Analysis of Recommender Systems' Algorithms

Article · September 2003

Source: OAI

4 authors, including:

Manolis G. Vozalis
University of Macedonia
32 PUBLICATIONS **499** CITATIONS

SEE PROFILE

Konstantinos G. Margaritis
University of Macedonia
403 PUBLICATIONS **1,850** CITATIONS

SEE PROFILE

Analysis of Recommender Systems' Algorithms

Emmanouil Vozalis, Konstantinos G. Margaritis

Abstract— **In this work, we will provide a brief review of different recommender systems' algorithms, which have been proposed in the recent literature. First, we will present the basic recommender systems' challenges and problems. Then, we will give an overview of association rules, memory-based, model-based and hybrid recommendation algorithms. Finally, evaluation metrics to measure the performance of those systems will be discussed.**

Keywords— **Collaborative Filtering, Recommender Systems, Machine Learning**

I. Introduction

RECOMMENDER Systems were introduced as a computer-based intelligent technique to deal with the problem of information and product overload. They can be utilized to efficiently provide personalized services in most e-business domains, benefiting both the customer and the merchant. Recommender Systems will benefit the customer by making to him suggestions on items that he is assumably going to like. At the same time, the business will be benefited by the increase of sales which will normally occur when the customer is presented with more items he would likely find appealing.

The two basic **entities** which appear in any Recommender System are the *user* (sometimes also referred to as *customer*) and the *item* (also referred to as *product* in the bibliography). A user is a person who utilizes the Recommender System providing his opinion about various items and receives recommendations about new items from the system.

The **input** to a Recommender System depends on the type of the employed filtering algorithm. Various filtering algorithms will be discussed in subsequent sections. Generally, the input belongs to one of the following categories:
1. *Ratings* (also called *votes*), which express the opinion of users on items. Ratings are normally provided by the user and follow a specified numerical scale (example: 1-bad to 5-excellent). A common rating scheme is the *binary rating scheme*, which allows only ratings of either 0 or 1. Ratings can also be gathered implicitly from the user's purchase history, web logs, hyperlink visits, browsing habits or other types of information access patterns.
2. *Demographic data*, which refer to information such as the age, the gender and the education of the users. This kind of data is usually difficult to obtain. It is normally collected explicitly from the user.
3. *Content data*, which are based on a textual analysis of documents related to the items rated by the user. The features extracted by this analysis are used as input to the filtering algorithm in order to infer a user profile.

The authors are with the Parallel and Distributed Processing Laboratory, Department of Applied Informatics, University of Macedonia, Thessaloniki, Greece. E-mail: {mans, kmarg}@uom.gr. Web: http://macedonia.uom.gr/ {mans, kmarg}.

The **goal** of Recommender Systems is to generate suggestions about new items or to predict the utility of a specific item for a particular user. In both cases the process is based on the input provided, which is related to the preferences of that user. Let m be the number of users $U = \{u_1, u_2, ..., u_m\}$ and n the number of items $I = \{i_1, i_2, ..., i_n\}$. Each user u_i, where $i = 1, 2, ..., m$, has a list of items I_{u_i} for which he has expressed his opinion about. It is important to note that $I_{u_i} \subseteq I$, while it is also possible for I_{u_i} to be the *null set*, meaning that users are not required to reveal their preferences for all existing items. Also, the count of items in I_{u_i} is n_i, or $n_i = |I_{u_i}|$, with $n_i \leq n$. User opinions are generally stated in the form of a *rating score*. Specifically, the rating of user u_i for item i_j, where $j = 1, 2, ..., n$, is denoted by $r_{i,j}$, where each rating is either a real number within the agreed numerical scale or $\perp$, the symbol for "no rating". All these available ratings are collected in a *m x n user-item matrix*, denoted by R. The proposed filtering algorithms employ various techniques either on the *rows*, which correspond to ratings of a single user about different items, or on the *columns*, which correspond to different users' ratings about a single item, of this user-item matrix. We distinguish a single user $u_a \in U$ as the *active user* and define $NR \in I$ as the subset of items for which the active user has not stated his opinion yet, and as a result, for which the Recommender System should generate suggestions.

The **output** of a Recommender System can be either a *Prediction* or a *Recommendation*.

- A *Prediction* is expressed as a numerical value, $r_{a,j}$, which represents the anticipated opinion of active user u_a for item i_j. This predicted value should necessarily be within the same numerical scale (example: 1-bad to 5-excellent) as the input referring to the opinions provided initially by active user u_a. This form of Recommender Systems output is also known as *Individual Scoring*.
- A *Recommendation* is expressed as a list of N items, where $N \leq n$, which the active user is expected to like the most. The usual approach in that case requires this list to include only items that the active user has not already purchased, viewed or rated. This form of Recommender Systems output is also known as *Top-N Recommendation* or *Ranked Scoring*.

II. Challenges and Problems

In this section we will discuss the fundamental problems that Recommender Systems suffer from. It is important for each new filtering algorithm proposed to suggest solutions for those problems.

Quality of Recommendations: Trust is the key word here. Customers need recommendations, which they can trust. To achieve that, a recommender system should minimize

false positive errors, i.e. products, which are recommended (positive), though the customer does not like them (false). Such errors lead to angry customers [1].

Sparsity: It is usual in e-business stores that even the most active customers have purchased or rated a very limited percentage of products, when compared to the available total. That leads to sparse user-item matrices, inability to locate successful neighbors and finally, the generation of weak recommendations. As a result, techniques to reduce the sparsity in user-item matrices should be proposed [2] [3].

Scalability: Recommender Systems require calculations, which grow with both the number of customers and the number of products. 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 is increased. Thus, it is crucial to apply algorithms, which are capable of scaling up in a successful manner [3].

Loss of Neighbor Transitivity: Assume that user u_i and user u_k correlate highly, and user u_k also correlates highly with user u_l. Then it is a possibility that users u_i and u_l correlate highly too, via their common correlation with user u_k. Nevertheless, such a transitive relationship is not captured in Collaborative Filtering systems, unless users u_i and u_l have purchased or rated many common items. It is important to employ methods in order to capture such relationships [4].

Synonymy: Recommender Systems are usually unable to discover the latent association between products, which have different names, but still refer to similar objects. A method should be employed to inquire the existence of such latent associations and utilize it in order to generate better recommendations [5].

First Rater Problem: An item cannot be recommended unless a user has rated it before. This problem applies to new items and also to obscure items and is particularly harmful to users with eclectic tastes. It is also known as the Cold Start Problem [6] [7].

Unusual User Problem: It is also known as the Gray Sheep problem. It refers to individuals with opinions that are "unusual", meaning that they do not agree or disagree consistently with any group of people. Those individuals would not easily benefit from recommender systems since they would rarely, if ever, receive accurate predictions [2].

III. Recommender Systems using Memory-Based Methods

In the following paragraphs, we will describe the recommendation process for systems that are utilizing *memory-based algorithms*, meaning that they operate over the entire user-item matrix, R, to make predictions. The majority of such systems belong in the *Collaborative Filtering (CF) Systems* category and mainly deal with user-user similarity calculations, meaning that they utilize user neighborhoods, constructed as collections of similar users. In other words, they deal with the rows of the user-item matrix, R, in order to generate their results. The main steps of this process are *Representation*, *Neighborhood Formation* and *Recommendation Generation*.

A. Representation

In the original representation, the input data is defined as a collection of numerical ratings of m users on n items, expressed by the *mxn user-item matrix*, R. As mentioned earlier, users are not required to provide their opinion on all items. As a result, the user-item matrix is usually sparse, including numerous "no rating" values, thus making it harder for filtering algorithms to generate satisfactory results. Techniques, whose purpose is to reduce the sparsity of the initial user-item matrix, have been proposed in order to improve on the results of the recommendation process. Such techniques will be discussed here.

A.1 Sparsity Reduction Techniques in the Recommendation Process

Default Voting

In the simplest technique used to reduce the sparsity of the user-item matrix, we simply insert a default rating, d, for appropriate items for which there exist no explicit ratings. "Appropriate" is the key word here, meaning that it is wise to choose the matrix entries where the default ratings would be inserted. Nevertheless in [8] where this technique is proposed, the authors do not specify where the default ratings will be inserted. In most cases, the value for default rating, d, is selected to be the neutral or the somewhat negative preference for the unobserved item. The cost of applying the method of *Default Voting* is rather low.

Preprocessing using Averages

This is an extension to the Default Voting method, proposed in [3]. The basic idea, again, is to scan through the user-item matrix and replace missing values by using some naive non-collaborative methods that provide a simple rating.

