UC Riverside
UC Riverside Electronic Theses and Dissertations

Title
Exploring Temporal Context for Collaborative Filtering

Permalink
https://escholarship.org/uc/item/3dx874qr

Author
Elsisy, Amr

Publication Date
2017-01-01

Peer reviewed|Thesis/dissertation
Exploring Temporal Context for Collaborative Filtering

A Thesis submitted in partial satisfaction of the requirements for the degree of

Master of Science

in

Computer Science

by

Amr Elsisy

June 2017

Thesis Committee:
Dr. Vagelis Papalexakis, Chairperson
Dr. Vagelis Hristidis
Dr. Eamonn Keogh
The Thesis of Amr Elsisy is approved:

_____________________________________________________________________

_____________________________________________________________________

_____________________________________________________________________

committee Chairperson

University of California, Riverside
Acknowledgments

Over the past year I have received a huge amount of support from Dr. Vagelis Papalexakis. From helping me find a research topic of interest to me, to guiding me through the research itself. Other than research itself, Dr. Papalexakis has always been there for me whenever I needed help, whether it be research related or not.
ABSTRACT OF THE THESIS

Exploring Temporal Context for Collaborative Filtering

by

Amr Elisy

Master of Science, Graduate Program in Computer Science
University of California, Riverside, June 2017
Dr. Vagelis Papalexakis, Chairperson

Thousands of new users join social media website everyday, generating huge amounts of new data. Twitter users for example, generate millions of new posts per day. This can flood our users with huge amounts of information, and thus overload them with information that for the most part they are not interested in. To fix this problem, we need to only show our users information relative to them, such as posts from people they are following. This thesis focuses on how to make accurate recommendations to each user, on which users/pages to follow, thus helping the user view information that is important to them. In particular, we focus on exploring the following research questions: 1) which features yield the best recommendation accuracy, and 2) given those features, what is the best granularity for them, that captures the underlying dynamics, leading to high accuracy.
Table of Contents

List of Figures v

Chapter 1 - INTRODUCTION 1
1.1 Problem Introduction 1
1.2 Literature Review 1
1.3 LibFM briefing 3
1.4 LibFM training methods 3

Chapter 2 - BEST USAGE OF LIBFM 5
2.1 KDD dataset 5
2.2 Modifying KDD default rate 5
2.3 LibFM training method to use 6
2.4 LibFM dimension of pairwise interactions 6

Chapter 3 - RUNNING TESTS 8
3.1 Best subset of features 8
3.2 Different aggregations of time 9
3.3 Other methods and datasets used 10
3.4 Bayesian Probabilistic Matrix Factorization 11
3.5 Bayesian Probabilistic Tensor Factorization 11

Chapter 4 - RESULTS 13
4.1 KDD accuracy using different methods 13
4.2 Movielens RMSE using different methods 14
4.3 Netflix RMSE using different methods 15

Chapter 5 - CONCLUSION 17
5.1 Conclusion 17

Reference page 18
List Of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Accuracy of different training methods for LibFM</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>Dimension of pairwise interaction used by LibFM</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>LibFM accuracy using subsets of features of KDD dataset</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>LibFM accuracy using different time aggregations on KDD dataset</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>Accuracy of LibFM, BPMF, and BPTF on KDD dataset</td>
<td>13</td>
</tr>
<tr>
<td>6</td>
<td>Accuracy of LibFM, BPMF, and BPTF on Movielens dataset</td>
<td>14</td>
</tr>
<tr>
<td>7</td>
<td>Accuracy of LibFM, BPMF, and BPTF on Netflix dataset</td>
<td>16</td>
</tr>
</tbody>
</table>
Chapter 1

1.1 Problem Introduction

Social networking services are growing day by day, and are in huge demand in today’s world. The social networking service we are mostly interested in Tencent Weibo, which is one of the largest blogging websites in China. Tencent Weibo is most similar to Twitter, in the sense that both allow the users to broadcast posts that are up to 140 characters.

