Lyric-Based Playlist Generation

Creston Brooks, Henrique Schechter, Elaine Wright

TRA 301

May 10, 2021

1. Introduction

1.1 Background and Motivation

Music streaming services like Spotify, Pandora, Google Play Music, and Apple Music offer song recommendations, and it is in their best interest to tailor these recommendations to individual users, since accurate and personalized recommendations make users more likely to subscribe to their services. The industry standard is to make these recommendations based on user data rather than the songs' content. This is most usually done through user-based collaborative filtering: the service will recommend songs to a user that others with similar tastes also liked [5]. This process (along with other forms of collaborative filtering) relies on large amounts of data about user preferences to build user "taste profiles". As such, popular songs are advantaged in that they are well-represented in user data, since newer songs by lesser-known artists do not have the amount of data points necessary to enter this positive feedback loop. Even if a less popular song would be very suited to a user's taste, it may not get recommended because it is not favored by collaborative filtering. Moreover, this form of collaborative filtering leads to poor recommendations for new users, as there is not enough data about their taste to determine what users they are similar to, and thus what to recommend to them.

Collaborative filtering neglects to consider the content of songs, such as musical and lyrical qualities, which has potential to produce more effective recommendations. Notably, while musical features are a key part of one's enjoyment of a song, lyrics may be just as important [4]. Moreover, because lyrical speed and stress patterns can act as proxies for some musical elements, like rhythm and melody [4], and because text is significantly easier than audio to store, analyze, and interpret intelligibly, an efficient text-based method of song recommendation promises to be more feasible, yet similarly effective as an audio-based analysis.

1.2 Objective

We posited that lyrical analysis is a promising means of generating song recommendations, given a number of seed songs upon which to base the recommendation. We sought to develop a procedure that could provide a list of song recommendations, i.e. a playlist, based on the lyrical analysis of seed songs, without considering information such as genre, artist, year of release, or even the playlists we used to evaluate our performance. The notable exception is the analysis of song structure, for which we used pre-existing labels of song parts, but in the absence of such data, these parts can be estimated well, as in Fell and Sporleder (2014) [4]. In addition, we do collect each song's duration for the purpose of estimating its tempo, which is not strictly a lyrical analysis, although it is closely tied to it.

2. Literature Survey

We first consulted the existing literature. In designing a lyrical analysis system, Fell and Sporleder (2014) modeled songs as vectors of features, divided into five main categories: vocabulary, structure, orientation toward the world, style, and semantics. There were thirteen feature classes, including POS type-token ratios, ratios of personal pronouns, distribution of verb tense, and presence of repetitive structures, among others. This vector representation was used for three classification tasks: genre detection (~52% accuracy), song quality prediction ("best"/"worst" ratings ~75% - 85% accuracy), and song publication time prediction (~47% accuracy) [4]. Though this paper did not tackle song recommendations, the results empirically indicated that song featurization can provide a meaningful result in the domain of lyrical analysis. We applied the concept of song featurization as a means to represent a song in our recommendation task.

However, we were dissatisfied with the computational efficiency of some features in Fell and Sporleder (2014), such as rhyme detection [4]. In addition, we were motivated to seek out other features they did not consider, so we searched for what other features have been used for the task of song recommendation and how they were implemented. One lyrical feature widely considered useful was sentiment. We found an efficient lexicon-based method to extract it presented by Chen and Tang (2018) [2] (we explain our adaptation of it in Section 3.3.5 below).

Next, we had to determine how to recommend songs based on our song vectors. Many recent papers presenting lyrical-based recommendation systems tended to use raw lyrical content

as input to neural networks that created song representations, which were then either used by a later part of the neural networks to predict song similarity, or were compared using the cosine similarity function to predict song similarity. We tried both approaches. For the neural network, however, we changed the goal of determining whether songs are similar to the more direct goal of determining whether they are likely to be in playlists together. We took our general architecture from the latter half of the neural net described in Balakrishnan and Dixit (2016) [1], who fed raw lyrics into a neural network. The ideas we take from them are explained in detail in Section 3.4.

Finally, in order to decide how exactly to evaluate our model's performance, we looked at how song recommendation systems deal with the problem of taking in a *set* of songs (i.e., a playlist) and recommending more songs based on those. The most common methods were aggregation of songs using the mean and using the max. The former may involve taking the mean of the input songs' vectors and treating that as a single song to which the model would compare possible songs to recommend (which involves a lot of information loss, for example a playlist of polarly opposite and extreme songs, with respect to some feature, which would be treated as average), or recommending the song that has the highest average similarity when compared to the input songs. The latter involves recommending the song that is most similar to any of the input songs. The consensus seemed to be that taking the mean of the vectors of the seed songs and treating that as a seed song is not ideal. This makes sense, given that, for example, this method would disregard a playlist with polarly opposite but extreme songs as one with average songs, with respect to some feature. Indeed, Vystrčilová and Peška (2020) [11] found that the best way to define the similarity between a prospective song to recommend and the seed/input playlist was to define it as the largest similarity between the prospective song and any of the playlist's songs. As such, we scoped our project to focus on maximizing the predictive ability between two individual songs being in a playlist together, and accordingly evaluated recommendations based on a single seed/input song, but the usefulness of our model generalizes to seed/input playlists by using the playlist-song similarity described by Vystrčilová and Peška [11] and recommending the song most similar to the playlist.

3. Methodology

3.1 Approach

With a similar aim to Fell and Sporleder (2014), who featurized lyrics for the purpose of content analysis [4], we determined features of interest with which to quantify and compare songs represented as vectors. We took some features employed by Fell and Sporleder (2014) and included new ones, with the goal of capturing a different dimension of lyrical content with each one. Our final features can be placed broadly into the following categories: vocabulary, structure, orientation, style, and semantics. For each feature, different types of preprocessing were necessary. Many involved different treatment of punctuation (tokenization, deletion, or replacement), the removal of stop words, the conversion of numerals to words (3 to three, 11 to eleven, etc.), deletion of metadata included within the lyrics (e.g. current singer, or divisions of song sections), and other efforts to achieve consistent measures of each song for each feature. Several features did not lend themselves to non-English content, and thus, we implemented a measure to filter out songs that were not detected as English with above a 99% certainty. After the implementation of our features, we attempted to train a neural network to predict if two songs co-occur in a playlist, and additionally, we considered the cosine similarity between the feature vectors of songs to determine what the best playlist recommendations would be for a given seed song (both described later in detail). We analyzed our results and compared our predictive performance with that of random song recommendation, as well as addressed challenges and points of interest and improvement for future research.