In the *user average scheme*, for each user, u_i, we compute the average user rating, $\bar{r_i}$, which is expressed by the average of the corresponding row in the user-item matrix. The user average is then used to replace any missing $r_{i,j}$ value. This approach is based on the idea that a user's rating for a new item could be simply predicted if we take into account the same user's past ratings. It can be formally stated as:

$$r_{i,j} = \begin{cases} \bar{r_i}, \ if \ user \ u_i \ has \ not \ rated \ item \ i_j \\ r, \ if \ user \ u_i \ rated \ item \ i_j \ with \ r \end{cases}$$

In the *item average scheme*, we utilize the item average, $\bar{r_j}$, of each item, i_j, as a fill-in for missing values $r_{i,j}$ in the matrix. Specifically, we calculate the column average of each column in the user-item matrix, and fill all slots of the same column that have no value, denoted by $\perp$, using that average:

$$r_{i,j} = \begin{cases} \bar{r_j}, \ if \ user \ u_i \ has \ not \ rated \ item \ i_j \\ r, \ if \ user \ u_i \ rated \ item \ i_j \ with \ r \end{cases}$$

The claim behind the item average scheme is that an item's average ratings can predict a user's opinion on that same item.

Applying either the user average or the item average scheme on every missing value of the user-item matrix will lead to a new, complete matrix, where the impact of sparsity is totally removed. On the other hand, highly scalable matrix computations or statistical methods, such as LSI/SVD discussed in subsequent sections, are more difficult to employ in a fully dense user-item matrix.

In the *composite scheme* the collected information for items and users both contribute in the final result. The main idea behind this method is to use the average of user u_i on item i_j as a base prediction and then add a correction term to it based on how the specific item was rated by other users. The correction is necessary if we take into consideration the way that different users state their preferences: a user may rate all items in a scale of 1-3, giving a 3 to items he liked most, while another user may be more lenient, grading in a scale of 3-5.

The scheme works like this: When a missing entry regarding user's u_i opinion on item i_j is located, we first compute the user average, $\bar{r_i}$, calculated as the average of the corresponding user-item matrix row. Then we look for existing ratings in the column which corresponds to item i_j. Assuming that a set of l users, $L = \{u_1, u_2, ..., u_l\}$, has provided a rating for item i_j, we can compute a correction term for each user $u_p \in L$ equal to $\delta_p = r_{p,j} - \bar{r_p}$. After the corrections for all users in L are computed, the composite rating can be calculated:

$$r_{i,j} = \begin{cases} \bar{r_i} + \frac{\sum_{p=1}^{l} \delta_p}{l}, \ if \ user \ u_i \ has \ not \ rated \ item \ i_j \\ r, \ if \ user \ u_i \ rated \ item \ i_j \ with \ r \end{cases}$$

An alternative way of utilizing the composite scheme is through a simple transposition: first compute the item average, $\bar{r_j}$, (average of the column which corresponds to item i_j) and then compute the correction terms, δ_q, by scanning through all k items $K = \{i_1, i_2, ..., i_k\}$ rated by user u_i. The fill-in value of $r_{i,j}$ would then be:

$$r_{i,j} = \bar{r_j} + \frac{\sum_{q=1}^{k} \delta_q}{k}$$

where k is the count of items rated by user u_i and the correction terms are computed for all items in K as $\delta_q = r_{i,q} - \bar{r_q}$.

Experiments in [3] have shown that the composite rating scheme has the best predicting accuracy, when compared to both user and item average schemes. Nevertheless, its main drawback is that it requires intensive calculations in order to be computed for all missing values in the original user-item matrix, R.

Use of Filterbots

Filterbots [9] [10] are automated rating agents that evaluate new items and enter those ratings in the user-item matrix. Recommender Systems should view Filterbots as ordinary users, with a single difference from the human users: filterbots are very prolific, generating numerous predictions but at the same time, they are very generous, never requesting predictions for themselves.

The incorporation of filterbots in a Recommender System is rather simple, as illustrated by the following procedure:

- First, they accept new items as they are inserted in the user-item matrix or request existing items.
- They apply the included rating algorithm to the fetched item. Their rating algorithm is directly related to the amount of intelligence that they carry. A couple of rating algorithms will be discussed in subsequent paragraphs.
- After generating the rating for the fetched item, they insert it in the user-item matrix, R.

The value of filterbots is crucial in both *Sparsity Reduction* and also in tackling the *Cold Start Problem*: Ratings generated by filterbots make the user-item matrix more dense. At the same time, by producing predictions for new items, as soon as they arrive, they make it easier and faster for those items to be suggested to human users as well. Otherwise, all new items would have to wait until they were rated by an adequate number of human users, before they would be suggested to new ones.

The integration of filterbots in a Recommender System is expected to improve the utility of the system for users who agree with the filterbots. At the same, users who do not share similar preferences with the filterbots will not be affected at all.

An interesting issue concerning filterbots has to do with the amount of intelligence that is incorporated in them. The most simple type of filterbots, called *Genrebots*, simply assign the highest possible rating to an item that matches the filterbot's genre, while their rating for not matching items is lower. For example, if our Recommender System deals with films, a "HorrorBot", whose appointed genre is clearly horror films, would rate "Friday the 13th" with a 5, assuming a rating scale of 1-bad to 5-excellent, while assigning a rating of 3 to other films such as "Citizen Cane", "Hollywood Ending" or "Beauty and the Beast". Clearly, there should be different Genrebots for each item genre.

A filterbot, including a greater amount of intelligence, could generate its predictions based on Information Retrieval/Information Filtering techniques, and more specifically by applying a modified TFIDF [11] on the content features of each item. Such an intelligent filterbot, called *content-filterbot*, can be using the describing keywords of the domain, in order to achieve personalized recommendations.

To produce better results, filterbots can be combined in a number of ways:

- By averaging the ratings of different filterbots together.
- By applying *regression* to create a best fit combination for each user. To achieve this, in our calculations we should utilize the user's known rating as the dependant variable, while the generated predictions from the filterbots, we wish to combine, would be the independent variables. Then, the resultant weights should be used in order to generate predictions for new items, by creating linear combinations of filterbots' recommendations.

• By inserting the ratings of different filterbots in a Recommender System and presenting the Recommender System's result as the final prediction.

In all cases above, after combining the filterbots' separate results into a single recommendation, that recommendation should be entered in the user-item matrix, R, to aid the recommendation process, as discussed earlier.

An idea, probably worth looking into, is mentioned in [3]: employ a community of filtering agents that operate under natural selection so that ineffective agents (their presence will not improve the quality of recommendations when compared to the plain system) are eliminated as time passes by and replaced by variants of effective ones.

Use of Dimensionality Reduction Techniques

A sparse data set may become denser if we apply a dimensionality reduction technique, like *Singular Value Decomposition* (SVD), on its data. Such a reduction may solve the Synonymy Problem, described earlier, capturing latent relationships among users [12]. After generating a reduced dimensionality matrix, we should use a vector similarity metric to compute the proximity between customers and hence to form the neighborhood [3] [1].

The neighborhood in the SVD-reduced space is created with the execution of the following steps:

1. Preprocess the initial user-item matrix, R of size mxn in order to eliminate all missing data values, and obtain normalized matrix R_{norm}.
2. Compute the SVD of R_{norm} and obtain matrices U, S and V, of size mxm, mxn, and nxn, respectively. Their relationship is expressed by: $R_{norm} = U \cdot S \cdot V^T$. Matrices U and V are orthogonal and span the *column space* and the *row space* of the initial user-item matrix, R, respectively. Matrix S is a diagonal matrix, called the *singular matrix*.
3. Perform the dimensionality reduction step by keeping only k diagonal entries of the matrix S to obtain a kxk matrix, S_k. Similarly, matrices U_k and V_k of size mxk and kxn are generated.
4. Compute $\sqrt{S_k}$ and then compute two matrix products $U_k \cdot \sqrt{S_k}^T$ and $\sqrt{S_k} \cdot V_k^T$, that represent m pseudo-users and n pseudo-items in the k dimensional feature space.
5. Proceed with the Neighborhood Formation (step 2 in the recommendation process, which will be described extensively in the following section). Note that in case SVD/LSI methods were used to reduce the dimensions of the initial user-item matrix, similarity between a pair of users should now be calculated by the dot product of the rows corresponding to those users in the new pseudo-users matrix, $U_k \cdot \sqrt{S_k}^T$.

An important issue when applying SVD/LSI methods in order to reduce the initial dimensions of the user-item matrix, R, is how to determine the value of k, which represents the dimensionality reduction. The value of k is critical for the effectiveness of the low dimensional representation. Specifically, k should be large enough to capture all the latent relationships in matrix R, while at the same time k should be small enough to avoid over-fitting errors. Usually, the appropriate value of k is evaluated experimentally, being different for different data sets.

