Report for CSCI-567 Project - Big Brother

Jashwanth Reddy Madem, Simrat Singh Chhabra, Jay Priyadarshi
University of Southern California
madem@usc.edu, simratsc@usc.edu, jpriyada@usc.edu

Abstract

In this report we discuss about the various approaches we tried in order to have a good performance on the ByteCup 2016 challenge. The task required us to build a type of recommendation system would predict if an expert would answer a particular question.

1 Introduction

Recommendation systems have become very popular in today's world. We see being them used in the Amazon App, Pandora, Netflix, Quora, etc. These systems are popular because everything is becoming more and more consumer centric. Pandora uses a Content-Based system to determine what type of songs a user would want to listen to next, where as Netflix uses algorithms like Collaborative Filtering as part of their system. In this report we summarize our efforts to build a recommendation system for the ByteCup 2016 challenge where we tried Content-Based, Collaborative Filtering as well as Hybrid models.
Code and execution details: `github.com/jashwanth9/Expert-recommendation-system`

2 Dataset

The dataset provided in the competition had three types of information:

- Expert tag data: - which contains IDs of all expert users, their interests tags, and processed profile descriptions. (around 28,800 records)
- Question data: - which contains IDs of all questions, processed question descriptions, question categories, total number of answers, total number of top quality answers, total number of upvotes. (around 8,100 records)
- Question distribution data: - 290,000 records of question push notification, each contains a question ID, an expert ID and whether or not the expert answered the question. This data was divided into training set(around 245,000 records), validation set (around 30,500 records)and test set(around 30,200 records).

The IDs provided had a hidden mapping of which each ID represented a word/tag/character. These IDs were generated by a random order and this order was not disclosed.The data set is highly unbalanced with 218428 '0' labels implying not answered questions and 27324 '1' labels implying a user has answered a question.

3 Feature Engineering

3.1 One hot vector representation

One hot vector features were generated for each of the unique data fields per expert and per question. These were stored as a sparse numpy matrix. The stored features were in an order maintained list of a mapping of unique expert id to their respective feature vector. A similar order maintained list was created for question id as well.

3.2 Tf-idf

The other kind of feature that was generated was the tf-idf for the word ids, character ids for all the experts and questions. The reason for constructing tf-idf was to assign more significance to the words with rare occurance. The tf-idf for the experts and questions were generated using the entire collection of the one hot vectors features.

3.3 Word Vectors

The data includes the Word ID and Character ID sequences. There has been a large developments generating vectors from symbolic data like words. Word2Vec[10] was used generate word embeddings in this case. The same algorithm can also be used to generate embeddings for the characters. For each user and question description word vectors and character vectors were averaged across the whole description to get two dense vectors (word and character sequence) for users and questions.

4 Methods

4.1 Linear and Logistic regression

We stared with a very basic model just to get the system working. For each question/expert pair in the training data, we constructed a feature vector by combining the question and expert features (by concatenating them). We tried both tf-idf vector for the profile as well a WordVec representation of it. After setting up the input data, we ran both linear and logistics regression on it. We got a score of 0.244 with Linear Regression and 0.245 with Logistic Regression.

4.2 Collaborative Filtering

Collaborative Filtering has been widely used in building recommendation systems. [4]
For this project, we started with a basic Collaborative Filtering system using K-Nearest Neighbors. Collaborative filtering can be either user-based or item-based. We implemented and tried both.

4.2.1 User-based Collaborative Filtering

For user-based collaborative filtering, for each user, we found the K nearest users who had similar profiles. Traditional collaborative filtering methods construct a user-item rating matrix. In our case, we gave a -0.125 rating if the user had refused to answer (or ignored) the question, 0 if we have no information for that user-question combination, and 1 if the user answered the question. (The value of -0.125 was chosen instead of -1 since there were roughly 8 times more 0's in the training data that 1's. Our decision was justified as it gave a better result). For each user, the user profile is just the entire row (of length 8095 - the number of questions) of the user-item matrix. The distance metric we used to find the distance between different users was cosine similarity. Even though most collaborative systems use Pearson Correlation, we went with cosine similarity since it is much faster to compute (due to it satisfying the Triangle Property). Another hyperparameter to tune here was K - the number of neighbors to consider. We did 8-fold cross-validation to tune K and found out that K=180 gave us the best result. Running it on the online validation set with K=180, we got a score of 0.4857. We also tried expanding the user profile to include their descriptions (by appending the tf-idf character and word vectors to their item scores) but it didn't help.

