A Half-Life Decaying Model for Recommender Systems with Matrix Factorization

Panagiotis Ardagelou and Avi Arampatzis

Database & Information Retrieval research unit,
Department of Electrical & Computer Engineering,
Democritus University of Thrace, Xanthi 67100, Greece.
panosard93@gmail.com, avi@ee.duth.gr

Abstract. We propose the use of an exponential decay function for modeling drifts in user interests in collaborative filtering systems. For that purpose, we introduce the notion of half-life of ratings and incorporate it as a bias into a matrix factorization model. Experimental results on movie ratings spanning a period of approximately 7 months show that employing a half-life of around 150 days yields large improvements in prediction accuracy, confirming that significant user interest shifts exist over time and that the proposed model offers a viable strategy.

1 Introduction

The evolution of recommender or recommendation systems in the past decade has influenced highly our lives, even though many times we are not aware of their existence. Much of the online information which reaches us derives from various recommendation techniques. From the news we read and the products we buy, to the movies we watch and the music we listen to, most of our on-line input is significantly not random. In this way, both from the user perspective and from the service side as well, everyone can take advantage of high quality recommendations because, at first, they increase customer satisfaction, while they also allow much better quality of services (QoS) from companies' side, leading to much higher earnings. For these reasons, well-performing recommendations techniques are vital for achieving and conserving high degrees of interaction between companies (services) and customers [4].

In order to assess users' preferences, recommender systems need to have some kind of feedback from users, either implicit or explicit. As explicit feedback we refer to ratings given by users for specific items, based on the rating system each service uses. For example, as explicit feedback can be characterized a 'like', or a 5-stars rating system. Implicit information is based on user's actions which can reveal an interest of a user for a specific item. Such actions include browsing a certain product page, watching a video, etc., and can many times be proven very useful in order to increase the accuracy of recommendations [16].

While there is a variety of ways to produce recommendations, two are the main techniques. The first is content-based filtering which is crafting a user's profile based on known attributes of the items she has liked or bought. Content-based filtering is capable to provide straight-forward recommendations which are generally easier to understand and more self-explanatory than those created by other techniques, because it generates user profiles in isolation using only the user's history. The second technique is called collaborative, or memory-based, filtering. It differentiates from content-based techniques in that it does not build a user's profile in isolation but it also uses other users' likings; furthermore, it does not use any item attributes.

Collaborative filtering methods can generally be classified into two sub-categories, called neighborhood models and latent factor models. In this paper we deal only with latent factor models. In such models, the utility matrix is usually factorized using matrix factorization techniques, which will be described more thoroughly later in this paper. We chose to work with matrix factorization mainly for its ease in embedding external information in the modeling process. In addition, it generally provides very good prediction accuracy and relatively fast running times.

A milestone in the evolution of recommender systems is set by the well-known Netflix Prize Challenge which was held between 2007 and 2009 and has revolutionized our knowledge on recommender systems [3, 5]. Ever since, significant research has taken place and many more new versatile recommendation approaches have been brought to life. One of the main methods the winners of the Netflix Prize Challenge employed in their recommendation techniques was the embedding of temporal information, taking advantage of some temporal modeling of users' preferences [9].

In this paper, we are concerned with temporal modelling schemes. We consider explicit user feedback and time-dependant recommendation techniques using a matrix factorization model which takes full advantage of dimensionality reduction methods for very large and sparse matrices. We propose a novel half-life decaying modeling embedded into a matrix factorization process which takes advantage of the temporal information available in the dataset we use for our experimental purposes. In this respect, we generate a blend of recommendations based on the more recent likings of each specific user while not completely ignoring her past likings which still carry significant information about her preferences.

2 Related Work

Several past studies have shown that incorporating new information into modeling can lead to a boost in prediction accuracy. For example, some studies use external information from social networks in order to create social regularization models which capture the social relationships between users and provide recommendations based on each user's social neighborhood [13, 18]. While the social aspect is one important class of information which can boost the prediction accuracy of a recommender system, another one which has received increasing attention in the past years is time.