B. Neighborhood Formation

In the most important step of the recommendation process, the "similarity" between users in the user-item matrix, R, should be calculated. Users similar to the active user, u_a, will form a proximity-based neighborhood with him. The active user's neighborhood should then be utilized in the following step of the recommendation process (step 3, which will be discussed extensively in the following section), in order to estimate his possible preferences.

Neighborhood Formation is implemented in two steps: Initially, the similarity between all the users in the user-item matrix, R, is calculated with the help of some proximity metrics. The second step is the actual neighborhood generation for the active user, where the similarities of users are processed in order to select those users from whom the neighborhood of the active user will consist. These two steps will be presented in detail here:

B.1 Similarity of Users

Assuming the existence of a user-item matrix, R, the similarity between two users from that matrix, user u_i and user u_k, can be calculated utilizing either the *Pearson Correlation Similarity* or the *Cosine/Vector Simirality*, which are the main proximity metrics employed in the Recommender Systems literature.

Pearson Correlation Similarity

To find the proximity between users u_i and u_k, we can utilize the Pearson Correlation metric. Pearson Correlation was initially introduced in the context of the GroupLens project [13].

$$sim_{ik} = corr_{ik} = \frac{\sum_{j=1}^{l}(r_{ij}-\bar{r_i})(r_{kj}-\bar{r_k})}{\sqrt{\sum_{j=1}^{l}(r_{ij}-\bar{r_i})^2 \sum_{j=1}^{l}(r_{kj}-\bar{r_k})^2}}$$

It is important to note that the summations over j are calculated over the l items for which *both* users u_i and u_k have expressed their opinions. Obviously, $l \leq n$, where n represents the number of total items in the user-item matrix, R.

Cosine or Vector Similarity

In the n-dimensional item space (or k-dimensional item space, in case dimension reduction techniques, like SVD/LSI, were applied), we can view different users as feature vectors. A user vector consists of n feature slots, one for each available item. The values used to fill those slots can be either the rating, r_{ij}, that a user, u_i, provided for the corresponding item, i_j, or 0, if no such rating exists.

Now, we can compute the proximity between two users, u_i, and u_k, by calculating the similarity between their vectors, as the cosine of the angle formed between them.

$$sim_{ik} = cos_{ik} = \sum_j \frac{r_{ij}}{\sqrt{\sum_j r_{ij}^2}} \frac{r_{kj}}{\sqrt{\sum_j r_{kj}^2}}$$

As in the case of Pearson Correlation Similarity, the summations over j are calculated over the l items for which *both* users u_i and u_k have expressed their opinions. $l \leq n$, where n represents the number of total items in the user-item matrix, R.

We select to use the Pearson Correlation Similarity metric in order to estimate the proximity among users, unless it is stated otherwise, since experiments have shown [8] that the Vector/Cosine Similarity measure does not perform as well in Recommender Systems.

An *mxm similarity matrix*, S, can now be generated, including the similarity values between all users. Specifically, the entry at the i-th row and the k-th column of matrix S, would correspond to the similarity between users u_i and u_k. Furthermore, the i-th row of similarity matrix, S, would represent the similarity between user u_i and all other users. Obviously, the diagonal entries of the similarity matrix, S, should be set to 0 so that a user cannot be selected as a neighbor of himself.

B.2 Neighborhood Generation

At this point of the recommendation process it is mandatory to distinguish a single user, called the *active user*, as the user for whom we would like to make predictions, and proceed with generating his neighborhood of users. Neighborhood Generation, which is based on the similarity matrix, S, can be implemented by several schemes:

Center-based Scheme

The *Center-based scheme* creates a neighborhood of size l for the active user, u_a, by simply selecting from the similarity matrix, S, and more specifically, from the row of matrix S which corresponds to the active user, those users who have the l highest similarity values with the active user.

Aggregate Neighborhood Scheme

The *Aggregate Neighborhood scheme* creates a neighborhood of users, not by finding the users who are closest to the active user, but by collecting the users who are closest to the centroid of the current neighborhood.

The Aggregate Neighborhood scheme forms a neighborhood of size l by first picking the user who is closest to active user, u_a. Those two users will now form the current neighborhood, and the selection of the next neighbor will be based on them. The procedure for the collection of the next $l-1$ neighbors resumes in the following manner: Assuming that at a certain point, the current neighborhood, N, consists of h users, where clearly $h < l$. The Aggregate Neighborhood scheme will compute the centroid, $\overrightarrow{C}$, of the current neighborhood as:

$$\overrightarrow{C} = \frac{1}{h} \sum_{j=1}^{h} u_j$$

Obviously, the centroid, $\overrightarrow{C}$, is calculated from the h users currently included in the neighborhood N, and it has the form of a vector, which is comparable to the vectors representing the users. A new user, u_k, who does not belong to the current neighborhood, $u_k \notin N$, will be selected as the next neighbor, only if he is the closest to the centroid $\overrightarrow{C}$. In that case, the neighborhood will now include $h+1$ users and the centroid should be recomputed for them. The process will continue until the size of the neighborhood becomes l, or $|N| = l$.

As we can conclude, the Aggregate Neighborhood scheme allows all users to affect the formation of the neighborhood, as they are gradually selected and added to it. Furthermore, this scheme can be proven to be beneficial in cases of very sparse user-item matrices.

C. Generation of Recommendation or Prediction

The final step in the recommendation process is to produce either a prediction, which will be a numerical value representing the predicted opinion of the active user, or a recommendation, which will be expressed as a list of the top-N items that the active user will appreciate more. In both cases, the result should be based on the neighborhood of users.

C.1 Prediction Generation

Prediction is a numerical value, pr_{aj}, which represents the predicted opinion of active user u_a about item i_j. It is a necessary condition that item i_j does not belong in the set of items for which the active user has expressed his opinion about. Also, the prediction generated by the Recommender System should be within the same, accepted numerical scale as all ratings in the initial user-item matrix, R.

Prediction generation requires that a user neighborhood, N, of size l is already formed for active user, u_a. A prediction score, pr_{aj}, on item i_j for active user, u_a, is then computed as follows:

$$pr_{aj} = \bar{r_a} + \frac{\sum_{i=1}^{l} (r_{ij} - \bar{r_i}) * sim_{ai}}{\sum_{i=1}^{l} |sim_{ai}|}$$

sim_{ai} represents the similarity, as expressed in the mxm similarity matrix, S, between the active user, u_a, and all users, u_i, for $i = 1, 2, ..., l$, belonging to the active user's neighborhood. It is natural to assume that from the l users in the active user's neighborhood, only those who have actually given their opinion on item i_j will be included in that sum.

An extension, called *Case Amplification* [8], can be applied as a transformation of the similarities used in the basic collaborative filtering prediction formula. Specifically, we transform the similarity factors, sim_{ai}, as follows:

$$sim'_{ai} = \begin{cases} sim^{\rho}_{ai}, \ if \ sim_{ai} \geq 0 \\ -(-sim^{\rho}_{ai}), \ if \ sim_{ai} < 0 \end{cases}$$

The transformation emphasizes similarities that are closer to 1, and downgrades similarities with lower values. A typical value for ρ is 2.5.

C.2 Top-N Recommendation Generation

Recommendation is a list of N items that the active user, u_a, will like the most. Items included in the generated list should not appear in the list of items already rated by the active user.

- *Most-Frequent Item Recommendation.* According to this recommendation scheme, we look into the neighborhood, N, of the active user and perform a frequency count of the items that each neighbor user has purchased or rated. After all neighbor users have been taken into account and the total counts for their rated items have been calculated, the system will exclude items already rated by the active user, sort the remaining items according to their frequency counts and return the N most frequent items, as the recommendation for active user, u_a. Most-Frequent Item Recommendation is commonly utilized in combination with the binary rating scheme.
- *Association Rule-based Recommendation.* Assuming that there are n items, $I = \{i_1, i_2, ..., i_n\}$, in the initial user-item matrix, R. A *transaction* $T \subseteq I$ is defined as a set of items that are rated or purchased together. An *association rule* between two sets of items, I_X and I_Y, such that $I_X, I_Y \subseteq I$ and $I_X \bigcap I_Y = \emptyset$, states that if items from set I_X are present in transaction T, then there is a strong probability that items from set I_Y would also be present in T. An association rule of that form is often denoted by $I_X \Rightarrow I_Y$. The quality of association rules is usually evaluated by calculating their *support* and *confidence*.

The support, s, of a rule measures the occurrence frequency of the rule's pattern $I_X \Rightarrow I_Y$. Rules with high support are important since they describe a sufficiently large population of items

The confidence, c, of a rule is a measure of the strength of implication $I_X \Rightarrow I_Y$. Rules with high confidence are important because their prediction of the outcome is normally sufficiently accurate.

