Spotify Song Popularity & Content-Based Recommendation System

Torre Elia
University of California - San Diego

March, 13, 2022

Abstract

This paper presents the research undertaken in the Final Project for DSC190 - Introduction to Data Mining course. The purpose of this project is dual. In the first part of the associated notebook DSC190 - UCSD Repository, Spotify Songs Dataset will be explored with the aim of developing and fine-tuning two Regression Models (XGBRegressor and RandomForestRegressor) to predict the popularity of a song and evaluating their performance against a baseline model (LinearRegression). In a second section, in light of a deeper understanding of the dataset and its characteristics, we delve into the construction of a Content-Based Recommendation System which relies on "vectorization" and Cosine Similarity. In this second part of the project, the user can interact with the system to receive song recommendations starting from one of its Spotify playlists.

I. Dataset

i. Data Sources

The dataset used to perform the following analysis is a dataframe containing approximately 170k songs, it covers an heterogeneous range of genres on a time-frame from 1921 to 2020. It can be retrieved from Kaggle at: Kaggle-Dataset. In particular, the data source presents 5 different datasets: "data.csv", "data_by_artist.csv", "data_by_genres.csv", "data_by_year.csv", "data_w_genres.csv". "data.csv" has been used to perform EDA and to perform the analysis relative to the Song Popularity Prediction Task, while "data_w_genres.csv" has been exploited to merge its information with the previous dataset in the context of building the Content-Based Recommendation System.

ii. Features Explanation

The dataset used for the regression task presents 19 features, the following definitions are taken from Spotify Developer Website and compensed with basic statistical analysis regarding type and ranges:

- **Valence**: It describes the musical positiveness conveyed by a track. Tracks with high valence sound more positive. Numerical, continuous. Ranges from 0 to 1.
- **Year**: The year of release of the song. Numerical, Ordinal. It ranges between 1921 and 2020.
- **Acousticness**: Songs with high "acousticness" will consist mostly of natural acoustic sounds while songs with a low "acousticness" will consists of mostly electronic sounds. Numerical, continuous. Ranges from 0 to 1.

- **Artists**: The names of the artists of the song. Text feature
- **Danceability**: Danceability describes how suitable a track is for dancing based on a combination of musical elements. Numerical, continuous. Ranges from 0 to 1.
- **Duration_ms**: The duration of the songs in milliseconds. Numerical, continuous. Typically ranges from 200k to 300k with some exceptions.
- **Energy**: It conveys a measure of intensity and activity. Energetic tracks generally feel fast, loud, and noisy. Numerical, continuous. Ranges from 0 to 1.
- **Explicit**: A boolean feature describing the presence of Explicit contents in the song.
- **Id**: The id of the track within Spotify systems.
- **Instrumentalness**: It conveys a measure of intensity and activity. Energetic tracks generally feel fast, loud, and noisy. Numerical, continuous. Ranges from 0 to 1.
- **Key**: The tonality of a song. All keys on an octave are encoded as values starting on C as 0, C# as 1 (and so on...) and ending on B as 11. Categorical, ordinal. Ranges from 0 to 11.
- **Liveness**: This value describes the probability that the song was recorded with a live audience. Numerical, continuous. Ranges from 0 to 1.
- **Loudness**: Loudness is a perception that determines how much sound pressure a particular source is emitting at a given time. Numerical, continuous. Typically ranges from -60 to 0 db.
- **Mode**: Mode indicates the modality (major or minor) of a track, the type of scale from which its melodic content is derived. Boolean.
- **Name**: The name of the track. Text feature.
- **Popularity**: The popularity of a song in terms of Spotify's streams. Numerical, discrete, our target variable. Ranges from 0 to 100.
- **Release_date**: The date of release of the song.
- **Speechiness**: Speechiness detects the presence of spoken words in a track. Numerical, continuous. Ranges from 0 to 1.
- **Tempo**: Tempo is the speed at which a piece of music is played. It is measured in beats per minute (BPM). Numerical, Discrete. Typically ranges from 50 to 150.

