Abstract

In this project, I build a music recommendation engine that takes in a song and outputs similar songs and artists. I implemented two of the more popular methods in the recommendation engines of today: collaborative filtering and content-based filtering. Collaborative filtering makes recommendations from user preferences; if user A listens to very similar songs as user B, user A will probably like the other songs user B listens to. Content-based filtering makes recommendations from the audio features of songs. For this project, I decided to look at the decibel levels at each frequency for every second of the song. Different genres of music tend to show a difference in decibel levels at various frequency ranges. For example, classical music won’t fill out the really high frequencies, and hip-hop will fill out the whole range with an emphasis on the lower part of the spectrum.

The engine consolidates the results of a collaborative filtering (CF) component and a content-based filtering component. I used a k-Nearest Neighbor model to recommend similar songs and artists. There’s not really a way to objectively test the results from the CF component, but most of the recommended songs and artists are reasonable suggestions. The content-based filtering component only recommends similar songs, and it relies on a convolutional neural network to find patterns in audio features across different genres; essentially, the neural network builds an n-dimensional genre space, where n is the number of genres you want to classify. Songs that are similar to each other will lie closer in this n-dimensional space. The model achieved a validation accuracy of 0.82750, a loss of 0.3345, and a 0.6075 testing accuracy. The model ended up being especially effective at classifying classical music and was the worst at classifying hip-hop music.
1 Introduction

Music streaming services now generate about half of the revenue in the global music industry market. As companies such as Spotify and Apple compete for consumers, they have had to develop features that differentiate their products from others. One major initiative was to find new songs that fit a user’s tastes. In 2015, Spotify released its "Discover Weekly" feature - a playlist of new songs that are similar to the user’s preferences. Discover Weekly is run by a recommendation engine, which relies on large amounts of data and has been a critical component of many online platforms such as Netflix, Amazon, and YouTube to ensure that their users continue to spend time on their products. There are many different kinds of recommendation engines that improve the experience of the user in different ways. In this project, I build a music recommendation engine that focuses on finding similarities in user behavior and the audio features of songs.

2 The Recommendation Engine

The music recommendation that Spotify uses has three main components: collaborative filtering, content-based filtering, and natural language processing (NLP). I decided to only implement collaborative filtering and content-based filtering components for my engine and not the NLP component because I don’t have song lyric data.

2.1 Collaborative Filtering

Collaborative filtering (CF) is a method of predicting a user’s preferences by collecting and processing the preferences and behavior of many users. CF essentially makes the assumption that if two people have similar music choices, whether that means they listen to similar songs, genres, artists, then chances are each person will like the songs that they haven’t heard yet but are well-received by the other person. There are two types of collaborative filtering: item-based and action-based. In item-based CF, it builds recommendations based on behavior of users who look at a particular item. For example, if I listen to the song Electric by Alina Baraz, then item-based CF will go through other users who listened to Electric and then look at the other songs those users have listened to. The CF in this engine was item-based.

The data I used for CF came from a dataset collected by the Music Technology Group of Pompeu Fabra University in Spain. They released data from Last.fm that contains profile data of 360,000+ users and song metadata. It has information like the number of times a user has listened to a song, the total number of times a track has been played, and more.

The first thing I did was process the data. I tried several things, but settled on simply removing songs and artists that haven’t been played very often (for an artist less than 40,000 times and for a song less than 1,000 times) to help reduce the noise. I also filtered out artists not from the United States.
I wanted to build a recommendation engine that could find both similar artists and songs. For similar artists, I constructed a sparse matrix from a matrix where each row essentially describes artist-user interactions. The number of rows is the number of artists, and the number of columns is the number of users. Each row, thus, contains how many times each user has listened to the songs of a given artist.

Similarly, I constructed a sparse matrix from a matrix where each row represents a user-song interaction. Each row has the ID of the user, the number of times that user has listened to a song, the song name, and the number of plays the song has.

From there, I fit two k-Nearest Neighbor models where the metric is cosine similarities. There’s one model for artists and one for songs. Here, I try to find the closest artists to the Red Hot Chili Peppers, an American rock band.

