CSCI 567 Final Report

David Becerra
Department of Computer Science
University of Southern California
Los Angeles, CA 90007
dbecerra@usc.edu

Elizabeth Boroson
Department of Computer Science
University of Southern California
Los Angeles, CA 90007
boroson@usc.edu

Sourya Dey
Department of Electrical Engineering
University of Southern California
Los Angeles, CA 90007
souryade@usc.edu

Abstract

We developed recommender systems to predict the likelihood of a user answering a question. We used several techniques, including linear regression, collaborative filtering, and neural networks. Overall, we achieved the best results on the validation data using collaborative filtering. We implemented several collaborative filters, including memory-based, model-based, and hybrid. We found that the best predictions were mostly based on users' history; users who had previously answered the same questions were likely to continue answering the same questions.

1 Introduction

This project was based on a website in which users could ask and answer questions. A user's decision to answer a question could be influenced by several factors such as the question's topic or popularity. The goal of this project was the development of a recommender system to predict the likelihood for a user to answer a question.

Our data contained information about users and questions. For users, we had unique IDs, tags for topics of interest, and the words and characters in their profiles. Each word or character was assigned an ID, so we could compare users with similar profiles, without reading the content. For the questions, we also had unique IDs, category tags (not necessarily the same as the user category tags), and the words and characters in the question. Additionally, we had information about the popularity of the question: the numbers of upvotes, answers, and answers ranked *top quality*.

The training data was a list of user-question pairs. Each pair consisted of a question and a user who had looked at it. The pair had an associated label: 1 if the user had answered the question and 0 if the user had not. The validation and test data were also user-question pairs, without labels.

2 Data Preprocessing

2.1 Features

For the various machine learning techniques we implemented, we considered a number of features outlined in Table 1. The max n value for features (g) and (h) represent the maximum number of characters or words for user-question pair (u, i) that we observed in the data set.

Table 1: A list of the features we considered. The features assume a user-question pair (u, i).

Feature	Description
(a) number of upvotes	total number of upvotes for question i
(b) number of answers	total number of answers for question i
(c) number of top quality answers	total number of top quality answers for question i
(d) percentage of top quality answers	ratio of top quality answers to total answers for i
(e) number of common words	number of common words between u and i
(f) number of common characters	number of common characters between u and i
(g) has $n \in \{0, \dots, 4\}$ words in common	1 if (u, i) has n common words
(h) has $n \in \{0, \dots, 9\}$ common characters	1 if (u, i) has n characters in common
(i) has n-th $\in \{1, \dots, 5\}$ most popular tag	1 if u has the n^{th} most popular tag to answer i

The popularity measures (i.e. features (a)–(c)) have a wide range of possible values. For our neural network implementation, we considered grouping questions into roughly equal sub-ranges. For example, the number of top quality answers can vary from 0 to more than a hundred, but is less than 10 for most questions. Thus, we grouped questions into the following sub-ranges: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11-12, 13-14, 15-16, 17-18, 19-20, 21-25, 26-30, 31-40, 41-50, 51-100, >100].

In addition to feature extraction, we also implemented polynomial feature expansion. More specifically, given a feature vector $\mathbf{x} = \{x_1, \dots, x_n\} \in \mathbb{R}^n$, we transformed $\mathbf{x}$ into $\mathbf{x}'$ as follows:

$$\mathbf{x}' = \mathbf{x} \cup \{x_i \cdot x_j | x_i, x_j \in \mathbf{x}\}$$

2.2 Feature Analysis

The statistics on our features show some important discriminating factors. *Users are more likely to answer popular questions*. For example, 2% of the questions that were ignored had 0 upvotes, while only 0.32% of answered questions had 0 upvotes. Similarly, 2.6% of ignored questions had no top quality answers, while only 0.5% of answered questions had no top quality answers

2.3 Analysis of Tags

Question and user tags represent specific topics such as fitness and parenting. We made an initial assumption that a user is interested in answering questions with a matching tag. Therefore, we aggregated the number of times a tag x was shared between user-question pairs, provided the user answered the question. Since the given data contains 20 unique question tags and 143 unique user tags, x varies from 0-19. The results are as follows: [1, 10, 321, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 3, 1, 0, 0, 0, 0, 0]. From this we observe that tag 2 is an overwhelmingly popular common tag between users and answered questions. Nevertheless, in general, users do *not* tend to favor questions with matching tags.

