

Connecting Songs and Movies: Spotiflix, the Movie Recommender

Mert Yapucuoglu
University of Oregon
1585 E 13th Ave.
Eugene, OR
merty@uoregon.edu

Ahmet Turan Bulut
University of Oregon
1585 E 13th Ave.
Eugene, OR
turanb@uoregon.edu

Murad Mikayilzade
University of Oregon
1585 E 13th Ave.
Eugene, OR
muradm@uoregon.edu

1. Introduction

This study aims to address the challenge individuals face in selecting movies that match their current mood. Although people have access to a vast and constantly evolving selection of movies, choosing a movie that complements one's current state of mind can be a daunting task. Unlike music, which can be a quick and effortless way to enhance one's mood, movies require significant time and effort investment. The goal of this study is to develop a movie recommender system that connects two distinct forms of media, namely music and movies, to recommend movies based on a user's input playlist or song.

Movies are a popular form of entertainment that can transport viewers to a different reality and provide a temporary escape from everyday stressors. However, the process of selecting a movie that matches one's current mood can be challenging. On the other hand, music has been recognized as a powerful tool for setting the tone and mood for any occasion. With the emergence of music streaming services, people have access to an extensive library of music that can cater to their moods. By bridging the gap between music and movies, the proposed recommender system aims to provide users with a more personalized and enjoyable entertainment experience.

The movie recommender system will utilize various techniques, including natural language processing, sentiment analysis, and machine learning, to analyze the lyrics and musical components of a song or playlist and determine the user's mood. Subsequently, the system will recommend movies that are most likely to align with the user's current state of mind. These recommendations may range from heart-warming comedies and romantic dramas to action-packed thrillers and suspenseful movies.

This study aims to contribute to the existing body of knowledge on recommender systems and personalized entertainment experiences. The proposed system has the potential to transform the way individuals select movies for leisure by leveraging the power of music. By providing users

with more intuitive and personalized movie selection experiences, the proposed system may enhance user satisfaction and improve the overall entertainment experience.

1.1. Background and related work

There are multiple codebases and blog posts that go into detail about audio analysis using the very methods we use in this paper. Audio features and sentiment analysis are separate and involved fields of their own with ongoing research and innovations. However, there is no app or publicly available codebase or guide for movie recommendations from audio. It is our conclusion from our research that the connection between these two forms of media is not that established in the machine learning field, and is waiting to be explored by the combined use of methods of each respective medium.

1.2. Approach

To tackle this problem, we utilized three different approaches, NLP sentiment analysis, Spotify audio features, and mel frequency spectrograms extracted from song waveforms. We trained three models using the results of these approaches and automated the recommendation process by matching the respective values generated by the models to clusters of movies.

For our first approach, we used unsupervised 3-dimensional k-means clustering to separately group songs and movie soundtracks using their Spotify audio features as parameters. We then matched the song clusters with movie soundtrack clusters using their average feature values and created a translation table from song clusters to movie soundtrack clusters. Taking Spotify features as input and the cluster number of songs as labels, we trained a model to predict the cluster of a song given Spotify features. This way, by just the name of a song, audio features can be used to predict a song cluster, which is matched to a movie cluster, which at the end contains the movies that are recommended for that song.

Secondly, we used NLP sentiment analysis on song lyrics and movie scripts to generate a positiveness score for each of the items. We then trained a model using Spotify audio features of the songs and movies as inputs and their positivity score as the label. As a result, we can get the positivity score of any song by just the name and can recommend movies with similar scores.

As our last method, we used mel spectrograms to extract Spotify features from waveform files. By transforming mp3 files to frequency parameters via a series of processes, we used these parameters as input and Spotify audio features as labels to train a model. As a result, we can predict the audio features of a song given an mp3 and can use our first or second models to generate movie recommendations.

