Lyric Extrapolatinator

CMSC 473 UMBC FALL 2020

Abstract

In the process of giving a song a tag of a particular attribute, there must be common trends in the song. In this case, the tags were applied and compiled by an open music metadata encyclopedia called MusicBrainz. So, assuming that there is a significant correlation between the type of song, and the lyrics, there must then be a correlation of the applied tag to the lyrical word choice. The first goal of this model is to be able to predict tags on collections of words, such as from a poem, in order to give insight on how it would best work as a song. The secondary goal of this project is to, given the predicted tags, be able to have the model generate a list of words to extrapolate on the input text.

1 Credits

This document was written by: Brandon Funk The data sources were: The Million Song Dataset, tags from MusicBrainz, and musiXmatch dataset, the official lyrics collection for the Million Song Dataset, all available at: http://millionsongdataset.com/

2 Introduction

The following paper was written for a class project in a Natural Language Processing class at UMBC 2020. I like artificial intelligence, language, and music, so I figured this would be an engaging and informative project to try.

3 Developing an Approach

I started the approach on the basis that there are linguistic trends between the chorus of a song, and the rest of the lyrics, likely varying by genre. One example of such trends would be: Eleanor Rigby- The Beatles lists specific people and their actions in the verse, but then has a generalized "all the lonely people..." This trend of specifics in the verse to generalization in the chorus might be detectable by the model if present in enough data.

The original goal of this project was to be able to identify trends and generate lyrical verse, but after familiarizing myself with the data, I realized that this would not be possible without additional linguistic data, since the musiXmatch dataset was only a bag of words format. And, in implementing a grammatical framework model from a different dataset, would only introduce bias from the additional dataset. The model, with the grammatical framework could very well be able to generate text using the word choice from the lyric analysis. But it would all be in the format of prose, where the framework would likely have been derived, and would not achieve the desired result. Therefore, the refined, realistic goal with this data and this aim, would be to generate a word cloud to assist the user in the writing process by categorization and extrapolation of lyrics.

Also, it must be said that this is not representative of music as a whole, seeing as many songs are only instrumental and breaking many songs into only the lyrics removes a lot of the meaning and feeling, but nonetheless, this was a fun project.

Another really cool project on the same subject is a program called Keywords to Lyrics by Mathias Gatti. I would love to know how this program actually works. More specifically, how creativity could have a score between 0.0 and 1.0 And although this program is cool, I decided that in terms of a tool in the creative process, it was a little demanding and unwieldy. The results were rather unpredictable, and could come from any sort of songwriting style.

4 Curating the Data

The data was available in a text file or an SQLite database. Of course I chose to use the SQLite Database because they said it was faster and more convenient. After obtaining the 2GB lyric file, 700MB metadata file, and the 130MB tag by artist

file, was all set to start training my model. Well, not quite yet. Not knowing that I needed to know how to use SQL, I watched a number of Socratica videos about SQL, and learned quite a bit. I had to inner join the 3 files, by artist id and by song id. The resulting file was 60GB in size, with each word having the tags of the associated song. After having to generate this table 4 times, each taking over an hour because Google Colab notebooks' autosave feature set the SQLite in read-only mode, and prevented me from using any of the data. This took a lot of waiting. Also, in order to normalize the probabilities, I need the counts of individual tag occurrences. This is easily achieved with another SQL command.

5 About The Tags

The tag dataset provided two sets of tags. One was user applied with 2321 unique tags through musicbrainz, and the other was 7643 unique tags from the Echo Nest. I chose to go with the Echo Nest terms because on completion of creating the model, they gave the results that I was going for and there didn't seem to be any problem with using them.

6 The Probability Section

This is using unigram probability with lambda smoothing. I started with lambda = 1 to prevent out of vocab errors. The probability of a tag applied to lyrics is:

$$p(lyrics, tag) = \sum_{i=1} log(p(word_i|tag))$$

Probability of the a tag applied to a word:

$$p(word|tag) = (c(word, tag) + \lambda)/Z$$

$$Z = \sum_{wordtypev} c(v, tag) + \lambda$$

7 Issues

There are a number of issues that I've encountered thus far:

I needed to learn how to use SQL commands effectively to get the desired data.

I don't know if my probabilities are correct. I may try log probability if I get unexpected results

Some categories are scarce, so the model may not be able to get enough data to be robust. This should however be solved when I move to the full 200GB dataset. See Table 1.

Tag	c(songs—tag)
alternative rock	8837084
alternative soul	2881
alternative urban	4386
alto	556
alto sax	253
alto saxophonist	309
alvaroidm	327
am pop	76425
amazing singer	157

Table 1: A sample of the tag distribution.

Also, some tags are not particularly intuitive. Ex. "alvaroidm" I don't know what that is. Though, this isn't that big of an issue, because the tags act as a sort of hidden layer in the generation of words.

Another issue that I'm currently facing is that from working with google drive, the tables that I've created with all of the necessary data is almost hitting the limit of my google storage. The current working capacity is 97.30/107.77GB This is only going up, as I execute SQL SELECT commands, so I'm going to connect to my local runtime environment in order to keep access to the combo meter in google colab.

Furthermore, in order to train on the full dataset, I must: 1.develop a more efficient way to generate the necessary data 2. use a much more powerful computer to complete these computations.

I also had to change my approach to a bag of words oriented task instead of trying to obtain a different dataset to fit my original approach.

8 What's Left

At the time of writing this section, I still need to code on how it breaks down an input sentence, returns the likely tags, and also returns additional words within those tags. Also, I need to regenerate the table on my localhost and start testing it out. Then, optimize for the best lambda. If I get access to a faster way of computing, I will set up everything using the full dataset and test there. If not, the 2GB sample would be a valid proof of concept. Finally, I want to make a song using this tool, but that will be on my own time.

