An Evaluation Methodology for Collaborative Recommender Systems

Paolo Cremonesi, Roberto Turrin
Politecnico di Milano, Italy
Neptuny, Italy

Eugenio Lentini, Matteo Matteucci
Politecnico di Milano, Italy

Abstract

Recommender systems use statistical and knowledge discovery techniques in order to recommend products to users and to mitigate the problem of information overload. The evaluation of the quality of recommender systems has become an important issue for choosing the best learning algorithms. In this paper we propose an evaluation methodology for collaborative filtering (CF) algorithms. This methodology carries out a clear, guided and repeatable evaluation of a CF algorithm. We apply the methodology on two datasets, with different characteristics, using two CF algorithms: singular value decomposition and naive bayesian networks[1]

1. Introduction

Recommender systems (RS) are a filtering and retrieval technique developed to alleviate the problem of information and products overload. Usually the input of a RS is a $m \times n$ matrix called *user rating matrix (URM)* where rows represent the users $U = \{u_1, u_2, \ldots, u_m\}$ and columns represent the items $I = \{i_1, i_2, \ldots, i_n\}$ (e.g., CDs, movies, etc.). Each user u has expressed his opinion about some items by means of a *rating score* r according to the *rating scale* of the system. A rating scale could be either *binary*, that is 1 stands for rated and 0 for not rated, or *not binary*, when the user can express his rating by different fixed values (e.g., $1 \ldots 5$) and 0 stands for not rated. The aim of a RS is to predict which items a user will find interesting or useful.

Collaborative filtering (CF) recommender systems recommend items that users similar to the active user (i.e. the user requiring a prediction) liked in the past. CF algorithms are the most attractive and commonly used techniques [14] and they are usually divided into *user-based* (similarity relations are computed among users) and *item-based* (similarity relations are computed among items) methods. According to [16, 15], item-based CF algorithms provide better performance and quality than user-based ones.

In this work we expound a methodology useful to evaluate CF algorithms. The remaining parts are organized as follows. Section 2 introduces the quality evaluation and the partitioning of a dataset. Section 2.1 presents methods useful to partitioning a dataset, highlighting pros and cons of each method. Section 2.2 describes metrics. Section 3 introduces the evaluation methodology. The application of the proposed methodology is shown in Section 5 using the algorithms and the datasets described in Section 4. Section 6 summarizes the contributions of this work and draws the directions for future researches.

2. Quality evaluation

When analyzing a recommendation algorithm we are interested in its future performance on new data, rather than in its performance on past data [20]. In order to test future performance and estimate the prediction error, we must properly partition the original dataset into *training* and *test* subsets. The training data are used by one or more learning methods to come up with the model (i.e., an entity that synthesizes the behavior of the data) and the test data are used to evaluate the quality of the model. The test set must be different and independent from the training set in order to obtain a reliable estimate of the true error.

Many works do not distinguish decisively and clearly between methods and metrics used for performance evaluation and model comparison. Moreover, they neither highlight a strict methodology to perform evaluation of algorithms nor use metrics to evaluate how well an algorithm performs under different points of view. The scope of this paper is to supplement the work done by Herlocker et al. [12] with suitable methods for dataset partitioning and an integrated methodology to evaluate different models on the same data.

[1] This work has been accomplished thanks to the technical and financial support of Neptuny.

paolo.cremonesi@polimi.it
roberto.turrin@neptuny.com
eugenio.lentini@mail.polimi.it
matteo.matteucci@polimi.it

2.1. Dataset partitioning

In order to estimate the *quality of a recommender system* we need to properly partition the dataset into a training set and a test set. It is very important that performance are estimated on data which take no part in the formulation of the model. Some learning schemes need also a validation set in order to optimize the model parameters. The dataset is usually split according to one among the following methods: i) holdout, ii) leave-one-out or iii) m-fold cross-validation.

