Final Project Writeup

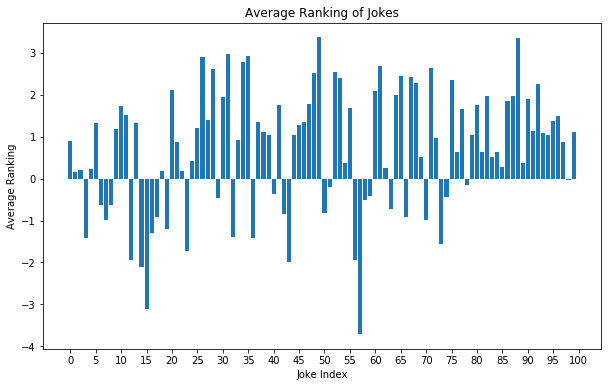
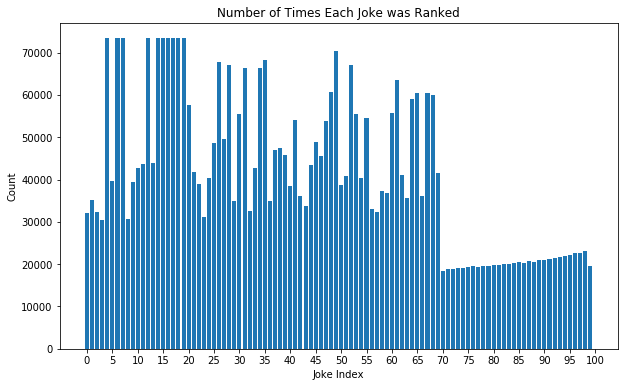
Adam Ten Hoeve

CSCI 4022

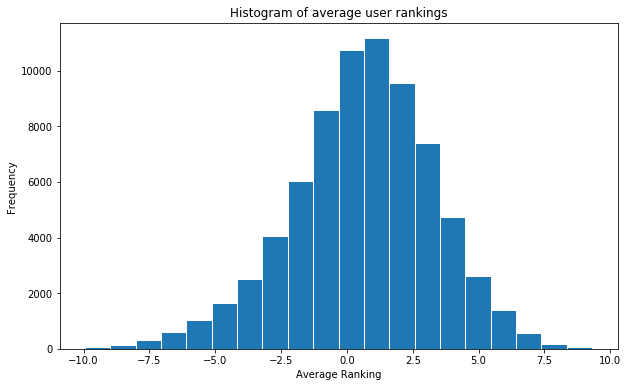
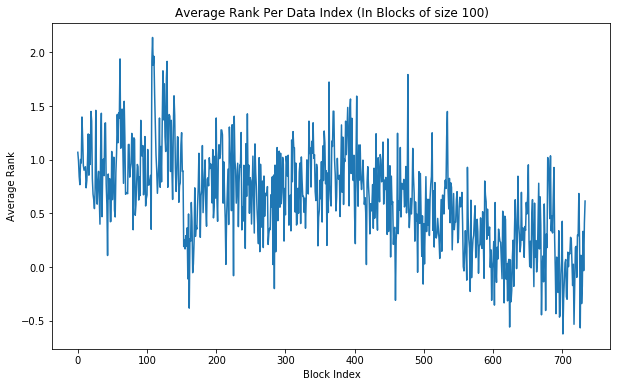
Have you ever wanted to tell better jokes? To be able to seamlessly weave the perfect joke into a conversation, resulting in the whole room devolving into laughter? Then do I have the project for you. I had a similar idea and decided to figure out which jokes are the best. And I solved that problem by throwing math at it. In this project, I built several different recommender systems for jokes to determine which jokes people will like the most.

The dataset came from a lab in University of California, Berkeley which showed jokes to different users and asked them for rankings. These rankings have been recorded since April 1999, so for this project I concatenated the three datasets from April 1999 – May 2003. The result was a dataset with 73,421 rows (each for an individual user) and 101 columns (the first being the number of jokes the user ranked, and the rest being the rank of each joke). The ranks we on a scale from [-10, 10], with -10 being the lowest and 10 being the highest. Any joke that the user didn’t rank was given a score of 99. The datasets can be found at: <http://eigentaste.berkeley.edu/dataset/>.