Taking the most advantage out of the temporal information available in the data has been proven to be of vital importance in the field of so-called time-aware recommender systems (TARS) as it improves the recommendation performance. One of the first temporal approaches which leads to increased recommendation accuracy sees the collaborative filtering task as a time-series problem, binning the data into different temporal bins [19]. Another one, based on implicit purchase data, takes into consideration the launch time of an item and the time it has been purchased to create recommendation based on the more recent purchases which reflect better the current preferences of a user

[11, 12]. Other studies, such as the one of the winning team of Netflix's competition [9], use external information in order to capture temporal effects within the data and create a more precise way of modeling users' preferences in time and items' popularity as well.

In this paper, within the general framework of collaborative filtering systems with temporal information, we introduce a new half-life decaying model by directly employing a weighting function. The idea comes from [1] where the authors introduced such a function into training user profiles with Rocchio in a content-based filtering context. In experiments with the OHSUMED corpus, a collection of medical abstracts spanning a period of five years, they found that effectiveness peaks when using a half-life of around 4 years on average in training. Nevertheless, further analysis revealed that there is a great distribution of optimal half-life values across topics. In our work, we adapt this idea to a collaborative filtering setup.

3 Matrix Factorization

A widely-used technique in many data mining and machine learning problems is dimensionality reduction. Broadly speaking, when too many features are available to describe a system's parameters, these features may form subsets and thus eliminate the need of referring to each specific one separately. By doing so, the computational cost decreases significantly and the whole process becomes more efficient as we end up from a high-dimensional space to a space of fewer dimensions.

In order to perform dimensionality reduction, we have to factorize the utility matrix. There are several ways to do so, with singular value decomposition and UV decomposition being two of the most common. We perform UV decomposition which means that we try to create two new thinner matrices of dimensions $m \times k$ and $n \times k$, so that the product of the first and the second transposed matrix gives the initial utility matrix. In our case, we do not want the product to reproduce exactly the initial utility matrix otherwise there will be no variation available in order to fill the initially blank entries, and thus, generate predictions. An exact reproduction would recreate the blank entries of the initial utility matrix and so there would be no predictions.

Let us denote an entry in the initial utility matrix, i.e. a known rating, as $r_{u,i}$, and the predicted value for this rating as $\hat{r}_{u,i}$. Furthermore, each user u is described by a vector $p_u \in \Re^k$ and each item i by a vector $q_i \in \Re^k$, where k is the dimensionality of the latent factors space which is always an integer and usually takes values between 5 and 200 [10]. The best number of latent factors is something vital for a model and varies between different implementations. The only way to discover the best value for the latent factors is by trial-and-error and there is no reason to assume the same values will fit in different models.

Vector p denotes how much interested or not is a specific user for each one of the total k latent factors. In a similar way, the item vector q shows the correlation of an item with each latent factor. In addition, the dot product of these two vectors captures the correlation between item i and user u, namely, the interest user u shows in items with similar characteristics to the item i and this gives us an approximation of the rating that the specific user would give on item i:

$$\hat{r}_{u,i} = p_u \cdot q_i^T \quad . \tag{1}$$

The main challenge in such models is how to decompose successfully the initial utility matrix. Once the decomposition is completed and the two derivative matrices are filled with the best possible values, we have reached a minimum in our system and we are ready to compute our final predictions. As said before, the utility matrix in almost all circumstances is very sparse, hence, only very few of the total entries are known.

For years in the field of recommender systems, it was common to pre-fill all the unknown entries with values using techniques such as zero-padding in a pre-processing step before decomposition. The problem with such approaches is that adding so much data in a large utility matrix can prove computationally very expensive. Furthermore, if the imputation is not done correctly, it can add significant amount of noise affecting tremendously the final predictions, lowering the accuracy.