Holdout is a method that splits a dataset into two parts: a training set and a test set. These sets could have different proportions. In the setting of recommender systems the partitioning is performed by randomly selecting some ratings from all (or some of) the users. The selected ratings constitute the test set, while the remaining ones are the training set. This method is also called *leave-k-out*. In [17], Sarwar et al. split the dataset into 80% training and 20% test data. In [18] several ratios among training and test (from 0.2 to 0.95 with an increment of 0.05) are chosen and for each one the experiment is repeated ten times with different training and test sets and finally the results are averaged. In [13] the test set is made by 10% of users: 5 ratings for each user in the test set are withheld.

Leave-one-out is a method obtained by setting $k = 1$ in the leave-k-out method. Given an active user, we withhold in turn one rated item. The learning algorithm is trained on the remaining data. The withheld element is used to evaluate the correctness of the prediction and the results of all evaluations are averaged in order to compute the final quality estimate. This method has some disadvantages, such as the overfitting and the high computational complexity. This technique is suitable to evaluate the recommending quality of the model for users who are already registered as members of the system. Karypis et al. [10] adopted a trivial version of the leave-one-out creating the test set by randomly selecting one of the non-zero entries for each user and the remaining entries for training. In [7], Breese et al. split the URM in training and test set and then, in the test set, withhold a single randomly selected rating for each user.

A simple variant of the holdout method is the *m-fold cross-validation*. It consists in dividing the dataset into *m* independent folds (so that folds do not overlap). In turn, each fold is used exactly once as test set and the remaining folds are used for training the model. According to [20] and [11], the suggested number of folds is 10. This technique is suitable to evaluate the recommending capability of the model when new users (i.e., users do not already belong to the model) join the system. By choosing a reasonable number of folds we can compute mean, variance and confidence interval.

2.2. Metrics

Several metrics have been proposed in order to evaluate the performance of the various models employed by a RS. Given an active user u, a model should be able to predict ratings for any unrated items. The pair $\langle p_i, a_i \rangle$ refers to the prediction on the *i*-th test instance and the corresponding actual value given by the active user. The metrics allow the evaluation of the quality of the numeric prediction.

Error metrics are suitable only for not binary datasets and measure how much the prediction p_i is close to the true numerical rating a_i expressed by the user. The evaluation can be done only for items that have been rated.

Mean Squared Error (MSE), adopted in [8], is an error metric defined as:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(p_i - a_i)^2. \tag{1}$$

MSE is very easy to compute, but it tends to exaggerate the effects of possible outliers, that is instances with a very large prediction error.

Root Mean Squared Error (RMSE), used in [3], is a variation of MSE:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(p_i - a_i)^2}, \tag{2}$$

where the square root give to MSE the same dimension as the predicted value itself. As MSE, RMSE squares the error before summing it and suffers of the same outliers problem [12].

Mean Absolute Error (MAE) is one of the most commonly used metric and it is defined as:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|p_i - a_i|. \tag{3}$$

This measure, unlike MSE, is less sensitive to outliers. According to [12] [18, 16] and [13], MAE is the most used metric because of its easy implementation and direct interpretation. However MAE is not always the best choice. According to Herlocker et al. [12], MAE and related metrics may be less meaningful for tasks such as *Finding Good Items* [19] and *top-N recommendation* [10], where a ranked result list of *N* items is returned to the user. Thus, the accuracy for the other items, which user will have no interest in, is unimportant. The top-*N* recommendation task is often adopted by e-commerce services where the space available on the graphic interface for listing recommendations is limited and users can see only the *N* ranked items. Thus MAE, and all the error metrics in general, are not meaningful for *classification* tasks.

Classification accuracy metrics allow to evaluate how effectively predictions help the active user in distinguishing good items from bad items [13, 18]. The choice of classification accuracy metrics is useful when we are not interested in the exact prediction value, but only in finding out if the active user will like or not the current item. Classification accuracy metrics are very suitable for domains with binary ratings and, in order to use these metrics, it is necessary to adopt a binary rating scheme. If a different rating scale is given, we must convert the not binary scale to a binary one by using a suitable threshold to decide which items are good and which are bad. For instance, with a rating scale in the range $(0, \ldots, 5)$, this threshold could be set arbitrarly to 4 according to common sense as in [13], or the threshold could be chosen by means of statistical considerations as in [12].

