DeMF: an online recommender system based on matrix factorisation

Mengjie Zhao
EPFL
mengjie.zhao@epfl.ch

Rhicheek Patra
EPFL
rhicheek.patra@epfl.ch

ABSTRACT

With the booming of e-commerce, customers are inundated with various choices during online shopping. Recommender systems are developed to help customers in filtering the huge amount of information and suggesting the most appropriate products. Recommender systems are normally built by using explicit feedbacks collected from customers such as ratings. Implicit feedbacks such as webpage visits and searching history are also available to improve the accuracy of recommendation. Recommender systems should also be capable of capturing customers' preferences, which may drift significantly along with time. To achieve this, recommender systems have to update recommendations in real time, however, the complexity of simply updating the whole model is expensive due to the huge amount of users and items. Within this project, we propose a recommender system which employs matrix factorisation method and can update recommendations in real time, meaning that preference drift of users can be captured. The experimental results show that our model introduces lower mean square error (MAE) while the trade-offs are extra small amount of time and memory consumptions.

1. INTRODUCTION

In recent years, researchers build recommender systems based on two main strategies, namely the content filtering approach and the collaborative filtering approach. In the content filtering approach, a recommender system firstly collects information from users or items then corresponding profiles are created. After that, the recommender system will match items based on obtained user and item profiles then makes recommendations. The drawback of content filtering approach is apparent: users may not happy to provide their personal information or simply provide inaccurate information due to privacy considerations resulting in reduction to recommendation accuracy.

Compared with content filtering, collaborative filtering approach only focuses on utilising users' past purchasing behaviours. By analysing relationships between users and items, the collaborative filtering approach generally provides better recommendation accuracy than content filtering approach[1].

Collaborative filtering approach mainly includes two methods, namely the neighbourhood method and latent factor model. Matrix factorisation method is a widely used latent factor model, within which users and items are characterised by vectors containing predefined number of latent factors. And the inner product of the two vectors represents the level of preference from this user to the item. However, one main drawback of matrix factorisation is the *cold start* problem – users who never rated some items, or items that never be rated by users, will not be well recommended.

Recommender systems can utilise both explicit feedbacks such as ratings, and implicit feedbacks like repeating times of a track to generate recommendations. Recommender systems using explicit feedbacks can provide confident and accurate recommendations when the collected explicit feedbacks is accurate and reliable. However, it is common that customers may give random feedbacks, as a result, the recommendation accuracy could be reduced. The idea of including implicit feedbacks in recommender systems is originally proposed in [2], indicating that using implicit feedback to build recommender systems is feasible.

Capturing preference drift of users is an important property of a successful recommender system, especially for applications where users' focusing topics switch quickly like Tweeter. There exist plenty of researches focusing on modelling users' preferences, aiming to improve performance of recommender systems and other data mining aspects [5]. To capture preferences drift of users, one recommender system can choose to update its entire model every time when new feedbacks from user arrive. However, considering the complexity of the model, time consumption of updating the entire model is significantly large and users' preference may already changed again.

TeRec, a recommender system that can be updated in real time by utilising temporal information was proposed to improve recommendation accuracy over streaming data like Tweeter or Weibo [6]. Based on timestamps of streaming data, TeRec determines whether it is necessary to update the recommender system or not. Further, if TeRec determines to accept the newly streamed data, only small parts of the system will be updated. As a result, the updating complexity is relatively low, meaning that TeRec can react to preference changes of users in real time. More importantly, mobile clients now are able to participate in the recommending process due to the relatively small computation of TeRec.

In this work, we present DeMF, a recommender system that achieves real time recommendation over streaming data. The experimental results show that DeMF further improves the MAE performances, compared with pure matrix factorisation method and TeRec. Our contributions are listed as follows:

1. We validate using the streaming data to capture preferences drifts of users is feasible.
2. We democratise mobile clients and they now are able to join the recommendation-updating process, instead of simply waiting for recommendation results from the central server.
3. We propose a new algorithm that further improve recommendation accuracy compared with exiting models.
4. We explore how parameters such as learning rate, regulation, number of latent factors can affect the recommendation accuracy.