Now that association rules have been defined, we can use them to generate a top-N Recommendation: For each of the l users who belong to the active user's neighborhood, we create a transaction containing all the items that they have purchased or rated in the past. Having as input these transactions, we can utilize an association rule discovery algorithm to find all the rules that satisfy some required minimum support and confidence constraints. To locate the top-N recommended items for active user u_a, we proceed with the following actions. First, we distinguish all the rules that the active user *supports*. By user support for a rule, we indicate that the active user has rated or purchased all the items in the left-hand side of that rule. Now we collect the set of unique items that are being predicted by the selected rules, and at the same time, have not yet been purchased by the active user. We can sort these items based on the *confidence* of the rules that were used to predict them. Items predicted by rules with higher confidence are ranked first. For items that appear in more than one rule, we select and use the rule with the highest confidence. Finally, we select the first N highest ranked items as the recommended set [1].

IV. Alternative Methods

A. Item-based Collaborative Filtering

A different approach in the area of filtering algorithms, that was suggested recently [14] [15], is based on item relations and not on user relations, as in classic Collaborative Filtering. In the Item-based Collaborative Filtering algorithm, we look into the set of items, denoted by I_{u_a}, that the active user, u_a, has rated, compute how similar they are to the target item i_j and then select the k most similar items $\{i_1, i_2, ..., i_k\}$, based on their corresponding similarities $\{s_{i1}, s_{i2}, ..., s_{ik}\}$. The predictions can then be computed by taking a weighted average of the active user's ratings on these similar items.

The first step in this new approach is the **Representation**. Its purpose is the same as with the classic Collaborative Filtering algorithm: represent the data in an organized manner. To achieve that, we only require an mxn user-item matrix, R, where element r_{ij} includes the rating that user u_i (row i from matrix R) gave to item i_j (column j from matrix R), or simply, a value of 1, if user u_i purchased item i_j, and 0 otherwise.

In the following step, the **Item Similarity Computation** should be calculated. The basic idea in that step is to first isolate the users who have rated two items i_j and i_k and then apply a similarity computation technique to determine their similarity. Various ways to compute that similarity have been proposed. We have already discussed Pearson Correlation Similarity and Cosine/Vector Similarity, when describing the classic Collaborative Filtering algorithm. We can apply the same ideas in Item Similarity Computation.

In the case of *Pearson Correlation Similarity*, the only difference is that we are not using the ratings that two users have provided for a common item, but the ratings that two items, i_j and i_k, whose similarity we want to calculate, have been given by a common user, u_i. As expected, the calculations are executed only over the l users, where $l \leq m$, who have expressed their opinions over *both* items:

$$sim_{jk} = corr_{jk} = \frac{\sum_{i=1}^{l}(r_{ij}-\bar{r_j})(r_{ik}-\bar{r_k})}{\sqrt{\sum_{i=1}^{l}(r_{ij}-\bar{r_j})^2 \sum_{i=1}^{l}(r_{ik}-\bar{r_k})^2}}$$

Cosine/Vector Similarity of items i_j and i_k is expressed by:

$$sim_{jk} = cos_{jk} = \sum_i \frac{r_{ij}}{\sqrt{\sum_i r_{ij}^2}} \frac{r_{ik}}{\sqrt{\sum_i r_{ik}^2}}$$

where r_{ij} and r_{ik} are the ratings that items i_j and i_k have been given by user u_i. Obviously, the summations over i are again calculated only for those l users, where $l \leq m$, who have expressed their opionions over *both* items.

Adjusted Cosine Similarity is a different way of calculating similarity of items. It is specifically used for Item-based Collaborative Filtering and attempts to offset the differences in rating scale between different users. Adjusted Cosine Similarity of items i_j and i_k is given by:

$$sim_{jk} = adjcorr_{jk} = \frac{\sum_{i=1}^{l}(r_{ij}-\bar{r_i})(r_{ik}-\bar{r_i})}{\sqrt{\sum_{i=1}^{l}(r_{ij}-\bar{r_i})^2 \sum_{i=1}^{l}(r_{ik}-\bar{r_i})^2}}$$

where r_{ij} and r_{ik} are the ratings that items i_j and i_k have received from user u_i, while $\bar{r}_i$ is the average of user's u_i ratings.

In case users do not provide explicit ratings for the items, they can be distinguished into those who purchased and those who have not purchased a specific item. The former are assigned a value of 1 while the latter are assigned a value of 0, leading to a binary rating scheme. Then, the similarity between a pair of items, i_j and i_k, can be based on the conditional probability of purchasing item i_j given that the other item, i_k, has already been purchased. Specifically, *Conditional Probability-Based Similarity* of items i_j and i_k, $cond(j|k)$, is expressed as the number of users that purchased both items i_j and i_k divided by the total number of users who purchased i_k.

$$sim_{jk} = cond(j|k) = \frac{Freq(jk)}{Freq(k)}$$

Here, $Freq(X)$ is the number of users who have purchased the items in the set X, thus, $Freq(jk)$ refers to the number of users who have purchased both items i_j and i_k. It should be clear that usually $cond(j|k) \neq cond(k|j)$, which leads to asymmetric relations. As a result, Conditional Probability-based Similarity should be used with caution.

A couple of suggestions on how to improve the quality of Conditional Probability-Based Similarity, regarding items being purchased frequently and users that tend to rate fewer items, thus being more reliable indicators, can be found in [15].

Once we have calculated the similarities between all items in the initial user-item matrix, R, the final step in the collaborative filtering procedure is to isolate the l items, i_k, with $k = 1, 2, ..., l$, that share the greatest similarity with item i_j, for which we want a prediction, form its neighborhood of items, N, and proceed with the **Prediction Generation**. The most common way to achieve that is through a *weighted sum*. Briefly, this method generates a prediction on item i_j for active user u_a by computing the sum of ratings given by the active user on items belonging to the neighborhood of i_j. Those ratings are weighted by the corresponding similarity, sim_{jk}, between item i_j and item i_k, with $k = 1, 2, ..., l$, taken from neighborhood N:

$$pr_{aj} = \frac{\sum_{k=1}^{l} sim_{jk} * r_{ak}}{\sum_{k=1}^{l} |sim_{ak}|}$$

B. Recommender Systems using Hybrid Methods

Hybrid methods that usually combine collaborative filtering techniques with content-based filtering algorithms will be introduced in the following sections. Such methods are utilized in order to realize the benefits from both approaches while at the same time minimize their disadvantages.

B.1 Content-Boosted Collaborative Filtering

The basic idea behind *Content-Boosted Collaborative Filtering* [6] is to use a content-based predictor to enhance existing user data, expressed via the user-item matrix, R, and then provide personalized suggestions through collaborative filtering. The content-based predictor is applied on each row from the initial user-item matrix, corresponding to each separate user, and gradually generates a *pseudo* user-item matrix, PR. At the end, each row, i, of the *pseudo* user-item matrix PR consists of the ratings provided by user u_i, when available, and those ratings predicted by the content-based predictor, otherwise:

$$pr_{i,j} = \begin{cases} r_{i,j}, \ if \ user \ u_i \ has \ rated \ item \ i_j \\ c_{i,j}, \ if \ user \ u_i \ has \ not \ rated \ item \ i_j \end{cases}$$

Clearly, $r_{i,j}$ denotes the actual rating of user u_i on item i_j, while $c_{i,j}$ denotes the rating generated by the content-based predictor.

The pseudo user-item matrix, PR, is a full dense matrix. We can now perform collaborative filtering using PR instead of the original user-item matrix R. The similarity between the active user, u_a, and another user, u_i, can be computed as in classic Collaborative Filtering. The only difference will be that when calculating similarities, instead of plugging in the original user ratings, we substitute with the ratings for u_a and u_i as they can be found in the pseudo user-item matrix.

B.2 Combining Content-Based and Collaborative Filters

This approach combines different filtering methods by first relating each of them to a distinct component and then basing its predictions on the *weighted average* of the predictions generated by those components [2]. In its simplest version, it includes only two components: one component generates predictions based on content-based filtering while the second component is based on the classic collaborative filtering algorithm. At the beginning, when the number of user ratings is limited and thus, adequate neighborhoods of similar users cannot be created, the content-based component is weighted more heavily. As the number of users is increased and more user opinions on items are collected, the weights are shifted more towards the collaborative filtering component, improving the overall accuracy of the prediction. It should be stressed that this approach is not exactly a *hybrid* approach, in the sense that both content-based and collaborative filtering components are kept separately, without affecting each other. This fact allows the system to benefit from individual advances occurring in either component, since there exist no interdependencies. Furthermore, this approach is more easily extensible to additional filtering methods by allowing each method to be added as a separate component which contributes to the weighted average with a separate weight.