3.2 Dataset and Tools

The dataset which inspired us to take on this project is the Spotify Million Playlist Dataset. It is part of the Spotify Million Playlist Dataset Challenge, which is currently in its second iteration. The dataset contains one million playlists and over two million unique tracks by almost 300,000 different artists. We ignored a lot of information contained in the dataset such as playlist names and song metadata because of our project's scope, although in practice it would be useful information in creating a maximally effective song recommendation system. We use just a subset of this dataset for our final evaluations and results: 141 playlists that cover 9,244 non-unique songs, of which 5,394 were unique, of which 4,994 were determined to be in

English. For the neural network, we used an even smaller subset: 840 unique English songs that were a part of 13 playlists.

We used Python 3.6.9 hosted on Google Colab, and stored our data on the cloud through Google Drive. What follows now are the different python libraries we used and what we used them for.

Lyric collection and preprocessing

We used the *Genius API* connected through *lyricsgenius* to get the lyrics for all of the songs we analyzed and the song part annotations (intro, chorus, etc.) (we used *json* to read the data file); we used *shelve* to (permanently) store our data (lyrics, song feature values, trained models, etc.) in our disk and in the cloud; we used *re* to edit song lyrics using regular expressions, for instance to remove song part labels, which occur inside square brackets; we used *gensim* for stopword removal; we used *num2words* to convert numbers in numerical form to word form, so as to not differentiate between them; we used *nltk* for intelligent text tokenization.

We used *langdetect* to filter out non-English songs below a confidence threshold of 0.99. This library's implementation was non-deterministic and varied noticeably between runs, and as a result, many non-English songs were not filtered out and affected results; however, its performance was within our requirements and still succeeded in filtering out a large amount of non-English songs.

Featurization

We also used *gensim* to create and train our Doc2Vec models, and *nltk* for text processing features such as parts of speech tagging; we used *sentiwordnet* (within *nltk*) to access SentiWordNet and get sentiments for individual words; since we found *CMUdict* to be too slow as a syllable dictionary, we used *syllables* to get high-speed syllable estimates with good accuracy; we used *fuzzywuzzy* to find similarity between two pieces of text, which we used for title detection, repetition detection; we used *pronouncing* for detecting rhymes.

Recommendation and evaluation

We used *pandas*, *sklearn*, and *numpy* to process the data and *pytorch* to actually create and train the neural network model; we used *heapq* (heap queues) to get n best song recommendations for arbitrary n; we used *sklearn*'s TSNE methods along with *pandas* and *numpy* to create two-dimensional representations of our song vectors, and *plotly express* to plot these; we made use of *random* at times to evaluate a random subset of data (e.g. ~ one million

song pairs as opposed to ~ twelve million); we also used *scipy* and *numpy* in both testing and visualization, to handle the song and feature data.

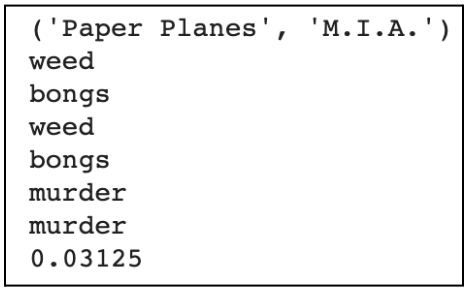
3.3 Song Featurization

3.3.1 Vocabulary

A natural consideration for a song's vocabulary is its use of profanity, or more generally, **words that are considered offensive or taboo**. Industry standard dictates that songs should be marked if they are explicit, since this can be an important consideration for consumers and their song choice. Moreover, offensive language can be significantly polarizing for playlist creation, with many users listening to solely non-offensive songs, or on the other hand, mostly offensive ones. The proportion of words in a song's lyrics that are offensive is also somewhat indicative of genre. High proportions of offensive language will be strongly correlated with rap and hip-hop, and lower, non-zero proportions will have some correlation to pop and rock. In general, any given genre is likely to have some consistency in its use of offensive language. Given that consumers will often listen to songs in one or few genres, a feature that strongly correlates with genre will be effective in song recommendation. Likewise, people will often listen to multiple songs by the same artists, and each artist will be relatively consistent in whether they use offensive words or not.

To determine what proportion of a song's words are offensive, it was necessary to have some dictionary of offensive words. We started with a list titled "Offensive/Profane Word List" from Luis von Ahn's research group at Carnegie Mellon. The list contains over 1300 words that could be considered offensive and is described as "good start for anybody wanting to block offensive or profane terms on their site" [10]. With this list as a starting point, we filtered out words that might be considered offensive in some contexts but would not be productive to include for our purposes (e.g. "Australian", "yellow", "toilet"). Several other preprocessing measures were taken, including the addition of plural noun forms and inflected verb forms, as well as alternate spellings. Our final list contained 1029 words that can commonly be considered profane, offensive, or taboo, ranging in content from swear words to violence to substances to slurs. No list can reasonably contain all offensive content nor be fully consistent with what should or should not be considered offensive, but we found it to be notably comprehensive and

consistent, in practice, as most songs only draw from a very small subset of the 1029 words on our list.



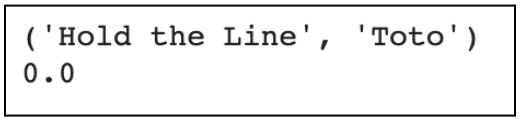


Figure 1: For example, the lyrics of "Paper Planes" by M.I.A. (left) are ~3% offensive, whereas "Hold the Line" by Toto (right) does not contain any offensive words.

Another consideration for how a song's vocabulary might impact song recommendation is the lexical complexity of a song. Analyzing if lyrics contain commonly or uncommonly used words has potential to correlate to genre. Additionally, **frequencies of words** will remain fairly consistent within songs by a single artist. Artists who write about popular topics and use common words will likely do so in many of their songs, whereas artists who write about nicher topics and use obscure words will tend to be consistent in this behavior. Moreover, it is logical for listeners to also have a preference in this regard, with some enjoying more complex diction and others preferring more mainstream topics and simple word choice.

