Data Mining II

Chiara Buongiovanni
c.buongiovanni@studenti.unipi.it
507164

Andrea Mattei
a.mattei3@studenti.unipi.it
546139

Academic Year 2020/2021

Contents

Module 1

1 Introduction, Data Understanding and Preparation

The Free Music Archive (FMA) is a large audio datasets that provides 917 GiB and 343 days of Creative Commons-licensed full-length and high-quality audio from 106,574 tracks, coming from 14,854 albums by 16,341 artists and arranged in a hierarchical taxonomy of 161 genres, together with pre-computed features, track- and user-level metadata, tags, and free-form text such as biographies [1].

1.1 Data Semantics

All metadata and features about the tracks are distributed in four different tables:

- `tracks.csv`: per-track, per-album and per-artists metadata, divided into 3 dedicated partitions (*album*, *track* and *artist*);
- `genres.csv`: names of all 163 genres, paired with their parent in the hierarchy;
- `features.csv`: common features extracted with `librosa`, which translated the spectrograms of each song into continuous values;
- `echonest.csv`: audio features provided by Echo Nest (now Spotify) for a subset of 13,129 tracks.

Within the *tracks* table, each row represents a track and columns represent features that provide a rich array of information: song and album title, artist name, and per-track genres; user data such as per-track/album/artist favorites, play counts, and comments; free-form text such as per-track/album/artist tags, album description and artist biography or his website url, categorical information such as the type of album (`album_type`) the track is taken from (*Album*, *Contest*, *Live Performance*, *Radio Program* or *Single Tracks*), or the type of license under which it was released, up to the amount of data that is stored in the sound file (`bit_rate`).
There are also various temporal variables concerning track recording date, album release date, the artist's period of activity (`active_year_begin` column and `active_year_end`) and per-track, per-album and per-artist date of insertion in the database.
Three columns provide information on the artist's geographical location: `latitude` and `longitude` are decimal numerical variables, while the `location` column contains information about the address, city, region or state.
The `genres` column contains the IDs corresponding to the genre labels indicated by the artist, while the `genres_all` column was constructed using the hierarchical information given in the *genres* table and contains all the genres encountered when traversing the hierarchy from the indicated genres to the roots. The root genres are stored in the `genres_top` column.

The *echonest* table provides various pieces of information for a subset of 13k songs. These are features measured and provided by the Echo Nest project. In particular:

- the *metadata* partition provided general information about the artist (name, geographic coordinates) and the album (title, release date)
- the *audio_features* partition contains 8 different variables that provide low-level audio analysis:
  - `acousticness` measures how much a track turns out to be acoustic based on the electronic or acoustic means used to produce it;
  - `danceability` describes how suitable a track is for dancing,

based on musical elements such as overall regularity, beat strength, tempo and rhythm stability;
  - `energy` represents the measurement of activity and intensity of a song;
  - `instrumentalness` is a measure of the quantity of vocals (spoken words and rap) contained in a track;
  - `liveness` detects whether a song has been recorded in a setting where there is an audience present;
  - `speechiness` detects the presence of spoken words. The higher the value is, the more speech there is exclusively, e.g. audio books, podcasts or talk shows;
  - `tempo` represents the overall beats per minute (BPM) for a song;
  - `valence` measures the positivity conveyed by a song;
- the *social_features* partition collects 5 attributes related to the popularity and social feeds of songs (`song_currency`, `song_hotttnesss`) and artists (`artist_discovery`, `artist_familiarity`, `artist_hotttnesss`). In the *ranks* partition, there are as many attributes which report the position of the artist or song based on each social feature in the rankings generated by the Echo Nest algorithms[1];
- other unspecified *temporal features* with incremental column names from 000 to 223.

Since our study aims to focus on genre recognition using the *audio features* provided by Echo nest, it is necessary to reduce the size of the original dataset to the 13129 tracks for which this information was available. Below, we carry out the analysis of the resulting dataset by discussing the distributions of its most interesting features, we investigate its quality and describe the data preparation activities carried out on it.

[1] The Echo Nest blog

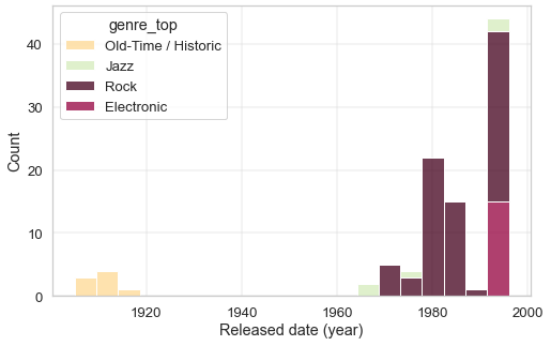

Figure 1: Album released from 1902 to 1996. The older songs belong to 4 genres in particular: Rock and Electronic in the nineties, Jazz and, unsurprisingly, Old-Time / Historic.

1.2 Exploratory data analysis

The tracks in our dataset were written by 2876 different artists and they come from 2472 different albums, released over the last two centuries (Figure 1), and collected in the dataset from 2008 to 2015.

To have an overview of the number of tracks available for each main genre (`top_genre`), we propose the graphs in Figure 2. The graph on the right immediately shows how the distribution of the different genres is not well balanced: the Rock (42% of the entire dataset) and Electronic (23%) genres are by far the most represented, while the percentage distributions of the other 10 genres do not exceed 10% (Figure 2, left).

In an attempt to identify characteristic features for different genres, we studied the average values of each audio feature, calculated on tracks belonging to the different genres (Figure 2, right). Each of them has a range of values from 0 to 1, with the exception of `tempo`, which has been appropriately normalized in order to be compara-

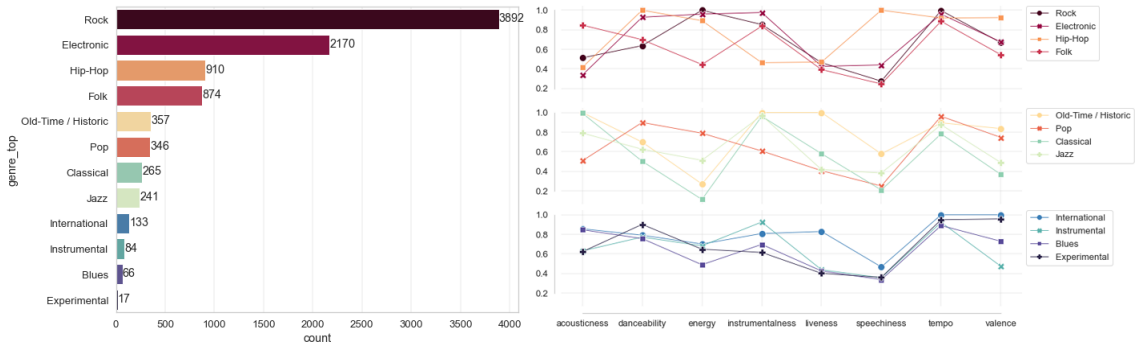

Figure 2: Left panel: percentage of songs by main genre (`top_genre`). Right panel: average value of normalized audio features by genre.

ble with the others. The graph in the Figure shows differently discriminating variables: while the time variable has on average high values for each genre, and as such does not significantly represent any of them, the `acousticness` variable records generally very high values for genres such as Folk, Classical and Old-Time, in which the electronic component of the sound is minimal, as opposed to what happens in genres such as Pop, Electronic, Hip-Hop and Rock, which have an `acousticness` value close to 0.5. Another example of how these variables actually seem able to discriminate one genre from another is also offered by the `speechiness` variable, which sees low average values for all genres except for the Hip-Hop genre, which includes the Rap subgenre.

There are several variables that present a very asymmetric distribution. The calculation of the *skewness* of the `comments`, `favorites` and `listens` attributes returned very high values (on average: 7.6 for the calculation made on favorites, listens and comments of albums, 17.2 for the comments and favorites of artists, and 39 for the respective attributes of the tracks), indicating a strong asymmetry with a larger tail towards the right hand side of the distribution. A percentage study of the distribution of these attributes, in fact, confirmed that

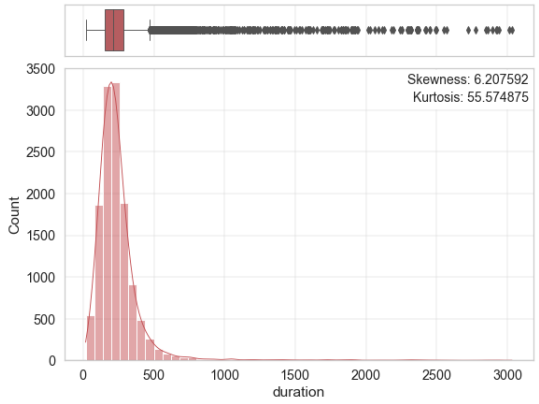

Figure 3: Distribution of `duration` attribute and its respective kernel density estimate line. Top panel: boxplot of the attribute.

most of the tracks, albums and artists in our dataset have a very low, often nil, number of comments or favorites: about 84% of albums, 80% of the artists and 97% of the tracks have no comments, while about 98% of the albums, 84% of the artists and 91% of the tracks have fewer than 10 favorites. The situation is different with regard to the listens variable, for which the high value of *skewness* (31.3) is to be justified by the presence of outliers (*kurtosis*: 1433.5, highest value: 543'252).

The last attribute of which we want to show the distribution is `duration` (Figure 3): also in this case the distribution asym-

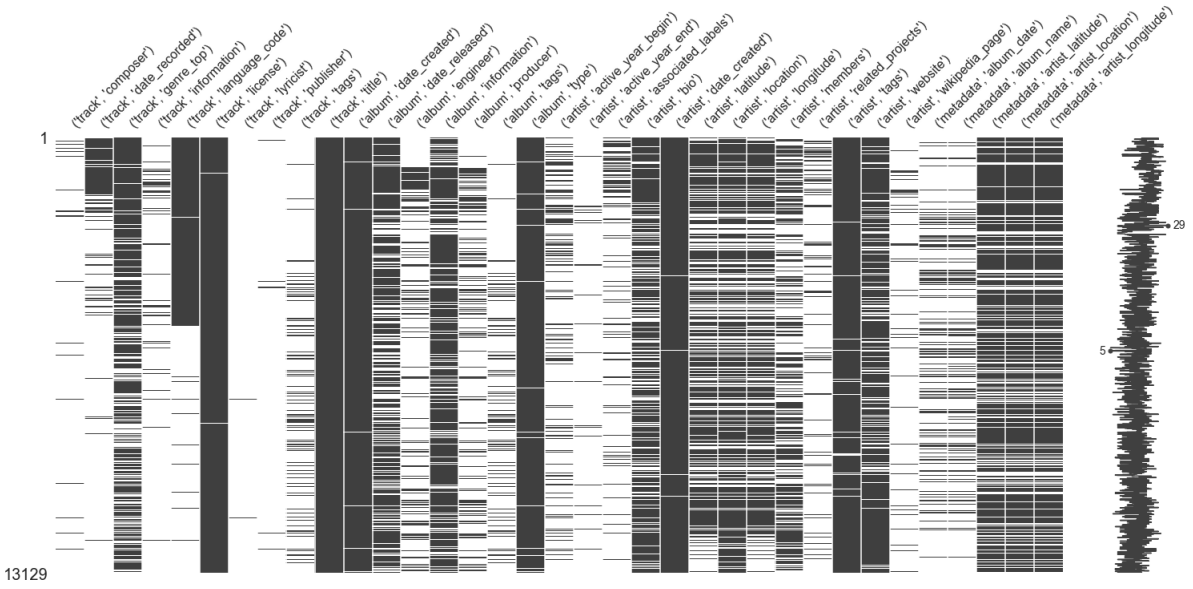

Figure 4: Visual summary of the completeness and lack of the dataset provided by `missingno` Library.

metry is very marked due to a high number of outliers (see boxplot in Figure 3).

1.3 Data Quality

In this section, we analyze the results of the data quality check and the operations of data cleansing. Specifically, the data of the various tables provided were collected in a single consolidated dataframe, within which only the reasonably populated fields of interest were selected. Based on the purposes of the analysis and the relationship between the data, they have been aggregated, reduced in dimensionality, discretized and transformed. Finally, on the dataframe obtained, operations were carried out to fill or delete the records that had missing and/or anomalous values.

1.3.1 Feature Selection

Of the 52 features of the *tracks* table, only 21 have been selected: plain text attributes have been excluded, such as the biography or artist name, the description and the title of the album or track, as well as those referring to composers, lyricists, publishers, and other figures of this kind, for which most of the records did not present any information (Figure 4).

In general, the columns that were not taken into account presented information that was not relevant to our analysis or was available for less than 50% of the data. An exception was made for the `wikipedia_page` attribute which, although it presented about 91% of missing values, we considered informative about the degree of notoriety of the artist. As for the features `tags` referring to albums, artists and tracks, we decided not to consider them because our analysis (see Figure 5) highlighted an inconsistent and sometimes redundant supply of information, for example in the case of the related tags, the name of the artist or the musical genre, already specified in the `top_genre` and `genres` columns.

In order to make the identification of the attributes more immediate, the homonyms have been renamed with the name of the feature followed by the name of the parti-

tion to which they belonged (e.g.: the variable `comments` of the *track* partition has been renamed `comments_track`).

From the `echonest` table, we selected all the audio and social features, while the columns from the *metadata* partition have been aggregated with columns containing the same type of information in order to minimize the presence of missing values.