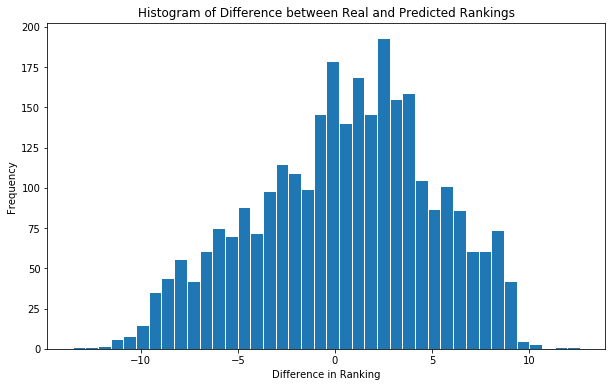
In the preliminary steps, I did some data cleaning and exploration. To clean the data, I converted all the 99 rankings to NaN values and removed the first column, which wouldn’t serve any purpose for the recommender systems. I then looked at some basic trends of the data, such as the average rankings of each joke and how many times each joke was ranked.

From these plots, we can see that there is a wide spread of joke rankings. We can also see that jokes {5, 7, 8, 13, 15, 16, 17, 18, 19, 20} have been ranked for most users, so they can be considered dense. This means if we had a new user, we should show them those jokes first as those will have the greatest number of comparisons (for the collaborative system) to get the most accurate results. Some other exploratory data we can do is to look at the average user rankings.

The left plot shows the frequency of average user rankings. We can see that the plot has a mean at about 1.5, meaning that most users were ranking slightly more positively than negatively. The right plot shows average user rankings per index (in groups of 100 for visualization). We can see that there is a general downtrend in the rankings of the data. Why that is, I don’t know, possibly some way the researchers sorted the results before posting the datasets. What matters is that we notice this and realize that if we train or evaluate only one “section” of the data, then there will be some systematic error in the results. This was a significant problem when I was building the models, as I was using a fixed holdout set for my method of evaluation, which skewed the results, as you can see in plot below.



I’m skipping ahead to the evaluation, but this plot shows the difference between the actual and predicted rankings. We can see that there is some systematic bias, making most rankings be about 2-3 less points than the actual. To reduce this error, we can build an evaluation set from a random selection of users. In practice, this did not correct the issue, but it did improve the results.

We’ve taken a through look through the data and are ready to move onto the recommender systems. There are three main types of systems: Collaborative, Content Based and a Hybrid of the two. For this project, I wasn’t sure which would work the best, so I built all three. I’ll go into each system in more detail in the following sections.

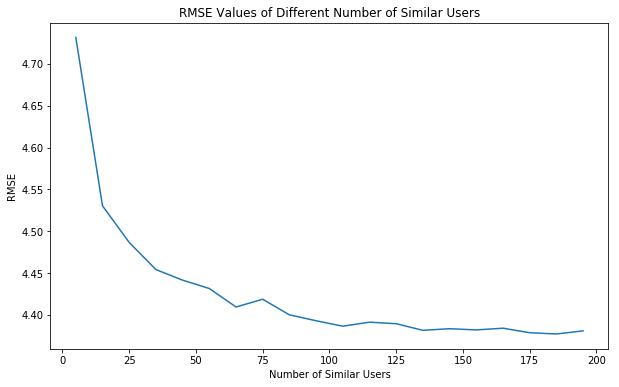
**Collaborative System**

The collaborative system is designed around the idea that we can use similar users to predict scores for items. If user A liked movies X, Y and Z and user B liked movies X and Y, then it’s likely that user B will also like movie Z. But our dataset has about 75,000 users, so if we have new user , then our task becomes finding the most similar users and using them to predict the most likely jokes for . We can calculate the similarity between two users by calculating the cosine similarity scores between their vectors of ranks:

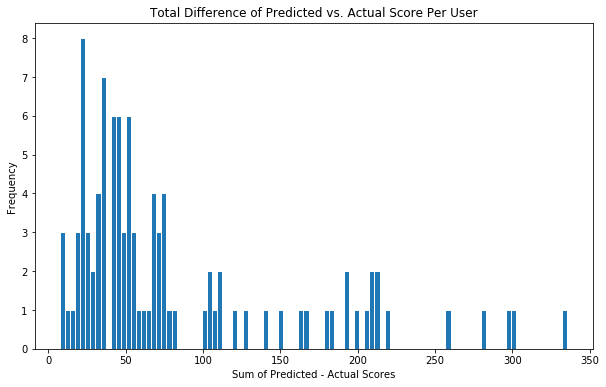
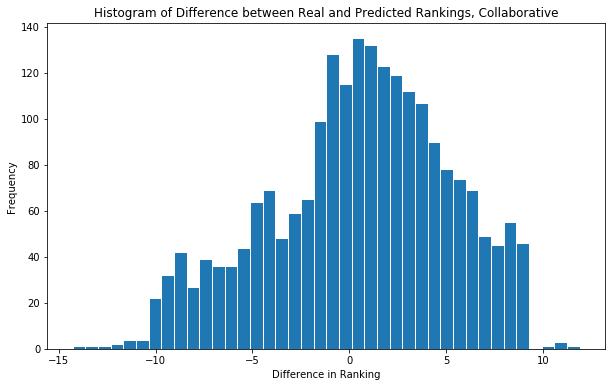
I will mention that I tried centered cosine similarity, but it produced results with higher error than results produced with cosine similarity. I’m not sure why that is, but I used cosine similarity in the final model. Now that we have our similarity scores for all 75,000 users, we find the users who have the highest similarity to user by taking the argmax of the similarity scores. Using those users, we can predict the rankings for each joke by taking an average of each similar user’s score, weighted by their similarity score. Formally, the ranking for user , for each joke , with the set of similar neighbors :