To measure lexical complexity, we scanned through every song in our database and found the frequencies of each word across the corpus, with common stop-words being removed, as well as any word that did not contain alphanumeric symbols. For each song, we then found the average frequency of its words. The mean frequencies were divided by the total number of words in the corpus of all lyrics, such that a result 0.001 would indicate that the average word in a song comprises 1/1000th of the overall corpus of words.

One illustrative example of the results is the comparison of "Red Barchetta" by Rush, which had an average frequency of 0.000998, and "Closer" by Ne-Yo, which had an average frequency of 0.010397. By our metric, "Closer" uses words that are on average over ten times more frequent than those in "Red Barchetta," observed easily in the excerpts from each song shown in Figure 2. Note that although stop-words were removed for calculations, they still appear below.

I just can't stop
I just can't stop
I just can't stop
I just can't stop
And I just can't bring myself no way

And on Sundays I elude the eyes
And hop the turbine freight
To far outside the wire
Where my white-haired uncle waits

Figure 2: Excerpts from "Red Barchetta" by Rush (right) and "Closer" by Ne-Yo (left). As one can imagine, "Closer" uses words that are more frequent in our song corpus.

Another common measure for lexical complexity is the **average number of syllables per word** in a corpus. The Flesch reading ease test is a standard metric for how difficult a text is to read, a direct consequence of lexical complexity [6]. The test captures readability from a 5th grade level to a complexity beyond a college graduate's expected level using only the metrics of words per sentence and syllables per word. Although we cannot expect lyrics to contain structured sentences, and creating some proxy for this measure lends itself to heavy inconsistency, we can still analyze syllables per word to get a sense of readability.

Perfect precision for this calculation requires a syllable dictionary, since syllable counts for English words cannot be precisely known from just spelling. We attempted to use the CMU Pronouncing Dictionary [8], however, having to search such a large dictionary for every word in every song proved infeasible for reasonable performance. Thus, we instead made use of the Python library *syllables* that estimated syllable counts quickly enough for our purposes, although some accuracy was lost.

Other metrics that indicated lexical complexity were **noun and adjective type/token ratios**. We calculated the proportion of noun tokens that were proper and common, perhaps evidence of whether a song is about a place or person, and the proportion of adjective tokens that were plain, comparative, and superlative, suggesting more exaggerated or emotional diction.

We also computed the **part-of-speech token ratios** for nouns, verbs, and adjectives to find the proportion of all tokens that were nouns, verbs, or adjectives and learn more about the lexical breakdown of a song as a basis for comparison.

3.3.2 Structure

We implemented two measures identified in the Fell and Sporleder (2014) paper to featurize song structure [4]. First, we checked whether or not the **song title was contained** in the lyrics. It is unusual for song lyrics to *not* contain the title, so we wanted to detect these unique

instances. Famous examples include “Bohemian Rhapsody” and “Viva la Vida”, and perhaps songs like these without the title in the lyrics can be considered more sophisticated or intricate.

We also looked for instances of **repetitive structure** (Figure 3). In the Fell and

[56] ’Cause now I see right through you	**[61]** But I see right through you
[57] Look into my eyes	**[62]** I look into your eyes
[58] Tell me what you see	**[63]** Tell you what I see
[59] I see a man who thought you loved me	**[64]** I see a girl who ran game on me
[60] You played me like a fool	**[65]** You thought you had me fooled

Figure 3: An example of two aligned blocks within the lyrics of a song, in this case NSYNC’s “See right through you” [4].

Sporleder (2014) paper, the authors searched for lexically and structurally aligned blocks of text within the lyrics [4]. A nested loop heuristic that compared each line or sliding windows of multiple lines to each other line or window of the same length in a song took too long to run for the purposes of our model, so we tried to simplify the approach by comparing each line against each other line. Lines were considered lexically and structurally repetitive if they exhibited an exact or fuzzy string match (*fuzzywuzzy*) that was thresholded at a token set ratio of 90 with any other line. The feature was represented as the proportion of lines that were repetitive in a given song text but unfortunately was still too time-intensive to be included in the model.

The Genius API breaks song lyrics down into sections called “parts”, e.g., “Intro”, “Verse”, “Chorus”. We wanted to determine how a song’s structure deviates from the conventional structure of the genre to which it belongs, or from the structure of another song to provide recommendations. This feature, which we termed **structure deviation**, would be quantified as an edit distance. In a simple case, an ABAB structure (“Verse”, “Chorus”, “Verse”, “Chorus”) and an ABCB structure (“Verse 1”, “Chorus”, “Verse 2”, “Chorus”) with different lyrics for Verses 1 and 2 would have an edit distance of 1. A dictionary was created for the types of song parts identified by Genius, including appropriate preprocessing to make the part names compatible, e.g., changing “Verse” to “Verse 1” if a song did not specify. We also considered weighting certain edits over others, because certain song parts are more similar, especially “Hook” and “Chorus.” However, we found that part names were inconsistent or even lacking for about 20% of the songs in Genius, meaning that we could not include this feature in our model.

This would be a worthwhile future step, but would require preprocessing and other lexical analysis to identify the parts of unlabeled songs.

3.3.3 Orientation

We calculated the **proportion of words that are first, second, and third person pronouns**, separately, for each song. The feature captures the subjects of the song—that is, narratively speaking, whether the character singing the song is mostly talking about themselves, talking *to* someone, or talking *about* someone or something else. Some strong examples are songs with very egocentric monologues, such as Kanye West's *Power* (large relative frequency of 1st-person pronouns) and songs addressed to a lover, such as Frankie Valli's *Can't Take My Eyes Off You* (large relative frequency of 2nd-person pronouns). We considered 60 personal pronouns, including some anachronistic, informal, and slang ones we hadn't seen before reading through lots of lyrics, as well as some possessive pronouns and some contractions that include pronouns. Some interesting ones include *whatcha* (*what are you*)*, youse, yalls, thy, yeer,* and their variants. We use total words as a denominator, not total pronouns, so as to capture how prevalent personal pronouns are in the text and to not determine that, for example, a song with a single pronoun, "I", is maximally egocentric.

