Machine Learning CS 567 Report

Rajbharath Rajendran (rajbharr@usc.edu)
SenthilKumar SampathKumar (senthils@usc.edu)
Sneha Srinivasan (snehasri@usc.edu)

Abstract

In this report, we describe our approach to competing in the **2016 Byte cup** competition. The challenge is to develop models to improve the strategy for matching users and questions in **Toutiao Q&A**, a mobile social platform promoting short form content creation and interaction on mobile devices in the format of Q&A. An accurate matching strategy will help in targeting the right set of experts for soliciting answers. First we introduce the problem and describe the dataset. Then we describe the machine learning models we used along with the **preprocessing and feature engineering required**. Finally, we discuss the results obtained along with some insight into how the performance can be further increased.

1 Introduction

Our aim is to develop recommender system models for improving the matching strategy of Toutiao Q&A, a mobile Q&A platform. Toutiao has a huge user base of 530 million users. It has tens of thousand users answering on a daily basis and such user created answers attract millions of views. Matching questions to users based on their interests and expertise is a crucial factor contributing to content generation, usability and growth of the platform. It's a challenging problem as the user-question interaction matrix is usually sparse.
We investigated recommender system techniques[1] and started with simple content based recommendation system strategies. As we continued our exploration, we tried collaborative filtering techniques and hybrid methods. Our best validation leaderboard score was achieved through Matrix factorization techniques which are the state of the art in recommender systems.

2 Data

The training dataset provided contains about 245,752 records with identifiers of users and questions. The data gives information about whether a user answered a particular question whenever a push notification was sent to him/her. There was about 218427 records where the user had not answered the question(answered=0) and 27324 records where user had answered the question(answered=0). Clearly, there is an imbalance in the data and this has potential to affect the model. Below, we describe the features of the users and the questions in detail.

2.1 Expert tag data

The expert tag data consists of details about each expert in the system with anonymised ids. Each line the dataset represents a unique user with the following properties:

- *Anonymized expert user ID:* Anonymised unique identifier of the user
- *Expert user tags:* Represents the expertise of the user. Some example analogies: Science/Astronomy

Submitted to 29th Conference on Neural Information Processing Systems (NIPS 2016). Do not distribute.

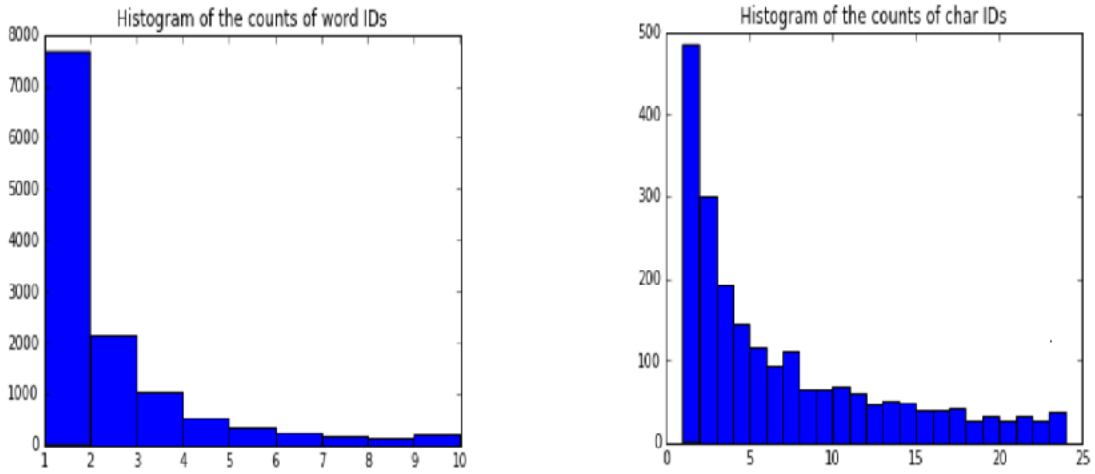

Figure 1: Distribution of word-ids and char-ids w.r.t qtag

- *Word ID sequence:* Represents the user description which is tokenized
- *Character ID sequence:* Same as word ID sequence, but tokenized at a character level

2.2 Question data

The question data contains details about unique questions and their linguistic features. The various features are described below:

- *Anonymized question ID:* Anonymized unique identifier of the question
- *Question tag:* Represents the category/domain into which the question has been tagged
- *Word ID sequence:* Represents the user description which is tokenized
- *Character ID sequence:*Same as the word ID sequence, but tokenized at a character level
- *Number of upvotes:* Total number of upvotes of all answers to this question. It may indicate the popularity of the question
- *Number of answers:* Total number of answers to this question
- *Number of top quality answers:* Total number of top quality answers to this question

