Lukasz Borowczak (lboro2), Tony Wu (tonywu3)
Kaggle: Tony and Lukasz
CS412

Final Report

Introduction

In this project, we aimed to predict how a user will rate a movie based on previous users' ratings, their profile, and information about the movie. The most challenging part of this project was cleaning the data and shaping it into a usable format for the classifiers we used. We used various techniques for dealing with missing data including: applying the most common value, using the average value, and treating missing values as their own value. In this project, we implemented our own classifier through an averaging methodology and accounting for previous ratings from the user and of the movie. We also implemented another classifier, a Naive Bayes classifier, to compare with and see which one performed better. The significance of this project can be extended beyond the scope of movie ratings. Large companies today are already using classification techniques to selectively push advertisements to individuals to boost their sales, as well as to suggest movies for users to watch that they think might fit them. This project will allow us to gain some insight to how that process might work.

Project Description

As stated earlier, the goal of this project was to implement a classifier to predict a user's rating for a movie given a user and a movie. In our previous efforts, we used the decision tree classifier from the scikit-learn package to see how well that classifier performed. Overall, this decision tree classifier did pretty well, scoring around 0.37, or 37% accuracy, on Kaggle. We were unsatisfied with the decision tree classifier due to its runtime and we wanted to improve our accuracy so we started experimenting with different classifiers before ultimately implementing our own.

The next classifier we looked at was the Naive Bayes classifier, which we implemented ourselves. The most challenging part of the Naive Bayes classifier was shaping the data to be able to calculate the conditional probabilities of each attribute. We resorted to using numpy arrays to store the data and dictionaries to store the conditional probabilities for each attribute. However, the classifier runtime was also below our desired threshold and its prediction performance was actually lower than the decision tree classifier from the scikit-learn package, averaging about 0.275, or a 27.5% prediction accuracy. We suspect that its long runtime was contributed to by the multiple dictionary lookups that we did while doing this work, and also due to not having a particularly efficient OneHotEncoding method to encode the movie genres from strings into categorical values. For movies which we didn't have data for, we simply ignored that attribute, which could have contributed to the classifier's low performance. Since we still needed to increase our accuracy, and our time left was running low, we looked to other resources for help on implementing our own classifier.

One resource we came across was a blog by Edwin Chen that explained some methodologies and techniques that were used in tackling Netflix's own rating competition. In his blog, one technique that was mentioned was the “Normalization of Global Effects”, where the predicted ratings of each movie and user combination were actually comprised of multiple components. The idea was simple: Each user tends to have a bias when rating movies, either they tend to rate movies lower than others, or they tend to rate movies higher than others. Each movie also tends to have a bias when being rated: if it is a bad movie, it will probably be rated lower than most other movies, and if it is a good movie, it will probably be rated better than other movies. We had the necessary data to implement this strategy so we decided to give it a try. This method broke down the rating for a movie into the sum of 3 components: a base rating, a specific user’s effect on that base rating, and a specific movie’s effect on that rating. Based on the information from that website, we came up with the following equation to calculate ratings:

$$rating = base\_rating + user\_effect\_rating * 0.75 + movie\_effect\_rating * 0.75$$

Where ***rating*** was the predicted rating of a movie, ***base_rating*** was the mean (average) rating of the movie, ***user_effect_rating*** is the rating variation due to how the specific user rates movies, and ***movie_effect_rating*** is the rating variation due to the quality of the movie itself. We decided to multiply the user and movie effects by 0.75 as to help prevent overfitting the data, and to help keep the rating more accurate if a user has only reviewed one movie, or if a movie has been reviewed only a few times.

To calculate each component, our algorithm first parses the training data and stores everything in a numpy array for easier processing. It then generates an array that keeps track of the sum of ratings that a movie has received along with how many ratings of that movie there were in total. It also generates an array of the sum of ratings a user gave along with the total number of ratings the user gave. From this point, it is easy to calculate the necessary data. For each user and movie, we calculate the average rating they gave or were given, and then store how that rating deviates from the average rating of the whole dataset. From that, we are able to predict the rating of a user and movie combination using the formula above. The classifier’s runtime was much faster than any of the previous classifiers we had tried, and it was surprisingly accurate, with an accuracy score of 0.408, or 40.8%, which was significantly better than our previous efforts, predicting the correct rating at over double the accuracy of random chance.

Although we were pleasantly satisfied with its performance, the classifier only took into account the ratings of each movie and ignored all other attributes. We wanted to see how integrating other attributes, like gender, would affect its performance. We started with incorporating gender into the formula because we felt that gender would be a good way to add some granularity into the equation, since different genders have different movie preferences and rating behavior. Calculating the gender differences was somewhat difficult because it needed to be calculated for each movie. However, incorporating genders actually negatively affected the classifier’s performance. We speculate that this was because we did not account for cases where a majority of the ratings of a movie were done by one gender. This would result in having a small amount, or even only one point, of rating data for the opposite gender, which was weighed the same as the other factors when we included it in the formula which would result in very skewed data for that specific movie. This often does happen in the real world, where the majority of viewers of a romantic movie may be female, and the majority of viewers of a fantasy movie may be male. We also did not apply weights to the gender differences and data amount differences, which could have mitigated this skewing. We also could have experimented by incorporating other attributes but were constrained by time, as we tried many other methods before coming up with the above formula. We attempted to try generating a decision tree for each user and each movie using scikit-learn's built in decision tree classifier to see if it would perform better, as we did not want to implement a decision tree manually without seeing if its performance could be improved. This did worse than having a decision tree for all the data, and so we scrapped the idea and then started to work on implementing the naive bayes classifier. This turned out to be somewhat difficult, took a long time to test due to its runtime, and in the end was still not as good as the original decision tree. Finally, we decided to go simple, and came up with the formula described above.

Final Results

In our previous effort with the decision tree from scikit-learn, the classifier achieved an accuracy of 0.37. However, the runtime was slow, so we looked to another classifier to improve runtime and accuracy. Our next approach was to implement a Naive Bayes classifier. This actually turned out to be slower due to the structure that was used to generate the probabilities, and only achieved an accuracy of .275 based on our own testing.

We then used the equation we came up with after viewing Edwin's blog about the Netflix Prize and implemented our own classifier, which achieved a .41 accuracy and was much faster than our previous efforts. Our attempt at incorporating the gender attribute into the classifier actually worsened its performance. This negative effect could be due to skewing on specific movies where a specific gender was responsibly for most of the ratings. Due to its negative effect, we decided to stick to the simpler and faster classifier that we originally had. In the future, the classifier could be improved by incorporating the other attributes and also assigning proper weights to each attribute, calculating them for each attribute.

Work Distribution

Lukasz: Implemented the classifier based on insights gained from Edwin's blog, worked on the report.

Tony: Implemented the Naive Bayes classifier, worked on the report.