

CS229 Final Project: Automatic Playlist Generation

Xu Chen and Xingting Gong

Introduction:

Automatic Playlist Generation:

The goal of our project is to generate a playlist of M songs based on a set of N user-selected songs that serve as the “seed” for our playlist.

Web applications like Spotify and playlist.net each have their own algorithms for suggesting songs to users based on user preferences.

Here we attempt our own method at Automatic Playlist Generation, by training for a similarity metric K for which to predict user song preferences.

Data

Million Song Data(sub)set:

The Million Song Dataset consists of data files of 1,000,000 popular songs, with information including both metadata and audio analysis features. For practicality, we downloaded the 10,000 song subset for our project.

Feature Selection:

From the dataset, we selected 5 features per song: Genre, Tempo, Average Loudness, Year, and Timbre. For all features except timbre, we binned the values into discrete categories. For timbre, we randomly selected 200 rows from the timbre matrix. Example values are shown in the table below:

Feature Vectors

| Features | Example (Raw) Values | Example (binned) Values |
|-----------------------------------|--|----------------------------|
| Genre | Rock, Indie, Pop, Country, Rap, Hip-Hop, Metal | 0, 1, 2, 3, 4, 5, 6 |
| Tempo (binned by 20 BPMs) | 130.861, 122.174, 80.149 | 6, 6, 4 |
| Year (binned by decade) | 1989, 1999, 2007 | 198, 199, 200 |
| Average Loudness (binned by 5dbs) | -13.366, -7.928, -14.367 | -2, -1, -2 |
| Timbre | Matrix of MFCC | Randomly selected 200 rows |

Method: Gaussian Process Regression

We model the unknown user preference f_s of song s as a **Gaussian Process** with variance σ :

$$f_s = \sum_{i=0}^N \alpha_i K(x_i, x_s), \text{ where}$$

$$\alpha_i = \sum_{j=0}^N (K(x_i, x_j) + \sigma^2 \delta_{ij})^{-1}, \text{ where}$$

x_i is the feature vector for song i

Thus our main goal is to learn a similarity metric, K , by **kernel-meta-training: Given a set of songs, we first pre-group these songs by some feature (genre, artist, etc), and use these “pre-playlists” to learn K .**

We tried two Kernels:

1. Linear Kernel (no training required): $K(x_i, x_j) = \|x_i - x_j\|^2$

(Note: The difference between the genre components of the feature vectors is 1 if the genres match and 0 otherwise)

2. Linear combination of a family of Mercer kernels:

$$K(x_i, x_j) = \sum_{n=1}^{N_\psi} \beta_n \psi_n(x_i, x_j), \quad N_\psi = 2^{\text{number of features}}$$

$$\psi_n(x_i, x_j) = \begin{cases} 1 & \text{if } a_{nl} = 0 \text{ or } (x_i)_l = (x_j)_l \quad \forall l \\ 0 & \text{otherwise} \end{cases}$$

(Idea: a serves as a “mask” so that we can compare a subset of features at a time)

Solve for coefficients by minimizing cost function:

$$\arg \min_{\beta} \frac{1}{2} \sum_{i,j} (K_{ij} - \sum \beta_n \psi_n(x_i, x_j))^2$$

where K_{ij} is the empirical covariance:

(Note: For the timbre feature, instead of comparing the timbre matrices element-by-element, we took the norm of the difference of the two matrices and set $\psi_n=1$ if the norm < a threshold value, and 0 otherwise)

Results

Example Playlist:

| | Title | Artist | Genre | Year | Tempo | Loudness |
|----------|-----------------------|---------------|-------|-------|-------|----------|
| Seed | Floating | Blue Rodeo | Rock | 1980s | 100 | -5dB |
| | Police Story | Black Flag | Rock | 1980s | 100 | -10dB |
| Playlist | Shadrach | Beastie Boys | Rock | 1980s | 100 | -15dB |
| | Persiana Americana | Soda Stereo | Rock | 1980s | 100 | -10dB |
| | Shed So Many Tears | Johnny Winter | Rock | 1980s | 100 | -15dB |
| | Voices Inside My Head | The Police | Rock | 1980s | 100 | -10dB |

| | Title | Artist | Genre | Year | Tempo | Loudness |
|----------|---------------------------------|---------------------------|-------|-------|-------|----------|
| Seed | Setting Fire to Sleeping Giants | The Dillinger Escape Plan | Indie | 2000s | 160 | 0dB |
| | The End Of The Line | Metallica | Indie | 2000s | 100 | 0dB |
| | Between Love & Hate | The Strokes | Indie | 2000s | 100 | 0dB |
| Playlist | Dancing Shoes | Arctic Monkeys | Indie | 2000s | 140 | 0dB |
| | Run's House | RUN-DMC | Indie | 1980s | 180 | -5dB |
| | Crack | RUN-DMC | Indie | 1980s | 80 | -5dB |

Evaluation:

Score the produced playlist with a standard collaborative filtering metric

$$R_j = \sum_{i=1}^{N_j} \frac{t_{ij}}{2^{(i-1)/(\beta-1)}}$$

Future Work

Support Vector Machine:

- Assume we have a large playlist of user-selected songs (I.e. a user has selected thousands of songs on his/her Spotify account over the course of a year)
- With a larger training set (**more seed songs**), we can apply an SVM and classify each new song as one that a user would like/dislike.

HMM for Timbre:

- HMM's are useful for sequential data
- From the database, timbre comes as an $S \times 12$ matrix of MFCC, where $S = \#$ of segments in the song.
- Can combine HMM with SVM above