Tencent Weibo has around 200 million users, who generate around 40 million messages a day. With the amount of information generated on Tencent Weibo, we run the risk of overwhelming the users with information, and thus we want to limit the amount of information every user sees. We want the users to only see information that is relevant and/or of interest to them. The KDD 2012 competition did exactly that, the participants’ task was to predict whether a user will follow an item that was recommended to them or not. Once we can predict which items a user will follow, we can learn the kind of information that interests the users, and this would ultimately improve the user’s experience, since users will now only see information relevant to them.

1.2 Literature Review

For this Thesis, we studied how temporal context can affect our recommendation accuracy. The recommendation systems we were focused on are social media
recommendations, and movie recommendations. Other people have done similar work, and in this section I’ll mention some of the work that has been done in this area of study.

Regarding temporal collaborative filtering for movie recommendations, Nathan Liu, and Luheng He, from the Hong Kong University of Science and Technology, have published a paper titled “Social Temporal Collaborative Ranking for Context Aware Movie Recommendation”, [1] that shows their research in this field of study. They look at how being time aware can affect the quality of your recommendations. The results they achieved show that with different time aggregations, you can get different accuracy results. They conclude that very small time aggregations will lead to overfitting, which results in poor accuracy. Another very interesting conclusion they draw is that item-oriented model is much more effective than user-oriented model, which implies that the effect of temporal context of the item is much more important than that of the user.

Our second research topic, was how temporal context can affect recommendations on social media. Hongzhi Yin, Bin Cui, Ling Chen, Zhiting Hu, and Zi Huang published a paper titled “A Temporal Context-Aware Model for User Behavior Modeling in Social Media Systems”, [2] which shows how temporal context has an affect on social media recommendations. They concluded that item-oriented model is more effective than user-oriented model. And just like our conclusion, they also concluded that different time aggregations resulted in different accuracy results.
1.3 LibFM briefing

We were most interested in the work done by Steffen Rendle, who achieved second place in KDD 2012 competition. Rendle's approach was based on factorization machines, more specifically he used LibFM [3]. LibFM is a software implementation for factorization machines that features stochastic gradient descent (SGD), Alternating Least Squares (ALS), and Bayesian inference using Markov Chain Monte Carlo (MCMC) as training methods.

Factorization machines can be used as a regression based model or a classification based model, which contains the unary interaction between every variable and the target, on top of that, LibFM also finds all pairwise interactions of all the input variables.

1.4 LibFM training methods

The details of how these training methods works is beyond the scope of this paper, but I will provide the logical intuition behind each one of them. Stochastic Gradient Descents’s (SGD) advantages is that it’s fast, but it’s main disadvantage is that it may not converge. The way SGD works is that we forget about all the variables we are trying to learn, and we just focus on one of the variables. For every iteration of SGD we pick out one user, and we try and learn that variable for this user, and for the next iteration we pick out a different user, and we try and learn the same variable and so on. We keep doing this until we have iterated over all our users. For Alternating Least Squares (ALS), the intuition is that from all the variables we are trying to learn, we fix a
set of variables $U$, and try and learn the remaining unfixed set of variables $V$. For the following iteration we fix $V$, and try and learn $U$. For every iteration we try and learn the unfixed set of variables over all the users, and thus ALS can be slower than SGD. The last training method, Montev Chain Monte Carlo (MCMC), assumes that we have prior knowledge, and will make future prediction based on this prior knowledge.
Chapter 2

2.1 KDD dataset

From all the attributes available in the provided KDD dataset, Rendle decided to use the following (user ID, item ID, user age, user gender, interaction of age and gender, number of tweets, tags, keywords, set of users that the user follows, and the timestamp). All the attributes were used in their original form, except for timestamp. Rendle used the provided timestamp to create user sessions, from which we can better predict whether to recommend an item to a user or not. The intuition behind these sessions is that if we see that a user has short frequent sessions, which means that the user is constantly logging on and off, we know that the user is most likely not in the mood, and will therefore ignore the recommendations you make, even if these recommendations are of interest to the user.

2.2 Modifying KDD default rate

On the original dataset we had a default rate of around 95%, which makes sense since most of the items recommended are usually not followed. Our whole dataset was composed of around 110M instances of which around 5M recommendations were successful. To fix our default rate issue, we first extracted all the successful recommendations to one file, and then shuffled all the remaining data, and picked out another 5M recommendations that were not successful, and thus we had a file of around 10M recommendations, with a default rate of 50%.
2.3 LibFM training method to use

The first question we had to answer was which training method to use. We decided to go with ALS since it had the least number of input parameters, and thus the easiest to use when trying to replicate results. SGD for example has a learn rate parameter that had to be specified, while ALS doesn’t. Other than ALS having the least number of parameters to tune, ALS also gave us the highest accuracy out of all our training methods.