1.3.2 Missing Values

Once the features of interest were selected, missing values had to be handled.

We used the entire dataset of 106,574 records to calculate central tendency measures of column data such as mean, median and mode, which are then used for replacing missing values appropriately.

In several variables, we found the use of the value -1 for missing values. In the numeric variable `bit_rate`, we replaced -1 values with the mode of the attribute; in `comments_album`, `comments_artist`, `favorites_album` and `favorites_artist`, with the respective median. The missing value of the `listen_album` attribute has been filled with the sum of the plays of the tracks it contains provided by `listen_track`. Similarly, the number of tracks per album (`n_tracks`), where absent, was obtained by adding the tracks present in the data set belonging to that album.

The missing values of `date_released` (29.2%) were first filled with the respective values of the `album_date` attribute of the *metadata* partition in the *echonest* table. The remaining null values were obtained by calculating the mode of the release dates of the artist's other albums or, when not available, the mode of the entire attribute.

1.3.3 Variables Transformation

In order to make the data as consistent as possible for the next modeling phase, we

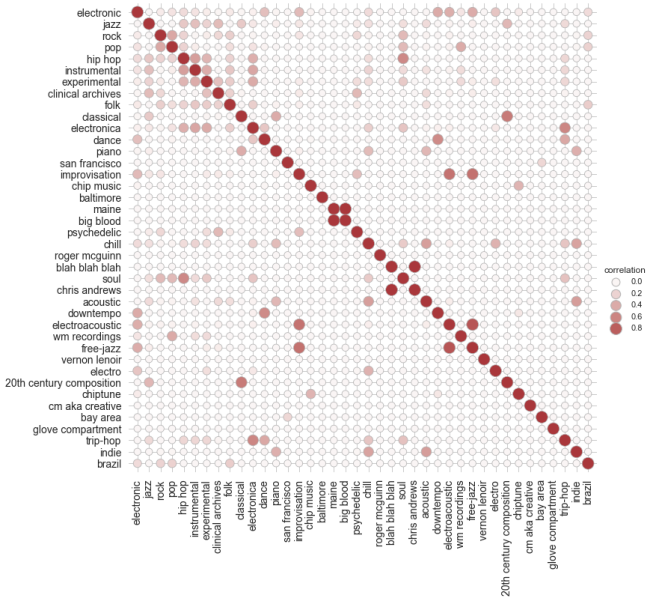

Figure 5: Study of the degree of association of the 40 most frequent tags (about artists), out of a total of 3555 different tags, obtained using the Pearson correlation coefficient.

have transformed some variables. In particular, we have chosen to make the website and `wikipedia_page` attributes binary, replacing the urls present with the value 1 and the missing values with 0. The result of this operation has shown that only 5.7% of the artists have a Wikipedia page dedicated to him, while 56.2% have a website.

To make geographical information understandable by human and machine alike, we decide to use the coordinates provided by the `latitude` and `longitude` attributes of the *tracks* table and, where absent, by the `artist_latitude` and `artist_longitude` attributes of the *echonest* table, to generate a categorical variable with the names of the countries, using the library `geopy`. With this approach, only 11% of the records did not obtain a precise geographical location, and were therefore classified as *Unknown*. Figure 6 shows a geographical distribution of our data set, the tracks of which come

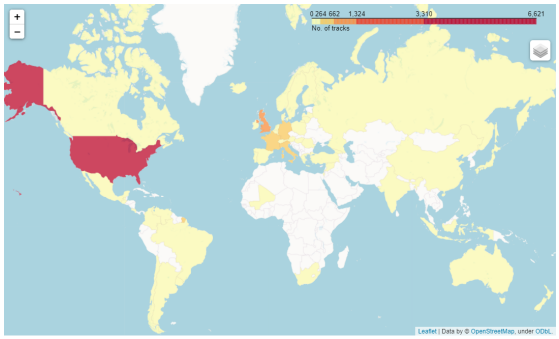

Figure 6: Geographical origin of the tracks.

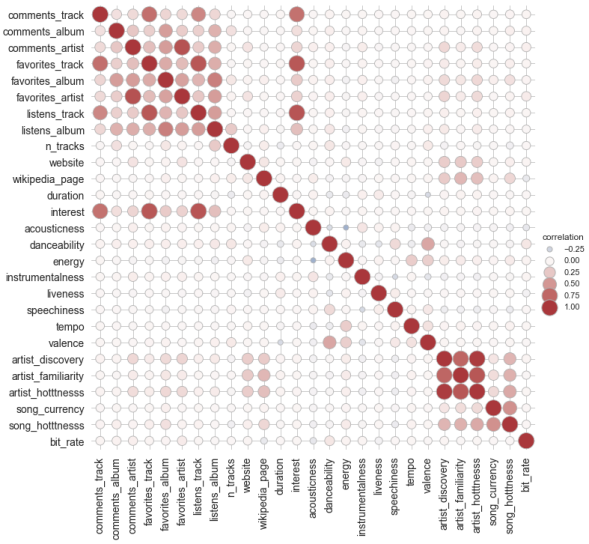

Figure 7: Person correlation heatmap of data.

from at least 53 different countries.

The analysis of dependencies and correlations between attributes was carried out through the analysis and visualization of the correlation matrix (Figure 7). This operation was useful in identifying redundant information, suggesting possible aggregations of variables.

For example, the matrix clearly highlights the strong correlation between the `interest` attribute (continuous numeric variable) and `comments`, `listens` and `favorites` tracks attributes. For this reason, as well as for the absence of specific information on its creation, we have chosen not to consider it in our data set.

We can also observe that the independent variable `artist_discovery`, `artist_familiarity`, `artist_hotttnesss` are highly correlated (person coefficient $>$ 0.9) with each other. Our idea was to aggregate these attributes using the Principal Component Analysis (PCA), obtaining a new features, called `artist_pop`, scaled in range [0, 1] using `MinMaxScaler`.

Lastly, we applied some transformations to the columns pertaining the musical genres of the tracks, which will be an important information for our analysis. Since most of the songs in the dataset are attributed more than one genre label, even when excluding sub-genres and considering only the ones higher in hierarchy (so-called `genre_top`), we wished to represent this information in a more clear-cut way. To do so, we discarded any info pertaining sub-genres and kept only 16 macro-genres: *Experimental*, *Electronic*, *Rock*, *Pop*, *Instrumental*, *Folk*, *Hip-Hop*, *International*, *Jazz*, *Classical*, *Country*, *Spoken*, *Blues*, *Soul-RnB*, *Old-Time/Historic* and *Easy Listening*. We then applied One-Hot Encoding to obtain a new binary variable for each of the selected genres, where a value of 1 means that this top-genre was presented in the label list for that track, 0 otherwise. In order to satisfy the possible need for an easy global outlook on this matter, the old `genre_top` column was then modified to become a categorical attribute, with the name of a genre as value if that row has only a 1 in the one-hot encoded columns, 'Ambiguous' otherwise. Figure 8 shows an example of how this information is encoded in our modified dataset, showing only some columns for better visualization.

1.3.4 Anomaly Detection

In this section, we describe three different outliers detection algorithms implemented

track_id	Experimental	Electronic	Rock	Instrumental	Folk	genre_top
41838	1	0	0	1	0	Ambiguous
12189	0	1	0	0	0	Electronic
1603	0	0	1	0	0	Rock
47921	0	0	1	0	0	Rock
15308	0	0	1	0	0	Rock

Figure 8: Example of genre encoding

to identify the top 1% outliers: the Isolation Forest algorithm, the Local Outlier Factor (LOF) algorithm and the Angle-Based Outlier Detection (ABOD) algorithm.

Isolation Forest The first outlier detection algorithm we employ in our analysis is a model-based one: Isolation Forest, in its Python implementation found in the `sklearn.ensemble` library. Taking advantage of the strong resistance to the curse of dimensionality this approach provides, we train the model on all the numerical and binary features present in the dataset, in addition the categorical variables `location`, `album_type`, `license` and `genre_top`, with their textual values encoded to numerical ones. We rely on a 3-dimensional version of the dataset obtained applying Principal Component Analysis to get a sense of how the algorithm is working. The hyperparameter *contamination* is set to 0.01 in order to identify the required number of top outliers. In figure 9 is shown the result obtained leaving the parameter `max_samples`, the number of samples to draw from the data to train each base estimator, as the default value of 256.

Increasing `max_samples` modifies the results slightly, but does not produce a visible improvement in quality: most changes consist in swapping labels for adjacent points in regions where both outliers and inliers are identified, whereas points that our visualization suggests being undetected outliers remain undetected.

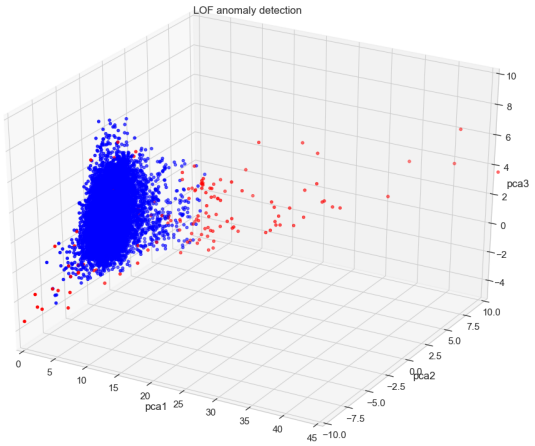

Figure 9: Isolation Forest, max samples = 256. Outliers are highlighted in red.

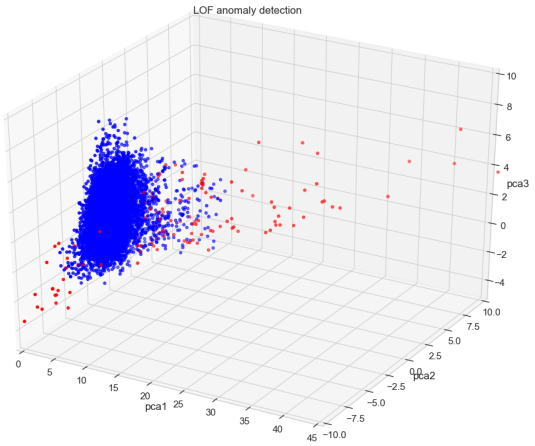

Figure 10: Local Outlier Factor, n_neighbors = 150.

Local Outlier Factor The second implemented outliers detection algorithm belongs to the family of density-based algorithms, LOF, implemented with from `sklearn.neighbors` library. Since the density becomes less meaningful in high-dimensional space, the PCA operator is used to reduce the high-dimensionality of our standardized dataset to 3-dimensions. In any case, we tested its performance also on fully-dimensional dataset in order to

identify how dimensionality affects the algorithm. Again, the hyper-parameter "contamination" is set to 0.01, while the number of neighbors to use for kneighbors queries is set to 150. The results obtained (Figure 10) with both methods were generally satisfactory as both correctly selected the outliers with the highest score (*negative_outlier_factor_*). In general, the two approaches gave the same result for about 50% of the total outliers identified. By comparing these values with those obtained by the IF algorithm, together with a visual check of the different pairplots of the entire dataset, we used the anomaly detection performed on the non-resized dataset.

Angle-Based Outlier Detection The third and last outlier detection algorithm applied to our dataset is an angle-based one, ABOD, with an implementation found in the library `pyod`. Results are shown in figure 11: although recommended for high dimensional spaces, our 3D representation seems to suggest that this algorithm is not particularly suitable for data at hand, since it fails to label as outliers most of the points that lie isolated in space (e.g., in the right part of the graph) and which, however, have been correctly classified by the other two algorithms implemented.

Conclusion Comparing the results obtained with the approaches tested, we find out that:

- 20 points are detected as anomalies by all three algorithms;
- 49 points are detected by two algorithm but not by the third one: in particular, ABOD and LOF agree upon 16 points, ABOD and Isolation Forest upon 19, and Isolation Forest and LOF upon 14;
- 238 are classified as outliers by only one algorithm: 77 by ABOD, 79 by Isolation Forest, 82 by LOF.

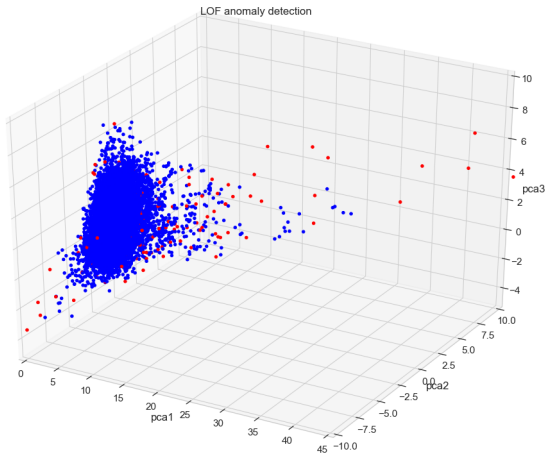

Figure 11: ABOD. Outliers are highlighted in red.

Without having a ground truth to evaluate the results, we devise a simple procedure to help us choose which algorithm to deem more trustworthy for the actual removal of outliers from the dataset. Following a boxplot-like approach, we classify the values of each numerical attribute as "local outliers" if they are higher than the third quartile plus 1.5 times the interquartile range, or lower than the first quartile minus the same amount. For each record, we then count the number of local outliers it contains and order the data by this number in a decreasing way. In figure 12 we show how many rows present a certain number of local anomalies.

