FMA-MUSIC ANALYSIS
ADVANCED TOPICS AND APPLICATIONS

Authors

Gaetano Antonicchio, Samuele Cucchi, Francesco Gemignani

DATA MINING PROJECT

Data Science & Business Informatics

Academic Year 2020/2021

Contents

Introduction

In this paper we propose a detailed analysis of the FMA-Archive-Music, which counts for 106,574 tracks from 16,341 artists and 161 genres. The study will be focused on imbalanced learning, anomaly detection, advanced classification methods, time series analysis and sequential pattern mining. For the proposed applications we experimented with many tasks ranging from binary and multi genre recognition, song popularity detection, years classification and last but not least songs'emotion recognition. However due to pages constraint, we decided to include in details only few of those methods, while the others are described in a more general way. Since we had a wide range of inter-changeable features we could employ (i.e. *echonest.csv* and *features.csv*), each method described the following section, was tested on both datasets, and only the one with the highest (most interesting) performance was included in the report. The data are provided by the FMA-Archive, which is partitioned into 4 major datasets: *tracks.csv, genres.csv, features.csv* and *echonest.csv*.

Chapter 1

Data Understanding

1.1 tracks.csv & genres.csv

The dataset "tracks" contains 106,574 tracks and 52 features while "genres" has 163 rows and 5 features. Tracks is partitioned in the following way:

- **Track**: stores information like listens, bit rate, duration, name of the song.
- **Artist**: contains information about the name of the artist, location and members of the band.
- **Album**: provides specific information about each single album (i.e. years created, name).

The dataset collects songs from 2008 to 2018. In Figure 1.1 we show the number of tracks and album published along the years.

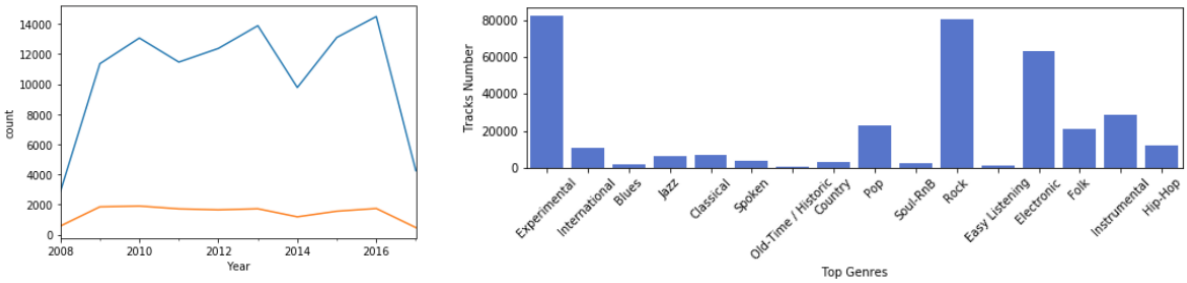


Figure 1.1: On the left we plot the number of tracks (blue line) and of album (orange line per year. On the right we show the number of tracks available per (top) genre.

Some features have less than 1% of available data, therefore they were discarded. The most significant ones did not contain missing values, except for *"genre_top"* which had 46.54% of available data. In order to replace the missing values we first tried to exploit the information provided in the dataset *"genre.csv"*. By doing so, we discovered that songs with missing *"genre_top"* label, were those for which a track was labelled with more than one genre. Some tracks were assigned more than 10 different genres, therefore it

was difficult to identify the most appropriate one. Since the 46.54% of *"genre_top"* provided us a sufficient amount of data to carry out the analysis, we decided to disregard the tracks for which the *"genre_top"* label was *Null*.

We noticed that the bit rate was expressed in bits per second. Since this value was considerably high w.r.t. to the other data, we re-scaled it in kilo-bit per second (scaled down by a factor of 1000). Majority of the tracks have a bit rate between 250 and 350 kbps.

On average tracks have a duration of 277 seconds. The distribution is slightly left skewed with a std of 305.51. This is a clear signal that our data might contained outliers, as some of the data have a duration of almost 1.5 std greater than the mean. Analysing instead the distribution of the attribute *"listens"*, we observed that it is highly left-skewed with less than 1,000 tracks having more than 2,000 listens. On average a track has 2,329 listens, even though this values is highly influenced by few "potential outliers" having around 5000 listens per track.

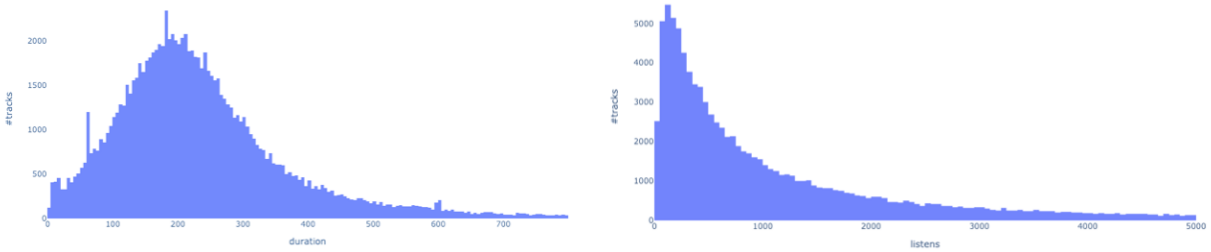


Figure 1.2: On the left we plot the distribution of the *"duration"* of a song. On the right we show the number of listens per track.

Experimental, Rock and Electronic are the genres for which there are more artists (2,609, 2,065 and 2,050 respectively). These 3, have also the highest number of tracks in the dataset (as shown in Figure 1.2). Regarding the geographical origin of the songs, we exploited the latitude and longitude provided in echonest.csv to discovered that most of them are American and European, with very few songs produced in Asia. The dataset includes songs belonging to 163 genres, which can be grouped in 16 top genres. For the analysis we decided to work only with a subset of those.

1.2 features.csv

The dataset consists of 106,574 tracks and 518 features. The dataset appears clean, as there are no missing values. However, we detected 2,111 duplicates which were properly removed. The attributes provided in this partition, were extracted using the Librosa library, which translated the spectrograms of each song into continuous values. The main features extracted are:

- **mfcc**: these features are useful for recognizing the timbre of a song.
- **chroma**: are able to capture armonic and melodic characteristics of a song.
- **tonnetz**: represents the tonal space.
- **spectral** (roll-off, contrast, centroid, bandwidth): describes the frequency and the energy of a song.
- **zrc**: rate at which a signal changes from positive to zero to negative or vice-versa. It is useful for recognising percussive sounds.

1.3 echonest.csv

The dataset contains 13,129 rows and 249 features. It is partitioned into *"echonest_audiofeatures"* (*acousticness, danceability, energy, instrumentalness, liveness, speechiness, tempo, valence*), *"metadata"* (general information about a song, like album name etc.) and *"temporal features"*. The features provided in the first partition have a values in range [0, 1]. To better understand the characteristics of each genre,

we plotted a radar chart, which highlights their differences. Below we show only few of the visualizations that were made.

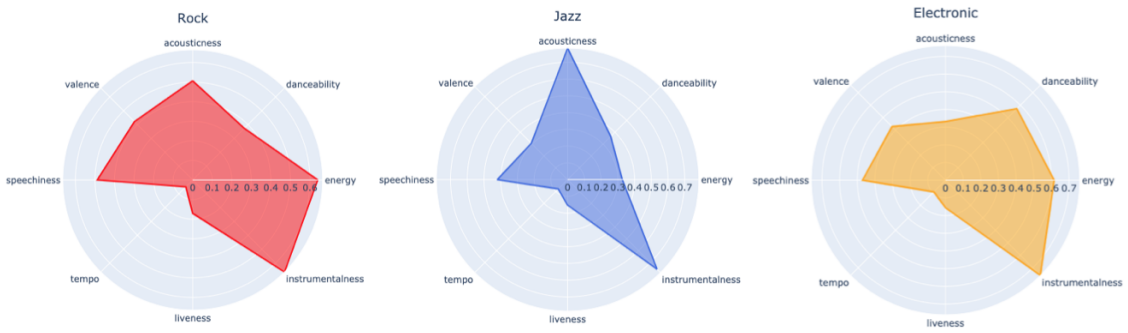


Figure 1.3: Radar chart of echonest audio-features for Rock, Jazz and Electronic.

We also researched the meaning of each of those features, and useful information that helped us in defining the tasks. For example *"liveness"* indicated whether a song was recorded during a live concert or in a studio, "valence" referred to the happiness of a song, and *"instrumentalness"* indicated the ratio between words in a song and instruments played. Analyzing the values of the latter, we trivially discovered some potential outliers (which will be detected in the following sections). In fact some tracks had an instrumentalness very close to 0 (meaning the tracks was composed of just words) and in combination with the information provided by the attribute "duration", we assessed that those tracks were not "real songs", but merely intros of an album, as their duration did not exceed 18 seconds.
The partition *"social features"* provided us information extracted from statistics about song hotness, artist familiarity and others. These were used to create the task of "Song popularity classification", which will be briefly described in the next sections.

Chapter 2

Imbalanced Learning and Anomaly Detection

2.1 Unbalanced Classification

2.1.1 Binary genre classification: Rock - Jazz

For this task we used the following features(*acousticness, danceability, energy, instrumentalness, liveness, speechiness, tempo, valence, duration and bit_rate*). The dataset has 4,133 tracks. The class distribution is very unbalanced, with the minority class covering only 5.83% of the data. We evaluated the performance of the classifier on the validation set using first all features and then techniques for features compression/reduction such as PCA, and a sequential feature elimination. In the next section we will include only the best results.

Decision Tree

We encoded the labels Rock and Jazz in binary values and we split the data into training set (70%) and test set (30%). We hyper-tuned the parameters of the model by testing 300 models obtained through a randomized grid search in a combination with a 5 fold cross validation. The model with the highest performance was the one without feature reduction. We noticed that PCA decreased substantially the

performance of the class Jazz (f1-score = 0.18, recall = 0.11) except for the precision which was higher (0.58) compared to model exploiting all features. In Table 2.1 we show the classification report. We see how even though the accuracy is 92%, given the high unbalancing among classes, this estimate is not a good indicator of the real performance of the classifier, hence we focused exclusively on the f1 score of the minority class (which we tried to improve after detecting anomalies and balancing the data). The feature importance of the model instead, revealed that the most import feature is "*energy*". The result is coherent with what are the main differences between genres, being Rock more energetic than Jazz on a musical level.

Class	Precision	Recall	f1 score	Support
Jazz	0.35	0.24	0.28	72
Rock	0.95	0.97	0.96	1168

Figure 2.1: Classification Report Unbalanced Decision Tree. The tree was constructed with criterion= entropy,max_depth=6, min_samples_leaf= 5 and min_samples_split=10.
Accuracy = 92%.

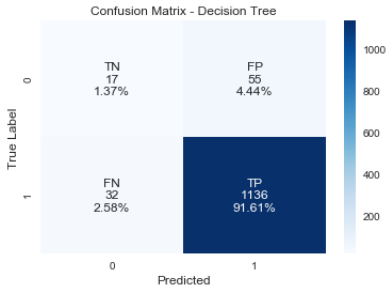


Figure 2.2: Confusion Matrix Decision Tree. AUC score is 78.8%

K-Nearest Neighbor

The best KNN model, was constructed with PCA features reduction (7 principal components explaining 90% of the variance were selected). The attributes were normalized before being given in input to the PCA and ultimately to the model. The optimal number of neighbors is 8, while the metric used is the Manhattan distance. In Figure 2.3 and 2.4 we show the results. We noticed that the model constructed without PCA, had a lower f1 score (0.27) and recall (0.18) for the minority class. Overall, reducing the features improved the ability of KNN of recognizing Jazz songs, even though in terms of recall, the performance are far worst than those of the decision tree.

Class	Precision	Recall	f1 score	Support
Jazz	0.57	0.18	0.27	72
Rock	0.95	0.99	0.97	1168

Figure 2.3: Classification Report Unbalanced KNN (PCA)
.
Accuracy = 94%.

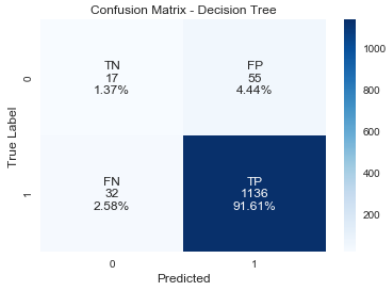


Figure 2.4: Confusion Matrix KNN (PCA). AUC score is 78.1%.