$ python collaborative_filtering_artists.py

Recommendations for red hot chili peppers:
1: john frusciante, with distance of 0.660864827653:
2: incubus, with distance of 0.73614441395:
3: rage against the machine, with distance of 0.780389013343:
4: audioslave, with distance of 0.798709608464:
5: sublime, with distance of 0.809430140516:

These suggestions are pretty good. John Frusciante is the former guitarist of the Red Hot Chili Peppers, and the music of Incubus, Rage Against the Machine, Audioslave, and Sublime can be considered as falling into the category of rock.

I also fit a kNN model using the song matrix to get the songs closest in cosine distance to a given song. These are the five most similar songs to Learn to Fly by the Foo Fighters.

$ python collaborative_filtering_songs.py

Recommendations for Learn To Fly — Foo Fighters:
1: The Pretender — Foo Fighters, with distance of 0.797653088865:
2: Everlong — Foo Fighters, with distance of 0.822280348719:
3: The Middle – Jimmy Eat World, with distance of 0.941457941763:
4: Good Life – Coldplay, with distance of 0.94668443556:
5: Luv Me, Luv Me – Shaggy / Samantha Cole, with distance of 0.948387598532:

The cosine distances here are bit bigger than the distances for the artist model. Nevertheless, the results are still encouraging because there’s two Foo Fighter songs and then songs from Jimmy Eat World and Coldplay. There’s a song from reggae fusion artist Shaggy, someone I’ve never heard of Shaggy.

Although CF generally outperforms other techniques of music recommendation, it is not without its disadvantages, namely the cold start problem (lack of data for a song) and gray sheep users (users who have very atypical tastes won’t receive accurate suggestions). These problems persist in our artist and song models, as recommendations for more unknown songs or artists are not as good as recommendations for more popular ones.

2.2 Content-based Filtering

To address the disadvantages of CF, many recommendation engines also use content-based filtering. Content-based filtering utilizes the musical data of a song and recommends other songs that share similar musical components. From using convolutional neural networks to classifier models, there are a variety of ways that content-based filtering can be implemented.

I chose to implement a convolutional neural network that uses information extracted from the spectrograms of the audio samples. There’s a very useful library called Librosa that builds these spectrograms for you, and gives you a spectrogram that tells you the decibel level at each frequency for every second of the song. The reasoning behind this is that different genres of music will be louder at different frequencies and have different rhythms; for example, hip-hop will fill out the lower end of the spectrum and if you plot out the spectrogram, there will be a noticeable pattern because hip-hop often has a repetitive structure. Classical music, on the other hand, may not have a lot of high-frequency instruments. The differences between two songs, one classical and the other hip-hop, can be seen in Figure 1.
Figure 1: Spectrograms for Hip-hop and Classical Song

From the spectrogram data, I can try to build a neural network that guesses the genre of a song (outputs probability song is hip-hop, rock, etc). Those probabilities can then represent an $n$-dimensional space where $n$ is the number of genres (classes) that you want to be able to classify. Songs that are similar to each other will have a smaller Euclidean distance within this space. Essentially, I am ”breaking down” a song into the probabilities that it falls under different genre, and songs that are more similar to each other will have similar probabilities of falling under the various genres. For example, the probabilities of the genre for a Kanye West song might lead you to conclude that it is hip-hop track, but it might still detect the influence
of pop.

I decided to try and classify four genres: hip-hop, rock, pop, and classical.

In order to build the neural network, I need to find a dataset that had a large number of audio samples of songs, genre labels for those songs, and a variety of genres. I eventually decided to use the GTZAN and Free Music Archive (FMA) datasets. The GTZAN data has been used in many genre classification papers, and contains 1000 30 second audio samples for 10 genres (100 songs for each genre). The FMA data is a collection of 30 second samples of songs from Free Music Archive, a library of legally downloadable songs. There’s four different kinds of FMA datasets that contain increasing numbers of tracks and genres. I decided to use the large dataset that had 106,574 tracks across 161 genres, mostly for storage purposes (the full dataset is 900 GB). Unlike GTZAN, FMA isn’t balanced across genres. The problem with FMA is that their songs are not songs that are popular today. I used the FMA dataset for training and the GTZAN dataset for testing. It is worth noting that the songs in the FMA dataset are by and large songs that are not that well-known.

