Scalable Online Top-N Recommender Systems

Alípio M. Jorge[1][2], João Vinagre[1], Marcos Domingues[3], João Gama[1][2]
Carlos Soares[1][2], Pawel Matuszyk[5], Myra Spiliopoulou[5]

[1] INESC TEC, LIAAD/CESE, Portugal
[2] Universidade do Porto, FCUP/FEP/FEUP, Portugal
[3] State University of Maringá, Department of Informatics, Maringá, Brazil
[4] Knowl. Manag. and Discovery, Otto-von-Guericke University, Magdeburg, Germany
`amjorge@fc.up.pt`

Abstract. Given the large volumes and dynamics of data that recommender systems currently have to deal with, we look at online stream based approaches that are able to cope with high throughput observations. In this paper we describe work on incremental neighborhood based and incremental matrix factorization approaches for binary ratings, starting with a general introduction, looking at various approaches and describing existing enhancements. We refer to recent work on forgetting techniques and multidimensional recommendation. We will also focus on adequate procedures for the evaluation of online recommender algorithms.

1 Introduction

"Recommender system" is a designation that is currently used in more than one sense. Nevertheless, it typically refers to an information system, or part of it, that assists users in making choices from a very large catalog of items [26]. Although quite common in e-commerce, recommendation capabilities appear in a great variety of applications and situations, including e-learning [15], health [8], human resource management [31] and public transports [19]. Central to each recommender systems is the filtering criterion that is applied to the whole collection of available items. This criterion can be defined in many different ways, depending on the approach and on the available information at the time of recommendation.

The impact of recommender systems is now clearly recognized by companies and society. Amazon, a company who had a major role in the popularization of modern automatic recommenders, acknowledges that 35% of consumer choices are driven by recommendations [20]. Netflix reports a considerably larger value of 75%, which is comprehensible given that the service is payed via a flat rate and the consumption of movies and tv series can be done in a more streaming fashion. It is also recognized that displayed recommendations can improve the reputation of items, even if the predicted rating is low [23]. Recommendations may reduce the variety of items that users consume (the filter bubble effect), but may also allow the discovery of novel music and books, timidly moving towards the *serendipity* effect. Recommendations may also promote a *rich get richer* phenomenon and make marketeers worry about getting noticed by algorithms rather than getting noticed by users [13].

The two central entities of a recommender system are *users* and *items*. The general setting is that users interact with items, are interested in new items and items are recommended to users. The algorithms that produce the recommendations rely on models of both users and items. The modeling of users is called *profiling* and is based on the selection of descriptive measurable dimensions. Among these we may use demographic information, user activity and existing interaction with items (e.g. user bought item, user viewed item description, user listened to music), user's social connections, textual reviews, numerical ratings, etc. Items are also modeled on the basis of item content, including meta-data, item textual description and, if available, images or audio. Any item usage data (user-item interactions) can also be used to characterize and profile items. In general, an automatic recommendation can have the form of a predicted score (estimation of how much a user likes an item), a *Top-N* list with the *N* most likely preferred items or even a structured suggestion such as trip with multiple stops.

In this paper we describe our contributions to the problem of Top-N recommendation exploring interaction data. We focus on online/incremental algorithms, as well as techniques that help deal with temporal issues and the exploration of multiple dimensions.

2 Profiling

It is costumary to divide the flavors of user and item profiling into *content based* and *usage based*. This results in *content based filtering* and *collaborative filtering*. An example of the former is: "If the user likes prize-winning novelists then recommend a book written by a Nobel prize of literature". In this case, characteristics of item content are used to produce the matching. In collaborative filtering, information is gathered from a set of users and their interaction with items. Since no content data has to be used, collaborative filtering recommenders do not have to possess any kind of capability of "understanding" the content and merely rely on the "wisdom of the crowd". Although this is, more often than not, an advantage, it can also be a source of the "tyranny of the average", making life harder to users in behavior niches. It also suffers more easily of difficulties in making recommendations to new users or in recommending new items, the so called *cold start* problems.

In collaborative filtering approaches, profiling a specific user can be reduced to collecting the interaction of that user with items. One possible interaction is the rating of items by the users. These *ratings* are values in a pre-defined range, e.g. one to five stars, and can be treated as continuous. In this case the profile of the user can be represented by a real valued vector with as many dimensions as items. Ratings express explicit opinions of users, require more effort from the user and are, for that reason, relatively hard to obtain. They provide, however, high quality information. Another possible interaction is an action of the user over an item that indicates some interest or preference of the user for that item. One common example of such an action is "user bought item". The profile of an user is then a *binary* vector with ones on the dimensions corresponding to preferred items. Another way of seeing the profile in this binary setting is as the set of items the user interacted with. The binary setting is also sometimes called *unary* since only positive information is collected. Binary data is typically much cheaper to obtain than ratings data since it can result from the normal interaction of users with items, as they browse the catalog or decide to buy. An intermediate possibility is to automatically infer the degree of preference of an user from the natural interactions with the items. For example, we can infer that a user likes a music track less or more given the number of times it is played. These are called *implicit ratings*.

3 Baseline collaborative filtering algorithms

Most state-of-the-art Collaborative Filtering (CF) algorithms are based on either neighborhood methods or matrix factorization. Fundamentally, these differ on the strategy used to process user feedback. This user feedback can be conceptually seen as a user-item matrix, known in most literature as the *ratings matrix*, in which cells contain information about user preferences over items.

3.1 Neighborhood-based algorithms

Neighborhood-based algorithms essentially compute user or item neighborhoods using a user defined similarity measure. Typical measures include the cosine and Pearson correlation [28]. If the rows of the user-item matrix R represent users, and the columns correspond to items, similarity between two users u and v is obtained by measuring the similarity between the rows corresponding to those users, R_u and R_v. Similarity between two items i and j can be obtained between the columns corresponding to those items R_i and R_j. Recommendations are computed by searching and aggregating through the user's or item's k nearest neighbors. The optimal value of k is data dependent and can be obtained using cross-validation. The main advantages of neighborhood methods are their simplicity and ease of implementation, as well as the trivial explainability of recommendations – user and item similarities are intuitive concepts.