Assuming that the system includes just a collaborative filter and a content-based filter, we can now discuss how the predictions output separately from the collaborative and the content-based components are combined into a single prediction at the end: It has been shown that a simple linear combination of scores returned by different Information Retrieval agents can improve the performance of those individual systems on new documents. We can utilize this idea by replacing the Information Retrieval agents with the filtering components of our system and the documents with items for which we want to generate predictions. Based on

these assumptions, we can combine the collaborative filtering prediction generated by the first component of our system, with the content-based prediction generated by the second component of our system, using a weighted average. The interesting part in that process would be to find the weights that result in the most accurate final prediction. In order to accomplish that, the proposed procedure is simple: We start by giving equal weight to both the collaborative and the content-based components. As users provide their ratings about various items, we collect the predictions generated by both components on these items and then calculate the mean absolute error (to be discussed extensively in the section of Evaluation Metrics) of those predictions when compared to the actual user ratings. Now we have to adjust the weights of each component separately, changing the contribution of the collaborative and the content-based filters in the final result, so as to minimize past error. The weights require quick adjustments at first, but as the number of ratings and predictions increase, the need for any weights' adjustment is gradually decreased.

C. Collaborative Filtering as a Machine Learning Classification Problem

Billsus and Pazzani propose a solution where they view Collaborative Filtering as a classification task [4]. Specifically, based on a set of user ratings about items, we try to induce a model for each user that would allow the classification of unseen items into two or more classes, each of them corresponding to different points in the accepted rating scale. If we agree on a binary rating scheme, then the set of probable classes, C, would include two class instances, $C_1 = like$ and $C_2 = dislike$.

Th initial purpose of this approach is to transform the data set of user ratings for items, corresponding to the initial user-item matrix, R, into a format where any supervised learning algorithm from the machine learning literature can be drawn and applied in order to induce a function of the form $f : U \rightarrow C$. The usefulness of this function is that it can classify any new, unseen item (taken from set U) to the appropriate class C.

To achieve such a data transformation, which would make our dataset suitable to be fed in a supervised learning algorithm, we have to implement the following steps during the *Training Stage* of the procedure:

- Starting with the original user-item matrix, R, associate one classifier with every user, u_i, where $i = 1, 2, ..., m$. If user u_i has rated item i_j with r_{ij}, then we can define a training example I_{ij} like this:

$$I_{ij} = \{r_{1j}, r_{2j}, ..., r_{mj} | C = r_{ij}\}$$

Clearly, the feature values used, r_{kj}, with $k = 1, 2, ..., m$ and $k \neq i$, are the ratings that all users except for u_i have given on item i_j, while r_{ij}, the rating of user u_i on item i_j, is utilized as the class label of the training example.

- Since users are not required to provide their opinion on all items and as a result, many feature values in the training examples we just described would be missing. If the learning algorithm cannot handle missing feature values, we have to apply some kind of transformation: Every user, whose previous representation in the training example was simply his rating on the item in mind, will now be represented by up to r Boolean features, where r is the number of points in the utilized rating scale. The resulting Boolean features, corresponding to each user u_k, with $k = 1, 2, ..., m$ and $k \neq i$, are of the form "User's u_k rating was ri", where $0 < ri \leq r$. If we denote feature "User's u_k rating was ri" with the more compact "$r_{u_k} = ri$", then the new *extended training example*, E_{ij}, representing the fact that user u_i has rated item i_j with r_{ij}, would be the following:

$$E_{ij} = \{r_{u_1} = 1, r_{u_1} = 2, ..., r_{u_1} = r, r_{u_2} = 1, ..., r_{u_m} = r | C = r_{ij}\}$$

The first r features correspond to user u_1 rating item i_j with 1,2,...,r, the next r features correspond to user u_2 rating item i_j with 1,2,...,r, and so on. We can now assign Boolean feature values to all these new features: Clearly, feature "User's u_k rating was ri" would take a boolean value of 1, in case user u_k has given item i_j a rating of ri, and a boolean value of 0, otherwise. From the r features corresponding to each user, only one feature would be assigned a value of 1, while the rest would be assigned a value of 0, since there is a single rating for each item. At the end, this procedure would result to n_i training examples of that form for user u_i, one for each item he has rated. Those training examples can now be collected into a single training matrix, T_i, which can be used in the following stages of the algorithm. At this point, we have to note that it is necessary to generate distinct training matrices, T_i, for every user, $i = 1, 2, ..., m$, for whom we want to produce personal predictions.

- This representation certainly will lead to an excessive number of features. Therefore, we need to apply some preprocessing steps before we can feed the training matrix into a supervised learning algorithm. Specifically, we need to remove all unnecessary features, compute the SVD of the resulting matrix, and select k as the number of dimensions to retain, obtaining a kxk matrix, $S_{k,i}$, along with matrices $U_{k,i}$ and $V_{k,i}$.

After those transformations are finished with, the training of the models, which will enable the classification of new, unseen items for users u_i, with $i = 1, 2, ..., m$, is completed. It is important to note that different models should be trained for different users. Now we can proceed with the *Prediction Stage*, where we will single out a certain user, called the active user, u_a, and generate a prediction of the rating that he would have given on a specific item, i_j. This can be achieved only if the item, for which we wish a prediction, can be represented in the form of the extended training examples, which we utilized in order to train the Machine Learning algorithm. The only difference in the representation used in the prediction stage will be that the class label of the example will be missing, since that is exactly what we want to predict: the class that item i_j belongs to, which represents the rating that it would have been awarded by user u_a. Thus, the representation of i_j as an example ready to be fed in the trained Machine Learning algorithm, in order to generate its class prediction, is the following:

$$v_j = \{r_{u_1} = 1, r_{u_1} = 2, ..., r_{u_1} = r, r_{u_2} = 1, ..., r_{u_m} = r\}$$

Before we can feed example v_j, which represents item i_j, in the trained supervised learning algorithm, we first need to scale it into the k-dimensional space:

$$v_{k,j} = v_j^T U_{k,a} S_{k,a}^{-1}$$

As we can see, the singular vectors from matrix $U_{k,a}$ and the singular values from matrix $S_{k,a}$, both generated during the Training Stage for active user, u_a, are used in that scaling.

Finally, the resulting real-valued vector, $v_{k,j}$, can be utilized as input into the selected Machine Learning algorithm. The output will represent the final prediction of the rating that active user, u_a, would have given on item i_j.

D. Use of SVD/LSI for Prediction Generation

In a previous section we discussed how we can utilize dimensionality reduction techniques, and specifically SVD/LSI, in order to make the original user-item matrix denser. Since the application of SVD/LSI techniques aimed strictly to the reduction of the sparsity of user-item matrix, R, when the Representation stage was over, we resumed with conventional filtering techniques in the succeeding stages of Neighborhood Formation and Prediction Generation. Yet, it is possible to expand the application of SVD/LSI throughout the recommendation process [3]. We will now see how SVD/LSI techniques can be utilized in a Recommender System, finally leading us to the prediction of a rating for active user, u_a, on item i_j:

1. Steps 1 to 4 remain the same as in the Use of Dimensionality Reduction Techniques which were described in the Preprocessing section of classic Collaborative Filtering. Those steps will lead to matrix products $U_k \cdot \sqrt{S_k}^T$ and $\sqrt{S_k} \cdot V_k^T$, that represent m pseudo-users and n pseudo-items in the k dimensional feature space.
2. Compute the similarity between the active user, u_a, and item i_j, by calculating the inner product between the a-th row of $U_k \cdot \sqrt{S_k}^T$ and the j-th column of $\sqrt{S_k} \cdot V_k^T$. The inner product calcuation can be achieved with the help of the Vector/Cosine Similarity metric.
3. In the final step, generate the prediction pr_{aj}, which corresponds to the *expected* rating of active user for item i_j, by simply adding the row average $\bar{r_a}$ to the similarity calculated in the previous step. Formally, $pr_{aj} = \bar{r_a} + U_k \cdot \sqrt{S_k}^T(a) \cdot \sqrt{S_k} \cdot V_k^T(a)$.

As we can see, the first difference when using SVD/LSI in order to generate predictions, rather than merely to make matrix R denser, is located in step 2. There, the similarity between user u_a and item i_j is computed. This comes in contrast with the similarity calculation of the conventional Collaborative Filtering algorithm, where the similarity computation involved only users. The second difference lies in step 3, where the final prediction regarding the rating of active user, u_a, for item i_j, is generated. Clearly, there is no weighted average calculation as in the case of the classic Collaborative Filtering algorithm.

E. Model-based Methods

If we decide to view Recommender Systems from a probabilistic perspective, then the collaborative filtering task can be thought of as a simple calculation of the expected value of a rating from a user on an item, given what data we have collected about that user. For the active user, u_a, we wish to predict ratings on items, i_j, where $j = 1, 2, ..., n$, for which he has not expressed his opinion yet. Assuming that ratings follow a numerical scale from 1:bad to r:excellent, we have:

$$pr_{aj} = E(r_{aj}) = \sum_{ri=1}^{r} Pr(r_{aj} = ri | r_{ak}, k \in I_{u_a})$$

where the probability expression is the probability that the active user, u_a, will give the particular rating value, ri, for item i_j, given his already reported ratings on other items, denoted by r_{ak}, where $k \in I_{u_a}$.

