TF-IDF and SVC Text Classification: *Is this text a song?*

Aditya Mehta
A16062688
a7mehta@ucsd.edu

Isabel Suizo
A15485717
isuizo@ucsd.edu

Emily Zhuang
A15592650
ezhuang@ucsd.edu

Abstract

In this paper, we explore a binary categorization task for determining whether a set of text belongs to a song's lyrics. To approach this problem, we rely solely upon the body of text to generate our features with the help of TF-IDF vectorization and implicit feature engineering. To determine the optimal model given our resources, we conducted several trials varying the characteristics of our text (considering n-grams, removing punctuation, etc.) as well as including additional implicit features (counting newlines, etc.) The backbone of our model consists of an SVC classifier due to its success in high-dimensional categorization tasks.

1 Introduction

This is the full length report on using text classification, TF-IDF vectorization, and support vector classifiers to predict whether or not a block of text is a song; please feel free to find an abridged version of our paper [here].

2 Dataset

This section will detail the methods we used to collect our own raw data and the reasoning behind key decisions made during data collection, the pre-processing we performed on said raw data to ensure the data we were training our model on was applicable and relevent to our predictive task, as well as the exploration and visualization of the pre-processed dataset that informed the types of techniques we used to design our features and model.

2.1 Collection

For this experiment, we constructed our own dataset to better suit the binary song lyric classification task at hand. The positive set of song lyrics was scraped from the popular song lyric website, Genius. First, we obtained a list of 160,000 songs from a Kaggle dataset which consists of songs released between 1921-2020, used that list to query for each song using the Genius API, and extracted the lyrics from each request. (Note: No data was used from the Kaggle dataset, as it was only used as a master list for songs to query.)

For the negative set of non-lyrics, we compiled data from several sources. A majority of the negative samples were taken from Reddit submissions, the online discussion forum website. We leveraged the variation of text content in different Subreddits to compile a subset of diverse text including poetry, personal stories, and professional writing. Please refer to Table 5 which includes hyperlinks to each of the Subreddits explored in the dataset. We scraped the 27 Subreddits using the Reddit API to retrieve the most recent submissions and extracted the text data from each request. To vary the negative dataset even further, we also included reviews from the book database site Goodreads and the video game distribution platform Steam. Both review datasets were retrieved from Professor Julian McCauley's database. Finally, to force our model to also learn the patterns of song lyrics, we included shuffled instances of lyrics from the positive set, as well as a shuffled Reddit submissions and reviews.

While building our dataset, we made several intentional design choices throughout the process. First, we chose the Kaggle set as the master song list of positive lyric samples since it was a very comprehensive list that featured a reasonable spread of genres across the past century until 2020. Because of time constraints, we decided to select only a subset of song lyrics from the master list. To maintain an even spread of samples, we first shuffled the master list before we took our subset of 40,000 samples. We had much more

Table 1: Distribution of Negative Samples

Type	Count	% of Negative Set
Reddit	25000	64.1%
Goodreads	5000	12.8%
Steam	2000	5.1%
Shuffled Song	4000	10.3%
Shuffled Reddit	1000	2.6%
Shuffled Goodreads	1000	2.6%
Shuffled Steam	1000	2.6%

Table 2: Train/Test/Validation Splits After Preprocessing. The last column represents the percentage of samples from the overall dataset.

Set	Count (Positive/Negative)	%
Training	31330/30744	80%
Validation	3916/3843	10%
Test	3916/3843	10%

flexibility when generating our negative samples and settled on gathering the bulk of our negative samples from different Subreddits. We reasoned that pulling from various Subreddits would generate a diverse collection of text genres since we chose Subreddits with relatively longer submissions that each had a different theme. For example, we pulled from a Subreddit that consisted of samples of poetry, while others featured story-telling submissions as mentioned above. A small subset of our negative set also consists of reviews from Goodreads and Steam. These samples were simply to add more diversity in text content to our negative samples since our Reddit submissions did not include review-style text. Furthermore, our negative sample also includes shuffled song lyrics. This set was crucial to training our dataset to encourage our model to also take sequence into consideration. If we implemented a naive model that only considers unigrams, our naive model would falsely categorize the shuffled songs as positive samples. Therefore, we included this shuffled set to motivate extra attentiveness to the structure of song lyrics. Finally, we included a small portion of shuffled Reddit submissions and reviews to maintain consistency since we included shuffled songs as well.

2.2 Pre-Processing

Once we had collected our raw data, we had quickly noticed that we would have to consider multiple things. How would we handle empty samples, non-English samples, emojis, internet language, and certain tags that are characteristic to only the song lyrics set?

Within our raw dataset, we noticed samples from the non-song lyric set that do not have any text. From there, we had to decide whether or not to leave said empty samples in the dataset. For the sake of training our model to label empty text as non-song lyrics, we left the empty samples in the dataset. It is reasonable to classify these as such since the definition of song lyrics are words that are sung along to a specific melody. The lack of words should not be classified as song lyrics for the same reasons you would not ask for the lyrics for Für Elise. This was a design choice just so that the model would be able to classify empty inputs.

To try to reduce the number of non-English samples from our dataset, we settled on using a 30% threshold to determine whether a sample would be discarded from the dataset. If 30% or more of the sample were words that did not exist in the English dictionary, we threw the sample out. Since our raw song lyric data was predominantly English song lyrics, we definitely wanted to limit the scope of our predictive task to focus on English text samples. The reason why we had to use a threshold as opposed to just throwing out all samples that contained a single non-English word was because many song lyrics include slang words or shortened forms of words that are not technically English words, but we still wanted to keep those positive samples in our dataset since that is characteristic of many songs and it would be oversimplifying the predictive task. It is also characteristic of many other forms of English written text to include variations of words, acronyms, initialisms, slang, and, in the recent years, emojis that we, as a society, have collectively assigned meaning to such groupings of letter. Therefore, to create a generalized model that would be able to distinguish between song lyrics and non-song lyric text, we wanted to ensure our dataset included enough of these non-English words to be functional.

We also noticed many, but certainly not all, of the song lyric samples included tags within the text like, ”[Chorus]” and ”[Verse 1]”, to indicate sections of the song. It is obvious that the majority of the time these tags appear, the sample is a song’s lyrics. Therefore, the question we had to ask ourselves was, do we remove said tags since it might make the model a trivial predictor? More

precisely, we were worried that leaving the tags in may cause the model to look only for such tags within the text and predict that a chunk of text came from song lyrics if and only if it saw one of those tags. In the end, we decided to leave those tags in for the time being to see how the model performed with the tags left in (SPOILER ALERT: we never ended up removing the tags from the positive sample set). We were thinking that there were enough positive samples that didn't contain song tags that it would not become an issue for a model and we already intended to build a more robust model that would be able to handle these complexities. Also, since it is very common for song lyrics to contain tags, it might in fact be a good indicator to the model that a set of text is a song's lyrics if it contains tags.

The last pre-processing decision we made on our dataset was to randomly shuffle the words a small subset of samples from the data that we had obtained from each of the other sources. We took 4,000 song lyric samples, 1,000 Goodreads reviews, 1,000 Reddit posts, and 1,000 Steam reviews to build some more negative samples that would give our model some more complexities since we wanted to train it to identify the difference between the original song lyrics and an unordered collection of the same words that appear in song lyrics. That is also why we included a larger sample of song lyrics within our set of shuffled samples. We were thinking that the model would have more of a difficulty differentiating between the scrambled and unscrambled song lyrics than differentiating between scrambled negative samples and unscrambled song lyrics, so we wanted to give the model more scrambled song lyric samples to work with.