User-based CF exploits similarities between users to form user neighborhoods. For example, given two users u and v, the cosine similarity between them takes the rows of the ratings matrix R_u and R_v as vectors in a space with dimension equal to the number of items rated by both u and v:

$$\mathrm{sim}(u,v) = \cos(R_u, R_v) = \frac{R_u \cdot R_v}{||R_u|| \times ||R_v||} = \frac{\sum_{i \in I_{uv}} R_{ui} R_{vi}}{\sqrt{\sum_{i \in I_u} R_{ui}^2} \sqrt{\sum_{i \in I_v} R_{vi}^2}} \tag{1}$$

where $R_u \cdot R_v$ represents the dot product between R_u and R_v, I_u and I_v are the sets of items rated by u and v respectively and $I_{uv} = I_u \cap I_v$ is the set of items rated by both users u and v. Other similarity measures can be used, such as the Pearson Correlation. Euclidean measures are typically not used, given the very high dimensionality of the problem.

To compute the rating prediction $\hat{R}_{ui}$ given by the user u to item i, an aggregating function is used that combines the ratings given to i by the subset $K_u \subseteq U$ of the k nearest neighbors of u – e.g. (2).

$$\hat{R}_{ui} = \frac{\sum_{v \in K_u} \mathrm{sim}(u,v) R_{vi}}{\sum_{v \in K_u} \mathrm{sim}(u,v)} \tag{2}$$

Similarity between items can also be explored to provide recommendations [28, 14]. One practical way of looking at it is simply to transpose the user-item ratings matrix and then apply the exact same method. The result of the training algorithm will be an item-item similarity matrix.

Neighborhood-based CF for binary data

The methods in Section 3.1 are designed to work with numerical ratings. Neighborhood-based CF for binary data can actually be seen as a special case of neighborhood-based CF for ratings, by simply considering $R_{ui} = 1$ for all observed (u, i) user-item pairs and $R_{ui} = 0$ for all other cells in the feedback matrix R. Both notation and implementation can be simplified with this. For example, the cosine for binary ratings can be written as:

$$\mathrm{sim}(u,v) = \cos(R_u, R_v) = \frac{\sum_{i \in I_{uv}} R_{ui} R_{vi}}{\sqrt{\sum_{i \in I_u} R_{ui}^2} \sqrt{\sum_{i \in I_v} R_{vi}^2}} = \frac{|(I_u \cap I_v)|}{\sqrt{|I_u|} \times \sqrt{|I_v|}} \tag{3}$$

where I_u and I_v the set of items that are observed with u and j, respectively.

The cosine formulation in (3) reveals that it is possible to calculate user-user similarities using simple occurrence and co-occurrence counts. A user u is said to co-occur with user v for every item i they both occur with. Similarly, in the item-based case, an item i is said to co-occur with item j every time they both occur with a user u.

Rating predictions can be as well made using (2). The value of $\hat{R}_{ui}$ will be a score between 0 and 1, by which a list of candidate items for recommendation can be sorted in descending order for every user.

3.2 Matrix factorization

Matrix Factorization recommendation algorithms are inspired by Latent Semantic Indexing (LSI) [3], an information retrieval technique to index large collections of text documents. LSI performs the Singular Value Decomposition (SVD) of large document-term matrices. In a recommendation problem, the same technique can be used in the user-item matrix, uncovering a latent feature space that is common to both users and items. The problem with SVD is that classic factorization algorithms, such as Lanczos methods, are not defined for sparse matrices, which raises problems on how to compute the factorization.

As an alternative to classic SVD, optimization methods have been proposed to decompose (very) sparse user-item matrices.

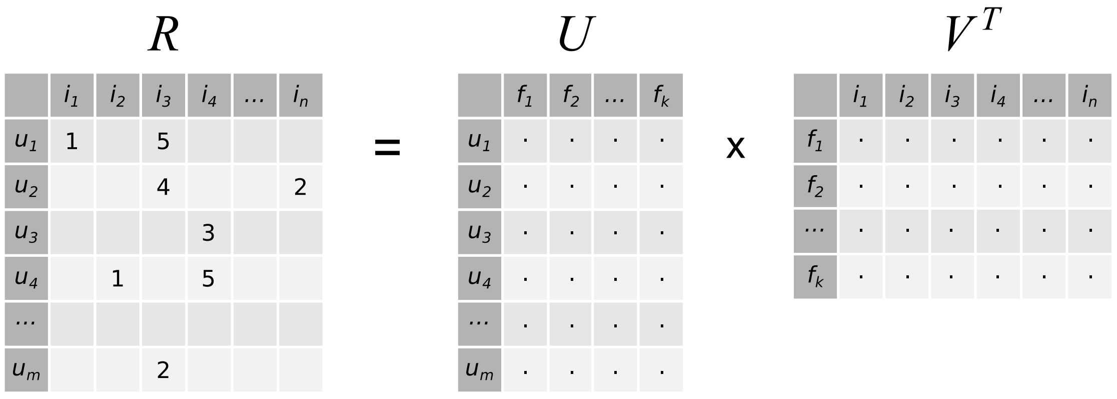

Fig. 1: Matrix factorization: $R = UV^T$.

Figure 1 illustrates the factorization problem. Supposing we have a user-item matrix $R_{m \times n}$ with m users and n items, the algorithm decomposes R in two full factor matrices $U_{m \times k}$ and $V_{n \times k}$ that, similarly to SVD, cover a common k-dimensional latent feature space, such that R is approximated by $\hat{R} = UV^T$. Matrix U spans the user space, while V spans the item space. The k latent features describe users and items in a common space. Given this formulation, the predicted rating by user u to item i is given by a simple dot product:

$$\hat{R}_{ui} = U_u \cdot V_i \tag{4}$$

The number of latent features k is a user given parameter that controls the model size.