2. BACKGROUND

Within this section, we provide information and concepts needed for understanding the problem and our model. We also give statistical properties of datasets on which we run our model.

2.1 Matrix factorisation

Matrix factorisation method becomes increasingly popular in developing recommender systems after the Netflix Prize, where the winners' method widely used this technique [7].

Matrix factorisation technique in recommender system context is close to the singular value decomposition (SVD), which is applied to decompose a $m \times n$ real or complex matrix M into production of:

$$M = U \times \Sigma \times V^*$$

Where U is a $m \times m$ unitary matrix, Σ is a $m \times n$ diagonal matrix and V^* is a $n \times n$ unitary matrix. We can merge Σ and V^* and denote the production as Q^T, then we get:

$$M = U \times Q^T$$

which means we can decompose the original matrix to production of two matrices. However, in developing recommender systems, we cannot use SVD directly since the matrix is usually sparse, for example, the data density in Netflix Prize is less than 2% and SVD is not defined on sparse matrix. However, we can approximate the original matrix by finding two matrices whose production is close to the original one. And we can choose different number of factors (n in SVD) as long as the approximation is accurate.

Considering a recommender system using explicit feedbacks and let the $m \times n$ matrix M as a rating matrix accommodating ratings of m users to n items. And let matrix Q be a $n \times f$ item matrix where numbers in each row q_j are latent factors that charactering this item. For example, in movie recommendation scenario, the latent factors can represent the characteristics of a movie such as style and director. Then let matrix U be a $m \times f$ user matrix, where numbers in each row u_i are latent factors, representing the level of preference of this user to the items' characteristics. Then our approximation to matrix M is:

$$\hat{M} = U \times Q^T$$

and our predictive ratings from user i to item j can be calculated as the inner product:

$$\hat{r}_{ij} = u_i \cdot q_j^T$$

Let r_{ij} be the actual rating of user i to item j, and $\hat{r}_{ij}$ as our predictive ratings of user i to item j, then the approximation problem can be solved by reducing the following cost function:

$$\min_{u,q} \sum_{r_{ij} \in \kappa} (r_{ij} - u_i \cdot q_j^T)^2 + \lambda(\|u_i\|^2 + \|q_j\|^2)$$

where κ is the set of known rating events from the dataset. The second term is added to prevent overfitting and λ is the regulariser parameter. We define our predictive error:

$$e_{ij} := r_{ij} - \hat{r}_{ij}$$

Then this optimisation problem can be solved by using stochastic gradient descent (SGD) algorithm:

$$q_j \leftarrow q_j + \gamma \cdot (e_{ij} \cdot u_i - \lambda \cdot q_j)$$

$$u_i \leftarrow u_i + \gamma \cdot (e_{ij} \cdot q_j - \lambda \cdot u_i)$$

Where λ is the regulariser parameter and γ is a constant.

2.2 Datasets description

We evaluate our model performance mainly on two datasets, namely the MovieLens-100k [1] dataset and the Ciao [2] dataset.

The MovieLens-100k dataset is a widely used explicit feedback dataset containing $100,000$ rating events from 943 users to 1682 movies. The ratings range from 1 to 5 and each user has rated at least 20 movies. Besides, each rating event also includes a timestamp indicating when the user rated the movie. Based on the dataset, we construct a $1000 - by - 1682$ rating matrix, accommodating ratings of each user to watched movies. Because each user may only watched a few movies, the rating matrix is considerably sparse.

The Ciao dataset contains 36065 rating events from 2378 users to 16861 items. The ratings also range from 1 to 5 and timestamp for each rating event is included as well. We construct a $2378 - by - 16861$ rating matrix to accommodate all ratings and this matrix is sparse too, similar to that in the MovieLens-100k dataset.

As mentioned in previous section, matrix factorisation method suffers severe *cold start* problem, as a result, we use 98% of dataset as training data while the remaining 2% as testing dataset among all datasets we used in all performance evaluation experiments.