2. Details of Approach

Our project made use of three different ways of classifying audio, NLP Sentiment Analysis of written text, Spotify Audio Features, and mel spectrograms extracted from audio waveforms. We will be explaining the steps we took for each approach.

2.1. NLP Sentiment Analysis

The sentiment analysis method that we developed aimed to assess the overall positiveness of both music lyrics and movie scripts. To achieve this, we first needed to gather datasets of movie scripts and songs. After some research, we managed to obtain a dataset of nearly 2800 movie scripts, which was a great starting point for our analysis.

For songs, we had to be a bit more creative in our approach. We wrote a code that could extract song information from a Spotify playlist and then used the Genius API to obtain the lyrics for each song. This method allowed us to gather data on close to 6000 songs, which gave us a significant amount of information to work with.

Once we had our datasets ready, we used the Natural Language Toolkit's (NLTK) VADER sentiment analysis tool to analyze the sentiment of each movie script and song. VADER is a lexicon and rule-based sentiment analysis tool that uses a combination of linguistic rules and heuristics to determine the sentiment of a piece of text. It is particularly effective for analyzing social media posts and other short-form text which worked perfectly for what we wanted to do..

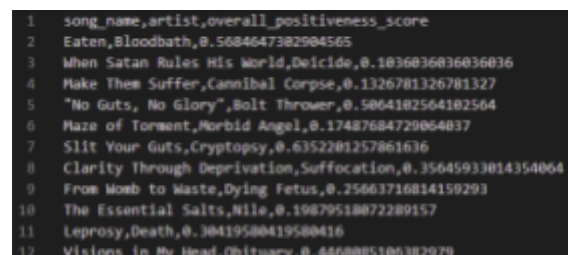
After running each script and song through VADER, we were able to generate an overall positivity score for each one. We then labeled our datasets using these scores, with the

overall_positiveness_score serving as the label for both the movie scripts and the songs.

Overall, our sentiment analysis method provided a comprehensive way to assess the positivity of both music lyrics and movie scripts. By utilizing NLTK's VADER sentiment analysis tool, we were able to quickly and accurately analyze large amounts of text data and generate meaningful insights into the emotional tone of each piece.

Here is the pseudocode for the song_analyzer.py:

```
# Set up the credentials for spotify and genius api
# Create an array of playlist ids
# Loop over the playlist ids
    # Initialize the sentiment analysis model
    # Initialize Output CSV file path
    # Check if the output CSV file already exists
    # Initialize the CSV writer
        # If the file doesn't exist, write the header
        # Get the playlist information from Spotify
        # Loop through each track in the playlist
        # Get the song name and the artist name
        # Getting the lyrics from GENIUS API
        # Calculate polarity and positiveness scores
        # Write the results to the CSV file
```



song_name	artist	overall_positiveness_score
Eaten	Bloodbath	0.5684647382904565
When Satan Rules His World	Deicide	0.1836836836836836
Make Them Suffer	Cannibal Corpse	0.1326781326781327
"No Guts, No Glory"	Bolt Thrower	0.5864182564182564
Maze of Torment	Morbid Angel	0.17487684729064837
Slit Your Guts	Cryptopsy	0.6352281257861636
Clarity Through Deprivation	Suffocation	0.35645933014354064
From Womb to Waste	Dying Fetus	0.25663716814159293
The Essential Salts	Nile	0.19879518872289157
Leprosy	Death	0.38419588419588416
Visions in My Head	Obituary	0.446888186382979

