CS 567 Project Report

Qijia Zhou
Department of Computer Science
University of Southern California
Los Angeles, CA 90089
qijiazho@usc.edu

Luke Yang
Department of Computer Science
University of Southern California
Los Angeles, CA 90089
yifeiyan@usc.edu

Calvin Legassick
Department of Computer Science
University of Southern California
Los Angeles, CA 90089
legassic@usc.edu

Abstract

In this paper, we detail our team's efforts in the ByteCup-2016 Competition. We discuss the data and our preprocessing, our algorithmic approach, the implementation details, and our results. Our focuses on applying Collaborative Filtering (CF) methods to tackle the challenge of recommending and distributing questions to users in an online question answering community. Ultimately, we combined a neighborhood model with a model for implicit feedback along with information derived from computations on invitation matrices. We find that this approach is competitive and a successful solution.

1 Introduction

In this competition, we researched a family of strategies based on collaborative filtering, and selected a few ideas to implement. Under the screen name `Calvin`, we reached an evaluation score of 0.4800001 with our final submission, ranking 42 of 129 teams in total according to the ByteCup final rankings. Further, at this point in the leaderboard and above, the leadings scores are very close together and variance is very small. This suggests that our method is a competitive approach.

2 Data Processing and Cleaning

2.1 Database Efforts

This subsection discusses a part of our early efforts on the project that we eventually deprecated.

At the beginning of our efforts on this project, we built a `SQLite`-based database using `SQLAlchemy` and dumped all provided data set into the database. We had three reasons:

- A database provides persistent storage of the data set. Once the data are loaded into the data set and the upfront time cost paid, all subsequent research engineering would directly take off from the database and wouldn't need to waste time on reading and parsing the data set.
- A database provides fast access to structured data, whose format we have defined for the various SQL tables we have.
- A database allows for fine-grained lookup operations that would make it easier to uncover patterns that connect data in different tables.

29th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain.

However, as we progressed, we discovered using `Numpy` alone was not a heavy performance drawback to deal with, and the capabilities it provided were more convenient that interacting with a database layer. More importantly, many linear algebraic operations would have to rely on `Numpy` and we would have to export data from our homegrown database to `Numpy` data structure anyway. Therefore, we switched to using `Numpy` directly and deprecated the database.

In hindsight, the story around our effort on the database gives us a good lesson on the engineering side of the development of a machine learning project. While our database later proved to be unnecessary, nevertheless we have heard success stories from classmates who used C++ and CUDA to rewrite part of their computing architecture and managed to achieve a boost in performance and productivity through parallelism. It is obvious that, especially under time and computation resource constraints, the software engineering part of a machine learning project is tantamount to the research and implementation of an algorithm. A sound decision on what part of the research architecture to build from scratch and what third-party libraries to bring in requires much experience and careful thought. It is certain working on this real-world machine learning challenge benefits us in many ways; in this particular instance, we gained precious experience on the engineering side of machine learning applications.

3 Details on Learning Algorithms Experimented

3.1 Experiment on a User Bias-based Model

To understand better the problem and competition environment, we tried to predict the validation set with only user bias. That is, for each tuple in validation set, we just focus on the user and used this user's bias b_u (average answer rate for user *u*) to predicate whether the invitation should be accepted or not. Equation (1) below describes the user bias:

$$b_u = \frac{1}{|R(u)|} \sum_{i \in R(u)} a_{ui} \tag{1}$$

Here, $|R(u)|$ is the set of questions that have been recommended to user *u* and a_{ui} is the indicator variable on whether user *u* answers question *i* or not. If *u* does, then a_{ui} is 1, otherwise it is 0.

This experiment implements a simple idea, yet it yielded an NDCG score of 0.459590 on the validation set; assuming similar performance across valdiation and test, this simple model would still outperform around 20% of the competition. This provided us with a better understanding of the data set and the competition.