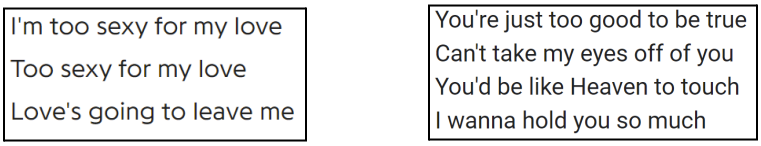


Figure 4: Excerpts from "I'm Too Sexy" by Right Said Fred and "Can't Take My Eyes Off You" by Frankie Vallie, the first with an aptly high first person pronoun score, and the latter with aptly high first and second person pronoun scores.

We also calculated the **verb type/token ratio** to find the proportion of verb tokens that are past, present, and future. This feature indicates how the song and its narrator or subject matter are aligned with respect to the world and its timeline. There may also be a semantic basis for this, as it is possible that songs about the past are more wistful and longing, for example, while songs about the future evoke eagerness and anticipation, and such songs ought to be grouped together in our recommendations.

3.3.4 Style

While the content of a song's lyrics might seem like their most important attribute, the lyrics' style is another dimension of lyrical analysis worth investigating. In their study of lyrical style for the purpose of genre identification, Fell and Sporleder (2014) find the length of a song to be a key stylistic feature [4]. Informed by this finding, we calculated the **total number of words**, as well as the **total number of lines** in each song and recorded each as its own feature.

Moreover, since this is a purely lyrical analysis, we have no explicit indication of any song's tempo, a factor which heavily impacts the style of a song. However, since Genius provides us with the duration of each song, we can use the measure of **syllables per second** as a proxy for tempo. In reality, this measure is impacted not only by how fast lyrics are sung, but also by how much of the song is instrumental (when time is passing without any lyrics). Although distinct considerations, both of these qualities could reasonably have an impact on music taste, and thus, syllables per second remains an enticing measure.

A final stylistic factor we considered was **rhyme**. Similar to the issue with syllables, it is impossible to predict whether two English words will rhyme purely based on their spellings. This necessitates the use of rhyme dictionaries, which proves too costly to query for each word in a song. Thus, we checked only the final word in each line, seeing if it rhymed with the final word in either of the two following lines. Note, too, that although near rhymes are very commonly used by songs, we only consider perfect rhymes, as this was the available rhyme feature of the chosen Python library. In addition to rhymes, this feature is also a strong detector of final word repetition between lines, since any word will "rhyme" with itself.

We encountered a similar issue when trying to detect internal rhymes by converting words to IPA and examining vowel clusters. The method proved successful; however, it was far too slow, taking about five seconds to process each song, and thus did not remain a part of the rhyming feature.

3.3.5 Semantics

The general idea with **Doc2Vec vector representations of song lyrics** was to treat a song's lyrics as a document or paragraph and apply the Doc2Vec, which generally applies to documents/paragraphs, to the lyrics and obtain a numerical vector representation of each song's lyrics. These representations served a dual purpose in our paper: as a benchmark for how a

relatively naive, non-interpretable representation could do compared to our interpretable featurization, but also as a feature itself, intended to capture word choice and word order, which are of course features of syntax and vocabulary. In combination, they are nearly representative of semantics, as they represent the chosen words *and* their contexts. For instance, Doc2Vec might be able to capture that a song says "I don't want you, leave me", which is more complex than just the words individually—a text that reads "I want you, don't leave me" is the polar opposite in meaning.

Generally Doc2Vec works by training the vector representation to be useful at predicting words in a given paragraph, as explained in the official paper that presented Doc2Vec [7]. The model that considers word order is the Distributed Memory Model of Paragraph Vectors (PV-DM). In it, the paragraph vector is concatenated to vector representations of a series of words and used to try to predict the next word in the paragraph. Stochastic gradient descent and backpropagation are then used to adjust the parameters that determine the vector representation, arriving at a final representation only after iterating many times, using different samples of text fragments. However, we also tried the Distributed Bag of Words Model of Paragraph Vectors (PV-DBOW), wherein word order is not considered. It utilizes the same process, but without appending words to the paragraph vector to create the input vector: the paragraph vector is used directly to predict words that are randomly sampled from the paragraph.

It turned out the DBOW model outperformed the DM model with our data. For instance, the average cosine similarity between pairs of songs that did occur in playlists together using our features (which we thus hope to be low) was ~1.02 times larger when using DM than when using DBOW, averaged over 4,994 songs. The difference might be negligible and by chance, but a possible explanation would be that songs have sporadic utterances and disjoint text inside them that make it such that it is better considered a bag of words than a paragraph. For example, "Yeah, yeah, yeah, yeah, yeah, yeah / Up in the club with my homies" is only coherent within each line, not in combination.

Our second semantics feature, **sentiment analysis**, had the goal of capturing one specific dimension of the semantics of the text: the general emotions (i.e. sentiment) that the text expresses. We opted for using lexicon-based sentiment analysis as opposed to corpus-based sentiment analysis—in other words, analyze individual words in a song's lyrics as opposed to the entire body of the lyrics. This makes our process computationally much more efficient, and does

not require a labeled set of songs whose emotions have already been tagged (a resource which we do not have). This is admittedly inferior in capturing sentiment, since the context of words affects their meaning and sentiment, but it is a very good heuristic, since the sentiment of a word tends not to vary much within its different acceptations.

Inspired by Chen and Tang [2], who used sentiment analysis of lyrics for Chinese music recommendation, we use term frequency–inverse document frequency (tf-idf) as a measure of word importance in a given song's lyrics to then get the document sentiment from these words. We define the tf-idf of term *i* in song *j* as $tf_{i,j} * [1 + \log((1 + N)/(1 + df_i))]$, where $tf_{i,j}$ is how many times term i appears in song j, N is the number of songs in our corpus, and df_i is the amount of documents in our corpus that term i appears in. The first add-1 ensures $tf_{i,j}$ is not zeroed-out by a term with $df_i = 1$, and the second two prevent division by zero (for new terms not in our original corpus); they are effectively pretending that we have one additional song where every word in existence appears once.