2.3 Statistics of datasets

Analysing datasets to obtain informative characteristics is a crucial step before carrying out further evaluating experiments. Within this section, we give some statistical properties of datasets we used. In all experiments, it is necessary to sort the given datasets since we need to simulate

[1]*www.grouplens.org/datasets*
[2]*www.jiliang.xyz/trust.html*

the real world, where new rating events are streamed to the recommender system step by step from each user. After processing the dataset, we realised that dataset characteristics can change significantly along with the sorting process, especially for rating events in the testing dataset.

2.3.1 *MovieLens-100k*

Following figures illustrate the distribution of rating events from users in the training dataset before and after the sorting process. The x-axis represents different rating event frequencies from each user of MovieLens-100k. It can be observed that most of the users have less than 100 rated movies while there exist users that rated more than 700 movies. It can be concluded that regarding the training dataset, the sorting process does not introduce much differences.

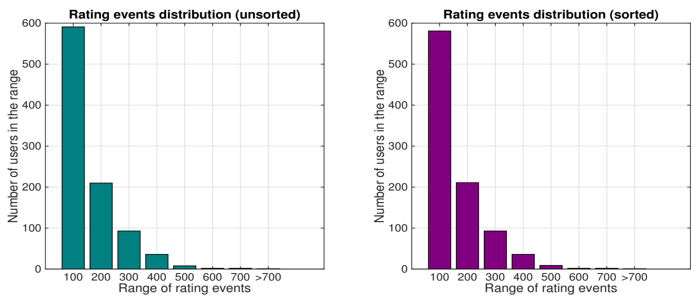

Figure 1: Rating event distribution in the training dataset from MovieLens-100k.

On the contrary, the sorting process introduces significant differences to the testing datasets. There are 629 distinct users show up in the unsorted testing dataset, and most of them present many times in training dataset. However, in the sorted testing dataset, there are only 49 distinct users show up, among which 8 users never show up in the training dataset. In addition, the consecutively appeared rating events from the same users in the sorted testing dataset introduces higher probability of overfitting in our model.

2.3.2 *Ciao dataset*

Following figures illustrate the distribution of rating event from users before and after the sorting process within the training dataset. Within the unsorted training data, a large

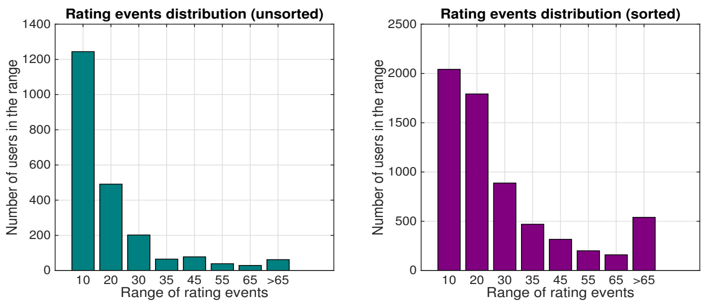

Figure 2: Rating event distribution in the training dataset from Ciao.

proportion of users rated less than 13 items and most users rated fewer items in the sorted training dataset too but we saw an big increase in the number of users who rated more than 65 items. However, the differences between unsorted and sorted testing datasets is not significant compared with that in MovieLens-100k dataset.

3. STRUCTURE AND ALGORITHMS

We provide detailed information about the algorithm we used in generating recommendation and the architecture of our proposed recommender system.

3.1 Core steps and algorithm

We make modifications to the structure of TeRec proposed by *Chen et al.* in [6]. The core steps are listed as follows:

1. Performing a pure matrix factorisation model on the training dataset and obtain an initial rating matrix $\hat{M}$ as current prediction from users to items.

2. We maintain a reservoir, where fresh rating events (rating events that have large timestamps) will be added to the reservoir according to:

$$P(add) \propto 1 - \frac{size\ of\ reservoir}{timestep}$$

   Once a new rating event is added to the reservoir, one relatively old rating event within the reservoir will be removed. As a result, the rating events within the reservoir are always new, reflecting users' real time preferences.

3. According to size of the reservoir and timestamps of rating events, we firstly stream the testing dataset to fill the reservoir. After that, when a new testing data point arrives and accepted based on probability, we put it into the reservoir and perform our proposed algorithm to update the recommender system and make corresponding recommendations to the user of the rating event.

