IMDB

The first thing we decided to do was work off a similar database. We would then ask different questions about the data.

Mike : Can you predict the genre a user is discussing in a review?
Robbie: Can you predict the overall movie rating based on the summary of a review?
Robbie: Can you predict the user's movie rating based on the summary of a review?

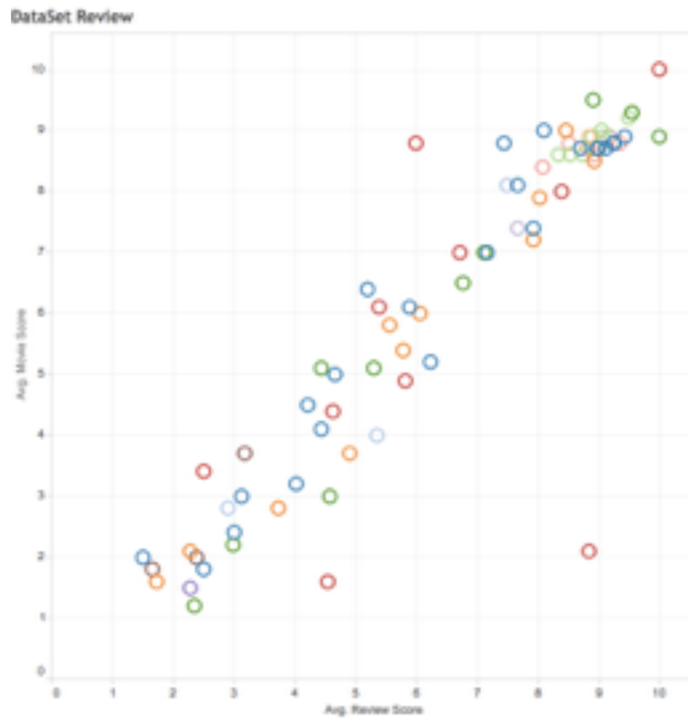
In order to get the best data, we wanted an evenly distributed data set based on genre and movie rating. We randomly chose 9 movies from 6 different genres.

We sampled Action, Comedy, Horror, Sci-Fi, Animation and Documentaries.

We decided to choose one movie from each rating score. IE. One for rating of 1, one for rating of 2, etc.

We added a sleep timer to our api function because we were encountered a 502 error and this was very helpful when we were downloading more than 200 reviews. We downloaded as high as 2000 but working with that much data was too much for our computers and we decided to move forward with the 200 reviews per movie.

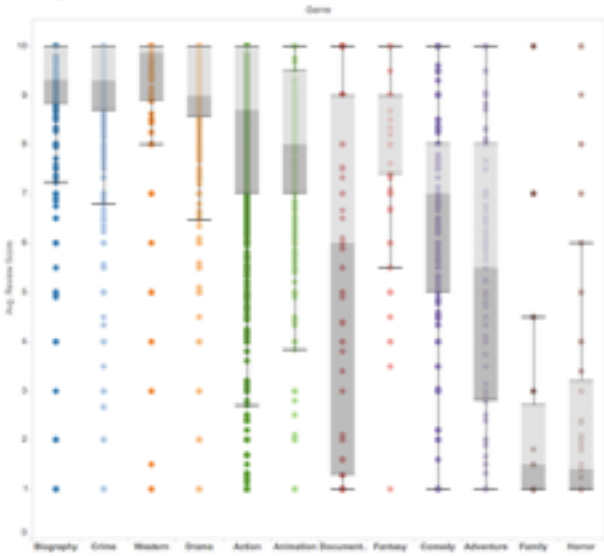
This graph shows that we did a pretty good job choosing a strongly correlated data set to start with.



Average of Review Score vs. average of Movie Score. Color shows details about Genre. Details are shown for Title.

