Project Fletcher

This project focused on classifying a submission to reddit.com/r/showerthoughts as a success or a failure. Success was defined as getting one or more upvotes, while failure was defined as not getting at least one upvote.

Data

I obtained my data using a dataset of all reddit submissions and comments available on Google BigQuery. Using a simple SQL query, I was able to get data on 1,000,000 showerthought submissions, including the title of the post (which was the NLP focus of my project), the time of submission, and the score the submission received. The showerthoughts I collected were submitted between 2015 and September 2018. I only went up to September of 2018 because I wanted to make sure each showerthought had at least one month to accumulate upvotes, which helps to put all showerthoughts on an even playing field.

I ended up using only 10% of this dataset, effectively decreasing my data to 100,000 observations. I found 1,000,000 showerthoughts to be cumbersome to work with locally. In the future, I would use AWS in this situation, but I was wary of moving my entire project to the cloud halfway through my project.

Tools

Python

- **Pandas**
- **Numpy**
- **Matplotlib**
- **scikit-learn**
- **Imb-learn**
- **Gensim (word2vec and LDA)**
- **Skopt (BayesSearch)**

Google Docs
Google Slides
Javascript
HTML
Flask
Tableau (to generate word cloud)
Jupyter Notebook

Modeling

This project felt like a mad dash to try as many methods as I could to get *something* out of my model. I felt confident in my ability to do the supervised learning part of the project, but getting predictive features turned out to be the hardest part. I decided on AUC as my model selection metric, because I saw no good reason that true positives should be valued over true negatives -- it's just as important to know when a showerthought is bad as it is when it's good.
Before I started using more advanced modeling techniques, I made a simple bag of words model using a count vectorizer with a random forest, and got a baseline of about .59 cross validated AUC. I also used TF-IDF with a random forest, but got worse results. Clustering on both the TF-IDF and count vectorizer models proved fruitless as well. I moved forward with my count vectorizer model as my model-to-beat.

My first goal was to use LDA to get topics, then use a random forest model to predict a showerthought's success based on its LDA-generated topic probability distribution. Both Gensim and scikit-learn implementations of LDA yielded essentially nothing in terms of predictiveness. I tried clustering LDA topics with KMeans++ and DBSCAN, but still got nothing from my model. I tuned my model for hours and hours, and eventually used BayesSearchCV to optimize my LDA's hyperparameters to give me the best ROC AUC, and ended up with an AUC of .52.

At this point, I abandoned LDA for word2vec. I found a word2vec embedding from the University of Ghent which was trained on 400 million tweets and used a 400 dimensional vector space. My document-embedding method was to add up the word2vec representations of each word in a document, then divide by the number of words in a document to get that document's average representation. The first time doing this I accidentally added up each letter in each document, rather than each word. When I retrained my model using the words, my AUC improved slightly. I attempted clustering using KMeans++ and DBSCAN, but once again saw no improvement in my model after doing so. I ended up with .60 cross-validated AUC, but when I looked at test data my AUC was .56.

I tried adding a "time of submission" feature as well. I created a day of week column and hour of day column, one hot encoded both of these to make them categorical, and tried a plethora of other techniques. None of these attempts added value to my model, so I chose to leave out time data.

Conclusions

My model failed to capture the heart of what makes a showerthought good or bad. I believe there are several reasons for this. For one, I think my model lost some important meaning of showerthoughts when I stemmed and removed stop words. Stemming and removing stop words would make perfect sense for topic modeling with an LDA model, like I originally thought I was going to use, but I think word2vec is actually capable of grabbing the meaning of stop words and understanding documents a little more than without stop words.

Regardless of this, word2vec is not capable of capturing the ordering of words within a showerthought. My understanding is that a word is mapped to a distinct point in the vector space, regardless of its context (when using a pre-trained model like I did). This means I could jumble the words in a showerthought around and not see different results. I think this is the heart of why my model failed. A showerthought is not a showerthought when you change the ordering of the words. A showerthought is “good” because of the way the writer uses and orders the words, not just because they happened to choose a good combination of words without regard to order.

I also have questions about how predictive the title is of a showerthought’s success. It *seems* like the title of a submission is pretty much everything that one would base whether they upvote or downvote a post on, and it might be. However, the important thing to keep in mind is that most showerthoughts probably only get seen by a very small number of people. The success or failure of a post has a high degree of randomness associated with it, which I think is a huge factor contributing to the difficulty of this problem.

What I’d do next

The first thing I’d try is leaving stop words in my model. I expect this would marginally improve my model’s performance, but it’s unlikely to give me a huge jump. This is because a word2vec model still doesn’t take in to account the *order* of the constituent words of a showerthought. I was adding up the word vectors that make up a showerthought and dividing by the word length of the thought. This means you could totally jumble up the words in a showerthought and get an identical representation of the document in the 400 dimensional vector space.

Another idea I had was to try a Poisson regression to predict the number of upvotes a post receives based on the title. I would likely have to remove showerthoughts that get thousands of upvotes if I took this approach, as it might throw the model off. I think this too would be a very difficult problem to solve, where your model is only as good as the features you generate.

To see a large performance increase, I’d need to use a model that takes into account the order of the words in a showerthought. A recurrent neural network seems like the right tool for that job. In particular, I’d probably focus on using a long short term memory model because my research shows that these are rather effective for NLP situations. I believe this would be the most fruitful next step in my project.