*Algorithm*1 in next page is applied to update the recommender system to capture real time preference of users. Compared with the original algorithm proposed in [6], we proposed several improvements and list them as follows:

1. Within the testing datasets, we notice that rating events from the same users appear consecutively. As a result, we add a decay parameter to learning rate α to prevent divergence in the learning phase. And the decay is a function of round T.

2. The function ***SamplePositiveInput*** samples informative rated items of the specific user from the reservoir. We define the positive items as: items that are rated with high scores from both users and our recommender system, or items that are rated with low scores from both users and our recommender system. This choice is intuitive since it means our model is on the correct direction of recommendation.

3. The function ***SampleNegativeInput*** samples informative rated items of the specific user from the reservoir too. We define the negative items as: items that receive higher predictive score than the actual rating from corresponding user. By using this function, we suppress high scores predicted by our model to prevent divergence.

4. For each user, we allocate different value of learning rate α since the number of rating events happened in testing dataset of different users varies significantly.

5. In the original algorithm, the authors did not indicate what η should be if there is no negative inputs. We set it to zero since we restrict our model to learn from informative rating events in the reservoir.

6. The original algorithm does not take actual ratings from user into consideration. It can be observed that the η in the original algorithm is a function of predictive ratings of positive items and negative items. Hence, the recommender system can generate very high scores since it keeps increasing ratings for sampled positive items. To prevent this effect, we define η as the distance between actual negative ratings and predictive negative ratings. We also introduce the parameter e to take actual ratings into consideration. And this parameter reduce the trend of predicting very high scores to positive items from our model.

Algorithm 1 Online algorithm for updating

```
 1: procedure
      INPUT: NEW RATING EVENT $l$ IN TESTING DATASET, LATENT FACTOR MATRICES $U$, $Q$
      OUTPUT: UPDATED LATENT FACTOR MATRICES $U$, $Q$
 2:     if Accept incoming data point then
 3:         Update the reservoir
 4:         SNuIdx = SamplePositiveInput(reservoir $\cup$ $u$)
 5:         SPuIdx = SampleNegativeInput(reservoir)
 6:         for each $p$ in SPuIdx do
 7:             for round in 1 to T do
 8:                 $\alpha = \frac{\alpha}{(1+T)^{0.01}}$
 9:                 $\hat{r} = R(u,p)$
10:                 if length of SNuIdx is 0 then
11:                     $\eta = 0$
12:                 else
13:                     $\psi = \textbf{hinge}(R(SNuIdx) - \hat{R}(SNuIdx))$
14:                     $\eta = mean(\psi)$
15:                 end if
16:                 $e = R(u,p) - \hat{r}$
17:                 if length of SNuIdx is 0 then
18:                     $Neg = 0$
19:                 else
20:                     $Neg = mean(Q_{SNuIdx})$
21:                 end if
22:                 $U_u = U_u + \alpha e Q_p - \alpha\eta Neg - \alpha\beta U_u$
23:                 $Q_p = Q_p + \alpha e U_u - \alpha\beta Q_p$
24:                 for each j in SNuIdx do
25:                     $Q_j = Q_j - \alpha\eta U_u - \alpha\beta Q_j$
26:                 end for
27:                 Update current prediction
28:             end for
29:         end for
30:     end if
31: end procedure
```

3.2 System architecture

Within this section, we present the overall architecture of our proposed recommender system DeMF. A typical recommender system structure consists of a central server and several client machines. The central server is responsible to carry out the pure matrix factorisation or other offline recommender algorithms to generate a predictive rating matrix $\hat{M}$ then send suitable recommendations to different users.

Normally, users simply wait for recommendations from the central server since the algorithms used in the server consumes a large amount of memory and time. It is not feasible to update recommendation on users' machines due to their limited computational capability.