2.1.2 Additional classification tasks

Song popularity classification

We constructed a model able to predict whether a song is popular or not. The labels were generated using the information provided by the features *"song_hotness"* and *"artist_hotness"*. Our reasoning is based on the fact that songs sung by known artists that hit the top of the chart, should be considered popular. Songs for which these two features were respectively > 0.15 and > 0.4 were labeled as popular. The thresholds were decided after inspecting the mean and the scatter-plot of the above mentioned features. Using the same dataset employed for for Decision Tree and KNN, we obtained 12,399 not popular (94.44%)

and 730 popular songs (5.56%). The task resulted very difficult to be solved. In fact the recall for class popular was extremely low (as shown in Table 2.1)

Class	Precision	Recall	f1 score	Support
Not Popular	0.10	0.02	0.03	219
Popular	0.94	0.99	0.97	3720

Table 2.1: Classification Report Decision Tree - Song popularity

Multi-genre classification & Years of Rock

We constructed a Decision Tree and a KNN to solve also these additional tasks. For the task Years of Rock we built a model able to predict the year of a given Rock song, while the multi-genre classificator, is able to classify tracks into 8 top genres ("Rock", "Pop", "Classical", "Hip-Hop", "Jazz", "Electronic", "Historical" and "Folk"). The multi-genre classifier performed discretely, even though classes such as Pop and Jazz (which were the ones with less data) were not detected by neither Decision Tree and KNN (the recall was approximately 0.01). The classifier for classifying the year of Rock songs had good performances in terms of recall. For pages constraint the results will not be discussed in this report at a deeper level.

2.2 Anomaly Detection

In this section we employed 7 different outlier detection algorithms. Since we didn't want to rely on just one method, we constructed an ensemble which provided us a more accurate outlier labelling (using majority voting). The analysis presented is conducted on the dataset used for the classification of Jazz and Rock (10 numerical features: *Acusticness, Danceability, Energy, Instrumentalness, Liveness, Speechiness, Valence, Tempo* and *Duration*) . Although we experimented also with other datasets and tasks (i.e. multi-genre classification and songs' emotion recognition) we decided to show the results of the classifier analysed in the previous section, as we wanted to improve its performance. The outlier detection algorithms were selected after carefully analysing the distribution of our data, as we didn't want to use an algorithm that wouldn't be suitable for the task at hand. The methods adopted for detecting the outlierness in the data, are: **DBSCAN, LOF, KNN, ABOD, AutoEncoder, Isolation Forest** and **Extended Isolation Forest**. The results of the last two were compared, because we wanted to observe if there were any differences in the scoring. Considering that the scores were very similar, in a majority voting system we decided that employing both of them would have been redundant, hence among the 2 we selected the Extended Isolation Forest. We fist analysed each feature individually using box-plots and then we calculated the **IQR** scores.
In Figure 2.5 we can observe that outliers are found in *danceability, liveness, speechiness, tempo, duration* and *bit rate*. We noticed that few tracks have a duration that exceeds 3,033 sec. This value is extremely high compared to the mean duration of Rock and Jazz songs, which is respectively 237 sec and 379 sec. One explanation is that some tracks were recorded during live concerts.
We also noticed few abnormal values of bit rate. After some research we discovered that music extracted from video has higher bit rate compared to mp3 tracks, hence we deduced that some of the tracks were extracted from video-recording. In Table 2.2 instead, we show the results of the IQR method. The IQR highlighted more outliers than the amount we wanted to remove. Furthermore, the IQR considers each feature individually, hence it cannot judge the outlierness of a record in the full context.

2.2.1 Methodology and Results

Before removing the outliers, our dataset consisted of 4,133 records and 10 attributes. Considering that some of the algorithms adopts metrics meaningless in high dimensions, we tested their performance on both PCA projection and fully-dimensional dataset. To our surprise methods such as DBSCAN and KNN

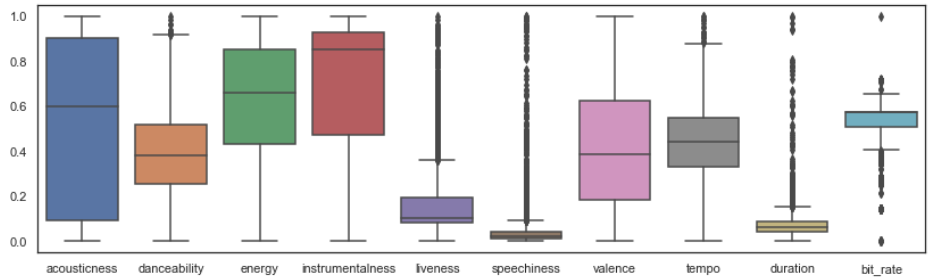


Figure 2.5: Normalized box-plots for outliers' visualization

Attributes	**Q1**	**Q2**	**IQR**	**Lower Whisker**	**Upper Whisker**	**Potential Outliers**
danceability	0.271	0.501	0.229	-0.072	0.845	11
liveness	0.104	0.209	0.104	-0.053	0.366	413
speechiness	0.035	0.065	0.029	-0.008	0.109	409
tempo	101.7	150.2	48.5	29.03	223.04	21
bit_rate	226	256	30	181	301	1182
duration	147	280	133	-52.5	479.5	226

Table 2.2: IQR scores per features

detected almost the same records regardless of the dimension's size. We concluded that 10 features didn't impact much on the goodness of the estimation, hence we performed the whole analysis without reducing the features.

After inspecting the distribution of our data, we focused on identifying the top 1% outliers. In order to do so, we used a contamination parameter to instruct the algorithms in identifying only the specified portion of anomalies. However, other methods such as DBSCAN and KNN were not provided with this option, therefore an hyper-parameter tuning was needed until the right configuration was found. The right setting for DBSCAN was identified with epsilon = 3.0 and min_sample = 15. We noticed that as we reduced the radius, the majority of the data were incorrectly labeled as outliers. KNN instead, performed best with the number of neighbors set to 50.

Majority of the samples labeled as outlier by LOF were perfectly matching those detected by the previous two methods. This was interpreted as an indicator that the chosen settings were optimal. ABOD however, was the only algorithm which labeled as inlier some records which were considered as outlier by the majority. We also tested the Isolation Forest (comparing the versions of sklearn and eif repository) and the Extended Isolation Forest. The forest was built using 300 classifiers with a sample size of 256. These algorithms, didn't provide an outlier label. Hence, we had to fix a threshold in order to label the data. After inspecting the histogram of the outlier scores, we decided to set the threshold to 0.60. The cut-off choice was crucial, as the majority of the data was attributed a score between 0.45 and 0.55, and no records had a score higher than 0.70. Using a threshold equal to 0.50 would have significantly increased the number of detection. Even with this threshold, compared to others, these approaches doubled the number of outliers detected. We noticed that Extended and Standard Isolation Forest performed similarly. We also experimented with AutoEncoders for detecting anomalies. We opted for an under-complete AutoEncoder with a bottle-neck of 3 neurons. The model was trained using the ReLU activtion function over 400 epochs. Finally, the outcome of model, was in line with the detections made by the previously described methods.

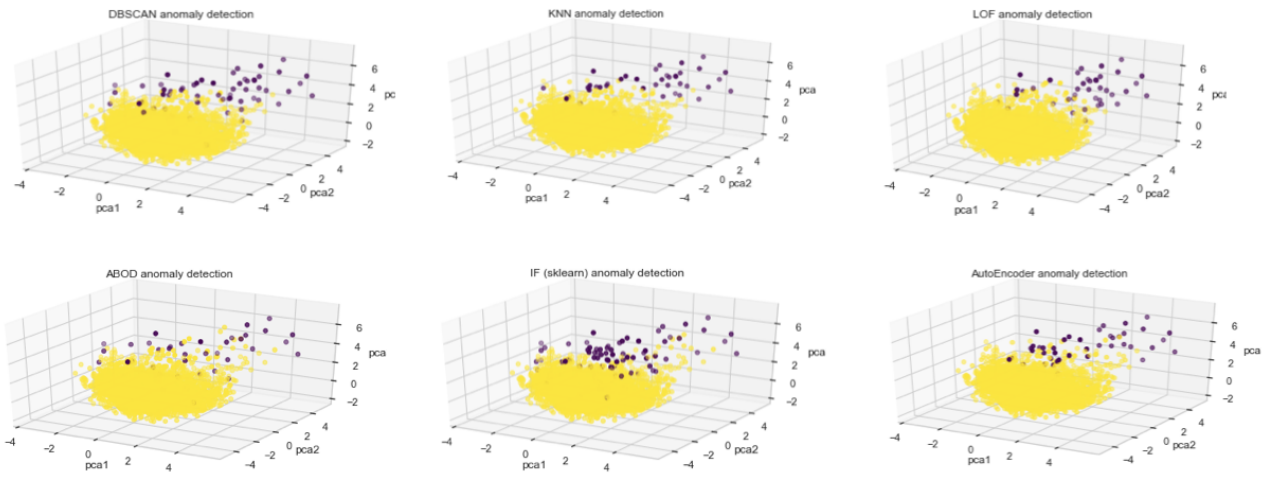


Figure 2.6: Scatter plots of anomaly detection methods. From left to right: DBSCAN, KNN, LOF, ABOD, Isolation Forest and AutoEncoder. In yellow are represented inliers, whereas in purple are indicated the outliers. The PCA representation is used for plotting the data in 3 dimensions. The anomalies represent the top 1% outliers.

	DBSCAN	KNN	LOF	IF	EIF	ABOD	AutoEncoder
outliers	62	40	42	62	61	41	42

Table 2.3: Number of outliers detected by each method.

2.2.2 Majority Voting

The final outcome of the anomaly detection was obtained through a majority voting schema. After evaluating the proposed methods, we assigned a label to each row of the dataset, indicating if the track was an outlier or not. The ensemble of anomaly detection methods, assigned the outlier label to a track, only if the majority of its constituent agreed with that assignment. This approach identified 38 outliers in total. After inspecting the scatter-plots of each pairs of attributes, we noticed that visible outliers were correctly detected (for some combination of features), as it can be observed in Figure 2.7. It is clear that the attribute *"duration"* is the one that in combination with other features, generated isolated points. In Figure 2.8 instead, we show a 3D representation of the data, with the labelling assigned by the ensemble of outliers algorithms.

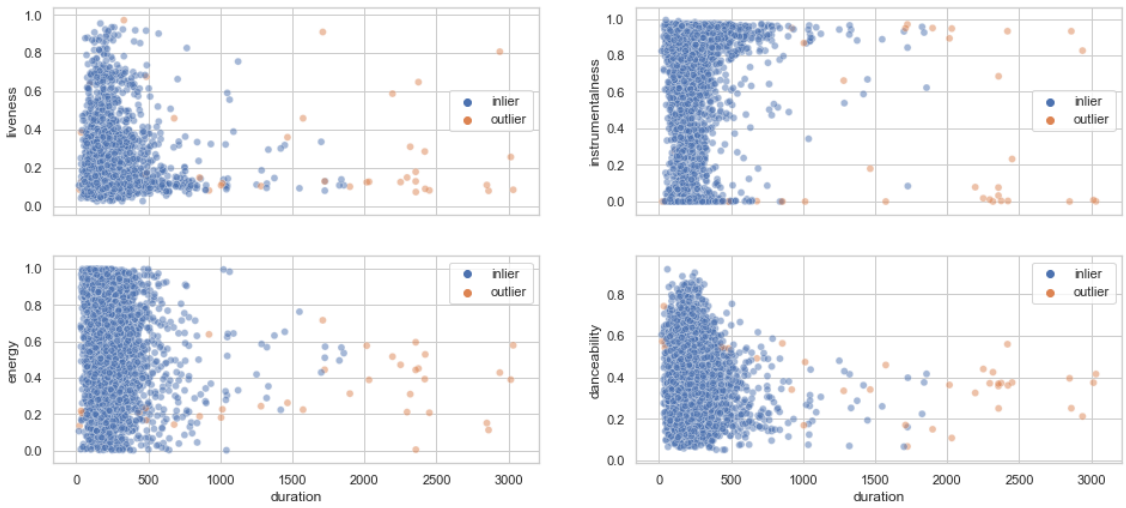


Figure 2.7: Top 1% anomalies on 4 pairs of attributes, with majority voting.

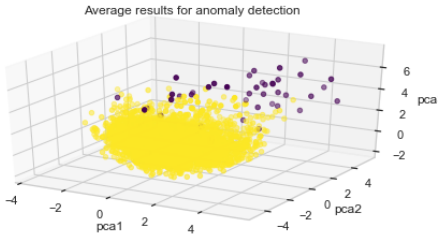


Figure 2.8: Scatter plot of Top 1% outliers detected using majority voting.

2.3 Imbalanced Learning

Binary genre classification: Rock - Jazz

In this section we applied several balancing technique on the dataset used for classifying Jazz and Rock songs. The data under analysis do not contains outliers (as they were removed in the previous section). We trained a Decision tree and KNN classifier, with the aim of improving the f1 score of the minority class (Jazz). We focused on this metric as we wanted to find a good trade-off between precision and recall, and make our model able to learn and classify correctly Jazz songs, without penalizing the performance on class Rock. The dataset is composed by 4,095 tracks and 10 attributes. Below we show the data distribution and unbalancing percentages.

