|  |
| --- |
| **KMeans Clustering with tf-idf Vectorization to Predict Song Lyric Genre** |
|  |
| **Elias Rosenberg**  **elias.k.rosenberg.22@dartmouth.edu** |
|  |
|  |
|  |
|  |

Abstract

This project utilizes KMeans clustering of song lyrics to try and predict the genre of input lyrics from the user. Song lyrics are converted into vectors using the TFidfVectorizer method from sklearn, and then grouped into 6 genres (Country, Rap, Rock, Heavy Metal, Pop, and Jazz) based on salient features from each genre. This program takes copy-pasted input from the user found from any lyrics website, and attempts to predict its genre. The project is split into three .py files that 1) scrape the web for song lyrics using the Genius API and python module *lyricsgenius* 2) cluster the lyrics using aforementioned algorithms 3) generates testing lyrics and returns accuracy data.

Introduction

In the past two decades, the explosion of music streaming software has led to the increased need of classifying, sorting, and suggesting music for consumers. Spotify, iTunes, Apple Music and other streaming services rely on prediction algorithms and user data to group their songs into the correct genres. This data is important for introducing users to songs they might enjoy and allowing for easier searching of new songs within the same genre. Doing this can be very difficult. All songs are multidimensional and self-influencing. That is to say, one song may encompass aspects of multiple other genres in its timbre, beat, melody, or lyrics. How can one effectively categorize Rock from Heavy Metal when both genres seemingly share so much? Natural Language Processing may be the answer. Through tf-idf vectorization and KMeans clustering, we can find the vectors of song lyrics, quantifying each word in n-dimensional space. We can then group them together into clusters based on the similarity of those values. If certain words show up in one cluster more so than others, it is possible that that cluster represents its own musical genre. We can take note of these salient features, and then feed the program new lyrics to see if they share similar features to any of the clusters. This could allow us to predict the genre of a song with lyrics alone. I concede that this is not the best way to group songs. Lyrics are just one puzzle piece in the grand structure of any song. The most affective song clustering algorithms take into account timbre, common rhythms, instruments, artists, user data, and many other aspects holistically. This serves as a toy example of what NLP can do for a musical problem. Others have also attempted to solve this problem in a more focused way. Some have tried to group songs based on their hierarchical structure through Recurring Neural Networks (RNNs)[[1]](#footnote-1), while other researchers have used more complicated clustering algorithms.[[2]](#footnote-2) This project shows what can be accomplished with limited programming knowledge and a group of common NLP algorithms.

Methodology

The data used to train the model was derived from two places. At first, I planned on using a dataset found on Kaggle of approximately 80,000 songs from three genres (Hip Hop, Rock, and Pop) to train and test the model. The predictions and clustering were consistent for Hip Hop and Rock, but Pop suffered from a low accuracy rate. It was often misconstrued with the other genres, as Pop is a rather nebulous genre in the American song tradition and encompasses aspects of both Rock and Hip Hop depending on which decade you pull your songs from. I decided to save this data set to use for testing, as all the songs were neatly kept in a csv file, with song lyrics stored in one row at a time. To gather more data, I shifted gears towards web scraping to get a fuller representation of lyrics that encompassed more genres. Using the Genius API and a python module called *lyricsgenius* I was able to scrape approximately six thousand songs from notable artists in Country, Rap, Rock, Pop, Heavy Metal, and Jazz. This created a text file of over 14 million words. The data was manipulated by removing English stop words with the NLTK stop words module, as well as more words I have found that permeate all genres and act as lyrical filler. Terms like ‘la,’ ‘da,’ ‘ah,’ ‘oh,’ (and its extensions like ‘oooh’) and ‘yeah’ were all added to the set of stop words to be removed. Through regular expressions I removed all punctuation (including apostrophes within conjunctions) as well as any digits. There are many papers online that give an in-depth explanation of how KMeans[[3]](#footnote-3) and tf-idf vectorization[[4]](#footnote-4) work, but I will give a brief overview here for context. Tf-idf, or “term frequency-inverse document frequency,” occurs in two parts. Term frequency is the measure of each token within a document. Inverse document frequency is the weight given to each term based on its frequency. It is called the “inverse” because the higher the frequency of a word in a document, the lower its weight, and vice versa. Tf-idf is important because it tells us which words are meaningful, and which words are so abundant that they are likely stop words.[[5]](#footnote-5) KMeans is a bit more complicated. By using the .fit() method of TFidfVetorizer from sklearn, we can shrink the n-dimensional vectors from tf-idf into a 2D plane of points, each representing a word in a song. KMeans chooses a set number of random places within that graph to place points (the number of points is the number of desired clusters from the user). Those points are then assigned to a closest centroid, which is the average center of the current points within that cluster. The program then assigns the initial random point to the new centroid and calculates a new average center for the cluster. This repeats until the points have converged and form distinct clusters of points with similar characteristics.[[6]](#footnote-6)