3.2 Experiments with User Based Collaborative Filtering

The GroupLens algorithm by Resnick et al in [2] presents a sensible way to perform collaborative filtering on news, and helps users find news articles they might be interested in. The algorithm makes a quantitative prediction on how much a user would rate a given article, based on known ratings multiple other users gave for different news items.

This algorithm builds on the idea that "people who agreed in their subjective evaluation of past articles are likely to agree again in the future." In other words, if two users, *A* and *B*, have both given numerical ratings to some items and the ratings from *A* and *B* demonstrate a certain level of similarity, then *A*'s rating on item *X* can be used as a good predictor on *B*'s rating on the same item (assuming user *B*'s rating missing) and vice versa. On the other hand, if *A*'s ratings and *B*'s are weakly correlated, then the predictive power of *A*'s rating on *X* with regard to the projection of *B*'s on *X* is low. Numerically, *A*'s predicted rating on *X* is computed as a sum of two parts, where the first part is the average rating from all known rating data from *A*, and the second part is a weighted average for the differences between each known rating from other users and the average known rating for *X*. See equation (2). The weights are the Pearson correlation factor, with values between -1 and 1, computed pairwise between *A* and each other user who has given a rating for *X*. Therefore, the projected rating takes into account both *A*'s overall rating level (average) and known ratings for the object from other users. As mentioned above, those users who appear to be frequently in agreement with *A* would have a larger influence and those who disagree often can shift the sum only a small amount.

$$A_X^{\text{proj}} = \overline{A} + \frac{\sum_{B \in \text{raters}} \left(B_X - \overline{B}\right) r_{AB}}{\sum_B |r_{AB}|} \tag{2}$$

A major advantage of this algorithm is that it embodies a very intuitive idea. The algorithm is simple and computationally feasible. Consequently, we decided to give it a try. In our scenario, we used the known invitation (push notification) history data to predict the likelihood a particular user should answer a particular question. By doing so, we admit the hypothesis that users who frequently pick up the same questions are likely to respond to the same question again. Our model would solely rely on this idea and set aside the tag and word information, since we believe the users' response record is the best predictor for responses upon future notifications.

Of course, our data set differs from some of the underlying assumptions made in the original paper. While the ratings from the paper's sample data are on a continuous positive integer scale, our training data is composed of binary labels — whether or not the notified user answered the invitation. This binary feature caused our data points to be very sparse. After we expanded the invitation record (which can be thought of as a sparse matrix expressed in its $(\text{row}, \text{column}, \text{value})$ form) into a standard matrix, we found that a majority of the data points to be zeros or missing (which we should be able to predict). The major problem with this prevalence of zero and missing terms is that the correlation became very difficult to compute. It was problematic because many computations would lead to `NaN` as a result.

In order to mitigate this problem, we made several tweaks to the algorithm. Only during the computation of similarity between two vectors, zero terms stayed as zeros but unknown terms were substituted by the average of all known values in the vector. Also, if the Pearson correlation coefficient couldn't be computed, the cosine of the angle between the two vectors was used. These tweaks were chosen through practical inspection rather than derived from theory. Despite a lack of theoretical foundation, these operations were indeed able to reduce the number of computation exceptions. Ultimately, the algorithm running on the unlabeled validation data set saw its output scored as 0.458845.

It turns out this hypothesis does not make the strongest predictor on the users' responses. Besides the non-conforming binary and sparse data set, two other reasons may have contributed to this result. One reason is obvious: a user history-based model could only go so far without analysis on the words and the tags. The other reason is that the sum term computed by the algorithm in the form of $a + b$, may actually be better modeled as $\alpha a + \beta b$ where α and β are parameters that could be tuned using empirical methods. With these understands in mind, we continued to build more sophisticated models.

