# Spotify Playlists Classifier

**Jaime del Prado Abril,[1] Martín Iglesias Goyanes[2]**
[1]Universidad Carlos III de Madrid
[2]Universidad Carlos III de Madrid
[1]100384114@alumnos.uc3m.es
[2]100384068@alumnos.uc3m.es

**Resumen:**
**Palabras clave:**

**Abstract:** Implementation that using a machine learning model and using Spotify's API to retrieve user-curated playlists, is able to determine user's song distribution among playlists and predict a possible user's playlist for a given song. We detail our data gathering, data analysis and model creations.
**Keywords:** Machine Learning, Spotify, Deep Learning, Embeddings, Data Science, Classifiers, ...

## *1 Background*

The main goal of the project was to create an application that could access into any user Spotify account, retrieve their liked songs and introduce each song to the Playlist that is more along to. Depending in the Artist and Audio Features of the song. This classification will be determined using Machine Learning and Deep Learning Models. The application will have four main sections:

- Data Gathering using Spotify´s API.
- Data Study.
- Models Creation.
- Users Implementation.

This article will include an exhaustive explanation of the three first sections.

### 1.1 Tools

The whole project will be done with Python3. In order to do gather data the Spotfy's API was used.

The Data study and the Model creation will be done in two different Google Collaborative notebooks, *data-handling.ipynb* and *models.ipynb* respectively. Each notebook uses different Python Libraries such: Pandas to handle the Dataframe, Seaborn for plorting in the data-Study, Numpy and Sklearn for the models, predictions...

**All scripts and Collab Notebooks will be in a Github Repository**[1]
This section will include an explanation of how the Dataset used for the rest of the project was created.

To gather Data to train a model, a file containing a list of 84 different Spotify Playlists with their correponding URLs was created and placed in the project Github Repository as *Playlist.txt*.

Using Spotifies API the information of each song was obtained, Information such Name, Artist and its Audio Features.

A Dataframe with every song and their characteristics from every Playlist in *Playlist.txt* were merged resulting 14851 rows. As stated before each row represents a song and has as columns: Name of the song, the Artist, Album, Duration(ms), Explicit, Popularity, Key,

[1]Github Repository available at https://github.com/dev-martin/spotify-playlist-recommender

Mode, Time Signature, Acousticness, Danceability, Energy, Instrumentalness, Liveness, Loudness, Speechiness, Valence, Tempo, Playlist. The Columns will be explained in section [3.2.1]. This DataFrame appears in *clean-playlists-one-artist.csv*

## *2 Data Study*

This Section will include a explanation of the process taken to study the data including three main subsections:

- Data Cleaning
- Feature Study
- Classes Selection

Each section will be detailed in the following subsections:

**The code run is in *data-handling.ipynb*[2]**

### 2.1 Data Cleaning

In this subsection a quick overview of the Data set is done, studying the imbalances of classes, checking missing values and restructuring the columns of the Data set.

First, lets remind that the Dataframe had as rows each song and the columns were values that categorized each song such: Artist, Name or all the Audio features and the last column was the Playlist that it belonged to.

Since the main goal of the project was to classify a song into one of the different Playlists of the user, it is concluded that the different Playlists are the different Classes that the output could take.

#### 2.1.1 Checking Imbalances

Once our classes are determined, the imbalances of classes were taken in consideration. It resulted that some Playlists had a much higher number of songs than others, this scenario would create a imbalanced Dataset and to solve it, a Sampling Process was executed. Every Playlists with more songs than the average length (168 songs) were sampled to that value.

#### 2.1.2 Missing Data Study

After checking any imbalances the Missing Data was handled. The steps taken during this proccess were:

Firstly, see the playlists have missing values and how much percentage of the each playlist has missing values. The results obtained were that the Classes with most nulls/NaN in their songs had around 53 percent of the total songs. Secondly, Removing all nulls will only significantly affect four classes(above 50 percent of nulls in their songs), Though it was step that must be taken.

#### 2.1.3 Restructuring the DataSet

The last step taken in the Data cleaning was a restructuring of the DataSet, This step included two main steps:

Firstly, All collaborations of artists were avoided and only the main Artist (The first in the Artists Column) was kept. The reason is that it is bit desired to give more importance to playlists (classes) which have songs with a lot of collabs. Since that would make the model bias towards the playlists with most collabs, instead of generalising.