2.3 Exploration

Now that we have collected and cleaned our dataset, it is time to unpack the features of our data. To have a comprehensive understanding of our our data looks, we visualize specific characteristics of the text data using various tables and graphs. We will explore the most common words and bigrams that appear, the most common words that appear in only the non-song lyric text and not song lyrics, the distribution of the number of newline characters, and the distribution of text length within the dataset.

Since we found that there was much overlap in the most common words that appear in both lyrics and non-song lyric text, we also analyzed the most common words that appear in non-song lyric text, but do not appear in song lyrics as noted in Table 6. Taking a deeper look at these words, we observe that many of these non-lyric words hold unique qualities that indicate this set must be representative of non-lyric text. First, we see many acronyms in this dataset like "lmao", "nsfw", "tldr". The absence of these acronyms in lyrics can be attributed to the fact that they are not true words with phonemes, hence would not fit in a melodic setting. Additionally, our negative set consists entirely of text examples scraped from websites, so it makes sense that our dataset contains these acronyms intended to communicate phrases in fewer characters in text format. Furthermore, we also observe several names in this dataset, including "mallory", "ashlyn", "becca". We presume this is because our negative consists of several story-telling Subreddits, so some of the text may reference specific names in each submission. Also, the reviews from Goodreads and Steam may feature discussion on characters of novels or videogames, which potentially explains the frequent presence of these names in our negative dataset. Finally, we can also see use of modern terms that have grown in popularity in recent years like "ghosting" and "redditors". Since these new words have established their place in modern language by propagating through the internet, it makes sense that these words would show up in our non-lyric text dominated by internet content. We also note the frequency of sequences of 0's and 1's in our non-lyric text. These sequences of 8 binary values represent characters and are frequently seen in internet contexts.

We also analyzed the differences between the most common unigrams and bigrams in each half of the dataset. Figure 6 and Figure 7 display the most common 100 words within song lyrics and non-song lyric text and the words' overall frequencies within each sample set. The majority of the words within both common word sets are stop words and the overlap between the sets is large. Meaning, within the most common words, there are not many distinct words that are characteristic of either half of the dataset. Certain notable differences between the two are that the most common words in song lyrics contain "verse" and "chorus" which, as we discussed earlier in the previous

section about pre-processing, these two words are highly characteristic of the song lyrics data. Other words like ”love”, ”time”, and ”life” are all words that appear very frequently within song lyrics but not as frequently in non-song lyric text. Figure 8 and Figure 9, showing the 100 most common bigrams within song lyrics and non-song lyric text, don't contribute much more insight to the nature of the dataset. The high frequency of stop words and pronouns in unigrams persists to bigrams and do not provide much more insight to the dataset. Figure 10 and Figure 11, the 101th-200th most common bigrams in each dataset, offer a better idea of the contents of each text set. We can see that song lyrics frequently contain ”i love”, ”love you”, ”my heart”, and ”in love” which makes sense since a large subset of songs are about love, while the non-song lyric text contain a lot of ”she had”, ”he said”, ”she said”, ”with her”, and ”as he” since within many of the Reddit samples, people discuss the interactions between individuals. As a generalization, we can conclude that song lyrics tend to discuss abstract ideas like emotions and ideas while non-song lyric text more frequently talks about concrete actions and items.

Table 3: Distribution of Newline Character Statistics. This table highlights the mean, median, mode, and standard deviation of our song lyric and non-song lyric text samples.

Set	Mean	Med.	Mode	Std. Dev
Song	89.805	52	47	369.505
Non-Song	19.058	5	1	45.938

To explore other variations between the song lyrics and non-song lyric text, we compared the distributions of newline characters within both sets. We visualized these two distributions in Figure 12 and Figure 13, plotting the logarithm of the frequency of newline characters. As the figures would indicate, the number of newlines in a song's lyrics are generally greater than the number of newlines in non-song lyric text. We can also clearly see, the distribution of the newline characters' frequency is also much more concentrated for song lyrics than non-song lyric text. This is largely attributed to the inherent structure of song lyrics. Typically, songs have many short lines of text that need to fit within the given song's short time frame. Since the majority of songs are around 3 to 4 minutes in length, the number of newline characters is limited by the speed at which the

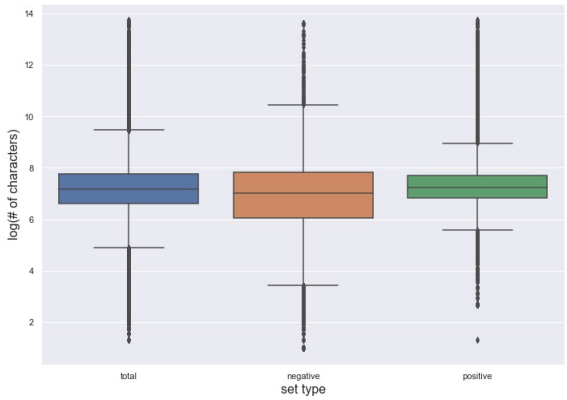

Figure 1: Distribution of Text Length in Each Dataset

artists can speak. The average statistics (Table 3) clearly show that song lyrics have significantly more new line characters than non-song lyric text. The song lyrics' large standard deviation is likely due to the large amount of outliers that exist within the set of song lyrics. Since our data was collected from open sources it is very difficult to control for fake samples. However, it is still important to recognize that the majority of the song lyric samples have a much smaller spread than the spread of the non-song lyric text samples. We also visualized the distribution of the length of each body of text as depicted in Figure 1. This visualization was used to verify the distribution of our personalized non-song lyric text dataset. As mentioned in the dataset section, we wanted to collect text samples of varying length that resemble the distribution of our song lyrics. Based on this visualization of the logarithmic transformation on the character length of each sample (Figure 1), we observe that the median of the negative set is only slightly below the median of the lyrics. The negative review set shows a greater distribution in length, given its much larger inner-quartile range, however, this can be explained by the variation in the length of our different non-lyric sources. While the Steam and Goodreads reviews were relatively short, some Subreddits like nosleep had an average length of 12453 characters which is well above the song lyric average of 5466 characters.

3 Predictive Task

This section will outline how we determined and implemented a predictive task. While all the data we collected is consistent in that each entry is a

large block of text, it is also separated into two distinct parts – song lyrics and non-song lyric text. We decided to center our predictive task around the binary nature of our dataset by focusing on a model which would predict whether or not large blocks of text are song lyrics. To accomplish this task and discern an optimal model, we had to build off of a basic baseline model by trying different approaches, fine tuning hyper-parameters, and experimenting with different methods of feature extraction.

3.1 Baseline

In order to create a naive implementation of the prediction, our initial intuition was to determine if a block of text identifies as song lyrics based on whether or not it contains words specific to either classification. In other words, if a block of text contains a word(s) that appears exclusively in non-song lyric texts, it might be reasonable to predict it as a non-song lyric block of text.

To implement this baseline, we first collected a set of unique words that appeared in all of the song lyrics in the training set. Next, we collected a set of unique words that appeared in all of the non-song texts in the training set. We then took the difference of the non-song set from the song set, which produced a set of words which appeared in the non-song set but *did not* appear in the song set.

