Numan Tok, Lily Djami,
Ilir Hajrullahu Systems and Software Engineering I 02.02.2023

Spotify Group Playlist ML Model Design

Problem to solve:

Create A Music Playlist, that matches the music tastes of different people.

The necessary Data for this task will be provided by the Spotify project group (with the help of the Spotify API). This may include Song Metadata (Artist, Genre, bpm, mood…), User Playlists etc.

The Spotify project group proposed 2 approaches:

1. All-in-one flow: We get the data we want and create the whole playlist from that. The playlist should be ready to use (Playlist created with the Spotify API)
2. Splitted flow: We get a (rule-based) preselection of songs to use as training data and should recommend x additional songs that match the songs in the preselection

Overall ML Process:

1. Get the necessary (model dependent) data
2. Apply Preprocessing
    1. numerical transformations
        1. Fill missing values with the mean (variable)
        2. Scale features (e.g. normalization)
    2. categorical transformations
        1. Fill missing values with "missing" (variable)
        2. turn into numbers: one-hot encoding
3. Train the ML model
4. Validate/finetune the Model parameters with the validation set
5. Test the final model against the test set

Possible Solutions:

We propose 2 potential solutions, namely a Clustering+ANN hybrid model and a Graph solution.

Solution 1 - Hybrid Model (Clustering + ANN):

We use a combination of two ML models, namely Clustering and an ANN.

Input (All-in-one-flow):

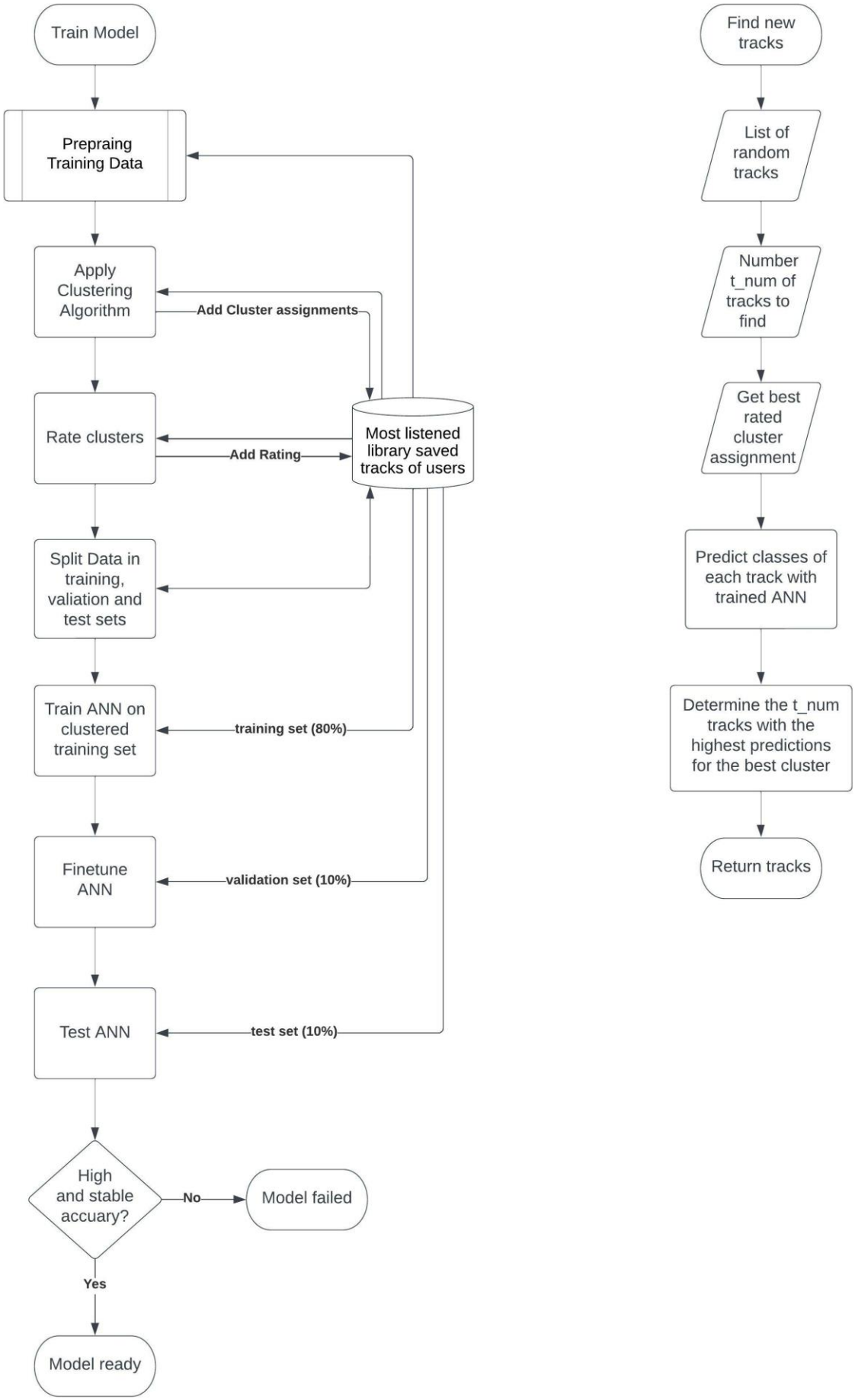
- all available metadata for the 100 (variable) most listened library saved tracks of each user
- t_num = Number of songs to add

