IDS 703 Final Report

Sentiment Analysis on Taylor Swift Lyrics

Cindy Chiu and Sydney Donati-Leach

12/12/2021

Github:

https://github.com/sdonatileach/sentiment_analysis_on_tswift_lyrics

If unable to load data through API, use this file from the Github repo: 00_source_data/Taylor_200_song_lyrics_with_sentiment.csv.

Step 1:

The NLP problem that we chose was sentiment analysis on lyrics from Taylor Swift. Our goal was to label each of her songs either positively or negatively. We collected the data from a lyric API and loaded 200 of her 206 total songs. To start, we applied a few filters to the resulting dataset. We wanted to alter any contractions to their full version. For example, "how'll" was changed to "how will" in order to appropriately identify all the words. Next, we removed any stop words (identified through the nltk package) as these do not hold any sentiment. There were a few other words we found in the data that needed to be filtered out such as "urlcopyembedcopy" and any voice memos or non-lyric messages. This was necessary as it would alter the length of the song and be an outlier in our distribution of song lengths. After filtering, we were left with a subset of data that only contained the lyrics concatenated for each song.

Step 2:

We split the 200 songs into 80% training data and 20% testing data. The 160 songs are used to train both our Naive Bayes and LSTM models.

To train the Naive-Bayes model, we followed the pseudo code given in the textbook. We calculated the prior probability for each class, which is the probability of getting either a negative or positive tag from the training documents. Then we calculate the probability of each word occurring in a particular class. We joined all the positive lyrics together and used the bag-of-word method to get the frequency for each word. This word frequency dictionary will be passed into our training function. However, if a word in our testing set does not exist in the positive text, it will cause the prediction probability to be zero when we're multiplying all the probabilities for each word. To prevent this, we implemented add-one smoothing to the numerator. We also implemented add-one smoothing to the denominator. Therefore, for each word w_i in the whole corpus V, the probability of seeing it in class c will be:

$$\hat{P}(w_i|c) = \frac{count(w_i,c)+1}{\sum_{w\in V}(count(w,c)+1)} = \frac{count(w_i,c)+1}{\left(\sum_{w\in V} count(w,c)\right)+|V|}$$

After generating the probability for each word under each class, we implemented another function for making the class prediction. We passed in the dictionary of probability given to the class from the training set, the prior probability for the class, and new lyrics in the testing set. The function will tokenize the new lyrics and multiply the probability for each word. To avoid underflow, we took a log transformation on all the probabilities. Then we compare the log-likelihood of getting a positive and negative class and assign the prediction to the tag with maximum likelihood.

Last, we generate our synthetic data using the probability dictionaries returned from the training function. In order to mimic the distribution of the real dataset, we extracted the mean and standard deviation of the lyric length. Then we sampled 200 data points from a normal distribution with a mean of 169 and a standard deviation of 51. We use this list as the length of each synthetic document. Then we did 200 samples with replacements from both positive word probabilities and negative word probabilities, having 400 synthetic documents in total.

Step 3:

For our neural network, we chose an LSTM. First, we need to tokenize our sentences using Keras' Tokenizer. This will assign a number to each word in our lyric vocabulary. Also, we set the max number of features to the maximum number of words in a song. Then we can pad the sequence of lyrics for each song to ensure they all are of the same length. Next, we build our model by initializing the sequence. We add a word embedding layer, LSTM layer with 16 units and the fully connected (dense) layer with 1 (output) neuron with the sigmoid activation function. We use binary cross entropy loss and Adam optimizer to train the model. Due to the binary nature of the LSTM, we had to assign a one to the positive sentiment words, and a zero to the negative sentiment words. Finally, we triggered the training of the LSTM neural network. We use a batch size of 10 and train the network for 6 epochs.

Step 4:

After applying both approaches to the synthetic data, we found that the Naive-Bayes approach performed with 100% accuracy. This should be the case as we generated our synthetic data from this method. When applying the LSTM to the randomly generated synthetic data, the model generally performs with accuracy between 62.75% to 84.5%. The accuracy score will change each time we create new synthetic data. The way we created our synthetic data was by getting the mean and standard deviation of songs and generating different size documents each time. Then we assigned a probability of seeing a word in a positive document versus a negative document using the Naive Bayes method. The LSTM does not perform as well because the way

Naive Bayes applies probabilities is entirely different from LSTM. Another possible reason is we use word embedding for the LSTM input. For real lyrics data, the composer might use a lot of related words in a song, so there are more closely related embedded vectors. For synthetic data, the words are randomly generated from the corpus, and thus the embedding might be less related to each other, causing the lower accuracy score.

Step 5:

After applying both approaches to the "real" data we collected from the lyric API, we found that the Naive-Bayes approach performed with 100% accuracy on the training data. The testing accuracy on the real lyrics is 87.5%. One possible reason why we have such a high training accuracy for Naive Bayes is because we use the sentiment labels generated by the NLTK package. It is possible that NLTK used the same Bayes method to determine their sentiment tag. When applying the LSTM to the real data, the model performed with 93% accuracy during training and 87.5% accuracy during testing. The fact that both methods have a training accuracy of 87.5% on the real data shows that either method is a plausible solution in predicting sentiment.

Step 6:

The high accuracy of the Naive Bayes for both the synthetic and the real data shows its potential for predicting sentiment. The same can be said about the LSTM, but there are a few drawbacks to both. First, Naive Bayes assumes that all features are independent of one another. The sentiment of one word alone does not necessarily mean that was the intended sentiment of the artist. Therefore, the estimations can be wrong and we should take its probability output with a grain of salt. For LSTM, it is easy to overfit. This could be the case here as we had a small amount of data, and could be the reason why we have a high training accuracy.

Since our dataset is relatively small, we did not face computational challenges. It took more time to run the model prediction code for Naive Bayes and the model fitting code for LSTM. However, we faced module version conflicts when trying to import keras when running the LSTM model on local resources. It turned out to be a specific version of keras having a conflict with the numpy module. Therefore, we have to move our code to a cloud environment using Colab or set up a virtual environment to resolve this issue.

Even though the "lyricgenius" package gave us the ability to extract lyrics from the Genius API, it took a long time for us to get the API results. Sometimes the API timed out for no specific reason. Therefore, we only focused on getting the 200 songs from Taylor Swift. In the future, we can expand this to study a broader dataset if we can find a more reliable method of scraping the lyrics.