fig 1. output of the song_analyzer.py

Here is the pseudocode for the movie_analyzer.py:

```
# Initialize the sentiment analyzer
# Initialize the path to the directory containing the .txt script files
# Initialize Output CSV file path
# Initialize the CSV writer
# Loop through each .txt file in the directory
# Construct the file path
# Read the script file
# Perform sentiment analysis
# Write the results to the CSV file
```

```

1 movie_file_name,overall_positiveness_score
2 file_1.txt,0.5797181449275363
3 file_10.txt,0.6666666666666667
4 file_100.txt,0.5983935742971888
5 file_1000.txt,0.5209580838323353
6 file_1001.txt,0.46768860836501896
7 file_1002.txt,0.4010416666666667
8 file_1003.txt,0.4351851851851852
9 file_1004.txt,0.6201117318435755
10 file_1005.txt,0.5408163265306122
11 file_1006.txt,0.3644859813084112
12 file_1007.txt,0.478021978021978
13 file_1008.txt,0.5282051282051281
14 file_1009.txt,0.5517241379310345
15 file_101.txt,0.5000000000000001
16 file_1010.txt,0.543778801843318
17 file_1011.txt,0.5256410256410257
18 file_1012.txt,0.5284090909090909
19 file_1013.txt,0.4429223744292238
20 file_1014.txt,0.529126213592233
21 file_1015.txt,0.5407725321888412
22 file_1016.txt,0.3888888888888889
23 file_1017.txt,0.489247311827957

```

fig 2. output of the movie_analyzer.py

After this, we used the Spotify API to acquire the Spotify audio feature values of the songs that we processed with VADER sentiment analysis tool. Then we merged the overall positiveness score of the songs and their Spotify features in one place, preparing for the model. The model we trained for this section takes in the list of Spotify audio features of a song as input and learns to predict its positiveness score supervised by the values generated by the VADER. The code, training, and results of this model are in the file `Spotifyflix-labeltosentiment.ipynb`, and the model is called `LabelToPositivityModel`. The pseudocode for its layer is as follows:

```
# Linear(7, 1)
```

We are well aware that this doesn't look like the most complicated model. However, after trying many models with different depths and widths such as the one below,

```
# Linear(7, 512)
# Linear(512, 256)
# Linear(256, 128)
# Linear(128, 1)
```

we arrived at the conclusion that in this case, the simplest is the best. For our optimizer, we chose Adam, with a learning rate of 0.001. We used `MSELoss` as our loss function since our result was a continuous value between 0 and 1.

For the last step of the recommendation, we will use our trained model to predict a positiveness score from Spotify features. Then we will return the 5 movies with the closest positiveness score from the movie dataset, finishing the process.

2.2. Spotify Audio Features

Our method for this recommendation is to cluster songs into groups where they have similar Spotify audio feature values, and do the same for movies using their soundtracks. Then we will use the average Spotify audio feature values of each song cluster to match them to a movie cluster using the its average Spotify audio feature values of the movie soundtracks in it. Then, we will train a model which can predict the song cluster of a song given its Spotify audio features, and then using this cluster prediction we can recommend the movies in the matching movie soundtrack cluster. This will be our recommendation mechanism.

Our reason for using the Spotify audio features is that we can rely on this better established values in some way in our models. For the mel spectrogram model, the features will be labels, for this model, the features are inputs.

For our song data, we chose the FMA dataset[1], from which we used the `echonest.csv`, since it contains the Spotify audio features of 13,129 songs, and filtered the information from it to `spotify_data3.csv`. For each song we had a song id, and the respective Spotify audio labels.

For our movie dataset, we used the Top 250 Imdb Movies and Tv Series [3] dataset, which contains 3132 soundtracks from 250 movies. We then sent each of these soundtracks to Spotify API to collect its Spotify audio features, and saved this information in `soundtrack_labels.csv`.

First, our goal was to see how well we could cluster the Spotify audio features with k-means clustering. For this, we used `sklearn` library for their `KMeans` class, and clustered the songs into 500 clusters. We did this by using all 7 Spotify audio features, *acousticness*, *danceability*, *instrumentalness*, *energy*, *liveness*, *speechiness*, and *valence*, and projected it onto a 3D space. The `KMeans` algorithm did the rest of the work by clustering the close songs. Then we followed the same procedure for movie soundtracks by using the same features and clustered them into 500 clusters. We recorded the movie names and their cluster id's into `movie_soundtracks_table.csv` for lookup at the very last step of recommendation.