With classification metrics we can classify each recommendation such as: a) *true positive* (TP, an interesting item is recommended to the user), b) *true negative* (TN, an uninteresting item is not recommended to the user), c) *false negative* (FN, an interesting item is not recommended to the user), d) *false positive* (FP, an uninteresting item is recommended to the user).

Precision and recall are the most popular metrics in the information retrieval field. They have been adopted, among the others, by Sarwar et al. [17, 18] and Billsus and Pazzani [5]. According to [1], information retrieval applications are characterized by a large amount of negative data so it could be suitable to measure the performance of the model by ignoring instances which are correctly *"not recommended"* (i.e., TN).

It is possible to compute the metrics as follows:

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \quad \text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}. \tag{4}$$

These metrics are suitable for evaluating tasks such as *top-N recommendation*. When a recommender algorithm predicts the top-*N* items that a user is expected to find interesting, by using the recall we can compute the percentage of known relevant items from the test set that appear in the *N* predicted items. Basu et al. [2] describe a better way to approximate precision and recall in top-*N* recommendation by considering only rated items. According to Herlocker et al. [12], we must consider that:

- usually the number of items rated by each user is much smaller than the items available in the entire dataset;
- the number of relevant items in the test set may be much smaller than that one in the whole dataset.

Therefore, the value of the precision and the recall depend heavily 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.

F-measure, used in [18, 17], allows a single measure that combines precision and recall by means of the following relations:

$$\text{F-measure} = \frac{2 \times \text{recall} \times \text{precision}}{\text{recall} + \text{precision}} = \frac{2 \cdot \text{TP}}{2 \cdot \text{TP} + \text{FN} + \text{FP}}. \tag{5}$$

Receiver Operating Characteristic (ROC) is a graphical technique that uses two metrics, *true positve rate (TPR)* and *false positive rate (FPR)* defined as

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}} \quad \text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}}, \tag{6}$$

to visualize the trade-off between TPR and FPR by varying the length *N* of the list returned to the user. On the vertical axis the ROC curves plot the TPR, i.e., the number of the instances recommended related to the total number of relevant ones, against the FPR, i.e., the ratio between positively misclassified instances and all the not relevant instances. Thus, by gradually varing the threshold and by repeating the classification process, we can obtain the ROC curve which visualizes the continuous trade-off between TPR and FPR.

3. Proposed methodology

According to our experience, given a dataset, we point out four important observations about recommendation algorithms:

1. the quality of the recommendation may depend on the length of the user profile;
2. the quality of the recommendation may depend on the popularity of the items rated by a user, i.e., the quality of the recommendation may be different according to the fact that a user prefers popular items against unpopular ones;
3. when comparing two algorithms, different metrics (e.g., MAE and recall) may provide discording results;
4. recommendation algorithms are usually embedded into RS that provide users with a limited list of recommendation because of graphic interface constraints, e.g., many applications show to the users only the highest 5 ranked items.

In this section, depending on previous observations, we describe a methodology for evaluating and comparing recommendation algorithms. The methodology is divided into three steps:

step 1: statistical analysis and partitioning;

step 2: optimization of the model parameters;

step 3: computation of the recall for the top-*N* recommendation.

3.1. Statistical analysis and partitioning

The partitioning of the dataset is performed by means of the m-fold cross-validation. Together with the partitioning we analyse the dataset in order to find groups of users and the most popular items based on the number of ratings. This will help us to identify the behavior of the model according to the length of the user profile and to the popularity of the items of the recommendation. In order to create groups of users we use the following method:

a) users are sorted according to the length of their profiles (i.e., the number of ratings);

b) users are divided into two groups so that each of them contains the 50% of the ratings;

c) a group can be further divided into two subgroups (as previously described in b) in order to better analyse the behaviour of a learning algorithm with respect to the length of the profiles.