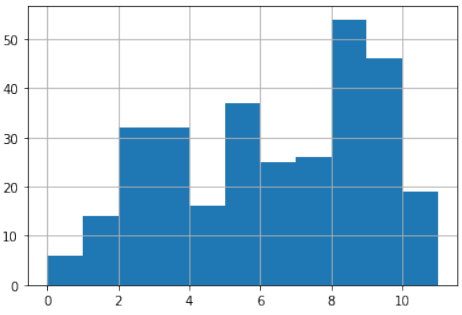

Figure 12: Distribution of local outliers

Checking the performance of the three algorithms on the rows with a high number of local outliers, we find out that Isolation Forest identifies most of the top records, whereas LOF fails at detecting some of the rows with 10 anomalies, while ABOD has trouble with nearly all of them. For the purpose of this analysis we therefore decide to carry on deleting the top-1% outliers following Isolation Forest.

2 Basic Classification Tasks: Genres recognition

For the tasks about classification we decide to focus on an analysis in the field of genre recognition. After transforming the *genre_top* variable as described in section 2.3.2, we try to predict if each song pertains to a certain musical genre or not. For this first exploratory task, which will be carried out with both Decision Trees and KNN, we try to identify songs which are labeled with a certain genre tag among all others that don't. We expect this kind of task to be a bit more difficult than if we were to select only tracks belonging to two genres, so we perform the classification twice, on two different genres, in order to better verify the feasibility of this approach on our data. The chosen genres are *Rock* and *Electronic*, the most frequent ones. In order to make the two classifications more comparable, we split the data in train and test set (70-30 proportion) stratifying both times on the categorical feature *genre_top*, not on *Rock* or *Electronic* (the binary ones we use for assigning the target classes). This should make the task a bit more challenging, too, since *genre_top* contains a large number of *Ambiguous* values, i.e. songs which have been labeled with more than one macro-genre.

2.1 Decision Tree

The features passed to our decision trees are those described in the variable selection section, with the categorical ones appropriately encoded to numerical values beforehand.

Table 1 shows the results on the test set obtained by classifying on the two chosen genres, both before and after a 5-fold cross validation performed with the F1-score as evaluation metric. We refrain from utilizing the accuracy even if our classes do not present that high of an imbalance, since our problem is set up in such a way that we care more about the performances on the positive class.

Acceptable values of precision and recall are reached even by the default unbounded decision tree, but the pre-pruning conditions discovered in the parameter tuning phase do indeed help in improving the classification quality (+0.01 F1-score and accuracy for both genres). The table also reports the results obtained by classifying utilizing the full length of the available dataset, passing to the algorithm the features for which enough sample have not-null values (`location`, number of comments, favorites, and listens for the track and its album, presence of website and/or wikipedia page for the author, `duration`, `interest`, `license`, `bit_rate`, `album_type`): as it can be seen, these classifiers are worse than the ones obtained employing nearly 10% of the data but with the possibility of using features which are clearly more meaningful for a classification task.

Focusing on our the results obtained after tuning the parameters, figure 13 shows the feature importance for each of the two decision trees we have built: we can see that for Rock songs (in blue) the most discriminating attributes are `energy`, `danceability`, `speechiness` and, perhaps more surprisingly, the type of album from which the track comes from and

	Rock			Non-Rock				
	Precision	Recall	F1	Precision	Recall	F1-Score	ROC-AUC	Acc.
Base Params	0.73	0.72	0.72	0.80	0.81	0.80	0.795	0.77
**Cross Val. F1	0.72	0.75	0.74	0.82	0.80	0.81	0.812	0.78
'Big' Dataset	0.37	0.32	0.34	0.72	0.77	0.74	0.544	0.63
	Electronic			Non-Electronic				
	Precision	Recall	F1	Precision	Recall	F1-Score	ROC-AUC	Acc.
Base Params	0.68	0.66	0.67	0.86	0.87	0.86	0.798	0.81
**Cross Val F1	0.72	0.64	0.67	0.85	0.89	0.87	0.842	0.82
'Big' Dataset	0.42	0.41	0.41	0.72	0.73	0.72	0.578	0.63

Table 1: Results for our exploratory genre recognition task performed with Decision Tree.

the total listens gathered by such album; for Electronic songs the most useful features for classification are `danceability`, `acousticness`, `instrumentalness` and the `location` where the track has been produced.

Overall, we can deem the results of this exploratory classification task satisfactory. This approach could be repeated for each of the other 14 macro-genres present in the dataset, remembering to apply some precautions to address the class imbalance for less represented genres, as will be shown in section 3.3. In section 4 we will instead focus on taking advantage of more sophisticated classification method to obtain better performances for the classification of Rock songs.

2.2 K-Nearest Neighbors (KNN)

Another type of supervised machine learning algorithm employed for our classification task is K-Nearest Neighbors (KNN). Since KNN is sensitive to magnitudes of features, before making any actual predictions, both training set and test set have been scaled with the `StandardScaler` method to be uniformly evaluated. For this study, we decided to exclusively select *audio features*, to test their ability to classify one genre over the others within our dataset, as the graph

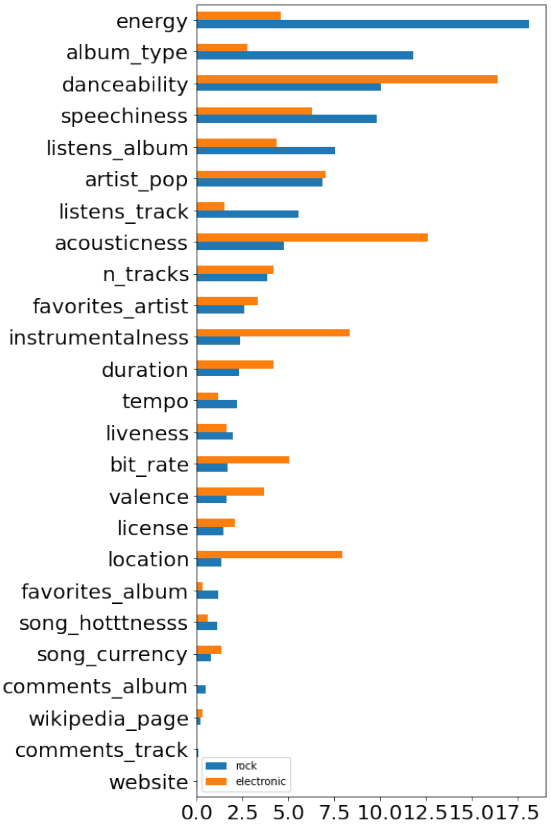

Figure 13: Feature importance in the best decision tree for our simple classification

in Figure 2 seems to suggest. In order to optimize the hyper-parameters of the model and figure out the optimal values to get the best results, we use `GridSearchCV`, specify-

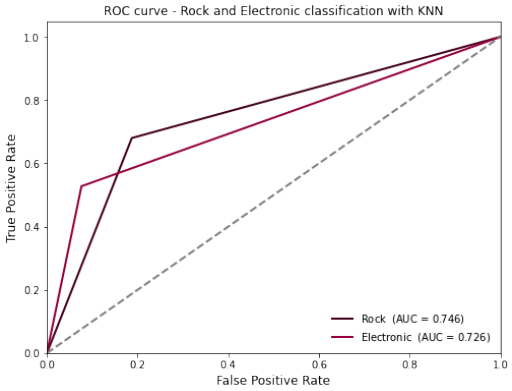

Figure 14: ROC curves for KNN basic classification tasks.

ing the number of folds at 3 and the F1-score as evaluation metric. For this model, we test:

- weights: *distance*, *uniform*;
- n. neighbors: from 20 to the square root of the number of records, with an increment of 10;
- metric: *euclidean*, *manhattan*.

The five best parameter configurations obtained for the classification of the *Rock* tracks use in all cases the *Manhattan* metric and *distance* as *weights* parameter. The number of neighbors considered, on the other hand, varies from 20 to 60. The value that maximizes the f1 score is 30. For the classification of the tracks belonging to the *Electronic* genre, the `GridSearchCV` again suggests to use the *Manhattan* metric and *distance* as *weights* parameter, while the suggested number of neighbors is 20. The results on test set for these classification tasks using KNN are reported in table 2.

The satisfactory F1-score for both classifications confirm the central role of *audio features*, capable of discriminating between genres, as was already evident from the study of the important features for Decision Tree.

2.3 Imbalanced Learning

For the task about imbalanced learning we remain in the field of genre recognition, defining a problem similar to the ones analyzed in the previous subsection: this time, however, the target variable chosen is the *Classical* genre, which appears in 437 tracks (roughly 3% of the total data). Table 3 show the performances of the decision trees trained employing various methods addressing the class imbalance, which will be discussed in the next subsections. Precision and Recall on the negative class are omitted for ease of visualization, since as expected their values are always extremely high. The parameters for pre-pruning each tree are obtained again through a 5-fold cross validation using F1-score as the evaluation metric.

First of all, we learn a decision tree on the dataset as is, to serve as a baseline thanks to which we will be able to compare the performances of the various balancing methods taken into consideration. This first model (**(1)** in table 3) shows quite high precision and recall scores on the positive class already, a sign that our target genre may be very well separated from the others. Looking at the feature importance, the most discriminating ones are *listens_track* (24%) and *energy* (17%): a glance at the first levels of the tree (shown in figure 15 reveals that Classical songs tend to have a high number of listens among the songs with lower energy.

In order to address the class imbalance, we first apply two undersampling and two oversampling methods through the implementations found in the library `imblearn`. A 2-dimensional version of our dataset created by applying PCA to the features used for classification is shown each time to highlight how these procedures modify the data distribution: Classical tracks are colored in blue, while songs belonging to other genres in orange. Figure 16 shows the starting class distribution.

	Rock			Non-Rock			
	Precision	Recall	F1	Precision	Recall	F1-Score	Acc.
**Cross Val. F1	0.72	0.68	0.70	0.78	0.81	0.80	0.76
	Electronic			Non-Electronic			
	Precision	Recall	F1	Precision	Recall	F1-Score	Acc.
**Cross Val F1	0.74	0.53	0.62	0.82	0.92	0.87	0.81

Table 2: Results for our exploratory genre recognition task performed with KNN

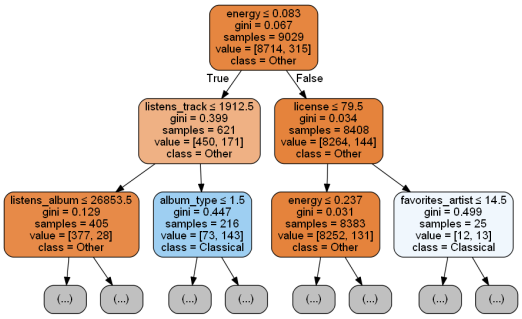

Figure 15: First 2 levels of the baseline tree for imbalanced learning.

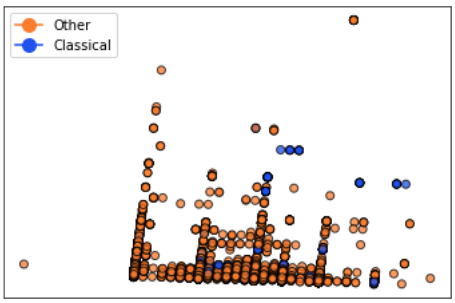

Figure 16: 2D version of the data used for imbalanced learning

2.3.1 Undersampling

Figures 17 and 18 show two undersampled versions of our dataset, obtained removing tracks not belonging to the Classical genre until the size of the two classes becomes equal (315 data points each). Looking at the decision trees results, we can see that removing points at random (row (2) in table 3) generates slightly better performances than applying the Condensed Nearest Neighbour (CNN) algorithm. Both methods, however, manage to improve the Recall score on the positive class with respect to the one obtained with our base dataset, but do so at the cost of dropping the Precision and consequently the F1-score significantly. This kind of estimation skewed towards predicting a lot more members of the positive class should be expected from an approach of this kind; it should also be noted that when dealing of dataset of this size, not particularly large, undersampling causes a not insignificant information loss.

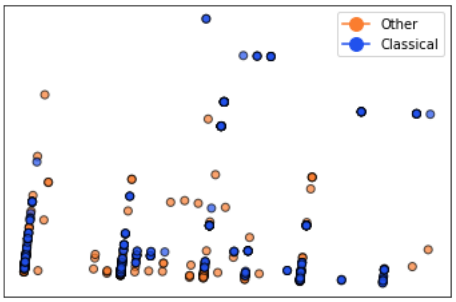

Figure 17: Random Undersampling

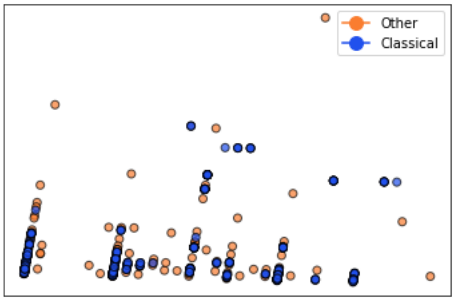

Figure 18: CNN Undersampling

2.3.2 Oversampling

Figures 19 and 20 show two oversampled versions of our dataset, obtained increasing the number of members of the positive class until the sizes of the two classes match, by copying existing ones (random selection, row (4) in the table) or by creating new points interpolating between them (SMOTE, row (5) in the table). Both methods achieve an improvement with respect to the recall for the positive class, however still at the cost of decreasing the precision, even if by a lesser amount compared to undersampling.