We recorded the resulting `cluster_id` of each song into `allSongsLabelwClusters.csv`, which we will use for training. Then we used the average Spotify feature values of song clusters and matched them with their closest movie cluster, and recorded these couples in `translation_table.csv`

Finally, it was time for training the model. We wanted the model to be able to accurately predict a song's cluster given its Spotify audio labels. All of our previous steps were a prerequisites to this goal.

The code of all the steps explained above are contained in `Spotify-k_means.ipynb`.

Using the data in `allSongsLabelwClusters.csv`, we trained our model, using the list of features of each song as input, and the `cluster_id` of the song as the label. The code and the training of the model is in `Spotify-labeltocluster.ipynb`, and the model class is named `LabelToClusterModel()`. Its layers are as following:

```
# Linear(7, 512)
# Linear(512, 256)
# Linear(256, 128)
# Linear(128, 500)
```

We use Adam as our optimizer, with a learning rate of 0.002, and our loss function is `CrossEntropyLoss`.

After we train our model, we will be able to recommend movies by just the name of the song as input. We will ask the Spotify API for audio features, input the features into the model for song cluster id, and use the song cluster to find the movie cluster corresponding to it from the translation table we created. Then we can look in our `movie_soundtracks_table.csv`, and list the movies that are in that movie cluster. This will finalize our recommendation process for this method.

In the next section, we will try to bypass the Spotify API process, and train a model to create Spotify audio features from waveforms of songs.

2.3. Mel Spectrograms

After using the two methods in 2.1 and 2.2, we wanted to explore other options that weren't reliant on the Spotify audio features as input. Because the most prevalent form of song storage is in mp3, we decided that we can extract information from the song that can be used to train a model. After examining the audio feature analysis tutorial [2] by Ketan Doshi, we decided mel frequency spectrograms were our best shot because of their ease of use and the large number of parameters created by them which we can use in our models. In light of the fact that the results of our first two methods were successful, we wanted to connect this approach with the previous ones in the way that we use the mel spectrogram parameters to generate Spotify audio features, which would work as an input for the other two models. After we managed to generate our own features from the waveform files, we could use either the NLP positivity model or the Spotify cluster model to produce recommendations.

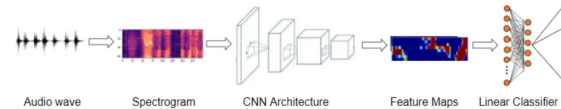


Fig 3. Diagram of the mel spectrogram process by the author of the tutorial[2]

There are multiple steps to making use of mel frequency spectrograms. First, we downloaded a large mp3 song dataset from Free Music Archive [1], containing 106,574 songs 30 seconds in length. The metadata that comes along with the dataset contains the Spotify audio features for 13,129 of the songs in the dataset. We filtered those songs from the mp3 files and turned them into waveform files using `pydub` python library. Then, we defined an `AudioUtil` class that opened the waveform files, resampled them to 44100 Hz, converted mono files into stereo, shrunk all of them into 20 seconds, and then processed them into mel frequency spectrograms and used augmentation to turn them into a tensor of shape (2,64,820). We then defined our model to work with this input and create an output for 7 values. The code for this method is in `Spotify-mel.ipynb`, and the layers of our final model are in the class `MultilabelClassifier`, the pseudocode for the model is as follows:

```
# Conv2d(2, 8, (5,5), (2,2), (1,1))
# ReLu()
# BatchNorm2d(8)
# Conv2d(8, 16, (3,3), (2,2), (1,1))
# ReLu()
# BatchNorm2d(16)
# Conv2d(16, 32, (3,3), (2,2), (1,1))
# ReLu()
# BatchNorm2d(32)
# Conv2d(32, 64, (3,3), (2,2), (1,1))
# ReLu()
# BatchNorm2d(64)
# Flatten()
# Linear(64,7)
# Sigmoid()
```

This way we take in 2 channels of frequency tensors and convolve through them, flatten for the linear and produce 7 values between 0 and 1 with the use of sigmoid at the end. We have also tried `LeakyReLU` and sigmoid activation functions, different stride and kernel sizes, fewer or more layers for the convolutions and the linear layers, and different widths and depths for the linear layer. The one we report is the one with the best performance, even though to a very small extent.