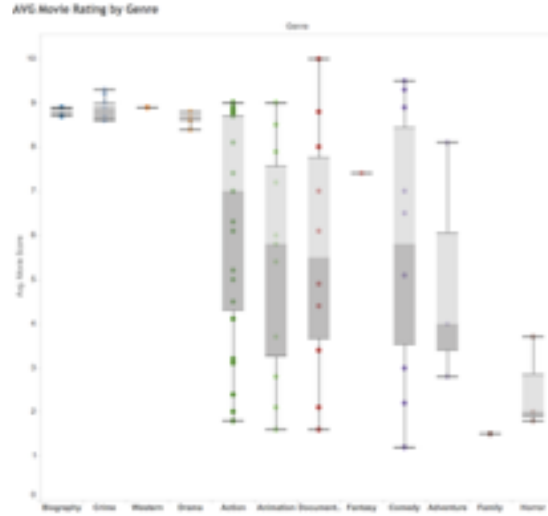
Additional visuals can be seen below.

AVG User_Review by Location



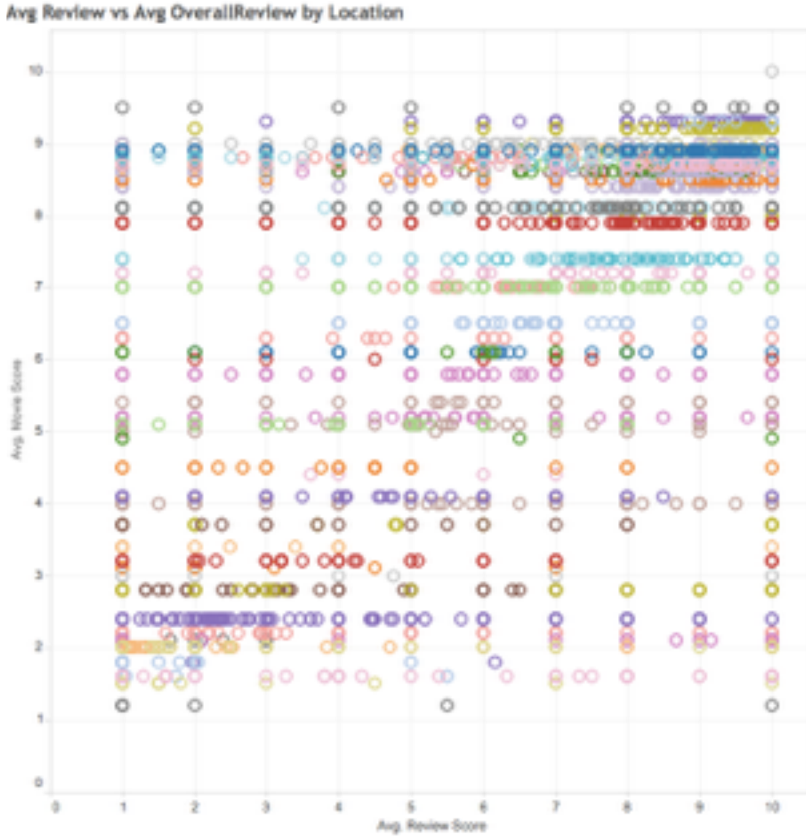
Average of Review Score for each Genre. Color shows details about Genre. Details are shown for Location.

AVG Movie Rating by Genre



Average of Movie Score for each Genre. Color shows details about Genre. Details are shown for Title.

Avg Review vs Avg OverallReview by Location



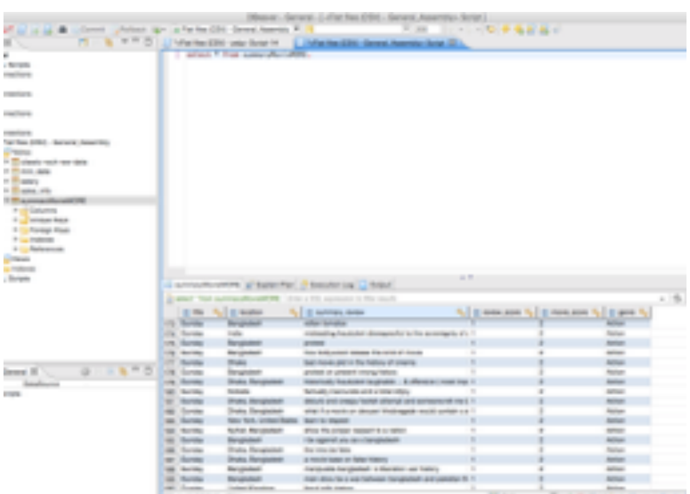
Average of Review Score vs. average of Movie Score. Color shows details about Title. Details are shown for Location.