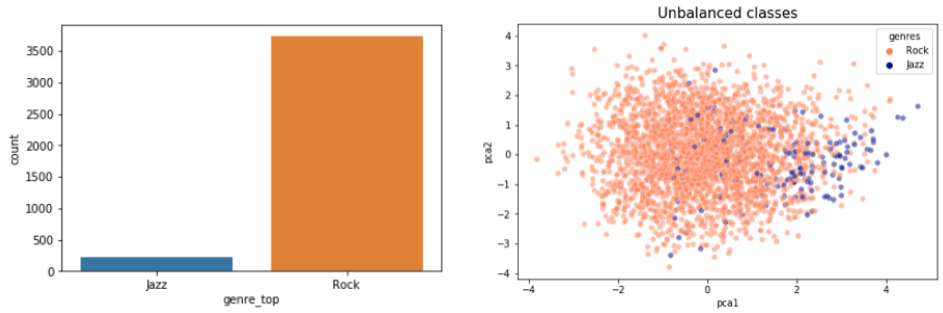


Figure 2.9: Count-plot and scatter-plot of the class distribution (94.33% Rock - 5.67% Jazz).

We experimented with the following algorithms: **Random Undersampling**, **Condensed-NN**, **Tomek's Link**, **Random Oversampling**, **SMOTE** and **ADASYN**. The SMOTE algorithm was also compared with the **K-Means SMOTE**. The latter improves the performance of the standard SMOTE algorithm. This is due to the fact that K-Means SMOTE groups the data into micro-clusters before generating synthetic data. This avoids that data are over-sampled in area in which there are noise points. In Figure 2.10 we show the differences between the two methods. We can observe that K-Means SMOTE is more efficient than SMOTE in identifying the area in which the minority class in located (and more concentrated). It is important to mention that all balancing approaches, were applied exclusively on the training data. In addition to that, since we didn't want to bias our classifiers with the hyper-parameters discovered in an unbalanced setting, we performed a random search to fine-tune the Decision trees and KNNs.

Decision Tree

The model with the highest performance in terms of f1 score is K-Means SMOTE, whose score is 40% (+12% w.r.t. the unbalanced dataset). We noticed that this algorithm, improved also the precision of class Jazz(45%). Instead, if we focus exclusively on the recall, we can see that random undersampling identifies 90% of the Jazz tracks on the test set. However the high recall, it's paid at the expense of an high false negative rate. In fact, the precision of class Jazz is decreased to 13% as well as the recall of class Rock which is 63% (was 98% on the unbalanced dataset). In Table 2.4 we compare the classification report of K-Means SMOTE and the decision tree with unbalanced classes, while in Figure 2.11 we summarize

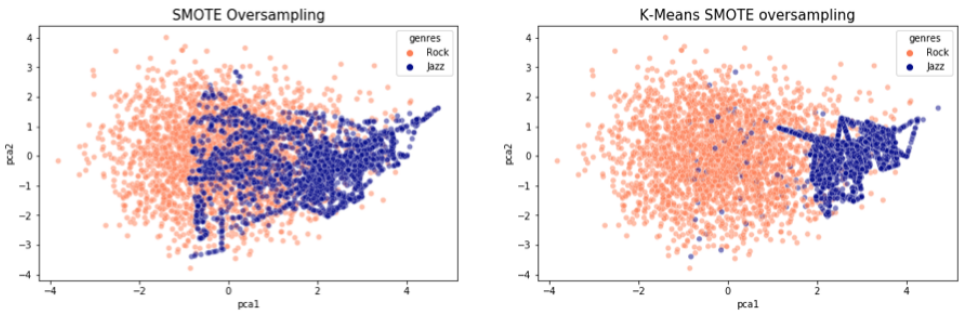


Figure 2.10: SMOTE and K-Means SMOTE oversampling.

the scores obtained by each balancing technique on the minority class Jazz.

	Unbalanced			K-Means SMOTE			
Class	**Precision**	**Recall**	**f1 score**	**Precision**	**Recall**	**f1 score**	**Support**
Jazz	0.39	0.21	0.28	0.45	0.36	0.40	70
Rock	0.95	0.98	0.97	0.96	0.97	0.97	1159

Table 2.4: Classification Report: Unbalanced and K-Means SMOTE. The Unbalanced decision tree has depth = 8 using Gini index for splitting nodes, while the balanced decision tree has depth = 9 and utilized the Entropy.

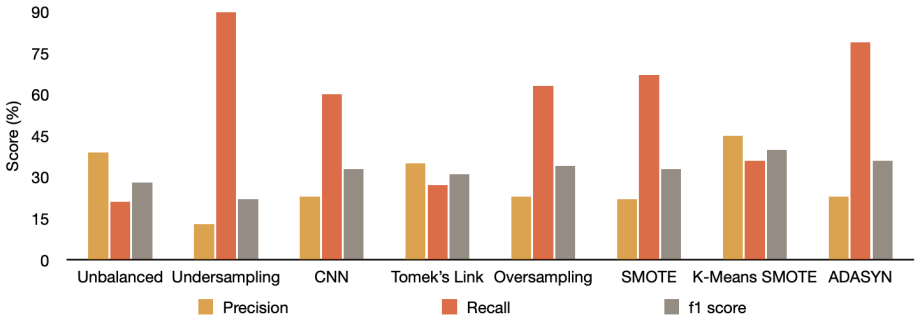


Figure 2.11: Decision Tree scores for each imbalanced learning method. The metrics refers to the minority class Jazz.

K-Nearest Neighbor

The highest f1 score is provided by undersampling (0.46). Good performance were obtained also using CNN and Tomek's Link (f1 score is 0.40 and 0.38 respectively). In terms of recall of the minority class, the best balancing technique is still undersampling, which increased the recall from 0.27 to 0.73. Although the precision decreases from 0.53 to 0.34, we observe that it didn't affect the ability of the classifier to recognize Rock songs with high precision and recall (for these the precision increased of +0.07 while the recall decreased from 0.99 to 0.91). In terms of precision, CNN outperformed the others, with a score of 0.60. However when using this method, the recall of the minority class has only a slight improvement (+0.03). In conclusion, undersampling resulted to be the best balancing algorithm for classifying Rock and Jazz songs using KNN.

	Unbalanced			Random Undersampling			
Class	**Precision**	**Recall**	**f1 score**	**Precision**	**Recall**	**f1 score**	**Support**
Jazz	0.53	0.27	0.36	0.34	0.73	0.46	70
Rock	0.96	0.99	0.97	0.98	0.91	0.95	1159

Table 2.5: Classification Report KNN: Unbalanced and Random Undersampling

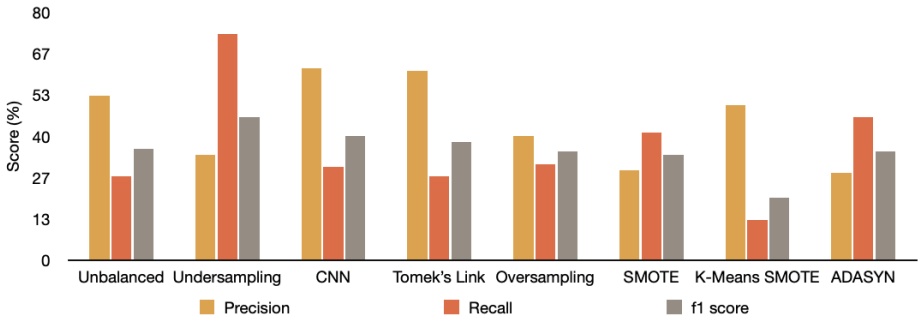


Figure 2.12: KNN scores for each imbalanced learning method. The metrics refers to the minority class Jazz

2.3.1 Comparing Decision Tree and KNN

Decision Tree: Although K-Means SMOTE has the highest f1-score, the best AUC score is obtained using ADAYSN (0.85). The unbalanced classifier has the lowest AUC (0.781). Furthermore, all balancing approaches increased this metric of at least +0.5, except for Random Oversampling which did not prove to be very effective provided the AUC similar to the unbalanced one.
KNN: The highest AUC score is obtained by the undersampling approach, which proved to be very effective for this task. Similar results are provided by Tomek's Link which has an AUC score of 0.888.
In conclusion for solving the binary classification task Rock-Jazz the KNN classifier with random undersampling provides higher performance compared to the other.

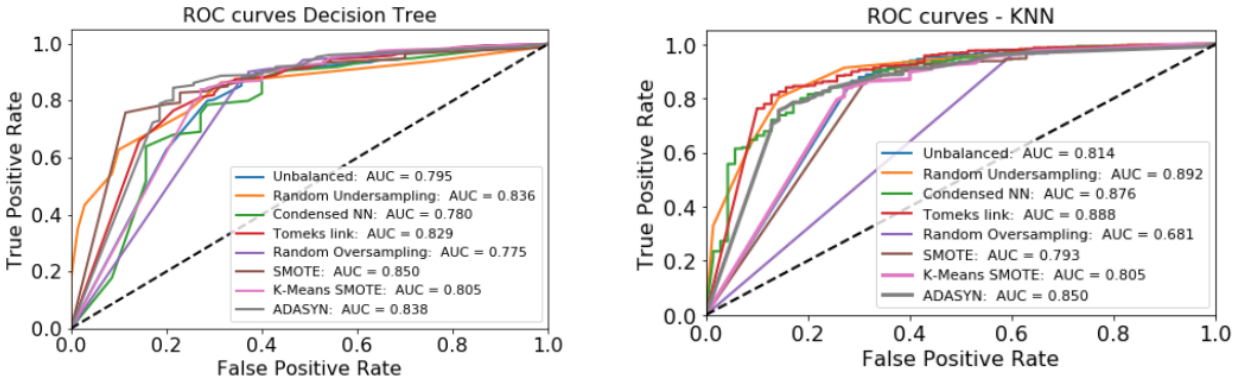


Figure 2.13: ROC curves of balancing techniques for Decision Tree and KNN.

2.3.2 Additional classification tasks

Song Popularity classification

The highly unbalanced task of song popularity classification, resulted to be quite difficult to be solved. The classes were extremely skewed with 12,399 not popular songs (94.44%) and 730 popular songs (5.56%). The minority class scores for the unbalanced Decision Tree classifier are: recall = 0.01, precision = 0.10 and f1-score = 0.02. In this task we focused exclusively on the recall, as we wanted our model to be able

to detect as many popular song as possible. The best method resulted to be SMOTE which boosted the recall up to 0.67.

Multi-genre classification

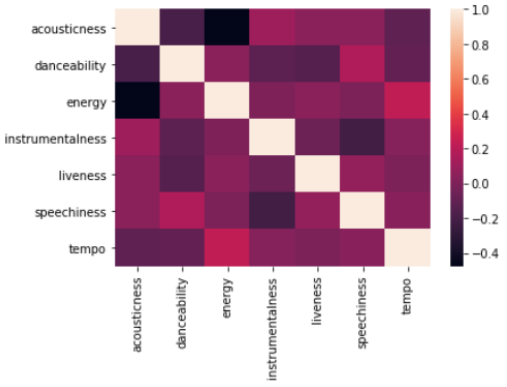
For this task we tried to improve the multi-genre classification of the following 8 genres: Rock, Pop, Classical, Jazz, Historical, Electronical, Hip-Hop and Folk. Genres like Pop and Jazz represented around 5% and 1% of the whole dataset, and for this reason the classifier was not able to detect them (in fact the recall was these classes did not exceed 0.05). Jazz and Pop had an f1 score of 0.08 and 0.04, which we were able to improved to 0.16 using SMOTE oversampling.

Chapter 3

Advanced Classification Methods

3.1 Naive Bayes Classifier

In this section we chose the task of classifying songs' emotion using the Naive Bayes classifier. The classifier solved a binary classification, through which a track was labeled either as "happy" or "sad". The labels were engineered starting from the attribute *"valence"* provided in *"echonest.csv"*. *"Valence"* is an attribute that has a range that goes from 0 to 1 (generated by Echo Nest using internal algorithms). The highest this value is, the more a given track transmits to the user positive vibrations. We set a threshold of 0.55, so that songs with a valence score lower than the threshold were labeled as sad songs while the others as happy. The classes are almost balanced, although the majority of tracks are sad songs. The partition is the following: 7,724 "sad" and 5,405 "happy" songs, for a total of 13,129 records. The features used for training the Naive Bayes classifier are: *acousticness, danceability, energy, instrumentalness, liveness, speechiness, tempo*. Their choice is driven by the fact that they shown low correlation(Figure 3.1), which is essential for the independence assumption made by the Naive Bayes Classifier. We performed a binary label encoding and split the data into training (70%) and test set (30%). We didn't apply a normalization, as it doesn't affect the way probabilities are estimated.



