Using Music Information Retrieval Feature Selection to Generate Music Suggestions

Leila Kassiri
Johns Hopkins University
9 E 33rd
Baltimore, MD 21218, USA
lkassir1@jhu.edu

Paul Watson
Johns Hopkins University
9 E 33rd
Baltimore, MD 21218, USA
pwatso14@jhu.edu

Abstract

Music-streaming services like Spotify and Soundcloud have revolutionized the music industry and have become increasingly competitive with their music suggestion generators. These generators utilize machine learning to suggest songs and to generate entire playlists of songs based on a users music listening history. We sought to create music suggestion models that are based on the features of the song themselves rather than the user data. We created a Song Suggestion model that uses distance between feature vectors to rank suggestions and a Playlist Generation model that is a two-layer model, the first layer using the distance to rank suggestions and the second layer using lambda-means clustering. The growing field of Music Information Retrieval (MIR) focuses on extracting meaningful features from the raw music files. We experimented with various features and distance metrics to finding the ideal combination for single song and playlist suggestions.

1 Introduction

Music-streaming services like Spotify and Soundcloud have revolutionized the music industry and have become increasingly competitive with their music suggestion generators. These generators utilize machine learning to suggest songs and to generate entire playlists of songs based on a users music listening history. We sought to create music suggestion models that are based on the features of the song themselves rather than the user data. We created a Song Suggestion model that uses distance between feature vectors to rank suggestions and a Playlist Generation model that is a two-layer model, the first layer using the distance to rank suggestions and the second layer using lambda-means clustering. The Song Suggestion model simply returns the top ranked songs and the Playlist Generation model selects songs from various clusters in the second layer to provide a playlist that is varied based on chosen features.

2 Background

Music Information Retrieval is an established field that focuses on finding ways to extract descriptive data from audio. We spent a considerable amount of time when coming up with our proposal and before we started designing the models finding features that would be good to use in our models. There is a wide variety of ways to extract features from waveform data, ranging from extremely low to high-level descriptors. We wanted a good range of features so we chose five features from different parts of this spectrum.

Our lowest-level features are zero crossing rate and root mean square. Both of these features are very self-descriptive: the zero crossing rate is the rate at which the waveform crosses zero, and the root mean square is the square root of the mean of the squares of the amplitude of the waveform. Both these features are calculated over a set window of 1/200th of the length the song, giving a vector with length 200. The root mean square describes the average power or loudness of the signal over a window. The zero crossing rate is used primarily in speech detection, as environmental noises generally have a much smaller zero crossing rate than speech, zero crossing rate can determine how much singing is in a song (Peeters).

Our higher-level features are spectral centroid, spectral rolloff, and Mel-frequency cepstral coefficient. The spectral centroid describes the center of mass of a signal. It is the weighted mean of the frequencies, using their amplitudes as the weights. A low spectral centroid indicates a darker sounding song, while a high spectral centroid indicates a brighter song. Spectral rolloff is the frequency below which 85% of the amplitude in a window is contained. In other words, it measures the right-skewness of the magnitude spectrum. Like zero crossing rate, this feature is often used to discern speech from non-speech sounds (Peeters and Rodet). The final and highest-level feature is the Mel-frequency cepstral coefficients (MFCCs). This feature was designed with the human voice in mind, as the bands in the Mel-frequency cepstrum was created to mimic on the human auditory system. Its primary use is speech recognition, but it is also used for audio similarity measures and genre classification (Jensen). The feature vectors pertaining to these features also depend on the length of the song; so median pooling was used to reduce each vector to a length of 200 in order to make comparisons.

While doing research after we had begun our project, we read a paper about l^p norms for high-dimensional data (Flexer and Schnitzer). The paper discusses issue with Euclidean and Manhattan distance metrics in terms of hubs and anti-hubs. Many high-dimensional datasets contain hubs and anti-hubs when using Euclidean or Manhattan norms. A hub is a data point that has a small distance to a large number of data points. An anti-hub is a data point that has a large distance to a large number of data points. This issue is closely related to the concentration of distances. The paper showed that the hubness of a dataset could be reduced by implementing l^p norms such that $p < 1$. We believe that our results could be improved if we implemented these l^p norms in our models.

3 Project Outline

3.1 Machine Learning Techniques

Song Suggestion: To suggest songs that a user would like, we used a recommendation algorithm that uses distance between feature vectors to rank suggestions. We assume that songs with similar feature vectors will be similar songs. Following this assumption, when a user inputs a song that they like, the model will return songs that sound similar to the seed song and therefore they will also enjoy the recommended songs.