While Chen and Tang [2] use a Chinese lexicon where sentiment is multi-dimensional (arousal and valence), we opted for SentiWordNet [3] as our lexicon, which contains human annotations of words' "positive" and "negative" values, from 0 to 1 each. For a given word, we determined its positive score and its negative score as the means of these scores over the word's different meanings. For a given song, we calculated the mean of the positive and negative scores of the ten words with highest tf-idf scores and returned those two scores separately. Ten words were chosen because keywords were often relatively low in the tf-idf list, beat by words that were part of choruses or repetitive portions (e.g. "yeah", "no"), and to ensure that enough words were sampled, since sometimes top words were not part of the lexicon (neologisms like "dougie", abbreviations like "yuh" or "tryna"), in which case they were not counted in calculating the song's mean sentiment scores. We tried using Canada's National Research Council's Emotion Lexicon [9], which also has binary dimensions "fear", "trust", "surprise" "joy", "anger", "sadness", "disgust", and "anticipation", but this representation of words seems limited to certain words for which these adjectives qualify, and yielded representations of songs which seemed arbitrary and which did not match our expectations for stereotypically and strongly negative/positive songs. We also considered using the difference between positive and negative scores as a single score, but this would not distinguish between an emotionally charged, bipolar song and an emotionally neutral song.

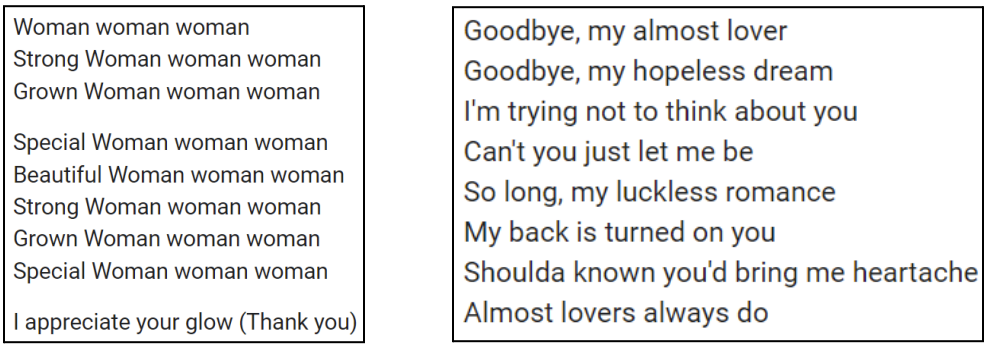


Figure 5: Excerpts from "Woman" by Raheem DeVaughn (left), one of the data's most strongly positive songs, and from "Almost Lover" by A Fine Frenzy (right), one of the data's most strongly negative songs. Note their sentiment score correctly captures the songs' sentiment, even though the latter uses deceivingly positive words such as "romance" and "lover" frequently.

3.4 Neural Network

Most papers in the literature trained neural networks to detect which songs are *similar*, assuming that similar songs tend to be in the same playlist. But we reasoned that playlists represent more than just a group of similar songs—for example, as a very rudimentary case, multiple sets of songs, each of which contain similar songs. As such, we thought we might get better results by training the neural network to directly predict whether songs would be in a playlist together, as opposed to whether they were similar. This also meant that instead of using pre-existing datasets that contain a list of similar songs for each song, we had to parse through the Spotify dataset and count co-occurrences of song pairs (i.e. what pairs of songs appeared in the same playlist somewhere in our data). We decided to keep our output binary (songs co-occurred in some playlist, or not) to make the song comparison a classification task with two outputs, but it seemed promising to change the formal task the neural network is solving in order to utilize the information of songs co-occurring multiple times.

Our neural network architecture was inspired by Balakrishnan and Dixit (2016) [1]. While they fed the raw lyrics (in the form of lists of word embeddings) into a long short-term memory neural network to create vector representations of songs, we used the vector representations we created by concatenating, for each song, the values of all of its features (including Doc2Vec). What we drew from their work is that we combined song vectors by taking their element-wise product, and that from then on we used a fully-connected neural network with softmax at the end (to produce whatever real-valued result we got into a probability in [0, 1]). To

summarize, the question our neural network tried to solve was: given the element-wise product of features of two songs, what is the probability that the two songs appear in a playlist together?

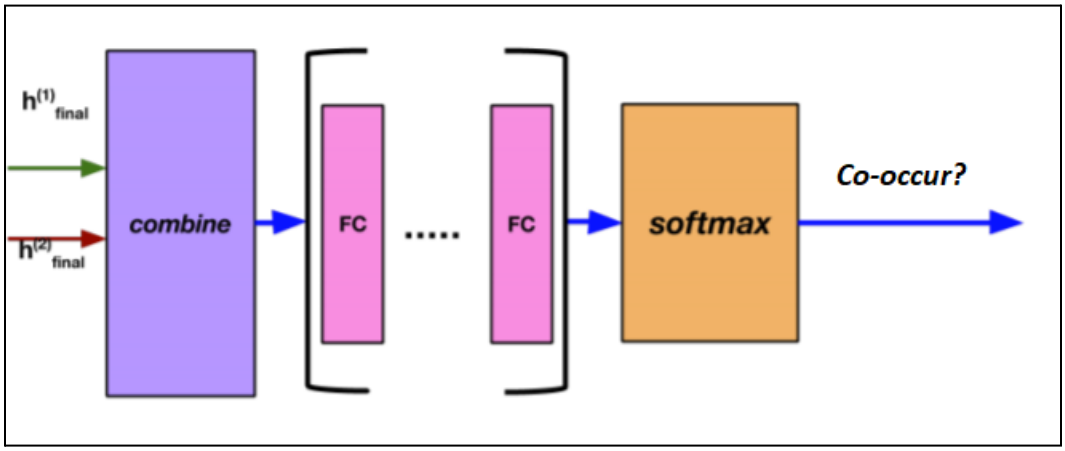


Figure 6: Pictured is the architecture we used, image adapted from Balakrishnan and Dixit (2016) [1]. Here $h^{(1)}_{final}$ and $h^{(2)}_{final}$ are vector representations of two songs, *combine* uses element-wise product, *FC* means fully-connected, and the output is a number in [0,1]

As for other parameters and hyperparameters we chose, we settled on the extremely standard and well-documented ReLU activation function between each of the layers. We tried using between one and three hidden layers, with between 25 and 100 neurons in each. We did not think the problem was so complex as to require a high amount of hidden layers or neurons, as we reasoned that features should almost directly indicate song similarity, which is a good proxy for playlist co-occurrence, but upon lack of good results we decided to try different values. We tried learning rates between 0.0001 and 0.1, varying learning rates by a multiple of 10 each time. Each time we trained, we did so over thousands of epochs over our data, which was 64,316 examples (80% of which was used to train the model and 20% to evaluate it), a decent portion of the 356,168 total song pairs that correspond to our 844 unique English songs. These consisted of all the 32,158 “hits” (co-occurrences) that corresponded to our 844 unique English songs, as well as a sample of all the “non-hits”, specifically the same amount as “hits”.