To evaluate our model, we will use a technique called a holdout set. This involves taking a subset of the rows of our data, “removing” about half of the rankings by setting equal to NaN and then using our model to predict new values for those rankings. We can compare our predicted rankings to the actual rankings to determine how well our model performs. The smaller the difference between the two, the better it is doing. One way of measuring this difference is with a Root Mean Squared Error:

But how do we choose the number of similar users, ? We can use hyper-parameter techniques, such as creating an elbow plot, to see where a larger set size stops improving the model.



From the above plot, we can see that we stop getting significant improvement when . We will use this value in our final model to get our best result. Now that we have all the pieces to build and evaluate our system, let’s see how well it does.



The two plots above show the results of our collaborative system. The left plot shows the frequency of differences between the actual rankings and the predicted rankings (residuals). We can see that the model performs… alright. There is still some systematic skew in the data, as the model is regularly predicting about 1-2 points less than the actual rankings. There are also some heavy tails on the model, meaning some predictions where very incorrect. The plot on the left shows the sum of each user’s residuals. From this, we can see that most users had a relatively small “total” error, having an average residual of about 0.5 per question, but some users had many severe, incorrect classifications, with average residuals of about 3 per question. This could be due to not having enough “similar” users in the dataset, or that their similar users voted differently in the holdout set than was expected. The RMSE score for this system with a holdout set of 100 users was about 219.73. We will use this to compare to the other systems later on.

In summary, the model isn’t perfect. But we don’t have anything to compare it yet, so let’s make the content-based system as well.

