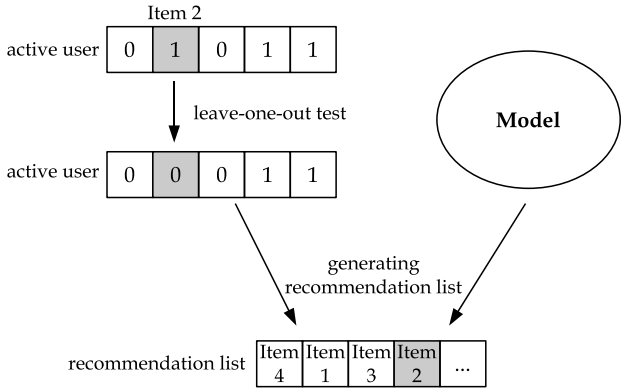


Fig. 9. Proposed testing technique.

Dataset	Users	Ratings	Items	Density	Average ratings x user	H
NM	26779	186K	771	0.90%	7	-
ML	6040	1M	3706	4.26%	165	0.7684
NFfull	480189	100M	17770	1.18%	209	0.7850
NF	47798	10M	17770	1.18%	209	0.7685

Table II. Statistical properties of the datasets.

Step 4. If the test item is in the top-N list of recommended items, then the recall or fallout value is updated, depending whether the user liked or disliked the test item, by using definition (23) or (24), respectively.

We repeat step 1 for all the users in the training set, and steps 2 through 4 for all the items rated by the active user. Precision and F-measure are computed using (26) and (27), respectively.

5.4 Datasets

The tests have been conducted on three different datasets:

—*NewMovies*, referred to as NM, a private implicit dataset made available by one of the most prominent IPTV providers in Europe.

—*MovieLens*, referred to as ML, a dataset largely used by researchers before the Netflix contest.

—*Netflix*, referred to as NFfull.

In MovieLens and Netflix ratings are in the range $[1, 5]$, while in MovieLens ratings are binary. Table II reports the statistical properties of the three datasets, where the density is the rate between the number of ratings and the size of the URM (i.e., the product between the number of users and the number of items), while H is the constant defined in (29). We can note that NM is quite different from the other two datasets: its density is lower and the average user profile is very short (only 7 ratings per user).

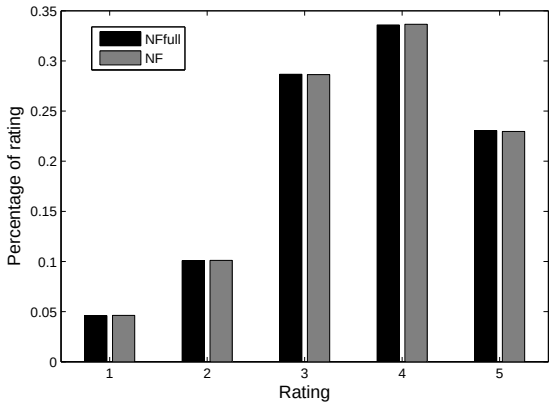


Fig. 10. Distribution of ratings in NFfull (original Netflix dataset) and NF (sampled Netflix dataset).

On the other hand, ML and NFfull have a larger number of ratings and, on average, richer user profiles (i.e., with more ratings per user) than NM. ML is the most dense dataset (4.26%) and every user has rated at least 20 movies. NFfull has a number of ratings 100 times greater than ML and user profiles with more ratings. We note that NFfull is an unusual dataset with respect to e-commerce settings, because, on average, each user has rated 209 items and there exist users with very long profiles, e.g., some users have rated up to 15000 items.

In order to speed up the tests for the NFfull dataset, we have randomly sampled one tenth of the NFfull users, obtaining a reduced version of the dataset, referred to as NF. Table II and Figure 10 show that NF preserves the statistical characteristics (same density, user profile length and rating distribution) of the original NFfull dataset.

6. RESULTS

In many e-commerce domains we only dispose of implicit datasets. In order to not excessively bore users, in fact, e-commerce applications avoid to ask user opinion about purchased items and collects only implicit information. Moreover, there exist recommender algorithms written specifically for binary datasets which are usually faster with respect to the explicit case (e.g., the DR algorithm proposed in this work and the biclustering technique proposed in [Symeonidis et al. 2008]).

In the following we want to show that by using an implicit dataset there is only a minimal worsening in the quality of recommendations with respect to the explicit case. In order to prove this assertion, we compare the recommender quality of several algorithms, tested with a common evaluation methodology on explicit and implicit datasets.

For these reasons we first need to convert the explicit datasets ML and NF into implicit datasets (this process is referred to as binarization), so that we have both implicit and explicit versions of the same datasets, obtaining comparable results.

