Predicting the Decade of Songs With Lyrical Content

Springboard Capstone I Final Report

Introduction:

What makes decades unique musically? One might say "Obviously the instruments." One may also say that some decades were defined by their popular genres. Is it possible that lyrics could also define decades? The answer to that question was the primary objective of this project. While it may seem obvious that lyrical trends could change between decades, I wanted to quantify this idea by creating an algorithm that would classify what decade a song was from, based purely on its lyrical content. The data used for this problem is the lyrical information from the Billboard top 100 for each year from the years 1965-20015. This specific data was chosen because I was most interested in using popular music to solve this problem, and the Billboard top 100 seems like a good proxy for gauging the popularity of the music. I primarily chose this problem because when I submitted my project ideas for the capstone project, I asked my mentor, Misael Manjares, what project would teach me the most out of the ideas that I chose, and he said that this project would lead to me learning the most. I think that the most relevant application of this project is academic, studying some of the variations in word use in some of the most popular music over time. This information may also be useful for songwriters in terms of understanding what words are popular, what words may be increasing in popularity, and what words may be decreasing in popularity.

The Data:

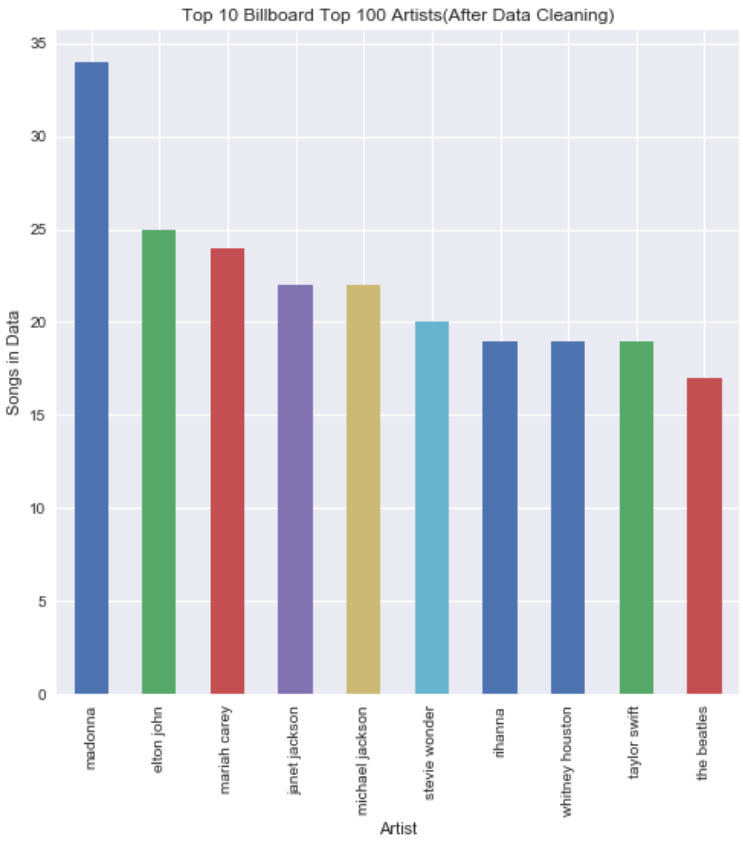
The data used for this project was obtained from a user generated Kaggle dataset that contained various information about each song, including:

- The title of the song
- The artist
- The year the song made the Billboard top 100
- The rank of the song in the Billboard yearly top 100
- The lyrics of the song

The user that generated this dataset used a scraping algorithm that checked three websites for the lyrics to the songs, so any songs that did not fit the patterns of the algorithm ended up not having lyrics in the dataset. One obvious example of this is the song "(I Can't Get No) Satisfaction)" by The Rolling Stones, which was ranked third in the top 100 of 1965, but did not have lyrics in this dataset because the formatting of the name did not fit with the scraping algorithm. Songs that were missing lyrics from this dataset were simply dropped for the sake of time and scope of the project. Instrumental songs were also necessarily dropped from the data, since they did not provide anything meaningful to the lyrics.

After dropping the aforementioned obvious data, the process for cleaning up the remaining meaningful lyrical content began. Initially, there were some obvious errors at the beginning of some lyrics where artist names and genres were included once or multiple times

before the actual lyrics for the song began. Since these were common obvious errors I was able to write some functions in order to clean the data and remove these errors as much as I could. After I managed to take care of these obvious issues, I proceeded to a few more general cleaning steps that are common in processing textual data. I made all of the lyrics in the dataset lowercase and removed anything from the data that was not an actual word, such as numbers. I then tokenized the lyrics for each song, which means I turned the lyrics of the songs into a list of words, and I removed stopwords from the data. Stopwords can be explained as words that happen frequently in text that tend not to convey meaningful information, these are words such as 'the' or 'an'. In addition to removing stopwords I also removed the words 'verse' and 'chorus' from the lyrics. While 'verse' and 'chorus' are not generally stopwords, I noticed that in many of the songs in the dataset those words were used purely for denoting the structure of the song as opposed to being part of the actual lyrics, so I chose to remove those words entirely from the dataset.