Our results were not very promising, with training and test accuracies floating around 0.5 most of the time. The best we found was a model with 0.515 accuracy on test data; however, as with some seemingly successful models we trained, this model had a lower accuracy on training data, significantly so, which leads us to believe the good test accuracy was due to chance variation. We believe the lack of success may be due to a lack of data. Although we had

significantly more training examples than many successful neural network models we investigated, our examples represented just over a dozen playlists. Since data preparation and model training took very long with the amount of examples we had, though, we were unable to re-train the model with significantly more data due to mere lack of computational power. We reason that sampling few hits and non-hits for a few songs in each playlist, but doing so over many playlists might yield better results, as more playlists would be better represented. However, it also might be the case that sampling too little from each playlist would prevent the model from understanding any single playlist's patterns very well.

3.5 Cosine Similarity

In another metric of comparison, we looked at the cosine similarities of the feature vectors for each song, with particular interest in if close cosine similarities could suggest that songs were more likely to be grouped together in playlists. We first computed and stored the feature vectors for every song in our database that were detected as English songs, amounting to 4994 unique songs across 141 playlists. We standardized for each feature (subtracting the mean and dividing by the standard deviation). After standardization, we computed the mean cosine similarity for every pair of songs that co-occurred in any playlist (420,914 total pairs). We then computed the mean cosine similarity for pairs of songs randomly selected from the total number of songs, selecting a random sample of just over one million such pairs out of a possible twelve million. We found these numbers to be 0.06694 and 0.0003915 respectively, indicating that on average, songs that co-occurred in playlists had cosine similarities about 171 times higher than randomly chosen pairs of songs (which includes some pairs that did co-occur). This result indicated that our composited features have a strong correlation to actual co-occurrence in playlists, a promising prospect for a playlist generation method using these features.

4. Experiments and Results

Building off our promising results using cosine similarity, we implemented song recommendation based on cosine similarity to one seed song. We then evaluated by checking if our recommended song is in fact present in some playlist where the seed song is also present (i.e. it co-occurs with the seed song). The precise method was as follows: we calculated the feature vectors for every song in our database (in this round of testing, we used 4994 unique songs

across 141 playlists); for a given seed song, we computed the cosine similarities between that seed song and every other song in our database; we returned the n closest similarities (and the corresponding identities of the closest songs); we performed this process for each of the 4994 unique songs in our database and recorded how many of the n songs with the closest cosine similarities to the current seed song actually co-occurred with it in at least one of our database's 141 playlists; we then repeated this process and returned n random songs and computed how many of the randomly selected songs co-occurred with the current seed song in some playlist. This random song recommendation was used as our baseline to distinguish the success of our song recommendation method from that of pure chance.

When recommending n = 5 songs for each of the 4994 songs in our dataset, we found that, on average, 0.636 of the recommended songs actually co-occurred with the given seed song, meaning that we were (0.636/5) **12.7% accurate** in recommending five songs given a single seed song. Random recommendation was only (0.1748/5) **3.5% accurate** for the same task.

We repeated the same procedure for recommending n = 1 songs, i.e. finding the song with the closest cosine similarity to each seed song and checking if the two songs co-occurred in a playlist. On average over each of the 4994 songs, we were **14.1% accurate** in this recommendation. Random recommendation for this task was again **3.5% accurate**. From these results, we find that our method of song recommendation was **4.0 times better than random** for recommending a single song for a given seed song.

Additionally, we tested to see when recommending n = 5 songs, at least one of them co-occurred with the seed song in some playlist. This is an important result to measure, as we wanted to ensure that our recommendations are not just very successful for some types of playlist and very poor for other types (as this cannot be determined from the average successful recommendations per playlist). We found that when recommending the five songs with the closest cosine similarities to the seed song, at least one song co-occurred with the seed **38.2%** of the time, whereas random recommendation had an accuracy of **15.0%** for this task. This result indicates that for almost 40% of seed songs (from 4994 songs across a multitude of genres and artists), we are producing at least one successful match.

Finally, we evaluated the strength of each individual feature by performing the same evaluation measures with each isolated value in the vector. This amounted to recommending the n songs with values closest to the seed song for just one feature at a time. We evaluated each

feature using only a random subset of the total 4994 songs to achieve quicker results, since we have many features, although reducing the amount of trials certainly reduced the accuracy of the results. We found that song recommendation with each individual feature on its own performed better than random recommendation and that the strongest features seemed to be the **average frequencies of words** and **the proportions of offensive words.** Possible reasons for why these features were strong might be the fact that both of them tend to be very consistent across works of the same artist; it is likely that songwriters will use a similar amount of offensive words across different songs, and likewise, they will draw from a personal lexicon of words for all songs they write, resulting in similar average frequencies of words. Since many playlists will contain multiple songs by the same artist, metrics that have good consistency across the same artist should be successful. Additionally, proportions of offensive words are very indicative of genre. Rap and hip-hop will often have high amounts of profanity, religious songs will likely have none, and pop, rock, and country will fall somewhere in between. Average word frequencies, too, will be prone to vary by genre, as pop songs will likely contain very frequent words, since their content is intended to be widely relatable. Playlists are regularly composed of one or few genres, and so, genre-differentiating will contribute heavily to playlist generation.

After evaluating our results, we used the dimension reductionality technique "t-distributed stochastic neighbor embedding" (t-SNE) to get two-dimensional representations to visualize clusters of playlists. To reduce clutter and best visualize whether our vector representations capture differences between playlists, we generated a random sequence of five playlists and visualized only those. In the image below, each dot represents a song. Songs of the same color are in the same playlist. Note that songs from the same playlist tend to cluster, even those with different artists and genres—the purple cluster at the bottom of Figure 7, for example, consists of songs ranging from pop to punk rock. The presence of the outlier "Tell the World" by Lecrae is possibly due to its extraordinary length and use of atypical words, reminding us that users, for whatever reason, will at times add songs to a playlist they have little connection to.

