Cloudmatch - A music recommender system

Fredrik Johansson
Johan Forslund

January 2020

Introduction

Spotify is globally available and is one of the most recognized music streaming platforms in the world. In recent years it has become popular to use machine learning algorithms to provide the users with recommendations. Spotify uses such a system. However, from personal experience, we noticed that most of these recommendations consists of already well established artists and barley any amateurs.

Therefore, we came up with the idea to recommend tracks to Spotify users from other more amateur friendly streaming platforms such as Soundcloud. To achieve this we have been developing a tool that gathers data and extracts feature vectors from Soundcloud. The tool is setup on a server and is accessed via a web application.

Research question

During the project, we have worked to answer the following question:

- How can we determine which tracks are considered to be good recommendations for our user, using the available data from Soundcloud?

Collecting data

For this project we use two external API's to collect data; Spotify and Soundcloud. Both of these API's are free to use and have lots of different data. For example, one can find information about a user's listening habits, meta information about a song, or which users has liked a particular song.

The data from Spotify is used to learn what kind of music our user is listening to. This consists of the user's most played artists and the genres that these artists play. This is all we need to create an overall picture of what the user often listen to. To get this data in a web application, we followed the

steps in Spotify's authorization guide [1] and then we called the API endpoint `/v1/me/top/artists`.

The data collection from Soundcloud is a bit more tricky. Here we want to get data about lots of different users on the platform. The idea is to pick out a few different songs with different genres, and then for every user that has liked any of these songs we extract that users favorite tracks. By doing it this way, we hopefully get a wide range of music listeners, which we want to be able to match all kind of music.

The data collection is achieved by calling a few different endpoints. The first endpoint is for finding a song with a specific genre, which can be done by calling `/tracks?genres=[]`. This returns a list of songs, where we choose one randomly. Next, we call an endpoint that returns what users has liked that song: `/tracks/:id/favoriters`. This returns a list of users, where every user has an unique ID. For each user we then call the last API endpoint, which returns all songs each respective user has liked: `/users/:id/favorites`.

When combining all these calls, we get a huge list of users and information about the songs that these users have liked. All the available information can be seen in the Soundcloud API - Reference [2]. By using the reference we could filter out the data which is unnecessary for our purpose.

In this project we have used 24 songs of different genres to find users that may like the same kind of music. For each song we have picked out 5000 users and for each user we get 200 liked songs. In total, this gives us information about $24 * 5000 * 200 = 24,000,000$ songs, and which users have liked each song. A snippet of this data is shown below.

```
[
  {
    "user_id": 122232513,
    "favorite_tracks": [
      {
        "id": 567349,
        "likes_count": 81113,
        "comment_count": 742,
        "genre": "EDM",
        "title": "Linkin park - meteora - breaking the habit",
        "stream_url": "https://api.soundcloud.com/tracks/567349/stream"
      },
      {
        "id": 657086237,
        "likes_count": 1876,
        "comment_count": 60,
        "genre": "EDM",
        "title": "Alan Walker X A$AP Rocky - Live Fast (Paris Looky
            Remix) [FREE DOWNLOAD]",
        "stream_url":"https://api.soundcloud.com/tracks/657086237/stream"
      },
      .
```

```
      .
      .
  },
]
```

Creating feature vectors

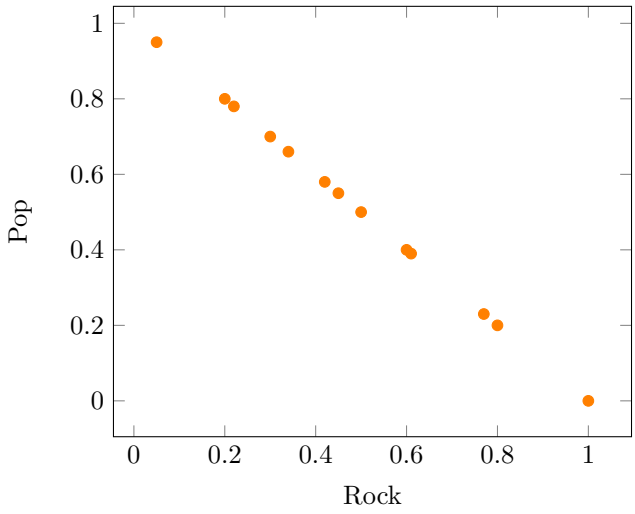
Not all the parameters from the collected data are necessary for the clustering process. For example, it is not that useful to compare the track titles when determining similarity between tracks. It is much more useful to compare the genre of of each track.

