Warsaw University of Technology

FACULTY OF
ELECTRONICS AND INFORMATION TECHNOLOGY

EARIN

Project
Song Recommender

Tymon Żarski 310992
Bartosz Peczyński 310703

WARSAW May 25, 2024

Contents

1. Introduction

1.1. Assigned task

Develop a song recommender system from datasets of song attributes. Students may choose one or a combination of datasets from the references below. Feel free to choose any approach to develop a solution; however, you are encouraged to use multiple methods to compare performance.

The goal of this project is as follows:

- Perform exploratory data analysis (EDA) on the dataset to understand data distribution and statistical significance of each feature and observe trends. Develop hypotheses and reasoning that can help when building a model.
- Perform data preprocessing to prepare the data before feeding into the model, e.g., feature cleaning, selection and rescale.
- Split the dataset to create separate training and validation datasets.
- Train and compare regression models (may use ML libraries like Scikit-Learn). Evaluate the model's performance metric and assess the impact of the preprocessing strategy (e.g., contribution of features). Entropy).

2. Datasets

We have chosen two datasets to work with, each of those dataset contains song titles and additional data.

1. Playlists dataset [1]. Most importantly, it contains playlists and songs which are in this playlist.
2. 10 millions tracks dataset [2] - Many metadata of songs, including audio features, such as loudness, danceabillity, etc.

2.1. Overview

This section will be dedicated to the **10+ M. Beatport Tracks / Spotify Audio Features** dataset as it will be a crucial part of our system. Not only is it the foundation for our song type classifier, but it will also later join with the Spotify playlist to generate the popularity metric. The second dataset **Playlists dataset** will be evaluated in the later fork while implementing the second part of our pipeline.

Table ***audio_features*** on shape (4687104, 15) contains columns 'isrc', 'acousticness', 'danceability', 'duration_ms', 'energy', 'instrumentalness', 'key', 'liveness', 'loudness', 'mode', 'speechiness', 'tempo', 'time_signature', 'valence', 'updated_on' that will partially save a purpose of the features used to classify the type of songs.

Listing 1. "Basic statistics 10 millions trakcs dataset - audio_features table"

```
       acousticness  danceability   duration_ms        energy  \
count  4.687104e+06  4.687104e+06  4.687104e+06  4.687104e+06
mean   1.115222e-01  6.762825e-01  3.395846e+05  7.149000e-01
std    2.241004e-01  1.573557e-01  1.909052e+05  2.097366e-01
min    0.000000e+00  0.000000e+00  1.000000e+03  0.000000e+00
25%    9.820000e-04  5.980000e-01  2.453330e+05  5.840000e-01
50%    9.170000e-03  7.130000e-01  3.411600e+05  7.500000e-01
75%    8.500000e-02  7.970000e-01  4.135420e+05  8.860000e-01
max    9.960000e-01  1.000000e+00  6.074945e+06  1.000000e+00

       instrumentalness           key      liveness      loudness  \
count      4.687104e+06  4.687104e+06  4.687104e+06  4.687104e+06
mean       6.336123e-01  5.556958e+00  1.746018e-01 -9.215520e+00
std        3.432355e-01  3.690800e+00  1.629871e-01  4.168868e+00
min        0.000000e+00  0.000000e+00  0.000000e+00 -6.000000e+01
25%        3.840000e-01  2.000000e+00  8.390000e-02 -1.105900e+01
50%        8.180000e-01  6.000000e+00  1.100000e-01 -8.640000e+00
75%        8.890000e-01  9.000000e+00  1.910000e-01 -6.590000e+00
max        1.000000e+00  1.100000e+01  1.000000e+00  5.485000e+00

               mode   speechiness         tempo  time_signature       valence
count  4.687104e+06  4.687104e+06  4.687104e+06    4.687104e+06  4.687104e+06
mean   5.422555e-01  8.774978e-02  1.268136e+02    3.959320e+00  3.957805e-01
std    4.982113e-01  8.655872e-02  2.113567e+01    3.182083e-01  2.555503e-01
min    0.000000e+00  0.000000e+00  0.000000e+00    0.000000e+00  0.000000e+00
25%    0.000000e+00  4.440000e-02  1.200000e+02    4.000000e+00  1.760000e-01
50%    1.000000e+00  5.810000e-02  1.260000e+02    4.000000e+00  3.660000e-01
75%    1.000000e+00  8.770000e-02  1.330000e+02    4.000000e+00  5.860000e-01
max    1.000000e+00  9.690000e-01  2.500000e+02    5.000000e+00  1.000000e+00

RangeIndex: 4687104 entries, 0 to 4687103
Data columns (total 15 columns):
 #   Column            Dtype
---  ------            -----
 0   isrc              object
 1   acousticness      float64
 2   danceability      float64
 3   duration_ms       int64
 4   energy            float64
 5   instrumentalness  float64
 6   key               int64
 7   liveness          float64
 8   loudness          float64
 9   mode              int64
```

```
 10  speechiness       float64
 11  tempo             int64
 12  time_signature    int64
 13  valence           float64
 14  updated_on        object
```

To get the rest of the information about the tracks, we need to use three more tables: table **bp_track**, *bp_artist* and *bp_artist_release* to extract the song name and artist performing the song. For that matter, we will use the **isrc** and **artist_id** and **release_id** as foreign keys to join all of the tables.