The model consists of U and V, so the training task consists of estimating the values in U and V that minimize the prediction error on the known ratings. Training is performed by minimizing the L_2-regularized squared error for known values in R:

$$\min_{U.,V.} \sum_{(u,i) \in D} (R_{ui} - U_u \cdot V_i)^2 + \lambda_u ||U_u||^2 + \lambda_i ||V_i||^2 \tag{5}$$

In the above equation, D is the set of user-item pairs for which ratings are known and λ is a parameter that controls the amount of regularization. The regularization terms $\lambda||U_u||^2$ and $\lambda_i||V_i||^2$ are used to avoid overfitting. These terms penalize parameters with high magnitudes, that typically lead to overly complex models with low generalization power. For the sake of simplicity, it is common to use $\lambda = \lambda_u = \lambda_i$, which results in a single regularization term $\lambda(||U_u||^2 + ||V_i||^2)$. Stochastic Gradient Descent (SGD) is an efficient method to solve the optimization problem above. It has been informally proposed in [9] and many extensions have been proposed ever since [22, 11, 30, 27]. One obvious advantage of SGD is that complexity grows linearly with the number of known ratings in the training set, actually taking advantage of the high sparsity of R.

The algorithm starts by initializing matrices U and V with random numbers close to 0 – typically following a gaussian $\mathcal{N}(\mu, \sigma)$ with $\mu = 0$ and and small σ. Then, given a training set with tuples in the form (u, i, r) – the rating r of user u to item i –, SGD performs several passes through the dataset, known as iterations or epochs, until a stopping criterion is met – typically a convergence bound and/or a maximum number of iterations. At each iteration, SGD updates the corresponding rows U_u and V_i, correcting them in the opposite direction of the gradient of the error, by a factor of $\eta \leq 1$ – known as step size or learn rate. For each known rating, the corresponding error is calculated as $err_{ui} = R_{ui} - \hat{R}_{ui}$, and the following update operations are performed:

$$\begin{aligned} U_u &\leftarrow U_u + \eta(err_{ui}V_i - \lambda U_u) \\ V_i &\leftarrow V_i + \eta(err_{ui}U_u - \lambda V_i) \end{aligned} \tag{6}$$

3.3 Matrix factorization for binary data

The above method is designed to work with ratings. The input of the algorithm is a set of triples in the form (u, i, r), each corresponding to a rating r given by a user u to an item i. It is possible to use the same method with binary data by simply assuming that $r = 1$ for all cases [33]. This results in the following optimization problem:

$$\min_{U.,V.} \sum_{(u,i)\in D} (1 - U_u \cdot V_i)^2 + \lambda_u||U_u||^2 + \lambda_i||V_i||^2 \tag{7}$$

In the end, the predicted "ratings" $\hat{R}_{ui} = U_u \cdot V_i$ will be a value indicating a user's preference level for an item. This value can be used in a sorting function f to order a list of items:

$$f_{ui} = |1 - \hat{R}_{ui}| \tag{8}$$

In (8), f_{ui} measures the proximity of a predicted rating to 1. U_u and V_i are always adjusted to minimize the error with respect to 1, so it is natural to assume that the most relevant items for a user u are the ones that minimize f. Note that since we are not imposing non-negativity on the model, we need to use the absolute value of the difference in (8).

4 Incrementality

In real world systems, user feedback is generated continuously, at unpredictable rates, and is potentially unbounded – i.e. it is not assumed to stop coming. Moreover, the rate at which user activity data is generated can be very fast. Building predictive models on these continuous flows of data is a problem actively studied in the field of data stream mining [5].

One efficient way to deal with data streams is to maintain incremental models and perform on-line updates as new data points become available. This simultaneously addresses the problem of learning non-stationary concepts and computational complexity issues. However, this requires algorithms able to process data at least as fast as it is generated. Incremental algorithms for recommendation are not frequently addressed in the recommender systems literature, when compared to batch-learning algorithms for recommendation, which are much more thoroughly studied.

User feedback data in recommender systems shares all the characteristics of a data stream. It is continuously generated online, at unpredictable rates and the length of the data is potentially unbounded. Having this in consideration, it becomes clear that the batch approach to recommender systems has fundamental limitations.

Stream mining algorithms should be able to timely process streams, at the risk of not being able to keep up with the arrival rate of data elements. To apply this principle to recommender systems, we simply have to look at the recommendation problem as a data stream problem. This approach has several implications in the algorithms' design and evaluation – see Sec. 7. Regarding the algorithms' design and implementation, one practical way to deal with data streams is to use algorithms that are able to update models incrementally.

4.1 Incremental neighborhood methods

Classic neighborhood-based CF algorithms – user- and item-based – have been adapted to work incrementally. The main idea in both cases is to maintain the factors of the similarity function in memory, and update them with simple increments each time a new user-item interaction occurs.

In [21], Papagelis et al. propose an algorithm that incrementally updates the values in the user-user similarity matrix. When a user u rates an item, the similarity values between u and other users are obtained with increments to previous values. Using the Pearson Correlation, the factors of the similarity calculation between user u and another user v are split in the following way:

$$A = \frac{B}{\sqrt{C}\sqrt{D}} \tag{9}$$

Given the set I of items co-rated by both u and v, factors A, B and C correspond to the following terms:

$A = sim(u, v)$,
$B = \sum_{i \in I}(r_{u,i} - \bar{r}_u)(r_{v,i} - \bar{r}_v)$,
$C = \sum_{i \in I}(r_{u,i} - \bar{r}_u)^2$,
$D = \sum_{i \in I}(r_{v,i} - \bar{r}_v)^2$

It is intended to obtain the new similarity A' from B, C and D, and the new incoming rating:

$$A' = \frac{B'}{\sqrt{C'}\sqrt{D'}} \Leftrightarrow A' = \frac{B + e}{\sqrt{C + f}\sqrt{D + g}} \tag{10}$$

where e, f and g are increments that are easy to calculate after a rating arrives. The actual definitions of e, f and g depend on whether we are adding a new rating or updating an existing one, and we omit them here for the sake of clarity and space. Note that to incrementally update the similarities, the values of B, C and D for all pairs of users must be always available, which requires additional memory. Nevertheless, this simple technique allows fast online updates of the similarity values between the active user and all others.