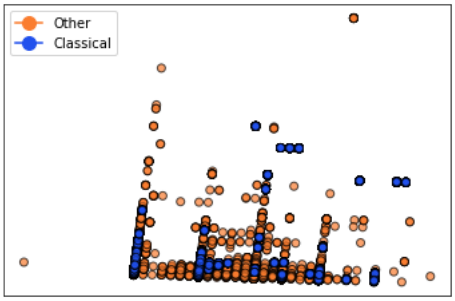

Figure 19: Random Oversampling

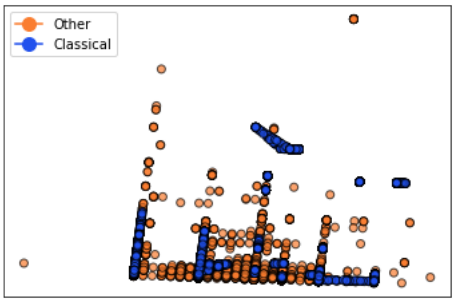

Figure 20: SMOTE Oversampling

2.3.3 Adjusting Class Weights

Striving for an actual improvement on both precision and recall, we try adjusting the weight of the two classes during classification in order to offset the imbalance. Many combination of weights have been tested, but the ones giving the best results are those where the weight for the positive class is five times that for the negative class. Results of such a classification model are found in row (6) of table 3: we can see that this time an improvement at the recall level comes with a slight improvement to the precision too, reaching a +0.04 F1-Score with respect to the base classifier.

We also try to apply this method on an oversampled version of the dataset, with the hope of compensating the discrepancy between precision and recall of that approach. The best weights proportion comes out to be 4:1 for the negative class in this case (row (7)): while performing worse than simply adjusting the weights on the base dataset regarding precision and f1-score, this model obtains the best area under the roc curve overall (shown in figure 21), 96.1%. It is worth noting that this approach, with proportions skewed towards the positive class (e.g. 4:10), manages to reach a Recall as high as 0.94 paired with a Precision equal to 0.45, which could be useful for applications which would want to maximize the former without needing to rely on random undersampling and its much lower precision.

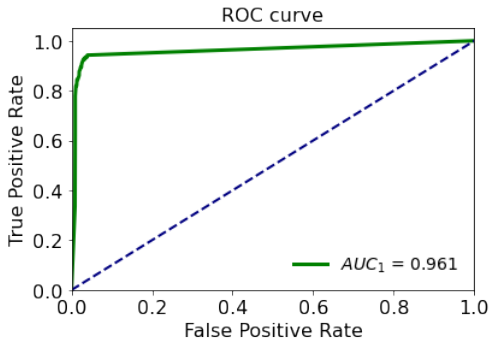

Figure 21: ROC Curve for adjusting class weights on oversampled data.

	Classical			Other		
	Precision	**Recall**	**F1**	**F1**	**Acc.**	**ROC-AUC**
(1) Base Dataset	0.73	0.75	0.74	0.99	0.98	0.909
(2) Random Undersam.	0.23	0.94	0.37	0.95	0.90	0.920
(3) CNN Undersam.	0.20	0.83	0.32	0.94	0.89	0.929
(4) Random Oversam.	0.58	0.85	0.69	0.99	0.98	0.917
(5) SMOTE Oversam.	0.58	0.84	0.69	0.99	0.98	0.915
(6) Adjust Class Weights	0.75	0.82	0.78	0.99	0.99	0.905
(7) Oversamp + Weights	0.67	0.84	0.75	0.99	0.98	0.961
(8) Cost Matrix	0.40	0.84	0.54	0.98	0.96	0.907
(9) Adj. Dec. Thresh.	0.85	0.63	0.72	0.99	0.98	0.934

Table 3: Results for the various expedients used to address class imbalance in classification (decisiontree)

2.3.4 Other approaches

Closing on our exploration of imbalanced learning methods, we try to add a cost matrix to our decision tree with the `costcla` Python module, in order to specify which kind of error the classifier should penalize more. We decide to set the cost for True Positive and True Negatives to zero and to vary the costs for False Positives (setting it to at least one) and False Negatives (which should have a higher cost than FPs, if we want to to improve the model precision). Unfortunately we could not find a cost matrix which allowed us to exceed the performances of adjusting the class weights.

Finally, we try to shift the decision threshold of the classifier: this is the only method among the tested ones which allows us to achieve a precision higher than the recall on the positive class. The best result, obtained with a threshold of 0.8, is shown in row (9) of table 3.

2.3.5 Conclusion

Adjusting the class weights has shown to be the most effective method for obtaining a balanced classifier, sporting overall good precision and recall scores.

Module 2

3 Advanced Classification Methods and Regression

3.1 Advanced Classification Methods

In this section we discuss the results of applying some more sophisticated algorithms to the classification problem defined in section 3, i.e. the recognition of a specific musical genre against. This time we choose to study this problem with respect to the *Rock* genre, the most frequent one in our dataset, due to time and space constraints.

3.1.1 Naive Bayes Classifier

The first advanced classification algorithm we take into considerations are Naive Bayes Classifiers, with their implementations found in the `sklearn` library. We consider the same group of features employed with Decision Tree and split them between numerical and categorical/binary ones. On the first subset of variables we apply a `GaussianNB` classifier, while the latter are given as input to a `CategoricalNB` classifier. Results for both are summarized in table 4

For `GaussianNB` there is just one parameter to tune, `var_smoothing`, the *Portion of the largest variance of all features that is added to variances for calculation stability.* With the default value of 1e-9, we obtain a result particularly skewed towards Recall for the positive class (0.92) and Precision for the negative class (0.80), at the cost of the other two components of confusion matrix, with a Recall for non-Rock tracks as low as 0.24. Testing various thresholds for the smoothing parameter, we find that increasing it, bringing it to values close to 0.5, we can achieve a more balanced result with better performances on the negative class. Lastly, since a classifier of this kind makes a strong assumption on the probability density of the attributes (in this case, that it follows a Gaussian distribution), we try to limit the number of variables, selecting only the eight *Audio Features* columns: the resulting performances come out to be better on each score for both classes, reaching F1 scores of 0.67 and 0.73 with an Accuracy equal to 0.70. In this scenario we keep the smoothing parameter as low as the default value, since we notice that increasing it leads the classifier to perform worse, even stopping to recognize members of the positive class altogether with values greater than 0.001.

For `CategoricalNB` we have to tune the parameter `alpha`, which controls again a smoothing factor and is set to 1 by default. Trying out different values, from as low as 0 to as high as 1000, we find out that the best overall accuracy, 0.67, is achieved with `alpha` between 85 and 500. In table 4 is displayed the result which maximizes the F1-score, obtained with an additive smoothing factor equal to 3.

3.1.2 Logistic Regression

Due to the high number of parameters available for `LinearRegression` (still from the `sklearn` library), we rely again on cross validation to tune them, performing 100 iterations of Randomized Search. The hyper-parameter matrix contained *newton-cg*, *lbfgs* and *liblinear* as solvers; *none*, *l1*, *l2* and *elasticnet* as penalties; and a C parameter (controlling the regularization strength) ranging from 1e-5 to 100 at logarithmic steps. The best configuration found has Newton-CG as solver, None as penalty and C equal to 0.0012. Fitting a Logistic Regression model on the training set with this parameters gives the results on the test set shown in table 5.

Gaussian NB								
	Rock			Other				
	Precision	Recall	F1	Precision	Recall	F1	Acc.	ROC-AUC
Low Smoothing	0.46	0.92	0.62	0.80	0.24	0.37	0.52	0.706
High Smoothing	0.55	0.68	0.61	0.72	0.60	0.66	0.63	0.691
Audio Feat. Only	0.62	0.72	0.67	0.77	0.69	0.73	0.70	0.771
Categorical NB								
Alpha = 3.0	0.53	0.73	0.61	0.74	0.53	0.62	0.62	0.692

Table 4: Results for Naive Bayes Classifiers.

Logistic Regression			
	Acc. 0.71	**AUC 0.76**	
	Precision	**Recall**	**F1**
Rock	0.69	0.58	0.63
Other	0.73	0.81	0.77

Table 5: Results for Logistic Regression Classification.

Rule-Based (RIPPER)			
	Acc. 0.77	**AUC 0.74**	
	Precision	**Recall**	**F1**
Rock	0.86	0.53	0.66
Other	0.74	0.94	0.83

Table 6: Results for Rule-Based Classification.

```
[favorites_artist=0.0-2.0^n_tracks=8.0-10.0^listens_album=3480.0-4879.0]
[tempo=129.16-139.96^listens_track=144.0-239.0^speechiness=0.04-0.05]
[energy=0.91-1.0^danceability=0.05-0.23^speechiness=0.06-0.07]
[danceability=0.23-0.31^energy=0.73-0.82^location=16.0-39.0]
[favorites_artist=4.0-6.0^artist_pop=0.65-0.7^wikipedia_page=1]
```

Figure 22: Example of frequent rules used for classification.

3.1.3 Rule-based Classifiers

Next we show the result obtained with a classifier which utilizes a set of patterns found in the data: the algorithm employed is RIPPER, with its implementation found in the library `wittgenstein`. Figure 22 five of the 76 rules extracted by RIPPER (each "$\hat{}$" is meant to be read as "AND"). Table 6 shows the results of such classification, which achieves high values for Precision on the positive class and Recall on the negative class, similarly to the previously discussed Logistic Regression classifier, but with better overall performances.

3.1.4 Support Vector Machines

We present below the two classification tasks solved with linear and non-linear support vector machine methods. In order to test this algorithm we choose to classify against each other two genres that our exploratory has suggested to be quite different (Hip-Hop and Folk), and two genres seems to be less clearly separable in the feature space, also due to their bigger support (Rock and Electronic). The selected variables for this task comprehend audio features, social features, the duration of the tracks and the bitrate of the audio file. The PCA representation is used for plotting the data in 2 dimensions (Figure 23).

After normalizing the data with a `StandardScaler` and having encoded the labels into binary values, the data was partitioned in training set (70%) and test set (30%). In order to find the optimal hyper-

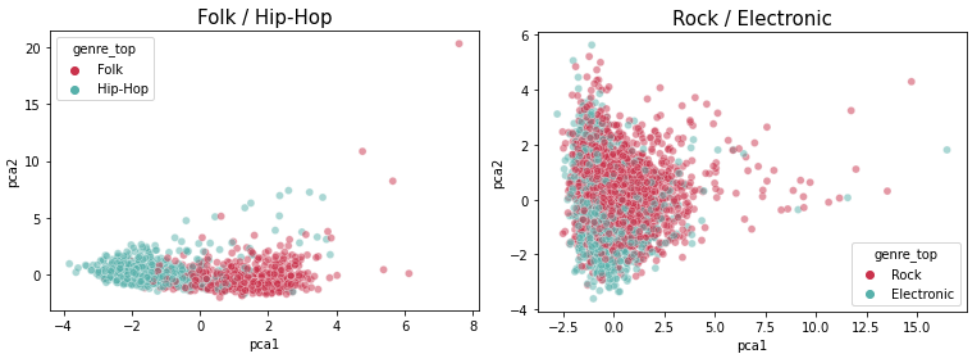

Figure 23: Our two-class separation problems

parameters configuration, we perform random search in combination with a 5 fold cross validation. The Soft Margin SVM was tested on RBF, polynomial, linear and Sigmoid. The best model configurations and respective classification reports are shown in table 7.

As we had hypothesized, the evaluation metrics are on average much higher in the classification of Folk against Hip-Hop, which are well classified by both linear and non-linear SVM. In any case, the results obtained in the classification of the Rock and Electronic genres are still satisfactory, in particular using the radial basis function kernel.

3.1.5 Neural Networks

We propose below the operations carried out for the construction of a Multi-Layer Perceptron able to classify the *Rock* genre, using the Keras library. In this case we have used all the variables of the dataset, both numeric and categorical, after having appropriately encoded them. In addition to the test set (30% of the entire dataframe: 3870 records), two other partitions were created from the remaining records: a training set (6320 records) on which to train the model and a validation set (30% of the dataframe still available: 2709 records), following the same probability distribution as the training dataset, used to tune the hyperparameters (i.e. the architecture) of the neural networks, in order to avoid overfitting on the test set.

As a first approach to the construction of the neural network structure with Keras, we have made some attempts to set the parameters on the basis of what we have read in the literature regarding the improvement of the model's performance as the number of hidden layers and nodes per hidden layer varies.

Furthermore, since the models for this binary classification gives us a score instead of the prediction itself, as a preliminary step we have converted this score into a prediction applying a threshold. The value of the latter was calculated following the procedure illustrated in [2] and was set to 0.42.

In building our different networks, we used the *HeUniform* activation function, which works best for layers with *ReLu* activation [2].