Paul:

We originally focused on the reviews and because I had the weakest computer, I decided to download summary's instead and see if that would help. These summary's potentially represented the same view they were expressing in the longer winded reviews. I encountered several issues like not knowing when to vectorize and could not get lemmatization or stemming to work. I was able to figure out the proper order and get to a decision tree. You can see this at Project6_Fail_FirstAttempt.ipynb. I started over and did the rest of my work in Project6_Paul.ipynb.

It was interesting to see that 'bad' was the most determining feature.

My data loaded into a database:



Blockers

Major blockers was lack of computing power.Having an underpowered computer, I was constantly waiting for my computer which has probably been pushed to it's limits. It would be nice if we could cover cloud computing as I have noticed a lot of companies are looking for people with AWS experience. It would be nice to work with a data set that is already clean so we can practice doing NLP.

Robbie:

This notebook picks up where after we scraped a random* set of movie reviews from IMDB's website that we felt represented the population of movies as a whole. Once we obtained the data we put it in a local database for storing. it in a local database. We will begin here by pulling our scraped data set in to our notebook. The data set contains: summary reviews(summary_review), summaries of reviews written by IMDB users; review scores(review_score), the score each reviewer gave the reviewed film; and movie score(movie_score), the overall average rating from each individual's review score. We have a total of 79 titles in 12 different genres. In total this gave us 11,038 individual reviews. The everage review score given was 7.05 and the average movie score is 6.97. The average of review scores and of the overall movie scores are nearly eqaul as we would expect. The minimum review score was 1.0 and the maximum was 10.0. The minimum movie score was 1.2 and the maximum was 10.0. Movie scores have a lower standard deviation than review scores indicating that the distribution of movie scores is more consistent and has less variability than the review scores. These summary-type statistics can be seen below.
*Random might not be the best description of our data selection. I'll attempt to explain: We selected 10 movies from 12 different genres. Each movie was selected by it's movie score 1-10. That is to say we selected one drama with a movie score of 1, 2, 3 and so forth up to 10 == 10 films per genre. These movies were selected at random.

Now that we have our data set we are ready to begin the process of finding words or phrases from which we can derive some quantative value. We found this to be an extraordinarily difficult task. After many hours of data wrangling we were able to get some value from our data. Unfortunately, however, we were not able to get exactly what we wanted.

Let's begin by a general explanation of our efforts. We began this project using entire reviews instead of summary reviews. This gave an exceptional amount of data, so much data in fact, that it crashed our computers. This process was painstakingly slow as we had to wait long periods of time while many of our steps were processing. Eventually we abandoned the idea of using entire reviews and adopted a new strategy where we would only use review summaries, which are much, much shorter than reviews, let me assure you. So...alas! we have our data set and we are now ready to vectorize the summary reviews into something we can work with--quantitative variables (we hope they will become our features, in fact).

This is where I experienced the bulk of my problems with this project. You can see from the hashed-out code below that we attempted to lemmatize our data before using any of sklearn Vectorizers. It didn't really work the way we'd hoped and we ended up with a lot of words but nothing we were able to really work with in the end. Next we tried using a regular expression to clean the summary reviews of all the nasty imperfections it contained. This sort of worked but we still had a lot of garbage we didn't want. We were never able to manipulate our data into a short, manageable list of words. Eventually we settled on simply putting our summary reviews into CountVectorizer. Some of the problems we encountered included not being able to get 'stop_words' to work and trouble with the regular expression used in the parameter 'token_pattern'. Eventually we settled on setting the max_features to 20 and using an ngram_range of (4,4). By doing this we were able to get some short phrases that seemed to have some value in predicting a movie's score based by it's review content.

The next several lines of code will show how we set our target variable, movie score, and our feature variables. Below you also see how we used the CountVectorizer. First we instantiated the vectorizer, then we fit and transformed the summary reviews and finally we changed it from a sparse matrix to a dense matrix. Following those steps we split our data down the middle into a training and test set.