It can be observed that the computational complexity of our proposed algorithm is much smaller compared with the pure matrix factorisation algorithm using SGD as learning algorithm running on the server. As a result, the proposed algorithm can be run on client machines and make recommendations in real time. In this case, services are democratised and client machines can participate in the recommending process. Following **Figure 3** illustrates our proposed system architecture:

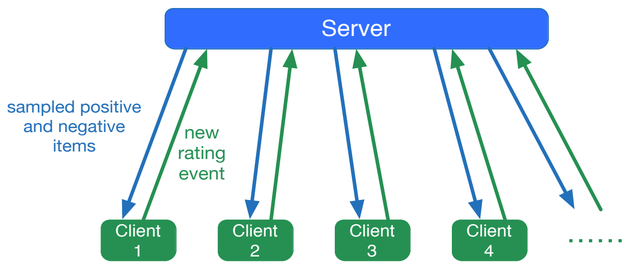

Figure 3: Proposed system architecture

After finishing the pure matrix matrix factorisation process, the server deliver latent factors of positive and negative items for each user. After the users rated some items, the generated new rating event will be sent to the central server to update the reservoir. In addition, the server will send latent factors of updated positive and negative items. Upon receiving latent factors, the client machine applies computation of the proposed algorithm and make corresponding recommendations.

Now the client machines do not simply wait for recommendations from the central server, but participate in the recommendation process. The trade-off is the server needs to store more informations for different users and client machines need to store related latent factors. However, in following experiments we show that the overhead of additional information is very small and the trade-offs are negligible. We demonstrate detailed memory and time consumption trade-offs in next section.

4. PERFORMANCE EVALUATION

Within this section, we evaluate performance of our model DeMF on the introduced two datasets and compare performance with TeRec. The mean square error (MAE) is the metric we mainly used, which is defined as follows:

$$MAE = \frac{1}{n}\sum_{t=1}^{n} |r_t - \hat{r}_t|$$

where n is total number of testing rating event. r_t is the actual rating from users while $\hat{r}_t$ is the predictive rating generated from different models.

4.1 Iteration T

In the applied algorithm in DeMF, parameter T represents how many times we run the algorithm for making each online recommendation. Evaluating the impacts of this parameter is important because the time consumption of DeMF mainly depends on T. We prefer a smaller T that can still achieve good MAE performance since recommendations that consume too much time result in bad user experience and the accuracy should be guaranteed too.

We evaluate how varying T can change the MAE and temporal consumption of TeRec and DeMF. Following **Figure 4** and **Figure 5** illustrate the experimental results of TeRec and DeMF on MovieLens-100k and Ciao datasets respectively. Because recommendations are generated for every user enjoying the service, as a result, time consumptions for different users are different. We use error bar to indicate standard deviation of time consumption among all users.

It can be observed that when T is relatively small, the MAE improvement is not a good choice but the time consumption is really small. On the contrary, large T introduces significant MAE performance reduction due to overfitting and the time consumption is significantly large. Hence, we prefer a smaller T that can still achieve good MAE performances.

It can be concluded that from following two figures, DeMF gives better MAE performance than TeRec on both MovieLens and Ciao datasets. However, the time consumptions between the two models are roughly the same.

We choose $T = 5$ in following experiments that run on MovieLens-100k dataset while $T = 11$ for Ciao dataset, since these choices give good MAE performance while the consumed time for making each recommendation is acceptable to users.

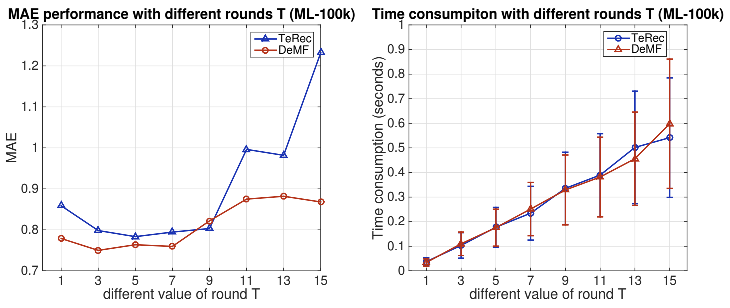

Figure 4: Effects of iteration T (MovieLens-100k)