Listing 2. "Basic statistics 10 millions tracks dataset - bp_track table"

```
Shape: (10685331, 17)
Data columns (total 17 columns):
 #   Column           Dtype
---  ------           -----
 0   track_id         int64
 1   title            object
 2   mix              object
 3   is_remixed       object
 4   release_date     object
 5   genre_id         int64
 6   subgenre_id      float64
 7   track_url        object
 8   bpm              int64
 9   duration         object
 10  duration_ms      float64
 11  isrc             object
 12  key_id           float64
 13  label_id         int64
 14  release_id       int64
 15  updated_on       object
 16  is_matched_spot  object
           track_id      genre_id    subgenre_id           bpm   duration_ms  \
count  1.068533e+07  1.068533e+07  715425.000000  1.068533e+07  1.068270e+07
mean   1.012787e+07  2.124393e+01     213.801935  1.207246e+02  3.445434e+05
std    5.188218e+06  2.791291e+01      63.433043  1.912638e+01  2.021613e+05
min    4.971000e+03  1.000000e+00       5.000000  0.000000e+00  0.000000e+00
25%    5.745950e+06  5.000000e+00     210.000000  1.200000e+02  2.505030e+05
50%    1.063708e+07  1.100000e+01     246.000000  1.250000e+02  3.480290e+05
75%    1.472322e+07  1.500000e+01     246.000000  1.280000e+02  4.147200e+05
max    1.815576e+07  9.900000e+01     268.000000  2.580000e+02  2.300677e+07

             key_id      label_id    release_id
count  1.067908e+07  1.068533e+07  1.068533e+07
mean   1.321047e+01  3.889163e+04  2.267419e+06
std    8.739974e+00  2.846908e+04  1.157646e+06
min    1.000000e+00  3.000000e+00  3.400000e+01
25%    6.000000e+00  1.520000e+04  1.365546e+06
50%    9.000000e+00  3.294200e+04  2.298173e+06
75%    2.000000e+01  6.093900e+04  3.235510e+06
max    3.400000e+01  1.162360e+05  4.271957e+06
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10685331 entries, 0 to 10685330
```

2.2. Pre-processing

As mentioned, we are starting by joining all the tables of interest. In that case, we will use only inner join to receive only rows that are present in all of the tables.

```
df_full_track_information = pd.merge(df, df_tract_details, on='isrc', how='inner')
unique_isrc = df_full_track_information['isrc'].nunique()
print(f"unique_isrc: {unique_isrc}")
print(f"df_full_track_information: {df_full_track_information.shape}")

df_full_track_information = df_full_track_information.drop_duplicates(subset=["isrc"])

fd_release_artist_names = pd.merge(
    df_artist_release_keys, df_artist_details, on="artist_id", how="inner"
)
```

```
df_full_track_information = pd.merge(
    df_full_track_information, fd_release_artist_names, on="release_id", how="inner"
)
```

In the end, we only want to receive unique isrc IDs to avoid data publication, and after moving duplicated entries, we received **4667811** rows.

```
Amounts od NA data cells:
isrc                     0
acousticness             0
danceability             0
duration_ms_x            0
energy                   0
instrumentalness         0
key                      0
liveness                 0
loudness                 0
mode                     0
speechiness              0
tempo                    0
time_signature           0
valence                  0
updated_on_x_x           0
track_id                 0
title                  197
mix                      0
is_remixed               0
release_date             0
genre_id                 0
subgenre_id       43708989
track_url                0
bpm                      0
duration              7111
duration_ms_y         7111
key_id               22668
label_id                 0
release_id               0
updated_on_y_x           0
is_matched_spot          0
artist_id                0
updated_on_x_y           0
artist_name            264
artist_url               0
updated_on_y_y           0
dtype: int64
Duplicated values within the dataset: 0
```

There is no data, and there are no duplicated values, but within the dataset, there are columns containing missing values. The columns with missing values are title, subgenre_id, duration, duration_ms_y, and key_id. In case of missing values in the column, the rows containing missing values will be removed to allow further analysis.

2.3. Exploratory data analysis

After joining the tables, we have a lot of redundant data within our dataset. We will start our analysis by defining the song featured and later checking which of them has a direct impact on the data.

The initial song features are described by categories: "title", "artist_name", "isrc", "acousticness", "danceability", "duration_ms_y", "energy", "instrumentalness", "key", "liveness", "loudness", "mode", "speechiness", "tempo", "valence". We will drop the "isrc", "title", and "artist_name" columns as core features that will be valuable for classification.

Listing 3. "Initial song features data frame details"

```
Index(['acousticness', 'danceability', 'duration_ms_y', 'energy',
       'instrumentalness', 'key', 'liveness', 'loudness', 'mode',
       'speechiness', 'tempo', 'valence'],
      dtype='object')
               title      artist_name          isrc  acousticness  \
801  The Whole World       Royal DJ's  ATAQ71300015      0.000452
802  The Whole World           Crew 7  ATAQ71300015      0.000452
803  The Whole World    Geeno Fabulous  ATAQ71300015      0.000452
804  The Whole World    Martin Weleno  ATAQ71300015      0.000452
805  The Whole World  Send and Return  ATAQ71300015      0.000452

     danceability  duration_ms_y  energy  instrumentalness  key  liveness  \
801         0.634       160000.0    0.92             0.876    1     0.345
802         0.634       160000.0    0.92             0.876    1     0.345
803         0.634       160000.0    0.92             0.876    1     0.345
804         0.634       160000.0    0.92             0.876    1     0.345
805         0.634       160000.0    0.92             0.876    1     0.345

     loudness  mode  speechiness  tempo  valence
801    -5.918     1        0.058    128    0.435
802    -5.918     1        0.058    128    0.435
803    -5.918     1        0.058    128    0.435
804    -5.918     1        0.058    128    0.435
805    -5.918     1        0.058    128    0.435
     acousticness  danceability  duration_ms_y  energy  instrumentalness  key  \
801      0.000452         0.634       160000.0    0.92             0.876    1
802      0.000452         0.634       160000.0    0.92             0.876    1
803      0.000452         0.634       160000.0    0.92             0.876    1
804      0.000452         0.634       160000.0    0.92             0.876    1
805      0.000452         0.634       160000.0    0.92             0.876    1

     liveness  loudness  mode  speechiness  tempo  valence
801     0.345    -5.918     1        0.058    128    0.435
802     0.345    -5.918     1        0.058    128    0.435
803     0.345    -5.918     1        0.058    128    0.435
804     0.345    -5.918     1        0.058    128    0.435
805     0.345    -5.918     1        0.058    128    0.435
(3114827, 12)
```