In the following sections we will examine alternative probabilistic models for collaborative filtering.

E.1 Personality Diagnosis

Personality Diagnosis was proposed in [17]: Each user, u_i, with $i = 1, 2, ..., m$, who has expressed his opinion about n_i items, where $n_i \leq n$, is assigned a *personality type*, which can be described as a vector of the following form:

$$R_i^{true} = \{r_{i1}^{true}, r_{i2}^{true}, ..., r_{in_i}^{true}\}$$

where vector values r_{ij}^{true}, with $j = 1, 2, ..., n_i$, correspond to the *true* ratings of user u_i on the n_i items for which he has expressed his opinion. One can realize the existence of a crucial distinction between *true* ratings and *reported* ratings. The true ratings encode the user's underlying, internal opinion for items, and are not directly accessible by the Recommender System. On the other hand, the reported ratings are those that users provide to the Recommender System. We can assume that the ratings that users tend to report to the Recommender System include Gaussian noise, meaning that the same user may report different ratings on different occasions, depending on the context. Statistically, we can claim that user's u_i *reported* rating for item i_j is drawn from an *independent normal distribution with mean* r_{ij}^{true}, where r_{ij}^{true} obviously represents the *true* rating of user u_i on item i_j:

$$Pr(r_{ij} = x | r_{ij}^{true} = y) \propto e^{-(x-y)^2/2\sigma^2}$$

σ is a free parameter, x is the rating value that the user has reported to the Recommender System, and y is the true rating value that user u_i would have reported if there was no noise involved. Given the user's personality type, his ratings are thought to be independent.

Let's further assume that the distribution of personality types in the user-item matrix, R, is representative of the distribution of personalities in the actual target population of users. Based on this assumption, we can conclude that the prior probability $Pr(R_a^{true} = v)$, that the active user, u_a, rates items according to rating vector v is given by the frequency that other users rate according to v. By simply defining R_a^{true} to be a random variable that can take one

of m possible values, R_1, R_2, ..., R_m, each with probability $1/m$, then:

$$Pr(R_a^{true} = R_i) = 1/m$$

Each of the m possible values, R_1, R_2, ..., R_m, corresponds to the reported rating vector of user u_i, where $i = 1, 2, ..., m$ and necessarily $i \neq a$.

Combining the two previous equations, we can apply Bayes' Rule, and compute the probability that the active user, u_a, is of the same personality type as any other user, u_i, given his reported (including Gaussian noise) ratings:

$$Pr(R_a^{true} = R_i | r_{a1} = x_1, ..., r_{an} = x_n) \propto Pr(r_{ai} = x_1 | r_{a1}^{true} = r_{i1}) \cdots Pr(r_{an} = x_n | r_{an}^{true} = r_{in}) \cdot Pr(R_a^{true} = R_i)$$

After computing this quantity for each user, u_i, we can calculate the probability distribution for the active user's rating of an unseen item i_j. This probability distribution corresponds to the prediction, pr_{aj}, that is generated by the Recommender System and represents the expected rating of active user, u_a, for item i_j:

$$pr_{aj} = Pr(r_{aj} = x_j | r_{a1} = x_1, ..., r_{an} = x_n) = \sum_{i=1}^{m} Pr(r_{aj} = x_j | R_a^{true} = R_i) \cdot Pr(R_a^{true} = R_i | r_{a1} = x_1, ..., r_{an} = x_n)$$

Personality Diagnosis can be thought of as a clustering method with *exactly* one user per cluster, since each user is represented by a single personality type, and we are attempting to link the active user with one of those personality types-clusters. Also, the implemented approach can be depicted as a naive Bayesian network, with the structure of a classical diagnostic model: we observe *ratings* and compute the probability that each *personality* type is the cause for those ratings. Ratings can be thought of as the "symptoms", while personality types can be thought of as "diseases", leading to those "symptoms" in the diagnostic model.

E.2 Bayesian Network Model

This approach, introduced at [8], is based on the utilization of a distinct Bayesian network for every user for whom we wish a prediction. Initially, we construct a Bayesian network with separate nodes representing different items from the original user-item matrix, R. The states of each node refer to the possible rating values for the corresponding item. We also need to introduce an extra state corresponding to cases where the user, to whom the Bayesian network refers to, has not expressed his opinion on certain items.

Now we can single out a specific user, active user u_a, and using as input the ratings he has provided for all items, apply an algorithm for learning Bayesian network models on his corresponding Bayesian network. The learning algorithm will search over various model structures in terms of dependencies for each item. In the network which will be selected, each item will have a set of parent items that are the best predictors for its rating values. Each conditional probability table attached to the nodes, will be represented by a decision tree encoding the conditional probabilities for that node.

V. Categorization of Recommender Systems' Algorithms

In this section we will attempt to divide the already discussed filtering algorithms into distinctive categories based on various criteria, all of which are mentioned in the related bibliography.

Memory-based vs. Model-based filtering algorithms

Model-based collaborative filtering algorithms usually take a probabilistic approach, envisioning the recommendation process as the computation of the expected value of a user rating, when his past ratings on other items are given. They achieve that by developing a model of user ratings, sometimes referred to as the *user profile*. The development of such a model is primarily based on the original user-item matrix, R, but once the model is trained, matrix R is no longer required for recommendation generation. The advantage of this approach is that, because of the much more compact user model, it avoids the constant need of a possibly huge user-item matrix to make recommendations. This would certainly lead to systems with lower memory requirements and rapid recommendation generation. Nevertheless, the model building step, which is equivalent to the neighborhood formation step in plain Collaborative Filtering algorithms, is executed off-line since the user model is expensive to build or update. As a result, it is recomputed only after sufficient changes have occurred in the user-item matrix, for example, once per week.

Model-based filtering algorithms include Collaborative Filtering as a Machine Learning Classification Problem [4], Personality Diagnosis [17] and Bayesian Network Model [8].

On the other hand, Memory-based collaborative filtering algorithms are basing their predictions on the original (or probably reduced, through statistical methods like SVD/LSI) user-item matrix, R, which they keep in memory throughout the procedure. This results in greater memory requirements and probably not so fast recommendations. Yet, the predictions are always in agreement with the most current user ratings. There is no need for off-line updating, which would probably have caused a performance bottleneck.

Memory-based filtering algorithms include the basic Collaborative Filtering algorithm [1], Item-based Collaborative Filtering [14] and the Algorithm using SVD/LSI for Prediction Generation [3].

Correlation-based vs. Machine Learning based algorithms

Billsus and Pazzani attempt, through their work described in [4], to transform the formulation of the recommendation problem, as viewed by the classic Collaborative Filtering algorithm, into a Machine Learning problem, where any supervised learning algorithm can be drawn and applied. They are based on the assumption that while correlation-based approaches seem to work well in the specific domain, they solve the recommendation problem in a rather unconventional way, which is not necessarily supported by sound theory. So, they select to see collaborative filter-

ing as a classification task and employing Artificial Neural Networks, among other alternative solutions, they induce a user model able to classify unseen items into classes, which correspond to points in the rating scale.

The use of Machine Learning methods in Collaborative Filtering is limited despite the fact they could possibly result to theoretically-well founded algorithms, which at the end may outperform conventional correlation-based approaches. From the rest of the filtering algorithms discussed, only the Bayesian Network Model [8] is basing its results on techniques drawn from the Machine Learning literature. Specifically, it employs the paradigms of Bayesian Networks and Decision Trees.

Hybrid Algorithms vs. Plain Algorithms

Hybrid Algorithms are those that draw ideas from different filtering paradigms in order to improve on the results achieved when the selected paradigms are employed separately:

Content-Boosted Collaborative Filtering utilizes Content-based Filtering to fill in the missing ratings from the initial user-item matrix, R. It then employs classic Collaborative Filtering techniques to reach a final prediction.

The *Weighted Combination of Content-based and Collaborative Filtering* defines two distinct filtering components. The first component implements plain Collaborative Filtering, while the second component implements Content-based Filtering. The final rating prediction is calculated as a weighted sum of those components, where the applied weights are decided by how close is the prediction of each component to the actual rating.

Collaborative Filtering as a Machine Learning problem utilizes a supervised learning algorithm on specially formulated data, whose representation dimension has been reduced after the application of LSI/SVD.