Class	Precision	Recall	f1 score	Support
Happy	0.62	0.52	0.56	1622
Sad	0.70	0.77	0.73	2317

Figure 3.1: On the left the Correlation Matrix of features used in the Naive Bayes. On the right the Classification Report.

Overall the model performed discretely with an accuracy of 67%. It is interesting to see that the model had better performance in predicting the class "sad". Furthermore, the number of False positive is quite high. In fact the classifier labeled as "sad" almost half of the happy songs present in the dataset (which explains the recall of 0.52).

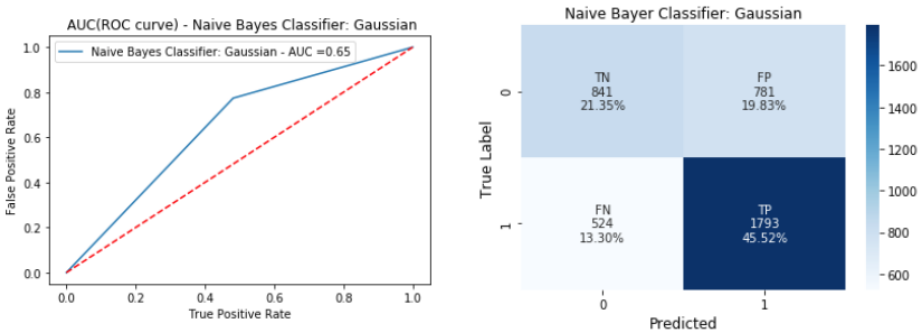


Figure 3.2: ROC curve and Confusion Matrix of Gaussian Naive Bayes.

3.2 Multi-Layer Perceptron

In this section we constructed a Multi-Layer Perceptron for solving the same task described in the previous section (songs' emotion recognition).

3.2.1 Data Preparation

The dataset chosen for this binary-task is *features.csv*, which counts of 13,129 input patterns and 518 features. As emotions are usually associated to the tonality and pitch of a song, we considered the attributes provided by *features.csv* more suitable than the ones of *echonest.csv* to train a Neural Network. We detached the labels from the dataset and we encoded them into binary values. Lastly, we normalized the dataset paying attention to don't bias the development set with information regarding the distribution of the test set.

3.2.2 Validation Schema & hyper-parameters tuning

The data was split into **development set** and **test set**, using a split ratio of 30%. The split was stratified, so to assure that the data in each partition represented the same proportion of the class. The test set was never used during the model selection and training phase, to assure an unbiased estimation. The development set was further split into **training set** and **internal test set**. The training set was used exclusively for model selection, whereas the internal test set was used to evaluate the performance and compare models. In Table 3.1 we report the data partitions.

Development set		
Training set	**Internal Test set**	**Test set**
6433	2757	3939

Table 3.1: Data partitions.

In order to find the right hyper-parameters' setting, we first tried different configuration so to get an initial idea of what architecture and parameters worked best for the task at hand. We then performed a coarse-to-fine random search using a **5 fold cross validation**.
The optimal configuration found by the coarse search, was given in input to a randomized grid search which tested random configuration of hyper-parameter within the range of the parameters discovered in the coarse phase. Different models were compared using the internal test set as internal performance estimator. In order to reduce the size of the hypothesis space, we used only the binary cross entropy for training the model. In Table 3.2 we show the coarse configurations used in the randomized grid search.

The best model resulted to be the one with 2 hidden layers of 60 and 20 neurons each. Below we show the configuration of the best model.

Hidden Layers	Learning rate	Solver	Momentum	lambda	Activations
(50,), (50,50), (50, 50, 50), (60,20), (20, 20, 20 ,20), (100,), (100, 20)	0.1, 1e-1, 1e-2, 1e-3, 1e-4	Adam, SGD	None, 0.2, 0.5, 0.8	None, 1e-1, 1e-2, 1e-3, 1e-4, 1e-5	Sigmoid, ReLU

Table 3.2: Hyper-parameters used in the corase randomized grid search. The lambda refers to the Tikhonov regularization coefficient. It's important to notice that the activation function of the output layer is not tested in the randomized grid search. Since we solved a binary classification task, the output activation function was set to be a Sigmoid.

Model's Architecture

Hidden Layers	Learning rate	Solver	Momentum	lambda	Activations
(60,20)	0.00112	SGD	0.7934	0.11744	ReLU

Table 3.3: Architecture of the best model

3.2.3 Analysing overfitting effects

When selecting the best model, we carefully inspected the loss curves of training and validation so to detect possible overfitting. We noticed that given the high number of features (518) and sub-sequentially the high number of free parameters, the model was prone to overfitting. However, the right regularization coefficient was helpful in controlling the complexity of the model and avoid overfitting. In Figure 3.3 we compare the models with and without regularization.

The loss curves for training and validation are stable and are not diverging for the model with regularization. This is a clear indicator that the model is not overfitting the data. The accuracy on the validation set is 77.53%, which is considered a good result considering the task we are attempting to solve. The model on the right instead (the one without regularization) shows a validation curve that starts to increase after 300 epochs, while the training curve decreases to 0. To get an estimate of the model performances on previously unseen data, we tested it on an internal test set. The accuracy on the final test was expected to be approximately 75.98%.

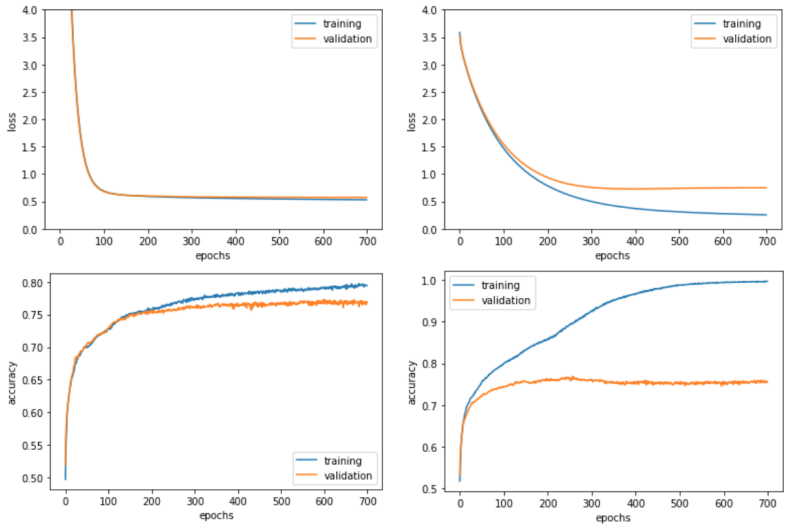


Figure 3.3: On the left we show the loss (binary cross-entropy) and accuracy of the model with regularization. On the right the loss (binary cross-entropy) and accuracy of the model without regularization.

3.2.4 Final Evaluation

The best model was retrained on the full development set (training set + internal test set) and then tested on the test set. Below we show the classification report along with the confusion matrix and ROC curve. The accuracy on the test set is 75.72% which is close to the one estimated on the validation set. Analyzing the intra-class scores, we observe that the precision recall and f1 score are high for both classes, although the model can detect sad songs with higher precision (80%). Overall the model was able to capture and learn the features needed for detecting emotions in a song.

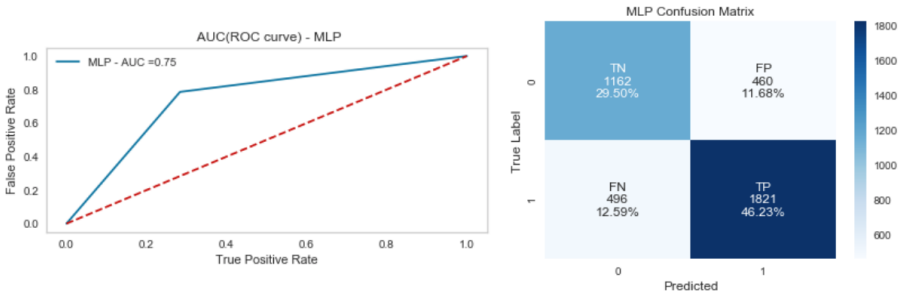


Figure 3.4: Confusion Matrix and ROC curve of MLP.

Class	Precision	Recall	F1	Support
Happy	0.70	0.72	0.71	1622
Sad	0.80	0.79	0.79	2317

Table 3.4: Classification report: Test set MLP.
Accuracy = 75.72%.

3.3 Support Vector Machines

In this section we experimented with linear and non-linear support vector machines for the task of Song emotion recognition. This binary classifier was able to detect sadness and happiness of a song. The attributes employed are the ones provided in *"features.csv"*.

3.3.1 Data Preparation & Validation Schema

The data was normalized using a standard scaler, while the labels were encoded into binary values. The data was partitioned in training set (70%) and test set (30%). The optimal hyper-parameters were obtained performing a random search in a combination with a 5 fold cross validation. We tested 300 configurations for each model (Linear and Non-Linear SVM), whose parameters were randomly generated from a uniform distribution.

3.3.2 Linear SVM

The optimal setting for the C parameter is 0.4329, which provided us an accuracy on the validation set of 74.61%. The performance on the test set instead was higher than our estimation. In fact the model reported an accuracy of 76%. The recall for both classes is quite balanced (78% sad and 75% happy). However the precision on the class sad songs is much higher than the one of happy songs (83% and 68% respectively). The AUC score instead, is approximately 76%.

Class	Precision	Recall	F1	Support
Happy	0.68	0.78	0.73	1622
Sad	0.83	0.75	0.79	2317

Table 3.5: Classification report: Test set Linear SVM.
Accuracy = 76%.

3.3.3 Non-Linear SVM

The SVM with non-linearity was tested using several kernels(RBF, polynomial, linear and Sigmoid) and C configurations. The best model resulted to be the one with the RBF kernel and a C = 1.419. It's important to mention that the performance with the other kernels were quite low and didn't reach more than 67% accuracy. With RBF kernel instead, we reached an accuracy on the test set of 77% (1% higher than the one obtained with linear SVM). The non-linearity introduced by the kernel, allowed us to improve the precision of the class Happy (which is now 74% against the 0.64% of linear SVM). The boost in performance is also noticeable in the f1 score of class sad songs (now 81%). Lastly, we didn't noticed substantial changes in the AUC score (which is still 76%).

Class	Precision	Recall	F1	Support
Happy	0.74	0.70	0.72	1622
Sad	0.80	0.82	0.81	2317

Table 3.6: Classification report: Test set Non-Linear SVM.
Accuracy = 77%.

3.3.4 Final Evaluation

Comparing the two models, we observe that apart from the improvement in the accuracy, the non-linear SVM increased the True Positive ratio(48.71% against 44.02%) and diminished the number of False Negatives down to 10.31%. Although similar, non-linear SVM resulted to be the best model.

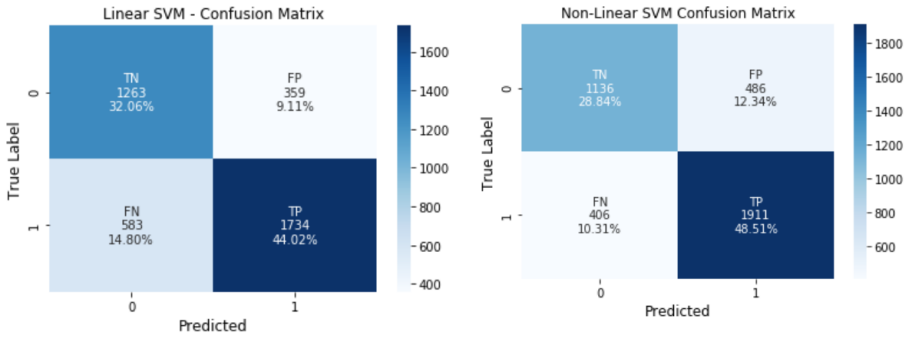


Figure 3.5: Confusion matrix for SVM with Hard Margin and SVM with Soft Margin.

3.4 Rule based classifiers: RIPPER algorithm

We employed the RIPPER rule-based classifier, for solving the binary classification task: Electronic - Classical. The task we proposed to solve was not of high complexity (the classes were well separated), as the two genres are noticeably different one from one another. However the dataset presents an high class imbalanced, which do not affect the RIPPER algorithm, as the latter is suitable for classifying instances of unbalanced datasets. The features used to train the models were extracted from the partition

of *"echonest audio_features"*(*acousticness, danceability, energy, instrumentalness, liveness, speechiness, tempo, duration* and *bit_rate*). Before growing the rules required for classifying the test-data, we removed the top 1% of outliers (using the techniques described in chapter 2) and made sure that the dataset didn't have missing or duplicate values. The dataset consisted of 2,288 tracks. Electronic songs represented 90.51% (2,071 rows) of the data, while Classical 9.48% (717 rows). The rule-set was grown by setting the majority class (Electronic) as default class and learn rules on the positive class (Classical). In order to maximize the results, we tested the performance of the model using different prune sizes. The latter ranged from 0.2 to 1. The highest performances were provided when the prune size was set to 0.5. As this increased the model was not able to discriminate the class labels. On this task, RIPPER performed exceptionally, obtaining an AUC score of 94% and a recall on the minority class of 90%.