To compute the distances between feature vectors, we give the user the choice between Euclidean and max distances. Euclidean distance is simply the square root of the sum of the squares of the differences of each element in the vectors. We chose Euclidean distance because it is a very popular distance metric. Its the definition of distance in 2D and 3D space, and that it takes into account differences between every element in the feature vector. We also chose max distance, which is the max of the absolute values of the differences of each element in the vectors. We chose max distance because it gives the difference of the elements that vary the most between the two vectors. This gives a sort of upper bound to the differences between two songs; no pair of elements in the vectors varies by more than the max distance value.

Playlist Generation: The Playlist Generation model is built as two parts. The first part uses the Song Suggestion algorithm to get a list of songs that are similar to the seed song provided by the user. From there, we use clustering to group those songs by different features from the ones used in the first part.

To cluster the songs, we implemented lambda-means clustering. Lambda-means clustering is like k-means clustering except the number of clusters starts at one, the mean of all the songs. When a song is beyond a certain distance threshold, lambda, from all the current clusters, a new one is made with that song as the mean of the new cluster. This algorithm is very useful because the user doesnt need to specify the number of clusters before running the algorithm, which fits our needs nicely because we have no prior knowledge of the songs that will be clustered except that they exist in our library.

The clustering uses the EM model to fit a model to our songs while we have latent variables. The E step assigns each song to the nearest cluster mean, and creates new clusters if none are under the lambda threshold away from a song. The M step recalculates the mean of each cluster, and if a cluster is now empty, it removes the cluster from consideration. The algorithm performs the E and the M steps for a given number of iterations. After each iteration, the clusters fit the data better and better.

We use clustering in the second layer because we want our playlists to be somewhat varied. We separate the songs into clusters and from there we can adjust the variety in a playlist by taking songs from more or fewer clusters. This relates to the cluster range parameter. If we let the cluster range be n, the algorithm ranks the final clusters and ranks them by their distance from the seed song. We iterate through the n closest clusters to the seed song and take songs from each to fill the playlist. A smaller cluster range means a playlist that is more homogenous; a larger cluster range means a more eclectic playlist. However, if the cluster range approaches the total number of clusters, then the playlist is essentially just the songs that would be suggested by the Song Suggestion model alone.

Hyper Parameter Tuning: The number of clustering iterations is a simple hyper parameter to tune. As it increases, the clusters fit the model better. The only tradeoff is the running time of the algorithm also increases. The algorithm ran in seconds during our tests but it would increase quickly as the number of songs in the library scaled up.

The lambda threshold value parameter determines the number of clusters that will be created. The number of clusters is inversely proportional to lambda. Too many clusters could mean that songs that should belong to the same cluster are split into separate clusters. This can be counteracted by a larger cluster range because the two means of the clusters would be close and therefore would likely both be within the group of closest clusters chosen to be used for the playlist. Too few clusters could group songs that should not be grouped together. Choosing a low cluster range can similarly counteract this.

The cluster range parameter determines the number of clusters that are used to make the playlist in the final step of the Playlist Generation model. A low cluster range means that songs in the playlist will sound more similar. A larger cluster range means that the songs in the playlist will have more variety. As stated earlier, if the cluster range reaches the number of clusters, then the clustering is essentially not used and the playlist will be a random list of songs suggested by the first layer.

The playlist length factor is a number that is multiplied by the length of the playlist to determine the number of songs that the first layer in the Playlist Generation algorithm returns. A larger playlist length factor means that the songs in the playlist will sound less like the seed song. A small playlist length factor means that the songs in the playlist will be more similar to the seed song.

3.2 Project Process

Data Preparation: For data preparation, we wrote ipython-extract-features to extract MIR features from a music file to create a Song object. Song objects are where we store the song title, song artist, and the feature information for use in our Song Suggestion and Playlist Generation classes. The song title and song artist are stored as strings and the features are stored in a dictionary, where the key is the name of the feature stored as a string and the value is a numpy array. Song objects have getTitle, getArtist, which return the song title and artist respectively. Song objects also have addFeature method, which takes in a dictionary of features as an argument, and a getFeature method, which takes the string name of a feature as a parameter and returns the numpy array of that feature.

For our data pool, we decided to use songs from our personal music collections on iTunes. We decided to use about 900 songs for our data pool to be big enough to ensure a wide enough variety of music genres while small enough that we would both be familiar with all of the music. We wanted to be familiar with all of the songs for ease of testing; so we would not have to listen to every single playlist completely through, but rather could look over the Song Suggestions and playlists and be able to quickly tell if the suggestions were good or not.

