Music Recommendations using NLP

Michael Behrens

28 Oct 2024

1 Problem Statement

The goal of this project is to create a NLP-based recommendation system for music, specifically for finding new artists. The goal of the system is to accept 1 or more artists that the user enjoys as input, and to then use NLP techniques to provide a list of similar matches.

2 Data Processing

2.1 Data Sources

The first step to completing this project is to compile a list of musical artists and source text for each of them. To source text for the artists I decided to use Wikipedia summaries, due to the high level of accessibility of the data (many artists have Wikipedia pages), as well as the ease of acquiring the data (python libraries exist that make fetching data through Wikipedia straightforward).

I sourced a list of approximately 3000 musical artists from this Kaggle dataset and used that as my bank of artists from which users can select and from which the system will recommend. This dataset contains, at the time of creation, the top artists by Spotify streams all-time.

Note that the obvious finite nature of this dataset which does not automatically update hinders the ability of this system to evolve, as there will certainly be new artists gaining popularity who will not be available in the system. However, this is a sacrifice I am willing to make as the goal of this system is more of a technical experiment and for fun rather than an attempt to make a viable product.

2.2 WikipediaParallelSearcher

Since the starting point of this project is a list of approximately 3000 artist names from the dataset (stored in **artists.csv**, Figure 1). As seen in Figure 1, artists.csv contains several columns containing streaming information, but for this project we only use the first column, containing the artists' names.

	A	B	C	D	E	F
1	Artist	Streams	Daily	As lead	Solo	As feature
2	Drake	85,041.30	50.775	57,252.60	32,681.60	27,788.70
3	Bad Bunny	67,533.00	44.82	40,969.60	23,073.00	26,563.40
4	Taylor Swift	57,859.00	85.793	55,566.70	50,425.70	2,292.40
5	The Weeknd	53,665.20	44.437	42,673.30	31,164.20	10,991.90
6	Ed Sheeran	47,907.70	17.506	42,767.90	33,917.00	5,139.80
7	Justin Bieber	47,525.70	18.868	27,988.00	17,183.90	19,537.70

Figure 1: artists.csv

The goal now is to map each of these artist names to their respective Wikipedia text. In this initial version of the project, the Wikipedia fetch will only use the first section of content, which we refer to as the summary. The remaining content sections are discarded. The justification for this is that the summary contains a high level overview of the artist and their music, describing the genre and style of their music. Later sections, particularly for well known artists, can focus on other details around the artist, such as any controversies they may have, or other content unrelated to their actual musical content.

We define a class, *WikipediaParallelSearcher* which wraps around the Wikipedia api available from the **Wikipediaapi** python package. This class handles searching the Wikipedia api for a list of given inputs, and returns the output. Internally it uses a *ThreadPoolExecutor* to make 10 fetches in parallel.

```
class WikipediaParallelSearcher:
  def fetch_all(self, queries)
```

After extracting the first column from **artists.csv** and running the *WikipediaParallelFetcher* to fetch all Wikipedia summaries, we save the output.

2.3 Data Cleaning

Upon examination of the Wikipedia summaries, a few issues become evident:

1. Certain artist names do not yield a Wikipedia summary, or yield an incorrect Wikipedia summary, due to an ambiguous name, e.g. Queen.

2. Certain artist names do not have a Wikipedia entry.

Due to the sheer size of the dataset (3k artists), we take a programmatic approach to evaluating the accuracy of the content of the summaries, and identify a list of artists that need to be corrected.

Entries with a missing Wikipedia summary are easy to detect - we simply check if the summary is equal to the empty string. For this initial experiment, any of these artists are discarded. The justification is that an artist who does not yet have a Wikipedia summary is not likely to be an extremely popular artist, so for the purposes of an MVP we accept the possibility that a user's artist they wish to input may not be present. Another reason why

this makes sense is that I feel it does not make sense to combine text from different sources. Wikipedia content has a certain style, and if we were to outsource content gathering to other sources when Wikipedia is unable to provide content, we would be introducing another variable that could affect our results. For these reasons and the sake of simplicity, we eliminate any artist who does not have a Wikipedia summary.