To represent some basic characteristics on the dataset, we presented mean values of the core song characteristics Figure 2.1, most common artists available calculated by the song count Figure 2.1 and distribution histograms of all the features.

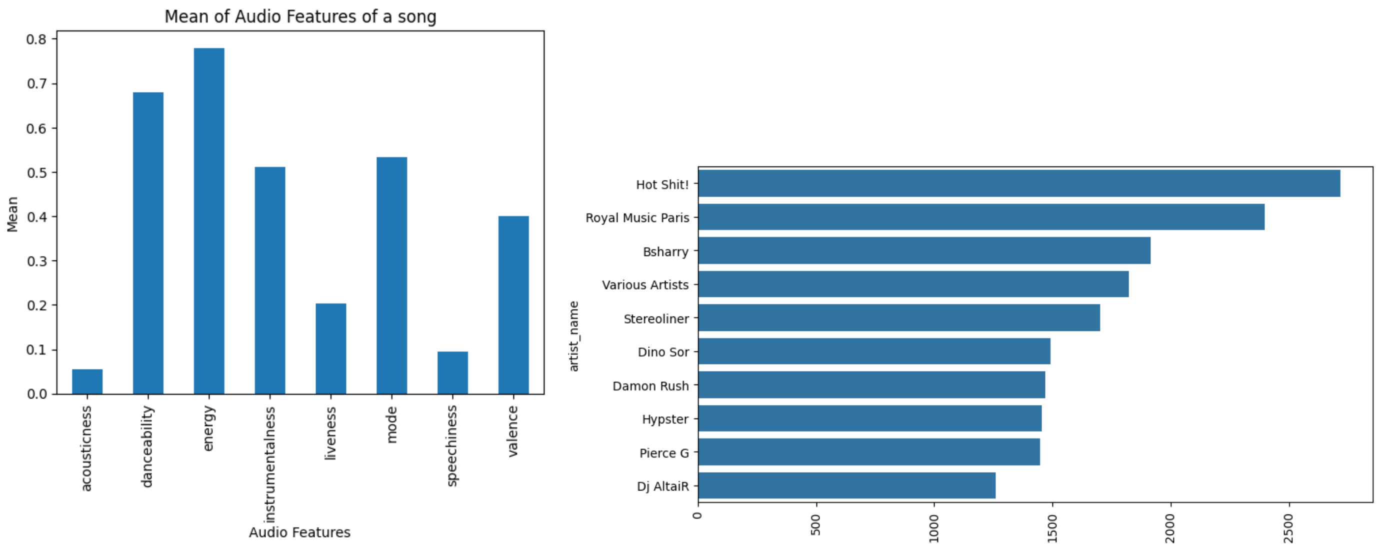


Figure 2.1. Mean values of song core features and most popular artists by song count