Feature section using Decision Tree Regressor. MSE score was: 6.3064

Feature section using Decision Tree Regressor. MSE score was: 6.3064

Feature Selection Using ADABoost Regressor. MSE score was: 6.2709

Feature Selection using Gradient Boosting Regressor. MSE score was: 6.3055

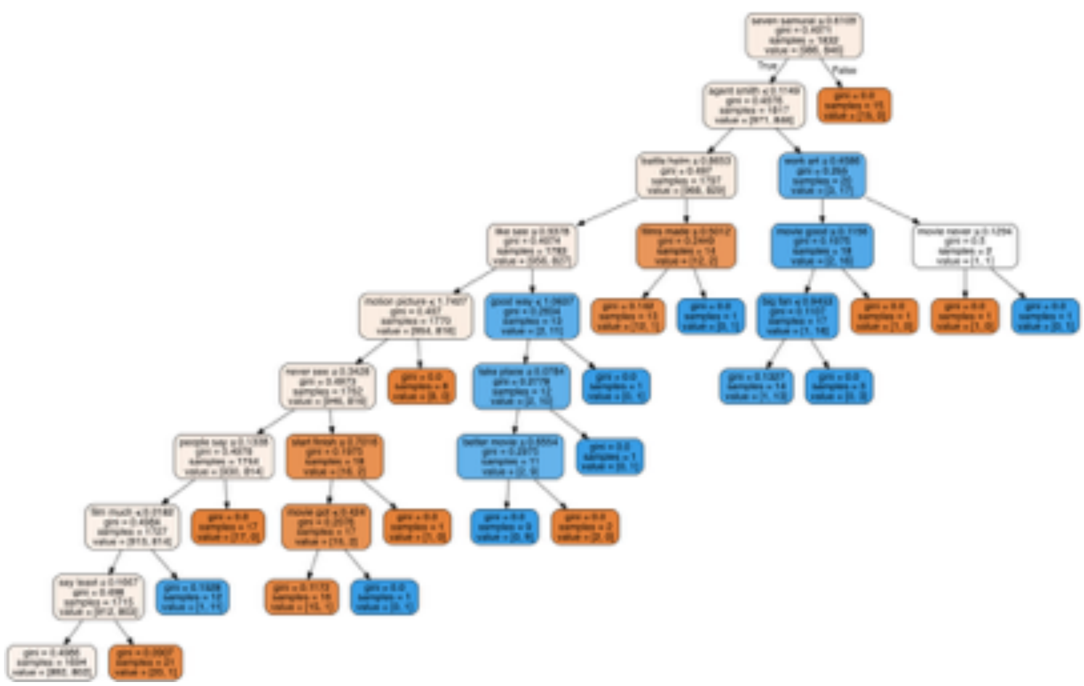
Mike: (Scripts can be found in MF folder of submission)

Can you predict whether a particular user will overrate or underrate a film based on the review? In a test market for a film, this would be useful for determining which reviewers write accurate reviews or pinpoint users with unusual tastes in film to inform advertising and marketing strategies.