II. Exploratory Data Analysis

i. Histograms and Boxplots

During the process of exploring the dataset, histograms and boxplots of the features have been plot to have a rough idea of their distributions. In particular, we can notice that:

- We have less data regarding songs released before approx. 1950, which might imply that our models could be less accurate in the timeframe 1921 - 1950.
- Danceability presents a distribution that resembles the normal one, centered at about 0.55.
- Duration, Instrumentalness and Speechiness are left-skewed with a peak at about 0.1.
- Loudness is mostly contained in the range [-20,0].

Then we plotted the boxplots of some chosen features to further explore the presence of outliers. From a quick look at Figure 1, we can notice that "Loudness", "Liveness", "Duration_ms" and "Speechiness" might be affected by outliers. We will proceed in a cleaning of these features in the Pre-Processing section.

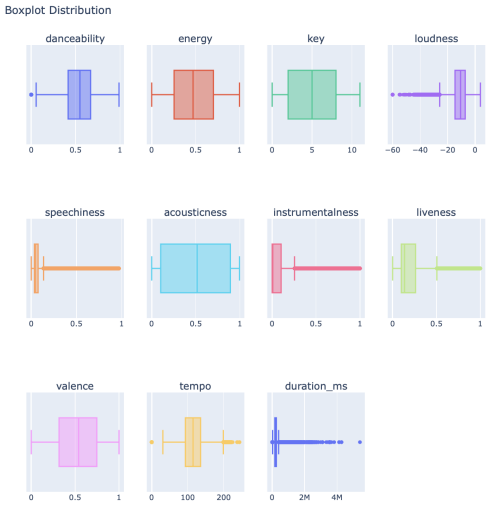

Figure 1: *Plotly Boxplots of "data.csv" features*

ii. Correlation Heatmap

Then we proceed in plotting a Correlation Heatmap (Figure 2) among the features in the dataset to explore the correlation between our target variable (Popularity) and the other features of the dataset. In particular, we can notice that the most positively correlated features with "Popularity" are: "Year", "Energy" and "Loudness". While among the negative correlations, the most significant are: "Acousticness", "Instrumentalness" and "Speechiness". Finally, the least correlated feature is: "Key".

A deeper analysis, with relative plots, of the relation between "Popularity", "Year" and "Key" is presented in the notebook. However, the main finding has to do with a peculiar increasing trend in the popularity of songs over the years. Since Spotify was founded in 2006 and became popular only around 2010, we might infer that the song released before that period could be affected in their popularity estimation. Furthermore, the increasing trend in recent years could be as well explained by a simultaneous increase in popularity of Spotify.

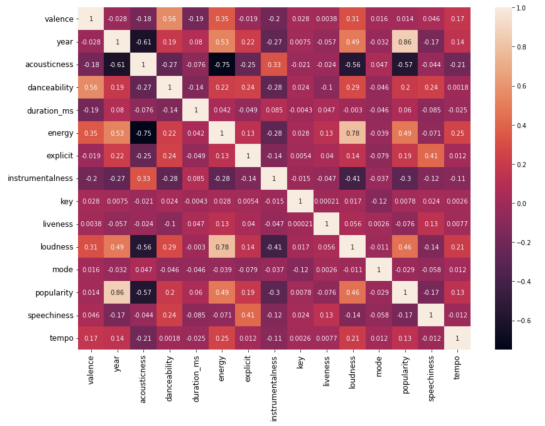

Figure 2: *Correlation Heatmap of "data.csv" features*

For what concerns the "Pre-Processing" of the datasets, we have to distinguish between the preprocessing performed for the "Regression Task" (Song Popularity Prediction) and the one performed for the "Content-Based Recommendation System", the same applies also for "Models and Training" and "Results". Hence, this paper will firstly address the Regression Task to move afterwards on an analysis of the work done on the Recommendation System.

III. Regression Task

i. Data Pre-Processing

The dataset does not present any missing value, hence no imputation will be required. However, as previously hinted from the analysis of the boxplots, we are going to perform a data cleaning process on the following chosen features:

- **Loudness**: since humans are not capable of distinguishing frequencies approximately below 20Db, I will clip any value below -20 to -20.
- **Liveness**: since it defines a probability estimation of whether the track was recorded in studio or live, hence I will encoded it as <= 0.5 (Studio), > 0.5 (Live).