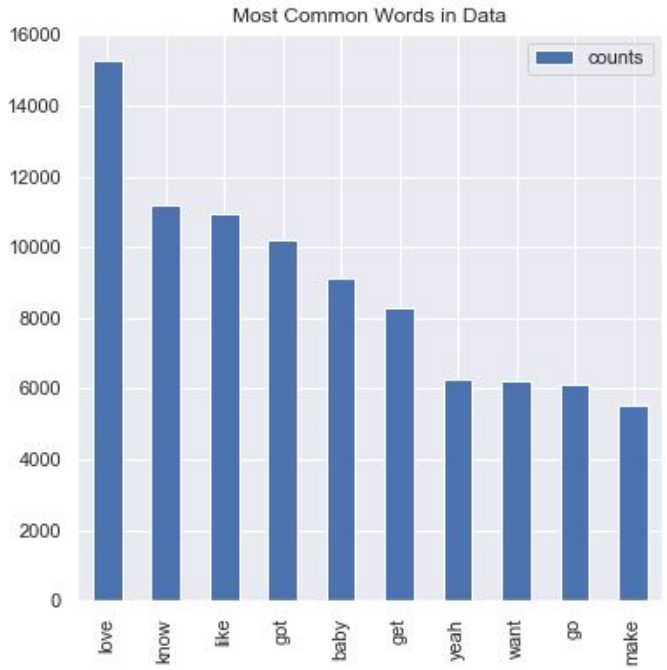
What Does the Data Say?:

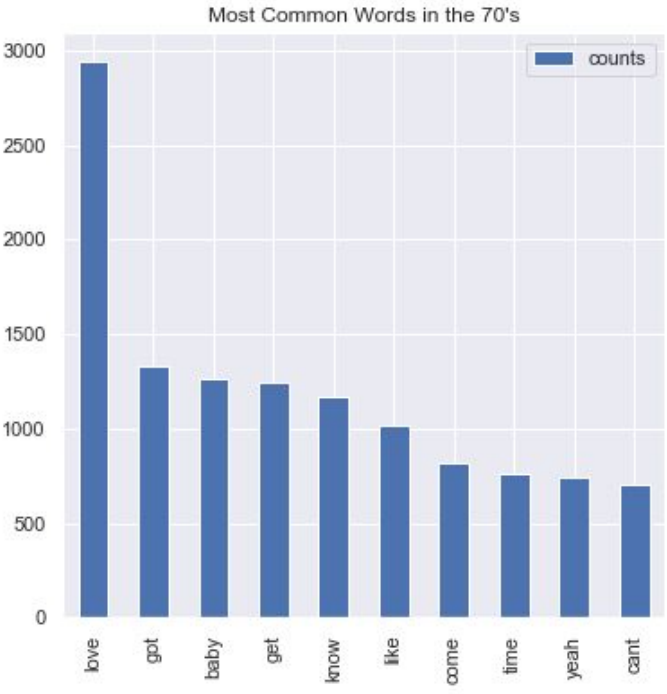
After performing this cleaning of the data, I began to look at what information may be gleaned from looking at the biggest pictures of the data. The first thing to note is that this problem is somewhat unbalanced, as the 60's have less than 500 songs and the 2010's have less than 600 songs as they both were divided in this dataset by the fact that it ranged from 1965-2015. The rest of the decades in the dataset have a bit less than 1000 songs each remaining after the cleaning process.

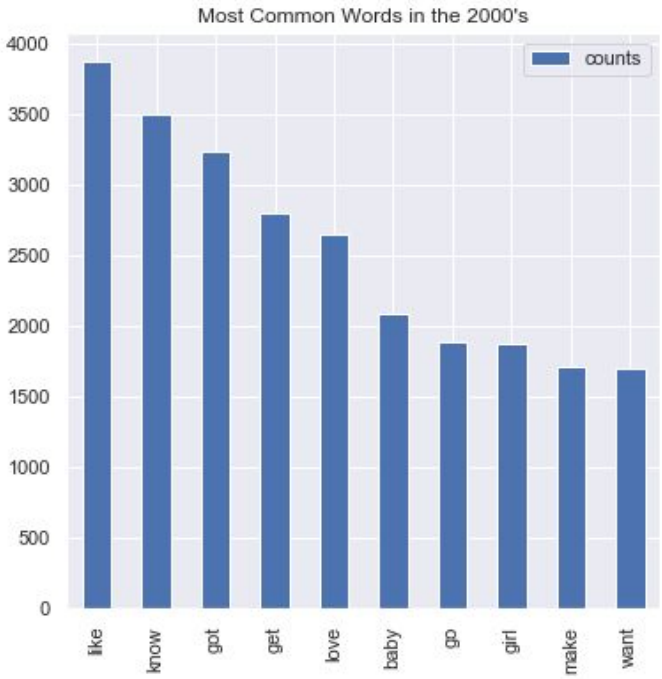
One aspect of the data that I thought it might be interesting to observe was who the most common artists were in the data. While this observation is not immediately relevant to the problem at hand, observing who had the most songs in the data could be useful in terms of understanding which artists lyrical choices were the most effective in terms of creating a popular song. Madonna is the most prevalent artist in the Billboard Top 100 for this timeframe, with Elton John and Mariah Cary trailing behind by a wide margin.