Finally, for the prediction, we iterated through each block of text in the training set and predicted it to be a non-song if even a single word from the exclusive non-song set appeared in the text block. Otherwise, we predicted it to be a song. After implementing this model, we recorded an accuracy of 65.64299424184261%. This is above average – but clearly sub-optimal – score provided a robust starting point to build a complex model off of.

3.2 Evaluation Metrics

Our evaluation metric will simply be the accuracy score of the predictions, as this will be the most relevant metric when judging the completeness of our task and balanced error rate (BER). When thinking about what makes a good model, we agreed that the more correct often it predicts correct labels, the better the model. Thus, we settled on finding the best model through the accuracy score, or in other words, the model whose score was closest to 1.00 (perfect accuracy) when using the equation:

$$Score = \frac{|yPredictions^{TP}|+|yPredictions^{TN}|}{|yPredictions|}$$

We also used balanced error rate as an additional metric to compare the success of different models. Since BER is an average of false positive and false negative rates, it reveals more about the success of a classifier, specifically in the case of imbalanced data sets. In our case, accuracy is a completely reasonable metric to judge our model given our training, testing, and validation sets each have roughly the same number of positive and negative samples. However, we decided to also incorporate BER to better compare our results in case it captured slight variations that accuracy did not reveal.

3.3 Feature Extraction and Model Choice

When choosing a model for this predictive task, we knew that choosing a text-feature based design was an absolute requirement. Since our dataset is comprised of exclusively blocks of texts and labels, we had to create a model which would be able to push out strong predictions based on the text alone. As we learned in class, the naive implementation of this idea, creating a bag of words vector, is both slow and inefficient, so this type of text based model was never considered. Additionally, we recognized the importance of bigrams and trigrams in our text, as certain groupings of words could go a long way in training our model to recognize songs. We decided to settle on using TF-IDF to build our feature vector. Using TF-IDF as our feature vectorizer would not only allow us to take advantage of a strong text-feature extractor, it would also allow us to fine tune important feature settings such as dictionary size and n-gram range.

After settling on a feature vectorizer, we considered which mathematical model to implement as our predictor. Logistic regression and linear support vector classification immediately stood out as two interesting predictors which could provide us with accurate predictions. Logistic regression seemed like a good fit for our task because of its binary nature, which mirrored our binary classifications. On the other hand, using a support vector classification also had its strengths - namely the built-in associated learning algorithms which would help us fine tune our accuracy. After weighing the advantages of both options, we decided to experiment with both predictors before finalizing a model.

Finally, we considered different means of extracting features from large blocks of text. In terms of tokenizing the text before passing it into the TD-IDF vectorizer, several approaches seemed worth experimenting with. Text tokenizations such as removing stop words, removing punctuation, stemming, removing emojis, and converting to all lowercase were all viable approaches which could help the TF-IDF vectorizer extract the most relevant features. Additionally, we planned to experiment with fine tuning our model's feature parameters so that it could consider different dictionary sizes (ranging from 5000 to 25000) as well as different combinations of n-gram ranges (lower/upper bounds ranging from 1-3).

4 Model

After much consideration, we decided to implement several different models before fixating on a singular optimal model. We divided the task of finalizing a model into several stages: finding the best feature settings, tuning hyperparameters on a model with these best feature settings, and finally combining the tuned model with SVC and Logistic Regression. After executing all of the listed stages, we would be left with the single most accurately tuned predictive model. Although we decided to keep many different variations of this model in play, one factor was consistent throughout – using a TF-IDF Vectorizer. The decision to use the TF-IDF Vectorizer can be easily justified because of how powerful it is in extracting key details from blocks of text, all while considering different n-gram ranges and dictionary sizes.

4.1 Architecture

Our model utilizes several different approaches to ensure optimization. Firstly, the decision to use TF-IDF is already a huge optimization over other popular approaches to text feature extraction. Unlike a naive implementation which may use a bag-of-words approach to extract features, TF-IDF is able to create feature vectors from blocks of text with more accuracy and efficiency. TF-IDF is unique in its functionality because it assigns weights and importance to the most common words that appear in the training set's corpus. This distinction from the bag-of-words approach is important because it greatly improves prediction accuracy. In terms of efficiency, TF-IDF is able to prioritize words and compact the already smaller feature vectors into sparse-matrices. Its ability to do this greatly improves the efficiency and speed of the model as opposed to other popular approaches.

Secondly, because of our extensive and diverse data collection, we were able to train the model on large volumes of data, with the training set containing 63,206 entries. We ensure that the split of song lyrics and non-song text within these 63,206 entries is nearly even, which means the training on the model is extremely balanced.

Additionally, we will optimize our model(s) further in the second stage of our model-selection process. As described above, the second stage will revolve around tuning the hyperparameters on the model after its best feature settings have been discovered. This is a crucial optimization because we will be specifically experimenting with regularization to increase the model's resistance to overfitting as well as other types of fine tuning.

Some issues we ran into when considering different constructions of the architecture of our model predominantly revolved around scalability. We found that trying to create a predictor which considered quadgrams (n-grams with a range up to 4 words), the computer's RAM would overload and the model would crash. This would also happen when trying to create a feature vector with over 40,000 entries. Crashes like these were definitely expected as we tried to push the boundaries of the model's scalability. While more features and n-grams definitely increased the scope of the model's learning, it became quickly apparent that there was a limit to this scope.

4.2 Model Trials

Our baseline model proved to be the most unsuccessful attempt along the way due to several factors. Not only did the model have sub-optimal accuracy, it also had an incredibly BER of over 34%. More importantly, it was clear that we had reached the ceiling in terms of performance when using this approach. The baseline's approach, which predicted labels whether or not the test set text contained words exclusive to non-song text, relied too heavily on the words that showed up in the training set. Additionally, it also required extreme similarity between the train and test set in order to work, which was not a realistic requirement when building a generalized prediction model. If the

test set contained no words that appeared in the training set, the model would predict 'song lyric' for every single entry, which was extremely problematic. After realizing that the baseline could no longer be optimized further, we pivoted to a different type of text-feature extraction while still being sure to retain the overall idea of text analysis through word composition – TF-IDF.

We decided to consider several different types of models for the model that came out of the first two stages of our model-selection process. Once we find the best feature settings and fine tune the hyperparameters, we decided to compare the results on several predictors such as the baseline, logistic regression and support vector classification.

Each of these model types have key strengths and weaknesses. For the baseline model, the weaknesses definitely outweighed the strengths (as discussed above). However, it was a valuable model for our process because it gave us a general foundation on how we wished to approach this predictive task – by analyzing word patterns within the text. When considering the support vector classifier, a key strength that stood out was its ability to be more generalized in its prediction pattern. This attribute is immensely valuable because it ensures that an SVC-based model would be significantly more resistant to overfitting. On the other hand, there was an argument to be made against SVC's, as they generally take longer to run as opposed to predictors that use linear regression. This could prove to be an issue for us in terms of scalability, as we are dealing with a large dataset to train on. Finally, using logistic regression for our predictor definitely had appeal because of how straight-forward and easy to understand it is. Unfortunately, its speed and readability are opposed by crucial weaknesses such as a lack of the precision and associated learning algorithms that support vector classifiers provide.