Newer approaches suggest that instead of trying to fill the blank entries, we can model directly the observed ratings, however, we should be really careful with overfitting. Thus, the function that we have to minimize is the regularized squared error function, or also known as loss function [14, 15]:

$$\min_{p,q} \sum_{u,i \in \kappa} \left[(r_{u,i} - p_u \cdot q_i^T)^2 + \lambda(||p_u||^2 + ||q_i||^2) \right] \quad ,$$

where κ is the training set consisting of a percentage of the total known ratings. The term in the parentheses multiplied with a regularization parameter λ is called regularization term and is responsible for avoiding overfitting.

In order to perform the minimization of the regularized squared error function, it is common in matrix factorization models to use either the stochastic gradient descent algorithm or alternating least squares. Both options have their own pros and cons but here we choose to implement the stochastic gradient descent with regularization algorithm (SGD-WR) because it provides fast convergence for sparse matrices, achieves high prediction accuracy, and makes it easier to embed external information in our modeling.

The stochastic gradient descent algorithm works iteratively on each training instance creating a predicted rating $\hat{r}_{u,i}$ in each iteration. Then, the algorithm computes the prediction error and then applies the update rules shown below in order to perform the learning:

$$\mathrm{err}_{u,i} = r_{u,i} - \hat{r}_{u,i} = r_{u,i} - p_u \cdot q_i^T \quad , \tag{2}$$

$$q_i \leftarrow q_i + \gamma_1(\mathrm{err}_{u,i} p_u - \lambda_1 q_i) \quad , \quad p_u \leftarrow p_u + \gamma_1(\mathrm{err}_{u,i} q_i - \lambda_1 p_u) \quad .$$

Once the best fit has been reached, the algorithm has converged to a minimum. The minimum it converges depends on many attributes of the model, such as the learning rate as well as the initialization of matrices p and q before the optimization begins, thus fine-tuning is vital for good performance. The algorithm may possibly converge to the global minimum but this is the ideal case, as it usually converges to a local minimum since everything depends on the starting point.

3.1 Modeling Biases

Biases, from the user perspective, can be explained as users who tend to rate higher or lower than others in a systematic way and thus deviate from the mean rating. From the item perspective, many items tend to go in and out of trend in time and thus receiving higher or lower ratings accordingly, whereas others may be considered "classics" and get substantial higher ratings all the time compared to the mean rating of the dataset. For example, suppose that the overall average ratings in a movies dataset is 3.3 and the average rating of a certain movie is 4.0. Then, there is a tendency for this movie to be rated 0.7 above the average rating and so we can say that the bias of this specific movie is $+0.7$. Doing exactly the same thing for a user can show us that a specific user, for example, tends to rate 0.4 below the average so this user has a bias of -0.4.

Because biases tend to cover a significant percentage of the total information, it is clear that the better they are modeled, the more accurate the predictions will be. Thus, it is better to not let the whole rating value be determined just by the interaction between user and item vectors. As a first and basic approach, we assume that biases are stable in time, so we can have an approximation of the bias term $b_{u,i}$ for the rating $r_{u,i}$ as:

$$b_{u,i} = \mu + b_i + b_u \quad . \tag{3}$$

As μ we denote the overall average rating of the dataset, as b_i the item bias, and as b_u the user bias. The equation presented above is usually called *baseline predictor*; we follow this specific naming throughout this paper. The reason it is called that way has to do with the fact that on its own, Equation 3 as written above, can provide a first-order approximation of the ratings we need to predict, derived purely in a statistical way without the personalization Equation 1 provides.

After defining the bias modeling, we can rewrite the prediction function of our model, including now instead of just the interaction between the user–item vectors, the biases part as well. A nice way to visualize all the above is to imagine Equation 3 as responsible for the DC part of the information signal and Equation 1 as responsible for the AC part:

$$\hat{r}_{u,i} = \mu + b_i + b_u + p_u \cdot q_i^T \quad . \tag{4}$$

Now that we have redefined the prediction function, we can also rewrite the squared error function which we have to minimize as:

$$\min_{p,q} \sum_{u,i \in \kappa} \left[(r_{u,i} - \mu - b_i - b_u - p_u \cdot q_i^T)^2 + \lambda(||p_u||^2 + ||q_i||^2 + b_u^2 + b_i^2) \right] \quad . \tag{5}$$

