PROJECT FINAL DELIVERABLE

PLAYLIST CLASSIFICATION[1]

Alberto Sánchez Pérez
Margherita Lovelli
Simone Melcarne

[1]**Github:** https://github.com/elalber2000/SpotifyPlaylistClassifier
Colab: https://colab.research.google.com/drive/1MQ0EWIMz59tFojg5fQvu-l8zGxN17Vtu?usp=sharing

1. Introduction

In our modern world, there is a plethora of content - especially musical content - generated every year. With this much choice, listeners can get in touch only with a small fraction of the whole existing music: this makes them unaware of potential songs and artists that they could appreciate.
As a possible solution to this problem, new technology could be used to help fix this gap, making it easier for each user to find pieces that suit their tastes and, at the same time, helping the artists to spread their music by being introduced into audiences that are already kindred to their style and sound.
To make this possible it was thought to implement a system that, given a newly released song (*the input*), is able to suggest a playlist (*the output*) whose tracks within it reflect and best approach the characteristics of the song in question. In this sense the suggested playlist can be considered as a suitable place for the specific song thus allowing its listeners to become aware of new music that fits their likings.
The task of the project was structured as a *multi-classification* problem.
This can represent a win-win idea for everyone: first of all for the artists that, independently of their level of fame, are able to reach a bigger audience; then for the music platform, which gains the possibility to increase engagement in the app; and last but not least for the users, that can easily discover new exciting music and songs.

2. Related Works

There is a wide variety of projects involving the use of machine learning with music: even Spotify has its own music recommendation systems, which use lyrical content and language, song features, and past listening habits to train deep learning algorithms (BaRT [2]). However, these models generally require a vast amount of data and computing power and have a low interpretability, problems which could be solved with the use of machine learning models.

3. Dataset and Features

3.1 General informations

The dataset was obtained from the platform *Kaggle* (Spotify Multi-Genre Playlists Data) and it was provided in *seven* different sub-datasets, one for each musical *Spotify* category: *alternative, blues, hip hop, indie, metal, pop and rock*. It was scrapped from the official Spotify API, so it has a lot of features provided by the company.
Each row of this dataset characterizes a specific track with a certain number of features (23), among which some in particular are more important and have greater weight than the others as they give fundamental information on the characteristics of a song like, for example, the *genre*, the *key,* the *danceability*, the *acousticness* etc.
For the aforementioned reason it was necessary to intervene on the dataset by doing some pre-processing in order to change and improve its structure, so that it could be easier to manage and more suitable to allow a prediction model to train on it.

3.2 Pre-processing

3.2.1 Dataset merge

The first step was to merge all the *seven* different sub-datasets, then to shuffle the songs and finally to reindex the new full dataset

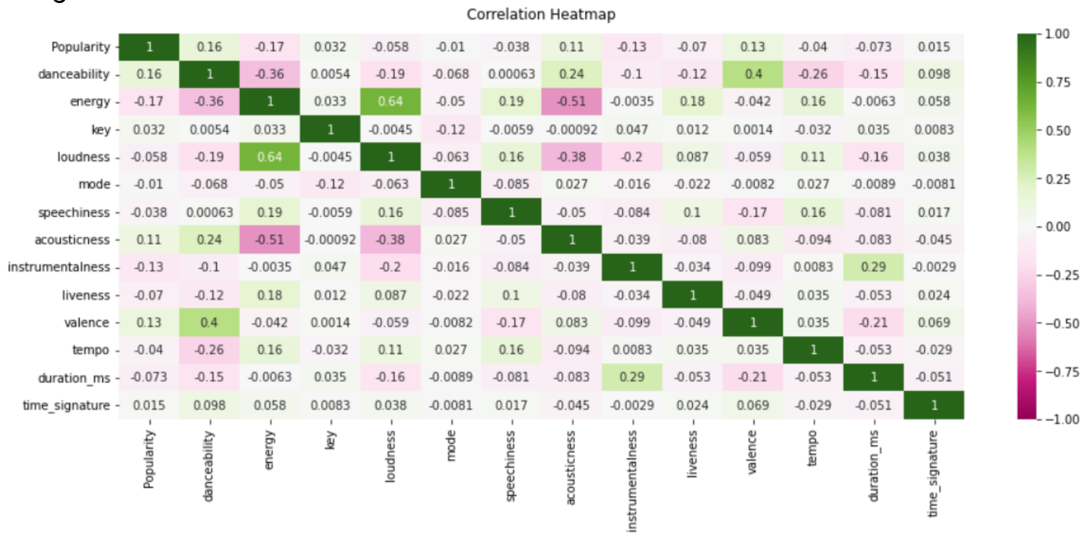
[2] Bandits for Recommendations as Treatments (or BaRT, for short) is the algorithmic system used by the music and podcast streaming company Spotify to offer personalized recommendations to its users.