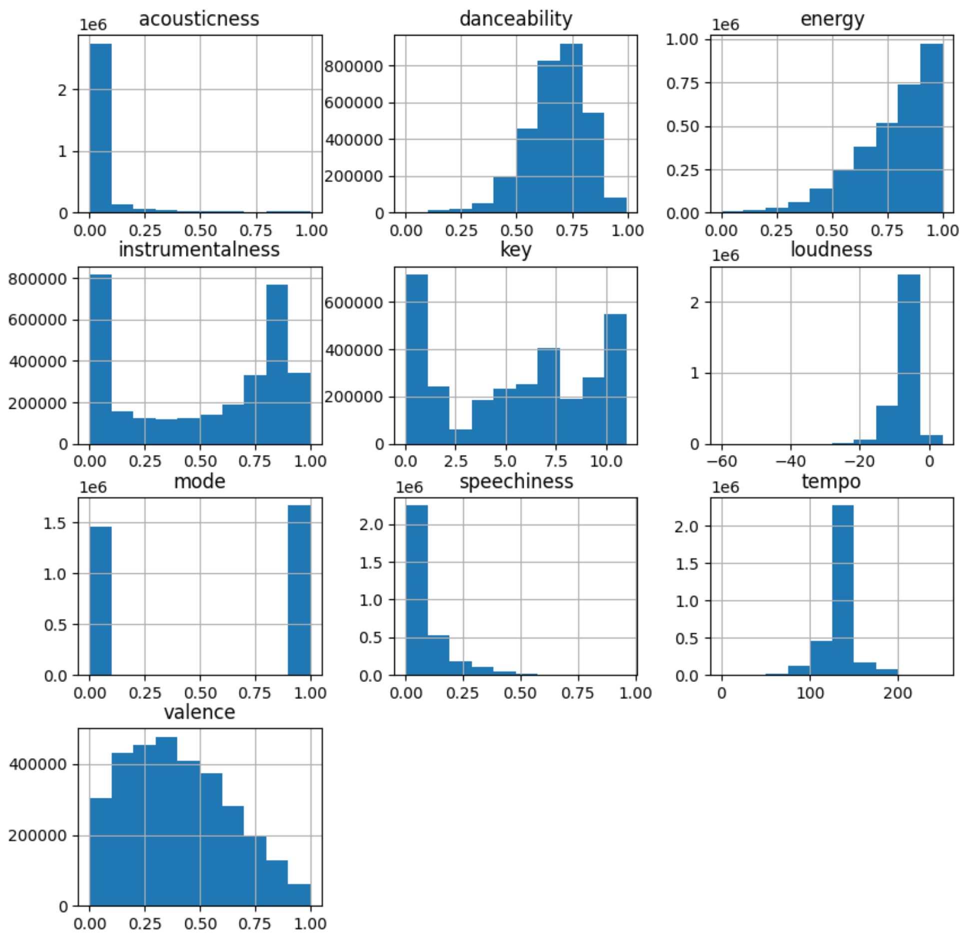
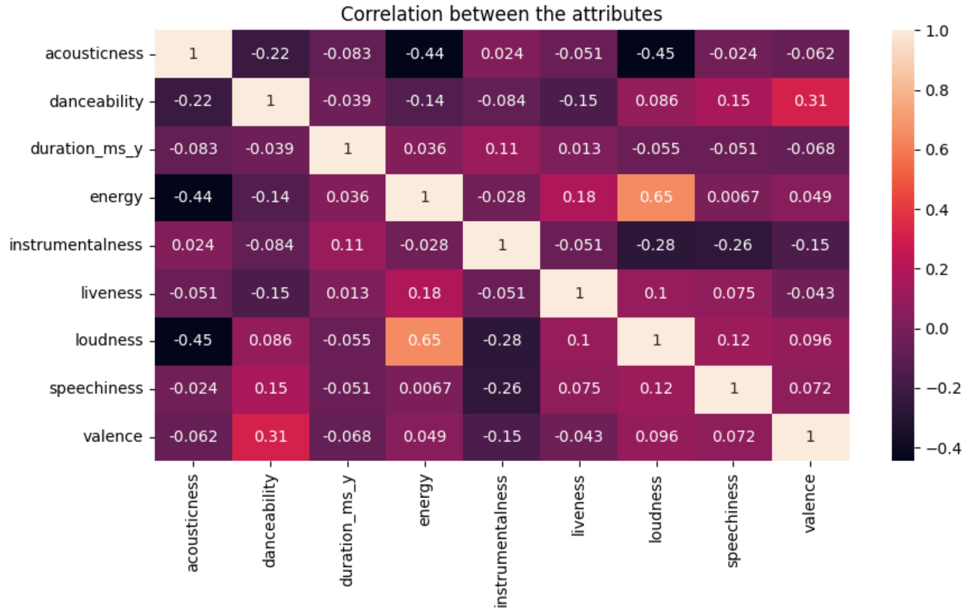


Figure 2.2. Histograms of the distribution of the song features

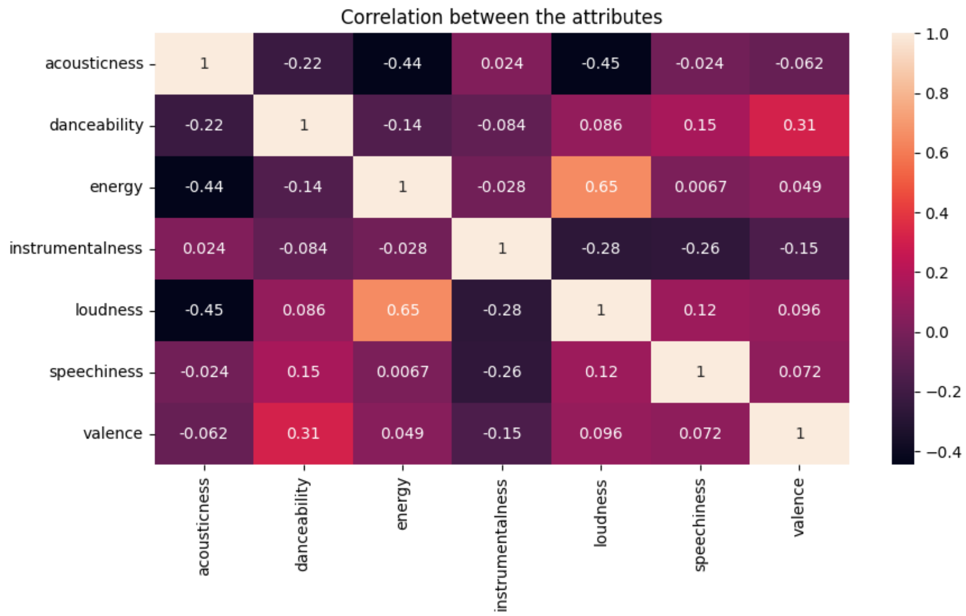
Now, we will analyze the underlying structure and connections of the factors and features within the dataset by creating a correlation matrix Figure 2.3.



	acousticness	danceability	duration_ms_y	energy	instrumentalness	liveness	loudness	speechiness	valence
acousticness	1	-0.22	-0.083	-0.44	0.024	-0.051	-0.45	-0.024	-0.062
danceability	-0.22	1	-0.039	-0.14	-0.084	-0.15	0.086	0.15	0.31
duration_ms_y	-0.083	-0.039	1	0.036	0.11	0.013	-0.055	-0.051	-0.068
energy	-0.44	-0.14	0.036	1	-0.028	0.18	0.65	0.0067	0.049
instrumentalness	0.024	-0.084	0.11	-0.028	1	-0.051	-0.28	-0.26	-0.15
liveness	-0.051	-0.15	0.013	0.18	-0.051	1	0.1	0.075	-0.043
loudness	-0.45	0.086	-0.055	0.65	-0.28	0.1	1	0.12	0.096
speechiness	-0.024	0.15	-0.051	0.0067	-0.26	0.075	0.12	1	0.072
valence	-0.062	0.31	-0.068	0.049	-0.15	-0.043	0.096	0.072	1

