Department of Computer Science
Master’s degree in data science and business informatics

DM2 – Data Mining

Submitted by:

Castriotta Antonio (685223)
Di Benedetto Giorgia (673201)
Gobbin Giorgio (683071)

Academic Year 2023/24

Index

Chapter 1: Data Understanding and Data Preparation

Introduction

The analysis we are going to present was done using 3 different datasets: 2 tabular datasets, Artists and Tracks, and a time series dataset. In this first chapter we are going to explore and prepare the tabular datasets, as we did for the Data Mining 1 project. In this same chapter we are also going to do the preprocessing of the Time Series Dataset to prepare it for the tasks that will be presented in chapter 2.

1.1 Tabular Dataset: Artists

The Artists tabular dataset contains 30141 rows, each representing an artist, and 5 features. We dropped the *id*, a categorical feature which we found quite useless for our analysis. We kept the *name*, *popularity*, *followers* and *genre* features. We checked for null values and found just 2 for *name* and 1 for the other features, we decided to drop them, as we considered them not significant, given the dimensions of the dataset. Then we checked for duplicates, we just found 3, so we dropped them.

1.2 Tabular Dataset: Tracks

The Tracks Tabular Dataset contains 109149 rows, each representing a spotify track, and 34 columns with the features of the tracks. We explored the different features and checked for null values, and we found none, and as for duplicates we found 398, that given the dimensions of the datasets, were dropped. We did another duplicate check based on the *id* column, and we were surprised to find 19587 duplicates. Analysing them we realised that the records signed as duplicates were almost identical on all the features, except for *genre.* Given the total number of duplicates over the number of total observations of the dataset we decided to drop them all.
We proceeded doing a correlation matrix, we decided to drop all the features that had a correlation higher than 0.8, we identified the columns *faeature_duration_ms, start_of_fade_out, n_bars, album_total_tracks, mode_confidence.*
As for the variable *Explicit*, we decided to convert it from [True, False] to [1, 0].

1.3 Time Series Datasets

In this section on time series, we will work with the provided dataset of 10,000 time series, with 500 time series for each of the 20 genres. We loaded the time series and decided to normalise them to a uniform length of 1280 observations. In the following section, we applied several transformations and visualisation steps. First, we grouped the dataset by *genre* in order to have a representative time series for each of the genres (Figure 1). For a better and more comprehensible visualisation, we also randomly selected 5 genres (Figure 2).

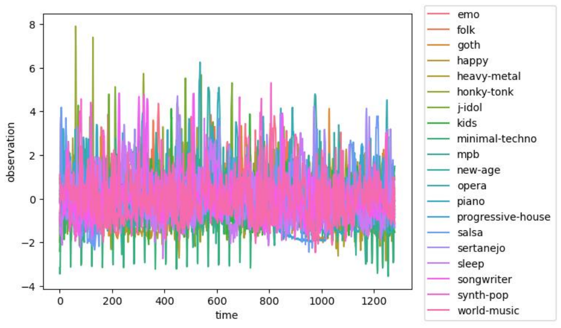

Figure 1.1: Time series plot

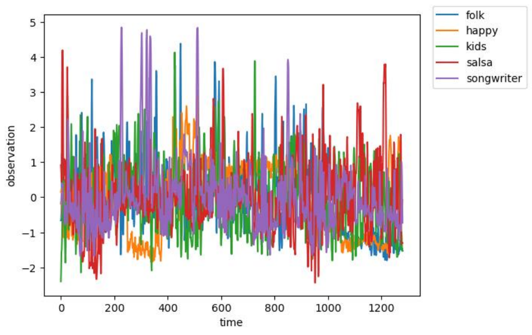

Figure 1.2 Genre Time Series Plot

As part of the time series amplitude scaling transformation, we applied standardisation to remove trends in the data and obtain a mean of 0 and a unit variance. Next, we applied a moving average with a window of 60 to reduce noise in the time series. To facilitate subsequent operations and to have less computationally expensive operations, we decided to adopt 3 different methods of approximating the time series:

- Piecewise Aggregate Approximation (PAA) with 100 segments;
- Symbolic Aggregate approXimation (SAX) with 100 segments and 14 alphabet symbols;
- Discrete Fourier Transform (DFT), using 32 Fourier coefficients.