Miranda and Jorge [18] study incremental algorithms using binary data. The incremental calculation of item-item cosine similarities can be based on user occurrence and co-occurrence counts. A user-user co-occurrence matrix F containing the number of items common to each pair of users can be kept. The diagonal of F contains the number of independent occurrences of each user – i.e. the number of items the user occurs with. Every time a new user-item pair (u, i) is observed in the dataset, the corresponding counts are incrementally updated. Using these counts, the similarities of user u with any other user v can be easily (re)calculated and stored.

$$S_{uv} = \text{sim}(u, v) = \frac{F_{uv}}{\sqrt{F_{uu}} \times \sqrt{F_{vv}}} \tag{11}$$

Alternatively, we can use this incremental approach in item-based algorithms.

4.2 Incremental matrix factorization

Early work on incremental matrix factorization for recommender systems is presented in [29], where Sarwar et al. propose a method to perform incremental updates of the Singular Value Decomposition (SVD) of the ratings matrix. This is a direct application of the Fold-in method [3], that essentially enables the calculation of new latent vectors (corresponding to new users or new items) based on the current decomposition and by appending them to the corresponding matrices. One shortcoming of this method is that it is applicable only to pure SVD, and it requires an initial batch-trained model.

Takács et al. address the problem of incremental model updates in [30] in a matrix factorization algorithm for ratings data. The idea is to retrain user features every time new ratings are available, but *only* for the active user(s), leaving item features unmodified, avoiding the whole process of batch-retraining the model. This method is a step forward in incremental learning, however it has the following limitations:

- The algorithm requires initial batch training;
- The whole ratings history R is required to update user features;
- Item features are not updated, and new items are not accounted for.

Both the above contributions still require batch learning at some point, that consists of performing several iterations over a learning dataset. While this may be an acceptable overhead in a static environment, it is not straightforwardly applicable with streaming data. As the number of observations increases and is potentially unbounded, repeatedly revisiting all available data eventually becomes too expensive to be performed online.

Fortunately, SGD is *not* a batch algorithm [12]. The only reason why several passes are made over a (repeatedly shuffled) set of data is because there is a *finite* number of examples. Iterating over the examples in different order several times is basically a trick to improve the learning process in the absence of fresh examples. If we assume – as we have to, in a data stream scenario – that there is a continuous flow of examples, this trick is no longer necessary. By this reasoning, SGD can – and should – be used online if enough data is available [2].

In [33], we propose ISGD, an algorithm designed to work as an online process, that updates factor matrices U and V based solely on the current observation. This algorithm, despite its formal similarity with the batch approach, has two practical differences. First, the learning process requires a single pass over the available data – i.e. it is essentially a memoryless algorithm, since there is no fundamental need to revisit past observations. Second, no data shuffling – or any other data pre-processing – is performed. Given that we are dealing with binary feedback we approach the boolean matrix R by assuming the numerical value 1 for *true* values. Accordingly, we measure the error as $err_{ui} = 1 - \hat{R}_{ui}$, and update the rows in A and B^T using the update operations in (6). Since we are mainly interested in top-N recommendation, we need to retrieve an ordered list of items for each user. We do this by sorting candidate items i for each user u

using the function $f_{ui} = |1 - \hat{R}_{ui}|$, where $\hat{R}_{ui}$ is the non-boolean predicted score. In plain text, we order candidate items by descending proximity to value 1.

One problem of ISGD is that the absence of negative examples leads to a model that converges globally to the positive class. Take the extreme trivial solution of making $\hat{R}_{ui} = 1$ for all u and i. In this case, $err_{ui} = 0$ in all cases, and no learning would be performed. In a more practical scenario, we would initialize U and V with values close to 0, which would enforce learning. Nevertheless, predictions will accumulate closer and closer around the target positive value. Eventually, the algorithm loses discriminative power, causing accuracy degradation.

We propose a solution for this problem in [34], using recency-based negative feedback imputation. The strategy is to introduce a mechanism that automatically selects *likely* negative examples. The intuition is that the items that have occurred the longest ago in the data stream are better candidates to be taken as negative examples for any user. These are items that no users have interacted with in the longest possible period of activity in the system. We maintain a global priority queue of items occurring in the stream, independently of the user. For every new positive user-item pair (u, i) in the data stream, we introduce a set $\{(u, j_1), \ldots, (u, j_l)\}$ of negative feedback consisting of the active – currently observed – user u and the l items j that are in the tail of the global item queue.

Incremental learn-to-rank

Learning to rank encompasses a set of methods that use machine learning to model the precedence of some entities over others, assuming that there is at natural ordering between them. The top-N recommendation task consist of retrieving the best ranked items for a particular user, so it is natural to approach the task as a learn-to-rank problem.

This is the approached followed by Rendle et al. in [25] with their Bayesian Personalized Ranking (BPR) framework. One shortcoming of this algorithm is that it is approached as a batch method. However, although not documented in the literature, the implementation available in the MyMediaLite[5] software library provides an incremental implementation of the algorithm. Essentially, the incremental version of BPR follows the same principle of ISGD, which is to process data points sequentially in one pass, enabling the processing of streaming data.

Another incremental algorithm for ranking that uses a selective sampling strategy is proposed by Diaz-Aviles et al. in [4]. The algorithm maintains a reservoir with a fixed number of observations taken randomly from a stream of positive-only user-item pairs. Every n^{th} pair in the stream is sampled to the reservoir with probability $|R|/n$, with $|R|$ being the number of examples in the reservoir. Model updates are performed by iterating through this reservoir rather than the entire dataset. The strategy is to try to always rely on the most informative examples to update the model.