3 Methods

3.1 Naive Tag

The first method we implemented was a naive discriminative approach. For user-question (u, i), we modeled the probability u would answer i given (u, i) either had or did not have a tag in common. We computed these probabilities as follows:

$$P(y_{u,i} = 1 | (u, i) \text{ have common tag}) = \frac{\sum_{u,i} y_{u,i} \mathbb{I}\big[(u, i) \text{ have common tag}\big]}{\sum_{u,i} \mathbb{I}\big[(u, i) \text{ have common tag}\big]}$$

$$P(y_{u,i} = 1 | (u, i) \text{ do not have common tag}) = \frac{\sum_{u,i} y_{u,i} \mathbb{I}\big[(u, i) \text{ don't have common tag}\big]}{\sum_{u,i} \mathbb{I}\big[(u, i) \text{ don't have common tag}\big]}$$

where $y_{u,i} \in \{0, 1\}$ is the label of (u, i) and $\mathbb{I}(\cdot)$ is the indicator function. Note that we only counted (u, i) in the training data set rather than all possible (u, i) pairings.

3.2 Linear Regression

Following our naive implementation, we considered a linear regression model. Define our training data as $\mathcal{D} = \{(\mathbf{x}_{u,i}, \mathbf{y}_{u,i})_{(u,i)}\}$ where $\mathbf{x}_{u,i}$ is a feature vector for user-question pair (u, i) as defined in Table 1 and $\mathbf{y}_{u,i} \in \{0, 1\}$ is the true label for whether u answered i. Given $\mathcal{D}$ as input, we trained a linear regression model to produce a predicted label $\hat{y}_{u,i}$.

3.3 Collaborative Filtering

Collaborative filtering (CF) techniques are an approach to building recommender systems based on the idea that users who have rated items similarly in the past will do the same in the future.

3.3.1 Memory-Based

Memory-based CF techniques use only rating histories to predict future ratings. The history is usually the correlation between two users or two items. For example, if two users gave the same item the same rating, then they are likely to also give other items the same rating.

We implemented a memory-based CF based on [6]. For each user-question pair, the similarity of the user with each other user was computed. The similarity was based on the **rating** that the user had given each question: +1 if they answered the question and -1 otherwise. Note that a user who had seen a question would have given it a rating even if they did not answer it. Two similarity measures were considered. The first was the vector cosine similarity:

$$w_{u,v} = \cos(u, v) = \frac{u \cdot v}{\|u\| \times \|v\|}$$

where u and v are the users' ratings on questions they both rated. The second was Pearson correlation:

$$w_{u,v} = \frac{\sum_i (r_{u,i} - \bar{r}_u)(r_{v,i} - \bar{r}_v)}{\sqrt{\sum_i (r_{u,i} - \bar{r}_u)^2} \sqrt{\sum_i (r_{v,i} - \bar{r}_v)^2}}$$

where $r_{u,i}$ is the rating of user u on question i and $\bar{r}_u$ is the average rating of user u on all questions.

We computed the predicted rating for a user-question pair (u, i) as the weighted sum of ratings for i across the neighboring users of u, $N(u)$, weighted by the similarity between u and its neighbors:

$$P_{u,i} = \bar{r}_u + \frac{\sum_{v \in N(u)} (r_{v,i} - \bar{r}_v) \times w_{u,v}}{\sum_{v \in N(u)} |w_{u,v}|}$$

We implemented this formula with both methods of computing similarity and also by considering different $N(u)$ (i.e. all users or only nearby users).