Class	Precision	Recall	F1	Support
Electronic	0.99	0.98	0.98	622
Classical	0.81	0.90	0.85	65

Figure 3.6: Classification report: Test set RIPPER. Accuracy = 97%.

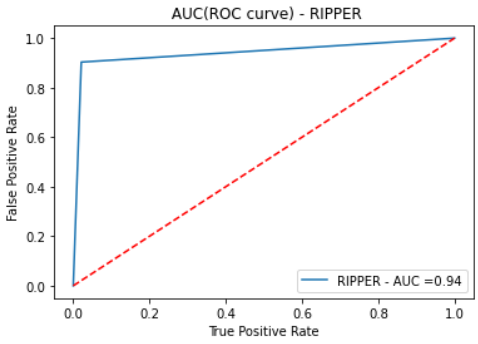


Figure 3.7: ROC curve - RIPPER

```
[[acousticness=0.99-1.0 ^ energy=0.0-0.08 ^ danceability=0.37-0.45] V
[acousticness=0.99-1.0 ^ energy=0.0-0.08] V
[energy=0.08-0.28 ^ acousticness=0.99-1.0 ^ danceability=0.26-0.37] V
[acousticness=0.9-0.99 ^ energy=0.0-0.08 ^ liveness=0.08-0.09] V
[acousticness=0.9-0.99 ^ energy=0.0-0.08 ^ liveness=0.11-0.12] V
[acousticness=0.9-0.99 ^ bit_rate=-0.0-160.0 ^ danceability=0.26-0.37] V
[acousticness=0.9-0.99 ^ energy=0.0-0.08 ^ instrumentalness=0.0-0.32 ^ tempo=26.13-82.65] V
[danceability=0.06-0.26 ^ energy=0.08-0.28 ^ liveness=0.35-0.94] V
[acousticness=0.99-1.0 ^ energy=0.08-0.28] V
[acousticness=0.9-0.99 ^ energy=0.0-0.08 ^ instrumentalness=0.69-0.8] V
[speechiness=0.04-0.05 ^ energy=0.0-0.08] V
[energy=0.0-0.08 ^ danceability=0.45-0.52]]
```

Figure 3.8: Ruleset extracted on the positive class. For better readability the symbol ^ can be read as **AND** while V as **OR**.

In Figure 3.8, we show the rules learned on the positive class. It is interesting to notice that the majority of the rules in the decision list, have conditions on the variables *"acousticness"* (present in almost 66% of the rules) and *"energy"*, whose appear to be determinant in classifying classical songs. Observing the rules, classical song has high acousticness(0.9 to 1) and low energy (usually in range 0 - 0.08).

3.5 Linear Regression

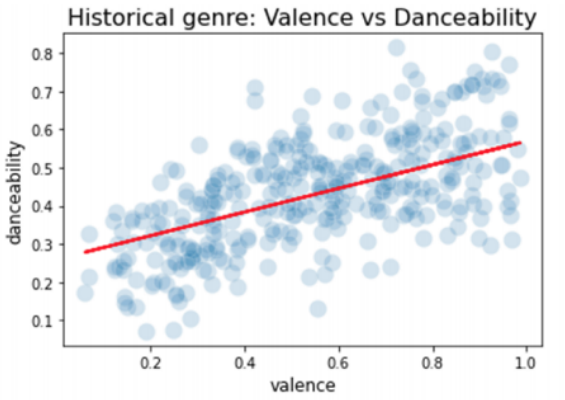
In this section we built a Linear Regression model able to predict the variable *"danceability"* using *"valance"* as independent variable. To achieve this, we filtered the dataset by considering only one genre at the time. The choice is driven by the fact that if we considered the genres all together, the linear correlations would have been hidden due to genres' diversity. The highest linear correlation between the dependent and independent variable was provided by the genre Historical and Jazz, for which the correlation is +0.61 and +0.5.

3.5.1 Data Preparation

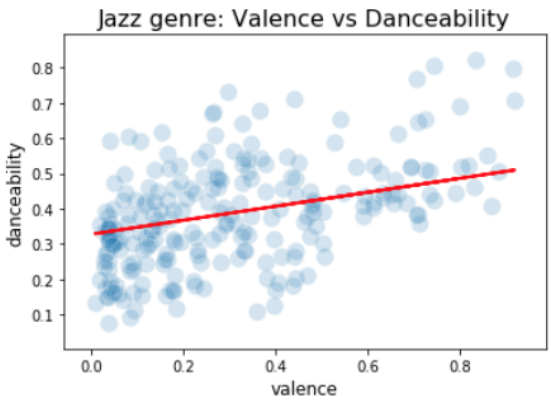
The dataset for Historical has 349 tracks, while the one for Jazz counts 233 records. Before constructing the model we removed 1% of the outliers, as they could have negatively influenced the computation of the hyper-parameters of the regression line. Ultimately, the datasets were split into training set (70%) and test set(30%). We also checked if introducing a regularization terms improved the fit of the regressor (we tested Ridge and Lasso regularizations). The parameter α (penalty term) was tuned using a randomized grid search and 5 fold cross validation (we tested values in range $[10^{-6}, 1]$).

3.5.2 Simple Linear Regression

The best fit was obtained on the dataset of Historical tracks without regularization. Analyzing the R^2 score, we could notice that the variable *"valence"* is able to predict almost 51% of the variance of the dependent variable *"danceability"*. The same score however, decreased when introducing a regularization. The model has a MSE of 0.011 on the test set (the MAE is 0.087). The model without regularization, has a better fit also on the Jazz dataset. The R^2 is 0.320, while the MSE is 0.017.



Model	Coefficient	Bias	Alpha	R2
No Reg	0.309739	0.259500	-	0.511
L1	0.208903	0.312832	0.005717	0.386
L2	0.288898	0.270523	0.998148	0.489



Model	Coefficients	Bias	Alpha	R2
No reg	0.26752015	0.30629223	-	0.320
L1	0.19853007	0.32712091	0.00393	0.262
L2	0.2595008	0.30871334	0.2928	0.314

Figure 3.9: From left to right we show the scatter-plots of the models with the best fit (on Historical and Jazz dataset). Below the graphs we show the results obtained on each model.

3.5.3 Multiple Linear Regression

We attempted to improve the performance of the linear regression model constructed for predicting the variable *"danceability"* of the "Historical" genre, by increasing the number of independent variables. The variables chosen for building a multiple linear regression model, were *"energy"* and *"speechiness"*. These additional variables are those which had the second and third highest correlation with the dependent variable (the others had negative or zero correlation). The best multiple liner regressor is the one without regularization, in fact the R^2 for this model is 0.509 (slightly lower than the one obtained with the linear regression). The MSE for the model without regularization is 0.011. The latter increases if we increment the variable "alpha", meaning the regularization is slightly causing our model to underfit the data. Overall, we noticed that w.r.t the linear regressor, the multiple linear regressor didn't improve the performance. This is probably due to the fact that the additional variables were not strongly correlated with the dependent one.

Model	Coefficients	Bias	Alpha	R2
No reg	0.31853349, -0.0521483 0.13026902	0.2462008	-	0.509
L1	0.29438168, 0, 0.10362199	0.25362256	0.0006984	0.496
L2	0.28854518, -0.01181835 0.1125352	0.25753402	0.99504	0.487

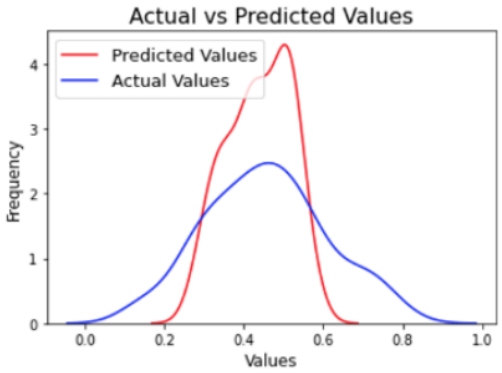


Figure 3.10: On the left we show a table summarizing the parameters of the model and the R^2 score. On the right we show the distribution of the predicted/actual values and their frequency. We can observe that using the regression line, we are predicting with more frequency values ranging approximately from 0.4 to 0.6.

3.6 Logistic Regression

In this section we employed a Logistic Regression to classify songs' emotion. In order to solve this binary task we chose the following features: *acousticness, danceability, energy, instrumentalness, liveness, speechiness, tempo, duration* and *bit_rate*.

3.6.1 Data Preparation & Analysis

The dataset containing 13,129 rows (7,724 sad and 5,405 happy songs) was split into training set(70%) and test set(%). The labels were encoded in binary variables and the features were normalized using a standard scaler. Before deploying the model, by means of a recursive feature selection, we checked if by removing attributes, the accuracy on the validation set increased. We noticed that the highest results were obtained when all features were utilized, hence we proceeded using all attributes available.

3.6.2 Model Evaluation

If we analyze the classification report we notice that even though the precision is lower for class "happy" (56% compared to 76% of class "sad"), the model is able to identify 71% of happy songs(as shown by the recall). Since the classes are not highly unbalanced, we considered the accuracy as a good performance estimator. The score obtained on the test set is 68% which is considerably low compared to the one obtained by the models described in the previous sections (for the same task). In Figure 3.12, we can observe that the strongest feature is *"danceability"*, which has an higher importance (+0.10) compared to the others.

Class	Precision	Recall	F1	Support
Happy	0.59	0.71	0.65	1622
Sad	0.76	0.66	0.71	2317

Figure 3.11: Classification Report Logistic Regression - Songs' emotion. Accuracy = 68%.

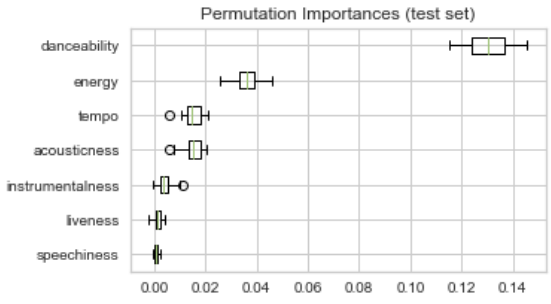


Figure 3.12: Features permutation of Logistic Regression run over 20 trials.

The confusion matrix highlights the fact that the False Positive and False Negative are quite high. In fact we noticed that 20.11% of "happy" songs, were wrongly predicted as "sad", while it misclassified only 11.98% of "sad". The AUC score instead is 68%, which is almost 10% lower than the score obtained using other models.

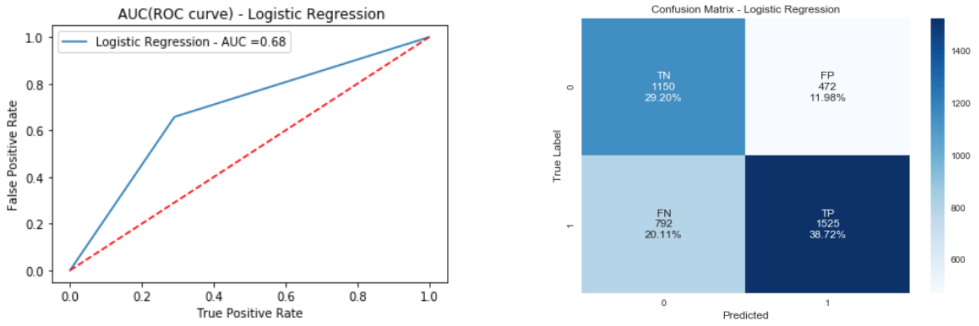


Figure 3.13: On the right the ROC curve, while on the left we show the Confusion Matrix of the Logistic Regression.

3.7 Ensemble methods

Ensemble classifiers were built for classifying songs' emotion. The dataset used for the classification task is *"features.csv"*. We also report an experimental evaluation of a single decision tree (base estimator) which was used as a benchmark to compare different ensemble techniques. Its hyper-parameters were fine tuned through a random search in combination with a 5 fold cross validation, which tested 400 configurations and returned the one with the highest accuracy on the validation set. Our aim was to improve the performance of the base classifier using ensemble techniques. In Table 3.7 we show the classification report of a single decision tree.

Class	Precision	Recall	F1	Support
Happy	0.70	0.67	0.63	1622
Sad	0.75	0.69	0.72	2317

Table 3.7: Classification report of base estimator: Decision Tree.
Accuracy: 68%

3.7.1 Random Forest

The random forest was built taking into account the performance of models constructed with different number of estimators. The base estimator of the forest was the single (optimized) decision tree described above. The optimal random forest employed 200 estimators for classifying samples as "happy" or "sad". As it emerges from the features permutation (run over 10 trials), the most significant feature was *"tonnetz"*. In terms of performance, the model was able to correctly predict only 50% of class "happy", with a precision of 71%. The scores reported for the latter, are definitely inferior w.r.t. those of class "sad". We noticed that even though the overall accuracy increased of +3%, the ability of recognizing "happy" song diminished when using a random forest. Substantial improvements were obtained on the recall and f1 score of class "sad", which were 86% and 78% respectively (as we can see in Figure 3.14).