For our optimizer, we used SGD after testing our speed and convergence rate with Adam. Our learning rate was 0.005, and we decided this to be the best after testing for 0.01 and 0.001. For our loss function, we used MSELoss as our values are continuous and are between 0 and 1. We also used a OneCycle Scheduler, starting from a 0.005 learning rate and slowly decreasing linearly. Our training function is nothing different than the usual, it is contained in the `Spotiflix-mel.ipynb` with the name `training2()`, and the pseudocode is as follows:

```
# initialize list for loss values
# initialize loss and optimizer
# for loop through epochs
#   refresh loss values
#   for loop through data loader
#     extract data
#     normalize input
#     reset gradient
#     use model to get output
#     use loss function and find loss
#     backpropagate with the loss
#     step through optimizer
#     step through scheduler
#     update average loss and print periodically
# use the validation data to calculate loss
# print the loss for each separate feature
```

3. Results

3.1. NLP Sentiment Analysis

For the training of the positiveness predictor model, we used the data that we have collected from VADER tool and Spotify API for each song in our dataset. The 7 Spotify audio features we used as inputs were *acousticness*, *danceability*, *energy*, *instrumentalness*, *liveness*, *speechiness*, and *valence*. All of these are continuous values from 0 to 1. Our labels were the positiveness score of each song, ranging from 0 to 1.

We had data for 4776 songs, a collection of different genres and moods. We split the dataset into training and validation sets, consisting of 3821 and 955 songs respectively. We trained the model for 40 epochs, and because we optimized our model with its simplicity to such an extent that after the first epoch, the average training loss and validation loss are both stuck at 0.0014. We suspect this to be the local minima that we can't get out of. No matter how we changed our model we couldn't reach a

better convergence, but only different speeds at which we reach there.

After inspecting the predictions of our model on the validation set along with the ground truth, we estimate a 0.3 error on average on the prediction of our model. This result matches with the MSELoss values, given that the square root of 0.0014 is 0.037, which is the average difference between the guessed values and the ground truth.

When we input the song, *Hotel California* by Eagles, we get the movies: *So I Married an Axe Murderer*, and *The Informant!*. Both of these movies are comedies, one of them being a romcom, and the other a drama/comedy. If we input the song *Ne Me Quitte Pas* by Jacques Brel, we get the recommendations: *Single White Female*, which is a psychological erotic thriller, and *Ocean's 11* which is a crime movie.

3.2. Spotify Audio Features

Before we even started training a model, we had to create our labels for the Spotify features. For this, we used KMeans clustering from sklearn library. We clustered the songs and movies into 500 clusters using 7 of the Spotify audio features, and we deemed the results to be satisfactory.

For the song clustering, we clustered 13,129 songs into 500 clusters with a silhouette score 0.13. This is an acceptable clustering given that scores above 0 mean a good distinguishability between clusters. When we delved deeper into statistics, each of our clusters had <0.1 standart deviation between the Spotify audio feature values of the songs in it, meaning our clustering successfully grouped similar songs together. The clusters had a varying amount of song in them, ranging from 31 to 1 songs, meaning some songs with outlying Spotify features were also present.