We then performed a random search using a 5 fold cross validation, testing:

- number of hidden layers = 1 to 3;
- hidden layer size = 10, 15, 20;
- activation function = ReLu;
- optimizer = SGD, Adam;
- *momentum* = from None to 0.8;
- learning rate = 0.1, 0.01, 0.001, 0.0001;

[2] Delving Deep into Rectifiers

	Hard Margin SVM					Soft Margin SVM				
Parameter	*C = 0,35*					*C = 5,7 — kernel = 'rbf'*				
	precision	**recall**	**f1**	**Acc.**	**AUC**	**precision**	**recall**	**f1**	**Acc.**	**AUC**
Hip-Hop	0.92	0.85	0.88	0,89	0,89	0,93	0.90	0.91	0,92	0,92
Folk	0.87	0.93	0.90			0.91	0.93	0.92		
Parameter	*C = 8,8*					*C = 7,8 — kernel = 'rbf'*				
Rock	0.67	0.78	0.72	0,79	0,78	0.79	0.73	0.76	0,83	0,81
Electronic	0.86	0.79	0.83			0.86	0.89	0.87		

Table 7: Results for both classification tasks with Hard Margin and Soft Margin SVM methods.

- *lambd* = 0.1, 0.01, 0.001, 0.0001.

The output activation function was set to *Sigmoid* as our task is binary. Particular attention was paid to the choice of the range of *lambd* values, without which the model tended to excessively increase its complexity, bringing it into over fitting. In any case, since the dimensionality of the input nodes is anything but high, our model proved to be particularly resistant to overfitting, barely visible only beyond 600 eras. The `RandomizedSearchCV` suggested a neural network with two hidden layers (20, 10) and set on the following parameters:

- activation function: ReLu;
- optimizer: SGD;
- *momentum*: 0.6995244855983086
- learning rate: 0.0014402562254234638,
- *lambd*: 0.10442267196915725.

In the end, we evaluated the model with the best parameter configuration identified, training it on the entire available dataset and finally testing it on the test set. Table 8 shows the results obtained. Although the values are generally acceptable and reflect, with some improvement, those obtained on the validation set, we were not able to obtain results in line with the other classifiers implemented. In particular, there is a rather low recall value, which indicates a barely acceptable ability to identify the *Rock* class.

	Neural Network		
	Acc. 0.76	**AUC 0.75**	
	Precision	**Recall**	**F1**
Rock	0.77	0.63	0.69
Other	0.77	0.86	0.81

Table 8: Results obtained by Neural Network on test set.

	Base Estimator (Dec.Tree)		
	Acc. 0.78	**AUC 0.78**	
	Precision	**Recall**	**F1**
Rock	0.72	0.75	0.74
Other	0.82	0.80	0.81

Table 9: Results obtained by the base estimator (Decision Tree) for our Ensemble Classifiers.

3.1.6 Ensemble Methods

In this subsection we describe the application of Random Forest, Bagging and Boosting to our Rock songs classification problem, with the goal of improving the performances with respect to the best Decision Tree obtained in section 3.1, which is used as base estimator. For reference, table 9 shows again the classification report of such decision tree.

We evaluate our Random Forest on a validation set of size 30% of the train set, to check the effect of increasing/decreasing the

	Random Forest		
	Acc. 0.89	**AUC 0.96**	
	Precision	**Recall**	**F1**
Rock	0.90	0.81	0.86
Other	0.88	0.95	0.91

Table 10: Results obtained by Random Forest (500 estimators).

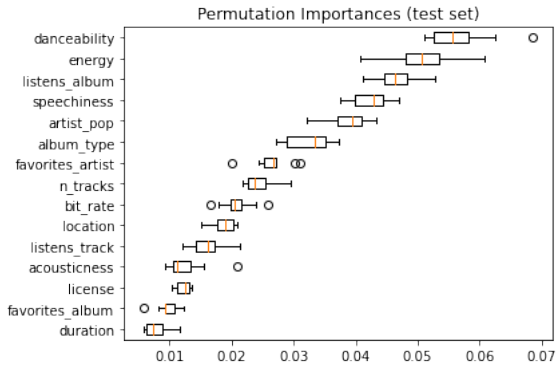

Figure 24: Permutation importance of the top 15 features in Random Forest.

number of estimators and of the parameter `max_features`, which regulates the number of features to consider each time when looking for the best split. We find out that the best results are obtained with around 500 estimators: using more than that actually leads to slightly worse results than if we were to use just 100. The best performing value for `max_features` is instead `log2`. The resulting Random Forest leads to outstanding improvements on every metric with respect to the base estimator, shown in table 10, especially on the precision for the positive class. Figure 24 shows the permutation importance for the top-15 most important features in this Random Forest model: *danceability* runs to the forefront with *energy* and *listens_album* substitutes *listens_track*, already mentioned when described the base estimator.

Training a `Bagging Classifier` using our optimized decision tree as base estimator gives slightly worse results than Random Forest, with a decrease of around 0.03 points on each metric in the confusion matrix. Passing that Random Forest model as base estimator for the bagging algorithm, we instead obtain the exact same results of just using the Random Forest, meaning that this procedure does not manage to further improve the performances of our best model so far.

Lastly, we test the performance of a Boosting approach on our problem, learning an `AdaBoost Classifier`. Similarly to what we experienced with Bagging, passing the optimized decision tree as base estimator returns a model with slightly but decidedly lower performances, while boosting directly our random forest does not produce any increase in precision or recall for neither of the classes. However, after reading that AdaBoost has been reported to perform worse when dealing with irrelevant features[3], we run it again using only the top-9 most important features observed in Random Forest (see figure 24). Taking more or less features is shown to lead to worse results. This manages to improve the Recall on the positive class by +0.02, while decreasing only the Recall for the negative class by -0.01. Due to approximations, accuracy, area under ROC curve and F1 scores are report to be equal to the ones of Random Forest, but overall we consider this to be an improvement, albeit small, on the performances of that model, especially since it shifts some of the goodness from the negative to the positive class. Table 11 shows results for this last classification approach, while its ROC curve is visualized in figure 25.

[3]The Ultimate Guide to AdaBoost, random forests and XGBoost

	AdaBoost		
	Acc. 0.89	**AUC 0.96**	
	Precision	**Recall**	**F1**
Rock	0.90	0.83	0.86
Other	0.88	0.94	0.81

Table 11: Results obtained by Adaboost on the top-9 important features for the Random Forest taken as base estimator.

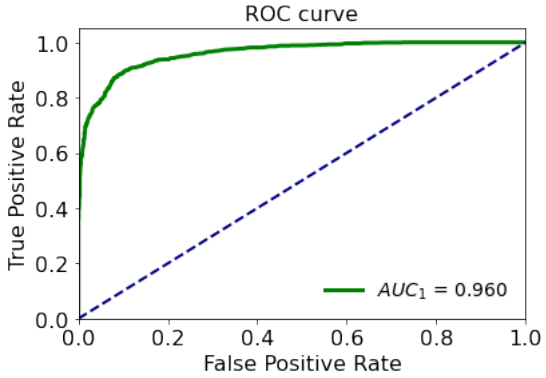

Figure 25: ROC curve for AdaBoost classifier

3.1.7 Conclusion

Ensemble methods have been shown to be the best approach for our classification problem, thanks to their performances, both in absolute terms and with respect to the other classifiers trained, and balanced confusion matrices. These models are also characterized by high computational speed and easy model setup thanks to the near absence of parameters to tune: the same cannot be said for the second best category of classifiers we trained, Neural Networks. Therefore Ensemble classifiers would be the ones we would rely on if we had to apply our methodology to the classification of other musical genres on this dataset or on a similar one.

3.2 Regression problem

In this section we describe our Linear Regression model, built in order to predict how *hot* a song is (`song_hotttnesss`, dependent variable, ranging from 0 to 0.5,) based on the popularity of the artist (`artist_pop`, independent variable, whose creation is described in section 2.3.3.): we want to find out how much a given song is or can become famous based on the popularity of its artist. The choice of this task was suggested by the analysis of the correlation matrix (figure 7), in which the variable `song_hotttnesss` turns out to be quite well correlated (0.5) to all three variables aggregated for creating `artist_pop`. In an attempt to improve the fit of the regression model on the data, we also used Ridge and Lasso regularizations, searching for the best parameter of alpha with a `GridSearchCV`.

3.2.1 Simple Linear Regression

The results obtained by implementing Simple Linear Regression with Scikit-Learn without any regularization were generally better than those obtained by implementing Lasso and Ridge regularization, although not fully satisfactory. R2 score, in fact, shows that just 10% of the variance of the dependent variable `song_hotttnesss` is correctly predicted by the variable `artist_pop`, with an MSE of 0.002 and a MAE of 0.034 on the test set.

As shown in Figure 26, the high number of zero values prevents the linear regression line from following the trend of the other points as we hoped. This is confirmed by the distplot on the right, which shows how our model cannot predict values that go above the 0.5 threshold. Lasso and Ridge regularization, at best, only managed to match the results obtained by SLR with no regularization (Table 12).

3.2.2 Multiple Linear Regression

By adding to the linear regression model other independent variables on the basis of which to predict the dependent variable, its

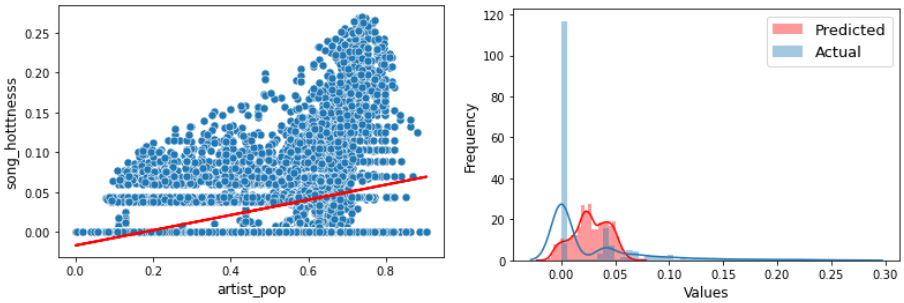

Figure 26: On the left: Fit of the independent variables (`artist_pop`) against the dependent variable (`song_hotttnesss`) using the SLR model with no regularization (*Base*). On the right: distribution of the predicted/actual values and their frequency.

Simple Linear Regression				
Model	**Coefficient(s)**	**Intercept**	**Alpha**	**R2**
Base	0.095	-0.017	-	0.108
Multiple Linear Regression				
Base	1.02e-01 3.86e+01 -2.19e-05	-0.023	-	0.503
L1 Reg.	8.04e-02 0 -1.03e-05	-0.006	0.0016	0.126
L2 Reg.	1.18e-01 1.78e+01 -3.25e-05	-0.027	0.0075	0.395

Table 12: Parameters of the regression model implemented and their R2 score. L1 and L2 regularizations for the Simple Linear Regression task are not reported as they are almost identical to those obtained without regularization (*Base*).

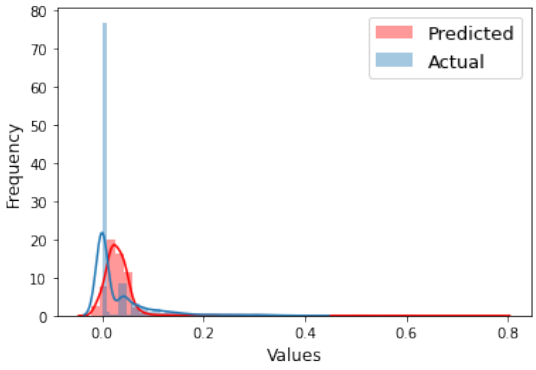

Figure 27: Distribution of the predicted/actual values and their frequency with Multiple Linear Regression with no regularization.

performance is significantly improved. The choice of the other independent variables fell on those with a statistically significant correlation with the dependent variable and in particular on `favorites_artist` and on `song_currency` (Figure 7).

Again, the best results were achieved by the multilinear regression model without regularization (Table 12). In this case, the R2 score is significantly higher than that obtained in the SLR, and indicates that our new model is able to correctly predict about 50% of the variance of the dependent variable `song_hotttnesss` based on the other independent variables.

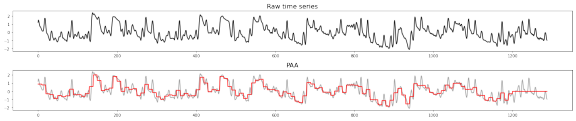

Figure 28: Example of PAA approximation applied to one of our time series.

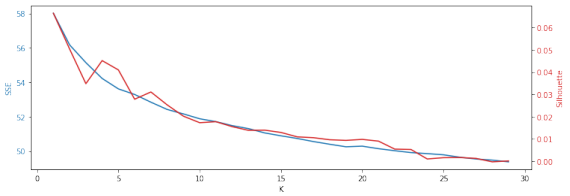

Figure 29: Elbow Method with SSE and silhouette score on TimeSeriesKMeans clustering with k from 0 to 30.

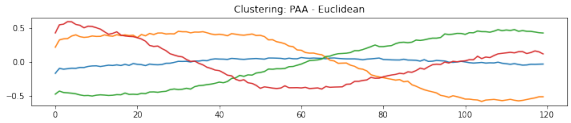

Figure 30: Centroids for the clustering obtained using PAA and Euclidean distance.

Module 3

4 Time Series Analysis

In this section we will discuss the analyzes conducted on the times series dataset provided by the web page of Data Mining 2 course.

4.1 Data preparation

In the dataset used, each time series corresponds to the spectral centroid of the track, a measure of the center of mass of the spectrum. As for the other tasks, here too we have selected from the total dataset only the 13k tracks with the audio features. Therefore, all traces containing null values have been eliminated.

On each time series an amplitude scaling was applied and noise cleaned by applying a moving average on a window of 4 time stamps.