Previous studies [9, 10] have shown that for the efficient modeling of biases, we insert them into the learning process of the stochastic gradient descent algorithm and thus we discover them simultaneously with the vectors p and q. Similarly to the learning process of the user-latent and item-latent feature vectors, in order to learn the biases the optimization algorithm has to change their values via applying some update rules in each iteration, until it converges to a minimum. The update rules for the biases are:

$$\begin{aligned} b_u &\leftarrow (1 - \lambda_2\gamma_2)b_u - \gamma_2 \mathrm{err}_{u,i} \quad , \\ b_i &\leftarrow (1 - \lambda_2\gamma_2)b_i - \gamma_2 \mathrm{err}_{u,i} \quad , \\ \mu &\leftarrow (1 - \lambda_2\gamma_2)\mu - \gamma_2 \mathrm{err}_{u,i} \quad . \end{aligned}$$

We can use different learning rates and regularization parameters from the ones we have set for the learning of the vectors p and q, but there is no optimal rule for finding the best values of each parameter other than trial-and-error and one could spend much time just optimizing by hand all the parameters to achieve fine-tuning.

4 A Half-Life Decaying Model

It is important for recommender systems to adapt accordingly to the user preferences as they change in time, and there are plenty of different ways to do so [8, 11, 17]. In some cases it is important to always ‘remember’ all past ratings of a user, while sometimes it may be more useful to provide ‘fresher’ recommendations based more on her most recent likings.

In [2], adaptive filtering systems are categorized as locally or asymptotically adaptive. Asymptotically adaptive systems spread the emphasis over the whole time-line, weighing all events of the past equally. Locally-adaptive systems rely more heavily on data collected in the recent past. The assumption behind using asymptotic adaptivity is that user interests or item characteristics are stable over time, while local adaptivity is suitable when there are drifts in either interests or item characteristics. In this study, we explore local adaptivity assuming drifts in user interests.

While locally-adaptive systems typically employ sliding windows over the most recent past, in our model we alter the significance of each rating in time using a half-life decaying function. Based on the temporal information available about each observed rating available in the training dataset (i.e. timestamps), we weight each rating in accordance to its recency. The resulting recommendations are based on all of the user’s past history, as in the ‘classical’ collaborative filtering processes, but are mostly correlated to the more recent likings of the user and less to the older ones.

For example, given a user who has recently enjoyed a certain item, she will continue to receive recommendations based on that specific item for some time until that specific rating becomes old enough and other newer likings of the user have more significant impact on the recommendations that are being produced for that user. Nevertheless, that specific item will continue to have some but lesser impact, depending on the parameter of the model, on the new recommendations being produced.

The half-life decaying function is visualized in Figure 1. This exponential decay function requires the setting of a half-life parameter denoted with the letter h which shows the number of days that need to be elapsed for a specific rating to acquire half of its initial significance in the collaborative filtering process. As t_n we denote the current time of the experiment whereas $t_n - 1$ is the day before the current day and so on until we reach $t_n - h$ which is the day after the half-life time has elapsed. At that time, the rating will only have half of its initial significance to the collaborative filtering process something which is achieved via under-weighting the values. Thus, the weight of a rating will be:

$$w_{u,i} = \exp\left(\frac{\ln 0.5}{h}(t_{\mathrm{n}} - t_{u,i})\right) . \tag{6}$$

It is very important to clarify that we cannot apply the weighting directly to the ratings before the learning process begins. Doing so, we would misrepresent the existing information and thus create false recommendations. For example, before the beginning of the learning process, a 4-star rating should not be downgraded to 2-star just because