There are two different flavors of incorrect Wikipedia summaries for an artist. The first possibility is that there exists a Wikipedia page for the artist's name as listed in the dataset, but that it refers to something other than the artist. For example, the Wikipedia page for Heart refers to the organ, rather than the band. The other possibility is that the search term yields a page with suggested results. For example, the search results for Queen yield the following page, in Figure 2: This page contains a list of possible pages that the query may

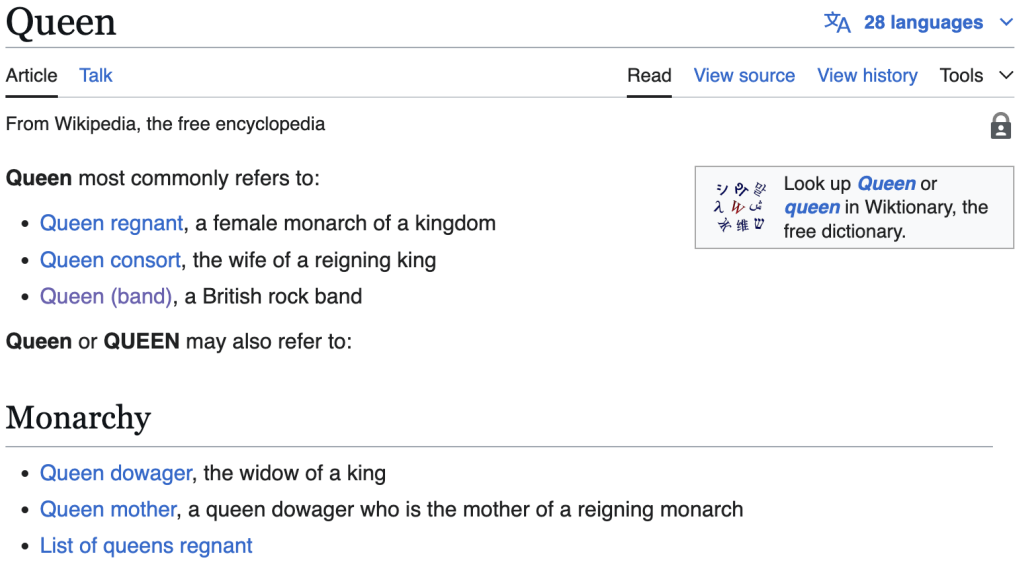

Figure 2: Wikipedia results for Queen

refer to, and our raw fetched summary contains this text, which of course is not a summary of the band Queen.

To combat these scenarios, we employ the following logic: any artist whose search yields a list of possible results rather than a summary will contain some variation of the phrase "may refer to:" or "may also refer to:", which was observed by viewing some sample results. By programatically checking each individual summary, we determine that any summary containing one of these two phrases is likely ambiguous, and we save that artist name for later disambiguation.

To combat the scenario of incorrect summaries (but not ambiguous), we employ the logic that a correct summary of a musical artist is likely to contain one of the following terms: music, singer, songwriter, rapper, band. We progamatically check each summary, and if the summary contains none of these predetermined keywords, we classify it as likely not sum-

marizing the artist, and save the artist for later disambiguation.

Admittedly, a robust process for disambiguating a large number of artist names should require some programmatic search of Wikipedia results. A possible long term solution for this would be to collect the top n page results for the artist name in Wikipedia, and execute the same logic as above, to find the summary we believe to be accurate. However, for the sake of time, I decided to simply pick out a section of the ambiguous artist names and manually resolve these to their correct Wikipedia page, and discard the others. If the end goal of this project was a robust product to be used in production, I would have begun this process and tweaked it as necessary. But since the purpose of this project was simply for fun and to experiment with some NLP techniques, I employed the manual approach and discarded many of the ambiguous artist names that I myself did not recognize. As the chief and possibly only user of this system I decided the effort required to build a truly robust solution to this issue was not worth the amount of effort required.

After mapping a subset of the ambiguous artist names to their correct Wikipedia URLs, the *WikipediaParellelFetcher* was once again utilized to fetch summaries and saved.

The end result after disambiguation was saved in a JSON file called **name-to-summary.json**. This file exists as essentially a large dictionary, where keys of the file are artist names, and values are their summaries. This file serves as the main starting point for the NLP portion of this project.