5 Forgetting

One of the problems of learning from data streams is that the concepts being captured may not be stationary. In recommender systems, users preferences usually change over time. This means that an algorithm that correctly models user preferences in a certain time frame may not accurately represent the same users' preferences some time later. Incremental algorithms benefit from being constantly updated with fresh data, therefore capturing these changes immediately, however the model still retains the concepts learned from past data. One complementary way

[5] http://www.mymedialite.net/

to deal with this is to forget this outdated information, i.e. data that no longer represent the concept(s) being learned by the algorithm.

5.1 Forgetting for neighborhood-based incremental CF

In our past work [32] we have used fading factors to *gradually* forget user feedback data using neighborhood-based algorithms. We do this by successively multiplying by a positive scalar factor $\alpha < 1$ all cosine similarities between all pairs of users – or items, in an item-based algorithm – at each incremental step, before updating the similarities with the new observations. If we consider a symmetric similarity matrix S containing all similarity values between pairs of users – or pairs of items –, this is achieved using the update $S \leftarrow \alpha S$. The lower the value of α, the faster the forgetting occurs. In practice, two users – or two items – become less similar as they co-occur farther apart in time.

Our results show that this type of forgetting is beneficial for the algorithms' accuracy, especially in the presence of sudden changes.

5.2 Forgetting with factorization-based incremental CF

We have studied forgetting strategies for incremental matrix factorization algorithms in a collaboration with Matuszyk and Spiliopoulou [16]. To achieve forgetting we use a total of eleven forgetting strategies of two types: *rating-based* and *latent-factor-based*. The first performs forgetting of certain past ratings for each user, while the latter performs forgetting by readjusting the latent factors in the user factor matrix, diminishing the impact of past ratings.

Ratings-based forgetting generally consists of forgetting sets of ratings. Formally, it is a function that operates on the set of ratings R_u of a user u and generates a new set $R'_u \subseteq R_u$:

$$f : R_u \to R'_u$$

We proposed the following six rating-based forgetting methods:

- *Sensitivity-based forgetting*, based on sensitivity analysis. The idea is to forget the ratings that cause changes with higher-than-normal magnitude in the user model. The rationale is that these ratings are typically not representative of the user preferences and should therefore be forgotten. Practical examples of such ratings are the ones on items that are bought as gifts, or when some person uses someone else's account.
- *Global sensitivity-based forgetting.* Like the previous technique, it also forgets ratings that have an impact that falls out of the regular one. The difference is that the sensitivity threshold is measured globally instead of being personalized.
- *Last N retention.* Here the strategy is to retain the latest N ratings for each user. This acts as a sliding window over the ratings of each user with at most N ratings.
- *Recent N retention.* Similar to Last N retention, except that N corresponds to time, instead of a fixed number of ratings, retaining only the ratings that fall into the previous N time units.
- *Recall-based change detection.* This strategy detects sudden drops in the incremental measurement of Recall – i.e. downward variations above a certain threshold, which is maintained incrementally as well – and forgets all ratings occurring before the detected change. This is particularly helpful in environments where changes are abrupt.
- *Sensitivity-based change detection*[6]. This is similar to Recall-based change detection, except that the criterion for detecting a change is the impact of new ratings. If a certain rating

[6] The term *sensitivity* is used here with its broader meaning, not as a synonym of recall.

changes the user profile dramatically, we assume that the change is real – the user has actually changed preferences – and forget all past ratings.

Latent-factor-based forgetting operates directly on the factorization model, adjusting user or item latent factors in a way that it imposes some type of forgetting. These adjustments to latent factors are linear transformations in the form:

$$A_u^{t+1} = \gamma \cdot A_u^t + \beta$$
$$B_i^{t+1} = \gamma \cdot B_i^t + \beta$$

In the above equations, γ and β are dependent on one of the five strategies below:

- *Forget unpopular items.* This technique consists of penalizing unpopular items by multiplying their latent vectors with a factor proportional to their frequency in the stream.
- *User factor fading.* Here, user latent factors are multiplied by a positive factor $\gamma < 1$. This causes the algorithm to gradually forget user profiles, benefiting recent user activity and penalizing past activity.
- *SD-based user factor fading.* This technique also multiplies user factors by a scalar value, except that this value is not a constant, but rather depends on the volatility of user factors. Users whose factors are more unstable have a higher forgetting rate than those whose profiles are more stable.
- *Recall-based user factor fading.* Similarly to the previous strategy, users have differentiated forgetting factors. This technique amplifies the forgetting factor for users that have low Recall.
- *Forget popular items.* This is the opposite of “Forget unpopular items”. Frequent items are penalized as opposed to the non-frequent ones.

Using the BRISMF algorithm [30] as baseline, we implement and evaluate the above strategies on eight datasets, four of which contain positive-only data, while the other four contain numerical ratings.

Our findings in [16] show that forgetting significantly improves the performance of recommendations in both types of data – positive-only and ratings. Latent-factor-based forgetting techniques, and particularly “SD-based user factor fading”, have shown to be the most successful ones both on the improvement of recommendations and in terms of computational complexity.

6 Multidimensional recommendation

The data which are most often available for recommender systems are web access data that represent accesses from users to pages. Therefore, the most common recommender systems focus on these two dimensions. However, other dimensions, such as time and type of content (e.g., type of music that a page concerns in a music portal) of the accesses, can be used as additional information, capturing the context or background information in which recommendations are made in order to improve their performance. For instance, the songs recommended by a music web site (e.g., Last.fm) to a user who likes rock music should be different from the songs recommended to a user who likes pop music.

We can classify the multidimensional recommender systems into three categories [1]: pre-filtering, modeling and post-filtering. In pre-filtering, the additional dimensions are used to filter out irrelevant items before building the recommendation model. Modeling consists in using the additional dimensions inside the recommendation algorithms. In post-filtering, the additional dimensions are used after building the recommendation model to reorder or filter out recommendations.

One approach that combines pre-filtering and modeling, called **DaVI** (*Dimensions as Virtual Items*), enables the use of multidimensional data in traditional two-dimensional recommender systems. The idea is to treat additional dimensions as virtual items, using them together with the regular items in a recommender system [7]. Virtual items are used to build the recommendation model but they can not be recommended. On the other hand, regular items are used to build the model and they can also be recommended. This simple approach provides good empirical results and allows the use of existing recommenders.