The first steps I took in looking at the lyrical data was looking at word counts. I generated word counts for the entire dataset as well as counts for over every decade, in order to see if there were any obvious patterns in word usage change that might be observed from simply observing

the word counts. ‘Love’ is the most common word and the data, is unsurprisingly the most common word in most of the decades of the data set. When observing the word count for each decade it becomes somewhat obvious that throughout the decades many of the most common words in the data set across decades are identical. Most of each decades’ top word counts are practically identical to that of the whole dataset, with only one or two words of the top ten in each decade being different from the whole data set. The 70’s wordcount is provided here to demonstrate this, looking at the word counts for each decade here would be mostly redundant, as they are all very similar to the 70’s count. One occurence of note is ‘like’ eclipsing ‘love’ as the most common word in the data during the 2000’s. In the 2000’s love was shoved down to the fifth most common in terms of occurence. While this is only a minor change, it is encouraging to note this occurrence as it does show that there are some significant shifts in word usage in our dataset across decades, which will be important for our classification algorithm in the future.

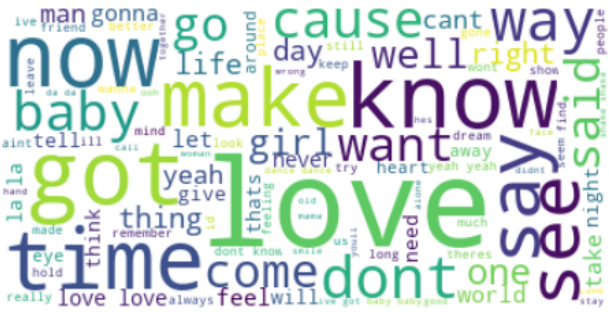






After analyzing the word counts, I needed something that could show more granular changes in the data. At the advice of my mentor I generated word clouds for each decade as they could show some changes that may not be as obvious when purely viewing top word counts. Conveniently there is a python package that generates word clouds called wordcloud, which I used to generate the following images:

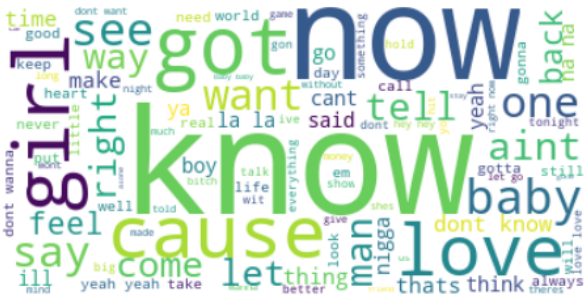
1960's



1970's

1980's

1990's



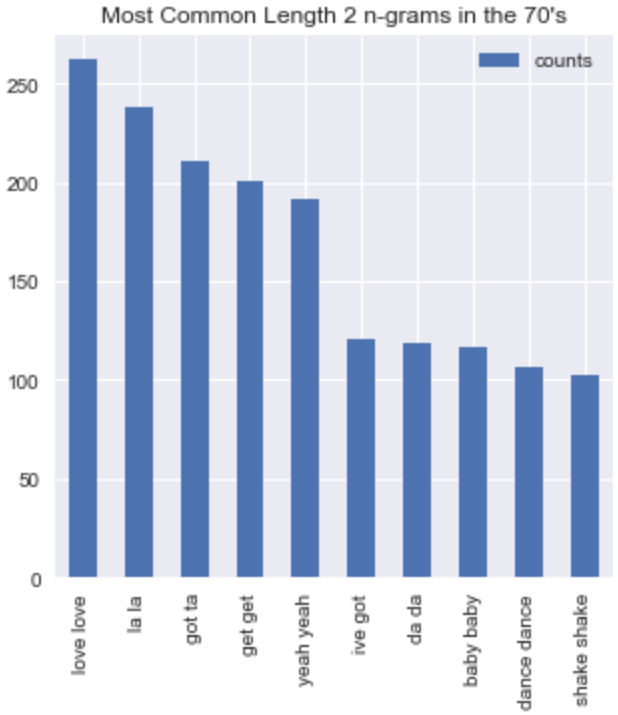
2000's

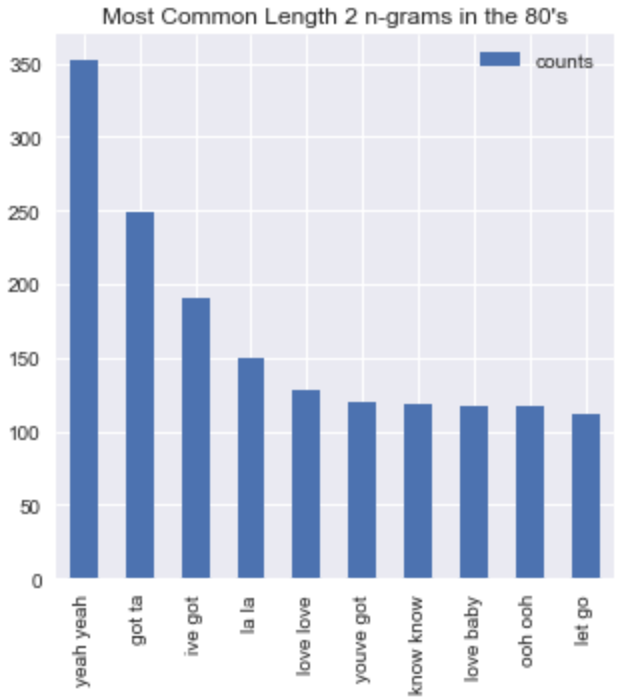
2010's

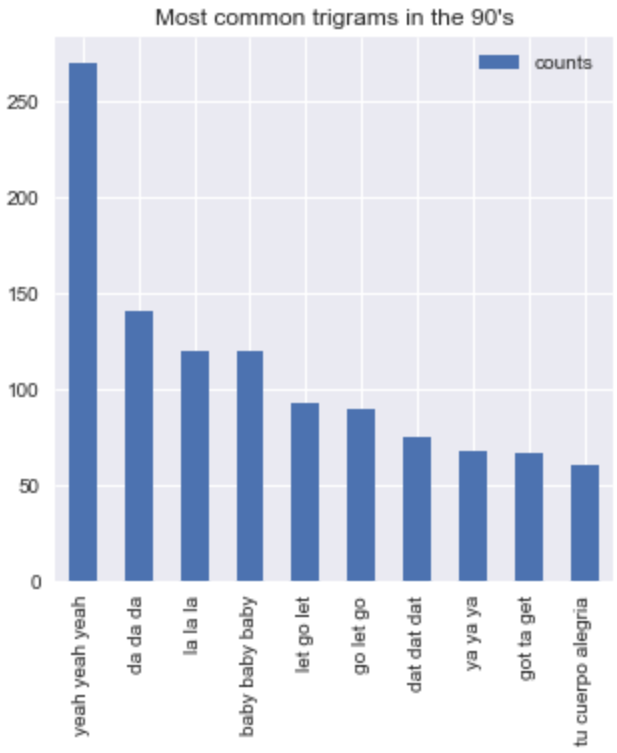
We can again see the many word count similarities across decades with these word counts. The word clouds also provide insights into the differences though, such as the obvious drop in prominence of love, as well as observing a few other changes between the decades. These word clouds seem to imply that in terms of straight word counts, the decades themselves are not very

dissimilar to each other, and that something more meaningful than word count will need to be used when trying to classify songs by decade.

In addition to word counts I also wrote a function to generate and count n-grams for the lyrical data. N-grams are generated by looking at a set of text and combining the text into groups of length n words in sequence. For example, 'The quick brown fox' has the following n=2 n-grams, or bigrams: 'The quick', 'quick brown', and 'brown fox'. My function counted the number of occurrences of each n-gram and I visualized the top ten bigrams and trigrams for each decade. The n-grams proved to be about as informative as the word counts, as many of the most common n-grams were the same across decades. The n-grams do provide some insight into the most common occurrences of words together, however many of them do seem to be somewhat nonsensical for human reading as the stop words of the text were removed. The n-grams show that nonsense lyrical sounds are common in the data, such as 'la la' or 'ooh ohh' which can both be observed in the provided visualizations. Another very common bigram is 'got ta' which provides some interesting insight into the fact that there are built in misspellings in the text, since one would assume the proper spelling of that phrase would be 'got to'. The trigrams of the data provided even more insight to the very nonsensical nature of text with stopwords removed, as most of the trigrams in each decade were dominated by more musical phrases that only make sense in the context of music such as 'la la la', 'ohhh ohhh ohhh', 'yeah yeah yeah' and 'love love love'. There was an interesting hiccup in the trigrams from the nineties that I was quite surprised by. If you observe the trigrams from the nineties in the diagram below, you will notice that the top ten trigrams are dominated by