Dataset	Algorithm	Latent size l	Recall	Fallout	Precision	F-measure$_1$
ML	SVD	50	0.1893	0.0325	0.9508	0.3158
	SVD - 3	50	0.1205	0.0056	0.9862	0.2148
	SVD - overall-mean	50	0.1102	0.0047	0.9873	0.1983
	SVD - user-mean	50	0.1051	0.0061	0.9828	0.1899
	SVD - item-mean	50	0.0620	0.0061	0.9712	0.1165
MLbase	SVD	50	0.1760	0.0585	0.8549	0.2919
MLpos	SVD	50	0.1232	0.0131	0.9485	0.2181

Table III. Results of the SVD-based algorithms applied to the explicit and implicit versions of ML.

6.1 Dataset binarization

Concerning the dataset binarization, we follow two different procedures, namely the *base binarization* and the *positive binarization*.

The *base binarization* considers all the ratings in the explicit dataset, regardless of their value and turns them into a 1 in the implicit version of the URM. This is a very simple binarization technique but, at the same time, it reflects accurately the characteristics of a real implicit dataset. In fact, in an implicit dataset such as NM, the ratings are collected regardless of the actual interest of users about items, simply because such interest is unknown. This binarization has been used by Deshpande and Karypis in [Deshpande and Karypis 2004]. We refer to the implicit versions of ML and NF obtained by means of the base binarization as MLbase and NFbase, respectively.

The *positive binarization* technique creates a new implicit URM by taking into consideration only the items interesting to the users, i.e., the positively-rated items. In the binarized URM, a rating is set to 1 whether it satisfies (30), 0 otherwise. We refer to the binarized versions of ML and NF by means of this technique as, respectively, MLpos and NFpos. Observe that, on the Netflix dataset, this binarization strategy removes about 50% of the ratings, considerably reducing the memory and time requirements of challenging algorithms such as the SVD-based algorithms.

6.2 Validation

In this section we present the results of the tests relative to ML and NF, in the explicit and implicit cases, obtained by means of the methodology described in Section 5.3. We present the results in terms of recall, fallout, precision and F-measure, as defined in (22), (23), (28) and (27), respectively. For each dataset, in order to produce comparable results, we select the set of ratings, commonly used in all the tests, as follows:

—The ratings satisfying (30) (i.e., positively-rated ratings) are used in (22) to compute the recall.

—The ratings satisfying (31) (i.e., negatively-rated ratings) are used in (23) to compute the fallout.

The values reported in this section are the average among the M values obtained by applying the M-fold test. Concerning the SVD-based algorithm, we show the

Dataset	Algorithm	Neighborhood size k	Recall	Fallout	Precision	F-measure$_1$
ML	Cosine similarity	-	0.1262	0.0283	0.8973	0.2213
		100	0.1603	0.0363	0.8964	0.2719
	Adjusted cosine	-	0.0004	0	1	0.0008
MLbase	Cosine similarity	-	0.1184	0.0361	0.8653	0.2083
		200	0.1475	0.0482	0.8570	0.2517
MLpos	Cosine similarity		0.0916	0.0096	0.9492	0.1671
		50	0.1090	0.0111	0.9506	0.1956

Table IV. Results of the item-based algorithms applied to the explicit and implicit versions of ML.

Dataset	Algorithm	Latent size l	Recall	Fallout	Precision	F-measure$_1$
NF	SVD	50	0.1658	0.0310	0.9467	0.2822
	SVD - 3	100	0.1112	0.0043	0.9885	0.1999
	SVD - overall-mean	50	0.0998	0.0051	0.9848	0.1812
	SVD - user-mean	50	0.0983	0.0060	0.9819	0.1787
	SVD - item-mean	50	0.0729	0.0091	0.9638	0.1355
NFbase	SVD	100	0.1487	0.0563	0.8976	0.2551
NFpos	SVD	100	0.0996	0.0078	0.9770	0.1808

Table V. Results of the SVD-based algorithms applied to the explicit and implicit versions of NF.

best value among the tests executed with different latent sizes l. In the case of explicit datasets, the tests on the SVD-based algorithm have been conducted: (i) without any normalizations (referred to as *SVD*), (ii) with the global rating pre–normalization defined in (19) where $\mathrm{G} = 3$ (referred to as *SVD-3*) and (iii) $\mathrm{G} = \bar{\mathrm{G}}$ (referred to as *SVD-overall-mean*), (iv) with the user pre–normalization formulated in (4) (referred to as *SVD-user-mean*), and (v) with the item pre–normalization formalized in (18) (referred to as *SVD-item-mean*).

Regarding the item-based algorithms, the results mainly address the cosine similarity metric, since it is applicable both on explicit and implicit datasets. We report the best value among the tests executed with different neighborhood sizes k. For comparison, we also show the results obtained by using the cosine similarity without limiting the neighborhood size (i.e., $k = \infty$), and, in the explicit case, the results obtained by using the adjusted cosine similarity, which, however, leads to a very low quality of recommendations. Optimal latent size (SVD-based algorithm) and optimal neighborhood size (item-based algorithm) have been selected by means of cross validation on the training sets.