Numerical Algorithms vs. Non-Numerical Algorithms

This categorization simply distinguishes those filtering algorithms which are employing some kind of numerical method in order to generate their predictions. The importance in the use of a numerical method lies in the fact that it is based on mathematical foundation and not on some ad-hoc technique. Such algorithms are:

- *Collaborative Filtering as a Machine Learning Classification Problem* and *Collaborative Filtering using LSI/SVD*, where LSI/SVD is utilized to reduce the dimensions of the original representation.
- *Personality Diagnosis*, where theory of probabilities and Bayes' Rule are employed in order to calculate the required probability distributions.

VI. Evaluation Metrics

A number of different measures have been proposed and utilized in order to evaluate the performance of the various filtering algorithms employed by Recommender Systems. Systems that generate predictions should be considered separately from systems whose output is a top-N recommendation, and for that reason distinct evaluation schemes should be used in each case. Our discussion on evaluation metrics will be divided into two categories. First, a couple of metrics which refer to the goodness or the badness of the output will be presented. Second, metrics that evaluate the performance of Recommender Systems in terms of time and space requirements will be introduced.

A. Metrics evaluating Prediction Quality

A prediction is a numerical value which according to the Recommender System corresponds to the rating, r_{aj}, active user, u_a, would have assigned to unrated item, i_j. We usually evaluate Recommender Systems that generate predictions in two dimensions: *accuracy* and *coverage*.

A.1 Accuracy

Two general approaches, one based on *statistical accuracy* and the other based on *decision suppport accuracy*, are used to measure the prediction accuracy:

Statistical Accuracy Metrics

Statistical Accuracy Metrics measure how close is the numerical value r_{ij} which is generated by the Recommender System and represents the expected rating of user u_i on item i_j, to the actual numerical rating, ar_{ij}, as entered by the same user for the same item. The most commonly used statistical accuracy metric is the *Mean Absolute Error (MAE)*.

Mean Absolute Error measures the deviation of predictions generated by the Recommender System from the true rating values, as they were specified by the user. The MAE is measured only for those items, for which user u_i has expressed his opinion. The count of these items is n_i, where $n_i \leq n$, since users are not required to express their preferences over all n available items. The predictions generated for those items are r_{ij}, for $j = 1, 2, ..., n_i$, while the actual ratings provided from the user are denoted by ar_{ij}, for $j = 1, 2, ..., n_i$. Then, the MAE for user u_i is computed by first summing the absolute errors of the n_i corresponding actual ratings-prediction pairs and then computing their average. Formally,

$$MAE_i = \frac{\sum_{j=1}^{n_i} |ar_{ij} - r_{ij}|}{n_i}$$

The total MAE can be calculated by averaging the Mean Absolute Errors of all users, MAE_i, for $i = 1, 2, ...m$, over the total number of available users, m:

$$MAE = \frac{\sum_{i=1}^{m} MAE_i}{m}$$

Lower Mean Absolute Errors correspond to Recommender Systems that predict more accurate user ratings.

Decision Support Accuracy Metrics

Decision Support Accuracy Metrics evaluate how effective the filtering algorithm employed in a Recommender System is at helping a user distinguish high quality items from the rest of the items. These metrics are based on the assumption that a binary rating scheme is preferred. Thus, in cases when a different numerical scale is utilized, we should agree

on an accepted conversion to the binary rating scale. Decision Support Accuracy metrics should be used in cases when we are not interested in the exact rating of a specific item, and instead, we simply want to find out if the user will like that item or not. The most commonly used Decision Support Accuracy metric is the *ROC Sensitivity*.

ROC Sensitivity, corresponding to the area under the *Receiver Operating Curve*, commonly referred to as the *ROC* curve, is a measure of the diagnostic power of a filtering algorithm. A ROC curve plots the *sensitivity* and the *specificity* of the filtering test, with respect to a separator variable X. More accurately, it plots *sensitivity* and *1-specificity*, obtaining a set of points by varying the *recommendation score threshold* above which a specific item will be accepted as a good item. The recommendation score threshold is represented by x. The ROC curve ranges from 0 to 1.

Sensitivity refers to the probability of a randomly selected *good* item actually being rated as good, and as a result being accepted by the filtering algorithm.

Specificity refers to the probability of a randomly selected *bad* item actually being rated as bad, and as a result being rejected by the filtering algorithm.

It is important to note that there exists a trade off between sensitivity and specificity. Any increase in sensitivity will be accompanied by a decrease in specificity.

The area under the ROC curve, corresponding to ROC sensitivity, increases as the filter is able to detect more good items and at the same time decline more bad items. An area of 1 represents the perfect filtering algorithms, while an area of 0.5 represents a random filter, which rates an item as good or bad by simply rolling a dice.

Obviously, in order to use ROC sensitivity as a metric to distinguish good from bad items, we necessarily have to either utilize a binary rating scheme, which already divides the items into those two categories, or transform the used rating scheme into a binary rating scheme. Then, the ROC sensitivity metric could be utilized as an indication of how effective the Recommender System is at suggesting highly rated items to the user as good ones, and correctly describe low rated items as bad ones.

A.2 Coverage

Coverage is a measure of the percentage of items for which a filtering algorithm can provide predictions. It is not a rare occurrence that a Recommender System will not be able to generate a prediction for specific items because of the sparsity in the data of the initial user-item matrix, R, or because of other restrictions, which are set during the Recommender System's execution. Such cases will lead to low coverage values. A low coverage value indicates that the Recommender System will not be able to assist the user with many of the items he has not rated, and for that reason the user will be forced to depend on other criteria to evaluate them. A high coverage value indicates that the Recommender System will be able to provide adequate help in the selection of items that the user is expected to enjoy more.

Assuming that n_i are the items for which user u_i has given a rating, and np_i is the number of those items for which the Recommender System was able to generate a prediction, where clearly $np_i \leq n_i$, then coverage can be formally expressed as:

$$Coverage = \frac{\sum_{i=1}^{m} np_i}{\sum_{i=1}^{m} n_i}$$

The total coverage is computed as the fraction of items for which a prediction was generated over the total number of items that all available users have rated in the initial user-item matrix, R.

B. Metrics evaluating top-N Recommendation Quality

In top-N Recommendation, the system outputs a list of N items that the user is expected to find interesting. The main focus in evaluating such systems is determining the value of that list, meaning that we want to find out whether the user would be interested in rating or purchasing some or all the items included in that top-N list. For that purpose, we are introducing a couple of evaluation metrics, one based on *recall-precision* and the second based on *expected utility*.

B.1 Recall-Precision related Measures

The most common practice in order to evaluate a top-N recommendation is to use two metrics widely used in Information Retrieval (IR), *recall* and *precision*.

For the case of systems that generate a top-N recommendation list, there is a need to slightly adjust the definitions of recall and precision from the standard way they are used in IR. In such Recommender Systems our goal is to retrieve a fixed number of N relevant items to be suggested as part of a list. To compute recall and precision, first we have to divide our data into two disjoint sets, the *training set* and the *test set*. The filtering algorithm employed by the system works only on the training set and generates a ranked list of recommended items, which we will refer to as the *top-N set*. The main goal is to scan through the test set, representing the portion of the initial data set which was not used by the Recommender System, and match items in the test set with items included in the generated top-N set. Items that appear in both sets will become members of a special set, called the *hit set*, if we decide to apply IR terminology. We can now define recall and precision for top-N recommendation systems in the following way:

- *Recall*, when referring to Recommender Systems, can be defined as the ratio of hit set size over the test set size:

$$recall = \frac{size\ of\ hit\ set}{size\ of\ test\ set} = \frac{|test \bigcap top-N|}{|test|}$$

- *Precision*, when referring to Recommender Systems, can be defined as the ratio of hit set size over the top-N set size:

$$precision = \frac{size\ of\ hit\ set}{size\ of\ top-N\ set} = \frac{|test \bigcap top-N|}{N}$$

Obviously, the denominator is equal to N since the size of the top-N set is N.

These two measures are clearly conflicting in nature. Increasing the size of number N, usually results in an increase of recall, while at the same time precision is decreased. But

since both measures are important in evaluating the quality of systems that generate top-N recommendations, we can combine them into a single metric, the $F1$ *metric*. The $F1$ *metric* is a widely used metric which assigns equal weight to both recall and precision:

$$F1 = \frac{2*recall*precision}{recall+precision}$$

$F1$ should be computed for each individual user and then, the avarage over all users would represent the score of the top-N recommendation list [3] [1].

B.2 Expected Utility related Measures

We wish to estimate the expected utility of a particular ranked list to a user. In order to achieve that we assume that the total expected utility is simply the probability of viewing a recommended item included in that top-N list, times the utility of that item.