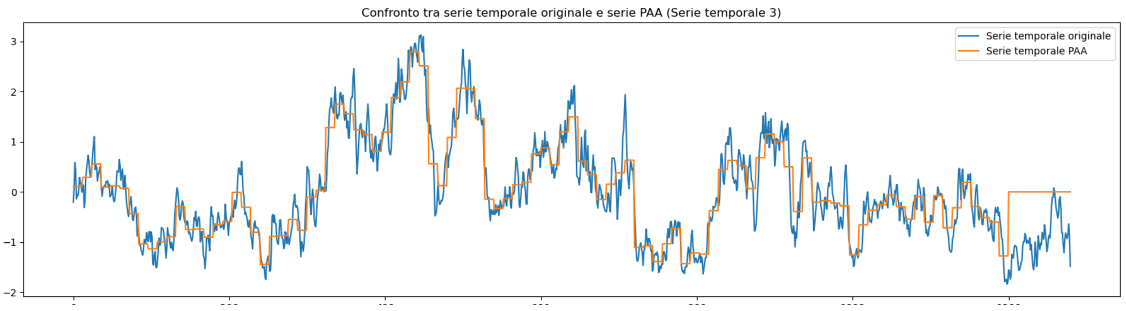

Figure 1.3 PAA approximation of a Time Series

We plotted (Figure 3) an original time series with on top the approximation of the PAA algorithm.

Chapter 2: Time Series Analysis

2.1 Clustering

The first analysis performed on the Time Series was clustering, carried out with the aid of two algorithms: *KMeans* and *Hierarchical Clustering*, using Euclidean and Dynamic Time Warping (DTW) as distances. In both cases, a search was carried out for the best k (number of clusters), observing the *SSE* and *Silhouette Score*. The procedure was repeated for all approximations listed above.

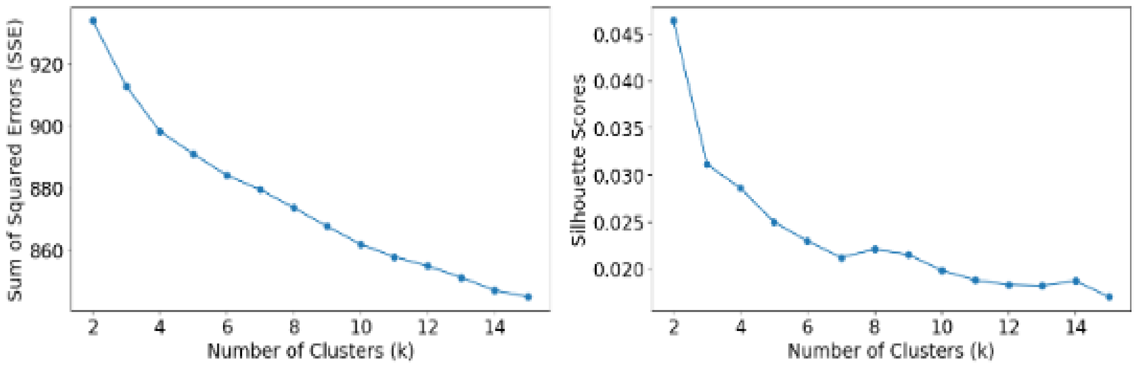

Figure 2.1 Plots of SSE and Silhouette Score approximation SAX

In the case of the *KMeans* algorithm, it was possible to search for the best *k* for the algorithm with *Euclidean distances*, constructing the graphs shown in Figure 2.1.
The best number of clusters identified for the *PAA* approximation is k=6, while for the *SAX* approximation it is k=7. For clustering with *DTW* distance, due to the excessive time required for calculation, we decided to adopt the same k identified for Euclidean distances.
Regarding the *DFT* approximation, given the low dimensionality, it was possible to find the value of k using both *Euclidean* distance and *DTW*, finding an optimal k of 6 in both cases.
All results are shown in the table. In all cases, the application of DTW in the algorithm significantly improves the *SSE* values.