The **DaVI** approach consists in converting each multidimensional session into an extended two-dimensional session. The values of the additional dimensions, such as "day=monday" are used as virtual items together with the regular items. The **DaVI** approach can also be applied to a subset of dimensions or even to a single dimension.

Once we convert a set of multidimensional sessions S' into a set of extended two-dimensional sessions S'', building/learning a multidimensional recommendation model consists in applying a two-dimensional recommender algorithm on S''. We illustrate the learning process using the **DaVI** approach in Figure 2, where the values of the additional dimension *Hour* are used as virtual items.

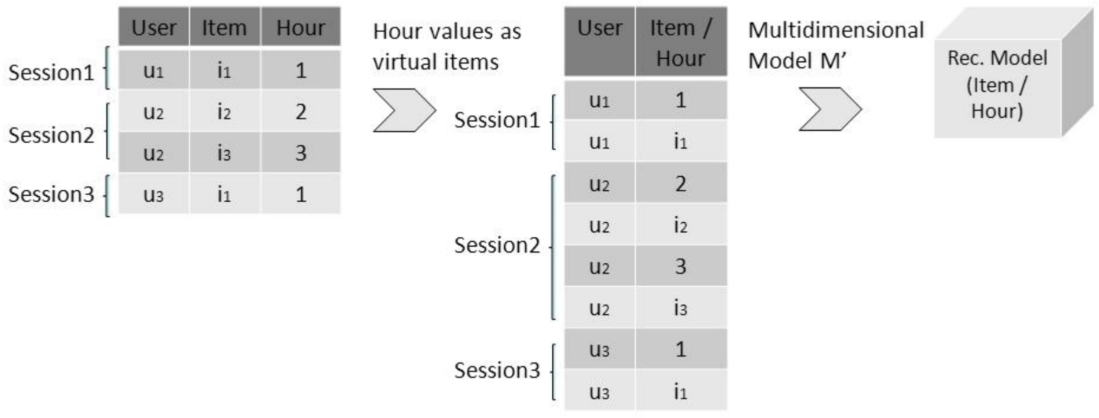

Fig. 2: Illustration of the learning process using the **DaVI** approach.

7 Evaluation

There are two main categories of evaluation procedures: offline and online. In the latter case, the recommender system is run on real users, and typically an A/B test is conducted [6]. In the former case the system is evaluated on archived data, where part is saved for training a model and the rest for testing the result. Offline evaluation is arguably not enough for assessing the power of recommendations [17]. Moreover, there is no guarantee that an algorithm with good offline results will have good online performance, from the users' perspective [24]. However, offline evaluation is important to assess the predictive ability and runtime performance of algorithms. It also has the advantages of being cheaper than online evaluation and of enabling repeatability.

7.1 Offline evaluation methodologies

Offline evaluation refers to evaluation methodologies using previously collected datasets, and conducted in a controlled laboratory environment, with no interaction with users. Offline protocols allow researchers to evaluate and compare algorithms by simulating user behavior. In the recommender systems literature, this typically begins by splitting the dataset in two subsets, the training set and testing set, picking random data elements from the initial dataset. The training set is used to train the recommender model. Evaluation is done by comparing the predictions of the model with the actual data in the test subset. Different protocols can be used. Generally, these protocols group the test set by user – or user session – and "hide" user-item interactions randomly chosen from each group. These hidden interactions form the hidden set. Rating prediction algorithms are usually evaluated by comparing predicted ratings with the hidden ratings. Item recommendation algorithms are evaluated performing user-by-user comparison of the recommended items with the hidden set.

Offline evaluation protocols are usually easy to implement, and enable the reproducibility of experiments, which is an important factor in peer-reviewed research. However they are typically designed to work with static models and finite datasets, and are not trivially applicable to streaming data and incremental models.

Prequential Evaluation

To solve the issue of how to evaluate algorithms that continuously update models, we have proposed a prequential methodology [10] to evaluate recommender systems. Evaluation is made treating incoming user feedback as a data stream. Evaluation is continuously performed in a test-then-learn scheme (Fig. 3): whenever a new rating arrives, the corresponding prediction is scored according to the actual rating. This new rating is then used to update the model.

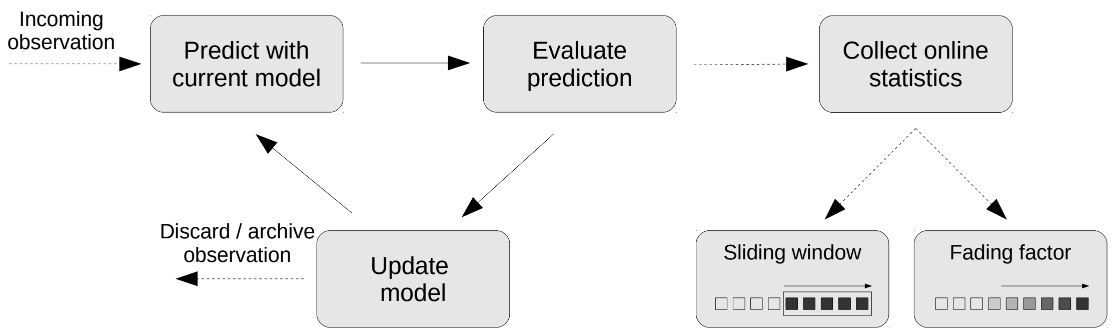
Incoming observation
Predict with current model
Evaluate prediction
Collect online statistics
Discard / archive observation
Update model
Sliding window
Fading factor

Fig. 3: Prequential evaluation

Take the example of a top-N recommendation task with binary data. In a binary data stream, observations do not contain actual ratings. Instead, each observation consists of a simple user-item pair (u, i) that indicates a positive interaction between user u and item i. The following steps are performed in the prequential evaluation process:

1. If u is a known user, use the current model to recommend a list of items to u, otherwise go to step 3;
2. Score the recommendation list given the observed item i;
3. Update the model with (u, i) (optionally);
4. Proceed to – or wait for – the next observation

One important note about this process is that it is entirely applicable to algorithms that learn either incrementally or in batch mode. This is the reason why step 3 is annotated as optional. For example, instead of performing this step, the system can store the data to perform batch retraining periodically.