We initially had debate over how to access the feature information for each song. The feature extraction libraries that we found required the music files to be in .wav format, however iTunes stores its music files in .m4a format. We discussed whether we should store .mp3 files and use those as input to our Song Suggestion and Playlist Prediction classes and when we should create our Song objects. Ultimately we decided to convert the .m4a files by directory to .mp3 files, then convert the .mp3 files to .wav files by directory, and then to iterate through the .wav files to create Song objects and pickle them for later use.

We adapted code from a library called rp-extract to convert the .m4a files to .mp3 files and used the library to convert the .mp3 files to .wav files. Then we iterated through every .wav file to create the Song objects. We used the name of the .wav file to find the .mp3 file of the song in order to use the eyed3 library, a library for retrieving metadata from .mp3 files, to retrieve the song title and artist. Then we created a feature dictionary and added the features using the string name as a key and a numpy array as the value.

The functions to calculate zero crossing rate, spectral centroid, and spectral rolloff were adapted from Alexander Schindlers MIR Feature Extraction library. We wrote the root mean square feature calculation ourselves. The feature we had the most difficulty in calculating was the mel-frequency cepstral coefficients. We tried adapting functions from both the python-speech-feature library and Alexander Schindlers MIR Feature Extraction library and kept getting arrays with filled with NAN values. Ultimately we got the librosa library, a library for audio and music analysis, working to calculate mel-frequency cepstral coefficients.

When we began testing the feature extraction, we noticed that the size of the array would vary both based on the song and based on the feature. Therefore we wrote methods for median pooling. In our median pooling method we divide the length of the vector by 200 to get the window size and for every window we store the median value into the output vector. Then the output vector is normalized to be zero-mean and unit-standard deviation. Thus we made all the feature arrays the same size and added them to a dictionary which was added to the Song object and pickled for later use.

Song Suggestion:Our Song Suggestion model suggests a list of songs similar to a provided seed song. Our model suggests songs by ranking the distance of two chosen features for the seed song and a library of songs. Our Song Suggestion class has a init, suggest, and getDist method and utilizes a min-heap to rank suggestions, using the heapq library.

The Song Suggestion class takes the distance metric to be used and the number of songs to be suggested as parameters. As delineated in our project proposal, we provided the choice of using either Euclidean distance or maximum distance. In initializing a instance of the class, the init method goes through the directory of pickle files of Song objects, unpickles the Song objects and adds them to a global music library list.

The suggest method takes in the name and artist of the song for the suggestions to be based on and the two features to determine the ranking. The method initializes the min-heap and finds and unpickles the seed Song object. For each song in the music library, getDist is called on that song and the seed for both features. The result of getDist for both features gets averaged and a tuple containing the average distance and the Song object is added to the heap. The n-smallest method from the heapq library is called on the heap, where the heap and the number of songs suggestions wanted are fed in as arguments. The n-smallest method returns a list of the n songs with the smallest distance to the seed song. The list is iterated through to print the song title, artist, and distance, and the list is returned.

Originally we wanted our Song Suggestion model to not return any songs by the same artist as the seed song. However when we began testing we realized that producing song suggestions by the same artist could help us indicate what features made the best suggestions. Obviously not all the songs by the same artist would be a good recommendation for a particular song, however some songs by the same artist would be. We kept this feature in as we continued to use it because our model still produced song suggestions from a variety of artists even when including a couple of songs from the same artist.

We also spent some time debating how to iterate through all the songs in order to make the comparison. We originally planned to get the distance measures as we unpickled the Song objects but we realized that it would be cleaner to unpickle all the Songs at once and then iterate through a list of Song objects when we were getting the distances.

Originally Song Suggestion only printed out the title and artist names of the suggested songs but when we integrated Song Suggestion into our Playlist Generation, we also had the class return a list of song objects that are suggested for the seed song.

Playlist Generation: Our Playlist Generation model creates a shuffled playlist around a seed song given by the user. Our goal for this model was to create playlists that sound good together with the seed song, and have a level of variety such that the playlist is not monotonous. We built a two-layer model. The first layer uses the Song Suggestion model and the second layer uses lambda-means clustering.

Our Playlist Generation model has an init method, suggest method which functions as the core of the model, train and predict methods which implement lambda-means clustering for the second layer. There are also 5 helper methods: get-num-songs, which returns the number of songs in each cluster, calculate-mean, which calculates the total mean based on provided features, calculate-lambda, which calculates the lambda value, closest-cluster, which calculates the closest cluster for a song, and cluster-ranking, which orders the clusters based on the distance of the cluster mean to the given song.