		k	SSE	Silhouette
SAX	Euclidean	7	879.40	0.02
SAX	DTW	7	285.37	-0.01
PAA	Euclidean	6	45.50	0.02
PAA	DTW	6	16.12	-0.16
DFT	Euclidean	6	211285.27	0.03
DFT	DTW	6	91050.43	-0.06

Table 2.1 Number of clusters k, SSE and Silhouette for each approximation and distance

For the second algorithm, we applied *hierarchical clustering* techniques on the datasets with the approximations as for the previous algorithm and evaluated the quality of the clusters obtained using the *Silhouette Score* here as well.
Hierarchical clustering was applied to the data to create a tree structure (*dendrogram*) showing the similarity between different data points.
First, we used the linkage method to create the dendrogram, then we set a threshold to obtain the clusters. For the *SAX* approximation the threshold set was 200, for *DFT* we set a threshold of 4000 and for *PAA* the threshold chosen was 40.

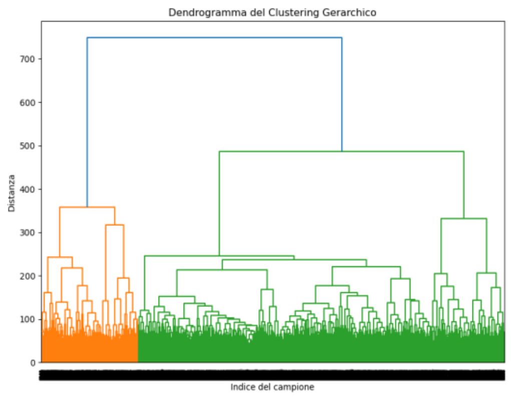

Figure 2.2 Dendrogram approximation SAX

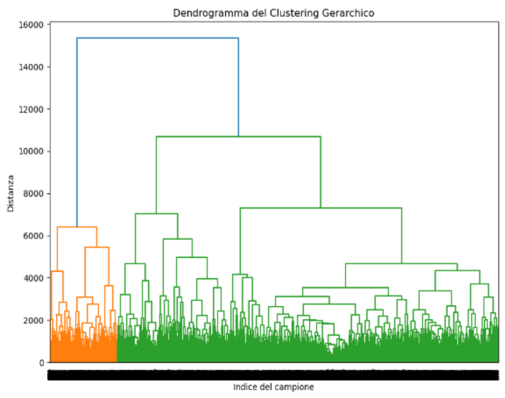

Figure 2.3 Dendrogram approximation DFT

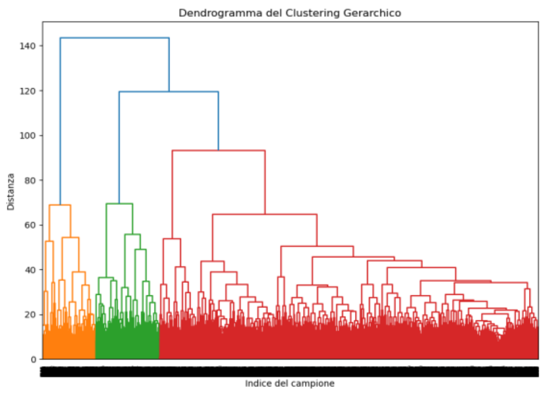

Figure 2.4 Dendrogram approximation PAA

We then evaluated for each approximation the number of elements in each cluster. The quality of clustering was assessed using the *Silhouette Score*, obtaining the results shown in the following Table 2.2.