First we had to determine which genres to accept. We wanted to have the most common genres, like "rock" and "pop", but we also wanted to incorporate some less known genres like "ska" and "afrobeat".

From the collected data we iterate through each users favorite tracks. For each track we look at the genre, if the genre match one of the accepted genres we increment that genre in a list of all genres by one. When all the tracks have been iterated through we divide the list with the number of tracks for that user. This results in a vector with a mean genre for the user which is much easier to handle than the initial collected data. One feature vector can be seen below, each element represent a genre. For example, this particular user listens to about 4.76% "acoustic" and 2.38% "ambient".

```
[0.047619047619047616, 0.0, 0.0, 0.0, 0.023809523809523808, 0.0, 0.0,
    0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.047619047619047616,
    0.0, 0.0, 0.0, 0.0, 0.0, 0.047619047619047616, 0.0, 0.0, 0.0, 0.0,
    0.0, 0.0, 0.0, 0.09523809523809523, 0.0, 0.023809523809523808,
    0.5714285714285714, 0.0, 0.023809523809523808, 0.0, 0.0, 0.0, 0.0,
    0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
    0.0, 0.0, 0.0, 0.0, 0.023809523809523808, 0.0, 0.0, 0.0, 0.0, 0.0,
    0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
    0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
    0.023809523809523808, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
    0.023809523809523808, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
    0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
    0.0, 0.0, 0.0, 0.047619047619047616, 0.0, 0.0, 0.0]
```

The feature vectors will give us a space of different users with different mean genres. Since we have 126 dimensions in our feature space, it is of course impossible to visualize. However, we can pick out two genres (say rock and pop) and create a scatter plot in two dimensions, see below.



Here we can see how different users have different taste. Some users only listens to rock while some tends to prefer pop, but most are somewhere in between. When extending this to 126 dimensions, we can imagine how this creates a space with feature vectors, and we are now able to match a new user from Spotify against all these users from Soundcloud.

Matching in feature space

The next step is to match our user from Spotify against all the users on Soundcloud. We want to find a group of users to match with, preferably users with some favorite tracks in common. To find these users (feature vectors), we use the Nearest Neighbour algorithm. This is available in Python with the machine learning library scikit-learn, using the function sklearn.neighbors. This way we can place our feature vector (that is created from Spotify data) into the feature space the is built from the Soundcloud data, and then find the feature vectors with the smallest Euclidean distance to.

The Nearest Neighbour algorithm will return the n closest feature vectors, and we keep track of which feature vector represents which user. There is no perfect n to choose, but we choose it by testing different values until we get satisfying results. Since music is subjective, we have to settle with an approximate value.

Giving recommendations

From these n users we then have to find some songs to recommend to our end user. One can think of many ways to do this. Our method was to implement a weighting system to prioritize songs that are a likely match. Each song belonging

to the group of n users gets a score, where a higher score means a better match. Mathematically we can describe it as

$$score = \sum w_i f_i$$

where w_i are the constant weights and f_i are functions that define some information about the song. The function that gets the largest weight in our application is the function that counts how many of the n users have liked that particular song. If many of the users that you have matched with have liked this song, it is reasonable to determine that song as a good match. Another weighted function returns the total number of listens that song has on Soundcloud. However, we use $f = \frac{1}{\#listens}$ to give the songs with too many listens a lower score. We added this to exclude songs that are too famous and established.

Finally, the application chooses the songs with the highest score, hopefully resulting in a list of tracks that matches the user's taste.

Result

Currently the web application is not hosted on a web server. Instead the application is hosted on a local server. The data collection and feature vector creation and matching is implemented in Python. The server setup is implemented with the server framework Flask. The web application client is implemented in JavaScript with the library React. Below are some images of the client. Figure 1 is a screenshot from the front page of the application. It consists of a simple button prompting the user to login with Spotify. Figure 2 is a screenshot from the Spotify login page provided by the API. After the user has given our algorithm authentication to collect user data, the algorithm begins to match our user with Soundcloud users. Once this process is finished, a list of tracks are displayed, see Figure 3.