Similarly, in order to find the most popular items, we adopt the following schema:

a) items are sorted according to the number of ratings;

b) items are divided in two groups so that each of them contains 50% of the ratings.

For example, Figure 4 presents a plot where the x axis shows the items sorted in decreasing order of popularity and the y axis shows the number of ratings (percentage). The point 0.5 highlights which are the most rated items responsible for the 50% of all the ratings.

3.2. Optimization of model parameters

ROC curves can be used to visualize the trade-off between TPR (i.e., recall) and FPR when we vary the threshold which allows us to classify an item as *"to recommend"* or *"not to recommend"*. If we change some parameters in the model (e.g., the latent size of the singular value decomposition that will be described in Section 4) we obtain a family of ROC curves. In order to optimize the model parameters, according to the type of dataset, we implement two techniques based on ROC curves: ROC1 and ROC2. Both the techniques use leave-k-out and randomly withhold the 25% of the rated items for each user.

ROC1 is suitable for explicit (i.e., not binary) datasets. For each item we have the pair $\langle p_i, a_i \rangle$ which represents the predicted rating and the real rating. To compare these values, let t be a *threshold value* chosen inside the rating scale that divides it in two different parts; when $r \geq t$, the corresponding item is classified as *to recommend*, otherwise as *not to recommend*. Thus, given a threshold t, the predicted rating and the real rating may fall in either of the two parts of the rating scale as shown in Figure 1. We classify each item as TP, TN, FP or FN and, subsequently, we compute the couple (FPR,TPR), according to (6), which corresponds to a point of the ROC curve for the given threshold. By varying the value of the threshold in the rating scale we obtain several points of the ROC curve. We repeat the application multiple times and, finally, we average the curves.

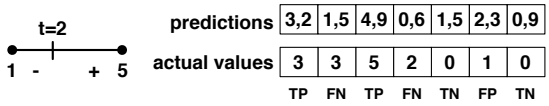


Figure 1. Example of the application of the ROC1 technique.

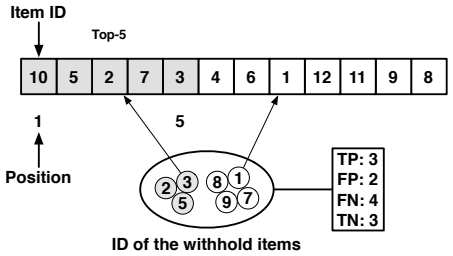


Figure 2. Example of ROC2 application on a binary dataset.

ROC2 is suitable for both binary and not binary dataset. For each user, we put the prediction vector in descreasing order thus obtaining the top-*N* recommendation. The first *N* items are those that should be recommended to the user. If the dataset is binary, the TPs correspond to the withheld items which are in the first *N* positions, while the FNs correspond to the withheld items which are not in the first *N* positions. To compute FPs, it is sufficient to compute the difference between *N* and the number of withheld items. Thus, we obtain the number of TNs, by computing the difference L-(TP+FP+FN), where L is the length of the user profile deprived of the rated items, but not withheld. Then we compute the pair (FPR,TPR) corresponding to the value *N*. Figure 2 shows an example of computation. By varying the number *N* of the recommended items we get the other points of the ROC curve.

3.3. Computation of recall

In this section we evaluate the quality of recommendations. The metric chosen is the recall, since, differently from other metrics such as MAE, expresses the effective capability of the system in pursuing the top-*N* recommendation task. The computation of the recall is made on the

test set by selecting a user (hereafter the *active user*) and repeating the following steps for each of the positively-rated items. According to the adopted rating scale, a rating is considered positive if it is greater than a certain threshold t. Observe that with a binary scale all the ratings are positive. The steps are:

a) withholding a positively-rated item,

b) generating the predictions for the active user by means of the model previously created on the training set,

c) recording the position of the withheld item into the ordered list of the predictions.

The recall is classified according to two dimensions: the user profile length and the popularity of the withheld item.

4. Algorithms and datasets

In this section we describe the algorithms and the datasets used in Section 5 as an example of application of the methodology .