In order to consider the capability of the algorithms in recommending interesting items and discarding uninteresting items, Tables III through VI show recall, fallout, precision and F-measure$_1$. Focusing on the F-measure, which synthesizes the global behavior of the recommender system, we observe that the base binarization offers better recommendations with respect to the positive one, both on ML and NF, and the results are comparable with the related explicit versions.

Moreover, these results give a different indication about the algorithm which offers the best quality. On the ML dataset, in fact, both on the explicit and

Dataset	Algorithm	Neighborhood size k	Recall	Fallout	Precision	F-measure$_1$
NF	Cosine similarity	-	0.0856	0.0158	0.9070	0.1564
		20	0.1406	0.0355	0.8770	0.2423
	Adjusted cosine	-	0.0033	0	1	0.0066
NFbase	Cosine similarity	-	0.0752	0.0197	0.8729	0.1385
		20	0.1253	0.0586	0.7937	0.2164
NFpos	Cosine similarity	-	0.0678	0.0056	0.9561	0.1266
		10	0.1064	0.0098	0.9513	0.1914

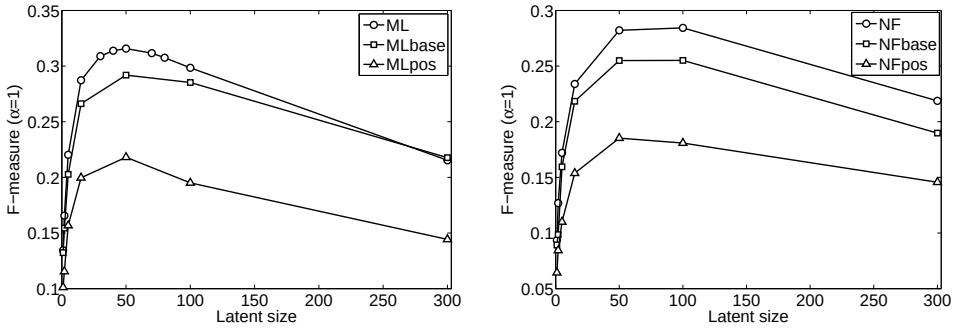
Table VI. Results of the item-based algorithms applied to the explicit and implicit versions of NF.

implicit cases, the best algorithm, according to the F-measure metric, is always the SVD with $l = 50$. On the NF dataset, instead, we have a different behavior in the explicit and implicit cases. In the NF and NFbase cases, the SVD with $l = 50$ and $l = 100$, respectively, are better than the other algorithms. In the NFpos case, instead, the best algorithm is the cosine similarity with $k = 10$. As we can see in Tables III and V, the SVD algorithm always offers better results than the SVD-based with the normalizations. However, note that the normalizations drastically reduce the fallout, with respect to the pure SVD-based. Therefore, they are effective in reducing the number of wrong recommendations.

Observing the results relative to the positive binarization, we note that, on the whole, it always offers a lower quality with respect to the base one, in terms of F-measure. Furthermore, despite the positive binarization leads to a lower recall than the base one, this is compensated by a fallout definitely lower. This is a very important aspect, mainly in a context where it is fundamental to not suggest items uninteresting to users. A high fallout, in fact, means a large percentage of uninteresting items suggested to users. This is a key parameter for evaluating a recommender system, because even if the algorithm had a high recall, a high fallout would make the user attenuate his trust in the received recommendations.

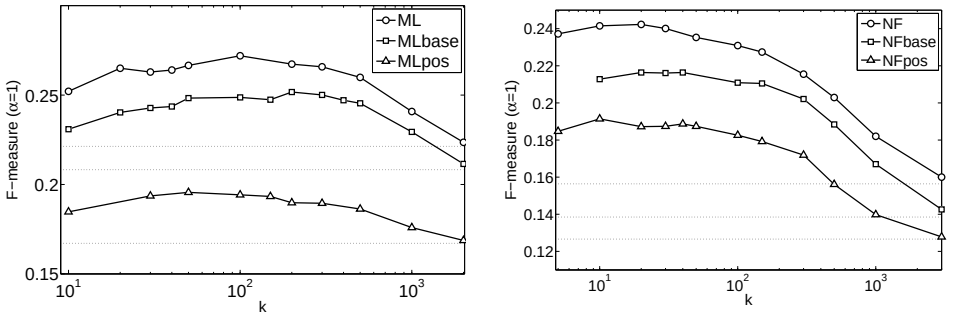
Figure 11 shows how the latent size affects the quality of the SVD algorithm on both the explicit and implicit versions of ML and NF. The metric is the F-measure with $\alpha = 1$. As we can see, the optimal latent size is not affected by the type of dataset (i.e., implicit or explicit). For instance, the optimal latent size for ML is $l = 50$, both in the explicit and implicit versions, and regardless of the binarization approach. Similarly, the optimal latent size for NF is in the range $50 - 100$.

