Ryan McNeil

Golvis Tavarez

CSC84040 Final Project Report

**1. Introduction and Background**

1.1 Research Question

Song recommendation engines are integral to modern audio streaming platforms, shaping how users discover and engage with music. While much of the current research aims to refine existing methods, we propose a novel approach and present a hypothetical pipeline for combining methods to produce a more effective recommendation system. Prior methods rely on characteristics of the audio focused on instrumentation (like tempo, pitch, and timbre). We posit that *lyrics* represent a non-insignificant proportion of the meaningful “content” and appeal of songs. Especially for lyric-heavy genres (like rap or folk), songs with similar instrumentalization may have very different meaning to listeners, and thus very different appeal. Further, we propose that a meta-model combining multiple different types of recommendation models may be more effective than a singular recommendation model, and propose a hypothetical design.

1.2 Collaborative Filtering

Traditional recommendation methods include collaborative filtering and content-based filtering. Collaborative filtering can be user-based or content-based. User-based collaborative filtering recommends items to a user based on the preferences of other users determined to be ‘similar’ to them. These systems identify users with similar preferences and then make recommendations by suggesting items liked by similar users but not yet interacted with by the given user. Content-based collaborative filtering looks at the similarity between the ‘content’ of items, recommending items similar to items already interacted with or preferred by a given user. [1].

1.3 CNNs

Recent advances in recommender models have incorporated deep learning methods. Convolutional Neural Networks (CNNs) have proved especially powerful for analyzing audio data, demonstrating an ability to capture complex patterns in audio data. CNNs were originally designed for *image* processing, but they have since been adapted for audio analysis. CNNs take tensor input data (like a 1D array for raw audio, or a 2D matrix for grayscale images) and convolves it with a filter. This filter is a smaller matrix of weights that iterates over positions in the input that represent local features like edges or frequencies. CNNs can be designed to convolve an arbitrary number of dimensions. 1-dimentional CNNs (1D CNNs) are used for 1-dimensional inputs like raw audio, 2D CNNs are used for 2-dimensional inputs like images (or spectrograms), and 3D CNNs can be used for video data. Typically, audio data is converted to a spectrogram, which encodes information about spectrum of frequencies over time as an image, with frequency (Y) over time (x), and pixel values for decibels (relative to full scale). This image data is then fed as input to a 2-dimensional CNN. This method of using 2D CNNs on audio spectrograms has proven more powerful than the alternative of using 1D CNNs directly on audio data. Through convolutions, these 2D CNNs can extract features like pitch and timber and have proved very useful for a variety of audio processing tasks. These kinds of models are employed by services like Spotify to compare audio tracks for the purpose of recommending songs to users. [2, 3].

1.4 Text Analysis

Natural Language Processing (NLP) is a field of AI focused on using AI methods to process and understand human language. NLP has seen a boom of research and interest amid the broader AI revolution, and has been revolutionized by breakthroughs in transformer architectures (like BERT and GPT). NLP is being applied to a range of tasks including chatbots, sentiment analysis, and autotranslation. We consider to major methods in NLP: Term Frequency-Inverse Document Frequency (TF-IDF), and Encoder models.

TF-IDF is a statistical method for measuring the relevance of a given word to a document in a corpus. For comparing documents via TF-IDF, each document can be converted to a vector where each dimension corresponds to a word in the total vocabulary of the corpus. Then, a similarity matrix can be calculated comparing these vectors. This is typically done using cosine similarity, a slight modification of vector cosine distance (cosine similarity = 1 – cosine distance). TF-IDF only considers words as spelled, and does not account for context or meaning. The presence of a word with multiple meanings, like “bark”, can contribute to the similarity of two documents even if one document is talking about dogs and the other is taking about trees. [4].

Neural network encoders (or encoder models) use embeddings to represent sentences as dense, fixed-size vectors. These embeddings capture semantic and contextual meaning in a way that methods like TF-IDF cannot. Encoder models use a transformer architecture focused on processing entire sequences of words to understand contextual relationships. These models are trained on vast bodies of textual data, and prove very powerful for text analysis. [5].

**2.1 COLLABORATIVE FILTERING MODEL**

We opt for an item-based approach using the Spotify Tracks Dataset [6]. This dataset contains records with information like track name, artist name, genre, tempo, key (encoded), mode (encoded), and track length, as well as abstract features devised by Spotify representing qualities like “danceability” and “energy” on a 0 to 1 scale, popularity on a 1 to 100 scale, a ‘loudness’ variable derived from average decibels. Using this data, a similarity matrix is calculated using cosine similarity Recommendations are then made by taking the top k songs by cosine similarity to a given input song.

**3. LYRICS RECOMMENDATION MODEL**

3.1 TF-IDF model

We present two models for a potential lyric recommendation engine. First we utilized TF-IDF. We begin with a song lyrics dataset found on Kaggle [7]. Songs are indexed by song & artist. For each song the full lyrics represent one “document”. For preprocessing, we remove songs that are essentially duplicates, things like lives, covers, and edits. These documents are TF-IDF vectorized, ignoring stopwords, which include vocalizations like “la” and “da”. We then calculate cosine similarities to build a similarity matrix. For a given song, the recommender takes the vector of similarities for that song, sort by similarity score, and returns the top k songs by similarity score as recommendations.

