YELP BUSINESS RECOMMENDER SYSTEM

Yuan Jiang, Harry Qi

1 INTRODUCTION

We aim to build a business recommender system for Yelp. We aim to predict a random user's rating of a random business, regardless of whether the user or the business has made or received any ratings. The prediction is based on available previous ratings made by users on businesses. We use matrix factorization as our algorithm to perform this task, and compare that to baseline models. So far, we have successfully implemented the baseline algorithms - weighted average and neighborhood - and matrix factorization on our data set, and have achieved promising initial results. We are definitely on schedule, and will refine our algorithm to improve our result in the upcoming days leading up to the final deadline.

2 PROGRESS SUMMARY

2.1 SUMMARY OF PROPOSED MILESTONE GOALS

As stated in the project proposal our milestone goals can be summarized as:

"Upon the time of milestone, we will have explored the training data, implement the code for both baseline and advanced algorithms, for future tuning and improvement."

So far we have successfully completed most of our milestone goals: finish processing all the data, building baseline and actual models, obtain results, and improve performance on them.

What we haven't been able to do was cross validation (although we wonder if it is really necessary), and the sole reason for this was it takes around 7-8 hours to run our algorithm on the millions of data, and we haven't had time to perform cross validation.

Therefore, our efforts can mainly be divided into four parts:

(1) processing data
(2) Write two python script for baseline models
(3) Write a python script for preliminary matrix factorization model
(4) obtain results and improve performance

We will devote the rest of this section in explain in detail these parts.

2.2 DATA EXPLORATION & PROBLEM FORMALIZATION

Data exploration is the first step and also critical. Initially we are given four dataset, which are business profile dataset, user profile dataset, review dataset and user login history dataset. We decided to first abandon the login history dataset both becasue the set is not well formated and because that we are checking our results against a given test review dataset whose actual ratings are not revealed by Kaggle, instead of checking against the users' record along the timeline. Therefore we can formalize the problem as follows: based

on three matrices, i.e. user-item rating matrix (R_{train}), user feature matrix (F_{user}) and item feature matrix (F_{item}), our RS should be able to learn a final rating matrix ($R_{predict}$).

A python script is written for statistical purpose. Counter-intuitively, we found in both training sets and testing sets, the set of distinct users/items appeared in feature matrices, is a subset of that appeared in rating matrices. We also found that among the 45981 distinct users in training user feature matrix and distinct 15001 users in testing user feature matrix, the overlapped users are only 5017, and for items (or businesses), the three numbers are 11537, 8341, 5544. That means about 66% of users and 25% of items in the final testing set are "new" to our recommendation system. Finally, we also found that missing entry in a matrix is a common thing for feature matrices. The above findings would pose challenges later on, and they are also part of the reason that we decided to start only on the rating matrix first. Because the feature matrices are not always necessary, though can be a plus if utilized well.

2.3 Building Weighted Average Model

The first method we are trying to build is weighted averages. As explained in previous section, we already detected potential "cold start" and decided the {user-item} pairs into four groups: 1. both user & item appeared in training rating matrix 2. only user appeared 3. only item appeared 4. both are new to the system. We assign each a weighted sum of user's average rating, item's average rating, and global average rating. The weighting coefficient can significantly affect the results, which would be discussed later. A python script is written to predict the results and output the results in csv format file so that we can upload them to Kaggle's website for RMSE evaluation.

2.4 Building Pearson Correlational Neighbourhood Model

The second model is based on the idea of neighbourhood. Neighbourhood methods can either center on user-side or item-side. We chose to implement the user-side model (though later we find item-side is better). The key of the model is to calculate a similarity matrix between each combination of user pairs. Among the three popular methods used for similarity estimation, i.e. cosine similarity, Pearson correlation, and modified cosine similarity, we chose Pearson correlation. The equation for Pearson correlation and the final estimation equation are as follows:

$$\hat{r} = \overline{r} + \frac{\sum_{v \in S(u,K) \cap N(i)} W_{uv}(r_{vi} - \overline{r_v})}{\sum_{v \in S(u,K) \cap N(i)} |W_{uv}|} \tag{2.1}$$

$$W_{uv} = \frac{\sum_{i \in I}(r_u i - \overline{r_u})(r_{vi} - \overline{r_v})}{\sqrt{\sum_{i \in I}(r_{ui} - \overline{r_u})^2 \sum_{i \in I}(r_{vi} - \overline{r_v})^2}} \tag{2.2}$$

$S(u,K)$ denotes the set of K users who are most similar to user u. $N(i)$ is the set of users who have rated item i. r_{vi} is the rating given by user v to user i. $\overline{r_v}$ denotes the average rating of user v.