Figure 12 shows the same analysis applied to the item-based algorithm with cosine similarity. The two graphs show the F-measure of the algorithm when varying the neighborhood size k. As a reference, the dashed lines report the asymptotic quality of the algorithms, i.e., when $k = \infty$. Again, the optimal value of the parameter (i.e., k) does not depend strongly on the kind of dataset (explicit or implicit), neither on the binarization strategy. Looking at the NF dataset, the optimal neighborhood size is in the range $10 - 30$, while for the ML dataset the optimal neighborhood size is in the range $50 - 200$.



(a) ML (b) NF

Fig. 11. Trend of the SVD algorithm on the explicit and implicit versions of ML (a) and NF (b) as a function of the latent size l.



(a) ML (b) NF

Fig. 12. Trend of the item-based algorithm with cosine similarity on the explicit and implicit versions of ML (a) and NF (b) as a function of the neighborhood size k.

6.3 NewMovies Results

In the previous section we have shown that the base binarization offers a quality close to the explicit case, and this is evident, for instance, in Figures 11 and 12. As a consequence, the scenario depicted in Figure 2 can be replaced by the one shown in Figure 13, where the explicit ratings are converted into implicit ratings, without penalizing the quality of recommendations. In the following we consider only implicit datasets and, in particular, we extend the previous analysis by evaluating additional algorithms on the real implicit dataset NM. Remind that NM has not been obtained by means of binarization, but the collected ratings are implicit by nature. The base binarization used with NF and ML emulates the mechanism used to form the NM dataset: for each user we know the list of movies purchased in a three-month period, but we do not know if the user liked or not such movies. Note that, with the NM dataset, the only metric that we are able to calculate is the recall. In fact, in order to compute fallout, precision and F-measure, we should know which items the user dislikes.

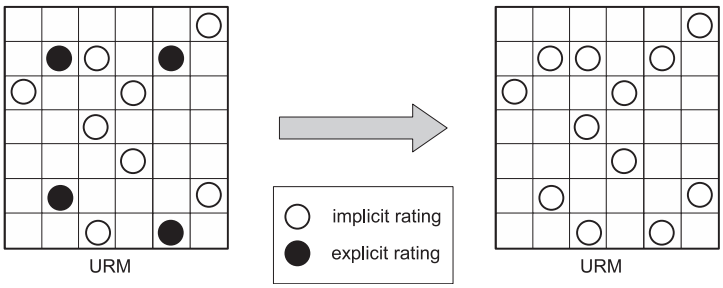


Fig. 13. Coexistence of implicit and explicit ratings. The explicit ratings are transformed into implicit ratings.

		β		
		0	0.5	1
γ	0	0.1444	0.1475	0.1527
	0.5	0.1548	0.1605	0.1305
	1	0.0064	0.0076	0.0079

Table VII. Recall of the item-based algorithms on NM, varying the parameters γ and β in (7).

Algorithm	Parameter	All
SVD	$l = 5$	0.1324
Cosine similarity	$k = 150$	0.1693

Table VIII. Comparison, in terms of recall, between the SVD-based and the item-based with cosine similarity algorithms. The dataset is NM.

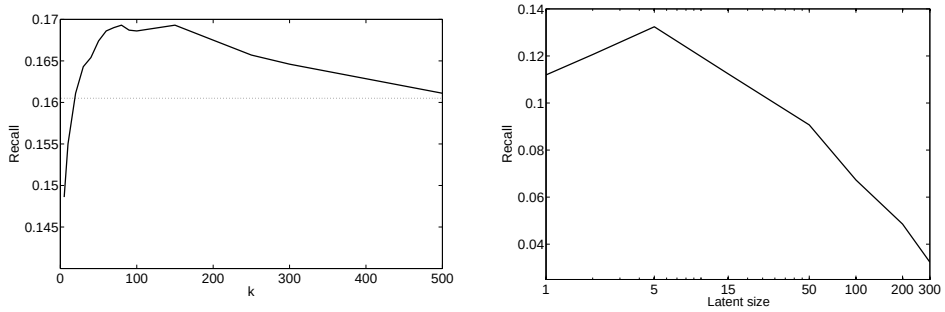
The tests are conducted on the implicit versions of the SVD-based and item-based algorithms, described in Section 4.2.2 and 4.1.2, respectively. The SVD-based algorithm depends on the latent size l, while the item-based algorithm is a function of the parameters γ and β, as follows from (7).

Table VII presents the recall of the item-based algorithms tested on the NM dataset, varying the values of γ and β in the range between 0 and 1. The results show that the best combination, in terms of recall, is given by $\gamma = 0.5$ and $\beta = 0.5$, which corresponds to the cosine similarity. Note that the worst performances regard the item-based algorithms with $\gamma = 1$, regardless of the β value. Given any value for β, the best recall always refers to $\gamma = 0.5$.