3.3 Experiment on a Neighborhood Model

The Neighborhood Model in [1] provides a method to solve the problem. As do most common neighborhood models, we transferred `invited_info_train.txt` into a user-question matrix M_{uq} where m_{ij} refers to user whether U_i answers question Q_j or not. If the user does, then m_{ij} is 1, otherwise it is 0. What we aimed to do is measure item similarity based on this matrix. The intuition is that a user is more likely to answer a question if the user has answered similar questions before. Then the key to the problem is determining how to evaluate item similarity.

Our answer to this can be divided into two parts based on the input data: the user-question matrix and the question profile, respectively. For the user-question matrix, with the given data, we can extract each pair of the question vectors from the M_{uq} and calculate their Pearson correlation as the similarity. As for the user profile, we include it into our consideration because we would like to extract as much useful information from the data we have as possible. We then calculate the profile similarity based on common question tag and common question description words. Equation (3) shows the final similarity computation:

$$w_{ij} = \alpha_1 \frac{n_{ij}}{n_{ij} + 100} \rho_{ij} + \alpha_2 \frac{n_{st} + n_{sd}}{n_t + n_d} \tag{3}$$

Here, n_{ij} is the number of the users that answered both question Q_i and question Q_j. ρ_{ij} is the Pearson correlation between question Q_i and question Q_j. n_{st} and n_{sd} are the number of same tags

and description words. n_t and n_d are the number of the tags and description words that appears in both questions. α_1 and α_2 are the linear combination coefficient.

Thus we have following equations (4) and (5), as per [1], for the measurement of the probabilities.

$$\hat{a}_{ui} = b_{ui} + \sum_{j \in R(u)} (a_{uj} - b_{uj}) w_{ij} \tag{4}$$

Here, $\hat{a}_{ui}$ is the probability that user U_u answers question Q_i and b_{ui} is the bias item for user U_u, question Q_i and overall bias b, i.e.:

$$b_{ui} = b_u + b_i + b \tag{5}$$

In our actual neighborhood model, we did not establish such a large matrix. Instead we stored components in a hash table where $(\text{user\_id}, \text{question\_id})$ is the key used to access the items more quickly while keeping the matrix more compact.

This method yielded sub-optimal performance on the validation set, with the NDCG valued at 0.4142038. This is why we have not included the source code for this implementation. Based on this version, we improved Neighborhood Model with implicit feedback, as detailed in the next section.

3.4 Neighborhood Model with Implicit Feedback

Based on the previous Neighborhood Model, we decided to take into account some Implicit Feedback. The decision to use this model came from the observation that some users are never invited for certain questions. We hypothesize that they are less likely to answer questions that are similar to questions they chose not to answer before. Therefore we introduced a c_{ij} term into the above equation (4), as follows in (6):

$$\hat{a}_{ui} = b_{ui} + \sum_{j \in R(u)} (a_{uj} - b_{uj}) w_{ij} + \sum_{j \in N(u)} c_{ij} \tag{6}$$

Here, $N(u)$ is the uninvited question set for user U_u. c_{ij} is the penalty term between question Q_i and question Q_j.

In fact, we used the negative similarity multiplied by a combination coefficient as the penalty term in our model. Finally, we linearly combine the $R(u)$ sum and $N(u)$ sum with some coefficients, which gives equation (7).

$$\hat{a}_{ui} = b_{ui} + \alpha_1 \cdot \sum_{j \in R(u)} (a_{uj} - b_{uj}) w_{ij} + \alpha_2 \cdot \sum_{j \in N(u)} c_{ij} \tag{7}$$

Here, coefficients satisfy $\alpha_1 + \alpha_2 = 1$.

With this method, we got a better result (0.443666), but continued to make further improvements.We continued applying our neighborhood model on the user space, in the same way we applied it to our neighborhood model on the question space.