- **Duration_ms & Speechiness**: since audiobooks and movies are also present in this dataset, I will drop all the entries with a duration >= 650k ms and with values of "speechiness" > 0.25 (i.e., the outliers).

Finally, I drop the columns: **artists**, **id** and **name** as they are either not useful or text features that would require an ad-hoc analysis to be exploited. During this process the dataset went from approximately 170k entries to approximately 155k.

ii. Models and Training

Since we are dealing with a regression task, i.e., we are regressing the features of the dataset to predict the popularity of a song, **LinearRegression** will be used as a baseline model. Indeed, it represents the simplest model and "building block" of regression analysis. Then, **XGBRegressor**, i.e. a very robust boosting algorithm which is often exploited in Data Science competitions due to its ability of achieving good performance with relatively simple hyperparameters optimization, will be implemented. In addition,also **RandomForestRegressor**, i.e., a tree-based decision algorithm that exploits bagging and randomness, will be tried. The performance of these three algorithms will be evaluated according to **Mean Squared Error**. The MSE obtained by the three architectures before any hyperparameter tuning is the following:

- LinearRegression: 119.17
- XGBRegressor: 99.15
- RandomForestRegressor: 95.68

Then, we proceed in the process of Hyper-Parameters Tuning, which has been performed through **RandomizedSearchCV**. In particular, the following search schemes have been used for the two models:

- XGBRegressor (30 iterations) tuned parameters:
  - "learning_rate"
  - "n_estimators"
  - "subsample"
  - "max_depth"
  - "min_samples_split"
  - "min_samples_leaf"
- RandomForestRegressor (25 iterations) tuned parameters - value:
  - "max_depth"
  - "min_samples_split"
  - "min_samples_leaf"
  - "max_features"

In addition, the feature_importance values of the tuned models can be found in the notebook.

iii. Results

The results discussed are obtained on a Train/Test split of 80/20 and cross-validated through KFold with a cv of 5. As we can notice from the MSE achieved with the "not-tuned" models, XGB and RandomForest already outperform the Linear-Regression baseline. In particular, RandomForest achieves a better performance than XGB. On the contrary, after hyperparameter tuning, the performances of the two models are the following:

- XGBRegressor: 93.98
- RandomForestRegressor: 95.28

It can be noticed that RandomForest did not increase significantly its performance, while XGBRegressor decreased of approximately 5 points the MSE. The performance of the two algorithms are also evaluated in cross-validation to avoid problems related to overfitting, where they achieve the following average performances:

- XGBRegressor: 94.78
- RandomForestRegressor: 95.66

Finally, as the performances of the two models were relatively similar in cross-validation, some ensemble algorithms were exploited to exploit eventual complementarity of the two models. The results

are evaluated in KFold cross-validation as previously and the average performances are reported:

- VotingRegressor: 93.93
- StackingRegressor: 92.96
- StackingRegressor (+ ElasticNetCV): 93.55

As we can see, the scores achieved by the ensemble models are an improvement over the single models.

The results are above our expectations, even if a MSE of approximately 93 is still too high to have significant information and exploit this algorithm in production. The performance could be increased by adding the dropped text features. However, if we were to transform this regression task in a classification task where we bucket the popularity feature in bucket of 10 points size, we could achieve a significant accuracy. Further improvements rely in more granular data, which is for sure in possess of Spotify.

IV. Content-Based Recommendation System

i. Data Pre-Processing

As previously mentioned, the datasets used for the recommendation system are "data.csv" and "data_w_genres.csv". In this case, the data cleaning performed on the two datasets is the following:

- Extract special characters to be able to define a new variable which concatenates artists and songs. It allows then to drop duplicated songs avoiding to drop songs with same title but from different authors.
- Associate each artist to the list of genre in the second dataset to create "artgen". Then we clean the genres to obtain "clean_genre" which is a list of lists containing the genres and we merge it to the first dataset.
- year feature is extracted from "release_date".
- "Popularity" feature is engineered such that buckets of 5 popularity points are defined.
- "Floats" variable is defined and initialized to the float64 features of the dataset to be later scaled.