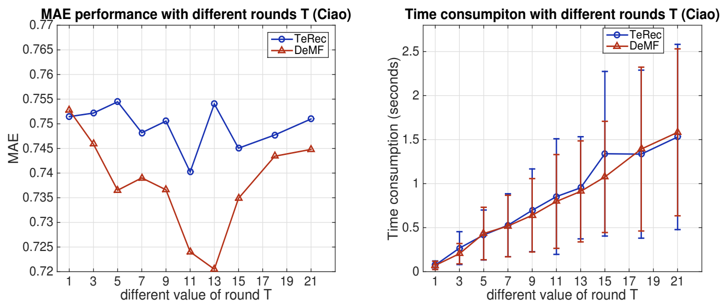

Figure 5: Effects of iteration T (Ciao)

4.2 Size of the reservoir

The reservoir plays an crucial role in our proposed recommendation architecture. Items in the reservoir are more informative and representing users' current preferences. Typically, smaller reservoir better reflects users' current preference since the items in it is even fresher than that in a relatively large reservoir. However, we need to indicate that for MovieLens-100k dataset, smaller reservoir increases the trend of overfitting, since rating events from the same user appears consecutively, result in repetitively training for the same user.

Following **Figure 6** illustrates the MAE performance of our model and TeRec using different size of reservoir, running on both MovieLens-100k and Ciao datasets. It can be observed that when using the MovieLens-100k dataset, for TeRec, reservoir having size of 40 gives best performance while 50 for our model. When using the Ciao dataset, for TeRec, reservoir having size of 60 gives best performance while 50 for our model. Hence we fix the size of reservoir for future evaluations.

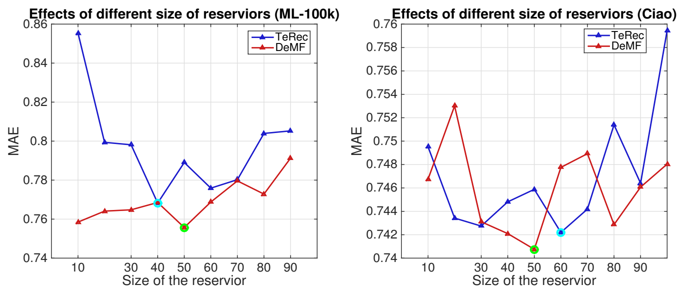

Figure 6: Impacts on MAE from different size of reservoir

4.3 Memory and time consumption

Within this section, we compare the memory and time consumption of DeMF with pure matrix factorisation model to motivate the superiority of proposed architecture. It can be easily found that the online updating algorithm is much more suitable for capturing users' real time preferences since only small part of parameters are required to updated, compared with the pure matrix factorisation.

Figure 7 in following page displays the consumed time and memory in log-scale for making one recommendation from our proposed model and pure matrix factorisation over MovieLens-100k dataset (50 epochs, 20 latent factors for each user and item). Again, since for DeMF, consumptions are measured for each user, thus we use the vertical red lines on the bars to represent the standard deviation.

It can be observed that for making one single recommendation, DeMF requires much fewer time and meomry compared with the pure matrix factorisation. We also demonstrate the comparison results of using Ciao dataset in **Figure 8**.

4.4 Learning rate and regulariser

Learning rate and regulariser are two crucial parameters in any machine learning or data mining models. Learning rate determines the speed of reaching convergence while regulariser determines how flexible the model is. Within this section, we demonstrate how MAE performance varies along with different value of learning rate α and regulariser β in

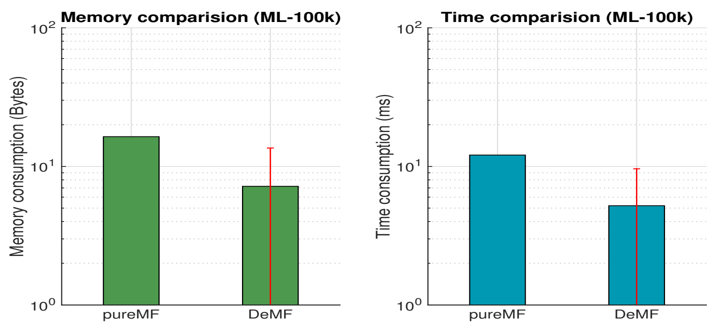

Figure 7: Comparison of memory and time consumption (MovieLens-100k)