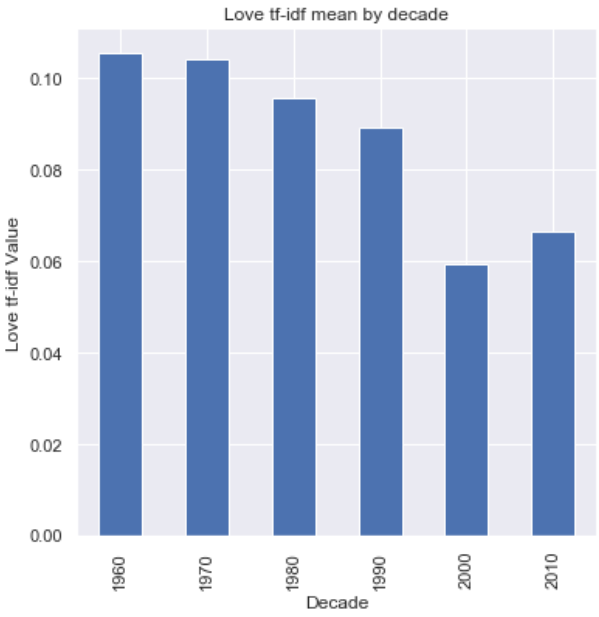


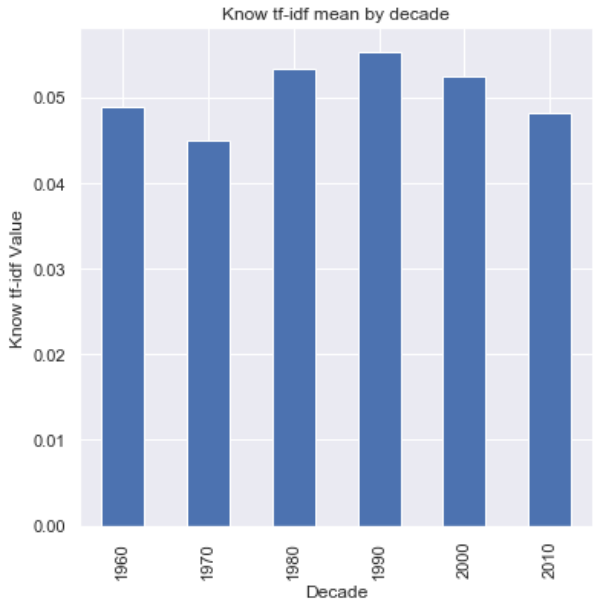


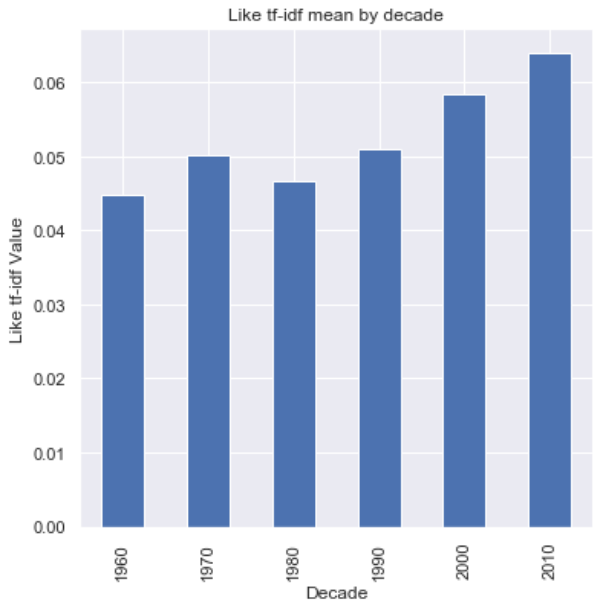
obviously musical phrases, except for the tenth most common trigram which is the Spanish trigram 'tu cuerpo alegria'. I was curious to see that one of the top ten most common trigrams was a spanish phrase, so I looked for the phrase in the data, as I expected it to occur in multiple songs across the dataset, however I found that I was quite mistaken. The phrase only occurs once in the dataset, in the infamous song by the duo Los del Rio, 'Macarena'. Finding something like this could be quite troubling when generating features for a model, as this single song caused such a strong outlier, it could create a large amount of noise for a classification model to work through if this type of thing happened enough across songs. While typically n-grams would be used as features for a predictive model while solving this type of problem, my mentor advised me to not use the n-grams as features for this specific project in order to save some computing time and limit the scope of the project moving forward.

After performing these in-depth looks at various ways to observe word counts in the data, I moved to generating the features that would be used for the predictive model for this project, the term frequency-inverse document frequency(tf-idf) values for each word and each song in the dataset. Tf-idf values quantify the relative importance of the words in a document relative to the words occurrence across documents. Scikit-learn has a feature extraction tool called TfidfVectorizer which can create a tf-idf vector for documents easily, and that is the tool I used to pull the tf-idf values out of the songs for this project. When the tf-idf values for this data were first generated, there was a total of 41789 words in the dataset. This large number was primarily a result of the fact that there were many typos in the dataset, obviously there may have been some songs with unique words, however many of the words that only occurred once in the data were typos, and the number of features simply needed to be reduced in order to generate a better model. After dropping all of the songs that occurred only once in the data, 15345 words remained. Based on my own intuition as well as advice from my mentor, I dropped every word that occured in ten or fewer songs. This allowed me to keep 3199 words in the data set.