2.2.1 The following are the statistics we collected from the data:-

- There are 8095 unique questions and 28763 unique users.
- There are 20 unique Question Tags(qtags) and 143 unique Expert Tags(etags). This shows questions and users do not use the same set of tags.
- There are 13232 unique word-ids and 2960 unique char-ids in question data. We initially thought each word-id represents a word and each character-id represents the characters of the corresponding word. But later we realized that same word-id has different char-id in different places.
- The distribution of char-ids w.r.t qtags were better than the distribution of word-ids. This is evident in Figure 1
- Similarly there are 31100 unique word-ids and 3779 unique char-ids in user data.
- The sum of *"Number of answers"* column for all questions is 330121. This is much greater than the total number of answered question in invitation data - 27324. This tells us that the invitation data does not contain all the user question interaction records.

- There are 810 unique users who are present in validation set but not in invitation set. These users are special cases that we need to handle separately in certain models. Similarly there are 761 unique users who are present in test set but not in invitation set.

- There are 2094 unique question which are present in validation set but does not have history in invitation set. These question are also special cases. Similarly there are 2064 unique question which are present in test set but not in invitation set.

3 Methods & Approaches

3.1 Per user Logistic Regression Classifier:

As a first attempt, we wanted to try a simple classification model - to predict whether the user will answer the question in hand. As the challenge required estimating the probability of user answering the question, we decided to build a logistic regression classifier as its output can be directly interpreted as a probability value (Later, on gaining understanding of the NDCG loss, we realized it doesn't necessarily have to be a probability value as long as it predicts the correct ordering).

The question data and user data were processed to create "bag of words" feature vectors from the respective textual description columns. Since character sequence IDs and word sequence IDs represented the same information, we decided to include only one of them to create the feature vectors. On inspecting the data, we found the number of unique character sequence IDs to be much less than the number of unique word sequence IDs in case of both question and user descriptions. To keep the feature vectors comparatively dense and the dimensionality low, we chose character sequence IDs for vectorizing. For questions, we also included the features indicating popularity - #upvotes, #answers, #top-quality answers.

With this setting, we trained a logistic regression model for each user using the invitation info records specific to that user. Predictions for the validation data are obtained using the corresponding user model. Since some users in the validation data did not have any invitation records, a probability of zero was assigned to such user-question records. We obtained a validation leaderboard score of 0.45773 using this method.

Continuing our exploration with this method, we tried using "bag of words" feature vectors from word sequence IDs (instead of character sequence IDs) and it gave us a leaderboard score of 0.46071.

3.2 Per user Support Vector Machines:

Encouraged by the scores from the simple per user logistic regression model, we decided to try support vector machines as they can easily model non linear decision boundaries as well. With the same setting as the previous method, we made an initial attempt with RBF kernel and hyperparameters C=1 and gamma=0.1. This gave us a leaderboard score of 0.48580. Later we did cross validation for the best SVM hyperparameter settings and found values for C and gamma to be 0.001 and 0.1 respectively. But unfortunately we never got to try this model on the leaderboard.

3.3 Feed Forward Fully connected Neural Networks:

Neural networks and deep learning being the state of the art in many machine learning tasks, we decided to give them a try. Layers of a feed forward fully connected neural network learn distributed and high level representations of the data. We modelled our network architecture along these lines.

Our intuition was to let the network learn high level representations of the questions and users separately and then combine them to learn high level representations of user-question interactions. The last layer is a softmax layer to predict the probability of user answering the question based on the features learnt by the network. With the outlined architecture, we obtained a leaderboard score of 0.26994. We tried playing around with the network architecture for a while but couldn't manage to get a substantial increase in performance. The networks saturated at 88% classification accuracy each time. Owing to limited time, we decided to move on with other methods.