Results

Accuracy of the program was derived by taking songs with genre tags from the original Kaggle data set and running them through sklearn’s predict() method within KMeans. Accuracy was simply the number of correct predictions out of the number of songs. To check the accuracy of the genres not present in the data set (Jazz, Country, and Heavy Metal) I was going to manually run new song lyrics through the same method and count the correct predictions. In an earlier version of this program, accuracy hovered around %60, correctly predicting Rock and Rap, but not Pop. With this go around of data, I am crushed to say that the program is not running as I had hoped. The clustering is phenomenal. Words that one would stereotypically find in country (‘beer’, ‘truck’, alcohol’, ‘Jesus’) are consistently clustered together. The same goes for the other genres, with Rap yielding clusters with (‘b\*tch’, ‘hoes’, ‘hundred’, ‘f\*ck’), and Heavy Metal containing (‘blood’, ‘death’, ‘sink’, ‘storm’, ‘pain’). When I run songs through the predict method, however, every song is predicted as rap. Even when I run 100 songs of two other genres I know to not be rap, they still yield a prediction of rap. This means that out of the three hundred songs I tested with known tags, by virtue of one hundred of them being rap, I got a %33 accuracy rate. This is much worse than the earlier model, and makes little sense considering how much larger/more evenly dispersed between genres my training data is. Earlier tonight I worked with a TA, looking over all my code, and I believe we found the culprit.

Discussion of Results

Despite the clustering yielding excellent, distinct results with no overlap, my TA and I found through calculating the cosine between cluster centers with numpy that they were remarkably close. So close in fact that it was hard to tell them apart, which is why the model predicted every input it had as belonging to one cluster center. I am dumbfounded as to how this is the case, considering that the features for the clusters have no overlap at all. I am also saddened that taking the time to add thousands of songs to my dataset made the outcome worse than when I started. I wish I could say what I did wrong to improve upon my program, but the TA and I were stumped as to how this happened. Suffice it to say, my program did the opposite of what I predicted it to. If my program had worked as intended, this code would be a great, lightweight way to implement song classification for streaming platforms. Beginner web designers could use it as a building block for their lyric databases and websites. I’m sure there is a way to reliably differentiate songs by genre with just their lyrics, but the outcome of this program makes me pessimistic about using KMeans.

My previous concerns about whitewashing and linguistic exclusion have been abated the more I understand how KMeans and tf-idf function. Since tf-idf derives its vectors from term frequency, and KMeans will cluster anything, adapting this for other languages would be very easy. Avoiding whitewashing can be solved by splitting the text into tokens at spaces and removing apostrophes from words like “’aint” to streamline the data while keeping the idea of the words intact. If there are ethical problems caused by genre identification of songs, it would come from the exclusion of niche genres themselves. But again, if KMeans isn’t equipped to tell the difference between rap and anything else, there is no way it could tell the difference between Samba and a Bosa Nova.

Conclusion

The only conclusion I can derive from expanding my dataset to encompass six genres is that KMeans and tf-idf should not be used to predict song genres. I’m not sure there is enough difference between contemporary American songs of different genres for the algorithms to make distinct enough clusters. If I were to expand this work to try and fix it, I would use another clustering algorithm entirely that widens the gap between clusters to better differentiate between lyrics.

# **References**

Barreira, Cavaco, and de Silva. “Unsupervised Music Genre Classification with a Model-Based Approach.” Accessed June 6, 2021. <https://link.springer.com/content/pdf/10.1007%2F978-3-642-24769-9.pdf>

Tsaptsinos, Alexandros. “Music Genre Classification by Lyrcs Using a Hierarchical Attention Network.” Accessed June 6, 2021. <https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1174/reports/2728368.pdf>.

Jin X, Han J. “*K*-Means Clustering.” In: Sammut C., Webb G.I. (eds) Encyclopedia of Machine Learning. Springer, Boston, MA. Accessed June 6, 2021. <https://doi.org/10.1007/978-0-387-30164-8_425>

Qaiser, Shazhad. “Text Mining: Use of TF-IDF to Examine the Relevance of Words to Documents.” International Journal of Computer Applications (0975 – 8887) Volume 181 – No.1, July 2018. Accessed June 7, 2021

https://www.researchgate.net/publication/326425709\_Text\_Mining\_Use\_of\_TF-IDF\_to\_Examine\_the\_Relevance\_of\_Words\_to\_Documents

1. Tsaptsinos, “Music Genre Classification”

   2 Barreira et al, “Unsupervised Music Genre Classification” [↑](#footnote-ref-1)
2. [↑](#footnote-ref-2)
3. Jin X, Han J, “*K*-Means Clustering.”

   4 Shazhad, “Text Mining: Use of TF-IDF”

   5 Shazhad, “Text Mining: Use of TF-IDF”

   6 Jin X, Han J, “*K*-Means Clustering.” [↑](#footnote-ref-3)
4. [↑](#footnote-ref-4)
5. [↑](#footnote-ref-5)
6. [↑](#footnote-ref-6)