This protocol provides several benefits over traditional batch evaluation:

– It allows continuous monitoring of the system's performance over time;
– Several metrics can be captured simultaneously;
– If available, other kinds of user feedback can be included in the loop;
– Real-time statistics can be integrated in the algorithms' logic – e.g. automatic parameter adjustment, drift/shift detection, triggering batch retraining;
– In ensembles, relative weights of individual algorithms can be adjusted;
– The protocol is applicable to both positive-only and ratings data;
– Offline experiments are trivially reproducible if the same data is available.

In an offline experimental setting, an overall average of individual scores can be computed at the end – because offline datasets are finite – and on different time horizons. For a recommender running in a production system, this process allows us to follow the evolution of the recommender by keeping online statistics of the metrics (e.g. a moving average of accuracy, or an error rate). Thereby it is possible to depict how the algorithm's performance evolves over time.

8 Conclusion

In this paper we presented an overview of recommender systems mostly centered on the work of our team. The main focus is on incremental approaches for binary data both to neighbourhood based and matrix factorization algorithms. We presented an incremental distance based collaborative algorithm where only an auxiliary co-frequency matrix is cached, besides the similarity matrix. We have also described a Stochastic Gradient Descent algorithm for binary matrix factorization that exploits negative feedback. A number of forgetting techniques that cope with the natural dynamics of continuously arriving data were mentioned. Another line of research is on multidimensional approaches, where we proposed the use of virtual items. Finally, we have described an offline prequential evaluation methodology adequate to incremental approaches.

Acknowledgements

(FourEyes) Project "NORTE-01-0145-FEDER-000020" is financed by the North Portugal Regional Operational Programme (NORTE 2020), under the PORTUGAL 2020 Partnership Agreement, and through the European Regional Development Fund (ERDF).

References

[1] Gediminas Adomavicius and Alexander Tuzhilin. Context-aware recommender systems. In Pearl Pu, Derek G. Bridge, Bamshad Mobasher, and Francesco Ricci, editors, *Proceedings of the 2008 ACM Conference on Recommender Systems, RecSys 2008, Lausanne, Switzerland, October 23-25, 2008*, pages 335–336, Lausanne, Switzerland, 2008.

[2] Léon Bottou. Stochastic learning. In Olivier Bousquet, Ulrike von Luxburg, and Gunnar Rätsch, editors, *Advanced Lectures on Machine Learning, ML Summer Schools 2003, Canberra, Australia, February 2-14, 2003, Tübingen, Germany, August 4-16, 2003, Revised Lectures*, volume 3176 of *Lecture Notes in Computer Science*, pages 146–168. Springer, 2003.

[3] Scott C. Deerwester, Susan T. Dumais, Thomas K. Landauer, George W. Furnas, and Richard A. Harshman. Indexing by latent semantic analysis. *JASIS*, 41(6):391–407, 1990.

[4] Ernesto Diaz-Aviles, Lucas Drumond, Lars Schmidt-Thieme, and Wolfgang Nejdl. Real-time top-n recommendation in social streams. In Padraig Cunningham, Neil J. Hurley, Ido Guy, and Sarabjot Singh Anand, editors, *Sixth ACM Conference on Recommender Systems, RecSys '12, Dublin, Ireland, September 9-13, 2012*, pages 59–66. ACM, 2012.

[5] Pedro M. Domingos and Geoff Hulten. Mining high-speed data streams. In Raghu Ramakrishnan, Salvatore J. Stolfo, Roberto J. Bayardo, and Ismail Parsa, editors, *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining, Boston, MA, USA, August 20-23, 2000*, pages 71–80. ACM, 2000.

[6] Marcos Aurélio Domingues, Fabien Gouyon, Alípio Mário Jorge, José Paulo Leal, João Vinagre, Luís Lemos, and Mohamed Sordo. Combining usage and content in an online recommendation system for music in the long tail. *IJMIR*, 2(1):3–13, 2013.

[7] Marcos Aurélio Domingues, Alípio Mário Jorge, and Carlos Soares. Dimensions as virtual items: Improving the predictive ability of top-n recommender systems. *Inf. Process. Manage.*, 49(3):698–720, 2013.

[8] L. Duan, W. N. Street, and E. Xu. Healthcare information systems: Data mining methods in the creation of a clinical recommender system. *Enterp. Inf. Syst.*, 5(2):169–181, May 2011.

[9] Simon Funk, 2006. http://sifter.org/ simon/journal/20061211.html [Accessed Jan 2013].

[10] João Gama, Raquel Sebastião, and Pedro Pereira Rodrigues. On evaluating stream learning algorithms. *Machine Learning*, 90(3):317–346, 2013.

[11] Yehuda Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In Ying Li, Bing Liu, and Sunita Sarawagi, editors, *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Las Vegas, Nevada, USA, August 24-27, 2008*, pages 426–434. ACM, 2008.

[12] Yann LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. Efficient backprop. In Genevieve B. Orr and Klaus-Robert Müller, editors, *Neural Networks: Tricks of the Trade, this book is an outgrowth of a 1996 NIPS workshop*, volume 1524 of *Lecture Notes in Computer Science*, pages 9–50. Springer, 1996.

[13] Dokyun Lee and Kartik Hosanagar. When do recommender systems work the best?: The moderating effects of product attributes and consumer reviews on recommender performance. In Jacqueline Bourdeau, Jim Hendler, Roger Nkambou, Ian Horrocks, and Ben Y. Zhao, editors, *Proceedings of the 25th International Conference on World Wide Web, WWW 2016, Montreal, Canada, April 11 - 15, 2016*, pages 85–97. ACM, 2016.

[14] Greg Linden, Brent Smith, and Jeremy York. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80, 2003.