At initialization, the user inputs the distance metric type to use, the number of clustering iterations, the value to use for lambda, the length of the playlist they would like to create, and the factor to multiply the playlist length by to determine the number of songs that the first layer will recommend.

When querying for a playlist, the user calls the suggest method and inputs the song name, song artist, four feature names, and the cluster range. The first two features are used in comparisons in the first layer and the last two features are used in comparisons in the second layer. The cluster range is the number of clusters that the algorithm will look in to pull songs for the playlist in the second layer.

For the first layer, the suggest method calls Song Suggestion and creates an instance of the class with the same distance type parameter. The parameter for Song Suggestion that determines the number of song suggested is calculated by multiplying the playlist length parameter with the first layer factor parameter, which is defined as the factor to multiply the playlist length by to determine the number of songs that the first layer will recommend. We added this factor so that we could easily control how big the pool of songs would be for the second layer.Then Song Suggestion is provided the title and artist of the seed song and the first two features and returns a list of songs that are similar to the seed song based on the first two features.

Then we train the lambda-means clustering on the list of songs that made it through the first layer; the list of songs returned by Song Suggestion. Clustering is used to give the variety in the playlists. Songs in different clusters are assumed to have differences in sound, such that taking songs from multiple clusters will achieve our goal of an interesting and colorful playlist. To insure that all the songs in the playlist are still enjoyable to the user, we only cluster songs that were returned by the Song Suggestion model. We decided to use lambda-means clustering in particular to achieve more organic groupings of the songs, we decided that it would be impossible to hard-code or select the ideal number of clusters that the songs would fit into.

The train method repeats the E and the M steps for the given number of training iterations. The E step assigns each song to the nearest cluster mean and creates new clusters if the distance is above the lambda threshold. The distances are based on the last two features parameters. The M step recalculates the mean of the each cluster and removes empty clusters. The methods get-num-songs, calculate-mean, calculate-lambda and closest-cluster are helper methods for the train method.

Then the predict method takes in the song title and artist, the features to based the prediction on, and the cluster range. The predict method first shuffles the songs in every cluster in order to provide variation, as the songs are otherwise in alphabetical order. Then the predict method calls the cluster-ranking function which orders the clusters based on the distance of the cluster mean to the seed song based on the two provided features. The ranking is done by pushing a tuple onto a min-heap that holds the list of songs in a cluster and the distance between the cluster mean and the seed song. By the structure of the algorithm, the seed song is in the cluster that is the top element of the min-heap.

Then the predict method iterates through the clusters to get songs for the playlist. The cluster range determines how many of the nearest clusters to use when creating the playlist. The default value is 0, in which case the model will take one song from each cluster until the playlist length is reached. For any other value of cluster range n, the method will get the n top clusters from the min-heap and take one song from each cluster until the playlist length is reached. The method checks to make sure that the number of songs in the n top clusters is greater than or equal to the desired playlist length. If the n top clusters do not have enough songs, then the method includes the n+i top clusters, where i is the minimum number of additional clusters for the size of the clusters to surpass the playlist length.

Finally if the seed song was not chosen for the playlist, we delete a song and add the seed song because we intended this algorithm to generate a playlist around a certain song. The playlist is shuffled and returned and the title and artist of each song in the playlist is printed.

4 Results

4.1 Song Suggestion

The nature of our project is not conducive to simple testing. The data is clearly unlabeled and all of the goals we wanted to achieve are extremely difficult to quantify, especially with time and resources we have. We ran our model with every permutation of features and distance metrics on 5 songs of different genres and the number of suggestions set to 15. We manually assessed each playlist together, as all of the songs in the library were familiar to at least one of us.

Here is an output of suggested songs using zero crossing rate and MFCC for features, and max distance as the distance metric:

> Suggested songs for Touch the Sky (Featuring Lupe Fiasco) by Kanye West:
> Feel Good Inc - Gorillaz
> What I Got - Sublime
> Florida - STRFKR
> Ice Ice Baby - Vanilla Ice
> Ni**as in Paris - Kanye West and JAY Z
> Cry (Just A Little) - Bingo Players
> The Recipe - Kendrick Lamar feat Dr. Dre
> Broken Jaw - Foster the People
> All These Things That I've Done - The Killers
> Vanished - Crystal Castles
> Zero 76 (Original Mix) - Tisto and Hardwell
> Extraordinary - Childish Gambino
> The Futile - Say Anything
> Infinity Guitars - Sleigh Bells
> Candler Road (1st half prod. Tim Suby 2nd half prod. Childish Gambino) - Childish Gambino

All of the suggested songs genres are either rap or electronic, fitting very well with the seed song Touch the Sky. When using these parameters, the algorithm consistently returns good suggestions across all genres. One caveat is that as the seed song diverges more and more from mainstream music, the recommendations become less consistent, with a few odd suggestions appearing. For example, when the seed song was Meme Generator by Dan Deacon, an intense electronic song, we consistently received Rocket Man by Elton John, a piano ballad, as a top suggestion.