3.4 Model-Based

Model-based collaborative filters construct a probability model based on the data to make rating predictions [1]. The probability model is created using techniques such as Naive Bayes classifiers, Bayesian Belief nets, and Markov Decision Models [6]. Clustering is another popular model-based approach. Sarwar et al. used a variant of k-Means clustering to partition users of a CF system. The clusters were then treated as neighborhoods in a memory-based approach [5].

We implemented this algorithm using standard k-Means clustering. Given a data set of users A, we cluster A into k partitions $A_1, \ldots, A_k$ such that $A_i \cap A_j = \emptyset$ for $i \neq j$ and $\cup_i A_i = A$. Then, we define the neighbors $N(u)$ of a user u as follows:

$$N(u) = \{u' : u' \in A_i\} \quad \text{where } u \in A_i$$

Using this definition of neighbors, we proceed with the memory-based CF outlined previously.

3.4.1 Hybrid

Another option for CF is to use the content or features of the users and questions in addition to the rating history. Combining the content and the history results in a *hybrid filter*.

We implemented a hybrid CF as described in [4]. In this filter, *pseudoratings* are computed for every user-question pair from their features. These are predictions of the users' ratings. We tried two methods of computing pseudoratings. In both methods, for each question, we trained a classifier with users who rated the question as training data and predicted the ratings for all other users. In the first method, we used a Multinomial Naive Bayes (NB) classifier with word frequency vectors of user descriptions as feature data. In the second method, the classifier was a kernel support vector machine (SVM) using the RBF kernel, with the entire feature vector from Table 1.

Next, the weighted predictions for each pair are computed in the same way as the memory-based CF. However, the prediction considers a neighborhood composed of all users, using true ratings where available and pseudoratings elsewhere. The weights are also adjusted based on the number of true ratings available, so users with few true ratings have lower weight than users with more true ratings.

3.5 Neural Networks

Neural networks are a powerful machine learning technique, versatile enough to approximate any function [3]. For each user-question pair in the training data, we constructed a binary feature vector whose entries corresponded to binary input neurons forming the network's input layer. Our output layer consisted of 2 neurons, corresponding to 'Ignore' and 'Answer', with softmax activation functions. To train the network, we fed the training data in batches, gradually improving performance over several epochs. Once trained, the network is made to evaluate the validation and test data.

Initially, we tried a small neural network of roughly 20 input neurons corresponding to features which discriminated the training data well based on label. However, to improve our results, we increased the number of input layer neurons to 530 by considering all the features mentioned in Section 2, not simply the discriminatory ones. The final feature set consisted of the following:

- Popularity metrics: one input neuron for each sub-range for each metric (see section 2.1).
- Tags: one input neuron for each question and user tag. One each for the number of user tags, which corresponded to questions such as 'Does the user have a total of x tags?' We also had one input neuron for each possible tag in common between question and user.
- Words and Characters: one input neuron for each word and character ID. Similarly, one for every possible common word and character.

The training data was divided into two sets, a training set (75%) and a validation set (25%) for testing results after every epoch. This was done to implement Early Stopping when applicable.

4 Results

4.1 Naive Tag

The Naive tag approach performed the worst out of all our methods with a score of 0.2217 on the validation data set. This is expected as the naive tag approach misses important features of the data.