A more granular description of the code is provided as comment in the notebook.
At this point four functions and an API script are defined:

- **"Encoder"**: it is exploited in the next step to encode and give a name to the features created through TF-IDF.
- **"Vectorizer"**: this function does the following:
  - exploits tf-idf algorithm to encode "clean_genre" such that uncommon genres are weighted more when they appear, the intuition behind is that a very specific genre might be a relevant information to give recommendations.
  - encodes "year" and "popularity" through the predefined "encoder" function.
  - scales "floats" through "MinMaxScaler".
  - concatenates the encoded df.
- "Spotipy" library has been exploited to retrieve the information of the user playlists and their cover images. In particular: "client_id" and "client_secret" can be retrieved from Developer API –> "Login" –> "Create an App". To run the code effectively, the user has to run the code on a local machine (i.e., no Google Colab) and set "redirect_uri" attribute within the token variable, i.e., you can copy and paste (http://localhost:8881/callback=), then going back on the Spotify Dashboard –> "Open the created app" –> "Edit Settings" –> paste the url in "Redirect URIs" –> "Add" –> "Save".

- **"Playlist"**: this function extracts the songs from a specific playlist given as input "playlist_name". In particular, it retrieves:
  - Artist Name
  - Track Name
  - Track Id
  - Track Album Image
  - Date at which the song has been added to the playlist.

  Finally, it sorts the playlist (dataframe) content according to "date_added".
- **"Visualizer"**: this function exploits the images retrieved in the previous "playlist" function to give a visualization of the album covers in a playlist.

ii. Recommendation System Model

The Content-Based Recommendation System Model is based on two main functions:

- **"Featurizer"**: this function represents the core of the recommender system implementation. Indeed, "featurizer" aims at giving a "vectorial" representation of the playlist such that it can be compared to the "vectorial" representation of other songs. In particular:
  - It receives in input the dataframe representing the playlist and stores it after having sorted it according to how recently has been added to the playlist.
  - It defines a variable that stores the dataset without the songs in the playlist (i.e., we don't want to recommend songs already in the playlist).
  - It then proceeds in evaluating the months elapsed from when a song has been added to the playlist, indeed we will assign a weight based on this. (The intuition behind is that a playlist lasts for a longer time than our interests in certain songs and we might not update them so frequently, hence songs recently added might be a more relevant information of our tastes and direct the recommendations better).
  - Finally, to obtain the vector representing the playlist, each feature is multiplied by the previously defined weight and column-wise summed to obtain the playlist weighted vector.
- **"recommender"**: this function aims at comparing the playlist vector obtained through the previous function to the vector of the other songs. In order to do so, "Cosine Similarity" is exploited.

$$\textit{Cosine Similarity} = S_C(A,B) := cos(\theta)$$
$$= \frac{\mathbf{A} \times \mathbf{B}}{||\mathbf{A}|| \times ||\mathbf{B}||} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

Cosine Similarity is of common-use both in Collaborative Filtering and Content-Based Recommender Systems. Intuitively, it computes the angle between two vectors and outputs a score in the range [-1, 1] with -1 meaning opposite directions and 1 meaning same direction. We now have a score over all of the songs in the dataset, we want to rank them for similarity with the playlist vector and extract a sample of the ones with the highest score.

iii. Results

The results of the Recommendation System are not easy to evaluate. However, given an ad-hoc created playlist, which targeted specific genres and artists, the recommendation system suggested songs from the same albums and artists that were not present in the playlist. Since the algorithm was not explicitly instructed to do so, but rather analyse the content of the songs in the playlist and compare it with the others in the dataset, we could consider it as a sign that the system is working properly.

In particular, if we look at the visualizations of the playlist used (Figure 3) and the suggested songs (Figure 4), we can notice what explained above.