After considering all of these factors, we were convinced that our final model would be one which identified the best feature settings, fine tuned the hyperparameters maximally, and chose the best predictor between a linear regressor and support vector classifier.

5 Literature

The nature of our predictive task necessitated that we conduct research about where and how we would obtain our dataset, how would we study our data in a meaningful way, and which cutting edge methods have been shows to perform well on similar data and for similar tasks. We will break down all of our references in this section that will explain much of our model's design.

5.1 Existing Dataset

As detailed in our Data Collection subsection, we compiled our own dataset. One of the existing datasets we did use, a Kaggle dataset, was used only as a comprehensive list of songs to use for our query of song lyrics [Ay, 2020]. The original Kaggle dataset included song data like artists, song duration, release year, danceability, acousticness, tempo, popularity, and song key. Many other individuals used this data for other predictive tasks like predicting a song's popularity, genre, or release year. This paper also uses two datasets from Professor McAuley's database, each containing review data from Goodreads ([Kang and McAuley, 2018], [Pathak et al., 2017],[Wan and McAuley, 2018]) and Steam [Wan and McAuley, 2018]. Both datasets were used to extract review text to add greater variation to our non-lyric samples.

5.2 Similar Datasets

Other similar datasets have compiled song lyrics to be used for song lyric generation. This other Kaggle dataset, grouping the works of many songs by artist and poems by poet, has been used to generate song lyrics for specific artists [Mooney, 2018]. For example, Amisha Jodhani used the dataset to generate songs that mimic the characteristics of Bruno Mars's songs using a neural network [Jodhani, 2020]. This dataset would have been insufficient for our use, however, since the sample size is relatively small, and we wanted to include a wide variety of negative, non-song lyric text samples, not just poetry.

Another individual also analyzed a song lyric dataset in nearly way, shape, and form and wrote about his/her findings in a Kaggle discussion post [Kratisaxena, 2018]. He/she details the distributions of number of songs by artist, the song lyric length, and song title length. He/she even dives deeper and explores the relationship between song title length and song length, the words used to express different sentiments within songs, the most common words used by a specific artist, and also any common rhythmic words. Although the author of this piece explores all aspects of song

lyrics, the techniques he/she uses to analyze the data are fairly standard. The use of frequency tables and n-grams is fairly consistent with other studies of text datasets.

5.3 Prior Research

For this binary text classification problem, our results and research both pointed to the success of Support Vector Classifiers (SVC). The 1998 research paper titled, "Text Categorization with Support Vector Machines: Learning with Many Relevant Features" by Thorsten Joachims, identifies the benefits of using SVMs [Joachims, 1998]. At the time this paper was released, SVMs were a particularly new classification method, however they continue to persist as a viable option for text classification due to their robustness in high-dimensional spaces. According to this paper, this success in high-dimensional spaces is explained by their ability to learn "independent of the dimensionality of the features" by biasing towards a margin that can partition our data. This quality is perfect for text classification given our feature representations are usually thousands of words long where each feature corresponds to some important word in our corpus. Based on the results of this paper, where SVM outperformed every other classification text method, including Naive Bayes and K-Nearest Neighbors, SVM was a clear choice for the backbone of our classification model.

5.4 State-of-the-Art Methods

In recent years, we've observed the exponential growth in the implementation of neural networks for these text categorization problems. According to PapersWithCode, the best model in 2019 for text categorization of question topics on the website, Yahoo! Answers, leverages a pretrained BERT to achieve 77.62% accuracy [pap, 2019]. The paper associated with the model details the process of fine-tuning the architecture of BERT, which stands for bidirectional encoder representations from transformers [Sun et al., 2019]. Typically, BERT is used for textual entailment tasks to predict the next word in a sequence, but this paper was able to adapt BERT to solve a categorization task. After hyperparameter turning, preprocessing of the Yahoo! Answers dataset using pretrained WordPiece embeddings, and evaluating the effectiveness of features in each layer, this model was able to achieve state of the art results. This optimal model was able to reduce the error rate of other modern neural network solutions by 18.57% on average.

While our task attempts to predict whether a body of text is from a song, we have also seen recurrent neural networks perform an opposite task of generating lyrics. In this Github blog post, we see the implementation of a character-by-character recurrent network where each subsequent character is determined by simply sampling the probability distribution (softmax) of the LSTM layer's output [Pakhomov, 2018]. This blog post also experiments with conditioning the network by artist, prompting the network to generate lyrics in accordance to a particular artist's style. The categorization of lyrics according to artist could also be an extension of our current predictive task that could be explored in the future. In addition, we could also rate the success of this generative model by evaluating how well it could trick our current network, which is a fundamental concept for training generative adversarial networks.

Beyond neural networks, we have also seen attempts to solve this categorization problem through hierarchical classification. According to the paper, "Hierarchical Text Classification and Evaluation," modern solutions only consider categories of text as isolated entities, as opposed to a natural hierarchy where a document can be classified into subcategories down the tree until it reaches a leaf category [Sun and Lim, 2001]. This paper also extends standard precision and recall error metrics to penalize methods that incorrectly classify documents into categories similar to the target less than an incorrect classification of that document into a very distant category. In order to classify a document, this method has binary classifiers at each category to determine whether the document should belong to the corresponding category, or if it should be given to a classifier in its subcategories. This method proved to be reasonably successful based on the personalized error metrics. However, the paper also suggests this novel solution has much room for improvement since it does not take full advantage of hierarchical properties and suffers from irrecoverable errors made by parent-category classifications.

5.5 Similar Papers

While we were unable to find a paper with an identical classification task, a Stanford paper, "Using Song Lyrics and Frequency to Predict Genre"

explores a similar task of categorizing rap vs. non-rap lyrics [Ram, 2017]. To generate its feature vector, this method simply used 500 words that had a frequency of 1000 to 3000 occurrences in the training set and counted the frequency of each word for each sample. This experiment achieved similar results with 93.2% accuracy in rap lyric classification using SVM, also outperforming naive bayes and regression models. Our model slightly outperforms their model, which can be attributed to our use of TF-IDF vectorization as well as the comparitive difficulty of our classification task. Overall, we share several core steps of feature extraction through words and underlying classification models, which explains the great similarity in results in our similar categorization tasks.

6 Results

We found that the model with the best test accuracy was the model using stemming on the input text, unigrams, bigrams, and trigrams and a feature size of 25,000 for the TF-IDF Vectorization, and the feeding the feature vector generated by the TF-IDF into an SVC model.

6.1 The Numbers

We identified that the best performing model for our dataset and specific predictive task was an SVC model that used TF-IDF with n-gram range [1, 3], feature size 25,000, and stemming the input text to build the feature vector. This model was able to differentiate song lyrics from non-song lyric text with an accuracy of 0.97203 and a BER of 0.02810. With the best performing model highlighted in green, Figure 2, Figure 3, Figure 4, and Figure 5 show the results of various combinations of different models, input processing, and feature design. The figures only show a subset of the results we obtained. Since we wanted to obtain the highest performing combination of different feature optimizations, we performed a grid search over 84 different models while making single variable changes to n-gram range, dictionary size, and input text processing in each iteration. We have condensed the results of the grid search to focus on the key trends.