During implementing the algorithm, we have faced two major challenges. First is that we came across cases where (2.2) fails by dividing a zero. We later found out it happens when the user gave a single piece of review, or that the user gave all previous rated items a single same rating. Either case would lead to failure. We address the case by letting W_{uv} to be zero, meaning that no explicit correlation when either case happens. We think it make sense but we are also open for any other suggestions. Another difficulty is still "cold start". The algorithm simply does not make sense for any user/item that are new to the system. That means we still have to assign weighted averages when these cases happens. A python script is written for implementation purpose.

2.5 Building matrix factorization model

If time permits we would improve our two simple memory-based models mentioned above, but first we decided to implement a more sophisticated matrix-factorization model and review the obtained results. "cold-start" problem can be well addressed in such model-based algorithms, at the expensive of being computationally expensive. Specifically, in our model, we want to find matrices P and Q such that we can estimate the known rating matrix R_{train} using $\hat{R}$, where $\hat{r_{ij}} = p_i^T q_i = \sum_{k=1}^{K} p_{ik} q_{kj}$. The error of each element, $e_{ij}^2 = (r_{rj} - \hat{r_{ij}})^2 = (r_{ij} - \sum_{k=1}^{K} p_{ij} q_{kj})^2$. Then, by taking derivatives with respect to each p_{ik} and q_{ik}, with an inclusion of regularization term, we find that the updated rule of $\hat{R}$ using gradient descent is:

$$p'_{ik} = p_{ik} + \alpha(2e_{ij} q_{kj} - \beta p_{ik}) \tag{2.3}$$

$$q'_{kj} = q_{kj} + \alpha(2e_{ij} p_{ik} - \beta q_{kj}) \tag{2.4}$$

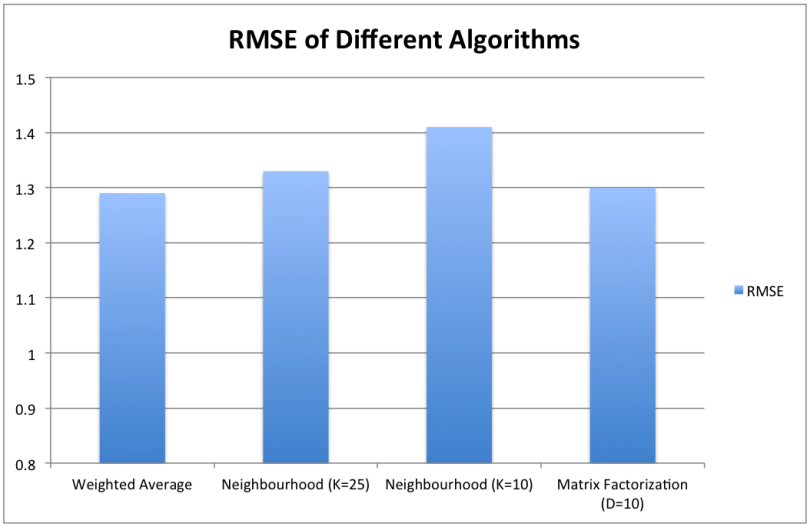
We then use sum of squared errors to determine when the program should stop (when the sum decreases by less than 0.001), it's time for the program to stop optimizing, we also set a maximum step of 5000.

Same as above, we implemented this algorithm using python, using only NumPy and SciPy packages for basic manipulation but not the popular Scikit-Learn.