Performances varying number of estimators

We observed how the model behaved when we varied the number of estimators. The number of configurations that were tested are: 50, 100, 200, 400 and 800. We noticed that the accuracy of the classifier did not exceed 70% when the estimators were ranging from 50 to 100. In general increasing the estimators,

Class	Precision	Recall	f1 score	Support
Happy	0.71	0.50	0.59	1622
Sad	0.71	0.86	0.78	2317

Figure 3.14: Classification report - test set: Random Forest with 200 estimators.
Accuracy: 71%

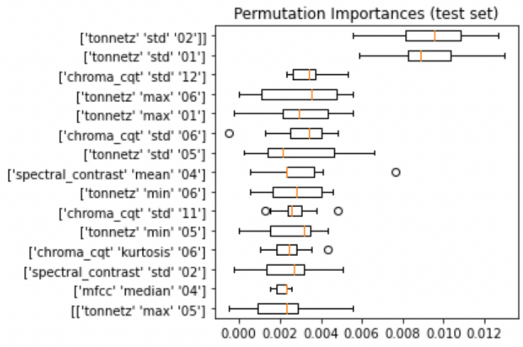


Figure 3.15: Features permutation.

improved the scores of class "happy" (i.e. with 50 estimators the f1 score was 58%, while it increased up to 59% when we added trees to the forest). For our model, 200 estimators resulted to be the "sweet-spot", as it improved the accuracy (71%) as well as the recall of class "happy". When the number of estimators were raised from 200 to 800, the overall performance appeared to converge, as we did not notice any significant gains.

3.7.2 Bagging

Noticeable improvements were provided by bagging. In terms of accuracy we noticed an increase of +1% (72% accuracy) w.r.t. random forest. The recall of class "happy" did not reach a score higher than 59% (which is still a better result than the one provided by random forest). We recorded a clear betterment in the f1 score of class "happy" and in the precision of class "sad" (74% against 71% of random forest).

Class	Precision	Recall	f1 score	Support
Happy	0.68	0.59	0.63	1622
Sad	0.74	0.81	0.77	2317

Table 3.8: Classification report - test set: Bagging Decision Tree.
Accuracy: 72%

3.7.3 Adaboost

Adaboost is known for providing improvement in the performance of classifiers built for solving even the most difficult task. The principles exploited by this algorithm proved to be effective also in our case. In fact this ensemble method is the one that significantly "boosted" the accuracy of the model. To build the ensemble we used 400 stumps (tree with depth = 1). We noticed that as we increased the depth of the trees, the ensemble was more prone to overfitting the training data. When using stamps, the accuracy estimated on the validation set is 72.6% (training accuracy = 75.89%). The expected score almost matched the one obtained on the test test (73%). The ensamble improved the f1 score of class "happy" (67% against 63% of the decision tree used as a benchmark). Important improvement were also observed for the recall and precision of both classes. The classification report is shown in Table 3.9.

Class	Precision	Recall	f1 score	Support
Happy	0.68	0.67	0.67	1622
Sad	0.77	0.78	0.77	2317

Table 3.9: Classification report - test set: AdaBoost.
Accuracy: 73%

3.7.4 Comparing Ensembles Techniques

All the ensemble methods that were tested, improved the performance of the single decision tree. Ensembles proved to be a powerful tool for refining the accuracy of our classifier. Among those that were analyzed, Adaboost was the one with the highest scores. The AUC observed for this method is 80.3% (+6.5% w.r.t. a single decision tree). Random forest and bagging have almost equal performances, as the AUC difference is minimal.

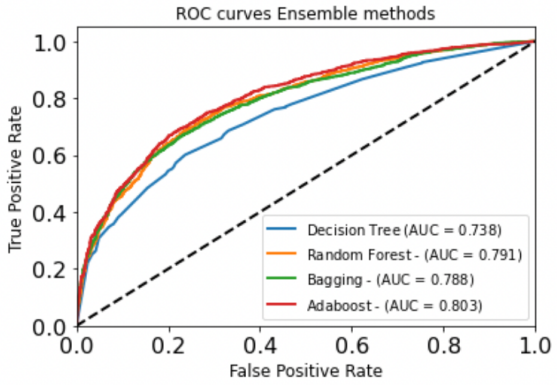


Figure 3.16: ROC curves ensemble.

3.8 Impact of Different Data Sizes

In this section we analyzed how different amount of training data, affected the performances of the following classifiers: SVM, Multi-Layer Perceptron(MLP), Decision Tree(DT), KNN and Random Forest. The models were tested on the dataset *"features.csv"* for solving the task of songs' emotion classification. Each model was tested on different percentages of training data. Ultimately, we compared their performances in terms of accuracy (as shown in Figure 3.17). We notice that SVM has the highest performaces, with a trend that grows almost linearly w.r.t. the increasing size of the training data. Random Forest has better performances than the MLP until the size of the training data exceeds 50%. In fact from that point on, the MLP is the second model with the highest accuracy (72.5%). The accuracy of the DT is negatively affected when the data passes from 40% to 50% (the accuracy decreases of 2.5%), while the KNN seems to start gaining better accuracy as we add records to the training data.

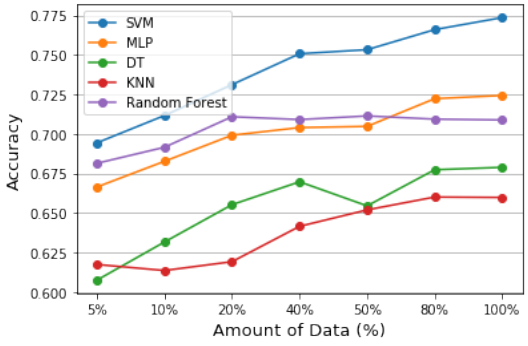


Figure 3.17: Performances in terms of accuracy for models trained on different subsets of training data.

Chapter 4

Time Series Analysis

4.1 Data preparation

The dataset was built by extracting temporal data from the mp3 file provided in the file fma_small.zip. The zip file contained 8,000 tracks with a duration of 30 seconds each. Each record was processed using the Librosa library, through which we extracted the time series using a sample rate of 8,000 (8 GHz). The latter guaranteed high quality tracks which are essential for performing our analysis. As the dimension and the storage-space required was exceptionally large (80,000 features and 14 GB of data), we decided to transform each audio wave into a time series of spectral centroid, which appeared to be more manageable and workable. We chose this feature transformation, as it is a well known measure used in digital signal processing to characterize a spectrum. During the extracting we detected few corrupted mp3 file which were discarded. In order to further reduce the size of our dataset, we filtered it down to 4 balanced genres: "Rock", "Hip-Hop", "Electronic" and "Experimental", for a total of 3994 time series with a length of 657 time stamps.

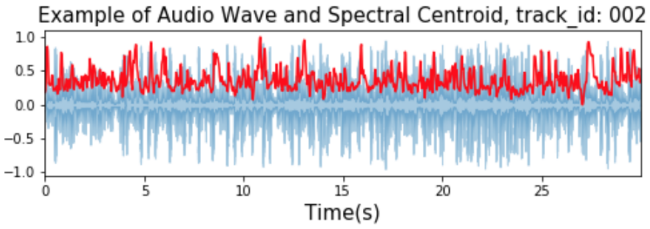


Figure 4.1: Audio wave (in blue) and Spectral Centroid (in red) for track_id: 002.

4.2 Time Series clustering: K-Means

For this task we used the dataset described in the previous section. In the pre-processing step, since our aim was to focus on the shape of time-series, we first applied an amplitude scaling to improve their comparability, then we removed the noise by applying a moving average on a window = 3 time stamps. The chosen clustering algorithm (K-Means) was applied on the dataset approximated with SAX and PAA, as well as on the "raw" audio signal. The metrics we employed are the Euclidean and DTW. The last one, was tested using both Sakoe-Chiba band and Itakura parallelogram. After trying many configuration of segments' number and symbols' we selected 50 segments and 8 symbols (as they better approximeted the shape of the time series). We noticed that with very few segments the approximation resulted in a straight line. Instead as we increased the number of segments and symbols, the results didn't change in terms of shape. Below we show the approximation on one track as an illustrative example.

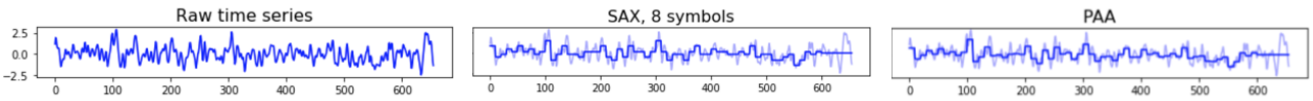


Figure 4.2: Raw time series, SAX and PAA approximation on track 002.

The optimal number of clusters determined by computing the silhouette score and the SSE for every clustering (using a K in range 30). As it was not possible to determine the number of clusters by the "elbow-method", we decided to select the k that maximized the silhouette score. Independently from the approximation or distance, we noticed that the silhouette was approximately 0.05 for every clustering. This meant that the decision boundaries of each cluster were very close. After several trials, we selected k=4. The choice was driven by the fact that we were using a dataset that contained songs belonging to 4 different genres, therefore we wanted to see if K-Means was able to separate songs by genre, assigning them to their own cluster. We noticed that when we employed the euclidean distance, the composition of each cluster was homogeneous and no interesting results could be observed. This highlighted the fact that this distance metric was not suitable for time series, as misalignment caused substantial dissimilarities, even for those time series which were actually similar to one another. When we applied the DTW the clustering goodness improved. Results shown that although the Itakura parallelogram was generally inferior to the Sakoe-Chiba band, it was still superior to the unconstrained DTW. When using SAX and PAA approximation in combination with DTW, we were able to obtain an almost pure cluster for Hip-Hop songs. In fact in clusters where this genre represented the majority, we had very few Rock, Experimental and Electronic tracks. Furthermore, we noticed that in some clusters the composition of Rock and Electronic songs was very similar. After digging deeper in the reason for this grouping made by K-Means, we realized that some tracks of these 2 genres were very similar in terms of shape, probably due to the face that some rock tracks featured electronic instruments. Although the clustering performance improved when using SAX and PAA, the clustering on "raw" time series with the Sakoe-Chiba band DTW provided us the best results in terms of cluster composition. In fact in Table 4.1 we can see (in yellow) that in each cluster there is a prevalence of each genre. It's interesting to notice that in cluster 2 and cluster 4, Hip-Hop songs are almost absent (with percentage not exceeding 0.5%). Experimental and Rock songs have almost the same composition in cluster 4, whereas in cluster 2 we can observe a clear dominance of the former. Overall, if we observe the results obtained with the euclidean distance and DTW, we can conclude that the latter improved the performance and made each cluster more heterogeneous.

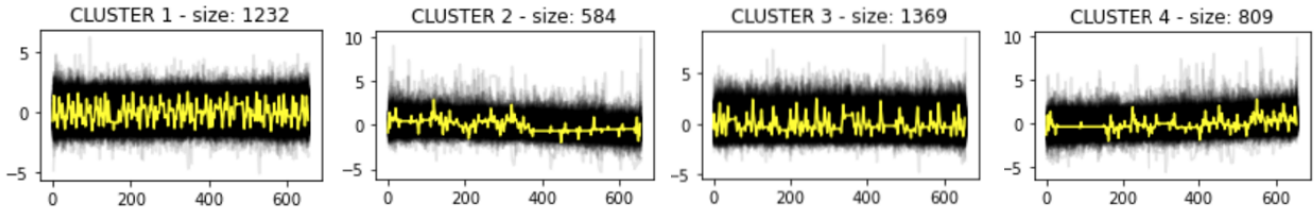


Figure 4.3: Centroids of K-Means clustring on raw time series with DTW (Sakoe-Chiba band).

Raw TS - DTW					
	Hip-Hop	**Rock**	**Electronic**	**Experimental**	***support***
Cluster 1	32.9 %	19.96 %	33.2 %	13.79 %	**1232**
Cluster 2	0.44 %	29.1 %	22.26 %	44.17 %	**584**
Cluster 3	37.25 %	20.23 %	21.54 %	20.96 %	**1369**
Cluster 4	0.06 %	37.82 %	20.27 %	35.10 %	**809**
Raw TS - Euclidean					
	Hip-Hop	**Rock**	**Electronic**	**Experimental**	***support***
Cluster 1	13.9 %	34.75 %	24.06 %	27.27 %	**561**
Cluster 2	20.42 %	27.35 %	23.39 %	28.83 %	**808**
Cluster 3	33.65 %	20.21 %	25.32 %	20.81 %	**1994**
Cluster 4	13.15 %	28.52 %	26.94 %	31.37 %	**631**

Table 4.1: Clusters composition for K-Means with DTW and Euclidean ditance on the raw time series.