	Silhouette
SAX	0.03
PAA	0.02
DFT	-0.06

Table 2.2 Silhouette score of hierarchical clustering for all approximations

The hierarchical clustering analysis performed on these datasets showed some difficulties in defining well-separated clusters, as indicated by the low Silhouette Score.
Looking at the clusters obtained from the various approximations in both algorithms, there is an imbalance in the number of Time Series in the clusters.
The only approximation for which this problem is significantly alleviated by using *DTW* as the distance is the case of *SAX*, of which the number of Time Series in each cluster in the case of the two distances is given. (with *tskmeans*)

	Euclidean	DTW
Cluster 0	749	1143
Cluster 1	1763	1259
Cluster 2	1188	1145
Cluster 3	782	727
Cluster 4	814	1019
Cluster 5	1301	1300
Cluster 6	1403	1407

Table 2.3 Number of TSs in each cluster SAX

2.2 Conclusions and cluster analysis

To further evaluate the composition of the clusters, we used those obtained from the *TSKM* algorithm with *DTW* applied on the dataset approximated with *SAX*, because it turned out to be the one that best distributed the TimeSeries across the various clusters.
It is important to note, however, that the value of *Silhouette* is -0.01, thus very close to zero and thus a clear sign of clusters that are not well separated. The different clusters can be observed with three different dimensionality reduction techniques: *Isomap, Pca* and *T-SNE* (Figures 2.5, 2.6, 2.7).

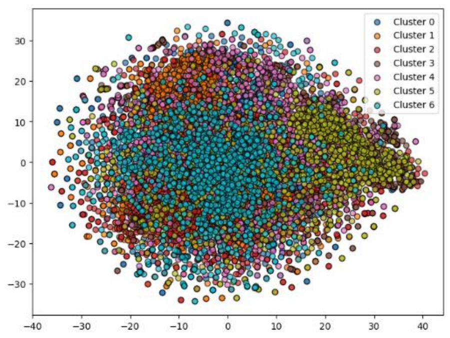

Figure 2.5 t-SNE

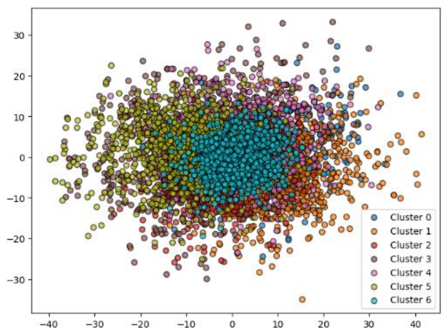

Figure 2.6 PCA

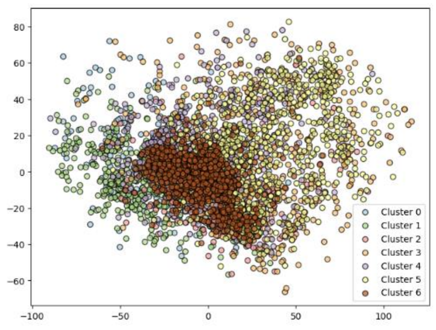

Figure 2.7 IsoMap

A further cluster analysis was then carried out, analysing their composition and observing the distribution of the different time series within the clusters.
A stacked bar chart showing the distribution of musical *GENRES* in the different clusters generated is shown in the Figure 2.8. Each bar represents a cluster, and the colours inside represent the different music categories.
Each cluster contains a variety of musical genres; there is no cluster dominated exclusively by a single genre, highlighting the even distribution of genres across clusters, with no obvious peaks for any particular genre.