3.5 Neighborhood Model with Implicit Feedback on User Space

This method is almost the same as that in Section 3.4 except the computation is performed on the user similarity. This method replaces the item similarity with user similarity. The rationale behind this change is that our first experiment with only user bias showed a better result than that of Section 3.4. Hence, we would like to integrate the consideration of user space into our model.

$$\hat{a}_{ui} = b_u + b_i + b + \alpha_1 \cdot \sum_{v \in R(i)} (a_{vi} - b_{vi}) w_{uv} + \alpha_2 \cdot \sum_{v \in N(i)} c_{uv} \tag{8}$$

In equation (8), $R(i)$ is the user set where all the users are invited to answer question Q_i. $N(i)$ is the set of all users not invited to answer question Q_i. Definitions for b_u, b_i and b are the same as they are in equation (5). Meanwhile,

$$w_{uv} = \alpha_1 \frac{n_{uv}}{n_{uv} + 100} \rho_{uv} + \alpha_2 \frac{n_{st} + n_{sd}}{n_t + n_d} \tag{9}$$

In equation (9), ρ_{uv} is the Pearson correlation. The meanings of n_{st}, n_{sd}, n_t and n_d are the same as those in equation (3). On the user profile. α_1 and α_2 are the coefficients which we empirically chose to be 0.67 and 0.33, respectively. As in equation (6), c_{uv} is the penalty item, subtracting from the score if a similar user were not previously invited.

This method led to our best results on the validation set, 0.465539.

4 Source Code and Implementation Details

Details on source code available and submission scores are listed in Table 1 and Table 2. We developed some code segments and obtained results in a notebook-like, more volatile environment. At this point some of the artifacts can't be retrieved and are thus unavailable.

Table 1: Validation Submission

Algorithm	Source file	Output file	NDCG score
User Bias	`bias.py`	`validate_result_user_bias.csv`	0.45959
User Based CF	`user_collab_filter.ipynb`	N/A	0.45885
Neighborhood Model	N/A	`item_based_weighted_414.csv`	0.41420
NM with Implicit Feedback	`item_based_nm.py`	N/A	0.44367
NM with IF on User Space	`user_based_nm.py`	`user_based_465.csv`	**0.46554**

Our model has excellent performance and can be trained in 3 hours with only local computation, thanks to our sparse matrices and hash table implementation (instead of huge invitation matrices).

Table 2: Test Submission

Algorithm	Source file	Output file
NM with Implicit Feedback	`item_based_nm.py`	`test_item_based_result.csv`
NM with IF on User Space	`user_based_nm.py`	`test_user_based_result.csv`

Final Score on Test: **0.4800001**

5 Conclusion

In this project, we mainly focused on implementing the Collaborative Filtering algorithm, which we believe to have a good performance for this challenge. We prefer to use a user-based method to evaluate the answer probability because our initial experiments showed pure user bias led to promising results. This observation provided a solid foundation for our later developments. In our later experiments we also considered the implicit feedback in our neighborhood model. However, we recognized that further complicating the model was not the best approach for performing well on the validation set. Therefore, we followed our intuition towards the completion of the competition and avoided the slippery slope of overfitting. Ultimately, our results on the testing set slightly exceeded our expectations.

References

[1] KOREN, Y. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining* (2008), ACM, pp. 426–434.

[2] RESNICK, P., IACOVOU, N., SUCHAK, M., BERGSTROM, P., AND RIEDL, J. Grouplens: an open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM conference on Computer supported cooperative work* (1994), ACM, pp. 175–186.

6 Separate Page for Algorithm and Test Scores

Table 3: Leaderboard Submissions

Algorithm	Source file	Output file	NDCG score
User Bias	`bias.py`	`validate_result_user_bias.csv`	0.45959
User Based CF	`user_collab_filter.ipynb`	N/A	0.45885
Neighborhood Model	N/A	`item_based_weighted_414.csv`	0.41420
NM with Implicit Feedback	`item_based_nm.py`	N/A	0.44367
NM with IF on User Space	`user_based_nm.py`	`user_based_465.csv`	0.46554

Final Score on Test
NM with IF on User Space: **0.4800001**