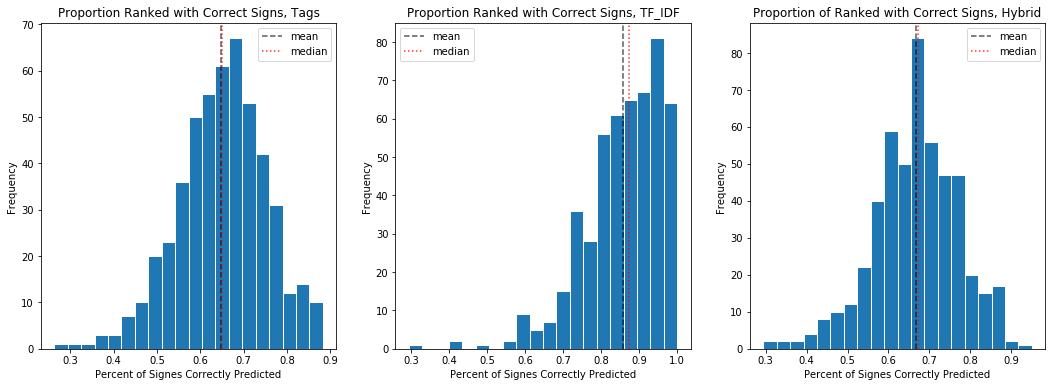
**Content-Based System**

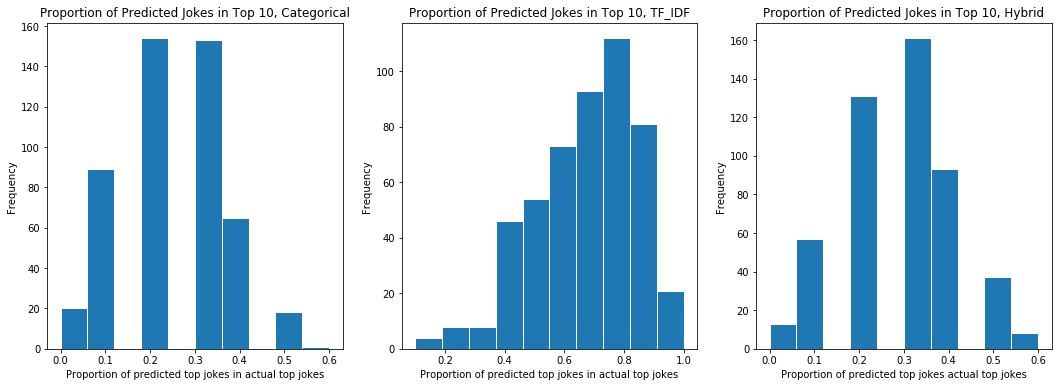
A content-based system uses the attributes of the jokes themselves to make recommendations. If a user ranks jokes about politics highly, then they will be given other jokes about politics. Before we dive into the math involved, we need to look at our data again. No “information” was provided about the jokes, just the text itself. To extract the “information”, we need to do some extra work, but we don’t know which “parts” of the text to extract. Instead, I decided to pursue three possible content-based systems, test them, and determine which worked the best. The first involved reading all 100 bad jokes and creating my own dataset of “tags” for each joke. These “tags” were categories that the joke fell into, including things like politics, religion, “screw in a lightbulb,” and so on. The second system used Term Frequency-Inverse Document Frequency (TF-IDF) which used the text strings as the features. The third system was a hybrid of the two, meaning it used both “tags” and TF-IDF scores as features.

TF\_IDF is a method of picking the most “important” words from a document by assigning each word a score. The more often the word appears in that document, and the fewer number of documents that word appears in, the higher the score. The formal equation is for word in document is:

Where is the number of times appears in , is the number of documents, and is the number of documents that contain string . This gives each word a score which we will use as features when training our system.

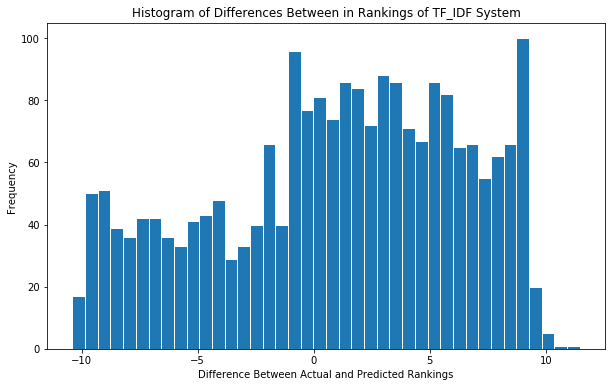
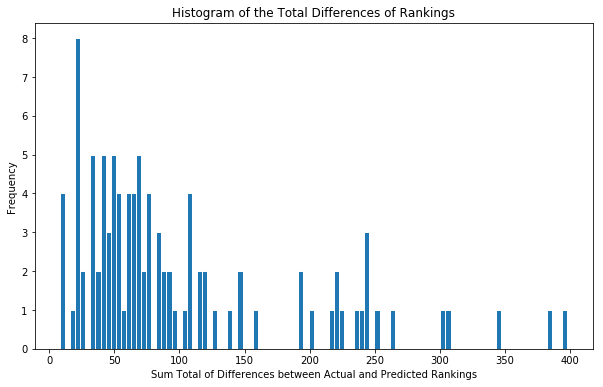
I read through all the jokes, created tags, and created the TF\_IDF system. We have our item profiles, now we can build the recommender systems. The content models all follow the same steps, of multiplying each row of the item profiles matrix by the user’s rank for that joke, then taking the average of each column to get a vector of the user’s feature scores. This is the user’s profile. We can then dot-product the user’s profile with any joke’s item profile to get the user’s predicted score for that joke. Note that these scores won’t be on the same scale as our original rankings, they’re a relative score. Because of that, finding the residuals of a holdout set isn’t a perfect process, as the scales will be off. Instead, we can compare how well the systems predict in the same direction as the user, and if they correctly predict the user’s top jokes.





The top plots show the frequency of proportions of time that the sign of the predicted rankings was the same as the sign of the actual rankings. This shows us a basic version of how well the model is performing. We can see that the system using the tag data and the hybrid system performed about the same, and worse than the TF\_IDF system. The bottom plots show the proportion of jokes predicted to be in the top 10 that were actually in the user’s top 10. Again, the TF\_IDF system worked much better than the other two systems. So from here, I decided the TF\_IDF system performed the best and would be my “final” content-based system.

With the TF\_IDF system, I decided to use the holdout method for evaluation. Even though I said earlier that it wouldn’t work as well (which is still true), I decided it would be a reasonable comparison to the collaborative model. I scaled the relative ranks to the [-10, 10] scale and computed the residuals.