Secondly, some columns were renamed to new names and the Final DataFrame was saved as *clean-playlists-one-artist.csv*. This will be the Dataset that will be modified and imported from now on.

### 2.2 Feature Study

In this subsection a detailed study of all the features was done, This study will include:

- Feature Explanation.
- Feature statistical study.
- Feature Correlation.
- Feature Conditional Entropy.

[2]data-handling.ipynb available at https://colab.research.google.com/drive/1ctWeVb-b3u0u3c3lEb6WfhqqJ3XgfCwUscrollTo=sQaexvvKh$_1$J

- Deductions.
- Analysis of Doubtful features.

At the end of this section a Set of Features will be selected as the strongest compared to the rest.

### 2.2.1 Feature Explanation

The Dataset has the following Features: Name, Artist, Album, Duration, Explicit, Popularity, Key, Mode, Time Signature, Acousticness, Danceability, Energy, Instrumentalness, Liveness, Loudness, Speechiness, Valence, Tempo. Each of them provides different information.

- **Name**: Name of the Song (Categorical).
- **Artist**: Artist of the Song (Categorical).
- **Album**: Album of the Song (Categorical).
- **Duration**: Duration of the Song (Continuous).
- **Explicit**: Boolean for explicit or not (Boolean).
- **Popularity**: The popularity of the track. The value will be between 0 and 100, being 100 the most. (Continuous)
- **Key**: The estimated overall key of the track. Integers map to pitches using standard Pitch Class notation .If no key was detected, the value is -1. (Continuous)
- **Mode**: Mode indicates the modality (major or minor) of a track, the type of scale from which its melodic content is derived. Major is represented by 1 and minor is 0. (Boolean)
- **Time Signature**: An estimated overall time signature of a track. The time signature (meter) is a notational convention to specify how many beats are in each bar (or measure).
- **Acousticness**: A confidence measure from 0.0 to 1.0 of whether the track is acoustic. 1.0 represents high confidence the track is acoustic. (Continuous)
- **Danceability**: Danceability describes how suitable a track is for dancing based on a combination of musical elements including tempo, rhythm stability, beat strength, and overall regularity. A value of 0.0 is least danceable and 1.0 is most danceable (Countinuous)
- **Energy**: Energy is a measure from 0.0 to 1.0 and represents a perceptual measure of intensity and activity. Typically, energetic tracks feel fast, loud, and noisy. (Continuous)
- **Instrumentalness**: Predicts whether a track contains no vocals.(Continuous)
- **Liveness**: Detects the presence of an audience in the recording. Higher liveness values represent an increased probability that the track was performed live.(Continuous)
- **Loudness**: The overall loudness of a track in decibels (dB). Loudness values are averaged across the entire track and are useful for comparing relative loudness of tracks.(Continuous)
- **Speechiness**: Speechiness detects the presence of spoken words in a track. The more exclusively speech-like the recording, the closer to 1.(Continuous)
- **Valence**: A measure from 0.0 to 1.0 describing the musical positiveness conveyed by a track. Tracks with high valence sound more positive (e.g. happy, cheerful, euphoric), while tracks with low valence sound more negative (e.g. sad, depressed, angry).(Continuous)
- **Tempo**:The overall estimated tempo of a track in beats per minute (BPM). (Continuous)

### 2.2.2 Feature Statistical Study

This is a brief section that gives the overall statistics for each Continuous features. The output is shown in the Google collab Notebook. It includes the mean, std, min value, the quarterlies and max value for each Continuous Feature.

### 2.2.3 Feature Correlation Study

In this subsection the correlation matrix using Pearson's Formula will be calculated. A correlation matrix is a table showing correlation coefficients between variables. Each cell in the table shows the correlation between two variables.

This Matrix would only include the Continuous variables. The way to calculate the Correlation Matrix including the Categorical Features is using the library: *dython*.

The Correlation matrix provide us important information such: Energy and Loudness are positively correlated by a 0.74, Energy and acousticness are negatively correlated with a -0.75.