The GTZAN data was nicely divided so that songs were divided by genre, but the FMA data was a large unfiltered mess of audio samples, and each sample was named by its track ID. There are metadata files that link track ID to genre and track title, so the first thing I did was iterate through the dataset, divide the songs by genre, and rename the file to the actual title of the song. I also stored metadata of all the songs into a song library file.

At the end, once I combined the GTZAN and FMA songs, I had a total of 21,596 30 second samples of songs. That number was not proportionally split into the genres, as I had 1,330 classical, 3,652 hip-hop, 14,282 rock, and 2,432 pop songs. I didn’t want to have an overwhelmingly large number of one particular genre, so I decided to only use 1,000 songs from each genre for my project. To sum things up, my training data set was 3,600 songs from FMA, my validation data set was 400 songs from FMA, and my testing data set was 400 songs from GTZAN.

I heavily relied on Keras to build the neural network. The first neural network I built was a three-layer model of 2D convolutional and pooling layers, followed by a dense and activation layer. It produced very funky results; the loss wasn’t decreasing and the validation accuracy wasn’t changing. I assumed that there was a problem with the data because such a model should at least learn something, but it was just making random guesses all the time. I tried training the model on one data point, and the model did overfit, but when I tried to train on two or three data points, the loss fluctuated a lot between epochs. I still thought it was a problem with my data, so I spent even more more time trying to fix the problem.

Eventually, I gave up and decided to just try a different, more complex model. I then built a neural network that was still three layers, but they were 1D convolutional and pooling layers. I also added other things such as a kernel regularizer, batch normalization, and dropout. Then, I added a LSTM layer with a dropout, two dense layers, and a final activation softmax layer. Finally, I added a callback that would reduce the learning rate if the validation accuracy doesn’t increase for a while. This model ended up working much better, as the loss ended up decreasing and the validation accuracy increased between epochs.
2.3 Consolidation

After building the CF and content-based filtering components, we have to unify their recommendations. I created a program that would take in an audio file named in the format of “[Song Name] - [Artist Name].[extension]”. Without an audio file as input, then we can’t use the content-based filtering component to classify songs that aren’t in the GTZAN/FMA data set because we don’t have their audio features. Once the audio file is given, the models built by these two components churn out song recommendations. There are some caveats to this program. First, because of the nature of our CF model, if the song or artist name is not found in the Last.fm data set, then we don’t have any user data on that particular song and we can’t produce any recommendations using our CF component. Second, the provided song should fall under hip-hop, pop, rock, or classical (or is adjacent to one of those genres) to achieve decent recommendations.

The program then takes the given song and runs both the CF and content-based components to gather five recommended songs from each component. After removing duplicates, a final list of recommended songs will be given to the user. Recommended artists will also be provided to the user if the song is by an artist that is in the Last.fm dataset that I used. Ultimately, a list of at most ten songs and five artists will be given to the user, and this list is not ranked by level of similarity to the target song.

A flowchart diagram of the music recommendation engine is shown in Figure 2.
2.4 Engine Flowchart

Figure 2: Flowchart of the song recommendation engine
3 Discussion

The discussion in this section will center on the content-based component and the convolutional neural network model because there isn’t a great way to objectively analyze the results from the CF component.

After training for 100 epochs, the convolutional neural network produced a model with a validation accuracy of 0.82750 and a loss of 0.3345, as seen in Figures 3 and 4. From Figure 5, the testing accuracy is 0.6075.

![Figure 3: Model accuracy over time](image)
Now, let’s see what the model predicts for some songs in hip-hop, pop, rock, and classical.