Figure 2.3. Correlation matrix of all features

From the correlation matrix, we can see that "loudness" and "energy" are the most correlated features, and "danceability" and "valence" are the second ones. On the other hand, "liveness" and "duration_ms_y" are the least correlated features with the rest of the features, so they can be removed from the dataset **for PCA analysis**. The results can be seen on the Figure 2.4.



	acousticness	danceability	energy	instrumentalness	loudness	speechiness	valence
acousticness	1	-0.22	-0.44	0.024	-0.45	-0.024	-0.062
danceability	-0.22	1	-0.14	-0.084	0.086	0.15	0.31
energy	-0.44	-0.14	1	-0.028	0.65	0.0067	0.049
instrumentalness	0.024	-0.084	-0.028	1	-0.28	-0.26	-0.15
loudness	-0.45	0.086	0.65	-0.28	1	0.12	0.096
speechiness	-0.024	0.15	0.0067	-0.26	0.12	1	0.072
valence	-0.062	0.31	0.049	-0.15	0.096	0.072	1

Figure 2.4. Correlation matrix of all features after dropping features

2.3.1. Exploratory factor analysis (with maximum likelihood)

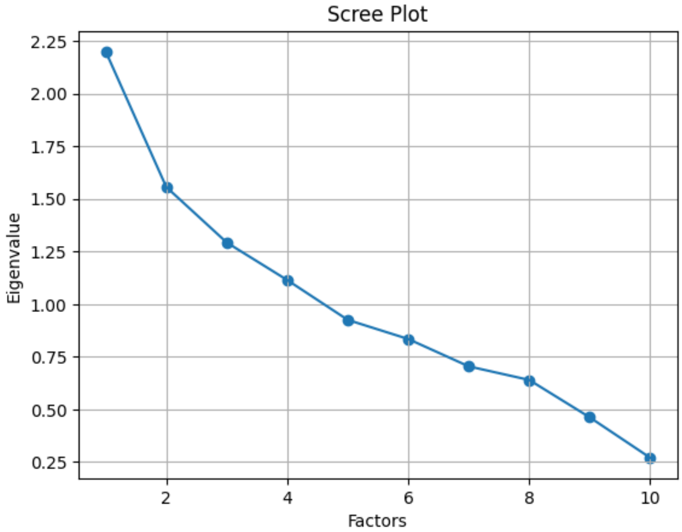


Figure 2.5. Plot of the eigenvalue versus the number of factors

Looking at the scree plot, we can see that the first 4,5 factors bring the most variance to the data. In our analysis, we will use 5 factors.

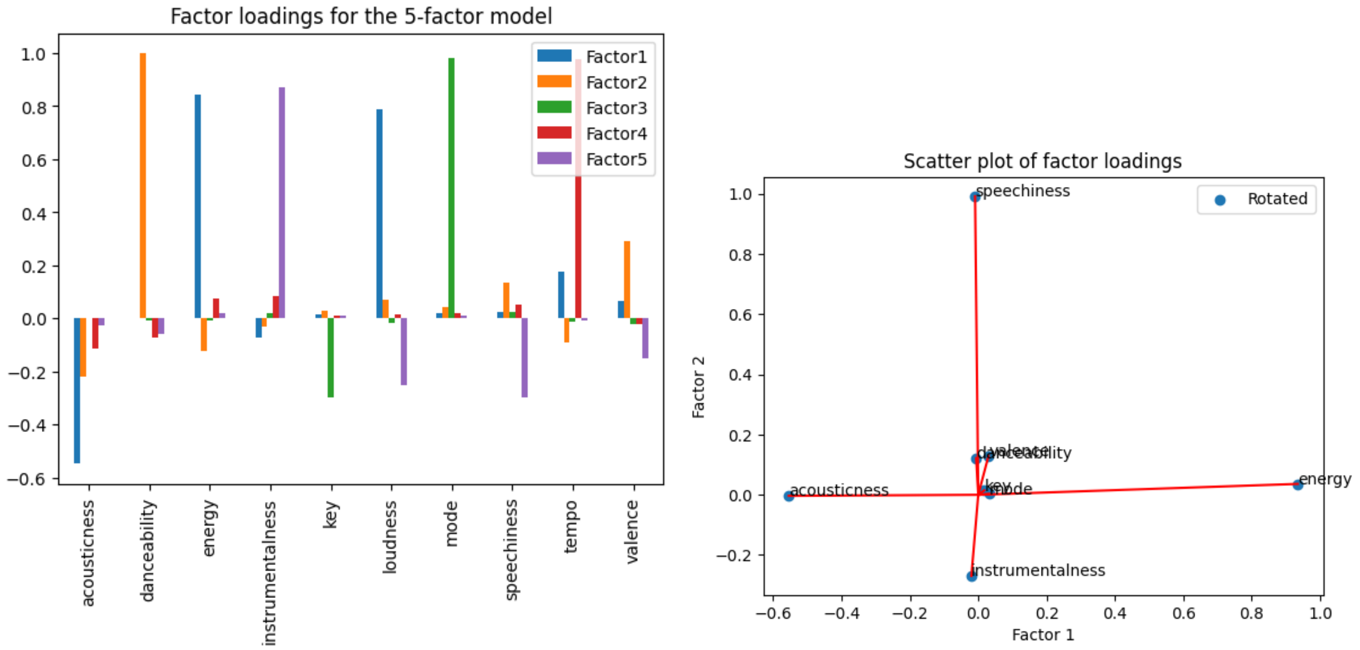


Figure 2.6. Factor loading plot and rotated actor loading plot

By looking at the factors, we will remove the most correlated features to avoid multicollinearity. This way, we should receive the most accurate clustering results. By looking at the loadings, we will remove "loudness" and "tempo" since they are the most correlated features with the factors.

```python
df_correlated_features = df_core_song_features.drop(
    columns=["key", "mode", "instrumentalness"],
)
```

2.3.2. Clustering

Continuing our work with clustering, we will use the elbow method to find the optimal K value in a k-means clustering algorithm. We tested cluster values from 1 to 12, displayed in the Figure 2.7.