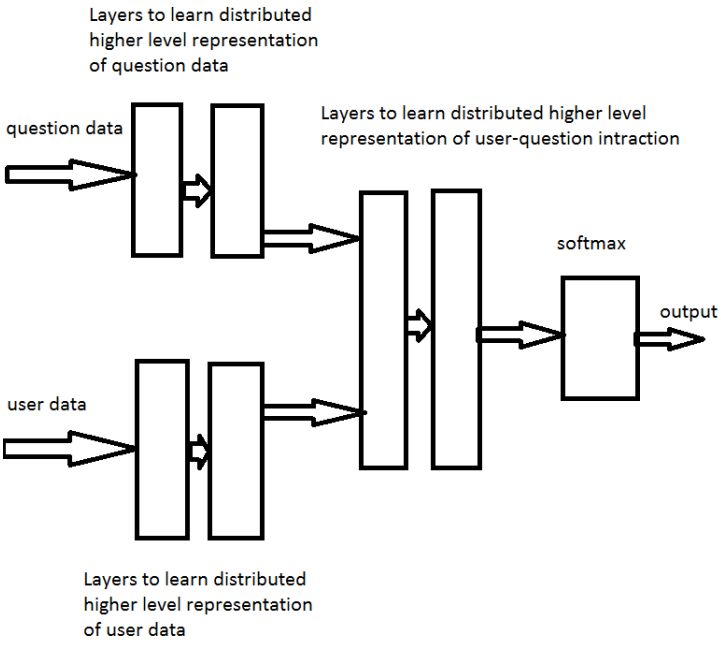

Figure 2: Neural Network Architecture

3.4 User based collaborative filtering:

From our review of various successful past implementations of Recommender Systems and as a popular method[2], we decided to try user based collaborative filtering. User based collaborative filtering is based on creating a user-item matrix which is filled with the ratings given by each user to each item available in the system. Naturally, this matrix tends to be sparse because it depends on the user's past ratings for a product and there is a tendency to often not rate products or there are a large number of products not yet purchased by the user(unseen items).

Given this user-item matrix, the idea is to predict the ratings of the user for unseen items based on his similarity with other users who have already rated the unseen items for that user. The ratings are computed using a weighted average and if there is an item that has not yet been rated by any user, the predicted rating is simply an average of all ratings made by that particular user.

For our problem, the users were experts and items being questions. We implemented the above algorithm using cosine-similarity as a metric for distance measures. The reason behind using cosine-similarity was because there was an absence of real valued ratings. Instead, we had a 0-1 label of whether the expert answered the question or not when he was invited. Cosine-similarity tends to work well with binary values and pearson correlation with real valued ratings. We then used the predicted values from the above algorithms as our predictions on the validation test. This method gave us a validation ranking of 0.46879. We used RMSE error on a test-train split to see which similarity measures perform best, but our choice of cosine-similarity gave us the lowest error value.

Based on a retrospective on this method, we believe that high sparsity could be one of the reasons for a lower performance. But this method definitely gave a significantly high score and had potential to be combined with content-based recommenders to improve the score.

3.5 Item based collaborative filtering:

Similar to User based collaborative filtering, we tried the item based approach where the similarity is computed based on the items. This involves looking at the ratings of all the items the user has rated. Then the k most similar items to the target item are taken from that set. A weighted average of these similar items is used to calculate the prediction.

Surprisingly, the item based collaborative filtering performed better than user based collaborative filtering generating a validation score of 0.48463. Again, this method suffered from sparsity issues and could not go beyond the above mentioned score.

3.6 Hybrid recommender system:

In the initial simpler classification models, we assigned 0 for users without invitation info records. From data inspection, we found that there are 810 such users (around 1:14 of unique users in validation). To handle them, we decided to try a hybrid strategy. We trained a logistic regression classifier on the entire data obtained by combining all invitation info records with the corresponding user and question features. We combined the results from SVM and used the predictions from the full logistic regression model for those users without invitation info. It gave us a score of 0.4874 in the validation leaderboard.

Motivated by results from using collaborative filtering and SVM, we then tried to create a hybrid recommender system by combining the collaborative filtering and content-based recommendation approaches hoping to resolve the sparsity issue faced by user based and item based recommenders.

The hybrid we created was a weighted hybrid where we try to optimize the weights for each recommender based on minimizing the RMSE error on the test-train split. We performed a grid search on weights ranging from 0-1 in steps of 0.1 to find the best weights for a combination of user based and SVM based recommender. Having optimized the best weights, we obtained a validation NDCG score of 0.47895.

The reason for a slight improvement and not a significant jump from user based recommendation could be because of an incorrect approach towards creating the weighted recommender. While creating the weighted recommender, we used scores from a train-test split. But since we were pushed for time, we happened to use scores from user based and SVM which could have been obtained from potentially different sets of the train-test split since we did not store the train-test split explicitly. This is a fundamental step, but this was a mistake we realized later on and could have cost us at large.

Hybrid recommenders could have definitely help if we had used the same train-test split and has good potential.

3.7 Bayesian personalized ranking:

We read about Bayesian Personalized Ranking(BPR) in one of the papers[3] which stressed on obtaining predictions from implicit behavior of the user such as clicks, purchases etc. We wanted to try this approach because it looked similar to the problem that we were to solve i.e. answering a question from a push notification can be formulated as an implicit behavior of the user, not as an actual rating of the question. Also, the paper showed excellent AUC results on the Rossman Online Shopping dataset as well as a subsample of the Netflix dataset.

Due to time constraints, we did not implement the BPR algorithm from scratch. Instead we used an existing implementation of the algorithm within LightFM which is an open source recommender library that uses implicit and explicit feedback. This method gave us a validation score of 0.24159. We tried using various measures like the AUC measure and RMSE error to improve the results. We do not have an exact criterion to analyze this method because we used the LightFM implementation. We would like to implement BPR from scratch in the future and see how well it performs on a different dataset to solidify our conclusion about this approach.