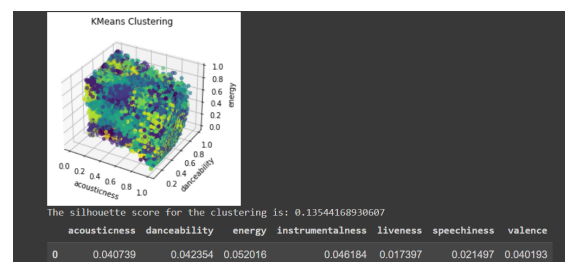


Fig 4. Song clustering result with silhouette score and average std of Spotify feature values in clusters

For the movie soundtrack clustering, we clustered 3132 soundtracks into 500 clusters. Surprisingly, soundtracks clustered better than songs with a silhouette score of 0.3. This is also very likely due to the fact that there are far fewer soundtracks in our dataset, even though we are putting both songs and soundtracks into the same amount of clusters. The soundtrack clusters also had <0.1 standart

deviation between the Spotify audio feature values of the soundtracks in them, meaning a nice grouping of similar sounding soundtracks.

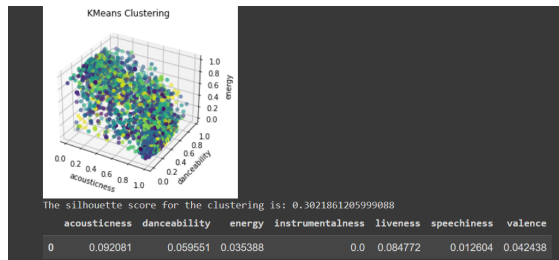


Fig 5. Movie soundtrack clustering result with silhouette score and average std of Spotify feature values

With these reliable clustering results, we trained our model with confidence. Our inputs were the 7 Spotify audio features of each song, and the label was their cluster_id determined by the k-means clustering. The training code is in `Spotiflix-labeltocluster.ipynb`.

We trained the model for 160 epochs, achieving a validation accuracy of 82%. If we trained for longer, it was possible that a better accuracy could be achieved, but for saving time and because of the already good result, we accepted the model and proceeded with the project.

When we tested our model with our 3 chosen songs, the results were as following. For *Hotel California* by Eagles, we got *Rush*, which is an action/sport movie, and *Kill Bill vol.1*, which is an action/drama movie. While these are not the perfect match, they also aren't appalling bad. With the second song, *Ne Me Quitte Pas* by Jacques Brel, we got *Casino* (crime/drama), *Game of Thrones*, and *The Terminator*. These aren't exactly the type of movies someone would want with a love song. And for the last song, *Volare* by Dean Martin, we also received the same movies as for *Ne Me Quitte Pas*, meaning there is a commonality. However, for *Volare*, these recommendations aren't horrible, but not great either.

After inspecting the Spotify features of the last 2 songs, we noticed that they both have near 0 instrumentalness label. Meaning it is very likely the reason why they were in the same cluster. Overall, even though the model results were satisfactory, the movie recommendation mechanism leaves much to be desired, due to some edge-case clusters that gather songs that don't really belong anywhere.

3.3. Mel Frequency Spectrograms

For the training of our spectrogram model, we directly followed the process explained in 2.3 to extract frequency parameters from waveforms, and we stored these in tensors for input. The labels were the Spotify audio features, all from the `spotify_data3.csv`, which is based on the metadata

file named `echonest.csv` from the FMA dataset. The 7 Spotify audio features we used as inputs were *acousticness*, *danceability*, *energy*, *instrumentalness*, *liveness*, *speechiness*, and *valence*. All of these are continuous values from 0 to 1. Our dataset had the data of 13,123 songs, and they were split into training and validation sets with ratio 8:2, 10,498 in training and 2,625 in validation.

The final model trained for 5 epochs, for 2.5 hours. The reason we stopped at 5 epochs is that additional epochs caused overfitting, as the validation loss wasn't getting better. The longest we trained the model with the same setup was 15 epochs, and it had the same validation loss as the 5 epoch model, which was 2.37 total loss over all 7 Spotify features. Individually, acousticness loss was 0.36, danceability loss was 0.33, energy loss 0.35, instrumentalness loss 0.34, liveness loss 0.34, speechiness loss 0.33, valence loss 0.32. Meaning on average 0.339 mse loss per label. This translates to ± 0.58 error in the prediction of our model when guessing a value from 0 to 1.