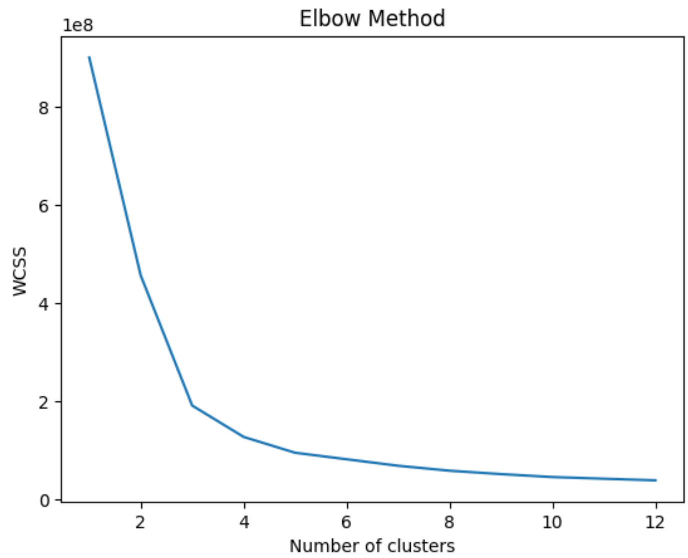


Figure 2.7. Visualization of the elbow method

The elbow plot shows that the most accurate number of clusters is 3 or 6. To make that decision, we will visualize scatter plots of those clustering results. If the results are acceptable, we would like to take the biggest amount for system optimization.

From the Figure 2.8 of flusters displayed by the duration and danceability features, we can see that clusters are split, giving us a reliable result for those features. That gives us hope that we can use a similar approach to training our classifier to optimize the type annotation within our system.

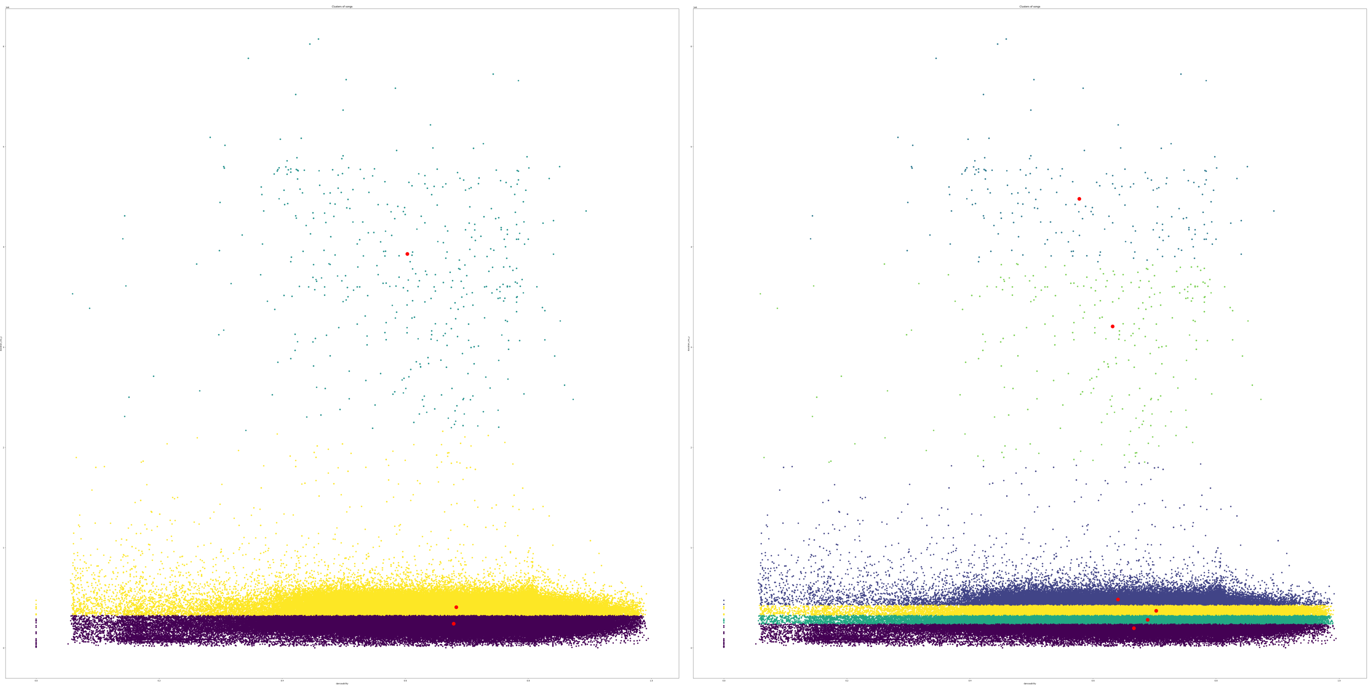


Figure 2.8. Results of plots for the 3 and 6 clusters, respectively

3. Technical Approach

3.1. Architecture

Architecture of our solution can be seen on Figure 3.1, Figure 3.2, Figure 3.3. A description of the operation of this solution can be found below.