3 Results & Discussion

3.1 Initial Results

Our initial result of matrix factorization is fairly good but not as desirable as we want it to be. We obtained and RMSE of 1.29 for the simple weighted average baseline model and 1.33 for the neighborhood model, choosing K to be 25. For the matrix factorization model, we obtained a RMSE of 1.30, choosing D to be 10 (number of latent features in matrix factorization), which is slightly higher than the simple weighted average model. That means for now, the simplest method actually performs the best. This is not very satisfactory, and we wish to improve our results to around RMSE 1.23 (the top on leaderboard in Kaggle is about RMSE 1.21) in our future endeavor. Below is the table of results so far. Unfortunately we can not give much visualization of the results, because it is simply just an RMSE for the testing dataset. We would later let the python program output objective value at each iteration for visualization purposes.



3.2 Results Reflection

We would present our thoughts and discussion on the results on the three models seperately. For the weighted average model, we have noticed that a balanced rating often gives us a smaller RMSE. For example, in the case that both user and item exists in learning datasets, we let the predicted rating be the average of user's average rating and item's average rating, and this gives us much better result (RMSE 1.29), compared to another case where we give user average weighting of 0.1, and item average weighting of 0.9 (RMSE in this case is 1.46, the worst), or the opposite.

For neighbourhood model, as mentioned above, Pearson correlation may not be a optimal choice, and that if time permits we would try modified cosine similarity instead. Also the results can presumably be improved by implementing a item-side method, so that the "sparsity" of the matrix is significantly reduced, since the number of items is only 1/5 of the number of users in given training rating matrix (R_{train})

For the matrix factorization model, our results are not quite as good. Not to mention that it takes over 7 hours to learn the model. Partial reason is that although we have implemented the algorithm, we haven't done much tuning yet because of the relatively long running time. But more important underlying reasons, as we think, are given in the following section.

First, we haven't used some user features and business features that are provided; such as the number of times a user has rated businesses and locations of businesses. In order to use such features, we will need to divide our testing data into groups, just like we did with our baseline models: when both user and business are in training data, when either is, and when neither is. We will only be able to use these additional features in the first 3 cases, which only constitute about one-third of our testing data. Therefore, we will need to also improve the algorithm in other ways to improve our performance on the rest two-thirds of the testing data. One way we can do this is to increase D, the number of latent features, in the matrix factorization procedure. However, as D increases, the running time increases as well, and too large a D would not be suitable since it already costs several hours to run with D = 10. Thus, we hope to look into algorithms such as SVD ++ to reduce our running time so we can model with larger D's for better results. After reducing the running time, we can implement cross validation to check for our results instead of using the testing data directly. Thus, there are two keys to improve our algorithm: find a way to use unused user and business features in our algorithm for a particular group of testing data (those that appear in the training set: one-third of them), and to reduce the running time.

4 Future work

We would try building SVD++ as our next model based on current matrix factorization model. But still, before that it is essential for us to try to figure out a better way to utilize the feature matrices. We may divide the user-item pairs into more groups and train some of them using different yet simple algorithms, and train some others using SVD++. Having looked at the Netflix prize papers, we feel like that in most cases better results in recommendation system are achieved using a combination of models, instead of an single advanced algorithm.

REFERENCES

[1] Xiaoyuan Su and Taghi M. Khoshgoftaar, *A Survey of Collaborative Filtering Techniques*, Advances in Artificial Intelligence, 2009

[2] Yehuda Koren, Robert Bell and Chris Volinsky, *Factorization Techniques for Recommender System*, IEEE Computer Society, 2009

[3] GÃąbor TakÃącs et al (2008). *Matrix factorization and neighbor based algorithms for the Netflix prize problem*, In: Proceedings of the 2008 ACM Conference on Recommender Systems, Lausanne, Switzerland, October 23 - 25, 267-274.

[4] Patrick Ott (2008).*Incremental Matrix Factorization for Collaborative Filtering.* , Science, Technology and Design 01/2008, Anhalt University of Applied Sciences.

[5] Daniel D. Lee and H. Sebastian Seung (2001). *Algorithms for Non-negative Matrix Factorization*, Advances in Neural Information Processing Systems 13: Proceedings of the 2000 Conference. MIT Press. pp. 556âĂŞ562.

[6] Daniel D. Lee and H. Sebastian Seung (1999). *Learning the parts of objects by non-negative matrix factorization.*, Nature, Vol. 401, No. 6755. (21 October 1999), pp. 788-791.