Since the cosine similarity reaches the highest recall among the different item-based algorithms, we evaluate the potentiality of such algorithm by applying the kNN approach. Figure 14(a) shows the recall as a function of the neighborhood size k. The dashed line is the cosine similarity without limiting the neighborhood size, i.e., with $k = \infty$. Note that the optimal value for the neighborhood size k is in the range between 50 and 150, leading to a recall equals to 0.1693. Where k tends to infinity, the recall slowly decreases tending to the value 0.1605.

Similarly, in Figure 14(b) we evaluate the recall of the SVD-based algorithm as a function of the latent size l. Observe that there is a maximum when $l = 5$.

Finally, Table VIII compares the best results on NM reached by the two algorithms, namely the SVD-based and the item-based with cosine similarity.



(a) Item-based (b) SVD

Fig. 14. Recall obtained on the dataset NM for (a) the item-based with cosine similarity algorithm as a function of the neighborhood size k, and (b) the SVD algorithm as a function of the latent size l.

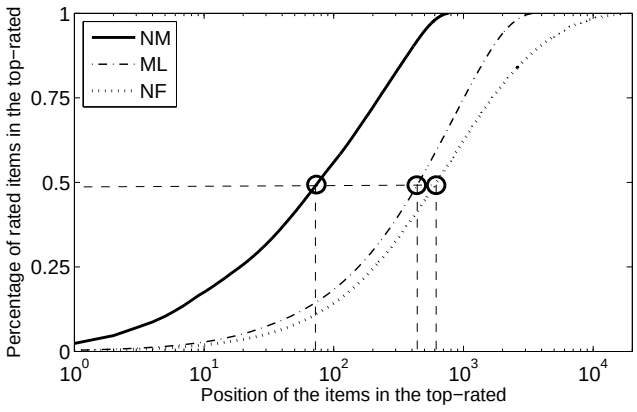


Fig. 15. Distribution of items in the top-rated list of NM, ML and NF.

6.4 Analysis of the recommendation of popular and unpopular items

In this section we keep on considering implicit datasets and we analyze the quality of the SVD-based algorithm in suggesting different classes of items, characterized by different popularities, in order to catch biases of the algorithm toward popular items. The analysis is motivated by the behavior of the algorithm when varying the latent size l. In the extreme case of $l = 1$, the algorithm roughly corresponds to recommend the top-rated list of items to all the users, regardless of the user profile.

We indicate with *top-rated list* the list of items sorted from the most popular to the least popular, according to the number of users who rated them. For instance, the optimal recommendation list for a *blockbuster* user (i.e., a user who tends to prefer popular items) is probably formed by taking the top elements from the top-rated list.

The analysis is conducted by splitting the items into *popular* and *unpopular*. In particular, we refer to popular items as the first t items in the top-rated list,

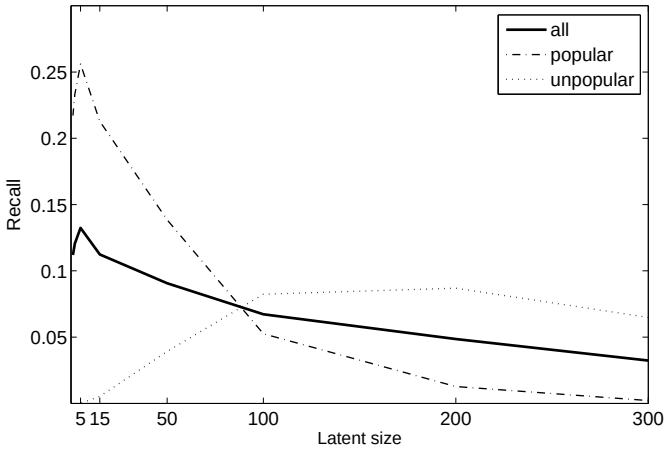


Fig. 16. Recall of the SVD-based algorithm as a function of the latent size l. The tests differentiate between popular and unpopular items and refer to the implicit dataset NM.

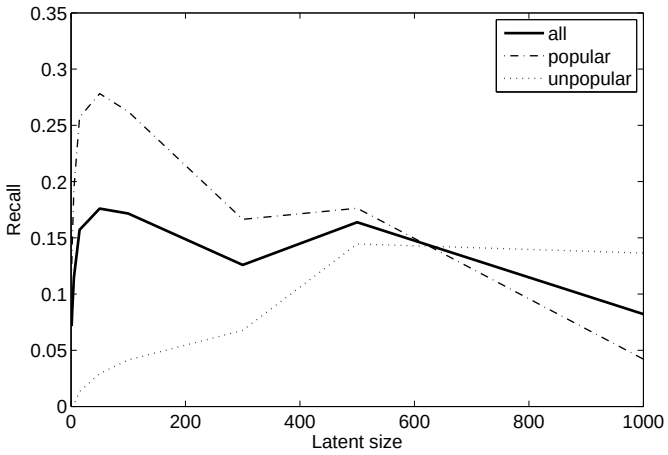


Fig. 17. Recall of the SVD-based algorithm as a function of the latent size l. The tests differentiate between popular and unpopular items and refer to the implicit dataset ML.

