CMSC726: Assignment #3 — Feature Engineering

Soheil Behnezhad

September 30, 2017

My username on Kaggle is "soben713" and the best accuracy that I got on Kaggle is 0.69341.

The methodology. To incorporate new features, I used a binary flag "debug" in the code. For "debug=True", I do a set of tasks. First, I randomly partition the training data into two partitions, the first partition is used to train and the second partition is used to test the classifer. Then I report accuracy based on the expected outputs versus what the classifier predicts. I also count the number of false positive and false negative predictions. Next, I print a sample of 50 sentences where the prediction is wrong.

A technical difficulty here is the random partitioning of the data, we need the partitioning to be random so that we do not design specific features for a particular subset of the training set. On the other hand, if in each run, the partitioning is truly random, comparison of different feature sets becomes hard. To handle this, I use random seeds. I check how different subsets of the features work on a fixed seed, then to make sure that it works for different seeds, I just change the seed multiple times.

Each feature that I use, is a function, that takes the sentence as input and updates it. All these features are called inside a function named `add features`. This makes it easier to disable different features and find out which subset of features work best.

The features and why they work. The first feature that I incorporated into my code was adding n-grams. I checked different combinations and came to the conclusion that the range (1, 5) is where the classifier gets the highest accuracy.

Appending "trope" and "page" into the sentences was the second feature that I implemented. On my local machine, this significantly improved the accuracy to as high as ~ 0.75. However, the result on Kaggle was much worse, something about ~ 0.62. After observing the datasets, I realized that none of the datasets in the testing set have the same "page" as the training sets. This describes why "page" is actually a very bad feature. The classifier learns to use page and indeed performs well on the training dataset where we randomly partition it into two subsets, leading to a high number of documents in the testing partition having the same page as the training partition. However, since the page property is a useless indicator for actual test datasets, we get a low accuracy on Kaggle.

Next, after observing the datasets, I realized that in some sentences there are certain patterns, such as (01x05) to refer to the 5th episode of the first season of a TV series. I added another feature to detect such patterns using regex. I then append `HASEPISODE_i` to the sentence where `i` is the number of times the pattern happens in the sentence. Another thing that I tried, instead of doing this, was to repeat the word `HASEPISODE` at the end of the sentence as many times as it occurred in the sentence. I expected this to work better since the CountVectorizer is based on the number of times a word appears in the text, but I got lower accuracy and sticked to the first idea.

Another useful feature that I used was to lemmatize the words. I downloaded the WordNetLemmatizer library of nltk and used it. It helped improving the accuracy.

I also counted the number of times different punctuations appeared in the sentence and for each occurrence, appended a specific word to the end of the sentence. Not all the punctuations were useful, however, by locally testing different subsets of punctuations I ended up using only exclamation mark, question mark and quotation. It makes sense that people who used exclamation mark, for example, are more likely to spoil a scene of a movie.

The final feature that I used was to remove stopwords. This was done by using the `stop_words` argument of the CountVectorizer and the stopwords list of nltk corpus.