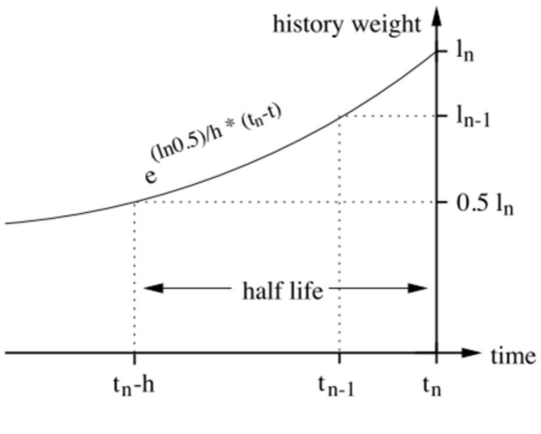

Fig. 1. A half-life decaying function

its timestamp is half-life days old, because it has a totally different meaning. A rating of 4 stars may mean that the user enjoyed enough the specific movie whereas a rating of 2 stars may mean that the user was quite disappointed by the movie. So it is clear that if we apply the time-decaying model, in a static way, as a pre-processing step would alter the initial information significantly and thus produce wrong recommendations.

For the reason above, we follow a different route for integrating half-life decay in our modeling. Instead of direct weighting in a pre-processing step as described above, we insert the decay in the learning process inside the stochastic gradient descent algorithm. More thoroughly, we now consider the user bias term as well as the users-latent features matrix as a temporal process which follows the half-life decaying function as described above. The reason why we pick to insert the decaying characteristics only into the user parts of the total rating value is based on the assumption that only users' interests change over time whereas the items' characteristics stay the same, thus items can be described in an almost constant way in time with respect to the latent features. Consequently, Equation 4 is modified as:

$$\hat{r}_{u,i} = \mu + b_i + w_{u,i} b_u + w_{u,i} p_u \cdot q_i^T \quad . \tag{7}$$

Previous works have shown that models which use temporal information into their modeling, can achieve an even better accuracy when slightly decreasing the total influence of temporal effects in the prediction process. In line with this, we also consider a second technique for the half-life decaying modeling which uses a dampening factor for this specific effect. The technique is based on the assumption that users' preferences do not change entirely over time while not staying stable either. Thus, an approximation of such situation is to consider both a stable DC part of the user bias modeling, and an non-stable AC part which is half-life modeled as described previously. Therefore, the new prediction equation has a stable user bias part and one that is being weighted:

$$\hat{r}_{u,i} = \mu + b_i + b_u + w_{u,i} b_u + w_{u,i} p_u \cdot q_i^T \quad . \tag{8}$$

The idea behind this damping factor is not novel as it comes from the previous work of Koren [9] who used this methodology in a different way in order to achieve a more accurate modeling of the drift of users' preferences.

5 Experimental Evaluation

In this section, we provide a thorough description of the different models we implemented as part of our experiments while highlighting the gradual increase in recommendation accuracy as we dive deeper into capturing different effects in the data. First, we provide some characteristics of the benchmark dataset we used in order to test our hypotheses and describe the evaluation measure.

5.1 Dataset & Evaluation Measure

The dataset we used in our experiments, Movielens 100k, is standard in the field of collaborative filtering recommender systems [6, 7]. It has been used widely in many experiments in academia, thus it is a suitable dataset conducting experiments on and testing new algorithms because new results can easily be compared to other baselines and state-of-the-art methods.

Movielens 100k consists of 100,000 unique ratings, from 943 users for 1,682 movies, spanning approximately 7 months. In addition, temporal information for each rating is available, i.e. the exact date and time the rating was given. In short, each row of the dataset, consists of a unique user-id, a unique movie-id, the given rating in a scale from 1 to 5 stars, and finally the timestamp. The sparsity of the dataset is 6.3%, meaning that only that amount of the total possible ratings is observed. Therefore, we have to deal with a sparse utility matrix, as it is usually the case in such tasks.

To perform our experiments, we use a 80%–20% split, performing the learning on the 80% of the earliest ratings and then test the model to the rest 20% latest ratings. In order to compute the prediction accuracy of our model, we use the most common evaluation measure in the field of recommender systems, namely, the Root Mean Squared Error (RMSE), which is computed with the following formula:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{u,i}(\hat{r}_{u,i} - r_{u,i})^2} \ .$$

The set (u, i) denotes all the ratings from a user u to an item i which belong to the test set. Also, n is equal to the amount of entries in the test set, $\hat{r}_{u,i}$ is the predicted rating for the rating of user u to item i, and finally, $r_{u,i}$ is the actual rating which our prediction is compared to.