6.2 Feature Reflection

Some notable trends within our results is that overall, a larger range of n-grams and a larger dictio-

Feature Variations	N-gram Range	Dictionary Size	Test Accuracy
original (no extra processing of inputs)	[1, 1]	5000	0.9151357279956128
remove stop words	[1, 1]	5000	0.9145873320537428
remove punctuation	[1, 1]	5000	0.9158212229229503
stemming	[1, 1]	5000	0.9141760350973402
remove emojis	[1, 1]	5000	0.9152728269810804
convert to lowercase	[1, 1]	5000	0.9151357279956128
remove stop words, punctuation, and emojis, stemming, and convert to lowercase	[1, 1]	5000	0.9123937482862626

Figure 2: Test Accuracies of SVC Model with Various Input Processing

Feature Variations	N-gram Range	Dictionary Size	Test Accuracy
original (no extra processing of inputs)	[1, 3]	25000	0.9714834110227585
remove stop words	[1, 3]	25000	0.9317247052371812
remove punctuation	[1, 3]	25000	0.9713463120372909
stemming	[1, 3]	25000	0.9720318069646284
remove emojis	[1, 3]	25000	0.9716205100082259
convert to lowercase	[1, 3]	25000	0.9714834110227585
remove stop words, punctuation, and emojis, stemming, and convert to lowercase	[1, 3]	25000	0.9422813271181794

Figure 3: Test Accuracies of SVC Model with Various Input Processing

Model Type	Features Type	N-gram Range	Dictionary Size	Test Accuracy
SVC	stemming	[1, 1]	5000	0.9141760350973402
SVC	stemming	[1, 1]	25000	0.9159583219084179
SVC	stemming	[1, 2]	5000	0.9621606800109679
SVC	stemming	[1, 2]	25000	0.9707979160954209
SVC	stemming	[1, 3]	5000	0.9636687688511105
SVC	stemming	[1, 3]	25000	0.9720318069646284
SVC	stemming	[2, 2]	5000	0.9439265149437894
SVC	stemming	[2, 2]	25000	0.9642171647929806
SVC	stemming	[2, 3]	5000	0.9433781190019194
SVC	stemming	[2, 3]	25000	0.9628461749383055
SVC	stemming	[3, 3]	5000	0.8925143953934741
SVC	stemming	[3, 3]	25000	0.9284343295859611

Figure 4: Test Accuracies of SVC Model with Various Features

Model Type	Feature Type	N-gram Range	Dictionary Size	Test Accuracy	Test BER
baseline	positive negative set comparison	N/A	N/A	0.656429942 4184261	0.347771226 18186473
logistic regression	stemming	[1, 3]	25000	0.960652591 1708254	0.039565171 19172476
SVC	stemming	[1, 3]	25000	0.972031806 964628	0.028096151 715940598
SVC	stemming and newline character count	[1, 3]	25000	0.970935015 0808884	0.029237853 51338759

Figure 5: Test Accuracies and Test BERs of Various Models Holding Features Constant

nary size performed better than unigrams and a small dictionary size. This is significant because, as intuition would tell us, song lyrics are highly structured forms of text that rely on specific repetitions of sounds and meter to create the lyrical flow that songs contain. This order is not maintained if we only inspect the contents of each input text using unigrams. With unigrams, the model would not be able to differentiate between song lyrics and the scrambled versions of song lyrics. A large dictionary size is also important because we want to include n-grams that potentially could appear in the dataset at a low frequency but could also be highly representative of either a song lyric or non-song lyric text as we had explored in the data exploration section. Also, since we are using a range of n-grams, this increases the number of unique values in our dictionary. Naturally, the size of our dictionary must increase to accommodate the complexities of the data.

We also found that out of the various different methods we used to process the input text, stemming the text performed marginally better than other variations while holding all other feature parameters constant. This is probably an effect of the fact that stemming is able to compress the information within the text and therefore allowed us to pass a more dense, informative feature vector to the model. Stemming text will group different words with the same stem as one value in the dictionary.

Removing stop words worked the least well out of all the variants by far when the n-gram range was [1, 3] and the dictionary size was 25,000. If we look at Figure 6 and Figure 7, we can see that the stop words appear in song lyrics at a significantly higher frequency than non-song lyric text. The frequency at which stop words occur in song lyrics is an order of magnitude greater than the frequency in non-song lyric text. This characteristic of the dataset is likely captured by the n-grams and removing stop words is removing a meaning indicator that the model uses to make its prediction.

Table 4: Accuracy of hyperparameter experiments

	Regularization (C)			
Loss Function	0.01	0.1	1	10
hinge	0.898	0.956	0.973	0.970
squared hinge	0.931	0.968	0.974	0.971

Our models all employed TF-IDF to build the feature vector. After some thorough research and experimentation, we decided to use TF-IDF to build our feature vector because it would both allow us to leverage the vast array of words that we discovered appear in the non-song lyric dataset at relatively low frequencies as well as allow us to make our model resistant to a large range of input text length. The TF-IDF models performed nicely.

6.3 Model Parameters

To tune our model even further, we also experimented with different combinations of hyperparameters, regularization and the loss function. We ran a 2-dimensional grid search on various combinations of regularization constants and loss functions. We found that the regularization parameter C=1 performs the best on the validation set. This makes sense because tuning the regularization parameter is a trade off between giving model complexity importance in our loss function to reduce overfitting and also giving enough importance to the actual loss of our predictions to learn an accurate model. Hence, we found the perfect sweet spot of just enough regularization without a major compromise on the model's classification ability.

In each of our experiments, squared hinge loss also performed better than hinge loss. This can be explained by the mechanics of squared loss which penalizes predictions with respect to their distance from the target label. Hence, predictions that are very incorrect are penalized more than only slightly incorrect predictions. This allows the model to converge to a solution that is sensitive to outliers, which would be highly applicable for our dataset with a large variation in textual features.

6.4 Comparative Analysis

Based on our results, SVC performed better than using a Logistic Regression classifier. This comes as no surprise since SVC is an implementation of a Support Vector Machine(SVM) classifier, and SVM is known to perform better for this particular classification task. As we mentioned in our literature section above, SVMs simply optimize a margin to partition samples of different classes. Since SVM uses this geometrical approach, as opposed to Logistic Regression which utilizes a probabilistic approach, it performs much better in high-dimensional spaces, such as our sample text feature space. In addition, SVMs are known for generalizing better since they do not penalize correct predictions if they are made with reasonable confidence due to its sole objective of partitioning our categories with the greatest margin. In this case, the characteristics of our features, in this case, the high-dimensional, sparse feature space, are most suitable for an SVM classifier.

7 Conclusion

7.1 Further Exploration

While our trained SVC model is able to categorize songs lyrics and non-song lyric text with high accuracy, there is still room for improvement. What could make the model more robust is to have an even wider variety of non-song lyric text within our negative sample. There is a lack of extremely long samples within our dataset. We were also limited to informal written pieces as opposed to published works. Adding such samples could make the model even more generalized.

Our model's complexity was also limited by our computing power, as using a larger dictionary size and a larger range of n-grams kept crashing our systems. We believe that with the ability to expand the size of the our feature vectors, the model would be able to capture even more nuances between the difference between the song lyric set and the non-song lyric set.

7.2 Potential Applications

Our original idea for designing this predictive system was to use it as a method for evaluating an Recurrent Neural Network, or RNN, that was trained to generate song lyrics. Our predictive model could evaluate the strength of the RNN based on how many generated samples were able to be labeled by our SVC model as song lyrics. This would provide a way to train such an RNN programmatically.