4.3 Motifs and Anomaly Discovery

In this section we extracted the top 10 motifs and top 5 anomalies from C-Doc's songs (42 tracks in total). In our dataset, this artist is one the principal exponents of the "Hip-Hop" genre. Each motif was discovered using time series with a length of 657 time stamps (corresponding to 30 seconds) approximated with the spectral centroid. Before searching for patterns, we applied an amplitude scaling to each track. The matrix profile was then tested varying the size of the time window. In total we experimented with 15 configurations, trying low window values (i.e. 5) up to large values (i.e. 200). We noticed that the most accurate results were obtained with a window of 10. Similar patterns were observed in several C-Doc songs. Below we show only the most interesting results.

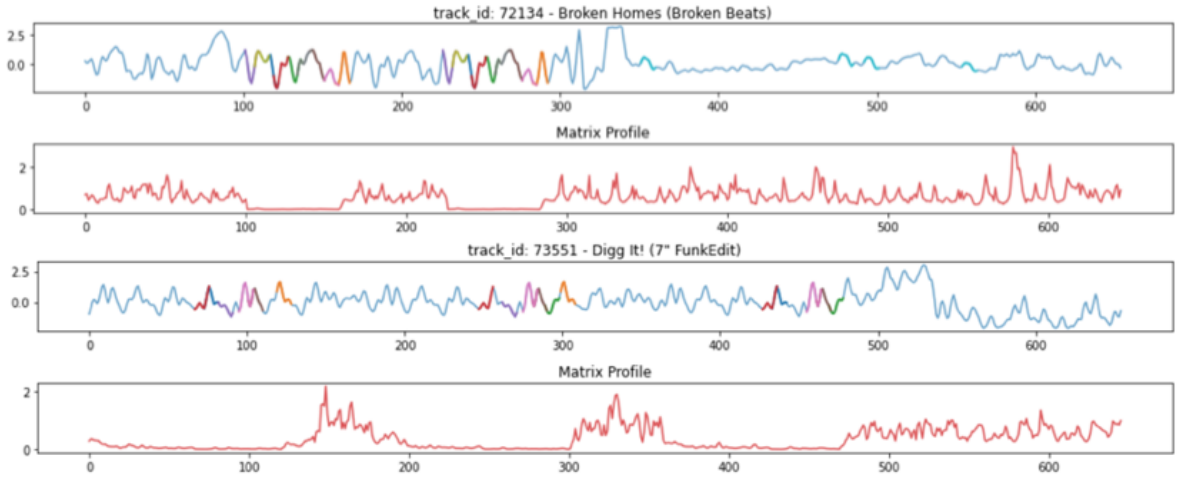


Figure 4.4: Motifs and Matrix Profile of C-Doc songs. "Broken Homes" and "Digg it!".

The Matrix Profile Index of the track "Broken Homes" shows motifs at index: [113, 238], [158, 283], [128, 253], [119, 244], [101, 226], [142, 267], [151, 276], [134, 259], [107, 232], [351, 476, 492, 554]. For those, the distances on the matrix profile (MP) are: [0.0040, 0.005, 0.005, 0.0071, 0.011, 0.013, 0.017, 0.0207, 0.0517, 0.245]. In figure 4.4, we can verify the presence of a repeating pattern starting from time stamp 100 up to 300. In proximity of time stamps 580, we notice a spike in the matrix profile which corresponds to possible anomaly. The song "Digg it!" instead, has a pattern repeating every 100 time stamps. Since the 30 seconds of each track were extracted from the central part of the song, it is intuitive to observe, that the motifs that emerged might be repetition of the melody in the main riff of the song. We also used the MP to detect discords in the time series. In order to detect anomalies, the exclusion zone was heuristically set to half of time window used in the MP (in this case the exclusion zone is 5). For simplicity, we show only few of them. The most evident discord of C-Doc tracks is in the song "Braggadocio" at time stamp 450. We can see a clearly see a peak in the spectrogram which is quite different from the overall pattern in the time series. In fact at that point the distance in the MP is the maximum.

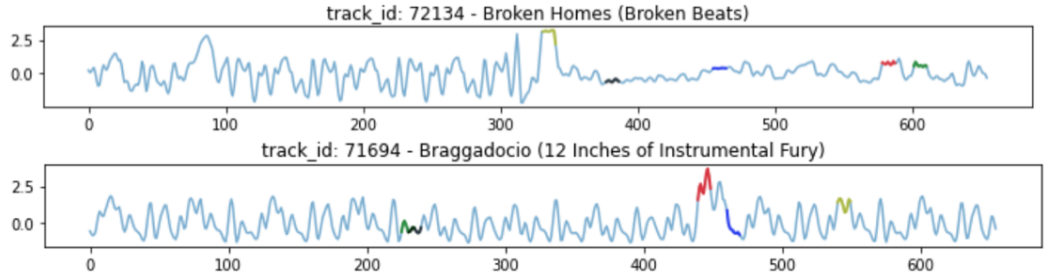


Figure 4.5: Anomalies in C-Doc songs: "Broken Homes" and "Braggadocio".

4.4 Shapelet-based classification

For this task we used the time series dataset described in the previous sections, filtered with only Hip-Hop and Rock songs as we wanted to solve a binary classification problem. The balanced dataset consisted of 1996 songs. To classify the genre, we built a KNN and a Decision tree using a dataset constructed using the top 5 shapelets. We also evaluated the results of the classification, using different shapelets' number and lengths, however the highest performance was provided by 5 shapelets of length 56 (approximately 0.08% of the time series length). Increasing the number of shapelets harmed the performance of our classifiers, with an accuracy that did not exceed 65%. Before proceeding with the extraction, we split the data in training set (70%) and test set (30%). The shapelets were extracted exclusively from the training set. The model used for the shapelet-extraction was trained using Adam optimizer with a regularization of 0.01, over 1500 epochs. The validation accuracy obtained during the extraction resulted to be satisfactory (82,66 % with a binary cross entropy of 0.4025). Below we show the 5 shapelets extracted, as well as their matching on 2 songs belonging to different genres, which were correctly classified on the test set. We observed that although the shapelets appeared very dissimilar from the motifs extracted for the Hip-Hop artist "C-Doc", we noticed a slight correlation in terms of shape between shapelet 4 and the motifs of the track "Broken Homes" (Figure 4.4).

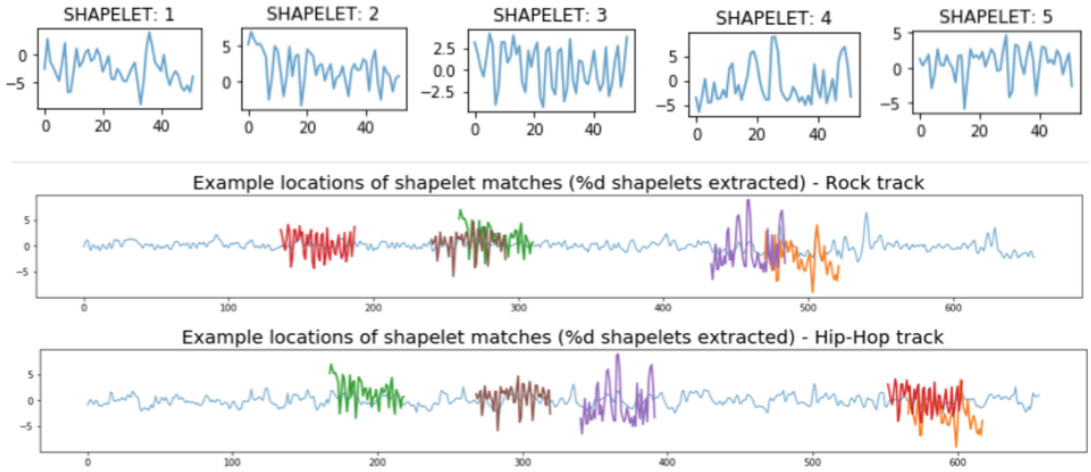


Figure 4.6: Example of matching between Rock and Hip-Hop time series with the extracted shapelets.

After we constructed the shapelet-based dataset, we used it to train a KNN and a decision tree classifier. The hyper-parameters of each model were fine-tuned through a coarse grid search and a 5 fold cross validation. The best accuracy was yielded by the KNN with K=23 in combination with the Manhattan distance. As we can see from Table 4.2, the KNN outperformed the decision tree, with an increment in the accuracy of approximately 4%.

	Shapelet-based KNN			**Shapelet-based Decision Tree**			
	Precision	**Recall**	**f1-score**	**Precision**	**Recall**	**f1-score**	**Support**
Hip-Hop	80%	73%	76%	76%	67%	71%	299
Rock	75%	82%	79%	71%	79%	75%	300
Accuracy		77.62 %			73.12 %		

Table 4.2: Calssification report of shapelet classifiers. KNN: n_neigh=23, metric = Manhattan. Decision Tree: max_depth = 12, min_sample_leaf = 8, min_sample_split = 2, criterion = Entropy.

Chapter 5

Advanced Clustering, Sequential Pattern Mining & AI Explainability

5.1 Sequential Pattern Mining

In this section we performed sequential pattern mining following 2 approaches. In the first approach, we used all the tracks produced by the Hip-Hop singer:"The Impossebulls". In total we worked with 68 tracks, each of which with a length of 657 time stamps. In the pre-processing step, we converted each time series into sequences. We first applied a SAX approximation using 60 segments and 20 symbols. At the end of this step each track was represented as an numpy array of size 60. However we noticed that our tracks were represented by sequences of 1 transaction of 60 events each. Clearly this transformation lost the concept of time. In order to recover that, we split each sequence into 30 transactions of 2 events each. This enabled us to discover frequent sequences over a span time of 30 transactions per sequence. Below we graphically summarize the transformation adopted.

Original sequence obtained using SAX tranform (20 symbols, 60 segments)
Sequence of 1 transaction with 60 events.
[16, 4, 9, 13, 10, 15, 10, 11, 9, 10, 19, 5, 6, 15, 11, 8, 8, 10, 9, 9, 17, 6, 2, 13, 5, 15, 11, 12, 6, 9, 17, 11, 5, 6, 9, 4, 9, 15, 7, 8, 14, 9, 11, 8, 8, 10, 7, 14, 8, 7, 9, 4, 3, 5, 9, 1, 3, 11, 9, 11]
Transformed: Sequence of 30 transactions with 2 events each.
[(16, 4), (9, 13), (10, 15), (10, 11), (9, 10), (19, 5), (6, 15), (11, 8), (8, 10), (9, 9), (17, 6), (2, 13), (5, 15), (11, 12), (6, 9), (17, 11), (5, 6), (9, 4), (9, 15), (7, 8), (14, 9), (11, 8), (8, 10), (7, 14), (8, 7), (9, 4), (3, 5), (9, 1), (3, 11), (9, 11)]

We then extracted frequent sequences using prefixspan with a min support of 18%. In Table 5.1 we show the top 5 frequent sequences that were obtained. It's worth to mention that the explainability of the results was limited by the approximation applied to the time series. As a second approach instead, we used the information provided by the features "tags" and "date_created", which were obtained from the dataset tracks.csv. Selecting tracks belonging to one single artist we could explore how the tags associated with the song of that artist changed over time. The dataset was composed by 266 sequences (tracks) of the singer Ars Sonor, span over a time period that goes from 2013 to 2015. After a pre-processing step, through which we converted each row into a sequence, we extracted the top 5 frequent sequences (as shown in Table 5.1).

Frequent sequences (spectral centroid)	**Matches**
[(7, 8)]	728
[(8, 7)]	743
[(6, 6)]	723
[(7, 7)]	726
[(8, 6)]	727

Frequent sequences (Ars Sonors' tags)	**Matches**
[('24-bit')]	176
[('era 3b')]	117
[('era 3b'), ('24-bit')]	91
[('kristin e')]	41
[('wallenberg', '24-bit')]	33

Table 5.1: Frequent sequences of spectral centroid time series (on the left) and songs' tags (on the right)

5.2 OPTICS

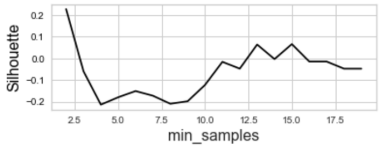


Figure 5.1: Silhouette by varying min_samples

For this task we decided to cluster Classical songs using both OPTICS and DBSCAN, in such a way that the differences emerging from the two algorithms could be compared. In order to better study the effects of OPTICS, we decided to filter our dataset and consider only 2 features, namely: "liveness" and "danceability". This choice is driven by the fact that density based algorithm lose their effectiveness in high dimensions, and since we wanted to compare the results of the two algorithms we decided to use only a restricted by meaningful set of features. The dataset is composed by 265 tracks, whose attributes were normalized before being used in the clustering. Although OPTICS reduces the number of hyper-parameters (as we don't need to specify the epsilon radius), it still needs the setting of the parameter min_samples, which appears to be crucial for a correct clustering. To choose the correct configuration, we plot the silhouette score by varying the number of the previously mentioned parameter (as shown in Figure 5.1). We noticed that when the min_samples is set to 13 the silhouette is optimal (approximately 0.063).