3.2.2 Playlists selection

As reported above, one of the features of the dataset is the *Playlist*, which is the label to predict. In the original dataset there were a huge number of different playlists (*347*) so, to simplify the problem, it was decided to filter and keep only the top *10%* of playlists with the higher amount of songs.
Practically, a minimum threshold of songs that a playlist must have has been set equal to *99* and the total number of playlists has been reduced from to *35*.

3.2.3 Features cleaning

At this point the dataset was analyzed with a correlation matrix, which showed that the dataset was well balanced; it was thus decided to keep only the most relevant features, reducing their number from *23* to *13*.



	Popularity	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo	duration_ms	time_signature
Popularity	1	0.16	-0.17	0.032	-0.058	-0.01	-0.038	0.11	-0.13	-0.07	0.13	-0.04	-0.073	0.015
danceability	0.16	1	-0.36	0.0054	-0.19	-0.068	0.00063	0.24	-0.1	-0.12	0.4	-0.26	-0.15	0.098
energy	-0.17	-0.36	1	0.033	0.64	-0.05	0.19	-0.51	-0.0035	0.18	-0.042	0.16	-0.0063	0.058
key	0.032	0.0054	0.033	1	-0.0045	-0.12	-0.0059	-0.00092	0.047	0.012	0.0014	-0.032	0.035	0.0083
loudness	-0.058	-0.19	0.64	-0.0045	1	-0.063	0.16	-0.38	-0.2	0.087	-0.059	0.11	-0.16	0.038
mode	-0.01	-0.068	-0.05	-0.12	-0.063	1	-0.085	0.027	-0.016	-0.022	-0.0082	0.027	-0.0089	-0.0081
speechiness	-0.038	0.00063	0.19	-0.0059	0.16	-0.085	1	-0.05	-0.084	0.1	-0.17	0.16	-0.081	0.017
acousticness	0.11	0.24	-0.51	-0.00092	-0.38	0.027	-0.05	1	-0.039	-0.08	0.083	-0.094	-0.083	-0.045
instrumentalness	-0.13	-0.1	-0.0035	0.047	-0.2	-0.016	-0.084	-0.039	1	-0.034	-0.099	0.0083	0.29	-0.0029
liveness	-0.07	-0.12	0.18	0.012	0.087	-0.022	0.1	-0.08	-0.034	1	-0.049	0.035	-0.053	0.024
valence	0.13	0.4	-0.042	0.0014	-0.059	-0.0082	-0.17	0.083	-0.099	-0.049	1	0.035	-0.21	0.069
tempo	-0.04	-0.26	0.16	-0.032	0.11	0.027	0.16	-0.094	0.0083	0.035	0.035	1	-0.053	-0.029
duration_ms	-0.073	-0.15	-0.0063	0.035	-0.16	-0.0089	-0.081	-0.083	0.29	-0.053	-0.21	-0.053	1	-0.051
time_signature	0.015	0.098	0.058	0.0083	0.038	-0.0081	0.017	-0.045	-0.0029	0.024	0.069	-0.029	-0.051	1

The following attributes have been selected: *Popularity*, *Genres*, *Playlist*, *Danceability*, *Energy*, *Key*, *Loudness*, *Speechiness*, *Acousticness*, *Instrumentalness*, *Liveness*, *Valence* and *Tempo*.

3.2.4 Split

For the split specifically it has been used the `train_test_split`[3] function in order to divide the dataset into random train and test subsets (*80%* and *20%*, respectively).
The first one was used to train and evaluate with cross-validation different models whereas the second one was used for the final testing phase.

3.2.5 Genres feature

The dataset presented most of the data already clean and ready to be easily processed by the categorization models; however, the *Genres* feature turned out to be quite problematic as each element of this feature consisted of a different list of strings with no specific layout which made it a very important source of information, but also very inconsistent.
The issue was tackled in three different ways:

- **Removing the Genres feature:** this was used as a baseline to see how effective the other methods were.
- **One-Hot-Encoding:** because of the vast number of possible genres, two filters were applied to reduce their number and keep only *n* "macro genres".
  So, firstly, a part-of-speech tagging was implemented with the *NLTK library* to identify and remove the adjectives in each genre in order to go back from the child genre (i.e "*conscious hip-hop*") to the parent genre (i.e "*hip-hop*"). Then, the most repeated genres were identified and encoded into different features.