References

2019. Papers with code - yahoo! answers benchmark (text classification).

Yamac Eren Ay. 2020. Spotify dataset 1921-2020, 160k+ tracks.

Thorsten Joachims. 1998. Text categorization with support vector machines: Learning with many relevant features. *Machine Learning: ECML-98 Lecture Notes in Computer Science*, page 137–142.

Amisha Jodhani. 2020. Bruno mars song generator.

Wang-Cheng Kang and Julian McAuley. 2018. Self-attentive sequential recommendation.

Kratisaxena. 2018. Let's analyze the song lyrics!

Paul Mooney. 2018. Song lyrics.

Daniil Pakhomov. 2018. Learning to generate lyrics and music with recurrent neural networks.

Apurva Pathak, Kshitiz Gupta, and Julian McAuley. 2017. Generating and personalizing bundle recommendations on steam.

Jayen Ram. 2017. Using song lyrics and frequency to predict genre.

Aixin Sun and Ee-Peng Lim. 2001. Hierarchical text classification and evaluation. *Proceedings 2001 IEEE International Conference on Data Mining*.

Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. 2019. How to fine-tune bert for text classification? *Lecture Notes in Computer Science Chinese Computational Linguistics*, page 194–206.

Mengting Wan and Julian McAuley. 2018. Item recommendation on monotonic behavior chains.

Table 5: Subreddits Scraped to Generate Negative Set. The count indicated the number of submissions used in our dataset and average length represents the average character count of each Reddit submission.

Name	Count	Average Length
copypasta	999	1272.0
tifu	997	1998.2
amitheasshole	996	6453.2
poetry	989	900.5
nosleep	997	12451.3
relationships	995	2419.6
prorevenge	897	6646.1
pettyrevenge	995	2115.6
maliciouscompliance	953	2708.3
stories	986	3591.9
lifeofnorman	973	1430.5
hfy	1000	10579.8
talesfromtechsupport	996	2692.0
talesfromthepizzaguy	998	1113.3
dating_advice	980	1184.0
unresolvedmysteries	996	5480.9
jacksonwrites	552	5912.3
idontworkherelady	996	2257.7
confessions	988	807.0
justnomil	996	2773.9
letsnotmeet	996	4453.1
poetry_critics	997	853.7
writing	933	673.5
askliterarystudies	998	586.8
filmtheory	655	436.0
needafriend	995	570.2
internetparents	996	1114.2

Table 6: Most Common Words That Appear in the Negative Set But Not the Positive Set

.	antti
goodreads	mallory
01100100	worldbuilding
nodrak	01110011
gef	human’s
niseph	redditors
vrynn	tfos
munin	jnmom
arcium	minzo
janson	antoth
aarmoo	khorwin
embrel	nsfw
fdh	01110010
xana	ect
netgalley	shil’vati
mantid	suika
drakus	telkan
raegn	sookie
atarah	ke’tet
lmao	avalro
wibta	fmil
sylnya	nikolen
rachuck	ernestine
01101000	01101001
01100001	tenna
borlian	wigglewiggle
niragi	ghosted
yirphayen	vulzan
nunuk	siers
delcan	shaar
01101111	01101110
lanaktallan	troubleshooting
isp	ashlyn
caelnaste	becca
jackalope	httpstoomanybooksnotenoughshelves
tifu	01110100
elvar	caden
adf	vuxten
hadasaphorus	tholmor
jakim	fimbilvr
senon	norman’s
lafalla	jnm
ankiah	01100101
reglen	bil
jamet	terrans
srettia	razer
troma	jnmil
00100000	draevin
aita	nbspnbspnbspnbspnbsp
tldr	x200b