If Categorical Features are considered, the correlation of each feature with the Target Value (Playlist) can be studied. As it appears in the notebook, Artist, Explicit, Popularity, Acousticness, Danceability, Energy, Instrumentalness, Loudness and Valence are the strongest correlated features with our Playlist feature. The correlations for the mentioned features shown in 2:

| **Features** | **Corr w/ Playlist** |
|---|---|
| Artist | 0.52 |
| Explicit | 0.62 |
| Popularity | 0.65 |
| Acousticness | 0.78 |
| Danceability | 0.72 |
| Energy | 0.79 |
| Instrumentalness | 0.80 |
| Loudness | 0.81 |
| Valence | 0.62 |

Tabla 1: Correlations Features with Playlist.

### 2.2.4 Conditional Entropies

In information theory, the conditional entropy quantifies the amount of information needed to describe the outcome of a random variable Y given that the value of another random variable X is known.

This subsection computed the conditional entropy among all the features including categorical features with continuous features. This was very useful for us since it allowed to see relationships that we were unable to visualize with the typical correlation heatmap since our target variable was a categorical feature. Here the little the better since it means

| **X** | **Conditional Entropy** |
|---|---|
| Artist | 0.42 |
| Name | 0.16 |
| Album | 0.19 |
| Loudness | 0.43 |
| Duration | 0.27 |
| Tempo | 0.27 |

Tabla 2: Conditional Entropies with Y= 'Playlist' .

that knowing X we need such amount of extra information to determine Y, in this case the Playlist label. As we can see from the table if we know the Artist we need very little more information to correctly determine the Playlist which makes a lot of sense. Album is also a very common sense feature to have that relationship with the Playlist. However, Name and Duration are very weird since its impossible to determine a Playlist label only by knowing the song's duration or name. On the other hand Loudness seems ok since it is true that a song sound is a lot determined by its loudness.

### 2.2.5 Features selection

Based On the whole section and analyzing doubtful Features as it has been analysed in the notebook, the ones kept were:

- Acousticness
- Danceability
- Instrumentalness
- Loudness
- Popularity
- Artist

### 2.3 Classes Selection

This last subsection will focus in the optimal number of classes selection for the classification. The selection of this number was based in two different hypothesis:

Fist, that the average number of playlists for a user in Spotify is between 5-15 Playlists.

Secondly, we created a quick classifier model and will run three different classifiers (Naive Bayes, Random Forest, Knn). These will classify with n classes, being n the number of playlist. An iteration from 1 class to 80 classes will be executed in order to see determine the best classifier and the appropriate number of playlist to use in our model.

The results, As shown in the Notebook show that Random Forest Is the best Classifier and that the optimal value would be around 10 classes, since the Accuracy would be over 75 percent.

In order to chose the different 10 Classes we created a score function which would sum the whole set of Audio Features. The results would differentiate some of the playlist in a simple way. The chosen playlist were intentionally chosen to be uniformly distributed so it is easier for the future model.

The list was:

- BlueBallads
- PunkEspanol
- RapEspanol
- Metal
- Romanticism
- Sleep Tracks
- GoldSchool
- CountryNights

## *3 Machine Learning Models*

**This whole section will be *models.ipynb*[3]** In this section we are going to study the different Machine Learning models that we considered for our project. In addition, we will also discuss how the different data pipes that feed each model affect its performance.

### 3.1 Problem Overview

As stated in the first section, our objective here is to given a set of user-curated playlists and a song that the user just heard for the first time and he likes, we want to be able to map this new given song to one of the user-curated playlists. Basically, the model we implement has to be able to learn user's musical taste and the distribution of his songs among his personal playlists.

We thought this was an interesting and fun problem to solve since it is always a time consuming decision to choose if a song belongs to x or y playlist, so if we have this automated it would be very handy. We were inspired by Spotify's Daily Mix which gives each user a new playlist based on its musical preferences on a daily basis.

### 3.2 Possible Implementations

At first we thought of each user's playlist as a simple cluster which was constructed by set of songs and each song with 19 dimensions describing its features. Therefore, we imagined that to map each new song to a given cluster we could just calculate the distance from each song dimension to the centroid of the cluster, and the closest playlist centroid would be our chosen playlist.