4.2 Clustering

The algorithm chosen for the clustering task is the K-Means, which has been applied on the dataset, approximated with PAA and SAX, using three different metrics: Euclidean, DTW and SoftDTW.

4.2.1 Clustering with PAA

The first step for this task was to choose an appropriate number of segments to correctly approximate the shape of the time series. After testing the approximation result with different parameters, we opted for 120 segments, computed using the *Piecewise Aggregate Approximation* (PAA) transformation provided by `tslearn` (Figure 28).

Therefore, we focused on choosing the optimal number of clusters by computing the silhouette score and the SSE with the "Elbow-method". As can be seen in the graph in the Figure 29, the curve described by the two scores not only does not have a real elbow, but stands at average high values of SSE and almost unchanged in *silhouette*: by varying the number of clusters from 0 to 30, the silhouette it never exceeds 0.06. The choice of the number of clusters, therefore, fell on 4 for two reasons: a) in correspondence with it there is a slight peak of the silhouette; b) in the analysis phase, a small number of clusters allows a better comparison of the various distributions.

The first attempt at clustering was carried out using the Euclidean metric (the centroids are represented in Figure 30). The 4 resulting clusters are distributed as follows:

- (In green) A cluster of size 2334 of time series with low values at beginning and higher value at the end;
- (In blue), a cluster of size 6849 with nearly stationary values across the time series;

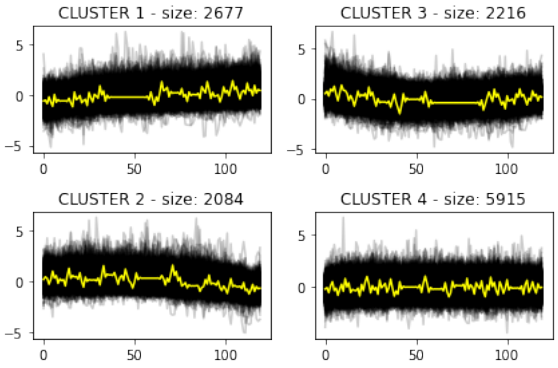

Figure 31: Centroids for the clustering obtained using PAA and DTW distance.

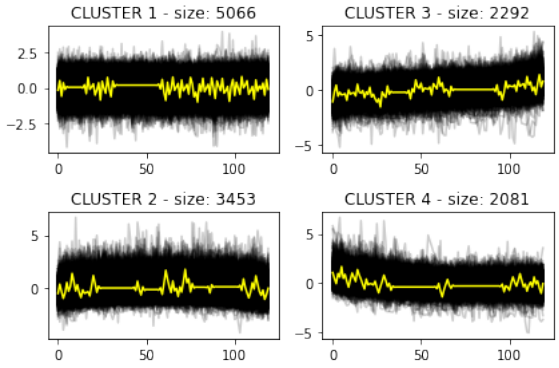

Figure 32: Centroids for the clustering obtained using PAA and SoftDTW distance.

- (In red), a cluster of size 1848 where high values are found both at the beginning and at the end of the time series, but not in the middle;
- (In orange), a cluster of size 1861 with high values at the beginning and low values at the end.

As expected, the SSE (54.95) and the Silouhette (0.048) confirm the values anticipated by the *Elbow-method* graph. From the study of the distribution of genres in each cluster, we have found that these are not well-defined clusters, as the distribution within them is completely homogeneous.

The results obtained by applying the DTW with Itakura global constraint show a slight decrease in the SSE (21.4), but also in the silhouette (0.03). Furthermore, the size of the clusters (Figure 31 is less unbalanced than the clustering implemented using the Euclidean distance. Also in this case, the distribution of genres in each cluster appears anything but characterizing.

The last attempt carried out on the approximated date with the PAA was conducted using the SoftDTW, with which, however, we obtained a value of SSE far superior to the SSE of the other clusterings, equal to 388.7. This value, an indication of the poor quality of clustering, is confirmed by the minimum silhouette value obtained

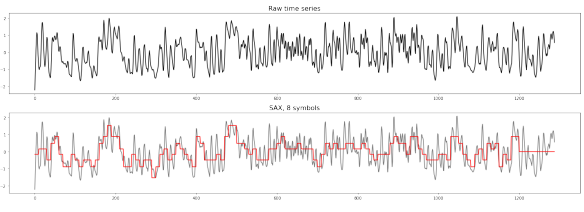

Figure 33: Example of SAX approximation applied to one of our time series.

so far (0.008).

4.2.2 Clustering with SAX

Next, we repeat our clustering procedure with another version of our dataset, this time containing time series approximated using the Symbolic Aggregate Approximation (SAX) method, with an implementation found in the `tslearn` library. The series are being divided again in 120 segments, each assigned to one of 8 different symbols. An example of SAX approximation can be found in figure 33.

The clustering algorithm employed is still K-Means, again with a number of clusters to find set to 4. For the first run we cluster considering the Euclidean distance between the time series, with results visualized in figure 34, where the four centroids are plotted. It is evident that the results obtained almost perfectly match those obtained with the same metric on the PAA approximated

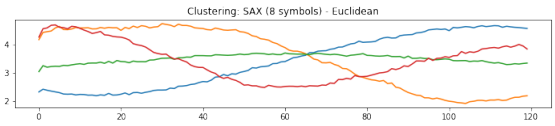

Figure 34: Centroids for the clustering obtained using SAX and Euclidean distance.

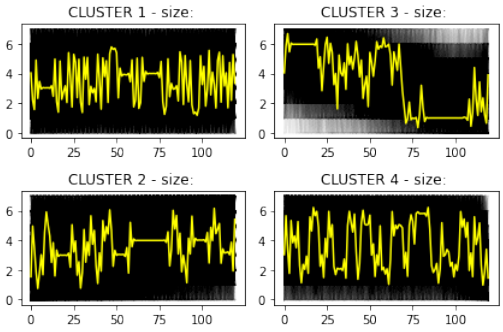

Figure 35: Centroids for the clustering obtained using SAX and DTW distance.

dataset. The resulting SSE for this clustering is 355.

Using DTW as distance metric we obtain more characterized centroids (shown in figure 35), even if they become less interpretable. The SSE for this clustering is 116, less than one third of the one obtained using the Euclidean distance.

Constraining the dynamic time warping with a Sakoe-Chiba band we notice a pattern similar to the one observed with Euclidean distance, with the centroids having higher values in specific zones of the series. The SSE for this clustering, shown in figure 36, is 274.

Applying an Itakura parallelogram as constraint the centroid describing the clusters become again more detailed, as shown in figure 37. SSE for this clustering is 131.

The study of the distribution of genera in the different clusters generated by the different clustering algorithms is once again not very informative. Each genre, in fact, tends to be distributed in the different clusters almost always with the same percentage, which reflects the distribution over the entire dataset.

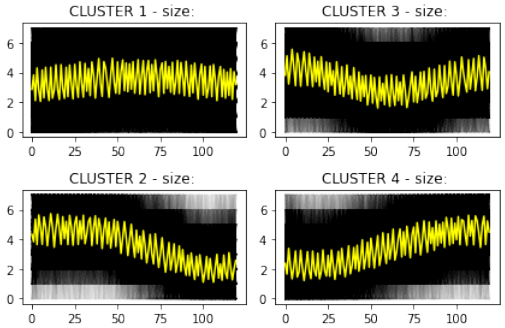

Figure 36: Centroids for the clustering obtained using SAX and DTW distance with Sakoe-Chiba band.

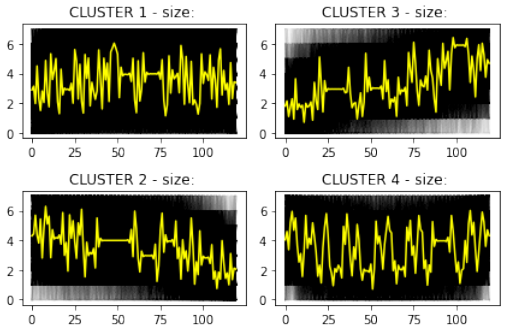

Figure 37: Centroids for the clustering obtained using SAX and DTW distance with Itakura parallelogram.

4.3 Motifs and anomalies

To search for motifs and anomalies we select a small subset of tracks, comprising the twenty most listened *Folk* songs (based on the `listens_track` feature). The following procedure has been tested first utilizing the full length of the available time series, but here we display the results obtained passing to the algorithms only the first sixth of each (215 time stamps instead of 1292), for a clearer visualization of the peaks and valleys of the matrix profiles.

For each track we build and plot the matrix profile using a window of size 10, through the use of the `matrixprofile` li-

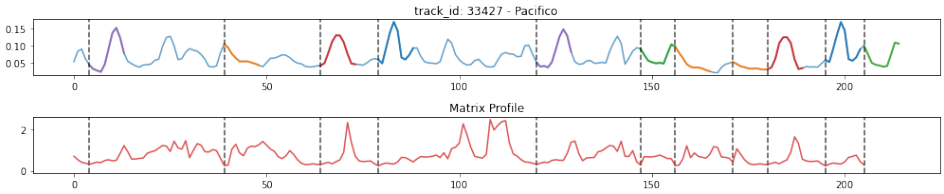

Figure 38: Time Series, Motifs and Matrix Profile for the song *Pacifico*.

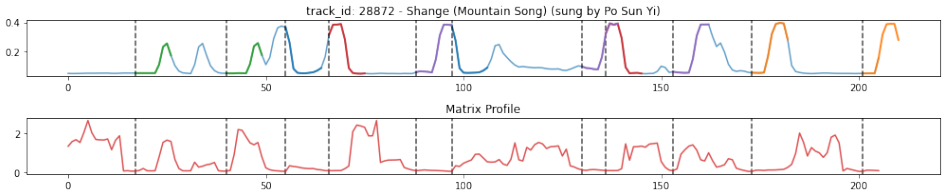

Figure 39: Time Series, Motifs and Matrix Profile for the song *Shange*.

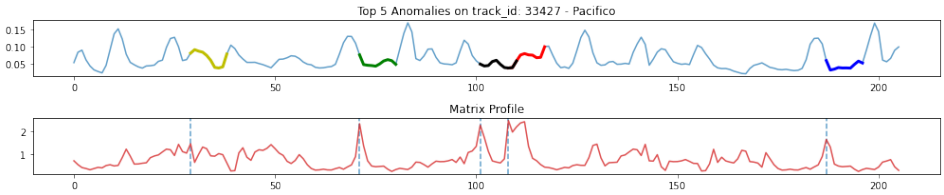

Figure 40: Time Series, Anomalies and Matrix Profile for the song *Pacifico*.

brary. As a first try, we then extract the top ten motifs, highlight them on the time series and plot it together with the matrix profile to evaluate the results. We quickly notice how hard it is to set a number of patterns to find which aptly fits all the twenty songs taken into consideration. Most of the tracks seems to have different quantity of *real* motifs, and while searching for 10 motifs highlights not-so-similar regions in time series with few of them, lowering the parameter in question leaves some very obvious motifs undiscovered in others which contain more. In figure 38 and 39 we show two time series for which finding 5 motifs produce visually satisfactory results. For *Pacifico*, the motifs are found at indexes [79, 195], [39, 156, 171], [147, 205], [64, 180] and [4, 120] (marked by dotted lines). Their heights in the matrix profile are 0.245, 0.265, 0.292, 0.309 and 0.319 respectively. For *Shange*, the motifs are found at indexes [55, 97], [173, 201], [17, 40], [66, 136] and [88, 130, 153]. Their heights in the matrix profile are 0.039, 0.042, 0.054, 0.0768 and 0.077 respectively.

We then employ the function `discords` to discover the top-5 anomalies on the time series for *Pacifico*. The methodology is the same as for finding motifs, except this time the algorithm searches for peaks on the matrix profile instead of valley. The `ex_zone` parameter has been set to 5. In figure 40 we can see that indeed outlier sequences are identified at indexes 108, 112, 71, 101 and 187, corresponding to the five highest peaks in the matrix profile.

4.4 Shapelet-based classifier

For this classification task we have chosen to use the time series dataset with exclusively the tracks belonging to the Hip-Hop

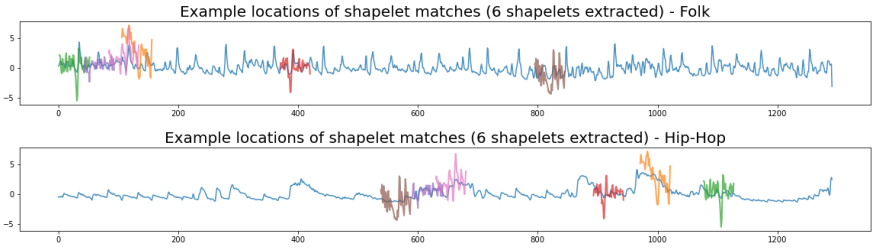

Figure 41: Example of location of extracted shapelets on two tracks belonging to Hip-Hop and Folk.

and Folk genres, which being very characterized, are well suited to be classified, for a total of 1775 records.

	KNN classifier			
	precision	**recall**	**f1**	**acc**
Hip-Hop	0.81	0.83	0.82	0.81
Folk	0.83	0.81	0.82	
	Decision Tree classifier			
Hip-Hop	0.78	0.83	0.80	0,80
Folk	0.83	0.77	0.80	

Table 13: Classification report of our two shapelet-based classifiers: KNN and Decision Tree.