and to unpopular items as the remaining part of the list. In order to choose the threshold t which distinguishes between popular and unpopular items, we analyze the cumulative distribution of the ratings in the top-rated list. Figure 15 shows, on the x-axis, the position of the items in the top-rated list, and, on the y-axis, the percentage of ratings for the top-t items. The value of t is choosen such as it corresponds to the 50-th percentile of such distribution. We have $t = 80$ for NM (Figure 15), which indicates that 50% of the ratings in the NM dataset refer to the first 80 movies in the top-rated, $t = 450$ for ML and $t = 600$ for NF.

We evaluate the SVD-based algorithm initially separating between popular and unpopular items, then comparing the results with those obtained on all the items

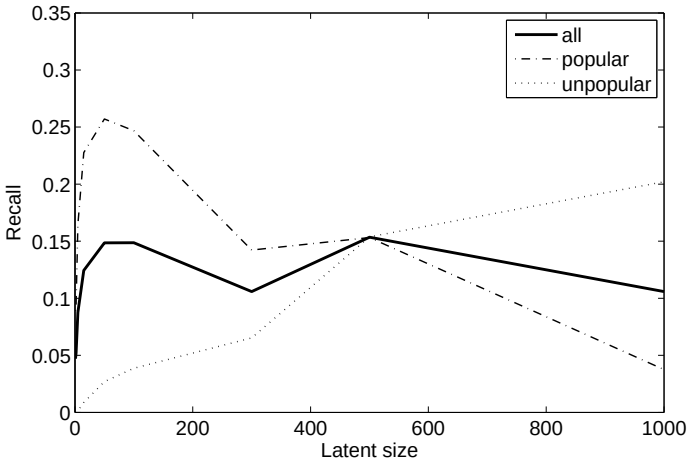


Fig. 18. Recall of the SVD-based algorithm as a function of the latent size l. The tests differentiate between popular and unpopular items and refer to the implicit dataset NF.

(with no partitioning). Figures 16, 17 and 18 show the trends of the SVD-based algorithm in terms of recall in recommending popular or unpopular items on NM, MLbase and NFbase, respectively, as a function of the latent size. We observe that, in all the cases, the recall is higher for popular items than for unpopular items, confirming the bias toward popular items; the bold curve, referred to as 'all', corresponds to the test performed on all the items and it results in the middle between the two curves related to popular and unpopular items. The three curves joint in a common point, which corresponds to a reversal of the trend. Beyond this value of latent size, in fact, the recall is higher for the unpopular items than for popular items.

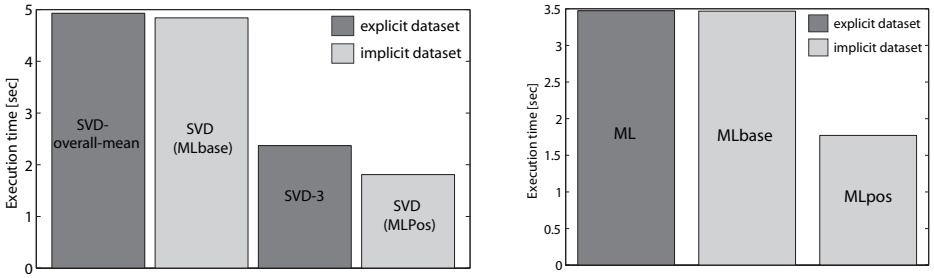
The main result from this last analysis is that the optimal latent size depends on the popularity of the recommended items. We can observe that, in order to recommend popular items, we should use low value for the latent size: $l = 5$ with NM, and $l = 50$ with ML and NF. On the contrary, in order to properly recommend unpopular items, we need higher values of the latent size: $l = 200$ for NM, $l = 500$ for ML and $l = 1000$ for NF.

6.5 Performance

In this section we analyze the performance of the algorithms in terms of the time required to generate the model. For the SVD-based algorithms, building the model means decomposing the URM into three matrices, as shown in Section 4.2, while for item-based algorithms it means computing the item-item similarity matrix.

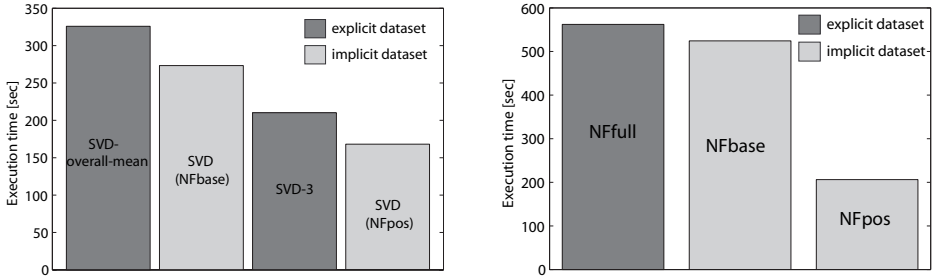
We used Matlab for the item-based algorithms, while the Lanczos-based implementation of the SVD [Berry 1992] for the dimensionality reduction-based algorithms. The tests have been executed on a HP DL 360 quad-core, with 2GHz clock speed and 4 GB RAM, running Linux 64-bit kernel.