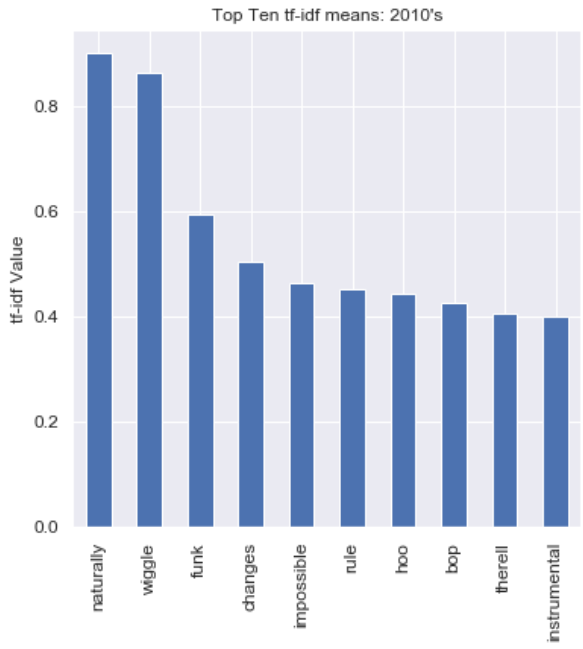
After generating the tf-idf values I began to analyze them in order to get an idea of how they may differ from looking at word counts. The first tf-idf values I was interested in looking at were the tf-idf values for 'love', 'know' and 'like' since those words were the three most common words in the dataset. I wanted to analyze how their tf-idf values might change over time and how those changes compared to the trends in word counts over time. I calculated the tf-idf





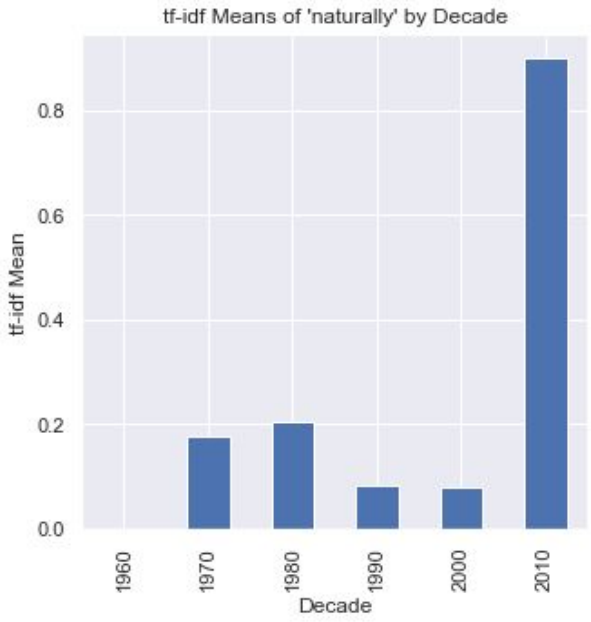


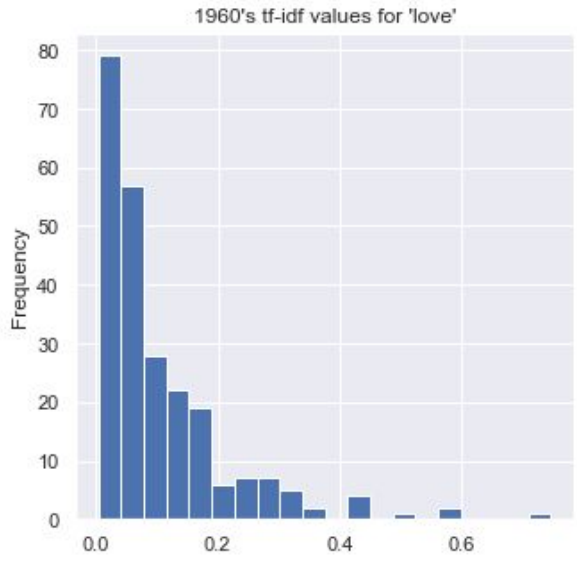
mean for each of those words and created the visualizations that you can see here. The mean tf-idf for ‘love’ seems to have a fairly significant drop over time, which is something that could have been expected by observing the changes in word counts over time and the word clouds that we observed before. The tf-idf mean for ‘know’ seems to remain relatively constant over the decades that we are looking at for this project. The tf-idf mean for like appears to increase slightly across time, with an obviously greater variance than the mean of ‘know’ across decades, but a smaller variance than the mean of ‘love. Observing these three words, we can hope that many of the words in the data set are more like ‘love’ and less like ‘know in terms of how the tf-idf values tend to vary over time, since that would tend to result in our model generating better results once we start creating our model. I also took time to plot the top ten highest tf-idf means for each decade in order to try to get an idea of how the tf-idf means could give a different picture of the lyrical content for each decade. The top ten tf-idf means across decades showed significant differences between decades which felt fairly promising for our model, considering the tf-idf values are going to be the features for our model, however, if you observe the tf-idf means for the 2010’s below, we are about to see that means can be quite deceiving.