4.2.2 Item-based Collaborative Filtering

Item-based Collaborative filtering was very similar to user-based - constructed a feature vector for each question - a much longer vector of length 28,763 (the number of users) and then computed the K nearest items for each item. 8-fold cross-validation gave us the optimal K as 160, and running it with that K on online validation gave a score of 0.449.

4.3 Content-Based Method

Content based systems solve some of the problems associated with collaborative filtering like the cold start problem. With collaborative filtering, there is no straightforward way to get recommendation for

a user who hasn't rated any items. It also suffers from the sparse data problem.
Content-based approach fixes these issues to some degree. We start off by building a model for each expert. This model is then used to predict whether the expert will answer a question or not. We used Naive Bayes(NB) to model each expert [11]. The features we considered were the TF-IDF feature vectors for the question descriptions (both words and characters); and the number of top quality answers, number of answers and number of upvotes. We also added the question topic as a feature. The number of top quality answers, number of answers and number of upvotes were modeled as Gaussian Naive Bayes while the others were modeled as Multinomial Naive Bayes. The resulting probabilities were multiplied together.
We also experimented with removing some of features and seeing the result in that case. Interestingly, we found that by removing all features and *just* by considering the prior of answering a question for each expert (which is simply the number of questions answered divided by the number of notifications to the expert) we got a score of 0.4868. Adding the question topic features, boosted the score to 0.4900. All the other features were harming the performance so we removed them.

4.4 Content-Boosted Collaborative Filtering

Based on this research by Melville [9], we tried Content-Boosted CF. The basic idea by this method is to remove the sparsity of the user-item matrix in Collaborative Filtering (CF), by substituting unrated items for a user by their content-based score. Thus, for each user and item pair for which we have no information, we replace the 0 value with the score predicted by the content-based method described in the previous section. Once we build the non-sparse user-item matrix, we can then apply the K-Nearest Neighbors method to find the nearest neighbors for each expert. With K=180, we got a score of 0.4718. Thus we see that this method lowered the score instead of increasing it.

4.5 Content-Based Method With K Nearest Neighbors(KNN)

One issue which we noticed with content-based method, despite the good score, was that many of the users had a score of 0, since they had not answered any question, and thus their prior of answering a question was 0. (This is a special case of the cold start problem). To fix this, since we did have information about the questions which the expert had answered, we found the K nearest experts to that expert and averaged out their probabilities given by content-based method. We then adjusted the weights so that half of the weight was for that particular expert's Naive Bayes probability and half was the weighted mean of the neighbors' Naive Bayes probability. We got a score of 0.4911, a slight improvement over the previous best score.

4.6 Neural Networks

In this approach, the main idea was to learn a network which would minimize the cross entropy between the output of the network and the ground truth i.e. whether a user answered a question or not. This is done by having a softmax classifier as the last layer which outputs probability of a user answering a particular question.

Input Features: The input features consisted of user features and question features. User features included information about user tags (in form of a sparse vector), word vectors for the word ID sequence and Character ID sequence were also used. For question features, question tag, word vectors for Word ID and Character ID sequence, number of upvotes, number of answers and number of top quality answers were used. For generating word vectors, Word2Vec was used. We generated 10, 50 and 100 dimensional word vectors and found that 50 and 100 dimensional word vectors did a better job at minimizing the loss function.

Network Architecture: Various architectures were considered. We started off with a simple two-layer architecture to see if the loss was going down over time. But it did not give a very good performance. It is evident from the recent research that wider networks do a better job at approximating function compared to having a large number of neurons in each layer. Hence, more layers were added to investigate the performance of the network. We then tried a 5-layer network (each layer had 512 neurons) and saw a larger drop in the loss, which motivated us to use a 7-layer with 512 neurons in each hidden layer with Dropout[14] and Batch Normalization [5].