[3] from Scikit-learn

- **A language model (BERT)*:*** language models use multi-head attention models to encode vector representation of natural text. In this specific case *BERT* was used in order to encode the genres of the model; then a Principal Component Analysis (*PCA*) model was applied to reduce the number of dimensions of each vector and to avoid the overriding of other important features from the dataset.

4. Method

In order to choose the best learning algorithm, it was conducted a brief review of the state of the art[4] on tabular classification and it was decided to focus on ensemble models using decision trees, paying particular attention to *Random Forest* and *Tree Gradient Boosting* algorithms:

- **Random Forest:** the *Scikit* implementation of the Random Forest, which uses a bagging model of ensemble learning, training multiple decision trees on different subsets of the data, and then combining their predictions to make a final prediction.
- **Scikit GradientBoosting:** the *Scikit* implementation of Gradient Boosting Trees, which trains decision trees iteratively trying to reduce the error of the previous trees. The idea is that by combining many weak predictors, it is possible to create a strong overall predictor that is able to accurately model the data.
- **XGBoost:** an implementation of gradient boosting which applies several regularization techniques (such as shrinkage) to help prevent overfitting.
- **CatBoost:** an implementation of gradient boosting which uses the symmetric tree algorithm to make predictions. It can also handle categorical features and missing values.

5. Experiments/Results/Discussion

5.1 Evaluation

Cross validation was the technique used together with a *RandomForestClassifier*[5] to evaluate the three different dataset refinement approaches mentioned above and identify which one gave better results.
To achieve this goal, the function `cross_val_score` from *Scikit-learn* was used, which also provides two metrics to evaluate the goodness of the prediction model: the `f1 score` and the `accuracy`. The number of folds `cv` was set to 5.

	No Genre	One-Hot Encoding	Language Model (BERT)
Accuracy	0.6151	0.6314	**0.7381**
F1 score	0.5673	0.5755	**0.7128**

In the end, the vector representation of the *BERT* model was the one that gave the best values.

[4] «Algorithms based on gradient-boosted tree ensembles still mostly outperform deep learning models on supervised learning tasks, suggesting that the research progress on competitive deep learning models for tabular data is stagnating.» - Deep Neural Networks and Tabular Data: A Survey

[5] The number of estimators for the *Random Forest* model was set to *100*.

5.2 Model
After choosing the best way to encode the *Genres* feature, some experiments were made to identify the best learning algorithm. As for the previous step the `cross_val_score` function was the one used to get the final result.

	Random Forest	XGBoost	CatBoost	Scikit Gradient Boosting
Accuracy	**0.7291**	0.6449	0.6523	0.4219
F1 score	**0.7064**	0.6347	0.6378	0.4300

The Random Forest model turned out to be the one with the best performance scores.