5.2 Experiments & Results

As part of our experiments, we implement several different models, ranging from the very basic baseline predictor to the more accurate matrix factorization model with half-life decay.

Specifically, the first model we implement is a first rank approximation for the ratings as we described previously given just by the baseline predictor, generating predictions purely in a statistical way using Equation 3 (baseline model). Next, we create a model with only the collaborative filtering part, described by Equation 1. This specific model captures only the interaction between the user and the items vector while performing the learning of the values via the stochastic gradient descent algorithm (SGD model). Our third model is derived from the combination of the two previous equations. In this way, the model becomes immune to bias effects that exist in the data as described in a previous section (Biased SGD model). The last two models are our proposed approaches with the half-life modeling. The first one uses the Equation 7 to compute the appropriate weights for the half-life decaying and then inserts them in the prediction equation as shown in Equation 5 (halflife model 1). The second half-life model we investigate, uses a damping factor to reduce the effect of the half-life decaying taken into consideration in the prediction process and achieve a smoother usage of the specific effect, as described in a previous section, using Equation 8 (halflife model 2).

The iterative process of minimizing the squared error function through the Stochastic Gradient Descent algorithm is performed while revisiting the elements of the utility matrix which belong in the training set in Round–Robin fashion and applying on them the update rules of each model as described.

One of the most important variables of our model is the learning rate γ which can possibly have different values throughout the learning process of the biases and the user/item vectors. Its physical meaning is the magnitude of the step is being taken each time in order to reach a minimum. Although different values of learning rates are possible, in preliminary experiments (not shown in this paper) the best overall performance was achieved by a learning rate equal to 0.01 for both the biases and the user/item vectors. This value of learning rate was set through via a grid search for various values of γ between 0.001 and 0.02 with the best results coming from the $\gamma_1 = \gamma_2 = 0.01$ value.

Another important variable for our modeling is the number k of latent features. The value of k has been set through cross-validation to 20, when using the Biased SGD model for testing several values, as shown in Table 1. As we can see from the table, this model achieves better prediction accuracy for 20 latent features. In preliminary experiments, we found that $k = 20$ is also a good value for all other implemented models, thus we keep this variable stable throughout our experiments.

Table 1. RMSE as a function of k latent features in Biased SGD model

k	10	20	30	40	50
RMSE	0.9401	**0.9344**	0.9426	0.9429	0.9460

Last, we set the regularization parameter $\lambda_1 = \lambda_2 = 0.1$ after testing several different values between 0.001 and 0.2. One can possibly use different regularization values for each update rule but all our models seem to perform better when these regularization values are equal.

As far as the temporal models are concerned, we use a history function in order to model the time-decay of the ratings. The basic idea is that the older a rating becomes, the less impact should have to the future likings of a user and thus should be down-weighted throughout the collaborative filtering process. Due to the fact that there is always some drift in user interests, we can assume that the older a rating is, the less correlation it has with the current preferences of a user. On the contrary, the newer the rating is, the more has to do with the current tastes of the user. As we have said before, we use Equation 6 to model this decay in the significance of ratings.