3 Modeling

Now that we have a list of artist-summary pairs in **name-to-summary.json**, we can begin experimentation with various models for making artist recommendations. As a reminder, the goal is given one or many input artists (though the initial MVP will only accept a single artist as input and later can be expanded to accept multiple), determine a list of the most similar artists. We can experiment with different models for computing the similarity between two artist summaries.

3.1 Model A: TF-IDF Vectorization

3.1.1 Background

As an initial simple implementation, we use a TF-IDF vectorizer to transform each artist summary into a vector, and then use cosine similarity as a metric of similarity between any two vectors.

As a reminder, TF-IDF is composed of the term frequency and the inverse document frequency. The term frequency of term t in document d is defined as:

$$TF_t = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}}$$

where $f_{t,d}$ denotes the number of times term t appears in document d. The inverse document frequency of term t is defined as:

$$\log \frac{N}{|\{d : d \in D \text{ and } t \in d\}|}$$

where D is the set of documents, N is the number of documents (ie $N = |D|$), and the denominator is the number of documents which contains the term t. It is also common to add 1 to both the numerator and denominator to avoid divide-by-zero errors for some term not in the corpus.

3.1.2 Code

We use the TF-IDF vectorizer and cosine similarity measure implemented by the python package **sklearn**:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
```

From there, we pass in the summaries to the vectorizer and compute a similarity matrix. An entry in this matrix, SIM_{ij} denotes the cosine similarity between summary i and j, a float between 0 and 1.

```
vectorizer = TfidfVectorizer()
fitCorpus = vectorizer.fit_transform(summaries)
simMatrix = cosine_similarity(fitCorpus)
```

We also save another JSON file **artist-indices.json** which maps an artist name to their index in the similarity matrix, and save the contents of the similarity matrix for unpacking in the frontend.

3.1.3 Computing Matches

Recall the goal of this project, which is to provide NLP based recommendations for musicians based on musicians that the user likes. We briefly discuss the mechanics of this.

Given an input artist, we can look up the similarity vector from the similarity matrix corresponding to that artist, by keeping track of what index each artist corresponds to. By slicing the similarity matrix, at any given index we have a vector of similarity scores, which we can denote SIM_i. We previously said that SIM_{ij} denotes the similarity between documents i and j, so to find the best matches for artist i, we simply sort the vector of similarity scores SIM_i and report the top results.

To extrapolate this process to handle multiple input artists, the goal is still to report the top n matches, but of course each input artist has its own similarity vector. Instead of reporting the artists with the top similarity to one input artist, we now expand this to report the artists with the top **average** similarity across each of the input artists.

Formally, for some set of k input artists, we define the similarity of some potential match m as:

$$SIM_m = \frac{1}{k} \sum_{i \in 1..k} SIM_{mi}$$

3.1.4 Performance

The TF-IDF model provides mixed results from manual evaluation. While some results are arguably reasonable, others are strange. We examine the results for a few inputs:

The Beatles - The system recommends the following artists given an input of The Beatles, in order of decreasing similarity: Paul McCartney, John Lennon, Rolling Stones, The Police, The Beach Boys. These recommendations make sense, as Paul McCartney and John Lennon of course were members of the Beatles. The Rolling Stones and The Beach Boys had some similarities with the Beatles, but were also similar in terms of time period. The Police is a bit of an outlier here, as they were most popular around 10 years after the Beatles had already disbanded, and had a noticeable different musical style, though they both made music considered by many to be rock.

Korn - The system recommends Poison, Selena Gomez, Aerosmith, REM, and Dropkick Murphys, all of which are not likely to be considered good recommendations. None of the recommended artists' music is a similar style to that of Korn.

Elvis Presley - The system recommends: Justin Bieber, Brenda Lee, Michael Jackson, Phil Collins, and Elton John. The only reasonable recommendation here I would consider to be Brenda Lee, as at least she was from a similar time period. The other artists share little in terms of musical similarity.

Kenny Chesney - The system recommends Luke Combs, Tim McGraw, Dolly Parton, Sam Hunt, and Keith Urban. These are all reasonable recommendations, as they all are country artists as is Kenny Chesney.

3.2 Model B:

As a second model implementation, we pivot to another method of vectorization, using pretrained word embeddings. We use spacy's en_core_web_lg model to generate embedding vectors for our Wikipedia summaries, and continue using cosine similarity as our similarity metric. Spacy offers a variety of models of differing sizes, but after experimentation we use the large model as it provided the best results.

3.2.1 Code

We call the spacy model on each document, and save the vectors. This makes up our embedding matrix, and from there we can use the cosine similarity function from **sklearn** once again to get our similarity matrix.