Training: All of the training and evaluation was done using TensorFlow. Cross validated with different values of hyper-parameters to find the best value of learning rate was found to be 0.001,

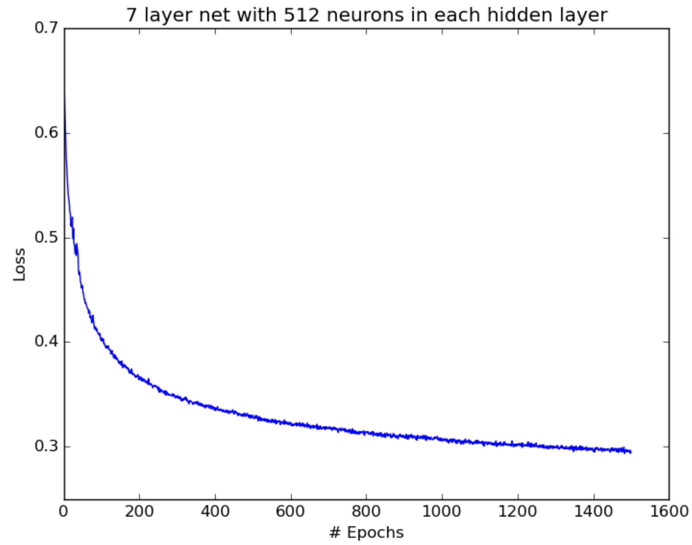

Figure 1: Training Loss vs Number of Epochs

regularization strength was found to be 5e-4. As the data was highly imbalanced (most of the ground truth labels were 0), the network cannot be trained in a traditional way. We employed oversampling. The idea was to have same number of positive and negative instances in each batch (batch size used was 256), so that the gradient flowing back does not just take the label ‘0’ in consideration but also takes ‘1’. This helped the network learn in a much better way compared to just having it trained in a traditional way. We noticed that loss started to plateau after a few epochs of training, therefore we used learning rate decay every 250 epochs with the decay factor being 0.95 and noticed an improvement in the loss. Adam Optimizer was used and the optimization was performed over 1500 epochs. The figure below shows that the network is learning well.

4.7 XGBoost

XGBoost [2] is short for “Extreme Gradient Boosting”, where the term “Gradient Boosting” is proposed in the paper Greedy Function Approximation: A Gradient Boosting Machine, by Friedman. XGBoost is optimized distributed gradient boosting library designed to be highly efficient, flexible and portable. It implements machine learning algorithms under the Gradient Boosting framework. XGBoost provides a parallel tree boosting(also known as GBDT, GBM, Dart Booster) algorithms. We tried a lot of variants of xgboost with various hyperparameters and used the cross validation with the NDCG score to measure which hyperparameters were performing well. It looked like the dart booster with forrest was performing better than gbtree and gblinear. We got a best score of 0.469855888 using dart booster.

4.8 Sparse Linear Method (SLIM)

Ning and Karypis proposed a novel recommendation method called SLIM (Sparse Linear Method) [12]. The key idea behind their method is to retain the sparse nature of the user-item matrix but learn a new sparse aggregation matrix W which captures their values in a more useful way. It reduces to a mathematical equation which they solve using gradient descent.

We implemented their algorithm by using the SLIM library [13] which also incorporates side information. (In our case, those are expert profile or question profile features). We got a score of 0.4526 with this method.

4.9 SVD++

In his paper Factorization Meets the Neighborhood [6], Yehuda Koren extends SVD by combining both neighborhood and latent factor approaches. He calls this method SVD++.
We used LibRec [3] to run SVD++ on our data, (after pre-processing the data to convert it into the required format for LibRec. We got a score of 0.4898 on online validation.

4.10 Matrix Factorization

Matrix Factorization Recommendation Method won the Netflix Prize in 2007 [7]. The method basically involves learning the latent variables by factorizing the User-Item Matrix using either Stochastic Gradient Descent [1] or Alternating Least Squares. One major advantage of this method is that it can even predict scores for users with no prior information (the cold start problem).
We implemented this method using GraphLab [8]. We set the ranking regularization to 0.05, the unobserved rating value to -0.5 and used Stochastic Gradient Descent to factorize the matrix. This gave us a score of 0.5016 which is the best score we got.