These plots are the same as the evaluation plots for the collaborative model. In comparison, the content-model had a much greater spread of residuals than the collaborative model, meaning there was a greater difference between the actual and predicted rankings. It also had larger outliers for the user’s sum of residuals, reaching up to 400, meaning that the outliers were still poorly represented. We can also note that the systematic error is still present, if not worse. Some of this general error is likely due to the scaling of the relative scores, but that can’t explain all of it. The RMSE score for the model, with a holdout set of 100 users, was about 254.46. This provides a similar conclusion to the graphs, that the content-based method performed worse than the collaborative system (collaborative had RMSE = 219.73).

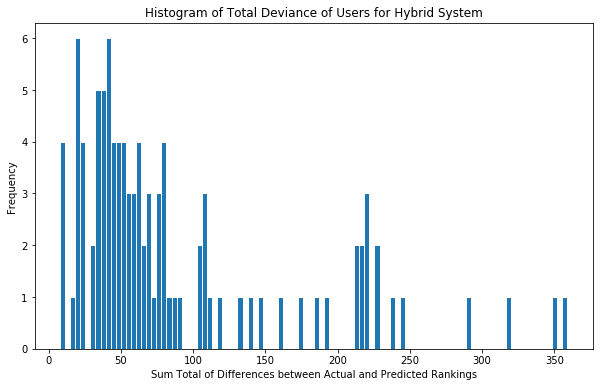
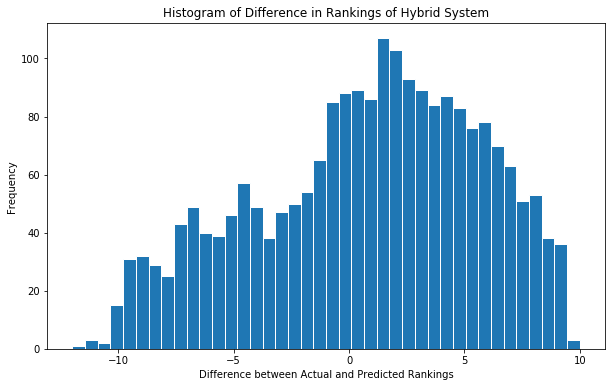
We now have two systems that, somewhat effectively, predict rankings for users. What kind of data scientists would we be if we didn’t try to staple them together?

**Hybrid System**

The hybrid system calculates the scores from the content-based and collaborative models, then aggregates them together into a (hopefully) more accurate result. The idea behind this is that it can pull from multiple sides of the data, both the attributes of the jokes and the communities around them, to create a more informed prediction.

The aggregation method I decided to go with was to take the average of the collaborative and scaled content-based predictions. I tried a fewer other methods, like multiplicative methods and averages with different scaling sizes, but the method that performed the best was:

This seems like a pretty stupid equation, and I would agree, but it produced the hybrid model with the least error. The results of that system are as follows.



Again, same plots as the other two systems. The long tails of the collaborative system have been shaved down a bit, most users have about an 0.5 average residual per joke, and the outliers are similar to the collaborative system. But there is also not a significant improvement over the other two models, at least visually, particularly with respect to the collaborative model. The RMSE score for the hybrid system was 230.44, which is also worse than the collaborative system.

**Conclusion**

Did the systems work? Yes. Did they work well. Kind of. From personal experience of putting my own rankings in, the collaborative and hybrid systems did recommend jokes that I enjoyed. The content-based system did not do so well.

There is still work to be done with this project. Foremost, I want to figure out why there is this systematic error in all the systems. I thought I found the answer already (having to randomize the holdout set) but that turned out not be the only culprit. Fixing the systematic error would significantly improve all the models, so would by my top priority. Other than that, I would work to improve the content-based model. I’m fairly sure that the hybrid model’s “stupid average” aggregation method performed the best because it put the least weight on the content-based system, which was pulling the whole system down. Having a better content system would, hopefully, improve the final model overall. I could probably do that by building more rigorous features for the jokes, such as shingling or other NLP techniques.

Some other problems I had to deal with included computational bottlenecks where my laptop was unable to evaluate large holdout sets, having to work with NaN values in my calculations to not inflate the averages with 0’s, and general debugging problems. I would also say that having to read and classify all 100 bad jokes was a problem.

Overall, I would classify this project as a success. I may not have become better at telling jokes, but I built and honed multiple recommender systems to accurately analyze user preferences and predict future rankings. That’s probably worth just as much in the long run.