Kyle Wojtaszek

Final Project

3/15/19

Analysis of Hacker News Comments

Introduction

This paper will explore the comments from the Hacker News archive posted on Kaggle[1] (via Google BigQuery). This data set includes 'scores' for each comment and story and we are going to attempt to determine what makes a top-rated comment vs. a poorly rated comment. We'll begin by doing some exploratory analysis on the comments and end by making a model to predict the quality of a comment.

Data Set Information

This started out as a very large dataset, close to 40GB, which seemed a bit unwieldy. To cut down a bit on this, I started by only looking at stories that had a score of at least five and comments with a score of at least two. This eliminated a lot of noise that was likely not seen by enough people to accurately score it. Obviously, I also excluded stories that had no comments. This brought the data set down to about 900MB, which is a lot more manageable. This left us with ~1.5 million comments.

For each story and comment, we have to following information:

- Author – Unfortunately these are anonymized, and the anonymization is not global. So, we cannot compare authors across threads. However, for a given thread the author codes are the same.
- Time Posted
- Score
- Full Text

And for the story itself, we also get a title and the URL it points to (if it points to a URL). From that, I engineered a few more features that would be more useful for our analysis:

- SameAuthor (Boolean) – Is the comment written by the same author as the story post?
- ScoreRatio – The ratio of the comment score to the story's score. This allows us to normalize the score across stories. For the most part, more popular stories have more popular comments, since more people are reading the story.
- MinutesAfterStory – The time after the story post that the comment post was made in minutes.
- WordCount and CharacterCount – The count of characters and words in the comment.
- Sentiment – We will be using VADER from the NLTK package to measure the sentiment of each comment.

There is one additional feature that we could conceivably pull from the data set, and that is the number of child comments that a comment has. In general, we would expect this to have a reasonably high

[1] https://www.kaggle.com/hacker-news/hacker-news

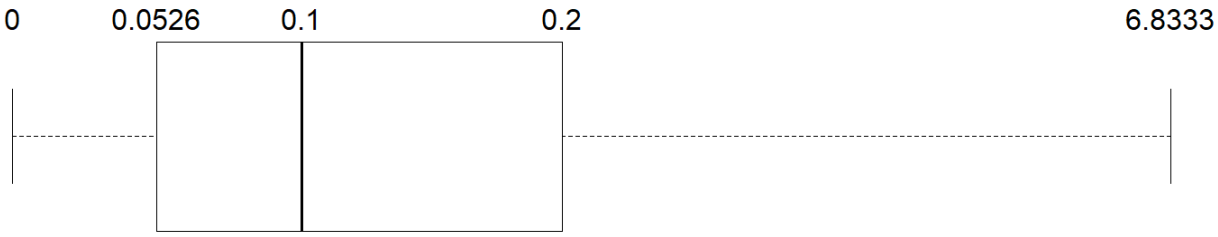
correlation with the comment score. However, this is not particularly useful when trying to solve our problem, which is to predict what a good comment would be. If we were in the state where we could count child comments, then we might as well just look at the score and not bother trying to predict. So the could of child comments will not be included as part of this paper.

Initial Exploratory Analysis

First, I wanted to take a quick look at whether or not certain properties had any effect on the score ratio so that we know what to feed into our model.

Score Ratio Distribution

Below is the boxplot for the ScoreRatio. Note that the max of 6.833 is not drawn to scale, as it would dwarf the rest of the plot:



As we can see, the median comment appears to be about 1/10th of the score of the story it is commenting on. From here on, we will define a 'good' comment to be any comment above the third quartile of 0.2, and a 'bad' comment as any comment below the first quartile of 0.0526. Comments in the middle are considered neutral and will be dropped from the model building process, as the difference between those comments are not as great and may just be noise.

SameAuthor

There were 14,400 comments in our 1.5 million comment sample that were written by the same author as the story. These same author comments have an average score ratio that is 0.06 higher than the overall average score ratio. This appears to be relatively significant, so we will provisionally include Same Author in our model.

MinutesAfterStory

I measure the Pearson correlation coefficient between MinutersAfterStory and the ScoreRatio. Unfortunately, there was not a strong correlation, only 0.2. But there is some correlation, so I'm going to leave it in, but I would not be surprised if it turns out to be insignificant.

WordCount And CharacterCount

Surprisingly, there is virtually no correlation between the ScoreRatio and word count or character count, the Pearson correlation coefficient is -0.04 and -0.03 respectively. I also manually graphed the counts and ScoreRatio in order to look for a non-linear relationship and there doesn't appear to be any relationship between them at all. We will be excluding both CharacterCount and WordCount from our model.

Sentiment

This one turned out interesting, I used VADER to rate each comment as either positive, negative, or neutral and them compared the distribution between the good comments and the bad comments. The good comments have a 59% positive, 21% neutral, and 20% negative spread, while the bad comments were 64%, 12%, 24%. So it appears that the neutral comments seem to trend towards the good side, while more sentimental comments on either side seem to push it more toward a bad comment. In our model we will use the raw compound, positive, negative, and neutral scores from the VADER algorithm. This will allow each category to be evaluated separately, just in case there are some complex interactions here.

Creating A Model

For my model, I'm going to take a two pronged approach. I will create one model from our engineered features and one model that compares the vocabulary of good comments vs. bad comments. We'll compare the accuracy of the two models and also attempt to create an ensemble model by combining the two.