![Training Method vs Accuracy](image)

*Figure 1: Keeping everything constant, except for the training method used. ALS gave us the highest accuracy out of our three training methods.*

2.4 LibFM dimension of pairwise interactions

As mentioned earlier, libFM features pairwise interactions, which leads us to our next question. Should we use pairwise interaction with ALS? And if we do, what should
the dimension of our pairwise interactions be? To answer these questions, just like we answered the previous question on which training method to use, we ran some tests, keeping everything constant, and only modifying the dimension of our pairwise interactions.

The figure above shows that our accuracy drops as our dimension of pairwise interactions increases. With a very high dimension of pairwise interactions, we are forcing LibFM to overfit on the training data, which has a negative effect when predicting the test data.
Chapter 3

3.1 Best subset of features

We tested all the features that Rendle used separately, to see which one would give us the best results. The three variables that were common along all test runs were user Id, Item ID, and the result. As mentioned earlier, we then separately added the remaining features, one by one, used by Rendle and measured our accuracy results. The feature that gave us the most accuracy when used by itself was timestamp. After finding that timestamp is our best feature, we tried combining other features with timestamp to see if our accuracy will be effected. With different combinations of features, our accuracy didn’t increase significantly, and sometimes even decreased.

Figure 3: Different subset of features gave us different accuracy. Timestamp by itself gave us the highest accuracy.
3.2 Different aggregations of time

After finding that timestamp is the single most useful feature, we wanted to see if slightly changing the time aggregation would have any effect on our accuracy. Our timestamp in its original form was in seconds, and so we aggregated it from seconds to hours, days, and weeks, to see if different time aggregations will have any effect on our accuracy.

We then tested our data, with different time aggregations, on LibFM, and we saw that the version of our data with timestamp aggregated to day gave us the highest accuracy. In this dataset, all the events that took place within a 24 hour period were mapped to the same value, instead of each of them having their own unique timestamp, as it was the case on the original data. This makes sense, since KDD data is a form of social media data, and on social media items usually take a couple of hours to a day to start trending, and after a couple of days this trend usually starts fading away. We saw a huge increase in accuracy when the time aggregation was changed from seconds to hours, and then a slight increase from hours to days. When we had our time in seconds, two users that followed the same page a second apart, were considered different, but when the aggregation changed, we start seeing more similarities between users, especially those users that followed the same item within the same time period.
3.3 Other methods and datasets used

Now that we know that timestamp is our most useful feature, and that time aggregating does in fact have a positive impact on our accuracy. We decided to run further tests on new datasets. We downloaded two new datasets, Movielens dataset, and Netflix dataset, and ran the same tests as we did on the KDD dataset. These datasets were of the format UserID, MovieID, Rating, Timestamp. For the Movielens dataset, the provided timestamp was in seconds, while for the Netflix dataset, the provided timestamp was in Days. It’s also important to note that the Movielens dataset was collected over a whole year, while the Netflix dataset was only collected over a single month.
Other than using new datasets, we also decided to use different methods other than just libFM. Our new methods to use were Bayesian Probabilistic Matrix Factorization (BPMF), and Bayesian Probabilistic Tensor Factorization (BPTF). Now we have three datasets, (KDD, Movielens, Netflix), and three methods (LibFM, BPMF, BPTF) to test on these datasets.

3.4 Bayesian Probabilistic Matrix Factorization

The intuition behind BPMF [4] is as follows. BPMF breaks down user preference, and item features into categories. The dot product of these categories is then taken, and a rating is given using these scores.

3.5 Bayesian Probabilistic Tensor Factorization

BPTF [5] is basically the same as BPMF, with the difference that BPTF takes time into account, while BPMF doesn’t. A simple example of how BPTF works is the following. Assume I like green shoes, and some new green shoes just came out, but it’s valentines week. BPTF won’t recommend these shoes for me, since the trend right now is red, not green. But if it was Saint Patrick’s day, BPTF would recommend these shoes for me, because I like green shoes, and the current trend is also green.