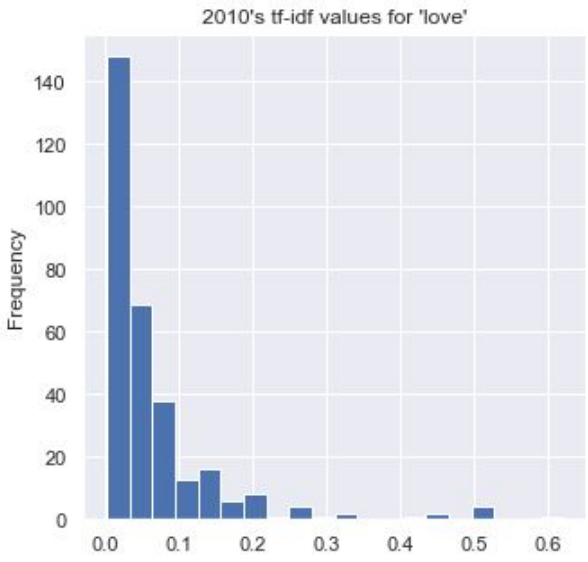


In Depth EDA:

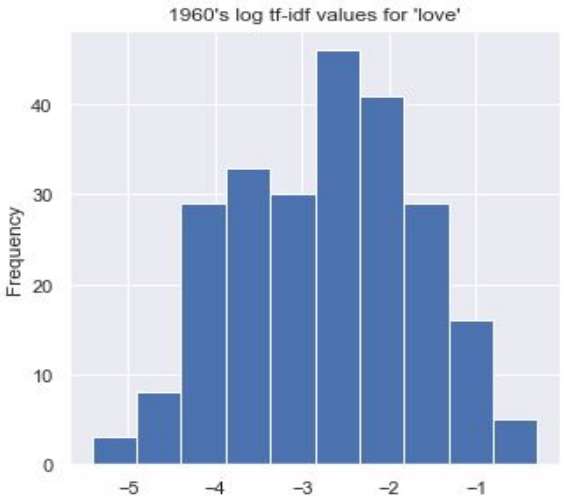
I previously alluded to means being deceiving, and would like to elaborate on that point now. I found the top ten tf-idf means for the 2010's to be especially strange, specifically the word 'naturally' being the word with the highest tf-idf mean, as that word had not popped up anywhere in any of my previous analysis, and I was curious as to what might be happening to cause 'naturally' to be the largest tf-idf mean in the 2010's. As we can observe in the figure to the right, 'naturally' had an incredibly large increase as opposed to any other decade, which is not like any of the patterns we had previously observed with the other words we looked at. When I took a closer look at the specifics of the data I quickly found out why this happened. In the data for the 2010's, 'naturally' only occurred in one song, which was a song by Selena Gomez titled 'Naturally'. When we observe this in contrast to a decade like the 2000's, where 'naturally' occurred in four songs, and had a low tf-idf value for each of them, we can see why such a spike would happen. With outliers like this, one would hope that a model we create would be able to ignore this type of noise, since it could be quite disruptive to a model.

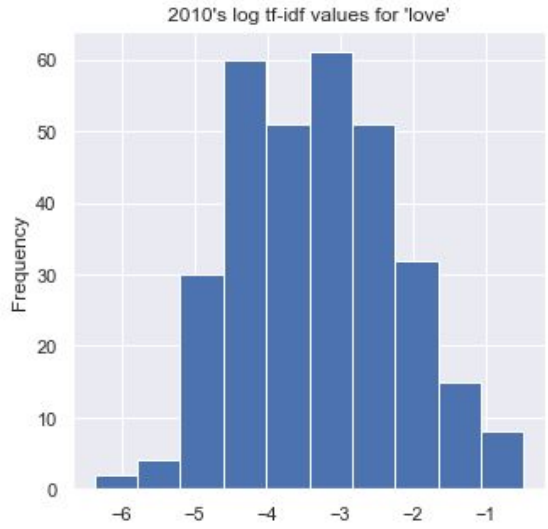






I also wanted to take an in-depth look at the tf-idf values for 'love', in order to check and see if the difference in means of the tf-idf values of love were significant across decades. While working through my Springboard coursework I learned various methods for performing experience and was interested in performing a t-test for difference of means on the 'love' tf-idf data. As we can observe in the histograms to the left, the tf-idf values for 'love' in the 1960's and 2010's are exponentially distributed. You cannot perform a t-test on data which is exponentially distributed, so I calculated the log values for each of the data and the histograms of





the log values can be observed here. The log values of the tf-idf data for 'love' are normally distributed, so one can perform a t-test for difference of means on this log data. Performing the t-test showed that the difference in means was significant with a p-value of 1.15×10^{-10}. After observing these two very different types of problems with the distributions of tf-idf data for the words, one would hope that the data tends to be more like the trends that were observed in 'love' as opposed to what happened with 'naturally', as that could lead to better results with the model.