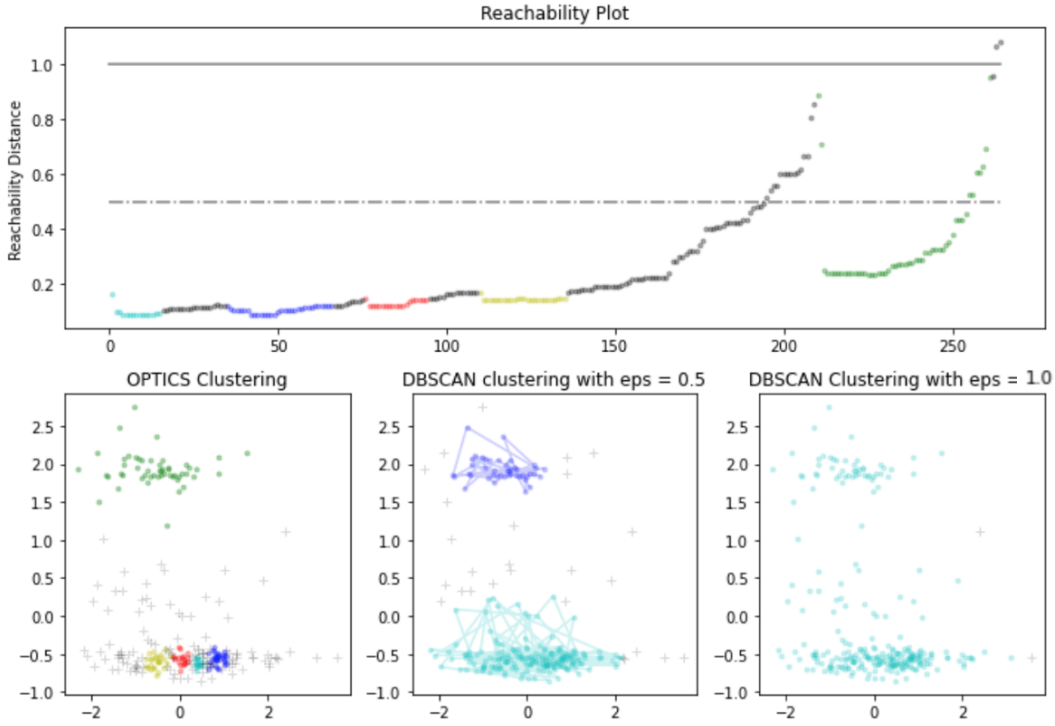


Figure 5.2: OPTICS clustering compared to DBSCAN.

In Figure 5.2 we show the results obtained using OPTICS and DBSCAN. The latter was used first with a small epsilon (0.5) and then with a large one (1.0) in order to see how the clustering changed in the 2 extreme cases. We noticed that OPTICS was able to generate 5 clusters. The green cluster (top left corner) identifies songs that are highly danceable and that were not recorded during live concerts. We notice that majority of classical songs are not suitable for dancing, as the remaining 4 clusters have low danceability regardless of the liveness coefficient. It is interesting to see that while the green cluster was detected also using DBSCAN, the latter grouped the 4 clusters identified by OPTICS in one single cluster. This highlights the difficulties that DBSCAN faces when clusters with different density are close to each other. OPTICS on the other hand, performed well in the task of differentiating clusters in area with heterogeneous density. On the reacheability graph, the dot line represents the cut off level after which data points are considered outlier in DBSCAN with epsilon = 0.5, while the straight line has the same meaning w.r.t. DBSCAN with epsilon = 1.

5.3 Transactional Clustering: K-Modes

For this task we used a dataset built with discretized continuous features extracted from echonest.csv and categorical attributes obtained from tracks.csv (from this dataset we used: language, license and genre). For discretizing the continuous variables, we processed each attribute and identified the best threshold for generating the categorical values. Below we summarize the results.

Features	***Discrete values***
Energy	['low', 'medium', 'high']
Emotions	['sad', 'happy']
Liveness	['recorded live', 'recorded in studio']
Speechiness	'[instrumental', 'balanced', 'spoken']
Danceability	['not danceable', 'danceable']
Listens	['low', 'medium', 'high']

Table 5.2: Continuous variable discretized for K-Modes algorithm.

In total the transactional dataset has 9 features and 4784 instances. To determine the optimal number of clusters, we performed several trials and we selected the one with the most interesting results. We clustered the data with K=3 using Huang initialization repeated 50 times. Below we show the centroids yielded by the K-Modes algorithm.

Clusters	**Centroids**
0	[language: 'English', 'low_listens', 'Rock', 'danceable', energy: 'medium', license: 'Attribution-Noncommercial-Share Alike 3.0 United States' , 'instrumental', 'happy', 'recorded_in_studio']
1	[language: 'English', 'low_listens', 'Folk', 'not_danceable', energy: 'low', license: 'Attribution-Noncommercial-Share Alike 3.0 United States', 'recorded_in_studio', 'instrumental', 'sad']
2	[language: 'English', 'low_listens', 'Rock', 'not_danceable', energy: 'high', license: 'Attribution-Noncommercial-Share Alike 3.0 United States', 'recorded_in_studio', 'instrumental', 'sad']

Table 5.3: K-Modes centroids.

We noticed that we obtained 2 centroids in which the genre Rock is present. One of which (cluster 0) groups songs which are considered happy, danceable and having medium energy, while the other centroid (of cluster 2) represent sad songs which strangely have high energy and are not danceable. The most interesting centroid is the 2nd, which is representative of sad Folk songs. Tacks in this cluster, are not danceable since they have low energy. We concluded that this cluster grouped relaxing songs that stimulate melancholic and sad emotions. To better visualize and understand the composition of the obtained clusters we built and inspected the related count-plot. Below we show only the most significant ones.

Figure 5.3: Count-plots of Energy, Danceability and Emotions

5.4 AI Explainability: LIME

In this section we extracted an explanation that provided the reasons why the Random Forest classifier described in the previous sections, classifying the test instances in certain way. To do so, we first built and trained a Random Forest classifier using the same parameters and partitions employed in Section 3.7.1.
We then used LIME algorithm, which uses a logistic regression to explain the behaviour of the black box of our classifier. We displayed the explanation for 2 randomly selected instances (correctly classified by the model) belonging to the 2 test classes.

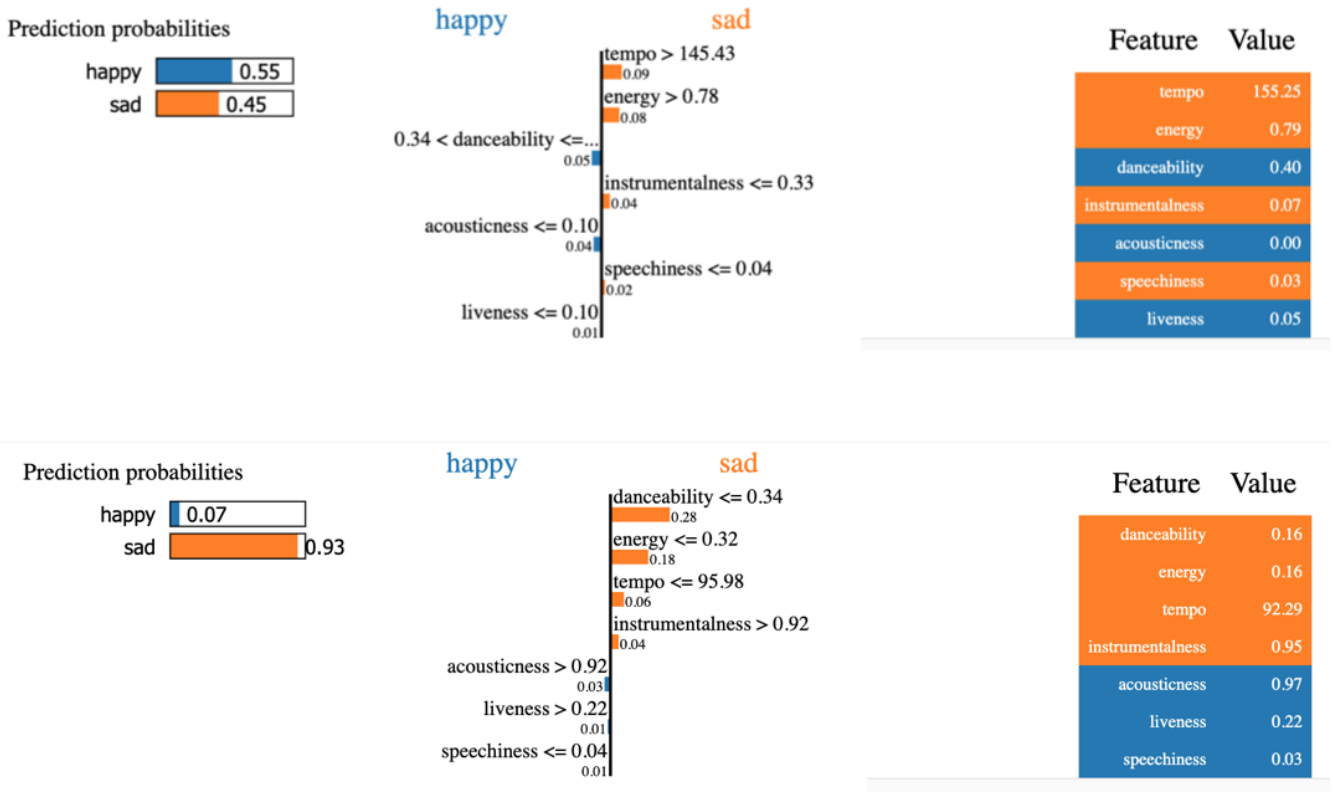


Figure 5.4: LIME explanation of randomly sampled instances of class "sad" and class "happy".

The explainer of instance classified as "happy" shows that the main factors that positively contributed to the result are danceability which is 0.40 (apparently a danceability >= 0.34 brings the classification closer to "sad" songs), tempo and energy which is 0.79. If we compare the two explainations, we see that when energy, danceability and tempo are low, and acousticness is high (>0.92) the songs are identified as sad songs by the model. Instead happy songs, are characterized by high danceability, energy (>0.78) tempo, and low acoustincness (<=0.10).

Conclusions

In this report, we had the opportunity of working with a large dataset that put at our disposal, many alternatives for solving various classification and clustering tasks.
In **Chapter 1** we performed data understanding, cleaning and features/dataset engineering.
In **Chapter 2** analyzed the negative effects that imbalanced dataset have on classification tasks, and we then improved the performances of simple classifiers such as Decision Tree and KNN, by applying and comparing various imbalanced techniques (Random Undersampling, CNN, Tomek's Link, Random Oversampling, SMOTE, K-Means SMOTE, ADASYN and Cost Matrix). We then discovered and removed the top 1% of anomalies present in our dataset (using DBSCAN, LOF, ABOD, Isolation Forest, Extended Isolation Forest and Autoencoders), and observed improvements in the classification metrics.
In **Chapter 3** we applied advanced classification methods such as Naive Bayes Classifier, Multi-Layer Perceptron, Linear and Non-Linear SVM, RIPPER rule-based classifier, Univariate and Multivariate Linear and Logistic Regression, and Finally Ensemble methods (Random Forest, Bagging and Adaboost).
In **Chapter 4**, we built a time series dataset from scratch, by extracting the spectral centroid from raw mp3 data. We then applied K-Means clustering (experimenting with Euclidean and DTW distances) and we then identified the top 10 motifs and top 5 anomalies of songs written by the singer "C-Doc". Finally, we extracted the top 5 shapelets, and based on those, we trained a Decision Tree and a KNN classifier.
In **Chapter 5** instead, we transformed the time series dataset into a sequential dataset and we performed sequential pattern mining. We also extracted frequent sequences of tags associated to the singer The Impossebulls span over a period of 3 years. We also compared the OPTICS and DBSCAN clustering algorithms and highlighted their differences in terms of clustering performance. We built a transactional dataset from scratch exploiting the information provided by the raw datasets, and performed the K-Modes clustering. As last task, we utilized the LIME explainer to provided a clear explanation of why the Random Forest described in Chapter 3 output a certain class label.
Overall this project enhanced our reasoning and analytic skills, but also consolidated the theoretical concepts explored in the course.