Using Twitter and Sentiment Analysis to Predict Metacritic Scores

Jacob Stovall **Aidan Spies** **Christopher Mendez**

Abstract

Metacritic is a review aggregate website that gathers critics' reviews of entertainment products and creates an aggregate score based on those reviews. Our group was interested in trying to create a regression model, using tweets as input, that could predict the Metacritic score for a product before it was released. We started by collecting tweets using Twitter's Search API. We then built a regression model using a training set of tweets, and then, we tested the model using a separate test set of tweets. After we had the model, we built a user-product prototype website where users could input a topic and then get tweets about that topic, sentiment scores for the topic, and then a predicted Metacritic score for the topic. The final regression model ended up being highly inaccurate due to a small dataset (which was due to the fact that Twitter's Search API only allows people to query tweets that were published 7 days or less from the query date). However, we still believe that our final project is interesting because our user product-prototype is easy and fun to use. We also believe that the current regression model could be improved by getting more data and looking at different data sources.

1 Introduction

Metacritic is a review aggregate website that collects critics' reviews (and scores) and creates an average score for a product. This score is on a scale from 1 to 100 (with 1 being the worst possible score and 100 being the best possible score).

Our group was interested in looking at if public tweets made before an entertainment product's release (whether it be movies or video games) could be used to predict the Metacritic score for a product. By being able to predict this score, customers could make better spending decisions (by knowing if a product is likely to be good or bad). In addition, companies could use this prediction to figure out how to change their marketing strategy and/or improve their product (if they still have the ability to do so).

2 Implementation

This project was implemented using Python (specifically Python 2) and has 3 different parts.

The first part is the data collection python file which was used to collect tweets from Twitter. Using a python library called Tweepy which utilizes the Twitter API, the tweet_collect.py file has a function that looks for tweets with a certain keyword (or set of keywords) and downloads them to a csv file. To use the file, the person wanting to collect tweets must import the file using a Python REPL terminal and then call the function to get the tweets. After the function is called, the tweets will be saved in a csv file in the Tweets folder inside the Data folder. More information about the actual data collection process can be found in the Data Collection section. Below is a screenshot example of this process:

```
In [1]: import tweet_collect as tc

In [2]: tc.collect_tweets_from_five_days_ago(1000,"Bad Santa 2")
```

The second part is model building which was done mostly using a Jupyter Notebook (formally known as an IPython Notebook) and a python file called model_building.py. The actual notebook is a little messy because it was modified whenever new data came in (because we were testing out different ideas), but it shows some of the processes that were thought about when creating the regression model. In the Jupyter Notebook, all of the tweets were loaded and cleaned (removing stop words, links, username references, retweets, and the topic words) for each topic. And for each topic, the average negative sentiment score per tweet, the average positive sentiment score per tweet, and the percent of unique tweets were tracked. These variables are what we used to create the linear regression model, and this model was used for testing and for the user-product prototype. The number of times each word was used for a topic was also tracked, but we decided to not include it in the model because usually a single topic would tend to dominate the use of a word or the number of times a word would be used in each topic would be really small. In addition, the average positive and negative sentiment scores for unique tweets was also tracked, but we decided not to use it because there wasn't a good correlation between Metacritic scores and those attributes. To be honest, there wasn't a good correlation between any of the variables and Metacritic scores, but that will be discussed more in the experiments section and the discussion and conclusion section.

The third part is the user-product prototype. By running the command "python data_server.py", the user can start a local web server, and, in their browser of choice, choose a topic they want to get a predicted Metacritic score for and choose the number of tweets they want to receive about the topic. The server will then start streaming tweets from twitter, calculate the sentiment scores for the tweets, check if the tweets are unique, and then it will send the data to the browser. The browser will then display this data to the user and calculate the predicted Metacritic score based on the data sent from the server and the model created from the data from parts 1&2.

3 Experiments

The following section will go into our process of collecting data, building the model, and testing it out.

3.1 Data Collection

The data was collected using a file called tweet_collect.py. This file would be imported into a python REPL console, and then the function *collect_tweets_from_five_days_ago* would be used to collect a certain number of tweets about the topic (as specified in part 1 of the implementation section). If not enough tweets were retrieved by the function, the dates would be modified in the code to give more of a timespan for which the tweets could be collected (also the previously collected data would be deleted in order to prevent possible duplicate tweets).

Since we were trying to predict a Metacritic score for products that have not been released yet, we had to pull tweets from the past. However, Twitter's Search API only allows for users to get tweets published seven days prior to the current date. Because of this, our dataset was very small because we needed to be able to pull tweets before a product's release, but we also needed to be able to get the Metacritic score after the product was released so that it could be incorporated into our model. In the end, we ended up getting 1000 tweets for 20 different movie and video game properties. We decided to use 15 datasets for training the model, and we kept 5 datasets in order to test the accuracy of the model. In addition to tweets, Metacritic scores and a sentiment dictionary were also collected as data. The Metacritic scores were collected by hand and entered in a text file to be used as input when building the model. In addition, the sentiment dictionary was imported from a text file created by Finn Arup Nielsen which has a list of words and phrases with sentiment scores from 5 (being very positive) to -5 (being very negative).

When analyzing the tweets for features, the tweets were first cleaned so that stop words were removed, links were removed, twitter usernames were removed, the string 'rt' (which means retweet) was removed, and words in the topic search were removed (for example, if the topic

was “Forest Gump”, then the words “Forest” and “Gump” would be removed from the tweet during feature extraction). After this was done, the tweets were split into arrays of strings and the arrays underwent sentiment analysis and uniqueness checking (basically, the code checks to see if the array of strings has occurred more than once in the dataset). Each tweet’s sentiment data and uniqueness was then aggregated for the topic and the final features were then created.