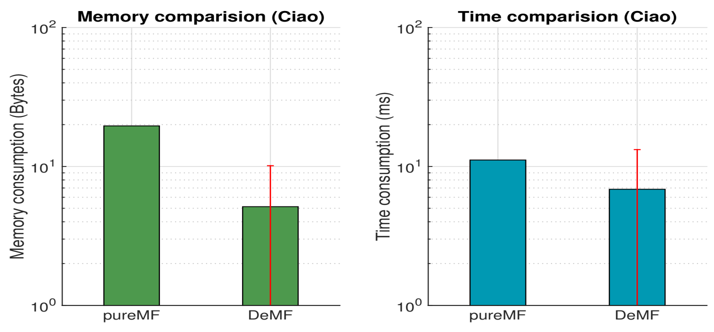

Figure 8: Comparison of memory and time consumption (Ciao)

DeMF and TeRec models. It should be noticed that the discussed learning rate α and regulariser β in this section are the parameters used in the online algorithm, instead of the similar parameters used in the SGD algorithm within the pure matrix factorisation model. We demonstrate our experimental results of the two models over the MovieLens-100k dataset.

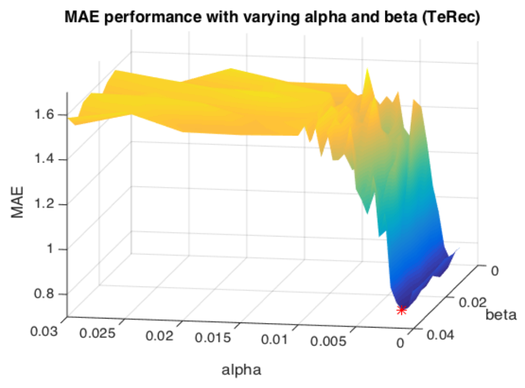

Figure 9: MAE performance of TeRec

Above **Figure 9** shows how MAE varies along with different α and β generated by the TeRec model. The red star indicates the lowest MAE achieved on the MovieLens-100k dataset which is 0.7799 when $\alpha = 0.001$ and $\beta = 0.04$. When α is relatively large (higher than 0.005), the model starts to diverge, resulting in very high MAE. Compared with α, the effects of regulariser β is less significant, hence, the figure focuses showing the effects brought by α.

It can be noticed that TeRec model diverges easily, the reason is that only predictive ratings are considered as suggested in the previous section. The TeRec algorithm focuses on increasing ratings for sampled positive items but the effects of actual rating is ignored. Hence, the predictive values can be large, resulting in large MAE.

Following **Figure 10** shows MAE performance of DeMF under the same parameter schemes. The red star indicates the lowest MAE achieved on the MovieLens-100k dataset which is 0.7420 when $\alpha = 0.008$ and $\beta = 0.035$. Similar to the result of TeRec, the effects of β is less significant compared with α, thus the figure mainly demonstrates the performance variation with different value of α.

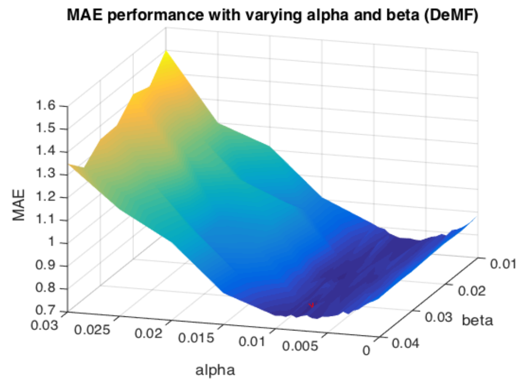

Figure 10: MAE performance of proposed model

DeMF achieves lower MAE compared with the results of TeRec. In addition, it can be easily observed that our model has smaller trend to diverge even when the learning rate α is large (higher than 0.015). The reason is that we also consider the effects of actual rating in addition to the effects of predictive ratings. Overall, we believe that our proposed model is more accurate and reliable than TeRec.