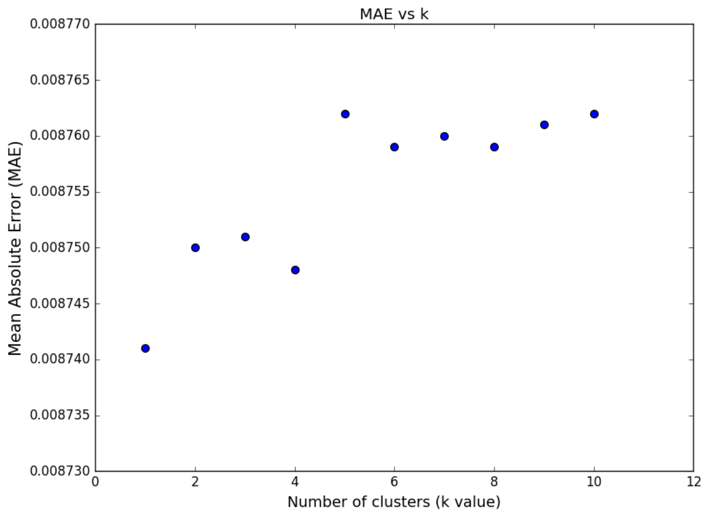
4.2 Linear Regression

For our linear regression implementation, we considered all the features outlined in Table 1 and utilized polynomial expansion. The performance on the validation data was 0.2327 because linear regression assumes our data is linearly separable which is unlikely, resulting in a low performance.

4.3 Collaborative Filtering

CF gave us the best results compared to all other methods we attempted. We first implemented the memory-based CF described in Section 3.3.1; however, we only evaluated Pearson correlation as a similarity measure. We tried different methods of computing the mean rating for each user. We

Figure 1: Results of cross-validation on k-Means model-based CF algorithm.



computed it based on all questions the user answered, only shared questions, and ignoring the mean ratings altogether. Overall, the best result used mean ratings over all questions, with a score of 0.4798.

We implemented the k-means CF algorithm using both the Pearson and cosine similarity measure with an initial k value of 8. Cosine similarity on the validation set scored 0.4848, slightly better than the score using Pearson similarity (0.4778), and better than our memory-based CF implementation.

We performed 15-fold cross-validation (CV) using the training data to select the best $k \in \{1, \ldots, 10\}$. We used the Mean Absolute Error (MAE) [2, 5] to evaluate each CV iteration. MAE is defined as

$$MAE = \frac{\sum_{u,i} |p_{u,i} - r_{u,i}|}{n}$$

where $p_{u,i}$ is u's predicted rating of question i and n is the total number of (u, i) pairs. The best k value was the one that minimized its MAE.

Our cross-validation results, outlined in Figure 1, suggest that $k = 1$ is the optimal k value. However, $k = 1$ is equivalent to the memory-based approach from earlier which performed worse than $k = 8$. Therefore, we implemented the second best k value of 4. With cosine similarity, this led to our best performance across all our attempts with a validation data set score of 0.4883.

For our hybrid CF implementation described in Section 3.4.1, we computed pseudoratings with multinomial NB on word frequency vectors of the user descriptions and SVM on all features. We used k-Means clustering with $k = 8$ to define the user neighborhoods. This was done for efficiency, as even a multithreaded implementation would have taken several days to complete the validation set using pseudoratings for every user.

The hybrid CF with pseudoratings from NB had a score of 0.2541, indicating that user descriptions alone do not contain information about ratings. However, the hybrid CF with pseudoratings from SVM had a score of 0.460475. While this did not improve on the memory-based CF, it indicates that SVM pseudoratings were consistent with the available true ratings.

4.4 Neural Networks

Our initial network with only discriminatory features gave a score of 0.22212. Increasing the number of input and hidden neurons led to better results; however, increasing the number of hidden layers alone showed a decline in results. We acheived a maximum score of 0.3361 for neural networks using the following network characteristics:

- Glorot Normal Weight Initialization; L2 Weight Regularization with parameter = 5×10^{-6}

Table 2: A list of our implementations and the commands to run them

Implementation	Type this in the Command Line
Naive tag	`$ python naive_tag.py`
Linear regression	`$ python linear_regression.py`
Memory-based CF (with Pearson similarity)	`$ python collaborative_filter.py`
k-means model-based CF	`$ python -k {K-value} kmeans_cf.py`
Hybrid-model CF	`$ python hybrid_cf.py`
Neural network (step 1)	`$ python feature_analysis.py`
Neural network (step 2)	`$ python neural_network.py`