Figures 19 and 20 show the execution time, expressed in seconds, of the SVD-



(a) SVD-based (b) Item-based

Fig. 19. Execution time of the SVD-based (a) and the item-based (b) algorithms on ML.



(a) SVD-based (b) Item-based

Fig. 20. Execution time of the SVD-based (a) and the item-based (b) algorithms on NFfull.

based (a) and the item-based (b) algorithms executed on ML and NFfull, respectively, with the original explicit version of the dataset and with the two binarized versions (base and positive). The performance of the SVD-based algorithm are evaluated setting the latent size to 50 and considering the data normalization (19), once with G equals to $\bar{\mathrm{G}}$ (referred to as SVD-overall-mean), and once with $\mathrm{G} = 3$ (referred to as SVD-3). The times reported in Figure 20 refer to the original full Netflix dataset.

These results confirm that the binarization can speed-up the execution time of the recommender algorithms up to about 50%.

7. CONCLUSIONS

In conclusion, we have compared explicit and implicit algorithms, showing that the simple implicit information allows to reach a quality of recommendations very close to the explicit case, with the advantage of not annoying the users with questions about their preferences. We have shown that the base binarization reaches better global results, in terms of F-measure, than the positive binarization, but the latter has the great advantage of rarely suggesting uninteresting items (i.e., it has a low fallout).

We have also seen that dimensionality reduction-based algorithms generally offer better recommendations with respect to item-based algorithms. Besides, among

the item-based techniques, the cosine similarity with the kNN approach reached the highest results.

Finally, the analysis aimed to study the behavior of the SVD-based algorithm in recommending popular and unpopular items showed that by varying the latent size we can tune the capability of the recommender system in suggesting popular or unpopular items.

Ongoing studies are exploiting such interesting analysis, for instance, by considering the characteristics of user profiles in order to determine the optimal latent size to use for a specific user. Further works are meant to extend the proposed tests to other algorithms both on implicit datasets and on binarized versions of explicit datasets.

REFERENCES

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

Anderson, C. 2006. *The Long Tail: Why the Future of Business Is Selling Less of More.* Hyperion.

Balabanović, M. and Shoham, Y. 1997. Fab: content-based, collaborative recommendation. *Commun. ACM 40,* 3, 66–72.

Belkin, N. J. and Croft, W. B. 1992. Information filtering and information retrieval: two sides of the same coin? *Commun. ACM 35,* 12, 29–38.

Bell, R. M. and Koren, Y. 2007. Scalable collaborative filtering with jointly derived neighborhood interpolation weights. *ICDM 2007. Seventh IEEE International Conference on Data Mining*, 43–52.

Bennett, J. and Lanning, S. 2007. The Netflix Prize. *Proceedings of KDD Cup and Workshop*, 3–6.

Berry, M. W. 1992. Large-scale sparse singular value computations. *The International Journal of Supercomputer Applications 6,* 1 (Spring), 13–49.

Buczak, A., Zimmerman, J., and Kurapati, K. 2002. Personalization: Improving Ease-of-Use, Trust and Accuracy of a TV Show Recommender. *Proceedings of the AH 2002 Workshop on Personalization in Future TV*.

Crammer, K. and Singer, Y. 2001. Pranking with ranking. *In Proceedings of the conference on Neural Information Processing Systems (NIPS), 2001.*.

Deerwester, S. C., Dumais, S. T., Landauer, T. K., Furnas, G. W., and Harshman, R. A. 1990. Indexing by latent semantic analysis. *Journal of the American Society of Information Science 41,* 6, 391–407.

Deshpande, M. and Karypis, G. 2004. Item-based top-n recommendation algorithms. *ACM Transactions on Information Systems (TOIS) 22,* 1, 143–177.

Duda, R., Hart, P., and Stork, D. 2000. *Pattern Classification.* Wiley-Interscience.

Flach, P. 2003. The geometry of ROC space: Understanding machine learning metrics through ROC isometrics. *Proceedings of the Twenntieth International Conference on Machine Learning*, 194–201.

Fung, G., Rosales, R., and Krishnapuram, B. 2006. Learning rankings via convex hull separation. *In Advances in Neural Information Processing Systems 18.*.

Furnas, G. W., Deerwester, S., Dumais, S. T., Landauer, T. K., Harshman, R. A., Streeter, L. A., and Lochbaum, K. E. 1988. Information retrieval using a singular value decomposition model of latent semantic structure. *SIGIR '88: Proceedings of the 11th annual international ACM SIGIR conference on Research and development in information retrieval*, 465–480.