On the other hand, we considered using classifiers that basing their algorithms on probability theory and different statiscs' fields are able to, after being trained on a set of example cases, given a new sample song classify it in to playlist with the most likelyhood to be in.

[3] models.ipynb available at https://colab.research.google.com/drive/1WClfrrKOKWQZGZlo$XrA$

### 3.2.1 Clustering

The two clustering algorithms that we tried out were KMeans and kNN.

- **KMeans**
  KMeans is a method of vector quantization, originally from signal processing, that aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean (cluster centers or cluster centroid), serving as a prototype of the cluster.

  We tried what was discussed in the previous section. First, we tried to apply KMeans to the whole set of songs among all user's playlists and obtain clusters by using a similarity function of our choice.

  The point of this was that after having the data clustered with the KMeans algorithm we would use the similarity function that KMeans used to cluster the data, apply it to a new given song and get the corresponding cluster. This would only work if the clusters that KMeans gave us matched the clusters or playlists the user had on his account. We implemented both unsupervised similarity functions and manual similarity functions. Sadly, no similarity function was able to correclty cluster the data and therefore it was useless.

- **kNN**
  In kNN or k Nearest Neighbours, the output is a class membership. An object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbor. Therefore it is not explicitly a clusterin method but since we are going to use it with cluster centroids in our model we include it on this section.

  We went down the road with kNN to see if we would have more luck than in the KMeans approach. Again, we encountered a lot of difficulties here since the centroids of the playlist were too close togehter to each other for them to be meaningfull. This resulted in kNN classifying even worse than at random songs to playlists centroids. It was actually expected that this happened since the common user do not usually has a wide range of genres he listens to and the audio features of the songs are very close together when you plot them all together.

### 3.2.2 Classifiers

After the disaster in the previous attempts, doing some more research in well-know data science web portals and the knowledge adquired in class we decided that the classifiers would most likely be the smart path to take. More specifically we focused on: Naive Bayes, Random Forest and Deep Neural Networks.

When using classifiers one of the biggest challenges we ecountered was how to treat categorical variables and the information gain the could provide our models with if we included them in the training data.

- **Numeric Data Points**
  Dealing with numeric data points when using classification models is the standard and should give no headaches to the data scientists.Basically, if you want to be save you standardize all your variables between 0 or 1 so the bigger numbers do not have stronger impacts and bias the model towards erroneous predictions.

  We wanted to be safe so that is what we did with the audio features for each song since they are all numeric and we did not want variables like Tempo which was in the range of the hundreds influence the model more than variables like Acousticness,

since they looked equally important for learning. After this, we can feed the data to the data pipe of the model without having to worry to much.

- **Categorical Data Points**
  Categorical data points are a whole nother story. The issues here come from the fact that most common and useful classifications models, the ones we are going to use, can only interpret data if this data is in numerical format. We will not get into further detail here but it has to do with the underlying implementation of the models.

  So how do we do in our case where we have a variable like Artist that can give us so much information about a song and the playlist it could go to, for obvious reasons, but we can not use it our model since it is made of strings and has like 6000 dimensions (1 per each artist)? Well, we used this thing called **Embeddings**.

  Embeddings is just a mapping from an input space to an output space mantaining the correlation between each member of the space. In other words, this embeddings will allow us to feed artist names to the model of our choice as vectors filled with numbers. To do so we will integer encode all the aritsts names in our datasets and then toss them trough an embedding layer that will give us the corresponding embedding for each artist name.

- **Naive Bayes**
  In statistics, Naive Bayes classifiers are a family of simple "probabilistic classifiers"based on applying Bayes' theorem with strong (naïve) independence assumptions between the features.

  These ones are the ones who gave us the worst results. This might have been due to the fact that the input dimensions were fairly big to use bayes theorem on them. Moreover, the addition of the Artist's embeddings did not seem to help the model at all since it actually reduced its accuracy.

- **Random Forest**
  Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean/average prediction (regression) of the individual trees.