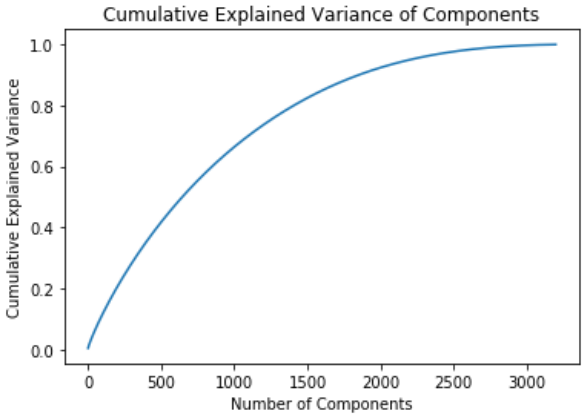
In-Depth Analysis:

After performing EDA on the data that had been generated for the project, it was time to begin creating predictive models for the project. I decided to try a few classification models to begin with to develop some knowledge about putting together those models, as well as seeing what the best performing model would be for this problem. As of writing this I am now aware that these types of problems are best solved using neural networks and the results that I obtained using the models that I used were somewhat limited, however the primary point of this project was to maximize the amount of learning that I obtained from the project, so I was not expecting the best results.

I decide to try three algorithms for classification using the tf-idf data: support-vector machines, a random forest classifier, and an XGBoost classifier. The SVM and random forest models were used because Springboard used them as good examples of classifiers, and I used the XGboost classifier based on advice from my mentor. To begin analysis, I checked the accuracy results of each of the models out-of-the box with a 80/20 train-test split. I focused on pure accuracy since other evaluation methods for this particular project were somewhat irrelevant since we were most interested in seeing how good the model is at predicting the decade of song based on lyrical data, and accuracy was the only metric that seemed appropriate given the project did not need to be too broad or in depth. The SVM classifier had fairly poor out-of-the-box results for the data, with approximately 20% training and testing accuracy, which is only slightly better than random guessing for this problem, as random guess with six categories is around 16.67%. The random forest classifier did a bit better out of the box with a 98.5% training accuracy and a 33% testing accuracy. The XGBoost classifier performed the best out of the box

with a training accuracy of 69.7% but a testing accuracy of 37.5%. Since the XGBoost classifier performed best on the data with its out of the box model, I decided to move forward with the XGBoost classifier as the model that I would use moving forward with this project.

After testing the performance of the model on the raw tf-idf data, I used principal component analysis on the tf-idf data to try to obtain better model performance for the data while reducing the dimensionality of the features used in the model. The initial PCA did not improve model performance at all, so I used sklearn's StandardScaler to scale the tf-idf values and ran PCA again. The primary components for the PCA with the scaled tf-idf were able to perform better, so they were the components that I used to move forward with the model. After observing the cumulative explained variance by number of components for the model, I decided to check the range of components between 1500 and 2000 as those components seemed to capture enough of the explained variance of the features as well as having significantly lower dimensionality than the raw features. I checked the accuracy of out of the box performance on this range of features in steps of 25 and performed five-fold cross validation with the best performing numbers of features in order to try to find the best number of features for the model. Using 1750 features had a 38.2% testing accuracy with the out of the box model and had 37.5% testing accuracy with five-fold cross validation, which was the best performing number of features out of the entire range. With the number of features taken care of I proceeded with parameter tuning the XGBoost classifier.



Parameter tuning of the model proved to be unfortunately unproductive. I used grid search cross validation to tune individual parameters of the model as well as a final randomized search in the ranges of the optimized parameters to try to discover the most optimal parameters for the model. After performing all of these grid searches and finding the optimal parameters, the tuned model using accuracy as the scoring metric, did not perform better than the 38.2% accuracy that the out-of-the box classifier had. The accuracy optimized model had a testing accuracy of 35.36%, which was worse than the out of the box model. Since the accuracy tuned model did not perform better than the out-of-the-box model I also tuned a model optimizing for log-loss and that model attained a testing accuracy of 37.38%, which outperformed the accuracy tuned model but was still not as accurate as the out-of-the-box model. These results somewhat vexed me and consulting with my mentor regarding these results seemed to reveal that I was not obtaining better results with parameter tuning because I was at the limits of how the XGBoost classifier could perform with the data that I was using.

Conclusion:

As I mentioned before it seems that an XGBoost classifier is not the best model to use for the classification problem being addressed here, however the final model still performed significantly better than random chance, being more than two times accurate than random chance. The results of this project seem to be that it is not very easy to use lyrics to identify what decade a song is from. This seems to make sense, as language in music does not seem to change too drastically between decades as we observed earlier in the project. If I had more time to spend on this particular project I would look into creating a neural network to address this problem, and include the n-gram information as features for the model. I would have also spent more time to find the lyrics missing from this particular dataset as well as some more extensive time cleaning the data to pull out some messy words and patterns, as well as removing some of the nonsense words that were not removed from the dataset due to time constraints and the difficulty of discovering the nonsense words. I learned a great deal about natural language processing throughout the process of completing this project, and I am glad to have experienced the challenge of addressing this problem, but I do wish that I could have obtained better results. I may return to this problem in my free time in the future to see how much better a neural network can perform on this particular problem.