<table>
<thead>
<tr>
<th>Hip-hop Song Classifications</th>
<th>Pop</th>
<th>Classical</th>
<th>Hip-hop</th>
<th>Rock</th>
</tr>
</thead>
<tbody>
<tr>
<td>Juicy - The Notorious B.I.G</td>
<td>0.224</td>
<td>0.000373</td>
<td>0.773</td>
<td>0.00268</td>
</tr>
<tr>
<td>Nuthin But a G Thang - Dr. Dre</td>
<td>0.000169</td>
<td>0.000019</td>
<td>0.998</td>
<td>0.000017</td>
</tr>
<tr>
<td>California Love - 2Pac</td>
<td>0.777</td>
<td>0.0793</td>
<td>0.0461</td>
<td>0.098</td>
</tr>
</tbody>
</table>

From these three songs above, we can see that the model does a good job classifying some hip-hop songs like *Nuthin But a G Thang* and *Juicy*. But, it thought *California Love* was more of a pop song than a hip-hop one, and perhaps even more interesting, it thought it was more classical than hip-hop.
Pop Song Classifications

<table>
<thead>
<tr>
<th>Song</th>
<th>Pop</th>
<th>Classical</th>
<th>Hip-hop</th>
<th>Rock</th>
</tr>
</thead>
<tbody>
<tr>
<td>Havana - Camila Cabello</td>
<td>0.969</td>
<td>0.00338</td>
<td>0.00576</td>
<td>0.022</td>
</tr>
<tr>
<td>I Gotta Feeling - The Black Eyed Peas</td>
<td>0.86</td>
<td>0.0605</td>
<td>0.0167</td>
<td>0.0612</td>
</tr>
<tr>
<td>Apologize - Timbaland</td>
<td>0.40628</td>
<td>0.0856</td>
<td>0.32415</td>
<td>0.1840</td>
</tr>
</tbody>
</table>

The model also does a good job for this sample of three songs. *Havana* and *I Gotta Feeling* fall pretty clearly in the pop genre, but *Apologize* is pop with some hip-hop elements to it.

Rock Song Classifications

<table>
<thead>
<tr>
<th>Song</th>
<th>Pop</th>
<th>Classical</th>
<th>Hip-hop</th>
<th>Rock</th>
</tr>
</thead>
<tbody>
<tr>
<td>Highway to Hell - AC/DC</td>
<td>0.114</td>
<td>0.00373</td>
<td>0.865</td>
<td>0.0176</td>
</tr>
<tr>
<td>Sweet Child O’ Mine - Guns N’ Roses</td>
<td>0.931</td>
<td>0.00103</td>
<td>0.00103</td>
<td>0.0666</td>
</tr>
<tr>
<td>Livin’ On a Prayer - Bon Jovi</td>
<td>0.649</td>
<td>0.0126</td>
<td>0.0441</td>
<td>0.294</td>
</tr>
</tbody>
</table>

Here is where things get a little weird. *Highway to Hell* is the only rock song out of these three classic rock songs to be classified as rock. *Sweet Child O’ Mine* and *Livin’ On a Prayer* are both classified as pop. This likely means that songs that should be classified as rock are being classified in other genres like pop. It also means that the model thinks that there are more pop songs than there really is.

Classical Song Classifications

<table>
<thead>
<tr>
<th>Song</th>
<th>Pop</th>
<th>Classical</th>
<th>Hip-hop</th>
<th>Rock</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cello Suite No. 1 - Bach</td>
<td>0.01605</td>
<td>0.965</td>
<td>0.00067</td>
<td>0.0187</td>
</tr>
<tr>
<td>Nocturne in B-flat - Chopin</td>
<td>0.00362</td>
<td>0.991</td>
<td>0.000626</td>
<td>0.00505</td>
</tr>
<tr>
<td>Symphony No. 5 - Beethoven</td>
<td>0.000093</td>
<td>0.998</td>
<td>0.000014</td>
<td>0.00011</td>
</tr>
</tbody>
</table>

Classical songs are classified pretty well, as the model definitively classifies all three of these songs into the classical genre.