First, we created a dataset of shapelets identified with ShapeletModel, using Adam as optimizers, a regularization of 0.01, and $l = 0.04$, which indicates the fraction of the length of time series to be used for base shapelet length. On 1500 training epochs, this parameter configuration returned a correct classification rate of 0.85. In this way we have identified 6 shapelets of length equal to 52 (Figure 42).

The shapelets appeared very dissimilar from the motifs extracted, especially since the motifs' smaller length makes such a comparison difficult.

Figure 41 shows an example of the matching of the 6 extracted shapelets with 2 correctly classified tracks belonging to the two genres of our dataset.

Once we obtained our shapelets dataset, we implemented two classification models,

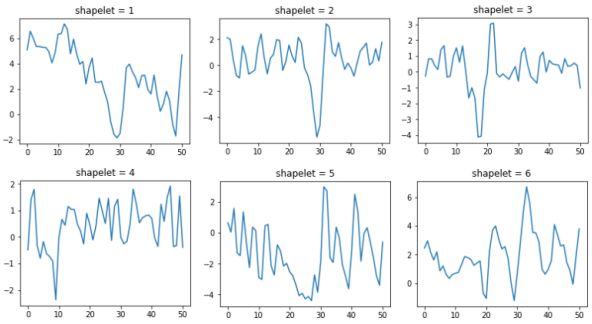

Figure 42: Extracted shapelets.

a Decision Tree and a KNN classifiers, of which we report the results obtained on the test set in the table. Based on the parameter configuration returned by the `GridSearchCV` on the KNN, we used the Manhattan distance, a number of neighbors equal to 13 and the distance as weight function used in prediction.

A GridSearchCV was also performed on the Decision Tree in order to find the best configuration ('`criterion`': 'gini', '`max_depth`': 5, '`min_samples_leaf`': 7, '`min_samples_split`': 5), however the results obtained are not were as good as those obtained with the KNN, which managed to correctly rank the records of our test set with an accuracy of 82%.

5 Sequential Pattern Mining

In this section we describe our methodology for performing Sequential Pattern Mining and briefly describe its results. First of all, due to the high computational complexity of this task, we selected a small subset of time series to perform it: the choice falls on those relative to the 437 *Classical* tracks (which have already been targets of analysis in the Imbalanced Classification section). In order to reduce again the computational load, we only consider the first time of each series for this task (about 600 time stamps instead of 1200). SAX approximation is then applied to such time series, choosing 60 as number of segments and 8 as alphabet size, maintaining the proportion of 10 time stamps for each segment we used in section 5, while working on time series clustering.

Afterwards, we run the pattern mining algorithm Prefix Span, in its implementation found in the library `prefixspan`[4]. We experiment by choosing different support thresholds and minimum pattern length, as well as extracting the top-n most frequent patterns. Interpreting these results is however very difficult, partly also due to the heavy approximation applied in the preprocessing phase, essential for making this approach runnable on our machines in reasonable time. Figure 43 shows the top-15 most frequent patterns of at least 2 segments extracted from the series.

	Support	Pattern
0	427	[3, 3]
1	423	[2, 2]
2	423	[2, 3]
3	423	[4, 3]
4	421	[3, 2]
5	417	[2, 2, 2]
6	415	[4, 4]
7	414	[2, 2, 3]
8	414	[5, 4]
9	413	[4, 2]
10	412	[3, 2, 2]
11	412	[3, 3, 2]
12	412	[3, 4]
13	412	[5, 3]
14	410	[3, 3, 3]

Figure 43: Top 15 most frequent sequential patterns extracted from Classical tracks

6 Advanced Clustering

6.0.1 X-Means

For the task about Advanced Clustering we choose to utilize the X-Means algorithm, with its implementation found in the library `pyclustering`, and to compare its results with its 'ancestor', K-Means.

Initially, to somewhat tie this task to the classification ones previously described we would have liked to run this clustering algorithm on our whole dataset and check if the results could be correlated in any way to the genre distribution, but after many plot-aided analysis we find out that songs from the most common genres (i.e. *Rock* or *Electronic*) tend to be spread out on the whole feature space with a lot of overlapping between them. This happens when we reduce the data dimensionality through PCA, but also if we consider any pair of features to build a scatterplot. We feel that this would make the clustering analysis not very interesting to perform and to read, since there would be no visual way of appreciating the results, leaving us with just aseptic numbers in the form of SSE and Silhouette.

For this reason we choose to perform clustering on a much smaller dataset, com-

[4] https://pypi.org/project/prefixspan/

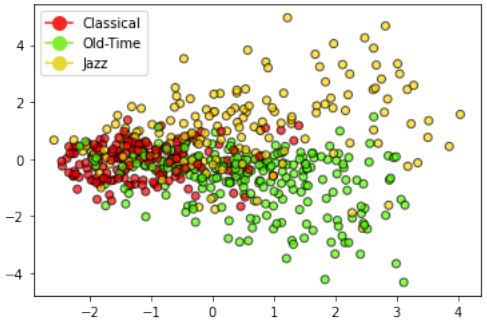

Figure 44: Genre Distribution for our clustering task

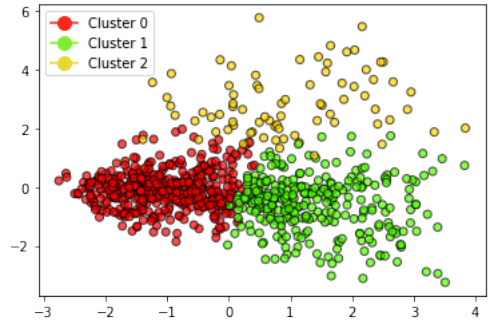

Figure 45: Clustering with K-Means (K=3)

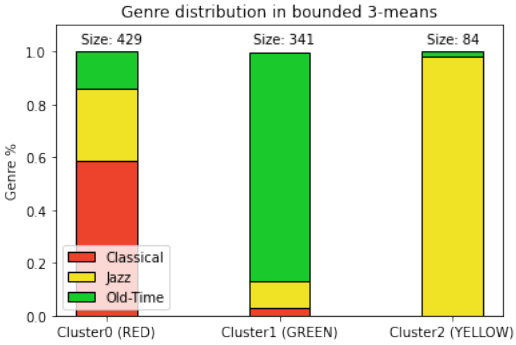

Figure 46: Cluster size and genre % in 3-means clustering

prised by records pertaining to songs belonging to less represented genres, hoping to more characterized spacial distributions. After some tries we select *Classical*, *Old-Time/Historic* and *Jazz* songs, 854 tracks in total. Applying PCA on the 8 audio-features column, often employed in our previous analysis, we plot the data in a scatterplot, highlighting the genre of each track with the color of its point: in figure 44 this genre distribution is shown. We can see that, while not completely separated, Jazz songs tend to lie on the upper part of the space (with the y component greater than 0), while Old-Time tracks on the lower section of the graph. Classical songs are found in a region where both other genres are present, but they are still contained in the left region of our 2-dimensional data.

First of all we run the base K-Means algorithm with k=3, to see how the clustering relates to genre distribution. Results are shown in figure 45, with points colored based on their cluster label this time.

We notice that the algorithm seems to separate the three genres quite well, albeit simplifying excessively in the region where Classical overlapped with the other two genres. For this clustering we calculate an SSE equal to 4785 and a Silhouette score equal to 0.23. Figure 46 shows the genre distribution in each cluster.

We then proceed by applying X-Means instead. The algorithm suggests an optimal number of clusters equal to 12, obtaining an SSE equal to 2303, less than half of that obtained with 3-means, and a Silhouette score of 0.22. The clustering is shown in figure 47, even though with so many clusters, often overlapping, the scatterplot becomes harder to interpret. The much lower SSE, however, signifies that the algorithms must be able to identify more detailed patterns inside the data.

Lastly, we want to see if the 12 clusters obtained with X-Means might be combined together to obtain a similar results to 3-Means. To do so, we run X-Means again setting to 3 the maximum number of clusters to identify. We obtain a result very similar to the one from 3-means, both in terms of SSE and Silhouette score (4980 and 0.22 respectively) and graphically (shown in fig-

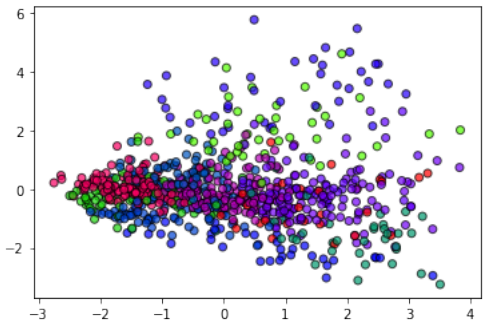

Figure 47: Clustering with X-means (12 clusters)

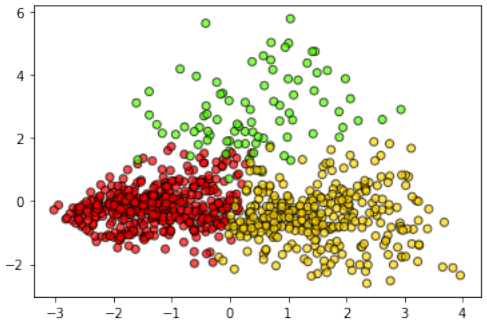

Figure 48: Clustering with X-means (Bounded to max. 3 clusters)

ure 48).

Figure 49 shows the genre distribution among the X-means clusters and how they can be easily grouped in order to form bigger aggregated clusters with nearly identical total distributions and sizes to those shown in 46 (slight oscillations can be attributed to the randomness in selecting the initial centroids).

Since X-means works bisecting clusters at each step to create new ones, this shows that the 12 clusters obtained are 'children' of these three macro-clusters which roughly corresponds to the genre distribution, confirming that the clusters it finds are in a way representing more fine-grained patterns present in the Audio Features variables among each of the genres.

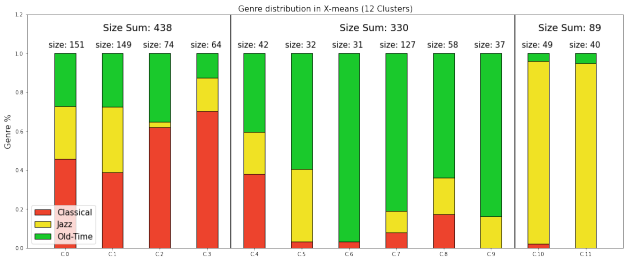

Figure 49: Cluster size and genre % in X-means clustering

7 Transactional Clustering

For this task, we used only records labeled with just one genre, thus eliminating all the records having *Ambiguous* as value for the `genre_top` attribute. Furthermore, a discretization of different continuous variables was carried out, and in particular:

- all *audio features* and the `artist_pop` feature have been discretized into 3 discrete intervals: *Low*, *Medium* and *High*;
- the `duration` attribute has been discretized on the basis of 4 intervals. Tracks with a duration between 0 and 180 (3 minutes) were considered short, songs with a duration from 180 to 360 (6 minutes) as normal, those up to 15 minutes long, while those that exceeded this threshold (1, 42% of the dataset) were considered outliers and therefore removed from the dataset;
- the attribute referring to the release date of the albums was first of all formatted taking into account only the month and therefore discretized according to the season. This choice was made thinking of the difference in sonority between summer and winter songs;
- the `listens_track` attribute has been discretized identifying 3 thresholds: class 0 has collected the tracks with a low number of plays (from 0 to 800), class 1 those with a number of plays

Cluster	Size	Centroids
0	2487	Duration: 'Normal', 'High_listens', 'Low_acoustic.', 'High_danc.', 'High_energy', 'High_instr.', 'Medium_valence', 'Medium_pop', 'summer', 'Electronic'
1	2652	Duration: 'Normal', 'Low_listens', 'High_acoustic.', 'Medium_danc', 'Medium_energy', 'High_instr', 'Medium_valence', 'Low_pop', 'winter', 'Rock',
2	1712	Duration: 'Short', 'Low_listens', 'Low_acoustic.', 'Medium_danc', 'High_energy', 'Low_instr', 'High_valence', 'Medium_pop', 'autumn', 'Rock',
3	2281	Duration: 'Normal', 'Low_listens', 'High_acoustic.', 'Medium_danc', 'Low_energy', 'High_instr', 'Low_valence', 'Medium_pop', 'autumn', 'Rock'

Table 14: K-Modes clusters centroids and size. For a clearer view, the values that repeated the same for each cluster have not been reported and in particular: `album_type` = 'Album', `wikipedia_page` = '0', `location` = 'United States', 'Low_live', 'Low_speech', 'Medium_tempo'.

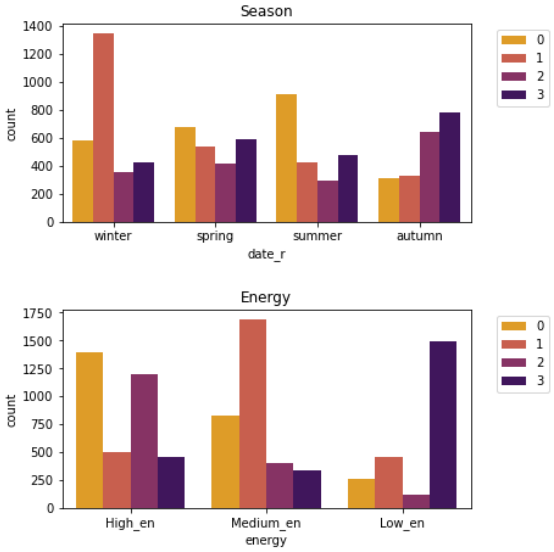

Figure 50: Count plot of Energy and Season (discretized `date_r`) attribute

between 800 and 7500, all the others have been labeled with the value 2.