1. Take identifiers of one or a couple of songs from the input.
2. Fetch song audio features using Spotify Feature Resolver API.
3. By using a song type classifier, classify input songs to the specific clusters.
4. For each input song, get songs from the cluster they are in and find the most similar songs to the input one among songs from this cluster.
5. Filter songs that satisfy the wanted value of popularity metric.
6. Return found songs to the output.

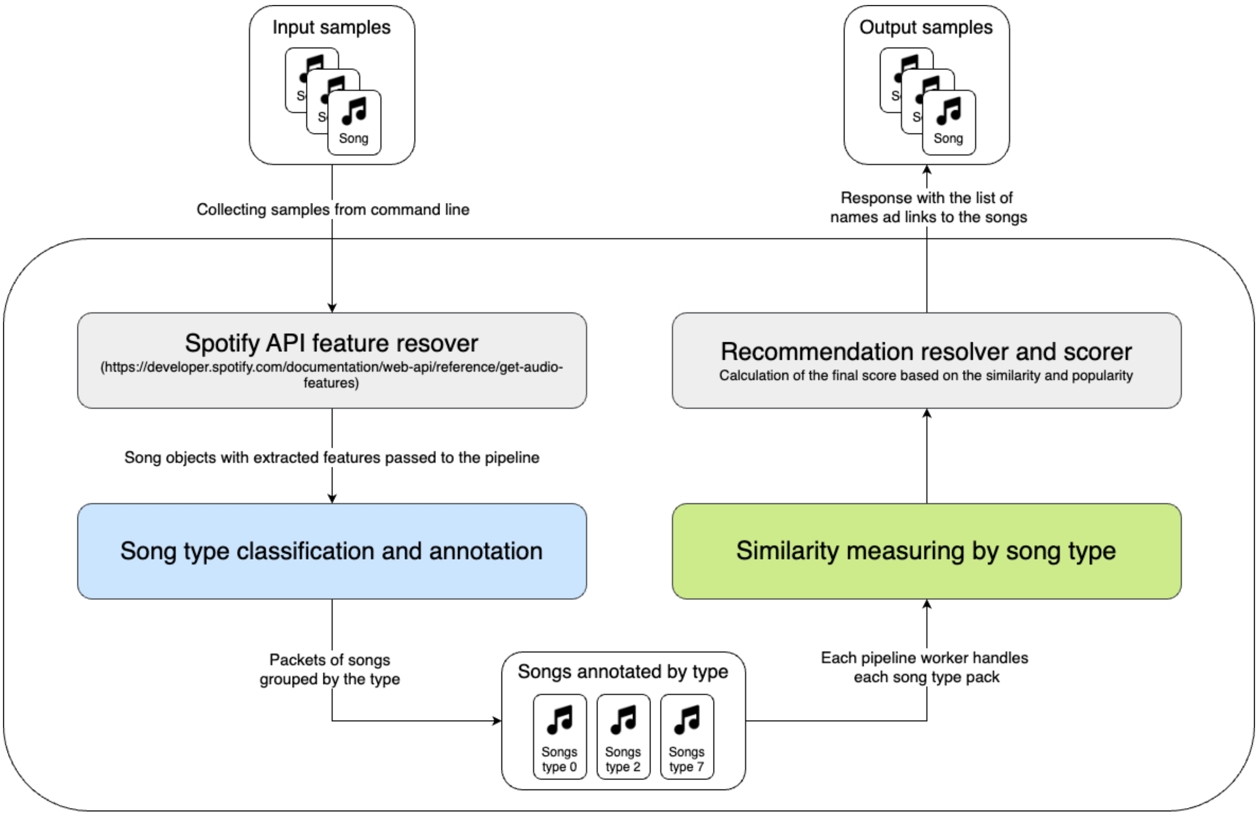


Figure 3.1. Architecture overview

3.1.1. Used algorithms, models

- Clustering - KMeans, probably with the use of MiniBatch - supports a large amount of input data, which will be important when using the classifier with the input dataset, which consists of 10 million records. Moreover, this algorithm is quite fast, what can be seen on comparison from Scikit-Learn [3]
- Calculating popularity - for calculating the popularity of songs, we will find the release the song is in and take the popularity of this release. To determine the popularity of the selected song, we will use