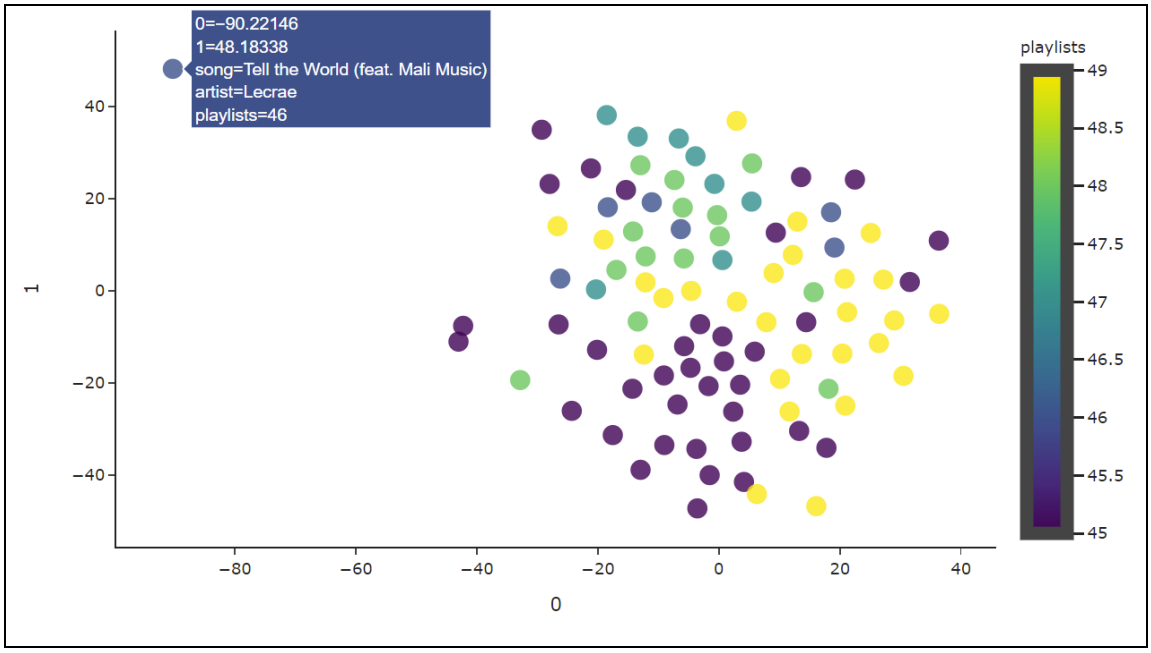


Figure 7: Visualization of a random set of 5 playlists. Each dot is a song, and songs of the same color are on the same playlist. Note how the songs on the same playlist tend to cluster together.

Figure 8 shows the same set of five playlists, but with an additional one (in yellow, playlist 50). The larger yellow cluster includes “Made In America” by Toby Keith, “Farmer’s Daughter” by Rodney Atkins, “Old Alabama” by Brad Paisley, and many songs by Corey Smith and Luke Bryan—all country songs and artists. The yellow songs that are clustered opposite to this main yellow cluster include songs by P!nk, Iggy Azalea, Rihanna, Ariana Grande, and Kesha—all very stereotypical pop artists. So, while our vector representations generally clustered playlists together notwithstanding variation in genre or artist, playlists with songs that varied to an extreme degree in content did produce distinct clusters. Nonetheless, this is still a successful differentiation between different types of lyrical content, in this case country and pop genres.

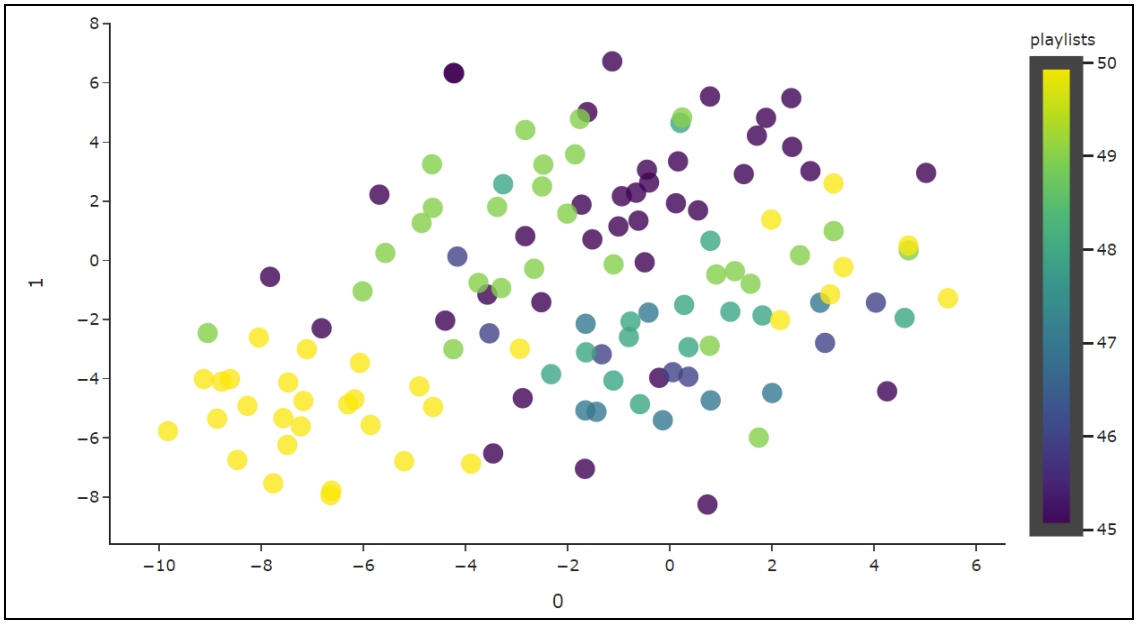


Figure 8: Visualization of the same 5 playlists from Figure 7, plus an additional playlist in yellow. The country songs are clustered at the bottom left, while the pop songs are clustered at the top right, showing that our model can distinguish between genres.

5. Conclusions and Future Work

While making decisions, we encountered and thought of many alternatives to choices we made that seemed promising and might even have reduced some issues we encountered. For Doc2Vec, it seemed that song lyrics as the concatenation of the lyrics' lines does not work well as a paragraph. It is possible that intelligently punctuating between sets of lines that should be treated separately, or otherwise treating song lines as more complex units, might yield better results. For song structure deviation, our methods were too computationally costly to work well with large-scale data, and so investigation into further heuristics might be fruitful. A lot of features and methods we used were also heuristical in nature so as to be computationally viable, such as syllable and rhyme estimation—with more computational power, more complete analyses of these features could yield better results. Perhaps most importantly, for alternative approaches to cosine similarity, neural networks might still be a viable choice—as described in Section 3.4, sampling songs over more playlists rather than sampling playlists in completion might be more successful, as would merely sampling more playlists with our same methods.