In addition to the attributes just mentioned, we also used the `album_type`, `wikipedia_page` and `location` attribute, for a total of 9132 records and 15 attributes.

The number of clusters was set to 4 in order to test the algorithm's ability to create clusters based on the publication period of a musical project, in our case the 4 seasons. For clustering our categorical variables we used K-modes with Huang initialization.

Our choice led us to interesting results. In table 14, we show the 4 K-modes centroids obtained. Cluster 0 seems to collect summer tracks, with a low acousticness value: the tracks are recorded with electronic instruments, as confirmed by the fact that the centroid well represents the Electronic genre. Furthermore, the traces of this cluster appear very danceable and energetic, typical characteristics of summer hits.

The second cluster seems to collect, however, the traces published in the winter months. The characteristics are often complementary to those of cluster 0: the tracks have high values of acousticness and instrumentalness, while maintaining a good energy and dance ability, characteristics that seem to belong to the songs of the Christmas holidays. The last two clusters, on the other hand, seem descriptive of two different types of Rock songs: in fact, if in cluster 3 the songs have lively characteristics (high valence, medium dance and energy and low acousticness) typical of the wildest Rock, cluster 4 it seems to collect more melancholy tracks, with a high acousticness value and a low valence value.

Module 5

8 Explainability

In this section we apply different explanation methods to interpret the classification made by a black box model. We choose to analyze the Random Forest for the recognition of tracks belonging to the *Rock* genre, described in section 4.1.6 and whose performances are reported in table 10. The explanation process will be carried out both in a global and in a local way: for the latter two random rows of different class correctly classified by the black box will be chosen.

8.1 Global Interpretation

A first simple approach to global interpretation involves the use of Decision Trees: training a model of this kind using as train labels the predictions given by the black box (instead of the ground-truth) gives us an insight on how our Random Forest could have carried out its decision process. Such a decision tree is shown in figure 52, with its maximum depth limited to 4 for visualization reasons. The first split is supposed to be made based on album_type, then on danceability or energy. Further down in the three we also find speechiness, acousticness, listens_track, listens_album and artist_pop, all of them being features recognized as relevant to the classification in the permutation importance plot shown in figure 24.

Another way to approach global explanability is to employ the game-theory based algorithm SHAP, which we are using through its Python implementation found in the library of the same name. Figure 51 shows the summary plot given by SHAP for our black box, containing the following information:

- Feature names on the left;
- Features are vertically ordered in decreasing order by their importance for classification according to SHAP;

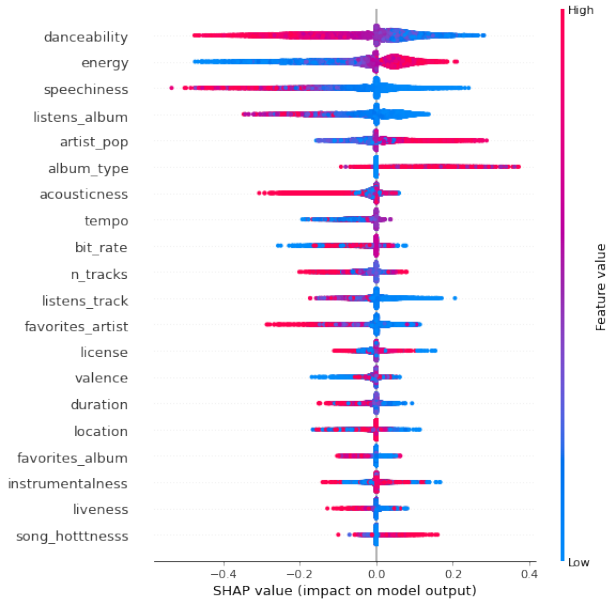

Figure 51: SHAP global explainer for our Random Forest model

- Each horizontal line is made up by all the points in the training data;
- The horizontal position of a point in the row shows whether the effect of the value of that point for that feature is associated with a positive (right) or negative (left) prediction;
- The color of a point shows whether it has high (red) or low (blue) values for that feature (color scale shown on the right, *Feature Value*).

For example, a high value for *danceability* (red points) brings the classification outcome closer to 0, i.e. "Non-Rock" (red points lie on the left side of the row) and viceversa, while the opposite happens with *energy*.

8.2 Local Interpretation

In order to explore two methods for local interpretation, we randomly select two records from our dataset which have been correctly identified by the black box model, one belonging to the positive class (Rock genre), while the other to the negative class

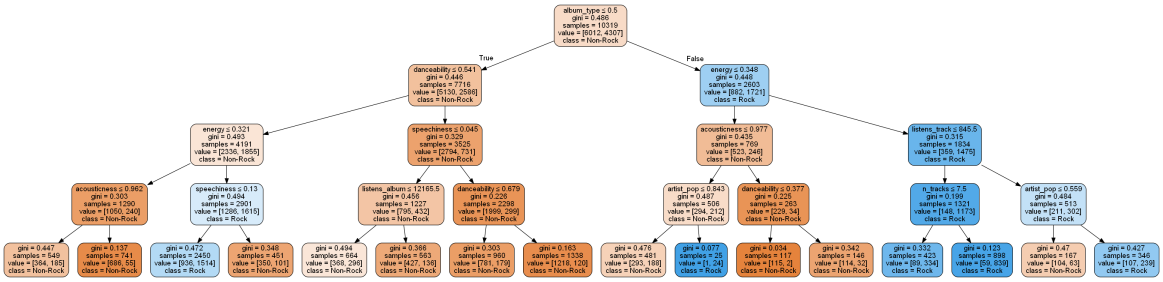

Figure 52: Decision Tree global explainer for our Random Forest model

(Other Genres). Both explainers will be tested on both of these songs.

8.2.1 LIME

The first method analyzed is the model-agnostic explainer LIME, in its version created for dealing with tabular data found in the `lime` library[5]: the results are shown in figures 53 and 54. The bars on the left show how probable is that the black model will predict a certain class for the given record. The plot in the middle shows how much each feature influences the classification and towards which class (only the top-9 relevant features are shown for ease of visualization), while the table on the right contains the values of each variable in the record taken into consideration. LIME provides a clear and intuitive way to visualize why a certain records has been classified in a certain way by the Random Forest. A downside of this implementation of the algorithm can however be seen in figure 53, where LIME assesses (correctly) that the black box will predict *Rock*, however looking at the feature weights we can see that the sum of the features pulling towards the negative class is clearly the higher of the two. This problem has already been noted in [3], where the authors point out that the displayed prediction probabilities must be assumed not be the sum of the feature probabilities, but rather the result of another calculation not obvious to a user. Searching in the library documentation, its creator explains for categorical features that "*since we perturb each categorical feature drawing samples according to the original training distribution, the way to interpret this is: if [a particular feature] was not [of the value it is], on average, this prediction would be [the amount shown in the middle graph] less [the class that value tends to]*"[6]. We feel like showing in a clear way the correlation between prediction probabilities and explanation values could be a big improvement in the readability of this implementation, especially since explanations are often needed to be shown to people with a less technical background than that of a data scientist (e.g. management team in a company), and these apparent inconsistencies could generate some confusion.

8.2.2 SHAP

Figures 55 and 56 show another local explanation attempt on the two selected records, this time carried out through the SHAP algorithm we already employed for global interpretation. In the graph we can see:

- An output value f(x), in bold: it is the probability with which the black box will predict the positive class. If the output is higher than 0.5, SHAP will say that the Random Forest would

[5] https://github.com/marcotcr/lime

[6] https://marcotcr.github.io/lime/tutorials/Tutorial%20-%20continuous%20and%20categorical%20features.html

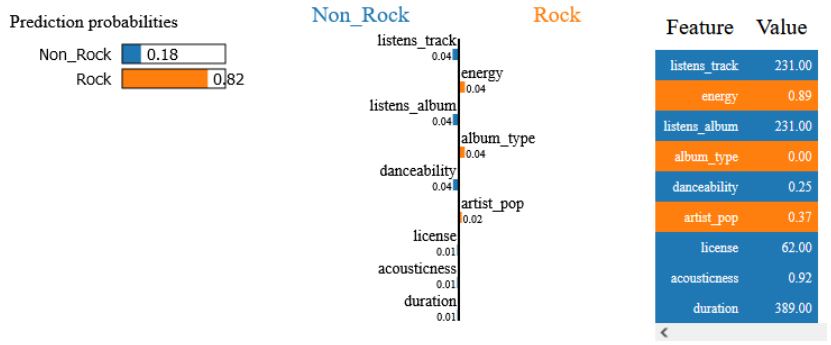

Figure 53: LIME local explainer for a Rock record on our Random Forest model

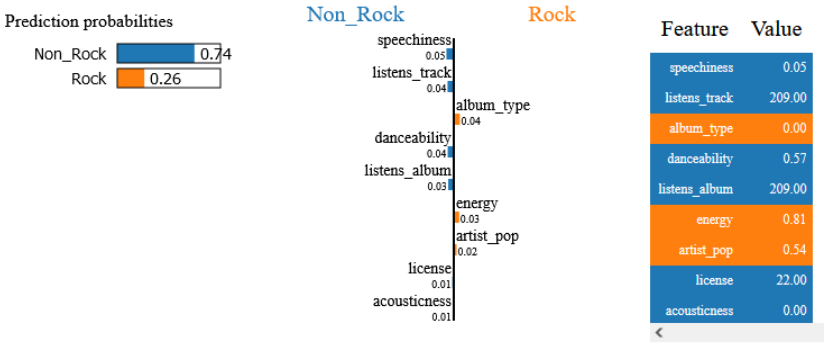

Figure 54: LIME local explainer for a NON-Rock record on our Random Forest model

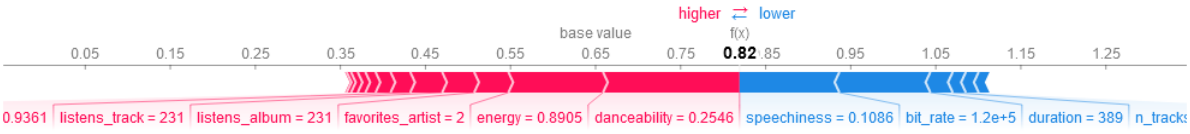

Figure 55: SHAP local explainer for a Rock record on our Random Forest model

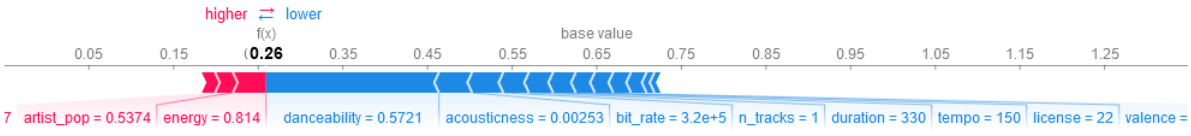

Figure 56: SHAP local explainer for a NON-Rock record on our Random Forest model

probably predict class 1 (Rock), class 0 (non-Rock) otherwise.

- A base value, in our case 0.65. In the original paper for SHAP the authors explain that it is "the value that would be predicted if we did not know any features for the current output", namely the mean prediction given by the black box;
- Each segment on the coloured line represents a feature: the bottom row shows the values assumed on the examined record by each relevant attribute;
- The features in red (to the left) are those that push the prediction towards the positive class (1), that is towards the right, hence the arrows along the lines. Features in blue, on the other hand, push the classification towards the negative class (0).
- The length of the segment corresponds to the importance of the feature value for that particular record, i.e. how much that value on that feature pushes the classification towards a class.

We conclude with two considerations about SHAP and its comparison with LIME. The first one is again about their graphical representations: while the way SHAP visualizes the explation is certainly less intuitive at a glance with respect to LIME's, here the output value has a visually appreciable correlation with the feature weights, thanks to the more geometrical representation. Looking at figure 55, for example, we can see that the sum of the red segments is 0.82-0.36 = 0.46, while the sum of the blue segments is roughly 1.11-0.82 = 0.29. Adding the red sum to the base value and subtracting the blue sum, we get 0.65+0.46-0.29 which is exactly equal to 0.82, the prediction probability given as output.

Secondly, we notice how the SHAP output values match the prediction probabilities given by LIME, even though the two algorithms work in different ways: LIME performs a logistic regression in the neighborhood of the point to locally approximate the behaviour of the black box, while SHAP creates coalitions of variables with a fixed value and analyzes the contribution of a given value for a feature when added to the coalition, taking inspiration from some game theory concepts. Indeed, we can see that the two algorithms can give opposite meanings to the same value for a given feature. Consider for example the variable *acousticness* and the record correctly classified as Rock: in figure 53 LIME represents it as pulling towards the negative class, while in figure 55 SHAP shows it as the one pushing towards the positive class the most.

References

[1] Michaël Defferrard, Kirell Benzi, Pierre Vandergheynst, and Xavier Bresson. Fma: A dataset for music analysis, 2016. arXiv preprint arXiv:1612.01840.

[2] Gianluca Malato. Are you still using 0.5 as a threshold?, 06 2021.

[3] Jürgen Dieber and Sabrina Kirrane. Why model why? assessing the strengths and limitations of lime, 11 2020.