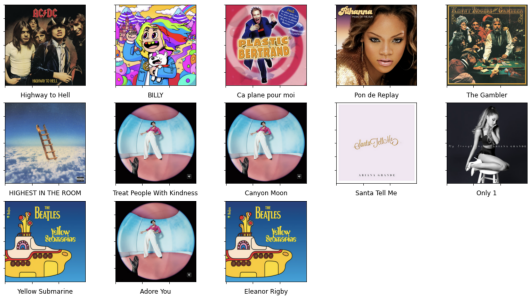

Figure 3: *Playlist created to test the Recommendation System*

Figure 4: *Suggested songs by the Recommendation System*

In Figure 5, it is shown the set of suggested songs for the same playlist by Spotify's algorithms. It is worth noting that among the suggestion of our algorithm and the algorithm of Spotify there is a coincidence, i.e., the first track. Obviously, Spotify algorithm will achieve an higher accuracy thanks to richer datasets and unmatchable computational power which cannot be replicated in this analysis.

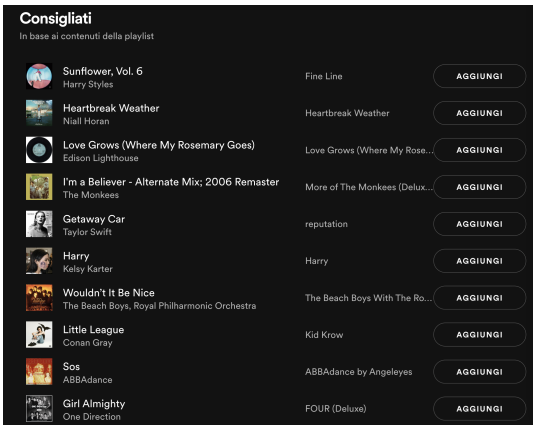

Figure 5: *Suggested songs by Spotify Recommendation System*

V. Literature

i. Regression Task

This dataset has been extensively analysed by the community and there are several notebooks on Kaggle worth a look. In this regard, as a statistics-lover, the implementation of "Bayesian Regression" to predict the popularity of a song was interesting, indeed it gives the opportunity to keep track of the changes that occur to the data in mathematical terms. Furthermore, complex solutions involving neural networks, NLP and data augmentation can be retrieved, which gives us hints on how to further extend this analysis.

ii. Recommendation System

Within the literature is worth mentioning one of the largest limitations of this recommendation system, i.e., the fact that it is able to effectively vectorize only songs in the user's playlist that are matched in the "data.csv". Indeed, this is a limitation, and we estimated that among our playlist approximately 30% of the songs were actually represented in the dataset. For what concerns other implementations of recommendation systems on the same dataset, the one provided by "Vatsal Mavani", which

can be found as a notebook connected to the Kaggle Challenge where the dataset is stored, is interesting as it deals more specifically with the audio features and a clustering of genres according to these audio characteristics. On the other side, the state-of-the-art of Content-Based Recommendation Systems relies on deep neural networks and the most recent challenges include accomplishing context-awareness.

VI. References

- https://thecleverprogrammer.com/2021/03/03/spotify-recommendation-system-with-machine-learning/
- https://www.kaggle.com/vatsalmavani/music-recommendation-system-using-spotify-dataset
- https://arxiv.org/pdf/2107.11803.pdf
- https://machinelearninggeek.com/spotify-song-recommender-system-in-python/
- https://en.wikipedia.org/wiki/Cosine_similarity
- https://towardsdatascience.com/using-cosine-similarity-to-build-a-movie-recommendation-system-ae7f20842599
- https://medium.com/@bkexcel2014/building-movie-recommender-systems-using-cosine-similarity-in-python-eff2d4e60d24
- https://www.kaggle.com/codebreaker619/spotify-eda-and-predictions
- https://www.section.io/engineering-education/building-spotify-recommendation-engine/
- https://www.kaggle.com/konome/data-visualization-spotify-tracks-db
- https://arxiv.org/pdf/2201.10528.pdf
- https://github.com/ddhartma/Spotify-dataset-analysis-160kTracks-1921-2020Modeling