4.5 Usage and performance improvements

Based on discussions and introduction in previous sections, one intuitive idea is that the performance of DeMF is positively related to usage (used frequency from users). In other words, for a specific user, the more usage, the better MAE performance received from DeMF. The reason is straightforward – when a user generates more rating events, the expected number of rating events to be added to the reservoir from this user is increased, as a result, the latent factors of this user get increasingly well trained, bringing larger MAE improvements. Within this section, we explore how the usage changes MAE improvements of DeMF and TeRec.

Following **Figure 11** illustrates the relationship between usage and MAE improvements compared with pure matrix factorisation from DeMF over MovieLens-100k dataset. As suggested in previous section, the appearances of different users in the training dataset vary significantly after the sorting process. As a result, the pure matrix factorisation training phase introduces significant performance discrepancies between different users. Thus in **Figure 11** we only show the MAE improvements of users that generated more than

150 rating events in the training dataset. From **Figure 11**, we can observe a trend that the MAE performance increases along with the increase of usage from different users. However, we can also see glitches, which come from different level of training from pure matrix factorisation phase or mis-predictions from DeMF.

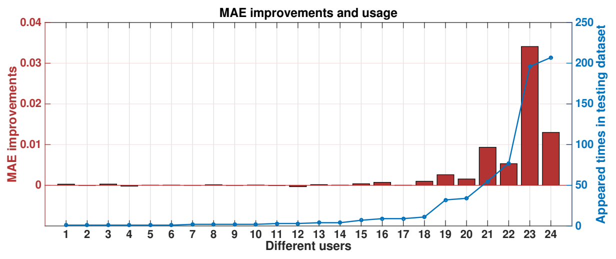

Figure 11: MAE performance and usage (DeMF)

Following **Figure 12** displays the performance of TeRec using the same evaluating framework. It can be observed that TeRec gives worse performance than DeMF. For user $No.$ 23 who generated more than 200 new rating events, the MAE is surprisingly decreased. From these experimental results we can argue that DeMF gives higher and reliable MAE performance.

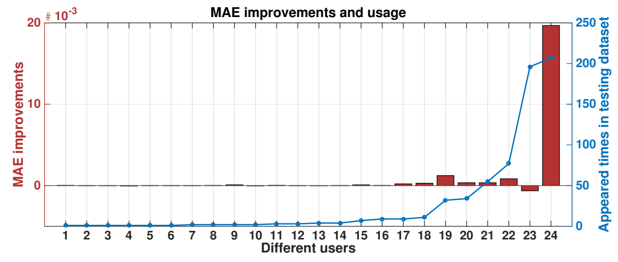

Figure 12: MAE performance and usage (TeRec)

5. CONCLUSIONS

We proposed DeMF, an online recommender system architecture based on matrix factorisation technique achieving real time recommendation by capturing users' real time preferences. Compared with offline recommender systems, our model is fast and the resources consumptions are small. Compared with existing online recommender system such as TeRec, our model is more stable and gives better MAE performance. Experimental results show superiority of DeMF.

6. ACKNOWLEDGMENTS

The author would like to thank Rhicheek Patra, for his suggestions and patient guidances during the whole semester. The author also would like to thank colleague Wei Ma, for our active and inspirational discussions.

References

Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, (8):30–37, 2009.

Douglas W Oard, Jinmook Kim, et al. Implicit feedback for recommender systems. In *Proceedings of the AAAI workshop on recommender systems*, pages 81–83, 1998.

Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, pages 263–272. Ieee, 2008.

Òscar Celma Herrada. Music recommendation and discovery in the long tail. 2009.

Ingo Schwab, Alfred Kobsa, and Ivan Koychev. Learning user interests through positive examples using content analysis and collaborative filtering. *Internal Memo, GMD, St. Augustin, Germany*, 2001.

Chen Chen, Hongzhi Yin, Junjie Yao, and Bin Cui. Terec: A temporal recommender system over tweet stream. *Proceedings of the VLDB Endowment*, 6(12):1254–1257, 2013.

Yehuda Koren. The bellkor solution to the netflix grand prize. *Netflix prize documentation*, 81, 2009.