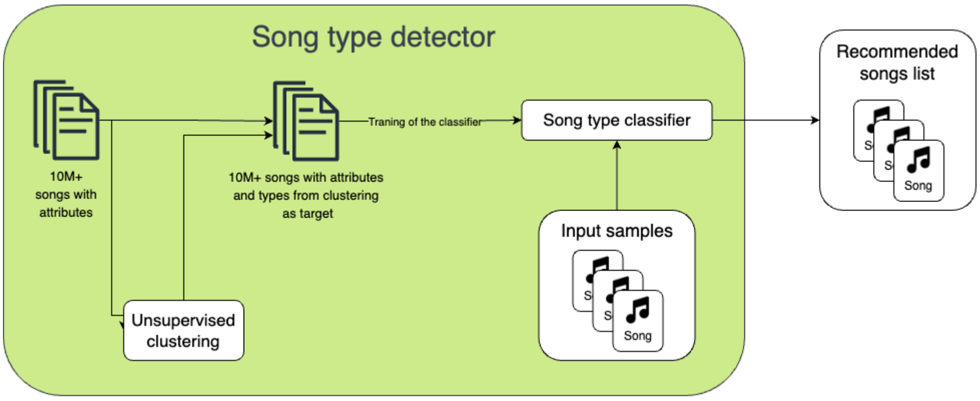


Figure 3.2. Song type detector architecture

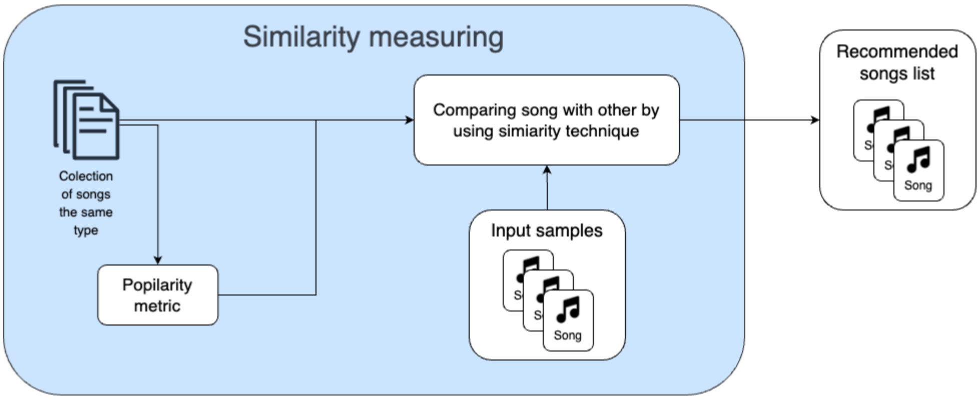


Figure 3.3. Similarity measuring architecture

values from the popularity column from file sp_releases, which we will divide by a number of songs in the release.

- Displaying song recommendations: In the end, the script displays recommended songs, showing their names, artists, and external URLs.

Additionally, command-line argument parsing is incorporated to allow users to specify the song name and popularity threshold directly.

Moving over to the heart of the system being implemented by the clustering.py, it handles cluster fitting, similarity calculation and data manipulation, some of the key functions are:

1. **train_clustering**: For clustering we used MiniBatchKMeans algorithm. The training process starts with loading and preprocessing the dataset, then configuring and training the clustering model with a specified number of clusters. In the end, the trained model is saved using Joblib for future use. This model will later be used to predict the cluster of new songs based on their audio features.
2. **find_most_similar** - This function calculates the cosine similarity between a given song (wanted) and a list of candidate songs within the same cluster (candidates). After preprocessing the candidates, it computes the similarity scores and returns the top five most similar songs. This method is particularly effective for recommending acoustically similar songs to a given track.
3. **find_popular** - The goal of this method is to judge the tracks based on their popularity. The popularity

metric is extracted by using the release popularity feature and then dividing it by the number of songs. It merges the song dataset with additional popularity data (loaded separately) and filters songs to only include those exceeding a specified popularity threshold. This ensures that the recommendations match the user's musical taste in audio features and are popular enough according to the defined metric.

Ultimately, the fundaments of our song recommendation system use clustering and similarity metrics to offer relevant and appealing song suggestions based on audio features.

3.2. Preliminary similarity results

The exemplary results of our system can be seen by running a command:

```
python main.py "Shape of You" --popularity 20
```

The result of the following command is 5 recommended songs with their similarity measures; for now, we are displaying 5 songs instead of the 2 we stated previously for debugging purposes.

```
Midnight Sadness
['Besomorph', 'broke', 'RIELL', 'WISNER']
{'spotify': 'https://open.spotify.com/track/34zqbYA9IoNzf1S6FUGaWd'}
Similarity: 0.9208420442530053

Paradise - Acoustic
['Stevie Appleton']
{'spotify': 'https://open.spotify.com/track/1uKavGXL05NzhPHSOUIg91'}
Similarity: 0.8913046067511546

Me Falta el Aire
['Puppy Sierna', 'Dayvi', 'HmGipsy']
{'spotify': 'https://open.spotify.com/track/4Qx9VNry4A0cmQ2bkeXsdi'}
Similarity: 0.7685539886283258

Ruger
['Ruger']
{'spotify': 'https://open.spotify.com/track/4fVN8qEZF8TQo2ybK8UUHj'}
Similarity: 0.6715958015888648

Once Upon a Time
['Max Oazo', 'Moonessa']
{'spotify': 'https://open.spotify.com/track/0KSoYBtvBJyN361xysQkic'}
Similarity: 0.6489239038351012
```

From the one million samples randomly chosen, we picked 5 samples with the best similarity, varying from 0.92 to 0.64. Knowing the sound of the "Shape of You," we can listen to the recommended songs and hear similar flows and song vibes. The tool works well for more popular music genres, mostly because of the many possibilities.

3.3. Experiment analysis

We tested our solution on different parameters to determine better song similarity results. We tried to increase the correctness of clustering by decreasing the batch size and choosing a different number of clusters. Additionally, we disabled the convergence detection based on inertia by setting max_no_improvements to *None*. We will conduct more experiments by using different cluster sizes when we have a metric for evaluating recommendation accuracy. Additionally, we will asses its influence on the time needed for the program to find the recommendation. The next thing that we tested in our system is the popularity feature. We concluded that this part is an objective parameter, so we used it as an input variable to be defined by the system user.

4. References

[1] *'spotify playlists' - playlists dataset*, Accessed at 10.05.2024, `https://www.kaggle.com/datasets/andrewmvd/spotify-playlists`.

[2] *10+ m. beatport tracks / spotify audio features*, Accessed at 10.05.2024, `https://www.kaggle.com/datasets/mcfurland/10-m-beatport-tracks-spotify-audio-features`.

[3] *Comparing different clustering algorithms on toy datasets*, Accessed at 11.05.2024, `https://scikit-learn.org/stable/auto_examples/cluster/plot_cluster_comparison.html`.