When we use this model to predict the Spotify features of a given song, we use the model from section 2.2 to predict which song cluster the music belongs to. Then we use our translation table as explained in 3.2 to find the matching movie cluster and recommend the movies in that cluster.

To use the same examples as 3.1, when we input *Hotel California* by Eagles, our model's output are: *Interstellar*, *Joker*, *The Truman Show*, *The Handmaiden*, and *Cosmos: A Spacetime Odyssey*. Most of these movies are fast and brimming films, which match the feeling of the input song. If we input *Ne Me Quitte Pas* by Jacques Brel, our model's outputs are: *The Shawshank Redemption*, *For a Few Dollars More*, *Prisoners*, and *Dune*. These too, make sense as they are slow and meaningful movies. Lastly, when we input *Volare* by Dean Martin, our model recommends: *Spider-Man: Into the Spider-Verse*, *3 Idiots*, *The Hunt*, *Casino*, and *Fargo*. The recommended movies match the feeling of the song, as they are movy, fast and vivid. Overall, we believe that despite the incompetencies of our model, these are good recommendations given the mood of the songs.

4. Discussion and conclusions

All of our models except the Spotify feature to cluster model (section 2.2 and 3.2) had huge error margins in their predictions. While the recommended films are not totally off the mark, they are not perfect by any means. There are many movies in the dataset that are better matches for the given songs, and our models failed to identify those movies.

We tried many different models, optimizers, loss functions, layers and ideas in our positivity model(2.1 - 3.1) and mel spectrogram model(2.3 - 3.3), but failed to overcome the huge errors that are likely the reason for the imperfection of the recommendations. However, the recommendations are not infeasible, and overall it can be called a successful project.

Our recommendation pool is also limited. Since we had to process movie soundtracks for their Spotify audio features, we needed a dataset of movies with their soundtracks. The biggest we could find was the Top 250 Imdb Movies and Tv Series [3], which contained 3132 soundtracks. So, even if our models were able to accurately pinpoint clusters of movies that are the best fit for the input songs, our limited option of movies could have impeded the overall performance. Given a better dataset with more movies of different types, our existing models could perform better.

We hope to discuss and talk about potential reasons of our shortcomings and possible paths that can be explored to improve on them. We will talk about each approach separately.

4.1. NLP Sentiment Analysis

After conducting our analysis using the NLP approach, we have come to the conclusion that our capabilities were limited. Although we were able to extract the positive, negative, and neutral sentiments of the song lyrics and movie scripts, we recognize that a model that could distinguish emotions such as hate, love, anger, disappointment, and loneliness would have greatly improved our movie recommendation system.

Furthermore, we found that the neutrality of the sentiment analysis for movie scripts was much higher compared to song lyrics. This realization led us to explore a different parameter that could be used to match movies with songs. We decided to disregard neutrality and instead, use the overall positivity score as the determining factor. The positivity score was calculated by dividing the positivity by the sum of the positivity and negativity scores. However, we acknowledge that this approach may have led to some inaccurate predictions.

To improve our recommendation system, we suggest building a more comprehensive model that can understand and recognize a wider range of emotions. This would require access to extensive datasets containing various emotional states. By using a more sophisticated model, we can provide more accurate movie recommendations that better align with the emotional preferences of our users.

In summary, our current approach has its limitations, and we recognize the need for

improvement. By incorporating a broader range of emotions and refining our methodology, we can develop a more accurate and effective recommendation system that meets the needs of our users.

4.2. Spotify Audio Features

As a result of our Spotify feature model, we have a working mechanism for recommending movies using the cluster matching method. High silhouette scores while clustering songs and soundtracks show that Spotify audio features can be efficiently used to group similar songs together.