4.1. Algorithms

We consider the two following learning algorithms to create the model from the training set: Singular Value Decomposition (SVD), Naive Bayesian Networks (NBN).

SVD is a matrix factorization technique used in *Latent Semantic Indexing (LSI)* [4] which decomposes the URM matrix of size $m \times n$ and rank r as:

$$\text{URM} = U \cdot S \cdot V^T \tag{7}$$

where U and V are two orthogonal matrices of size $m \times r$ and $n \times r$, respectively. S is a diagonal matrix of size $r \times r$, called *singular value matrix*, whose diagonal values $(s_1, s_2, \ldots s_r)$ are positive real numbers such that $s_1 \geq s_2 \geq \ldots \geq s_r$. We can reduce the $r \times r$ matrix S in order to have only $L < r$ largest diagonal values, discarding the rest. In this way, we reduce the dimensionality of the data and we capture the *latent* relations existing between users and items that permit to predict the values of the unrated items. Matrices U and V are reduced accordingly obtaining U_L and V_L^T. Thus the reconstructed URM is

$$\text{URM}_L = U_L \cdot S_L \cdot V_L^T. \tag{8}$$

URM_L is the best *L*-rank linear approximation of the URM. According to Berry et al. [4], the URM resulting from SVD is less noisy than the original one and captures the latent associations between users and items. We compute $U_L\sqrt{S_L}^T$ of size $m \times L$, and $\sqrt{S_L}V_L^T$ of size $L \times n$. Then, in order to estimate the prediction for the active user u on the item i, we calculate the dot product between the u^{th} row of $U_L\sqrt{S_L}^T$ and the i^{th} column of $\sqrt{S_L}V_L^T$ as:

$$(U_L\sqrt{S_L}^T)_u \cdot (\sqrt{S_L}V_L^T)_i. \tag{9}$$

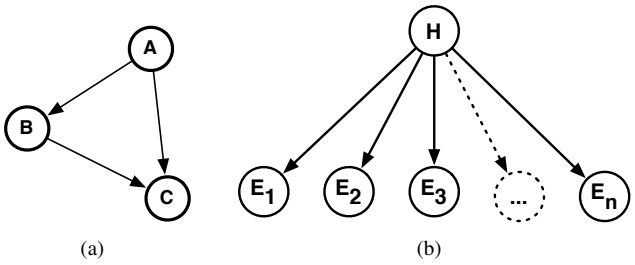


Figure 3. (a) is a simple bayesian networks. (b) is a naive bayesian network.

A **BN** is a probabilistic graphical model represented by means of a directed acyclic graph where *nodes* (also called *vertices*) are random variables and *arcs* (also known as *links*) express probabilistic relationships among variables. A directed graph is used in order to express causal relationships between random variables [6]. By using as an example the BN of Figure 3(a), the product rule of probability, deduced from the definition of conditional probability, allows to decompose the joint distribution as:

$$P(A, B, C) = P(C|A, B)P(B|A)P(A), \tag{10}$$

where A, B and C are random variables and A is parent of B and C.

In order to build a RS by means of BNs, we can define a random variable for each item in the URM. If the URM is binary also the random variables are binary. However, it is very difficult to obtain the optimal structure of the network. In fact, according to [9], the problem is NP-hard. A simpler and faster alternative to BNs are *Naive Bayesian Networks (NBN)*, obtained assuming that all variables have the same parent. Thus, given an active user u and an unrated item i, we have the structure represented in Figure 3(b) where: $H = 1$ is the hypothesis on item i and $E_1, E_2, \ldots, E_n$ are the *evidences*, i.e., the items rated by the active user. We compute the prediction as the probability that the active user will be interested in the item i given the evidences as:

$$P(H = 1|E_1, \ldots, E_n) = P(H)P(E_1|H) \ldots P(E_n|H). \tag{11}$$

$P(H)$ is the probability that the item i will be rated independently from the active user and $P(E_w|H)$ is the similarity between the item w and the item i. We apply the naive model only on binary dataset by computing:

$$P(H) = \frac{\text{\# ratings of item } i}{\text{\# ratings of most rated item}}$$

because we do not have any information about the movies, and

$$P(E_w|H) = \frac{P(E_w \cap H)}{P(H)} = \frac{\text{\# 1s in common}}{\text{\# ratings of item } i}.$$

4.2. Datasets

In the following example we use the MovieLens (ML) dataset and a new commercial dataset, from here on NewMovies (NM), which is larger, sparser and closer to reality with respect to ML.