5 Evaluation Procedure

We wrote our own script for estimating the performance of the algorithms with the NDCG score. The challenging part was figuring out how to average out the NDCG scores for different questions (we found out through trial and error that we had to take a weighted mean of each question's score). Once we had the script, we could do local validation and thus tune our parameters without the restriction of 3 submissions per day on the contest website.
We performed 8 fold cross validation on many of our algorithms. We chose 8 folds as that gave roughly the same number of question-expert pairs in our local validation set when compared with the validation set given by the contest organizers.
Since 8 fold cross validation would take up a lot of time, we had to parallelize it Cross validation was performed on various algorithms like collaborative filtering, Naive Bayes with KNN, xgboost, svd, svd++. we saw that NB with KNN was giving a lot of 0 as there were many experts who did not have any prior. half of the results had probability 0. We had to fix this so we needed to add some kind of bias to users with unknown information.

6 Analysis

Linear Regression and Logistic Regression seem to do just as well as the score for Random (see table on last page) which means they are not really learning much. **Collaborative filtering** does a decent job but suffers from the issue of cold start leading to a lot of unknown values. We observed that **content-based method** (building a Naive Bayes model for each expert) gave better results than CF. Since both of them are learning different things, we wanted to combine them to get better results. We used a few different Hybrid Methods - Content-Boosted CF didn't help but **Content-based with KNN** pushed our score up. The **Neural Network and XGboost** approach seemed to do a good job at deciding whether a user would answer the question or not, but we believe that it was not able to capture the ordering constraint provided by NDCG algorithm and therefore did not give really good performance. Another problem was that part of feature vector was sparse due to the topic id of users and it becomes harder to learn from a sub-sparse vector as compared to having a dense feature vector. Moving on to more complex methods, it was disappointing that **SLIM** wasn't able to improve on our best score. One major reason could be that SLIM is meant for getting top-N recommendations for a particular expert. Since in our case, we were given question/expert pairs for which we wanted probabilities, it was very likely that a particular question for which we wanted the predicted probability will not occur in that users top-N recommendations (even if you set the value of N to be high). Giving a value of 0 in such cases seems disingenuous but there are no obvious alternatives. **SVD++** and **Matrix Factorization** perform very well, the latter especially so. This wasn't surprising since they have been proven to work well in the Netflix Challenge, which shares a lot of similarities without contest. Matrix Factorization had a bigger impact mainly because we used a ranking variant of it which optimizes for rank, thus learning exactly what we needed (with regards to getting a better NDCG score). It also takes care of users with little or no information and gives the probabilities accordingly.

7 Conclusions

To conclude, the project was a great learning experience. We went through highs and lows as we tried method after method. It seemed like an accurate representation of how frustrating but rewarding research can be. One could spend hours implementing a method just to get a worse result. There was no 'known' way to follow and that made the final result even sweeter (We got 5th position among the class!). Another interesting takeaway was that often simpler is better. We've learned this in theory of how simpler models should be preferred but we observed this first-hand as our very simple model of just using priors in Naive Bayes outperformed many of our complex methods such as CF and Neural Networks.

References

[1] L. Bottou. Stochastic gradient tricks. *Neural Networks, Tricks of the Trade, Reloaded*, pages 430–445, 2012.

[2] J. H. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[3] G. Guo, J. Zhang, Z. Sun, and N. Yorke-Smith. Librec: A java library for recommender systems. In *Posters, Demos, Late-breaking Results and Workshop Proceedings of the 23rd International Conference on User Modeling, Adaptation and Personalization*, 2015.

[4] J. L. Herlocker, J. A. Konstan, A. Borchers, and J. Riedl. An algorithmic framework for performing collaborative filtering. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 230–237. ACM, 1999.

[5] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

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

[7] Y. Koren, R. Bell, C. Volinsky, et al. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[8] Y. Low, J. E. Gonzalez, A. Kyrola, D. Bickson, C. E. Guestrin, and J. Hellerstein. Graphlab: A new framework for parallel machine learning. *arXiv preprint arXiv:1408.2041*, 2014.