Engineered Feature Model

First, we'll put together a model using the various engineered features described in the previous section. We've got six features: MinutesAfterStory, SameAuthor, compound, neg, neu, and pos (from the sentiment analysis) and we are trying to predict a binary variable (whether the comment is good or bad). With this layout we can try a variety of model types. The table below shows the algorithms I tried and their accuracies. We should keep in mind that since we selected the upper 25% of the data as good comments and lower 25% as bad comments, our data set is perfectly balanced at 50/50, so our benchmark for comparison is 50% accuracy.

Logistic Regression	50.4%
Multinomial NB	51.7%
SVM	55.6%
Random Forest	59.6%

These results aren't great, but by the end they were a bit more promising. Logistic regression essentially performs the same as random chance. This isn't surprising as logistic regression only captures simple relationships, and as we already discovered, the relationship between sentiment and good vs. bad is not simple. Naïve Bayes does a little bit better, but still not much better than the benchmark. SVM is the first breakthrough where we see some actual separation from the benchmark. Unfortunately, the SVM model took about 12 hours to train, so I did not want to iterate much on tuning it. I tried a few different sets of parameters and they only made minor differences (on the scale of ~0.1%) in the accuracy. Finally, random forest is able to get us pretty close to 60%. While this isn't an astounding success, it is definitely a large improvement over our initial logistic regression model. My initial random forest model, with 100 trees and a max depth of 5 was at 58.9% accuracy. By increasing the max depth to 10 and the number of trees to 200, I was able to get up to 59.6%. Any further increases didn't have much of an effect. Let's take a quick look at the feature importance from our random forest model:

Minutes After Story	0.308
Same Author	0.017
Compound	0.144
Neg	0.202
Neu	0.184
Pos	0.145

Based on this, we can see that the minutes after story variable is the most important, followed by the VADER output. The SameAuthor column is the least important, however I tried removing it and the model became about 1% less accurate, which over the approximately 750,000 comments we have in our sample is about 7,500 comments, which is a significant amount, so I chose to leave it in. If there was a requirement to minimize the number of columns, SameAuthor would be the first to go.

Vocabulary Model

For our vocabulary model we are going to try both Multinomial and Bernoulli Naïve Bayes models. Based on our experience with the binary model above, we are not going to try SVM due to run time constraints. I tried eight different combinations of parameters to the count vectorizer and NB model, using 10-fold cross validation to score them. Below are the results:

Naïve Bayes Model	**Stop Words Included**	**N-grams**	**Accuracy**
Multinomial	Yes	1	64.6%
Multinomial	No	1	64.1%
Multinomial	Yes	1,2	65.6%
Multinomial	No	1,2	64.8%
Bernoulli	Yes	1	63.1%
Bernoulli	No	1	65.0%
Bernoulli	Yes	1,2	64.0%
Bernoulli	No	1,2	63.8%

So, as we can see, the Multinomial NB model with stop words and 1,2 grams included is our highest accuracy model. Based on the entire table, it appears both stop words and 2-grams are important to this particular data set. Let's take a look at the most and least important words in our model:

	Least Important		Most Important
-16.9838	00 000	-6.884	com x2f
-16.9838	00 000000	-6.879	want
-16.9838	00 00110010	-6.8696	code
-16.9838	00 00z	-6.8366	ve
-16.9838	00 017437	-6.8215	really
-16.9838	00 027	-6.8083	make
-16.9838	00 037139	-6.7983	http x2f
-16.9838	00 05	-6.693	use
-16.9838	00 067151	-6.6356	think
-16.9838	00 08	-6.6262	good

All of our least important words are numbers (this trend continues through at least the bottom 50 words). We could remove them, but as we see in the most important column, numbers show up there too (x2f), so we are going to leave them. The only trend that really sticks out in the most important column is that there are a lot of pieces of URLs (a trend that continues through the top 30 or so). “X2f” is the Unicode character code for the backslash. This information tells us that it may be worth our time to add a new feature to the dataset that indicates if a comment contains a URL or not. That would also allow us to remove the URL from the comment text, which might clean up the vocabulary a bit. This is something I would consider for the future if further iteration happened on this model.

Ensemble Method

Now that we’ve got both of our models, we can take the probability from each model and average them together and use that to predict the quality of the comment. Our Random Forest model had an accuracy of 59.6%, while our vocabulary based Naïve Bayes model has an accuracy of 65.6%. When we combine the two, we end up with a 66.5% accuracy, which is almost a full percent better than the vocabulary model. So, this is a valid technique for this data set and does provide value.

Conclusions

Overall, this turned out to be a more difficult problem that what I imagined when I started. I figured I’d be able to at least get accuracy in the 70’s, maybe even the 80’s. However, after dragging the data through many different algorithms, the best we were able to achieve is 66.5%. I think, given the circumstances, that it is a pretty good model, but nothing great. Through this experience I learned a lot about applying this type of modelling to real world datasets and some of the challenges in doing so. I also learned a lot about Google BigQuery, which I had never used before.

In the future, if we wanted to continue to improve this model, there are a few things we could try. First off, pulling out URLs from the comments and making a new feature that indicates whether a comment contains a URL or not seems like something that would be a potential gain. On top of that, it would be nice to use LDA or something similar to determine of comments are “on topic” with the story that was posted. I tossed around a few implementations of this in my head, but was never able to come up with something solid enough worth trying.