Our results warrant a few important observations. Firstly, we used heuristics and alternative methods to extract features from song lyrics in a **computationally efficient** manner. For example, we used lexicon-based sentiment analysis as opposed to corpus-based and we used

syllable estimation methods as opposed to querying a syllable dictionary. Second, we illustrated with our t-SNE visualizations and cosine similarity results that the set of features we selected yielded an **effective vector representation of song lyrics** (for playlist recommendation), with songs being closer to, and generally clustering with, songs they co-occur in playlists with (as well as songs they are lyrically similar to). We were unable to show that such a representation could yield good recommendations through a neural network, likely due to the amount of playlists which our data sampled (a result of our computational power). However, using a comparatively simple recommendation system that uses cosine similarity, we showed that **a recommendation system built with these vector representations for songs is successful at recommending** songs based on an input song. In fact, beyond performing much better than random chance, it performs well enough to be **useful in practice** (for example, 14% of recommended songs given one input song are "matches"). Moreover, since we have shown that we can effectively recommend songs with lyrical analysis alone, synthesizing our lyrical methods **with audio analysis and collaborative filtering would yield even more successful song recommendations**.

6. Roles and Responsibilities

Elaine: Structure parsing of songs and preprocessing to standardize names of structure parts (re); calculating a song's deviation from standard structure as edit distance (Levenshtein); exact and fuzzy alignment of related lexical/lyrical structures within a song (fuzzywuzzy); proportion of rhyming lines (pronouncing, num2words); title contained in song lyrics, exact and fuzzy (fuzzywuzzy); proportions of types of nouns, verbs, and adjectives (nltk); proportion of noun, verb, and adjective tokens (nltk).

Henrique: scraping lyrics from song names (lyricsgenius, json); putting all data (names, lyrics, urls, features, etc.) into python objects and saving to disk/drive (shelve); putting together all features (which included debugging some) and writing the song recommender (scipy, numpy); preparing examples/data for, creating, training, and evaluating the neural net (torch, pandas, sklearn); lyric preprocessing and tokenization for most features using regular expressions, stopword removal, etc. (re, gensim, nltk); sentiment analysis feature using tf-idf (sentiwordnet); training/creating doc2vec model (gensim); 1st, 2nd, 3rd person orientation feature; getting and analyzing results and performance of model w/ and w/o doc2vec, and of different individual features; using t-SNE to create 2-D visualizations of song vectors.

Creston: lyrical preprocessing and tokenization of features, (num2words, regular expression, etc.) Lexical complexity feature spanning complete lexicon. Quick syllable estimator (syllables) to compute syl/sec and syl/word without dict reference. Lexical lengths of lyrics. Adapting CMU list of offensive words, making list of 1029 "taboo" words, implementing taboo feature. Conversion to IPA vowels (eng_to_ipa) for near rhyme measure. Compilation of all songs that co-occur in playlists for training and eval. Data compression of song pair uri's for good search/storage. Creation of dictionaries to go between compressed uri's, lyric indices, and lyrics. English language detection (langdetect) and threshold setting. Final song recommender, testing, and results evaluation.

7. References

1. Balakrishnan, A. and Dixit, K. 2016. DeepPlaylist: Using Recurrent Neural Networks to Predict Song Similarity.
2. Chen, X. and Tang, T.Y. 2018. Combining Content and Sentiment Analysis on Lyrics for a Lightweight Emotion-Aware Chinese Song Recommendation System. In *Proceedings of the 2018 10th International Conference on Machine Learning and Computing* (ICMLC 2018). Association for Computing Machinery, New York, NY, USA, 85–89. DOI:https://doi.org/10.1145/3195106.3195148.
3. Esuli, A. and Sebastiani F. 2006. SENTIWORDNET: A Publicly Available Lexical Resource for Opinion Mining. In *Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC '06)*. European Language Resources Association (ELRA), Genoa, ITA.
4. Fell M. and Sporleder C. 2014. Lyrics-based Analysis and Classification of Music. In *Proceedings of {COLING} 2014, the 25th International Conference on Computational Linguistics: Technical Papers*. Dublin City University and Association for Computational Linguistics, Dublin, IE, 620-631.
5. Jacobson K., Murali V., Newett E., Whitman B., and Yon R. 2016. Music Personalization at Spotify. In *Proceedings of the 10th ACM Conference on Recommender Systems (RecSys '16*). Association for Computing Machinery, New York, NY, USA, 373. DOI:https://doi.org/10.1145/2959100.2959120.
6. Kincaid J.P., Fishburne R.P. Jr, Rogers R.L., Chissom B.S. 1975. Derivation of new readability formulas (Automated Readability Index, Fog Count and Flesch Reading Ease Formula) for Navy enlisted personnel. Research Branch Report, Millington, TN, USA.
7. Le Q.V. and Mikolov T. 2014. Distributed Representations of Sentences and Documents. In *Proceedings of the 31st International Conference on Machine Learning.* PLMR, Beijing, CHN, 1188–1196.
8. Lenzo, K. The CMU Pronouncing Dictionary. Retrieved from http://www.speech.cs.cmu.edu/cgi-bin/cmudict
9. Mohammad, S.M. and Turney, P.D. 2013. Crowdsourcing a Word–Emotion Association Lexicon. Computational Intelligence, 29: 436-465. https://doi.org/10.1111/j.1467-8640.2012.00460.x
10. von Ahn, L. Offensive/Profane Word List. Carnegie Mellon School of Computer Science. www.cs.cmu.edu/~biglou/resources/bad-words.txt.
11. Vystrčilová M. and Peška L. 2020. Lyrics or Audio for Music Recommendation? In *Proceedings of the 10th International Conference on Web Intelligence, Mining and Semantics (WIMS 2020).* Association for Computing Machinery, New York, NY, USA, 190–194. DOI:https://doi.org/10.1145/3405962.3405963

I pledge my honor that this paper represents my own work in accordance with University regulations.

/s/ Creston Brooks
/s/ Henrique Schechter Vera
/s/ Elaine Wright