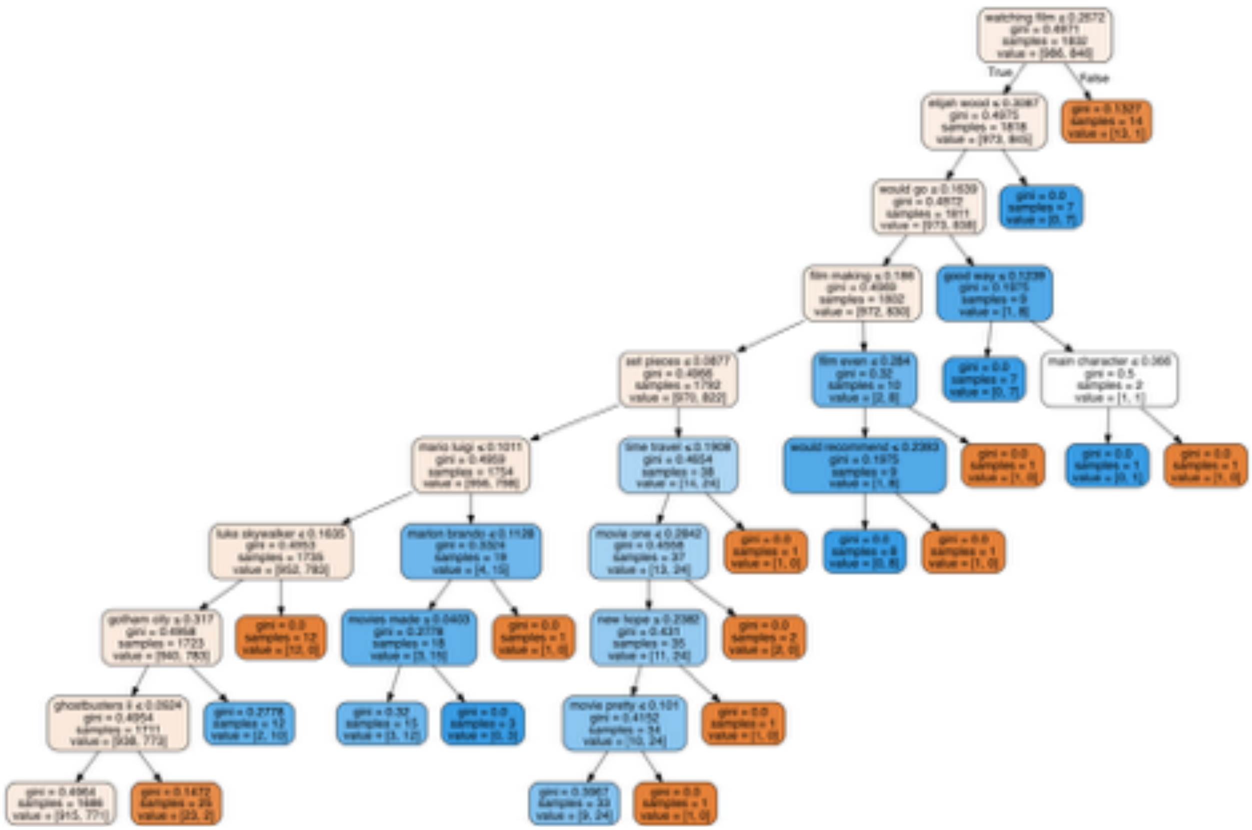
I first attempted to scrape reviews from IMDB using beautifulsoup. However, after finding that beautifulsoup couldn't load the entire page, I abandoned this approach and used api calls from the imdbpie library because it simplified getting matched information for review, review score, genre, user, etc. The scraping code and data can be found in the file MF_BS_project_6_scraping.ipynb, and the work with API calls can be found in Frantz_proj6_final.ipynb.

First, I created a column that was the difference between the user rating and the movie rating. I found that the median difference between the review score and the movie score is about 1.2 points out of 10. I then variable that identifies users that were not within 1.2 points of the movie's score. I used natural language processing to identify words and phrases that identify outliers within the population of reviewers.

To do this, I first used a 'bag of words' model. The target was outlier/no outlier, and the features were the count-vectorized reviews. Since there were many features, I limited the features here to 1000. I then fit this data to a random trees classifier. The model was relatively poor, with an accuracy score around .54, and depending on the split, the false positives and false negatives almost always outnumber the true negatives, which in this case are the outliers. Here is a graph of a decision tree for bi-grams that are influential in determining whether a review is an outlier.



Since this model was relatively poor, I decided to perform tf-idf to account for the prevalence of low-importance words in reviews. However, this led to a model with a similar accuracy but seemingly worse predictors. Notice that, in the following tree, many of the splits are done based on movie titles and characters in the movie. This may indicate that outlier reviewers are more familiar with the characters, story, or backstory and like or dislike its portrayal in the movie.



It's possible, because we only had 81 movies, that our models are overfitted. We have many rows per movie, which reduces the variance of review scores for each movie, thus increases the bias. Perhaps sampling from more movies would add the variance we need within score differentials to build a more accurate model.

One issue we encountered as a group was that with large numbers of reviews our kernels would crash. To improve upon our work and build more accurate models, we would need to collect reviews from more films and use AWS or another cloud computing service to run our scripts.