ML is a not binary public dataset taken from a web-based research recommender system. The rating scale adopted is $[1 \dots 5]$ and 0 stands for items not rated. There are 943 users and 1682 items. Each user rated at least 20 items. The dataset has 100.000 ratings with a sparsity level equal to 93.69% and an average number of ratings equal to 106 per user.

NM is a commercial non-public dataset provided by an IPTV service provider. The URM is composed by 26799 rows (users) and 771 columns (items) each of which has at least two ratings. This dataset contains 186.459 ratings with a sparsity level equal to 99.34% and an average number of ratings equal to 7 per user. NM dataset is binary, where 0 stands for a not rated item and 1 for a rated item.

5. Application of the methodology

In this section we show an example of application of the methodology explained in Section 3.

5.1. Statistical analysis and partitioning

As we described in Section 3.1, by means of a simple statistical analysis we can divide the users in different groups according to the number of ratings that each user has rated and items based on their popularity.

In the *ML* dataset about 50% of the ratings refer to users having a profile with less than 200 ratings while the remaining 50% is due to users with more than 200 ratings. Since ML contains users with very long profiles, in the first group there are too many users, thus we decide to further split it obtaining the following groups:

- $[1 \dots 99]$ 579 users, $\sim$25% of all the ratings;
- $[100 \dots 199]$ 215 users, $\sim$25% of all the ratings;
- $[200 \dots \infty]$ 149 users, $\sim$50% of all the ratings.

The top 210 items account for 50% of the ratings, as shown in Figure 4.

In the *NM* dataset we first partitioned the users into two groups having 50% of the ratings each one. We split furthermore the second group obtaining:

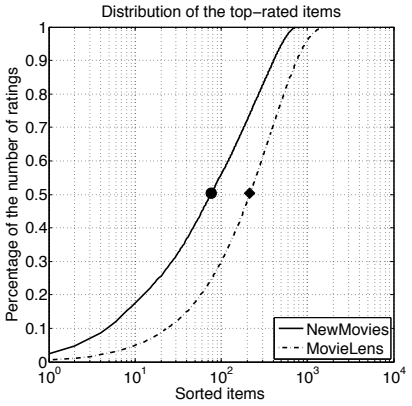


Figure 4. Distribution of the top-rated items.

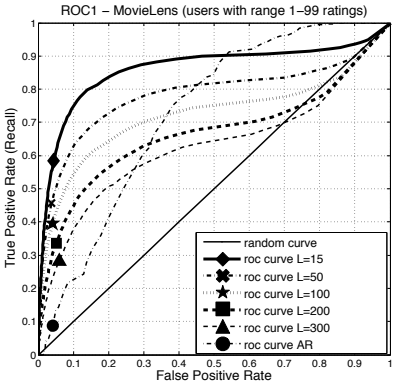


Figure 5. Optimization, using ROC1 technique, of the SVD algorithm on the ML dataset. Model created with users who have at most 99 ratings.

- $[2 \dots 9]$ 21810 users, $\sim$50% of all of the ratings;
- $[10 \dots 19]$ 3400 users, $\sim$25% of all of the ratings;
- $[20 \dots \infty]$ 1568 users, $\sim$25% of all of the ratings.

Figure 4 shows that the top 80 items account for 50% of the ratings.

5.2. Optimization of the parameters

As we described in Section 4.1, SVD allows to compute the closest rank-L linear approximation of the URM. In order to find the optimal parameter L we can use the ROC technique.