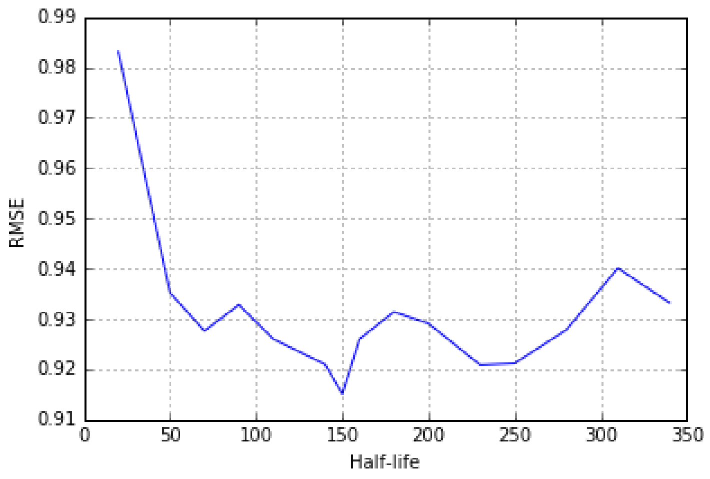

Fig. 2. Model performance for various values of h

After testing several half-life values h, we found out that the value of h which yields the best predictions is about 5 months (or 150 days). This means that the significance of a rating starts dropping as the rating becomes older and reaches 50% of its initial significance after 150 days approximately, in the specific dataset we use for our experiments. The best value for the half-life parameter h was set via a grid search in our first half-life model while the results seem to confirm on the second model, with the damping factor, too. The results of the grid search are visualized in Figure 2.

Table 2. Effectiveness and approximate running time per model

	Baseline	SGD	Biased SGD	Halflife 1	Halflife 2
RMSE	0.9858	0.9428	0.9344	0.9151	**0.9122**
Exec. time	1.2 min	2 min	2.4 min	3 min	3.3 min

As we can see in the figure, small values of the half-life decaying parameter tend to minimize the impact of the collaborative filtering process resulting to accuracy similar to our baseline model, which has no collaborative filtering involved in it. While increasing the half-life parameter, we observe some fluctuations in the overall performance of our model but we can clearly identify that the accuracy maximization is achieved by $h = 150$ days. As we can see from Table 2, the incorporation of half-life decay methodology works in favor of the recommendations accuracy. Furthermore, the two half-life models give similar results with the second model with the damping factor having a slightly better accuracy.

Comparing our results to the closely related, state-of-the-art, svd++ algorithm [9] for the Movielens 100k dataset, we see that they are very close to those of svd++ which scores between 0.91096 and 0.9076, depending on the tuning of model's parameters, as reported in the MyMediaLite library's benchmark results.

In general, as it can be concluded by the results above, each time we add to our prediction methodology a specific effect which lets us capture user–item interactions in greater depth, the model achieves a better accuracy. The first significant increase in the prediction accuracy happens when we embed the modeling of biases, lowering the RMSE from the simple collaborative filtering approach from 0.9428 to 0.9344. Bias modeling has been a well-known technique for improving recommendations accuracy and many have used it before. When we insert the half-life decaying methodology to our prediction process, recommendation accuracy increases significantly for both our models against the simpler models with no temporal information. The improvements in prediction accuracy indicate that there is indeed vital information in temporally-dependent effects in the data which we can use in order to generate better recommendations.

6 Conclusions and Future Work

We have introduced a method for dealing with drifts in user interests in collaborative filtering by exploring the notion of half-life of ratings. The method incorporates an exponential decay function into a matrix factorization model, and comes in two 'flavors': pure decay, or decay with a dampening factor. Experimental results have shown that both variations yield large improvements in prediction accuracy, confirming that significant user interest drifts exist. On the Movielens 100k dataset, spanning approximately 7 months of ratings, accuracy was maximized when using a half-life value of approximately 150 days.

Continuing this work, we plan to examine temporal effects in data with much longer timespans. Furthermore, we intend to experiment with similar methods which will learn via a learning algorithm, such as stochastic gradient descent, the half-life parameter for each user individually. What motivates us to do so is the fact that not all users are characterized by the same half-life parameter, as some of them tend to change preferences with a difference pace than others.