Our feature-to-cluster model achieved high accuracy levels on both training and validation sets, showing that the results of the clustering can be understood and replicated by a machine. This shows promise for future experiments where sharper classifications can be tried by models using similar inputs to Spotify features. This way, many intermittent steps can be avoided and better results can be achieved by sheer simplicity and higher efficiency.

However, poor performance in recommending movies when the input song has an edge-case value for a Spotify audio feature, (like 0.0012 for instrumentality), shows that either the clustering method, or the nature of Spotify features represent a challenge for handling the cases where a song is lacking in one aspect. Further testing of this hypothesis are problematic since deciding whether a recommendation is good or bad is a challenging task for automation.

As the result of our mel spectrogram model in 2.3 and 3.3, we have achieved better recommendations for the same songs. This brings in the question whether our high error margins on the mel spectrogram model were actually detrimental to the ultimate recommendation process. It could mean that the poorly generated Spotify feature values of our model are a better fit for understanding songs, and subsequently be used for recommending movies.

To conclude, we believe mel spectrograms provide a better opportunity for further development in this field. While Spotify API and Spotify audio features were a good stepping stone, more complicated parameters are needed when trying to grasp an involved concept like the mood of a song.

4.3. Mel Spectrograms

The results of our mel spectrogram recommendation model are acceptable. The movies that are being recommended are in similar mood to the input songs, but as we admitted, they could be

much better. As we explained in 3.3, the ± 0.58 error in Spotify feature prediction can be the biggest reason for us being uncertain of the quality of our recommendations. Most of our group's time was spent trying to figure out how we could get over this huge error margin, whether it be simple solutions like different optimizers and learning rates, or completely changing our model layers. We have trained for longer than 60 hours using 30 different setups, and this was the best result we have had, and as mentioned, by a very small error difference.

A greater understanding of audio features and mel frequency spectrograms could have been useful in augmenting our data better or developing a different model using more suitable techniques. Our lack of knowledge in the relevant field and low compute levels held us back from further experimentation. While our results are not unacceptable, we are also not able to pinpoint the reason why our errors are so high, or figure out how we can fix it.

As mentioned above, the biggest holdback in the training of the model was the compute and the massive dataset. We are using roughly 130gb of mp3 files, which expand up to roughly 350gb when converted into waveforms. Processing these files during training was heavily time and compute power consuming, causing even the bare minimum training to take more than 2 hours.

For future experimentation, our next step would be trying Librosa library to see if it can provide better input for training. It is definitely a better established method for audio feature extraction, which we tried to avoid to try out a novel approach.

5. Statement of individual contribution

5.1. Ahmet Turan Bulut

Implemented Sentiment Analysis on song lyrics and movie scripts and created the dataset with the overall_positiveness score to use as a label during the training of the Spotify Audio Features as an input.

5.2. Murad Mikayilzade

Implemented the k-means clustering algorithm using the Spotify Audio Features API. This involved collecting audio features data from Spotify for a given set of songs, such as tempo, loudness, and danceability, and using this data to group the songs into clusters based on their similarity. I utilized the scikit-learn library in Python to implement the k-means algorithm and evaluate the effectiveness of the clustering.

5.3. Mert Yapucuoglu

Found the FMA song dataset and preprocessed the music files in it. Implemented the mel spectrogram model and did all of the training. Also designed and trained both the NLP sentiment model and the Spotify audio features model, and handled their integration together for the recommendation system.

6. References

[1]

<https://github.com/mdeff/fma>

[2]

<https://towardsdatascience.com/audio-deep-learning-made-simple-sound-classification-step-by-step-cebc936bbe5>

[3]

<https://www.kaggle.com/datasets/ravineesh/soundtracks-of-top-250-imdb-movies-and-tv-series>

[4]

<https://towardsdatascience.com/how-to-analyze-emotions-and-words-of-the-lyrics-from-your-favorite-music-artist-bbca10411283>