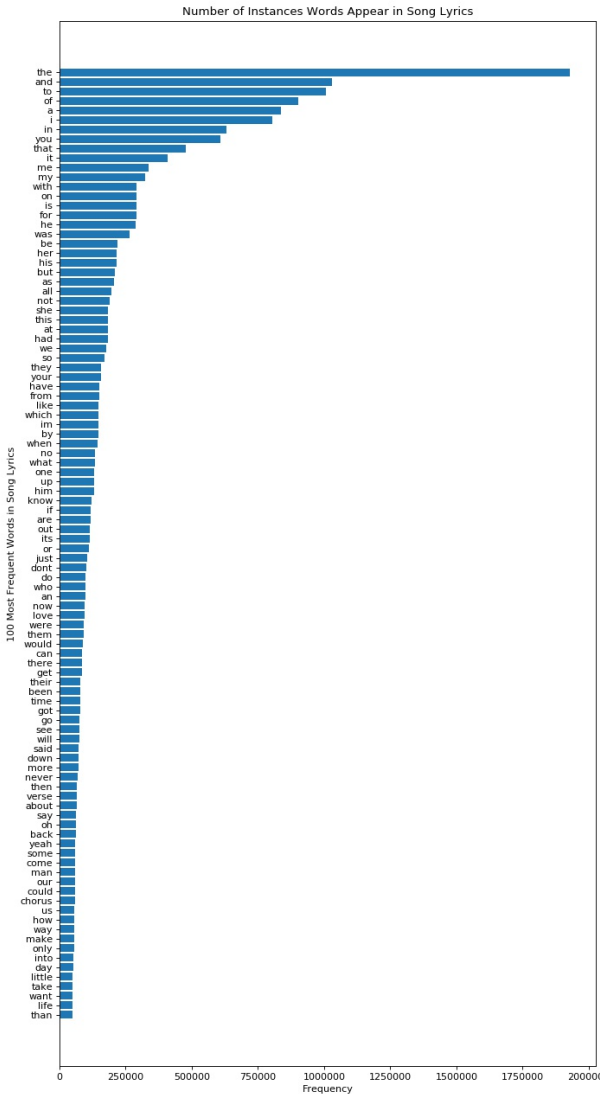

Figure 6: Number of Instances Words Appear in Song Lyrics

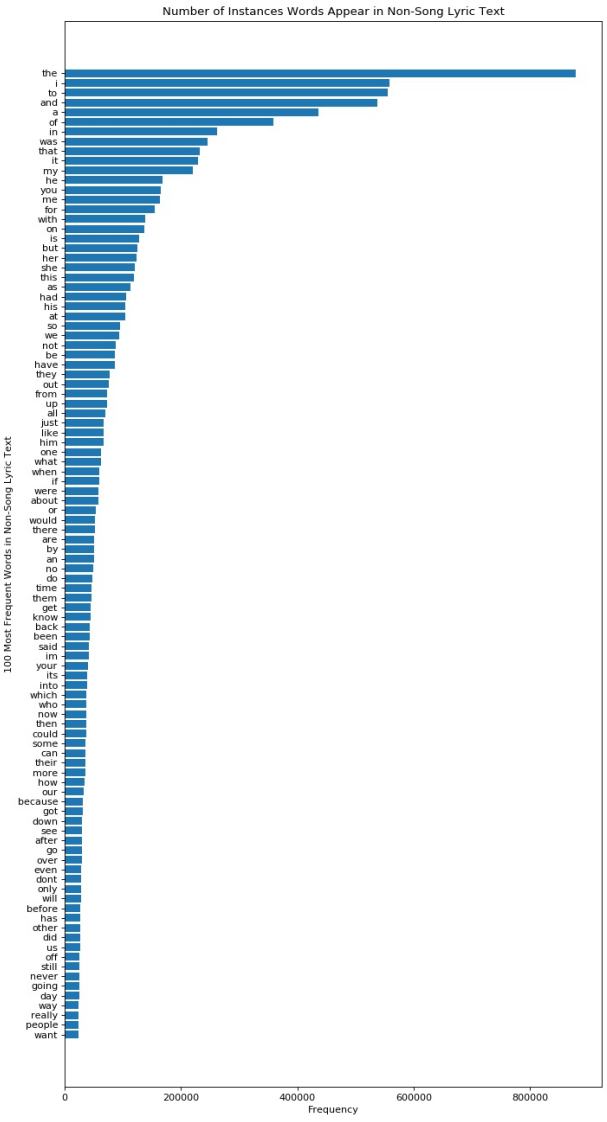

Figure 7: Number of Instances Words Appear in Non-Song Lyric Text

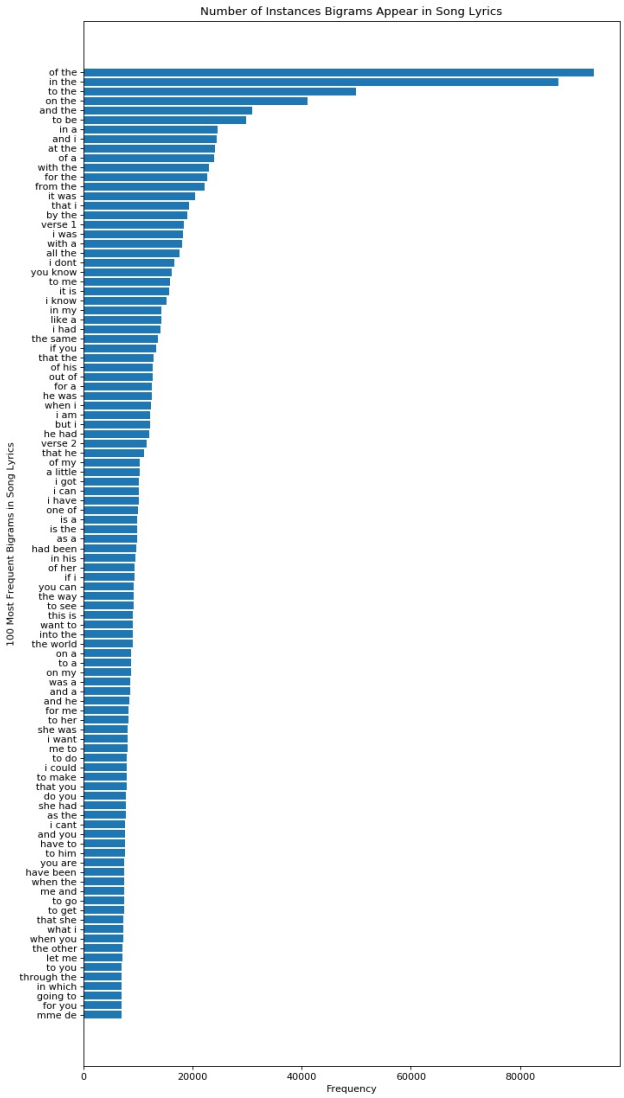

Figure 8: Number of Instances Bigrams Appear in Song Lyrics

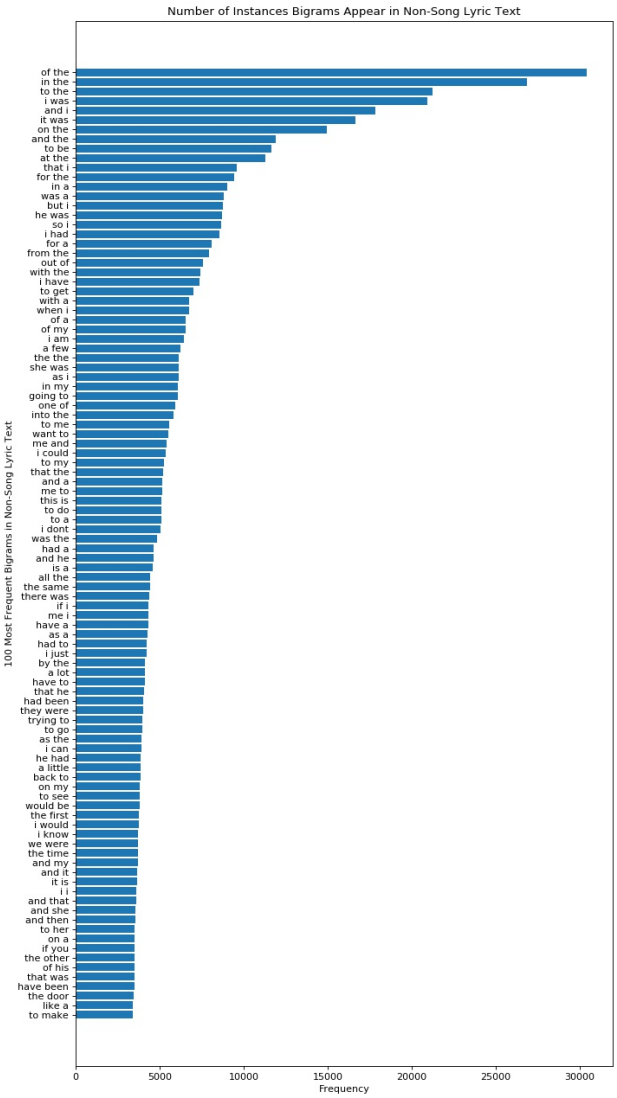

Figure 9: Number of Instances Bigrams Appear in Non-Song Lyric Text