9 Finishing up Plan

At the point of writing this, I must prioritize getting the model to work correctly. The steps are as such: First, parsing the sample input into the stemmed vocab. Second calculating the tags with which to search. Third, use probability to get the adjacent words not in the sample input, but in the applicable tags. If all goes smoothly, then begins the quest for crafting the perfect SQL query in order to optimize precomputation runtime and minimize the size required. Currently the required data extrapolates the given 3GB files into 60GB with a lot of duplicate data and is then condensed to 185KB. So to use the full 200GB dataset, there must be major optimization in the matching and counting of words to terms as to avoid a 4TB intermediary file.

10 Addressing Issues

I changed from a google drive hosted runtime to a local runtime, and that eliminated the fears of losing data on disconnect, and space issues for now. To get around the lengthy precomputation times for ”training” the model, I implemented a way of saving the compact vocab and term dictionaries by pickling in python. Since, these are both based on the input data and are independent of the model, it was much faster to access the data this way. Also, in the process of calculating the probabilities for each word based on relevant tags, I realized you can't prime combined probabilities with 0. 0*p(word—tag)*p(word—tag2)*... = 0 This bug was particularly confusing to find. Also, it was important to be able to sanitize the input lyric text file to the same stemming standard as in the musiXmatch dataset. This was done rather easily by implementing the same code that they used and provided in the creation of the dataset. Most of the rest of the work was figuring out how get the correct data and pipe it into the correct functions. This wasn't particularly noteworthy.

11 Results Discussion

So, I used the Led Zeppelin - Immigrant Song because it was the first song to come to mind. At first I was surprised to see the tags of celtic and sea shanties because the song does not sound like either. But, on closer inspection, the words ships, land, Valhalla, tales, horde, etc. would be indicative of these tags. Although, I would very much call it fantasy metal from the sound of it. These results

Tag	P-score(closer to 0 is better)
'contemporary celtic'	-554.44621285570
'traditional irish'	-554.54939475948
'sea shanties'	-555.25128472370
'fantasy metal'	-555.83980428733
'celtic music'	-555.88784327524
'heavy power metal'	-556.34196178326
'celtic fusion'	-556.70198333150
'british psychedelia'	-556.74383684804
'acid folk'	-557.21718571857
'epic power metal'	-557.55466351756
'symphonic power metal'	-557.77430877351
'battle metal'	-557.95003997388

Table 2: Relevant Tags for Led Zeppelin - Immigrant Song

Tag	P-score(closer to 0 is better)
one	-75.503787559250
time	-76.092816597878
know	-76.456840449725
like	-77.291802372399
see	-77.455967303019
night	-78.982042304802
heart	-79.056774507307
never	-79.320031014289
go	-79.338076133997
take	-79.620800235718
away	-79.632160351338
love	-79.796195365065
eye	-79.864629599649
way	-80.045227800807
life	-81.060453071322
hand	-81.543947185832
world	-81.77090921465
oh	-81.88050190329
man	-82.065359902321
dream	-82.662491594255

Table 3: Generated words for Led Zeppelin - Immigrant Song.

just highlight the difference between songs genre and lyrical tags. Additionally, the words generated, were as thematically relevant as I hoped they would be. Albeit, not all of the words generated were, but this could be solved by having a more exhaustive stop list or another way of ruling out words that are less tag specific across the board.

12 Limitations to this Approach

Since this was a bag of words approach, word choice was the only thing that mattered. In order to get this to be more representative of real lyrics, I would first need access to a lot of lyrics in verse style as the way they were sung in the song and a structural framework of some sort. So this program only serves a niche, and rather restrictive goal, but that isn't really an issue considering its intended place in the creative process. With the data I did have, I think I made about as useful a model that I could have.

13 The Hyper-parameter Shuffle

Three hyper-parameters that were relevant in this project: nterms - The number of relevant terms allowed per input song. This is important because it affects the terms that are used to choose which words are returned. The way that the program is laid out, each term is weighted the same. It is just the words' prevalence within those terms that determines if it is in the top n words returned nwords - The number of relevant words to be returned per input song. This limit was just in place to not have to return the entire vocabulary on each call of the otherwords function (the one that chooses adjacent words not already in your song lmbda(lambda) - This is added just to prevent probabilities with c(word—tag)=0 And doesn't have much of an effect until about 500, where it changed the order of the words, and replace 1 word from the control. Since there was no method of scoring in place, the lambda's relevance is up to preference. For most of other testing, I kept it at 1.

14 Future Developments

First, given the resources, and some additional effort, I would like to run this on the full dataset to get higher quality results. But as it stands now, the model still works in the way I hoped it would. Also, I want to do is use this tool that I've made and see about its effectiveness in practice. And since I chose to only use the bag of words unigram approach on this project, a further project would be to make a model that analyzes song and verse structure as well, but this would need a different dataset and approach.

14.1 References

I only put these in once, but it prints twice when I compile. Don't know why, but that's fine.

- The Million Song Dataset: (Bertin-Mahieux et al., 2011).
- Learning how to SQL: (Socratica, 2019).
- Keywords to Lyrics: (Gatti, 2019).

References

Thierry Bertin-Mahieux, Daniel P.W. Ellis, Brian Whitman, and Paul Lamere. 2011. The million song dataset. In *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011)*.

Mathias Gatti. 2019. KEYWORDS TO LYRICS generate custom songs from an ai.

Socratica. 2019. Introduction to sql(computer science). https://www.youtube.com/watch?v=nWyyDHhTxYU&list=PLi01XoE8jYojRqM4qGBF1U90Ee1Ecb5tt.