Clustering:

- Try to find the most common similarities of the tracks by
    1. grouping the tracks into clusters based on the similarity of the track data
    2. finding the most promising cluster to use it as an ideal feature characteristics representative
        - Rate by factors number of tracks, listening count, user balancing by multiplying:
            - Percentage of covered tracks
            - Percentage of covered number of times the tracks were listened → This information seems to be archived by Spotify. Solutions:
                - Discard this information → all songs are equally liked
                - Or rank songs by the date they were added (more recent = higher ranking)
            - Normalized standard deviation of the track counts per user

Classification / ANN

- Train a Classification model (here ANN) on with the track data as input
- Use the cluster assignments as class to learn
- Predict the class of new songs → The higher the predicted value for the most promising cluster, the more suitable it is for the playlist


Train Model
Prepraing Training Data
Apply Clustering Algorithm
Add Cluster assignments
Rate clusters
Add Rating
Most listened library saved tracks of users
Split Data in training, valiation and test sets
Train ANN on clustered training set
training set (80%)
Finetune ANN
validation set (10%)
Test ANN
test set (10%)
High and stable accuary?
No
Model failed
Yes
Model ready
Find new tracks
List of random tracks
Number t_num of tracks to find
Get best rated cluster assignment
Predict classes of each track with trained ANN
Determine the t_num tracks with the highest predictions for the best cluster
Return tracks

Solution 2 – Graph Data structure and finding path:

We use a directed Graph data structure, where a node represents a song and a path through the graph represents a playlist.

Input (Splitted Flow):

- user_playlists = all playlists of the users & their numerical metadata
- playlist_start = The first song for a playlist & its numerical metadata
- all_songs = As many random songs as possible (possibly all songs in Spotify) & their numerical metadata
- t_num = Number of songs to add

Calculate the importance of each feature as weights:

Here is some pseudocode of the calculation algorithm. A smaller weight indicates a more important feature. This will get us smaller vertices weights in the graph.

```
dist = Vec(0)

for each playlist in user_playlists:
        for each song in playlist:
                last_song = playlist[-1]
                if song != last song:
                        dist_s = Vec(0)
                        dist_s = differences between the features of song and the next song
                        dist += dist_s
dist_sum = sum(dist)
dist_weights = [(i/dist_sum) for i in dist]    // normalization of weights
```

Build a graph data structure:

- Nodes = all_songs (+ playlist_start if not already included)
- Vertices = sum of differences of the features weighted with dist_weights
    - Note: We could also try to add categorical metadata but instead of calculating the difference of them, we could add fixed values for the cases that they are equal or unequal (e.g. if song_i[“Artist”] == song_j[“Artist”] then 0 else 0.1)

Find the shortest route including the songs in playlist_start and n additional songs:

- Just look for the nearest neighbor node of playlist_start and select it as the next song. Repeat this n times, each time for the newly selected song (without the already visited nodes)
- Apply a TSP (Traveling Salesman Problem) like algorithm for playlist_start that also uses additional nodes from the graph and a constraint min_additional_nodes = t_num

Note: Further research must be done to find an appropriate algorithm.

Spotify Group Playlist ML Requirements:

Data Requirements:

- At least 100 tracks from each user (with labeling from which users library the track is)
- There should be the same amount of tracks from each user
- Track data should be from tracks that the users saved in their libraries and listened to the most
- Track data should include all metadata and analysis data available
    - Artist
    - Genre
    - Publishing date
    - Length
    - Bpm
    - Mood

      …