ROC1 technique. Figure 5 shows the application of the SVD algorithm on the ML dataset in order to evaluate which latent size is the best choice. In this example we considered only the users who have at most 99 ratings. Each curve is

	Popularity	
UPL	First80	NoFirst80
2-9	22.2%	6.9%
10-19	24.1%	0.1%
≥ 20	25.7%	10.1%

Table 1. Recall obtained with the SVD algorithm with *L=15* on the NM dataset. Model created with users who have at least 2 ratings. *UPL* stands for User Profile Length.

related to a specific value of the latent size *L* of SVD algorithm. In the same graph it is also represented a trivial recommendation algorithm, used as a term of comparison, called *Average Recommender (AR)*. This algorithm computes, for each item, the mean of all its ratings. Then, it creates a sorted list of items according to such mean values. This list is used for the recommendation to all users. Each point of the ROC curve represents a pair (FPR,TPR) related to a specific threshold. In this case the value of the threshold is in the range $(1 \dots 5)$. For example, in Figure 5 we highlight the points related to the threshold equal to 3. The best curve is obtained by setting the latent size *L* to 15.

ROC2 technique. According to Section 3.2, we apply ROC2 techique in order to optimize SVD algorithm on the binary dataset NM. Figure 6(a) shows an application of the ROC2 techinque, where the model is generated using all users but predictions are made only for users who have rated between 2 and 9 items. It is also shown the curve corresponding to the top-rated recommendation, that is we recommend to each user the most rated items, in popularity order. The model with a latent size of 15 is the best one. For each curve we highlight 2 points that correspond to $N = 5$ and $N = 20$, respectively. If the dataset is not binary we can use, for example, the result obtained by means of the ROC1 technique. Figure 6(b) shows that 15 is again the best value for parameter *L*. Figure 6(c) shows again the application of the ROC2 technique, but on the NM dataset when the NBN algorithm is adopted. The model is created using all the user with a number of rating greater or equal than 2. By means of this model we compute the prediction for the three groups of users. In any case, the best prediction is obtained for users having ratings into the range of $[2 \dots 9]$.

5.3. Computation of recall

We use the recall in order to compare the quality of the algorithms described in Section 4. Table 1 shows the recall obtained applying the SVD algorithm on the NM dataset. The model is created considering all the users in the training set. The users in the test set are divided according to the 3 groups previously defined. In Table 1(a) we select all users with at least 2 ratings to create the model by means of naive bayesian networks. In Table 1(b) and 1(c) we create the model using users with a more rich profile, i.e., with at least 10 and 20 ratings, respectively. The test considers again all the users. This allows to see if a more rich and compact model can improve recommendations.

6. Conclusions

RS are a powerful technology: they help both users to find what they want and e-commerce companies to improve their sales.

Each recommender algorithm may behave in different ways with respect to different datasets. Given a dataset, by means of the methodology proposed in this paper we can analyze the behavior of different recommender algorithms.

On the basis of several results obtained, we can declare that a RS should have different models to make recommendations to different groups of users and with respect to the popularity of the items.

References

[1] *Readings in Information Retrieval*. Morgan Kaufmann, 1997.

[2] C. Basu, H. Hirsh, and W. Cohen. Recommendation as classification: Using social and content-based information in recommendation. *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, 714720, 1998.

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

[4] M. Berry, S. Dumais, and G. O'Brien. Using Linear Algebra for Intelligent Information Retrieval. *SIAM Review*, 37(4):573–595, 1995.

[5] D. Billsus and M. Pazzani. Learning collaborative information filters. *Proceedings of the Fifteenth International Conference on Machine Learning*, 54, 1998.