3.8 Matrix factorization:

On doing a survey of recommender systems literature[1,4], we found Matrix factorization techniques to the be state of the art. We stumbled upon GraphLab[5] which had super fast implementations for recommender system algorithms. It had multiple matrix factorization algorithm implementations - Factorization Recommender, Ranking Factorization Recommender.

Factorization Recommender trains a model capable of predicting a label for each possible combination of users and questions. The internal coefficients of the model are learned from known labels of users and questions.

Ranking Factorization Recommender learns latent factors for each user and question and uses them to rank recommended questions according to the likelihood of observing those (user, question) pairs.

We did hyperparameter search using GraphLab's APIs for both the methods. We obtained a validation score of 0.48611 using Factorization Recommender and 0.50294 for Ranking Factorization Recommender, which is the best leaderboard score we achieved.

4 Prediction accuracy analysis:

In order to be able to compute the accuracy of our predictions without depending on an online submission, we came up with various techniques to analyse our model performance. Doing this helped us and guided us in our search for improvisation and optimisation. We used cross validation to split the data into a train and validation set and for every algorithm we evaluated the quality of predictions using Root Mean Square Error, Mean Average Error and Area Under Curve. Later, we used the ndcg script and implemented ndcg scoring for our training set.

5 Results and Other Insights:

The best results we achieved on the validation dataset was 0.50294 which was ranked 38/524 and a final score of 0.48514 ranked 26/129.

One of the things we wanted to try was to train a Neural Network that optimized the NDCG loss directly instead of optimizing the surrogate cross entropy loss. This is non-trivial as NDCG loss is not smooth. On exploring, we found some papers [6] on training neural networks via direct loss minimization. But we couldn't find any off-the-shelf implementations and owing to time constraints, we couldn't explore further.

Some other algorithms that we would have loved to try and could potentially work are random forests, hybrid recommenders using feature augmentation and meta-level hybrid recommenders. Also, since the dataset had some good amount of features, trying different feature selection methods is another approach to generate a good model.

References

[1]Recommender Systems, Encyclopedia of Machine Learning, Prem Melville and Vikas Sindhwani (http://www.prem-melville.com/publications/recommender-systems-eml2010.pdf)

[2]Collaborative Filtering Recommender Systems - Ekstrand et al. (http://files.grouplens.org/papers/FnT%20CF%20Recsys%20Survey.pdf)

[3]BPR: Bayesian Personalized Ranking from Implicit Feedback - Rendle et al. (https://arxiv.org/pdf/1205.2618.pdf)

[4]Matrix Factorization Techniques for Recommender Systems - Yehuda Koren, Robert Bell and Chris Volinsky. (https://datajobs.com/data-science-repo/Recommender-Systems-[Netflix].pdf)

[5]https://turi.com/learn/userguide/recommender/choosing-a-model.html

[6]Training Deep Neural Networks via Direct Loss Minimization - Yang Song, Alexander G. Schwing, Richard S. Zemel, Raquel Urtasun. (http://jmlr.org/proceedings/papers/v48/songb16.pdf)

Execution of the code

Two system requirements to run our code:-

- *Code has to be executed in Google Cloud*. To get best performance, we implemented multi processing tasks in our code, Thus it doesn't run in single processing machine.
- *Graphlab has to be installed* We have used graphlab library in our program.

The files are named by the algorithm name. To run it , you need to pass a parameter [0 or 1]. For 0, it runs for validation set and for 1, it runs for test set. For more details, refer the file named *"RunMe.txt"* in the root folder.

Table 1: Table of methods and scores

Algorithm	Filename	Brief description	Leaderboard score
Per user Logistic Regression Classifier	per_user_logistic_model.py	Logistic Regression at user level	0.46071
Per user Support Vector Machines	per_user_svm.py	SVM at user level	0.48580
Feed Forward Fully connected Neural Networks	neural_net.py	Network learns questions and users separately and then combine them to learn high level representations of user-question interactions	0.26994
User-based collaborative filtering	user_based_recommender.py	Method of predicting user interest in products based on similar users who have rated similar items	0.4688
Item-based collaborative filtering	Itembased_recommender.py	Method of predicting user interest in products based on similar items to the target product	0.4846
Hybrid Recommender	hybrid_recommender.py	Weighted recommender using combination of predictions from collaborative filtering and SVM	0.47895
Bayesian Personalized Ranking	bpr.py	Uses implicit information to create a ranked list of items for each user	0.24159
Matrix factorization	matrix_factorization.py	Ranking Factorization Recommender learns latent factors for each user and question and uses them to rank recommended questions according to the likelihood of observing those (user, question) pairs.	0.50294