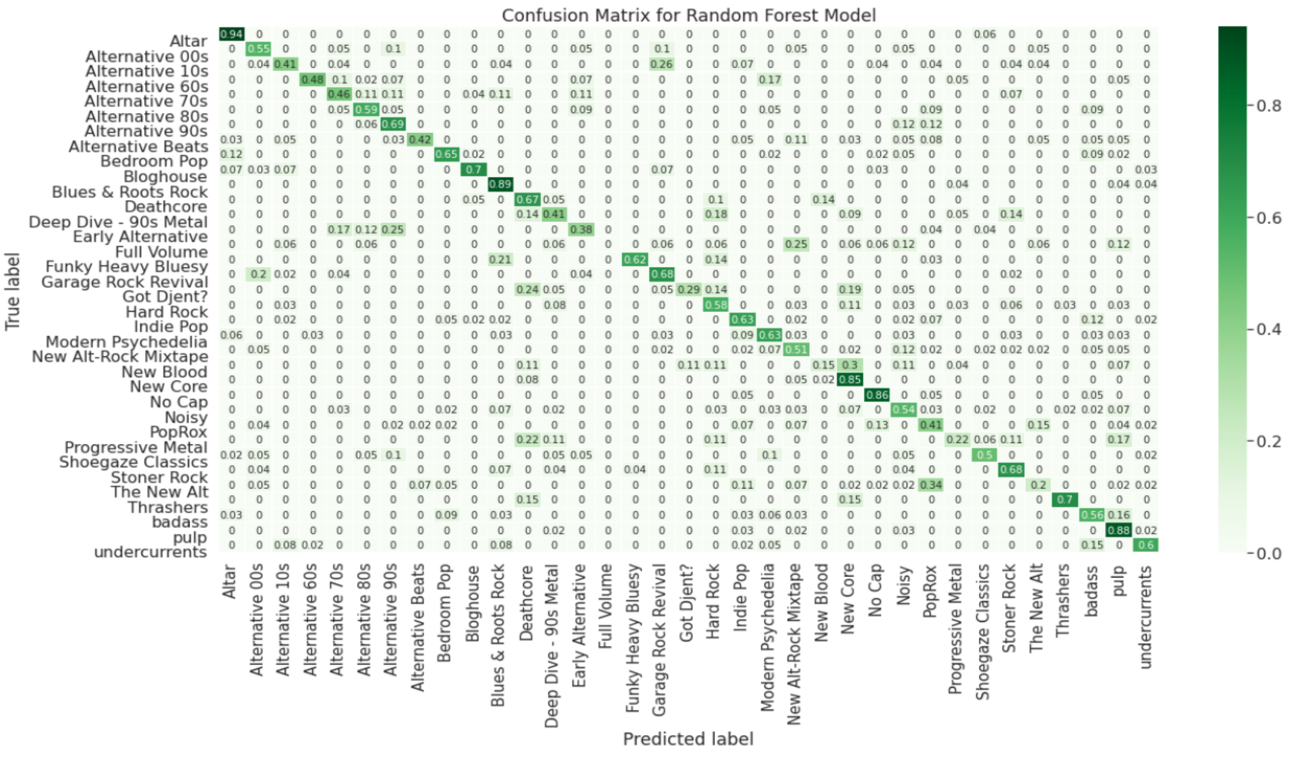
5.3 Hyperparameters
Once the best prediction model was chosen, the *GridSearch* technique was implemented in order to compute the optimum values of the most important hyperparameters of the Random Forest model.
The highest accuracy score (*0.7306*) was obtained with the following parameters values:
- `max_depth: None`
- `min_samples_leaf: 2`
- `min_samples_split: 5`
- `n_estimators: 100`

5.4 Final Testing
After all the evaluation, the final model was built based on the following scheme:
- *Genres* feature encoded with *BERT*
- *RandomForest* classifier with the parameters tuned as mentioned in section *5.3*

At this point, the test split was used to see the *real* performance of the model and the following results were achieved:

	Precision	Recall	F1 score	Support
Accuracy			0.57	1143
Macro avg	0.56	0.55	0.54	1143
Weighted avg	0.59	0.57	0.56	1143

From the results obtained it can be observed that during the test phase the results are slightly worse than those obtained during the training; this could suggest that there may be some overfitting.
A confusion matrix was used in order to have a more in-depth overview of how the model behaves and to visually identify any misclassifications.
It is possible to notice that there is in general a predictable normal trend with more or less negligible misclassification values; the only main exception is represented by the "Full volume" playlist which has 0 as value on the diagonal. Since the diagonal elements of the matrix represent the number of correctly classified samples for each class, a 0 there means that no correctly classified samples were present for that particular class. This could indicate that the model is not performing well for that class or that the songs of that specific playlist are hard to identify.



6. Conclusion/Future Work

To summarize, the best performing algorithm was the *Random Forest* with the *Genres* feature encoded using language models. Every model performs differently depending on the dataset, but there are some factors that can be considered on why *Random Forest* outperformed *Gradient Boosting* models:

- The model is generally more robust against overfitting.
- *Gradient boosting* models are more sensible to hyperparameters. A previous hyperparameter grid search before the model comparison could probably make the difference; with more resources it could be a possible ampliation of the problem.
- *Gradient boosting* is generally more sensible to the data, so it may have performed worse because of the nature of the dataset.

In the end the results obtained are not so great (barely below *0.6*) but, given the fact that we are dealing with multi-classification with a lot of categories and in which there can be two possible good answers, they are acceptable.

For future research it could be a good idea to explore more ways of preprocessing the data, especially regarding the *Genres* feature; moreover it could be interesting to experiment with a wider range of models - maybe even deep learning ones - and to use different and looser evaluation metrics, as similar playlists may admit the same song.

7. Contributions

The *pre-processing* was done by all the group together. For the task of dealing with the *Genres* feature, Margherita Lovelli and Simone Melcarne focused on the *One-Hot Encoding* approach while Alberto Sánchez focused on the *BERT* and *PCA* approach.

The *model comparison* and *hyperparameter tuning* was done by all the group together.