For tuning parameters, we followed the same approach as before, and looked for suggestions that were extremely out of place. This was an indication that the pair of features and distance metric used to generate the suggestions did not adequately predict songs that would be similarly enjoyable. We used this technique to rule out MFCC, zero crossing rate, and both Euclidean and max distance because the model recommended Waves by Kanye West for I Am A Rock by Simon and Garfunkel. We also ruled out MFCC, spectral rolloff, and max distance for recommending Heartbeat by Childish Gambino for I Am A Rock by Simon and Garfunkel.

4.2 Playlist Generation

We tested the Playlist Generation model after testing and tuning the Song Suggestion model because this model relies on good suggestions. The testing and tuning was done in a very similar way. We set the features for the first layer to MFCC and spectral centroid, the number of clustering iterations to 50, and the playlist length to 10. We iterated through all pairs of the remaining three features. We iterated through both distance metrics. We iterated through zero, one, five, 10, and 30 for the cluster ranges. We iterated through 3, 5, and 7 for the playlist length factor.

Here is the output for a playlist generated using zero crossing rate and spectral rolloff as the features for clustering, Euclidean distance, a cluster range of

10, and a playlist length factor of 7:

> Playlist generated for Flash Delirium by MGMT
> Flash Delirium - MGMT
> Hang Me Up to Dry - Cold War Kids
> Alive With the Glory of Love - Say Anything
> 15 Minutes - The Strokes
> Poke ft. Steve G Lover III - Childish Gambino
> New Routine - Fountains Of Wayne
> Sour Cherry - The Kills
> Don't Bring Me Down - Electric Light Orchestra
> This Better Be Good - Fountains Of Wayne
> Bee of the Bird of the Moth - They Might Be Giants

This playlist has a variation in the genre yet all the songs match the eclectic and full sound of Flash Delirium.

Overall, the playlists that our models generated hit both of our goals: cohesiveness and variety. One observation that we made during testing was that if the playlist length factor is small (less than 4) the other parameters, especially the cluster range, did not have a significant effect on the output. We believe this is because the second layer is going to return a large portion of the songs, so songs that are in far clusters might still be added to the playlist simply because few songs wont be put in the playlist. In addition, if the playlist length factor and the cluster range were large, the playlists were much more varied.

5 Comparison to Project Proposal

5.1 Must achieve

- We must achieve a working Song Suggestion model as outlined

  We achieved this completely and exactly as we described in our proposal.

- We must achieve a working single layer Playlist Generation model that returns a single cluster as a playlist

  We achieved this completely and exactly as we described in our proposal.

- We must implement both Euclidean and maximum distance metrics

  We achieved this completely and exactly as we described in our proposal.

5.2 Expected to achieve

- We expect to achieve a two-layer playlist as outlined

  As we started designing our model, we decided that the implementation in our proposal would not let us insure that the number of songs given to the second layer for clustering would be large enough. The Song Suggestion model does this better than a clustering model could so we decided to repurpose it to be used standalone and as the first layer for our Playlist Generation algorithm.

5.3 Would like to achieve

- We would like to test our Playlist Generation model with different features at each layer. We want to do this because a good playlist has a balance between homogeneity and variety and we hope to clarify what features should be consistent within a playlist and what features can/should be varied.

  We tested the Playlist Generation algorithm with all permutations of the features. We identified the most successful features for the Song Suggestion model, the first layer, to be MFCC and spectral centroid. We have not been able to narrow down the best features for the second layer because we received similar results but we would like to continue testing on a larger pool of music with additional features.

References

Arthur Flexer and Dominik Schnitzer. 2015. *Choosing lp norms in high-dimensional spaces based on hub analysis*. Austrian Research Institute for Artificial Intelligence.

Geoffroy Peeters and Xavier Rodet. 2003. *Signal-based Music Structure Discovery for Music Audio Summary*

Generation. International Computer Music Association.

Geoffroy Peeters. 2003. *A large set of audio features for sound description (similarity and classification) in the CUIDADO project*. Institute for Research and Coordination in Acoustics/Music.

Jensen, J. H. 2009. *Feature Extraction for Music Information Retrieval*. Processing, Institute of Electronic Systems, Aalborg University.