Goldberg, D., Nichols, D., Oki, B. M., and Terry, D. 1992. Using collaborative filtering to weave an information tapestry. *Commun. ACM 35,* 12, 61–70.

Gorrell, G. 2006. Generalized Hebbian Algorithm for Incremental Singular Value Decomposition in Natural Language Processing. *11th Conference of the European Chapter of the Association for Compuational Linguistics*.

Herlocker, J., Konstan, J., and Riedl, J. 1999. An algorithmic framework for performing collaborative filtering. *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, 230–237.

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

Husbands, P., Simon, H., and Ding, C. 2000. On the use of singular value decomposition for text retrieval. M. Berry, Ed. *Proc. of SIAM Comp. Info. Retrieval Workshop*.

Kitts, B., Freed, D., and Vrieze, M. 2000. Cross-sell: a fast promotion-tunable customer-item recommendation method based on conditionally independent probabilities. *Knowledge Discovery and Data Mining*, 437–446.

Konstan, J. A., Miller, B. N., Maltz, D., Herlocker, J. L., Gordon, L. R., and Riedl, J. 1997. GroupLens: Applying collaborative filtering to Usenet news. *Communications of the ACM 40,* 3, 77–87.

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

Mooney, R. J. and Roy, L. 2000. Content-based book recommending using learning for text categorization. *Proceedings of DL-00, 5th ACM Conference on Digital Libraries*, 195–204.

Morita, M. and Shinoda, Y. 1994. Information filtering based on user behavior analysis and best match text retrieval. *SIGIR '94: Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, 272–281.

Nichols, D. 1998. Implicit rating and filtering. *Proceedings of 5th DELOS Workshop on Filtering and Collaborative Filtering*, 31–36.

Papagelis, M. and Plexousakis, D. 2005. Qualitative analysis of user-based and item-based prediction algorithms for recommendation agents. *Engineering Applications of Artificial Intelligence 18,* 7, 781–789.

Paterek, A. 2007. Improving regularized singular value decomposition for collaborative filtering. *Proceedings of KDD Cup and Workshop*.

Pazzani, M. and Billsus, D. 2006. Content-based recommendation systems. *The Adaptive Web: Methods and Strategies of Web Personalization, Lecture Notes in Computer Science*, 325–341.

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

Powers, D. M. W. 2000. Evaluation: From Precision, Recall and F-Factor to ROC, Informedness, Markedness and Correlation.

Raghavan, V., Bollmann, P., and Jung, G. S. 1989. A critical investigation of recall and precision as measures of retrieval system performance. *ACM Transaction on Information Systems 7,* 3, 205–229.

Resnick, P., Iacovou, N., Suchak, M., Bergstorm, P., and Riedl, J. 1994. GroupLens: An Open Architecture for Collaborative Filtering of Netnews. 175–186.

Saad, Y. 1992. *Numerical methods for large eigenvalue problems*. Halsted Press New York.

Salton, G., Ed. 1988. *Automatic text processing*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Sarwar, B., Karypis, G., Konstan, J., and Reidl, J. 2001. Item-based collaborative filtering recommendation algorithms. *Proceedings of the 10th international conference on World Wide Web*, 285–295.

Sarwar, B., Karypis, G., Konstan, J., and Riedl, J. 2000a. Analysis of Recommendation Algorithms for E-Commerce. *Proceedings of the 2nd ACM conference on Electronic commerce*, 158–167.

Sarwar, B., Karypis, G., Konstan, J., and Riedl, J. 2000b. *Application of Dimensionality Reduction in Recommender System-A Case Study*. Defense Technical Information Center.

Sarwar, B. M. 2001. Sparsity, scalability, and distribution in recommender systems. Ph.D. thesis. Adviser-John T. Riedl.

Schafer, J. B., Konstan, J. A., and Riedi, J. 1999. Recommender systems in e-commerce. *ACM Conference on Electronic Commerce*, 158–166.

Shardanand, U. and Maes, P. 1995. Social information filtering: Algorithms for automating "word of mouth". *Proceedings of ACM CHI'95 Conference on Human Factors in Computing Systems 1*, 210–217.

Symeonidis, P., Nanopoulos, A., Papadopoulos, A. N., and Manolopoulos, Y. 2008. Nearest-biclusters collaborative filtering based on constant and coherent values. *Information Retrieval 11,* 1, 51–75.

Wang, J., de Vries, A. P., and Reinders, M. J. T. 2006. Unifying user-based and item-based collaborative filtering approaches by similarity fusion. *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, 501–508.

Witten, I. and Frank, E. 2005. *Data Mining: Practical Machine Learning Tools and Techniques.* Morgan Kaufmann, San Francisco.

Zhang, X., Berry, M. W., and Raghavan, P. 2001. Level search schemes for information filtering and retrieval. *Information Processing and Management 37,* 2, 313–334.

...