```
import spacy
nlp = spacy.load("en_core_web_lg")
embeddingMatrix = [nlp(doc).vector for doc in docs]
similarityMatrix = cosine_similarity(embeddingMatrix)
```

3.2.2 Performance

The model using word embeddings seems to provide slightly better recommendations than the TF-IDF model, though it is still prone to odd recommendations at times. We reexamine the same examples as with the TF-IDF model.

The Beatles - The system recommends The Beach Boys, The Doors, REM, Bee Gees, and Pearl Jam. Overall these seem slightly less reasonable than those provided by the TF-IDF model, though one could argue they are more interesting because recommending members of the band is a bit obvious.

Korn - The system recommends Nine Inch Nails, Rascal Flatts, Disturbed, Depeche Mode, and Bruce Springsteen. Two of these are good recommendations (Nine Inch Nails and Disturbed) as they are part of the same genre (1990s and 2000s rock and metal), two of them I would consider to be poor (Bruce Springsteen and Rascal Flatts), and the Depeche Mode recommendation is a bit interesting. They are an electronic band, and Korn does possess some electronic influence in their music, so I would classify this as a reasonable recommendation. Overall this set is probably more reasonable than that of the TF-IDF model, which provided little useful recommendations for Korn.

Elvis Presley - The system recommends Michael Jackson, Miles Davis, Elton John, Bing Crosby, and Prince. This set of recommendations is not entirely useful, and not necessarily better or worse than those of the TF-IDF model. Bing Crosby and Miles Davis could be reasonable recommendations, but the others are certainly of a different genre.

Kenny Chesney - The system recommends Tim McGraw, Jordin Sparks, Alan Jackson, James Blunt, and Lorde. Tim McGraw and Alan Jackson are also country music artists as is Kenny Chesney, and the other three are not great recommendations. Overall this test seems to have regressed from the TF-IDF model.

4 Conclusion

Overall this system has yielded some promising results. In many cases, it recommends artists in similar genres and of similar time periods. However, the system is not perfect, nor should it be expected to be. I would argue there is only so much similarity to be learned from Wikipedia summaries and there are always going to be issues with using a text based system to recommend non-text based content. Musical similarity can transcend a biography and is always subjective - what one person thinks is similar to an artist someone else may not. Furthermore, similarity is not necessarily an accurate metric for what a user might like. For example, my personal music taste spans a wide variety of genres consisting of groups that

would not be considered similar by any means, and on the other hand, there are certain groups within a similar genre to a group I enjoy that I may not be a fan of.

Despite the subjective nature of this problem, I think the system offers some interesting results. A possible future direction would be an extension to consume other types of text. How do the results change if we examine the full Wikipedia page? Perhaps the content that goes into further detail about a musician's career mentioning more details about their musical style can yield smarter recommendations. Could the approach be shifted more drastically, and attempt to provide recommendations based on actual audio content of an artist, or use a combination of text and audio to provide recommendations using multi-modal data? Can the system incorporate user feedback, which can help formalize and quantify an evaluation metric? Currently the system was evaluated manually through my own interpretation of its recommendations, but a more formal approach would be necessary to bring this project to the next level. Perhaps when presented with recommendations, users could be provided with a like/dislike system to grade their perceived accuracy of those recommendations, which can be used to fine tune the recommendation system through a reinforcement learning approach. Or maybe the system could be improved with a clustering approach, where we use k-means or another method to partition the artists into clusters, and then recommend artists within the same cluster. This system provides a solid starting point for further investigation and experimentation with a wide variety of possible future ideas.