[9] P. Melville, R. J. Mooney, and R. Nagarajan. Content-boosted collaborative filtering for improved recommendations. In *Aaai/iaai*, pages 187–192, 2002.

[10] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[11] R. J. Mooney and L. Roy. Content-based book recommending using learning for text categorization. In *Proceedings of the fifth ACM conference on Digital libraries*, pages 195–204. ACM, 2000.

[12] X. Ning and G. Karypis. Slim: Sparse linear methods for top-n recommender systems. In *2011 IEEE 11th International Conference on Data Mining*, pages 497–506. IEEE, 2011.

[13] X. Ning and G. Karypis. Sparse linear methods with side information for top-n recommendations. In *Proceedings of the sixth ACM conference on Recommender systems*, pages 155–162. ACM, 2012.

[14] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Algorithm	Brief Description	Score
Random Scores	Gave a uniform random probability to each question/expert pair	0.226793796
Linear Regression	Applied Linear regression after concatenating feature vector for question and expert	0.244279653
Logistic Regression	Applied Logistic regression after concatenating feature vector for question and expert	0.244916514
User Collaborative Filtering (KNN)	Found K (180) Nearest neighbors for each expert who have answered the question, and took weighted mean of this cosine distance from the expert	0.485752276
Item Collaborative Filtering (KNN)	Found K (160) Nearest neighbors for each question answered by the expert, and took weighted mean of this cosine distance from the question	0.449131331
Content-Based Naïve Bayes	Built a Naïve Bayes model for each expert based on the question tags of the questions he has answered	0.490014322
Content-Boosted Collaborative Filtering	Replaced 0 values in user-item with scores from Content-Based Naïve Bayes and then applied User CF KNN	0.471756232
Content-Based Naïve Bayes with KNN	Combined the expert's Naïve Bayes probability with the NB probabilities of its K Nearest Neighbors	0.491114802
Neural Network	Built a 7 layer Neural Network with 512 neurons in each hidden layer.	0.400662199
XGBoost - DART booster	Dart Booster with sample_type:-weighted;norm_type:-forest;rate_drop:-0.2;skip drop:-0.9;	0.469855888
Sparse Linear Method (SLIM)	Learn a sparse aggregation coefficient matrix by solving an optimization problem	0.452624144
SVD++	Used SVD++ implementation for recommendation systems from librec framework.	0.489817194
Matrix Factorization	Used matrix factorization to map questions and experts to a hidden latent space, and then estimating new ratings	0.501604997

Algorithm	Parameters	fold 0	fold3	fold7	Average	online_validation
svd++	factors=50, iters = 200	0.482992348	0.464020421	0.443213788	0.463408852	0.489817193
naïve_bayes_withcollab		0.464577477	0.458393165	0.461891952	0.461620865	0.4911
nmf	factors=100 iters=10	0.476823961	0.462295355	0.442324729	0.460481348	
nmf	factors=200 iters=20	0.479006518	0.45216265	0.429280575	0.453483247	
svd++	factors=75 iters=300	0.485779045	0.463038776	0.44415035	0.464322724	
pmf	factors=10 iters=75 learning_rate=0.005 reg=0.05	0.458853735	0.451460432	0.422533365	0.444282511	
pmf	factors=30 iters=200 rest:same	0.46767265	0.452197398	0.42826294	0.449377663	
svd++	factors=100 iters=200	0.484656582	0.466464969	0.446362391	0.465827981	
userknn	dis=pcc neigbors=150	0.411615351	0.397005679	0.386258285	0.398293105	
svd++	factors=150 iters=200 reg=0.05	0.487180737	0.467141443	0.444469155	0.466263778	0.487724059
userknn	dis=pcc neighbors=50	0.411389785	0.39677585	0.386258285	0.398141307	
trustsvd	factors=10 iters=100	0.470239781	0.461208697	0.43878979	0.456746089	
bpmf	factors=9 iterations=100	0.431694838	0.420296618	0.391610108	0.414533855	
bpmf	factors=50 iterations=200	0.401885147	0.380660403	0.356969181	0.379838244	