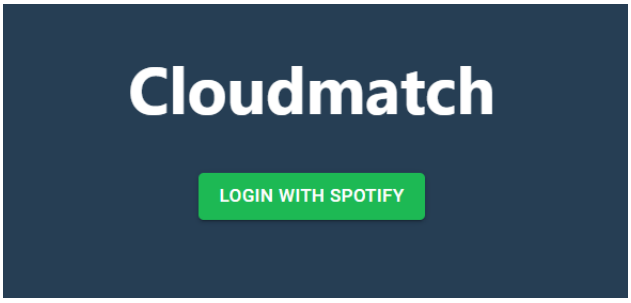


Figure 1: The front page of the web application

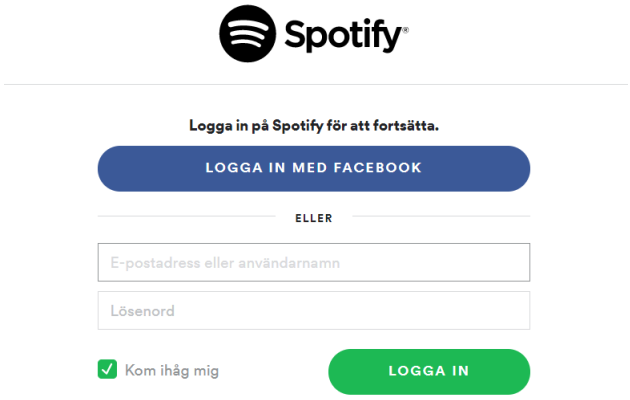


Figure 2: The user is redirected to the Spotify authentication login page

Evaluation

It is hard to evaluate the application, especially its scoring algorithm since music can't be objectively judged. However, we came up with a decent solution using a confusion matrix and user tests. The idea is to first present tracks with the highest score and let the user evaluate the result (true positives and false positives). Then we present tracks with the lowest score and let the user evaluate the result (true negative and false negative). From this we can calculate the accuracy of the algorithm.

		prediction outcome	
		p	**n**
actual value	**p**	True positive	False negative
	n	False positive	True negative

Summary

It is hard to find a perfect machine learning solution when the answer to the problem is subjective. However, we can still create a tool that is able to go through more data than a human ever could, which gives us the opportunity to divide users into theoretic groups. In this project we were able to use data

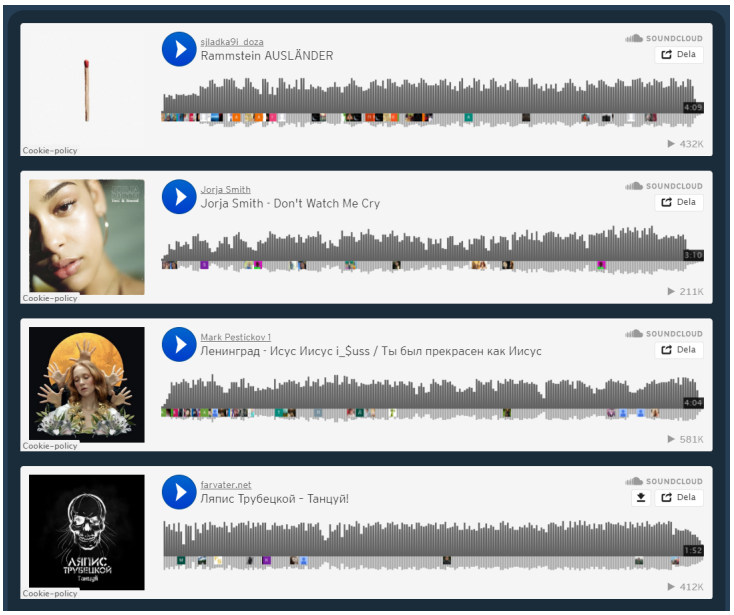


Figure 3: The recommended tracks are shown in a list consisting of embedded players provided by Soundcloud

about a user from Spotify to find a group of users on Soundcloud who likes the same kind of music, at least the same kind of genres.

In the beginning of this report we stated a question: *How can we determine which tracks are considered to be good recommendations for our user, using the available data from Soundcloud?*. The general algorithm for solving this question is summarized below:

1. Use Spotify API to fetch the top artists for the end user and calculate a mean genre vector.
2. Use Soundcloud API to collect lots of songs and keep track of which user has liked each song.
3. For each user from Soundcloud, calculate a mean genre vector.
4. Build a feature space from the Soundcloud mean genre vectors.
5. Match our user with n nearest Soundcloud users using Nearest Neighbor algorithm.
6. Go through all n users and pick out songs using a weighted algorithm. Songs that many of the n users like are considered a good match.

References

[1] Spotify, *Authorization Guide*, Retrieved: 27/01/20
https://developer.spotify.com/documentation/general/guides/authorization-guide

[2] Soundcloud, *HTTP API Reference*, Retrieved: 27/01/20
https://developers.soundcloud.com/docs/api/reference