The utility of item i_j is defined as the difference between user's, u_i, actual rating on that item, denoted by ar_{ij}, and the default or neutral rating in the domain, denoted by d. As for the probability for the item to be selected, we assume that each successive item in the list, i_j, for $j = 1, 2, ..., N$, starting from the top of the list and moving down, is less likely to be viewed by user u_i following an exponential decay. Combining those two assumptions, the expected utility of a ranked list of items for user u_i, over the N items included in the top-N recommended list, is:

$$EU_i = \sum_{j=1}^{N} \frac{max(r_{ij}-d,0)}{2^{(j-1)/(\alpha-1)}}$$

We should note that the items are listed in declining order. α is the *viewing halflife*. It is defined as the ranking of this specific item on the list, for which there exists a 50% chance that the user will actually review it. Normally, that ranking should have a value between 1, representing the top of the list, and N, representing the bottom of the list. A usual value for halflife is $\alpha = 5$.

The final score of the top-N recommendation list is calculated over the set of all m users included in the initial user-item matrix, R:

$$EU = 100 \frac{\sum_{i=1}^{m} EU_i}{\sum_{i=1}^{m} EU_i^{max}}$$

EU_i^{max} represents the maximum utility achieved for user u_i in case all viewed items had been at the top of the ranked list [8].

C. Metrics evaluating Performance

After discussing some quality metrics that measure the accuracy, the utility or the coverage of a Recommender System, we can define a couple of standard evaluation techniques that refer directly to the performance of the system. More specifically, we will talk about performance metrics related to the time and storage requirements of a Recommender System.

C.1 Response Time

Response Time is a widely used performance metric, which is utilized for various purposes and in different domains. In the case of Recommender Systems, it defines the time that elapsed between a user's stated request and the system's response to that request.

The user may request either a prediction or a top-N recommendation from the system at time t_1. The Recommender System will accept the request, process the input, and after a succesful completion of the required task, it will provide a response at time t_2, where necessarily $t_2 > t_1$. Then, the response time, t_r, for that system is defined as the difference between these two times. As a result, $t_r = t_2 - t_1$.

C.2 Storage Requirement

Another way of evaluating a filtering algorithm is based on its *storage requirements*. It is natural to expect from modern Recommender Systems to provide services that involve millions of users and items. Such counts of users and items though, would possibly force systems to their limits. For that reason, it may be wise to evaluate how these systems manipulate the space provided to them. Storage requirements are usually analyzed in two ways: by checking their *main memory* requirement, which represents the on line space usage of the system, and by checking their *secondary storage* requirement, which refers to the off line space usage of the system.

C.3 Computational Complexity

Typically, most filtering algorithms can be divided into two separate steps: First, there is a *model building* step, usually executed *off-line*, followed by a second *execution* step, which is always executed *on-line*. Preprocessing, Representation, Similarity Calculation and Neighborhood Generation, which appear in most discussed filtering algorithms, can be thought of as part of the off-line step. Prediction or Top-N Generation is the on-line step.

The off-line step is a crucial performance factor in the Recommendation Process, since it is always executed in the initiation of the procedure, where based on the original user-item matrix, R, different user models are constructed. The model building step is also necessarily executed during the Recommendation Process, when new user ratings appear, making it mandatory to update the user-item matrix in a procedure known as *Updating*. It is obvious that when the Updating procedure is delayed, all generated predictions do not reflect the latest user-item relations and may be incorrect.

It is vital for the algorithm performance to evaluate the computational complexity of the off-line step: There may exist algorithms that utilize theoretically sound mathematical or machine learning techniques in order to generate their results, but nonetheless require complex calculations, making the off-line step too expensive and time consuming to execute. At the same time, there may exist filtering algorithms which are based on ad-hoc methods, leading to

scientifically debated predictions, but those methods would normally require considerably less time and give faster responses. Clearly, there is a trade-off here which we should take into consideration when deciding on which filtering algorithms to employ.

VII. Conclusions and Future Plans

In this paper we attempted to make a brief but complete presentation of the most popular filtering algorithms employed in modern Recommender Systems. First, we formally analyzed the problem that a filtering algorithm is asked to solve. We also talked about the biggest challenges that it is required to tackle. Then we proceeded with a description of classic Collaborative Filtering along with ways which were proposed in the bibliography in order to improve the basic algorithm. Special mention was made to several Preprocessing methods. Not stopping there, we gave considerable space to alternative filtering techniques. Some of them were based on Machine Learning algorithms, others were inspired by some kind of numerical method, while others were simply attempts to combine already existing techniques. At this point we thought it was crucial to include a categorization of the previously described algorithms based on different criteria. Finally, we presented a number of utilized evaluation metrics, from which some were used to measure quality, while others to measure performance.

Our future plans include a series of experiments whose purpose will be to first implement and then understand the usability of the basic filtering algorithms we described here. Through these experiments we will be able to compare the performance of these methods and realize the conditions under which each of them generates better results. The final step will be to enhance and extend existing filtering algorithms by employing various intelligent techniques, preferably drawn from the Machine Learning literature.

References

[1] Bardul M. Sarwar, George Karypis, Joseph A. Konstan, and John T. Riedl, "Analysis of recommendation algorithms for e-commerce," in *Electronic Commerce*, 2000.

[2] Mark Claypool, Anuja Gokhale, Tim Miranda, Pavel Murnikov, Dmitry Netes, and Matthew Sartin, "Combining content-based and collaborative filters in an online newspaper," in *ACM SIGIR Workshop on Recommender Systems-Implementation and Evaluation*, Berkeley, CA, 1999.

[3] Bardul M. Sarwar, *Sparsity, Scalability, and Distribution in Recommender Systems*, Ph.D. thesis, University of Minnesota, 2001.

[4] Daniel Billsus and Michael J. Pazzani, "Learning collaborative filters," in *15th International Conference on Machine Learning*, Madison, WI, 1998.

[5] Bardul M. Sarwar, George Karypis, Joseph A. Konstan, and John T. Riedl, "Application of dimensionality reduction in recommender systems - a case study," in *ACM WebKDD 2000 Web Mining for E-Commerce Workshop*, 2000.

[6] Prem Melville, Raymond J. Mooney, and Ramadass Nagarajan, "Content-boosted collaborative filtering," in *ACM SIGIR Workshop on Recommender Systems*, New Orleans, LA, 2001.

[7] Andrew I. Schein, Alexandrin Popescul, Lyle H. Ungar, and David M. Pennock, "Methods and metrics for cold-start recommendations," in *ACM SIGIR-2002*, Tampere, Finland, 2002.

[8] John S. Breese, David Heckerman, and Carl Kadie, "Empirical analysis of predictive algorithms for collaborative filtering," in *Fourteenth Conference on Uncertainty in Artificial Intelligence*, Madison, WI, 1998.

[9] Bardul M. Sarwar, Joseph A. Konstan, Al Borchers, Jon Herlocker, Brad Miller, and John T. Riedl, "Using filtering agents to improve prediction quality in the grouplens research collaborative filtering system," in *Conference on Computer Supported Cooperative Work*, 1998.

[10] Nathaniel Good, J. Ben Schafer, Joseph A. Konstan, Al Borchers, Bardul M. Sarwar, Jon Herlocker, and John T. Riedl, "Combining collaborative filtering with personal agents for better recommendations," in *Conference of the American Association of Artifical Intelligence (AAAI-99)*, pp. 439–446.

[11] Gerard Salton and Christopher Buckley, "Term weighting approaches in automatic text retrieval," *Information Processing and Management*, vol. 24, no. 5, pp. 513–523, 1988.

[12] Michael W. Berry, Susan T. Dumais, and Gavin W. O'Brien, "Using linear algebra for intelligent information retrieval," *SIAM Review*, vol. 37, pp. 573–595, 1995.

[13] Paul Resnick, Neophytos Iacovou, Mitesh Sushak, Peter Bergstrom, and John Riedl, "Grouplens: An open architecture for collaborative filtering of netnews," in *ACM 1994 Conference on Computer Supported Cooperative Work*, New York, NY, 1994, pp. 175–186.

[14] Bardul M. Sarwar, George Karypis, Joseph A. Konstan, and John T. Riedl, "Item-based collaborative filtering recommendation algorithms," in *10th International World Wide Web Conference (WWW10)*, Hong Kong, 2001.

[15] George Karypis, "Evaluation of item-based top-n recommendation algorithms," in *CIKM 2001*, 2001.

[16] Lyle H. Ungar and Dean P. Foster, "Clustering methods for collaborative filtering," in *Workshop on Recommendation Systems at the 15th National Conference on Artificial Intelligence*, 1998.

[17] David M. Pennock, Eric Horvitz, Steve Lawrence, and C. Lee Giles, "Collaborative filtering by personality diagnosis: A hybric memory- and model-based approach," in *Sixteenth Conference on Uncertainty in Artificial Intelligence (UAI-2000)*, Morgan Kaufmann, Ed., San Francisco, 2000, pp. 473–480.