- 1 hidden layer with 200 neurons and rectified Linear Unit (ReLU) activation function
- Categorical cross-entropy loss function
- Stochastic Gradient Descent optimization with Learning rate = 10^{-3}, Decay factor = 5×10^{-5} and Nesterov momentum applied (momentum coefficient = 0.99)
- 250 epochs of training, minibatch size = 139

5 Software

All the files inside 'MLpokemon.zip' should be extracted to the same location in order to run the code successfully. You will need Python 2.7, *numpy*, *scipy*, *argparse* and *keras*. Note that the file 'neural_network.py' borrows code from 'hw_utils.py' provided to us in Homework 4. For the neural network code, the files 'temp.csv' and 'final.csv' will be created, corresponding to the predicted results on the validation and test data, respectively. See Table 2 for directions on running the code.

6 Conclusion

In this project, we developed recommender systems to predict the likelihood of a specific user answering a given question. We used several techniques, including linear regression, collaborative filtering, and neural networks. Overall, we achieved the best results on the validation data using collaborative filtering techniques. We implemented several collaborative filters, including memory-based, model-based, and hybrid. The best predictions were mostly based on users' history, so users who had previously answered the same questions were likely to continue answering the same questions. Some of the features we computed provided some information about whether a user was likely to answer a question, as seen in the neural network and SVM results. However, our best results were from a model-based collaborative filter combining user histories and clustering.

References

[1] Breese, J.S., Heckerman, D. & Kadie, C. "Empirical analysis of predictive algorithms for collaborative filtering." *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence* (1998): 43-52.

[2] Ekstrand, M.D., Riedl, J.T. & Konstan, J.A. "Collaborative filtering recommender systems." *Foundations and Trends in Human-Computer Interaction* 4.2 (2011): 81-173.

[3] Hornik, K., Stinchcombe, M. & White, H. "Multilayer feedforward networks are universal approximators." *Neural Networks* 2.5 (1989): 359-366

[4] Melville, P., Mooney, R.J. & Nagarajan, R. "Content-Boosted Collaborative Filtering for Improved Recommendations." *Proceedings of the Eighteenth AAAI Conference on Artificial Intelligence* (2002): 187-192.

[5] Sarwar, B.M., et al. "Recommender systems for large-scale e-commerce: Scalable neighborhood formation using clustering." *Proceedings of the fifth international conference on computer and information technology* 1 (2002).

[6] Su, X. & Khoshgoftaar, T.M. "A Survey of Collaborative Filtering Techniques." *Advances in Artificial Intelligence* (2009).

Table 3: Algorithms and results

Algorithm	Description	Score
Naive tag	A naive probabilistic model based on number of users who answered questions with or without a common tag.	0.22173
Linear regression	Linear regression model using the features in Table 1 and polynomial expansion.	0.23268
Memory-based CF	CF comparing past ratings/labels of users to predict future ratings	
	Pearson Similarity, mean ratings from shared questions	0.46869
	Pearson Similarity, mean ratings from all questions	0.47976
	Pearson Similarity, no mean ratings	0.43579
k-means model-based CF	Memory-based CF but using k-means clustering to define the neighbors of a user	
	k=8, Cosine Similarity	0.48480
	k=8, Pearson Similarity	0.47781
	k=4, Cosine Similarity	**0.48829**
Hybrid-model CF	CF using pseudoratings when true ratings unavailable	
	Pseudoratings from multinomial NB	0.25411
	Pseudoratings from SVM with RBF kernel	0.46048
	Pseudoratings from SVM, weighted by number of true ratings	0.43107
Neural network [input - hidden - output]	Neural Network with input neurons for each feature described in Section 2, softmax output layer, and ReLU hidden layers.	
	Small network [20-10-2]	0.22212
	Bigger network [60-20-2]	0.23641
	Biggest network [530-200-200-2]	0.31768
	Final optimized network [530-200-2] trained for 100 epochs	0.32607
	Final optimized network trained for 250 epochs	0.33613
	Network with cluster data incorporated	0.32075
	Network with pseudorating data incorporated	0.25879