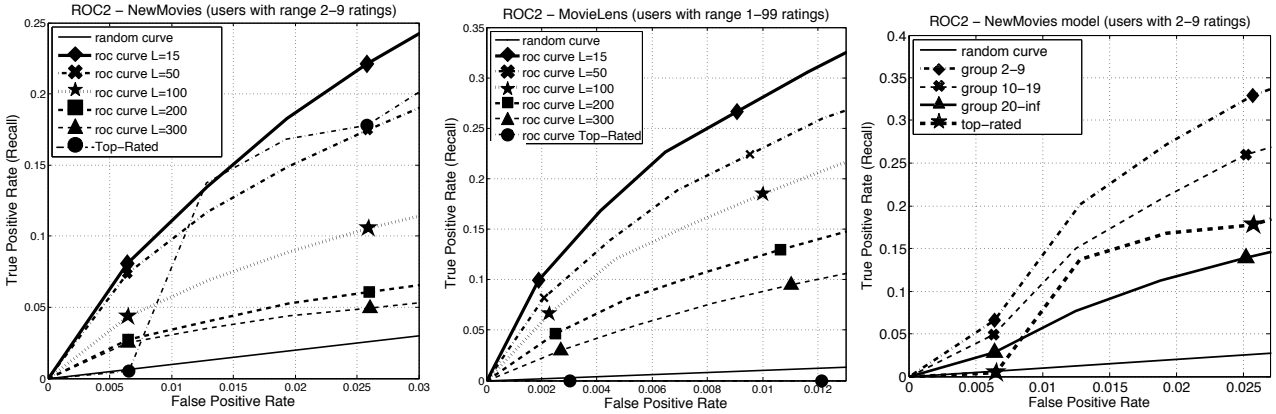
[6] C. Bishop. *Pattern recognition and machine learning*. Springer, 2006.

[7] J. Breese, D. Heckerman, C. Kadie, et al. Empirical analysis of predictive algorithms for collaborative filtering. *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, 461, 1998.

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

[9] D. Chickering, D. Geiger, and D. Heckerman. Learning Bayesian networks is NP-hard.

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



(a) (b) (c)

Figure 6. Optimization, using ROC2 technique, of the SVD algorithm on: (a) NewMovie dataset, users who have at most 9 ratings. (b) ML dataset, users who have at most 99 ratings. (c) shows ROC2 technique applied to the NBN algorithm on the NewMovies dataset.

(a)

	Popularity	
UPL	First80	NoFirst80
2-9	31.6%	0.13%
10-19	24.7%	0.19%
≥ 20	20.7%	0.01%

(b)

	Popularity	
UPL	First80	NoFirst80
2-9	30.6%	0.04%
10-19	26.3%	0.13%
≥ 20	21.7%	0.02%

(c)

	Popularity	
UPL	First80	NoFirst80
2-9	29.3%	0.01%
10-19	23.7%	0.01%
≥ 20	23.2%	0.02%

Table 2. Recall obtained with the NBN algorithm applied on the NM dataset. Models are created with users who have, respectively, at least 2 (a), 10 (b) and 20 (c) ratings. *UPL* **stands for User Profile Length.**

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

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

[13] J. L. Herlocker, J. A. Konstan, A. Borchers, and J. Riedl. An algorithmic framework for performing collaborative filtering. In *SIGIR '99: Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 230–237, New York, NY, USA, 1999. ACM.

[14] J. Konstan, B. Miller, D. Maltz, J. Herlocker, L. Gordon, and J. Riedl. GroupLens: applying collaborative filtering to Usenet news. *Communications of the ACM*, 40(3):77–87, 1997.

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

[16] B. Sarwar, G. Karypis, J. Konstan, and J. Reidl. Item-based collaborative filtering recommendation algorithms. In *WWW '01: Proceedings of the 10th international conference on World Wide Web*, pages 285–295, New York, NY, USA, 2001. ACM Press.

[17] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Analysis of recommendation algorithms for e-commerce. *Proceedings of the 2nd ACM conference on Electronic commerce*, pages 158–167, 2000.

[18] B. Sarwar, G. Karypis, J. Konstan, J. Riedl, and M. U. M. D. O. C. SCIENCE. *Application of Dimensionality Reduction in Recommender System. A Case Study*. Defense Technical Information Center, 2000.

[19] U. Shardanand and P. Maes. Social information filtering: algorithms for automating "word of mouth". *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 210–217, 1995.

[20] I. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 2005.