3.2 Encoder model

For our encoder model, we preprocess similarly. Again, we consider the lyrics for each song as a document. We then use sentence-BERT to encode the lyrics data to embeddings (and convert those to tensors) and collect these into a matrix. We then build a similarity matrix using cosine similarity. Last, as with the TF-IDF method, for a given song, we take the vector of similarities for that song, sort by similarity score and return the top k songs by similarity score as recommendations. We observe that the songs recommended by the encoder model seem to have higher similarity scores than those recommended by the TF-IDF model. Whereas the TF-IDF model tends to find maximum similarity around 0.25, the encoder model often provides songs of similarity of around 0.5. This suggests to us that the encoder model is successfully encoding more information that can be used for comparison than our TF-IDF model. Testing revealed that the encoder more often recommends songs by the same artist than the TF-IDF model, further suggesting to us that the model is successfully recommending truly similar songs.

3.3 Considerations

Spotify reports a total song base of around 100,000,000 songs [8]. Assuming a moderate embedding vector of size 768, at 4 bytes per value, the total size of the embedding matrix would be around 100,000,000 \* 768 \* 4 = 307.2 GB. This size is well within the capabilities of modern commercial servers, where configurations with 512GB or more of RAM are increasingly common. However, the *similarity matrix* would take up 100,000,0002 \* 4 bytes = 40PB of data, making it unreasonable to simply calculate and store this matrix. As a consequence, we propose several options for working around this and include one in our meta-model design.

The solution we choose is, rather than computing the entire similarity matrix for all songs, to compute the similarity matrix for only a subset of songs returned by a different recommendation model. Another option is to leverage hierarchical navigable small world (HNSW), an approximate nearest neighbor algorithm designed for use on vector database datasets sufficiently large to make computing the distance from a given query to every other point in the database computationally prohibitive. This would allow returning recommendations with similarity scores (in this case, distances) without having to run through tens of millions of records. Given the size of such a dataset, it may be reasonable to shrink similarities below a high threshold to zero, and then store values in a sparse matrix. Further, if feature values can be simplified, we could switch from 32-bit floats (4 bytes each) to 8-bit integers (1 byte each). Lastly, we suggest that any sufficiently large dataset could be stored and manipulated by employing RDDs.

**4. META-MODEL**

A purple rectangular box with white text

Description automatically generated

Fig 1. Meta-model pipeline

We propose a hypothetical recommendation engine design combining three methods: a 2D CNN, collaborative filtering, and lyrics-based recommendation. First, recommendations would be calculated via a 2D CNN and by a collaborative filtering model independently. Being that a 2D CNN would already be recommending based on qualities of the audio data, *user-based* collaborative filtering may be most appropriate to maximally diversify recommendation methods in our meta model. For a value *n* much larger than the final number of songs we want our meta-model to recommend, obtain the set of top *n* songs by similarity, *RCNN*, from the 2D CNN model, along with their similarity scores. Here we assume that the CNN is extracting song features into a vector, which can be used to compute similarities via cosine similarity, Euclidean distance, or some other metric. If this metric is not inherently between 0 and 1, normalize this value to between 0 and 1. Next, obtain the set of top *k* songs by similarity, *RCF*, from the collaborative filtering model. Combine these to form RCNN∪RCF, including both models’ similarity scores. Weight the similarity scores for songs in RCNN∩RCF to privilege them for appearing in both models’ recommendations. Pass RCNN∪RCFto the lyrics recommendation engine where we retrieve the embedding matrix and calculate a similarity matrix for *only* these songs. Append the lyrics-based model’s similarity scores for each song in RCNN∪RCF. To obtain final similarity scores, we can simply average (given that all scores should be on the same scale of 0 to 1). If we want to privilege one model’s predictions over another, we can weight that model’s predictions prior to averaging. Using the combined similarity scores, we could sort by score and return the top *k* songs by score as the model’s final recommendations. A period of beta testing would be useful for finetuning weights based on user feedback.

As proof-of-concept for our lyric recommendation system and as a starting point for a hypothetical meta-model, we have produced a combined class incorporating all three of the model types we have built (a collaborative-filtering model, a TF-IDF lyrics-based model, and an encoder lyrics-based model) along with some scaffolding for further functionality (like weighting scores). All are available on GitHub [9, 10].

WORKS CITED

1Murel, J. PhD. (2024). What is collaborative filtering?. *IBM*. <https://www.ibm.com/think/topics/collaborative-filtering>

2What are convolutional neural networks? (2024). *IBM*. <https://www.ibm.com/think/topics/convolutional-neural-networks>

3Gaffar, S. (2022). Guide to Audio Classification Using Deep Learning. *Analytics Vidhya*. <https://www.analyticsvidhya.com/blog/2022/04/guide-to-audio-classification-using-deep-learning/>

4Saha, R. (2023). Understanding TF-IDF (Term Frequency-Inverse Documents Frequency). *GeeksforGeeks*. <https://www.geeksforgeeks.org/understanding-tf-idf-term-frequency-inverse-document-frequency/>

5Encoders in NLP. (2022). *Everyday Series*. <https://everydayseries.com/encoders-in-nlp/>

6Pandya, M. (2022).Spotify Tracks Dataset [Dataset]. *Kaggle*. <https://doi.org/10.34740/KAGGLE/DSV/4372070>

7Shah, D. (2018). Song Lyrics Dataset [Dataset]. *Kaggle*. <https://www.Kaggle.com/datasets/deepshah16/song-lyrics-dataset/>

8Spotify. (2024). Company Info. *Spotify Newsroom*. <https://newsroom.spotify.com/company-info/>

9Tavarez, G. (2024). csc84040\_final\_v2 [Github Repo]. <https://github.com/gtava5813/csc84040_final_v2/tree/main>

10McNeil, R. (2024). CSC84040\_Final\_Project [Github Repo]. <https://github.com/Disco-Gnome/CSC84040_Final_Project/tree/main>