[15] Nikos Manouselis, Hendrik Drachsler, Katrien Verbert, and Erik Duval. *Recommender Systems for Learning.* Springer Briefs in Electrical and Computer Engineering. Springer, 2013.

[16] Pawel Matuszyk, João Vinagre, Myra Spiliopoulou, Alípio Mário Jorge, and João Gama. Forgetting methods for incremental matrix factorization in recommender systems. In Wainwright et al. [35], pages 947–953.

[17] Sean M. McNee, John Riedl, and Joseph A. Konstan. Being accurate is not enough: how accuracy metrics have hurt recommender systems. In Gary M. Olson and Robin Jeffries, editors, *Extended Abstracts Proceedings of the 2006 Conference on Human Factors in Computing Systems, CHI 2006, Montréal, Québec, Canada, April 22-27, 2006*, pages 1097–1101. ACM, 2006.

[18] Catarina Miranda and Alípio Mário Jorge. Incremental collaborative filtering for binary ratings. In *2008 IEEE / WIC / ACM International Conference on Web Intelligence, WI 2008, 9-12 December 2008, Sydney, NSW, Australia, Main Conference Proceedings*, pages 389–392. IEEE Computer Society, 2008.

[19] Luís Moreira-Matias, Ricardo Fernandes, João Gama, Michel Ferreira, João Mendes-Moreira, and Luís Damas. On recommending urban hotspots to find our next passenger. In João Gama, Michael May, Nuno Cavalheiro Marques, Paulo Cortez, and Carlos Abreu Ferreira, editors, *Proceedings of the 3rd Workshop on Ubiquitous Data Mining co-located with the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013), Beijing, China, August 3, 2013.*, volume 1088 of *CEUR Workshop Proceedings*, page 17. CEUR-WS.org, 2013.

[20] Tien T. Nguyen, Pik-Mai Hui, F. Maxwell Harper, Loren Terveen, and Joseph A. Konstan. Exploring the filter bubble. *Proceedings of the 23rd international conference on World wide web - WWW '14*, pages 677–686, 2014.

[21] Manos Papagelis, Ioannis Rousidis, Dimitris Plexousakis, and Elias Theoharopoulos. Incremental collaborative filtering for highly-scalable recommendation algorithms. In Mohand-Said Hacid, Neil V. Murray, Zbigniew W. Ras, and Shusaku Tsumoto, editors, *Foundations of Intelligent Systems, 15th International Symposium, ISMIS 2005, Saratoga Springs, NY, USA, May 25-28, 2005, Proceedings*, volume 3488 of *Lecture Notes in Computer Science*, pages 553–561. Springer, 2005.

[22] A. Paterek. Improving regularized singular value decomposition for collaborative filtering. In *Proceedings of KDD Cup and Workshop*, volume 2007, pages 5–8, 2007.

[23] Pearl Pu, Li Chen, and Rong Hu. Evaluating recommender systems from the user's perspective: survey of the state of the art. *User Modeling and User-Adapted Interaction*, 22(4):317–355, 2012.

[24] Pearl Pu, Li Chen, and Rong Hu. Evaluating recommender systems from the user's perspective: survey of the state of the art. *User Model. User-Adapt. Interact.*, 22(4-5):317–355, 2012.

[25] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. BPR: bayesian personalized ranking from implicit feedback. In Jeff A. Bilmes and Andrew Y. Ng, editors, *UAI 2009, Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence, Montreal, QC, Canada, June 18-21, 2009*, pages 452–461. AUAI Press, 2009.

[26] Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor, editors. *Recommender Systems Handbook.* Springer, 2011.

[27] Ruslan Salakhutdinov and Andriy Mnih. Probabilistic matrix factorization. In John C. Platt, Daphne Koller, Yoram Singer, and Sam T. Roweis, editors, *Advances in Neural Information Processing Systems 20, Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 3-6, 2007*, pages 1257–1264. Curran Associates, Inc., 2007.

[28] Badrul M. Sarwar, George Karypis, Joseph A. Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In Vincent Y. Shen, Nobuo Saito, Michael R. Lyu, and Mary Ellen Zurko, editors, *Proceedings of the Tenth International World Wide Web Conference, WWW 10, Hong Kong, China, May 1-5, 2001*, pages 285–295. ACM, 2001.

[29] Badrul M. Sarwar, George Karypis, Joseph A. Konstan, and John T. Riedl. Application of dimensionality reduction in recommender system - a case study. In *WEBKDD'2000: Web Mining for E-Commerce – Challenges and Opportunities August 20, 2000, Boston, MA*, 2000.

[30] Gábor Takács, István Pilászy, Bottyán Németh, and Domonkos Tikk. Scalable collaborative filtering approaches for large recommender systems. *Journal of Machine Learning Research*, 10:623–656, 2009.

[31] Miguel Veloso, Alípio Jorge, and Paulo J. Azevedo. Model-based collaborative filtering for team building support. In *ICEIS (2)*, pages 241–248, 2004.

[32] João Vinagre and Alípio Mário Jorge. Forgetting mechanisms for scalable collaborative filtering. *J. Braz. Comp. Soc.*, 18(4):271–282, 2012.

[33] João Vinagre, Alípio Mário Jorge, and João Gama. Fast incremental matrix factorization for recommendation with positive-only feedback. In Vania Dimitrova, Tsvi Kuflik, David Chin, Francesco Ricci, Peter Dolog, and Geert-Jan Houben, editors, *User Modeling, Adaptation, and Personalization - 22nd International Conference, UMAP 2014, Aalborg, Denmark, July 7-11, 2014. Proceedings*, volume 8538 of *Lecture Notes in Computer Science*, pages 459–470. Springer, 2014.

[34] João Vinagre, Alípio Mário Jorge, and João Gama. Collaborative filtering with recency-based negative feedback. In Wainwright et al. [35], pages 963–965.

[35] Roger L. Wainwright, Juan Manuel Corchado, Alessio Bechini, and Jiman Hong, editors. *Proceedings of the 30th Annual ACM Symposium on Applied Computing, Salamanca, Spain, April 13-17, 2015.* ACM, 2015.