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. We have successfully implemented the baseline algorithms - weighted average and neighborhood - and simple matrix factorization, SVD, and Hybrid SVD neighbourhood on our data set. We have achieved promising results with our SVD model.

2 ACHIEVEMENT SUMMARY

2.1 SUMMARY OF IMPLEMENTATIONS

Our project goals can be summarized as:

"Upon the completion of project, we will have explored the training data, implement the code for various baseline and advanced algorithms, make reflections and comparisons across different models."

We have successfully achieved the following:

(1) Processing data
(2) Write a python script for data feature exploration
(3) Write two python script for baseline models
(4) Write a python script for preliminary matrix factorization model
(5) Write a python script for advanced matrix factorization model
(6) Write a python script for Hybrid SVD neighbourhood model
(6) 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

After building our baseline models, we implemented a more sophisticated matrix-factorization model and reviewed 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.

2.6 Building SVD model

We then continued to develop a more advanced model using Singular Value Decomposition. SVD requires us to finally obtain helps reduce the matrix dimensions after the computation of three reconstructed matrices. We use largest gradient ascent as objectives to train the model. Specially, the SVD algorithm would obtain two matrices $U \in R^{f \times n}$ and $V \in R^{f \times m}$, where m, n denotes the number of items and the number of users respectively. f is the dimension of SVD.
The prediction function p is given by: $p(U_i, V_j) = {U_i}^T V_j$, therefore this becomes a matrix factorization problem with the predicted rating matrix $R_predict \approx U^T V$, while the entries might be less than 0 or higher than 5, in our implementation use just used confine them to the range [0, 5].
The optimization of U and V is found by minimizing the sum of squared errors between the existing scores and their prediction values. Specifically, the objective is to minimize the following:

$$E = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{m} I_{ij} (R_{predict} - p(U_i, V_j))^2 + \frac{k_u}{2} \sum_{i=1}^{n} ||U_i||^2 + \frac{k_m}{2} \sum_{j=1}^{m} ||V_j||^2 \tag{2.5}$$

In the training process, we would use largest gradient descent to train our algorithm. We set the iteration to be 100 and 5000 to observe the results.
In addition, our SVD model can be perceived as only the improvement on one subset of the provided testing set by kaggle. Since we can only as before assign weighted average to those unknown user-item pairs, we could only use SVD when our trained system 'knows' already the user-item pair that we want to predict.

2.7 After Poster Presentation: Building Hybrid model

Specifically after poster presentation, as suggested by the professor that we should consider a combined approach of our algorithms, we implemented an extra hybrid model based on SVD to observe the results. By Hybrid we mean a combination of SVD and neighbourhood model. This is actually one step further compared to previous SVD model, in that it first does the exact same SVD, and then it uses the following equation to compute

the predicted rating:

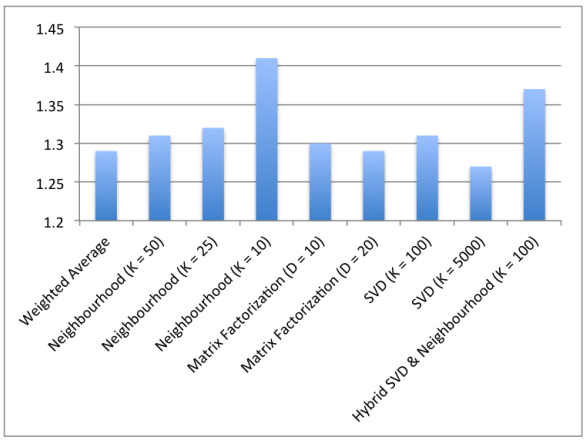
$$\hat{u_i j} = \frac{\sum_{j \in S^k(i;u)} S_i j r_{uj}}{\sum_{j \in S^k(i;u)} S_i j} \tag{2.6}$$

Where $S^k(i;u)$ denotes the set of k items rated by user u that are most similar to item i, and s_{ij} denotes the similarity accordingly.

3 Results & Discussion

3.1 Results

Our 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 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.

However, the SVD model performs better than the first matrix factorization model we implemented. When we choose $K = 5000$ and train our SVD algorithm, we obtain a testing RMSE of 1.27, which is by far the best of all. Although the SVD model performs better, the hybrid neighbourhood SVD actually performs worse (with RMSE of 1.37 compared to 1.34 without using it when $K = 100$). This may be due the scarcity of our data causing the neighbourhood to not work as well compared to a simple SVD. The improvement of SVD is not very significant however, mainly due to the fact that a large portion of testing set provided by kaggle belong to cold-starts, with which we simply use weighted average to deal.
To further demonstrate its actual effectiveness, we then split the testing dataset into 80% training and 20% for testing. In this way, we achieved RMSE as follows using the RMSE and MAE module from numpy:

	SVD	SVDNeighbourhood
RMSE	0.92	0.88

4 FUTURE WORK

As suggested by professor, we could also implement the rankSVM to generate top-N recommendations. The would be an interesting way, yet the only concern is that we lack a good methodology to evaluate how good the recommendation list itself, as opposed to calculating RMSE of predicted ratings against testing ratings. Also, because SVD, or matrix factorization model are generally computationally expensive, we may try another way, which is to generate more features from the given user and item feature matrices, and develop the model still based on the idea of neighborhood.

5 IMPLEMENTATION DETAILS

So far We have implemented five models in python, i.e. weighted average, correctional neighbourhood model based on pearson correlation on the user-side, matrix factorization model, SVD model, Hybrid model combining SVD and neighbourhood.
Below is a list of python packages we have imported during coding process:
$JSON, math, operator, collections, pylab, numpy, scipy, recsys, time, panda$
In the first matrix factorization model, we have also refered to Albert Au Yeung's article on implemention suing gradient descent.
We didn't use existing training software for our project. All the other codes are hand-coded by ourselves.

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.

[7] Chih-Chao Ma, *A Guide to Singular Value Decomposition for Collaborative Filtering*