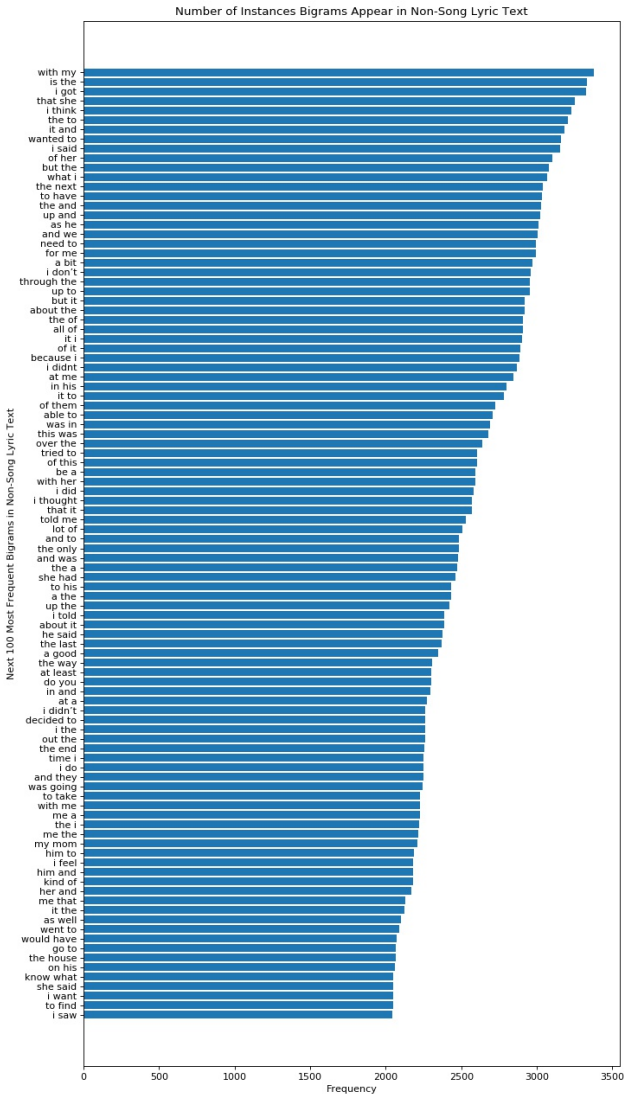

Figure 10: Frequency of 101th-200th Most Popular Bigrams in Non-Song Lyric Text

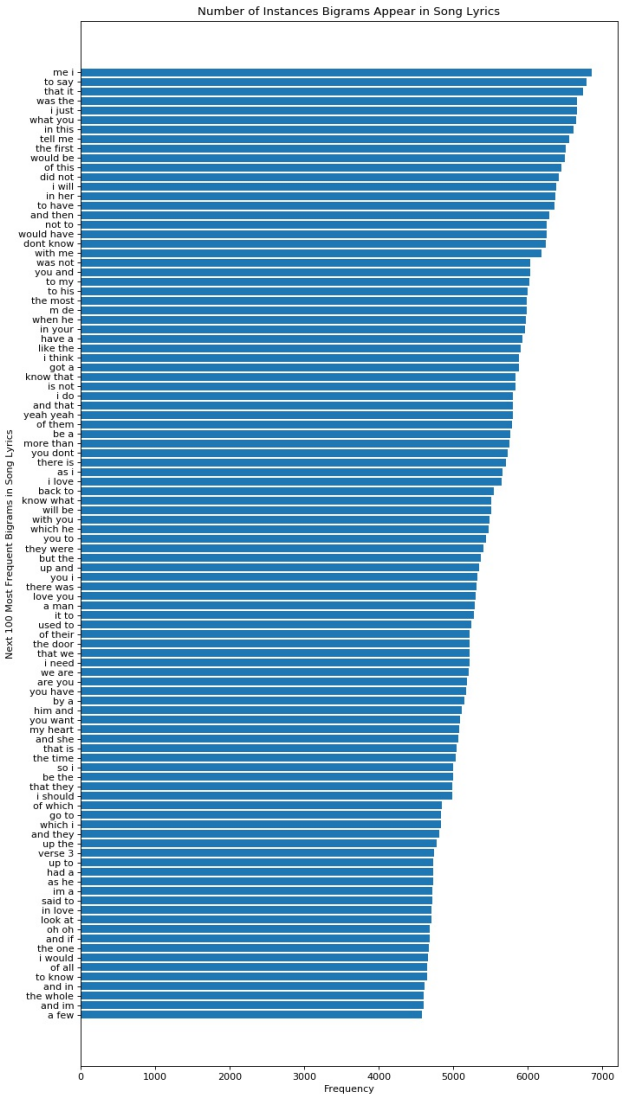

Figure 11: Frequency of 101th-200th Most Popular Bigrams in Song Lyrics

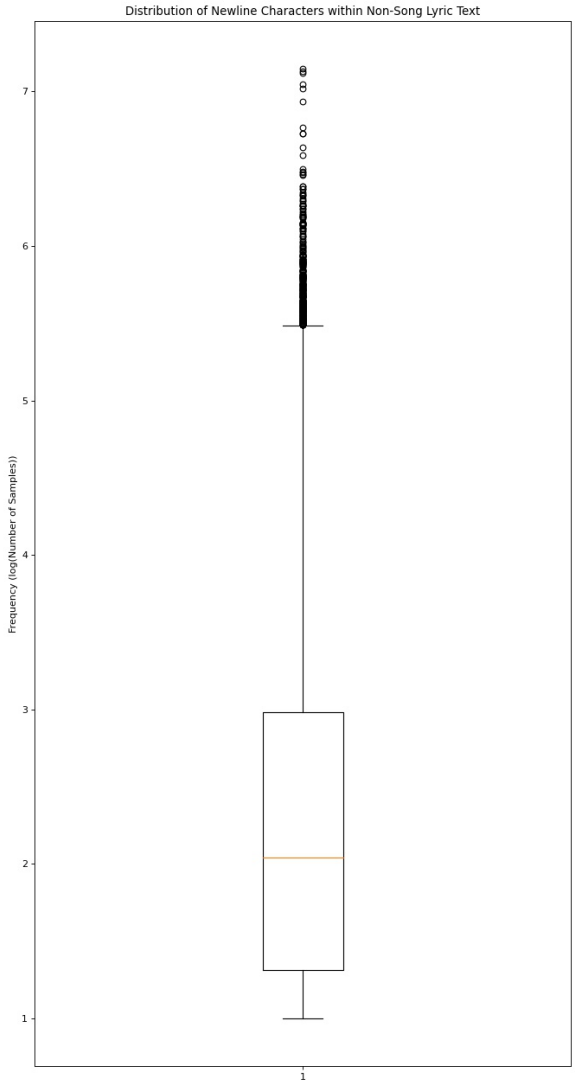

Figure 12: Distribution of Newline Characters in Non-Song Lyric Text

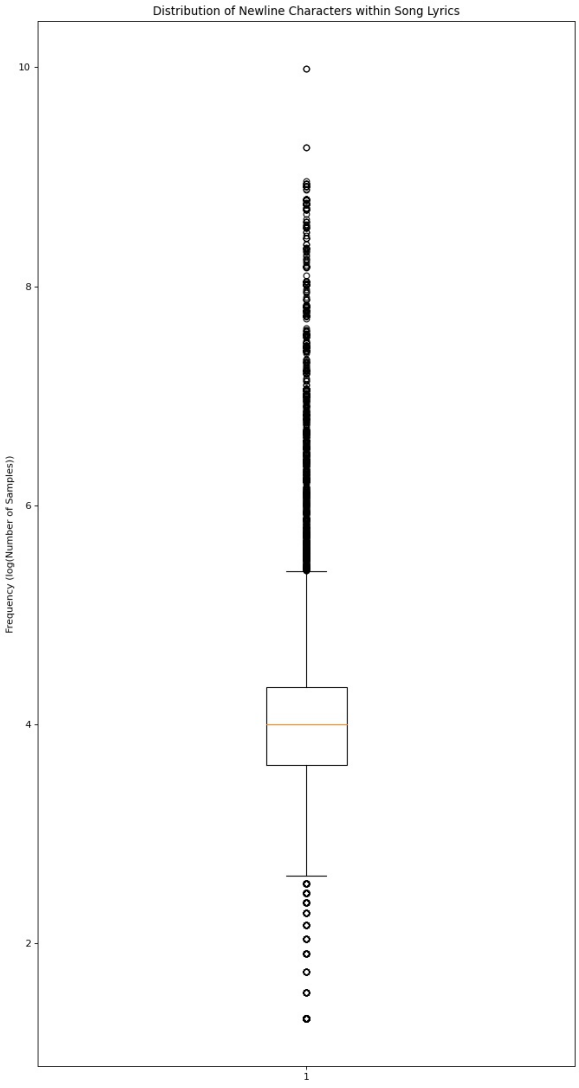

Figure 13: Distribution of Newline Characters in Song Lyrics