After getting all of the features, we then built the regression model. Our final model ended up being:

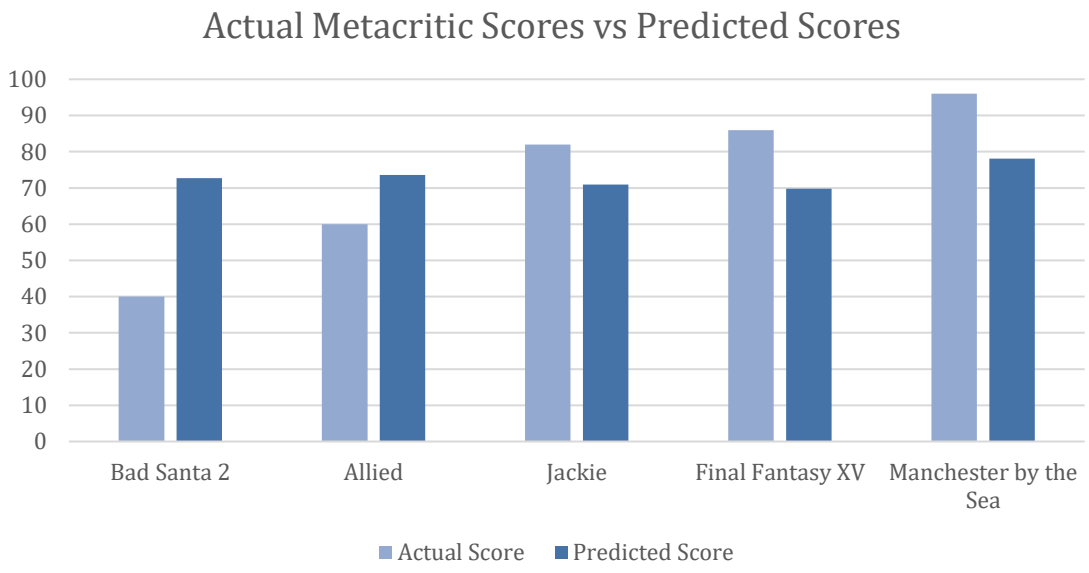
Predicted Score = 65.6814 + 18.9509 (avgNeg) + 0.5334(avgPos) – 3.2850 (unique)

where avgNeg = Average Negative Sentiment for Topic (Total Negative Score / # Tweets)
avgPos = Average Positive Sentiment for Topic (Total Positive Score / # Tweets)
unique = Unique Percentage of Tweets (# Unique Occurences of Tweets / # Tweets)

The final model isn’t that good for a couple of reasons. The first reason is that t-tests for the coefficients show that none of the variables are statistically significant. The only variable that did relatively well when looking at different combinations of features was average negative sentiment. This makes a little bit of sense because non-advertisement tweets are more likely to have negative sentiment words, and thus, more critically acclaimed films may have more people talking about it (and thus, negative sentiment could be positively correlated to a better Metacritic score). However, whenever more datasets were brought into the model, all variables tended to shift in wildly different directions. The second reason the model isn’t that good can be found in the Experimental Results section.

In addition to data collection for the regression model, data is also collected when using the user-prototype. Using the Twitter Streaming API, tweets are sent to the server based on the user’s topic query. Then, the tweets are cleaned and features are extracted from the tweets. Then, using Socket IO, the tweets and features are sent to the user to look at and learn about.

3.2 Experimental Results



As you can see by the chart above, when predicting the Metacritic score, our model tended to stay in the same range (even though the products we tested on had a wide range of scores). The average percent error for the model when testing was 31.05%, which isn’t very good when trying to predict some sort of continuous value. However, the one small victory for the model is that the highest scoring movie also got the highest predicted score.

Despite the issues with our model though, we are quite happy with the user-product

prototype that we built. When testing it out with high frequency topics (such as Trump), it could handle the amount of traffic very well when the user was requesting a few hundred tweets (although there is a bit of slowdown when trying to look at thousands of data points in the browser).

4 Discussion and Conclusion

In the end, while the regression model ended up being lackluster, the user-product prototype was pretty neat and fun to use. If we had more time to work on this project, here would be some of the next steps we could take in order to improve it:

- **Gathering More Data –** It's hard for a regression model to work properly unless there's a lot of data points to work from. As stated in the Data Collection section, it seemed like there could be connection between average negative sentiment and Metacritic scores, but without more data, it's not possible to know if there truly is something there and/or if the coefficient of the current model should be changed.
- **More Exploration of Advertising Impact** - We didn't want to filter out duplicate tweets because if something is retweeted, then someone probably has the same sentiment about a topic as the original poster of the tweet. So, we included uniqueness as variable in the model just in case there was a correlation between a large number of duplicate tweets and Metacritic score. But, there ended up being no significance according to the t-test. However, there is a chance that advertising tweets could be skewing the sentiment scores, and thus, possibly hiding a good insight. Therefore, a next step might be to find a way to classify advertisement tweets vs non-advertisement tweets and remove the advertising from the model.
- **Using Different Data Sources –** Even after getting more data and exploring advertising tweets, the most beneficial thing might be to look at different sources of data to try and predict Metacritic score. While tweets might not give a lot of data, there might be other social media websites or market trends that could be a better indicator of what the Metacritic score for a product will turn out to be.