  These were 'a priori' the winners, we will why we say 'a priori' later. Since the problem can be solved by a somewhat logical and step-by-step approach: if acousticness is this and artist name is that then song goes to playlist A, random forests were able to have a very good accuracy on the dataset. They were strong with the little features we selected during our data analysis and also using all of the features as RFE adviced us to do. Moreover, unlike in the case of Naive Bayes, the introduction of Artist embeddings did seem to have a positive impact on the model since its accuracy improved when we feed the artist information to it.

- **Deep Neural Networks**
  A deep neural network (DNN) is an artificial neural network (ANN) with multiple layers between the input and output layers.[12][2] There are different types of neural networks but they always consist of the same components: neurons, synapses, weights, biases, and functions.[106] These components functioning similar to the human brains and can be trained like any other ML algorithm.

  These architecture worked very well while predicting the playlists for a given song. As we said before the

playlists songs could not be easily separated in their space by for example a linear function. Therefore, deep neural networks allowed us to draw convex functions that could isolate the songs playlists correctly. This time adding the artist information through embeddings did help a lot as it did in the case of Random Forests. It was also helpfull to include all features instead of only the ones that we chose during data analysis. Their acuraccy was very close to Random Forests.

These last two clearly outboasted Naive Bayes. However, what we were referring to by saying that 'a priori' Random Forest are the best model to use is that on paper they did get the best results while predicting the classes.
However, Deep Neural Networks were pretty accurate as well and very close to Random Forests. We went for an architecture with a batch size of 16 and few epochs combined with 2 hidden layers. We most likely did not optimize the network to its fullest potential and so it could very likely outperform Random Forests if you optimize it. One approach to optimize it is to use genetic algorithms with populations formed by the parameters of the network, but we did not have the time or resources to go down that path despite how interesting it sounds.

| **Model** | **Numeric Features (RFE)** |
|---|---|
| NB | 38.4 % |
| RF | 77.88 % |
| DNN | 75.6 % |

Tabla 3: Only Numeric Features provided by RFE.

| **Model** | **Numeric Features** |
|---|---|
| NB | 38.58 % |
| RF | 61.32 % |
| DNN | 63.6 % |

Tabla 4: Only Numeric Features provided by our analysis.

| **Model** | **All Features (RFE)** |
|---|---|
| NB | 24.4 % |
| RF | 82.1 % |
| DNN | 78.9 % |

Tabla 5: Artist and Numeric Features provided by RFE.

| **Model** | **All Features** |
|---|---|
| NB | 29.2 % |
| RF | 73.3 % |
| DNN | 74.6 % |

Tabla 6: Artist and Numeric Features provided by our analysis.

## *4 Possible Practical Implementation*

One possible practical implementation of these model is to host a web server with a front end web page that allows users sign into the service we provide using Spotify's OAuth 2.0.

Once we have their permission we can store their songs and playlists in our server's database and run the model on their data so it learns their songs distributions among playlists.

Afterwards, we keep track of the new songs they like using the Spotify API and we use the model on real time to move those songs to the corresponding playlists. Aquí tenemos distintos tipos de referencias según su publicación.

## *5 Sources*


- **General Knowledge Base:** https://developers.google.com/machine-learning/clustering/similarity/supervised-similarity

https://github.com/google/eng-edu/blob/master/ml/clustering/clustering-supervised-similarity.ipynb

- **Text and Audio Analysis:**
  https://www.researchgate.net/publication/331417812$_{G}enerating_{P}laylists_{o}n_{t}he_{B}asis_{o}f_{E}motion$

- **Clustering Info:**
  https://towardsdatascience.com/k-means-clustering-using-spotify-song-features-9eb7d53d105c
  https://towardsdatascience.com/breaking-spotifys-algorithm-of-music-genre-classification-31ecf8453af1
- **MultiLabel Classification:**
  http://www.apsipa.org/proceedings/2018/pdfs/0001957.pdf
- **Autoencoders:**
  https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798
  https://distill.pub/2016/misread-tsne/
- **PCA:**
  https://towardsdatascience.com/a-one-stop-shop-for-principal-component-analysis-5582fb7e0a9c
- **Cateforical Correlation:**
  https://towardsdatascience.com/the-search-for-categorical-correlation-a1cf7f1888c9
- **Data Preparation:**
  https://www.kaggle.com/kanncaa1/feature-selection-and-data-visualization