Now that we have some intuition behind all of our methods, it’s time to test these methods on our datasets, and see what results each method can achieve. On the Netflix and Movielens datasets we calculated root mean squared error (RMSE) to show how
close our rating was to the actual rating the user gave the movie, rather than calculating the accuracy as we did for the KDD dataset. The lower our RMSE was, the better our accuracy was. For BPMF our RMSE remained constant, since BPMF doesn't take time into consideration when predicting a rating. Meanwhile for BPTF, our RMSE changed when our timestamp aggregation changed.
Chapter 4

4.1 KDD accuracy using different methods

The first dataset to test out methods on, was the KDD dataset. We tried all three methods with varying time aggregations, to see how the accuracy of each method varied with respect to the time aggregation. BPMF performed the worst, and that is because BPMF doesn’t take time into consideration, and the KDD dataset is heavily reliant on time, since it’s a social media dataset. We then see that LibFM, and BPTF achieve their highest accuracies when time is aggregated to days.

Figure 5: KDD. Performance of LibFM, BPMF, and BPTF on KDD dataset, with different time aggregations.
4.2 Movielens RMSE using different methods

The second dataset to run our methods on, was the Movielens dataset. We tried all methods with varying time aggregations, to see if the RMSE of each method changed with respect to the time aggregation used. Note that we used RMSE rather than accuracy. We used RMSE because it'll give us a better idea of how accurate our predicted rating was, compared to the actual rating made by the user. LibFM performed the worst, and that is because LibFM doesn’t work as well as BPMF, and BPTF on sparse data, and the Movielens dataset is very sparse. BPTF gave us the lowest RMSE, and thus it performed the best out of all three methods. The lowest RMSE achieved by BPTF, was when the time aggregation was in terms of weeks. Movies, unlike social media trends, take a couple of days to a week to start trending. For example, romance movies will start trending during valentine's week, and not necessarily during a specific day of that week.

Timestamp Aggregation vs RMSE (Movielens)
4.3 Netflix RMSE using different methods

Our third test was on the Netflix dataset. Note that we again used RMSE rather than accuracy. LibFM performed the worst, and for the same reasoning provided earlier, which is that the Netflix dataset is very sparse, and LibFM doesn’t perform that well on very sparse data. BPTF gave us the lowest RMSE, and thus it performed the best out of all three methods. The lowest RMSE achieved by BPTF, was when the time aggregation was in terms of weeks, the reason behind that, is as mentioned above, movie take a couple of days to a week to start trending. One thing to note here, is that the RMSE for the Netflix dataset was generally higher than the RMSE achieved for the Movielens dataset, and thus we achieved better accuracy on the Movielens dataset, than we did on the Netflix dataset. This is because the Netflix dataset is very imbalanced, which means that it includes many users who have made few ratings. The more movies a user rates, the easier it is to recommend a new movies to that user. The Movielens dataset was cleaned out, any user that made less than a certain number of ratings was deleted. Thus all the users left in the Movielens dataset were users that have made a lot of ratings, which makes it easier to make movie recommendations for those users.
Figure 7: Netflix. Performance of LibFM, BPMF, and BPTF on Netflix dataset, with different time aggregations.

Timestamp Aggregation vs RMSE (Netflix)
5.1 Conclusion

After running all our tests, using different methods (LibFM, BPMF, BPTF), on different kinds of datasets (KDD, Netflix, Movielens), and seeing the results achieved by each method on each of the datasets with varying time aggregations, we draw the following conclusions. Different collaborative filtering methods will work better on different datasets. We saw that LibFM doesn’t work that well on very sparse data, while BPMF, and BPTF do. One thing we didn’t see in our tests, but is still true, is that LibFM works very well on high ranked data, while BPMF, and BPTF assume low rank. The other main conclusion we draw from our tests is that different time aggregations work better on different datasets. If your data is a social media dataset, then the best time aggregation to use is between a couple of hours to a day, meanwhile if your dataset is a movie recommendation dataset, then the best time aggregation to use is between a couple of days to a week.