References

1. Arampatzis, A., Beney, J., Koster, C.H.A., van der Weide, T.P.: Incrementality, half-life, and threshold optimization for adaptive document filtering. In: Proceedings of The Ninth Text REtrieval Conference, TREC 2000, Gaithersburg, Maryland, USA, November 13-16, 2000. vol. Special Publication 500-249. National Institute of Standards and Technology (NIST) (2000)

2. Arampatzis, A., van der Weide, T.P.: Document Filtering as an Adaptive and Temporally–dependent Process. In: Proceedings of BCS–IRSG European Colloquium on IR Research (ECIR01) (April 2001)
3. Bell, R.M., Koren, Y.: Lessons from the netflix prize challenge. SIGKDD Explorations 9(2), 75–79 (2007)
4. Gomez-Uribe, C.A., Hunt, N.: The netflix recommender system: Algorithms, business value, and innovation. ACM Trans. Management Inf. Syst. 6(4), 13:1–13:19 (2016)
5. Hallinan, B., Striphas, T.: Recommended for you: The netflix prize and the production of algorithmic culture. New Media & Society 18(1), 117–137 (2016)
6. Harper, F.M., Konstan, J.A.: The movielens datasets: History and context. TiiS 5(4), 19:1–19:19 (2016)
7. Herlocker, J.L., Konstan, J.A., Borchers, A., Riedl, J.: An algorithmic framework for performing collaborative filtering. In: SIGIR '99: Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, August 15-19, 1999, Berkeley, CA, USA. pp. 230–237. ACM (1999)
8. Kim, Y., Kim, C., Tantatsanawong, P. (eds.): 2012 International Conference on Information Networking, ICOIN 2012, Bali, Indonesia, February 1-3, 2012. IEEE Computer Society (2012)
9. Koren, Y.: Collaborative filtering with temporal dynamics. In: IV, J.F.E., Fogelman-Soulié, F., Flach, P.A., Zaki, M.J. (eds.) Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Paris, France, June 28 - July 1, 2009. pp. 447–456. ACM (2009)
10. Koren, Y., Bell, R.M., Volinsky, C.: Matrix factorization techniques for recommender systems. IEEE Computer 42(8), 30–37 (2009)
11. Lee, T., Park, Y., Park, Y.: A time-based approach to effective recommender systems using implicit feedback. Expert Syst. Appl. 34(4), 3055–3062 (2008)
12. Lee, T., Park, Y., Park, Y.: An empirical study on effectiveness of temporal information as implicit ratings. Expert Syst. Appl. 36(2), 1315–1321 (2009)
13. Ma, H., Zhou, D., Liu, C., Lyu, M.R., King, I.: Recommender systems with social regularization. In: King, I., Nejdl, W., Li, H. (eds.) Proceedings of the Forth International Conference on Web Search and Web Data Mining, WSDM 2011, Hong Kong, China, February 9-12, 2011. pp. 287–296. ACM (2011)
14. Symeonidis, P., Zioupos, A.: Matrix and Tensor Factorization Techniques for Recommender Systems. Springer Briefs in Computer Science, Springer (2016)
15. Takács, G., Pilászy, I., Németh, B., Tikk, D.: Investigation of various matrix factorization methods for large recommender systems. In: Workshops Proceedings of the 8th IEEE International Conference on Data Mining (ICDM 2008), December 15-19, 2008, Pisa, Italy. pp. 553–562. IEEE Computer Society (2008)
16. Tian, G., Wang, J., He, K., Sun, C., Tian, Y.: Integrating implicit feedbacks for time-aware web service recommendations. Information Systems Frontiers 19(1), 75–89 (2017)
17. Ullah, F., Sarwar, G., Lee, S.C., Park, Y.K., Moon, K., Kim, J.T.: Hybrid recommender system with temporal information. In: Kim et al. [8], pp. 421–425
18. Wang, Z., Lu, H.: Online recommender system based on social network regularization. In: Loo, C.K., Yap, K.S., Wong, K.W., Jin, A.T.B., Huang, K. (eds.) Neural Information Processing - 21st International Conference, ICONIP 2014, Kuching, Malaysia, November 3-6, 2014. Proceedings, Part I. Lecture Notes in Computer Science, vol. 8834, pp. 487–494. Springer (2014)
19. Zimdars, A., Chickering, D., Meek, C.: Using temporal data for making recommendations. In: Proceedings of the Seventeenth Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-01). pp. 580–588. Morgan Kaufmann, San Francisco, CA (2001)