From the results above and playing around with other songs, it’s worth doing a little bit more digging as to why songs seem to be classified as pop when they shouldn’t be, and why hip-hop songs aren’t classified well. We can do this by looking at the precision and recall of each genre.
Figure 5: CNN Testing Statistics

Precision is the percentage of positive predictions that were correct, and recall is the percentage of positive examples that the model caught. Looking at the figure above, it is clear that the model did the worse at classifying hip-hop songs, as the precision and recall are low compared to the other genres. The model did very well at classical songs (the genre that’s most different from the other ones). Interestingly, the precision is low for pop and high for rock, while the recall is high for pop and low for rock. This means the model casts a wide net for what it considers to be pop songs and catches a lot of things that are not pop songs. For rock songs, the model casts a narrow net for what it considers to be rock songs, but the majority of the things it catches are indeed rock songs. Why is this happening? Not completely sure, but it could be that the songs categorized into pop in the FMA database have similarities to rock and hip-hop; that is, they don’t only fall into the pop genre.

Finally, let’s take a look at what the full engine recommends for a given song like Learn to Fly by the Foo Fighters (remember that the list isn’t ranked).

$ python song_recommender.py

The recommended songs for Learn to Fly – Foo Fighters are:
1. The Middle – Jimmy Eat World
2. Obama Logs – Great Plains
3. Good Life – OneRepublic
4. Everlong – Foo Fighters
5. Luv Me, Luv Me – Shaggy / Samantha Cole
6. Live at WFMT (Full Set) – Sic Alps
7. Down the Tubes – Falling Down – Cheap Time
8. The Pretender – Foo Fighters
9. The Best of British Luck – Nightingales
10. Van v Art – Necropolis

The recommended artists for Foo Fighters are:
1. far from finished
2. darkbuster
3. street dogs
4. dropkick murphys
5. audioslave

This list is intriguing. It has the same five songs that the CF component suggested earlier on, and five new ones from the content-based filtering component. Because the content-based filtering component relies on FMA music, all of its suggestions are songs that I’ve never heard of. Doing a quick search on some of these songs show that the genre of all of them are rock (can’t find anything on Obama Logs - Great Plains, though).

4 Future Work

One of the biggest weaknesses with my engine is the datasets. Like I mentioned earlier, the songs in the FMA, GTZAN, and even the Last.fm datasets are all kind of out of date and don’t reflect what is most commonly listened to today. Because song recommendation engines should take into account today’s top trends and hits, my engine is lacking in that regard. Ideally, I would re-train my engine on a large collection of popular songs in the last five to ten years.

There are also various improvements that can be made to the engine itself. Let’s start with the collaborative filtering component. First, it is important to remember that the Last.fm dataset of user activity is very, very small compared to the full user activity dataset of companies such as Spotify. I would likely run into scalability issues with how I currently handle things, especially if I try to do real-time computation. In order to improve the scalability of the CF component, I could do some kind of hashing or caching. Second, we still face the cold start (not enough data for song/artist) and gray sheep problems (unusual activity by a user) mentioned earlier. The cold start problem is a little hard to fix (hence the need for the content-based filtering component), but the gray sheep problem could be addressed by mapping all users into a some kind of user-behavior space and then finding the most similar users (kind of like a kNN model but for users). Then, I would use the activity from those users, who still might be quite different than the gray sheep user but will at least provide us something.

Likewise, the content-based filtering component of the engine can be improved. There is the problem of
data, as the songs in the GTZAN/FMA datasets are not very up-to-date. Moreover, I can play around more with the various parameters (batch size, number of layers, filters, epochs, LSTM layers, dense layers) to see if we can achieve a better model. Because I only implemented genre classification for 4 genres (hip-hop, rock, pop, classical), the engine is missing out on a lot of different genres of music and our CNN model won’t do well at generating recommendations for music that are not adjacent to one of those four genres. But, it’s not a good idea to try and classify all genres of music, because the dimensionality of the genre space will be far too high and nothing will be classified well. There probably is an optimal number of genres where the dimensionality is not too high and the genre space can represent a wide range of genres.

5 Acknowledgements

I would like to thank my advisor Scott Petersen, who has been a great source of help and encouragement throughout this project. I thoroughly enjoyed the conversations we’ve had about things related to not just my project but technology and the world as well.

I would also like to thank Yale for an incredible four years and a place to call home. I have been able to grow so much as a student and a person, and I will forever appreciate my time here.