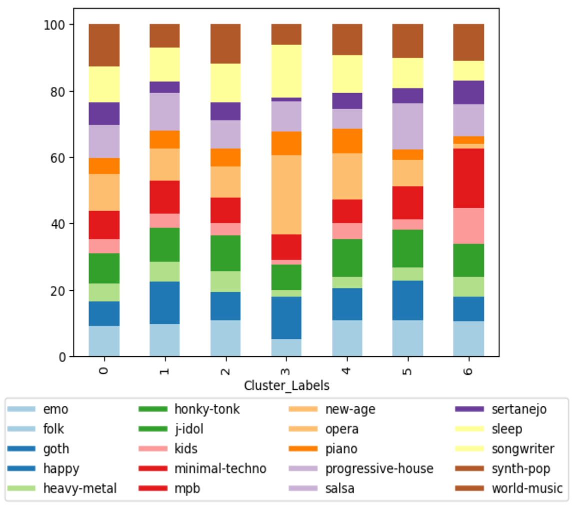

Figure 2.8 Stacked Barchart

2.3 Motifs and discords

In this section, we are going to discover and analyse the *motifs* and *discords* present in the time series. To do this analysis, we took a sample of 100 time series, five for each genre in the dataset. For the sake of simplicity, we will present only one time series (TS 1871), the one that was closest to the centroid of the largest cluster obtained from the previous analysis, but the steps were performed for each time series within the sample. We decided to analyse the TSs by taking them from the original dataset, in order to capture information that might be lost in the approximations.

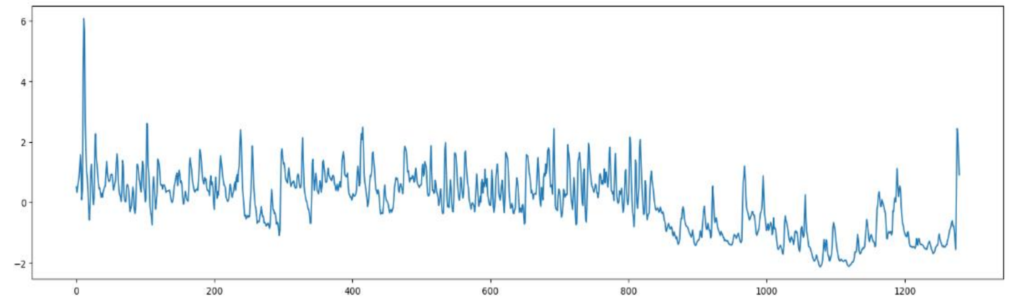
Figure 2.9 Original time series (1871) on which the analysis is performed

An important aspect in the analysis of motifs and discords is finding the right value of the *w-window* on which to search. During the search phase, we performed several tests with windows of different widths (3, 10, 15, 20, 25, 50, 75) and calculated the matrix profile for each. For smaller windows, the results were not satisfactory, as only a very small part was identified as motif, whereas in the case of larger windows, the entire Ts was evaluated as motif. Finally, we found the window w = 15 to have the best results for this analysis.
By setting the maximum number of motifs = 10 and discords = 5, we can see how the algorithm manages to identify the main motifs and discords. In the figures we show the various tests performed to find the right value of w, the corresponding *matrix profile* and the TS with the motifs and discords highlighted.

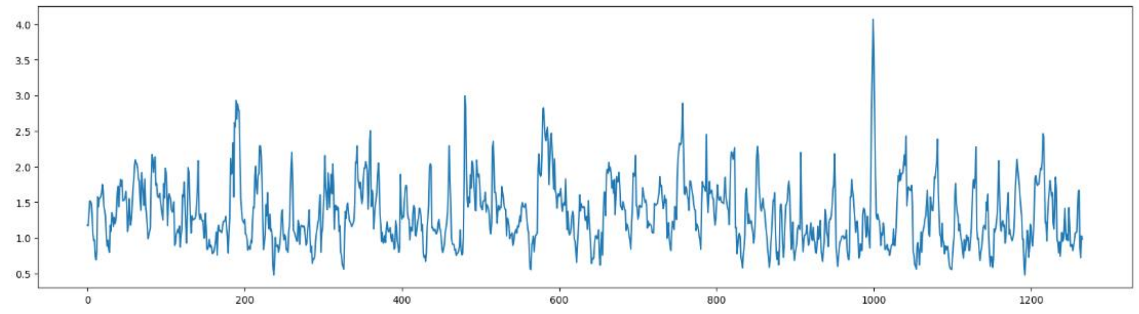
Figure 2.10 MP with w=15

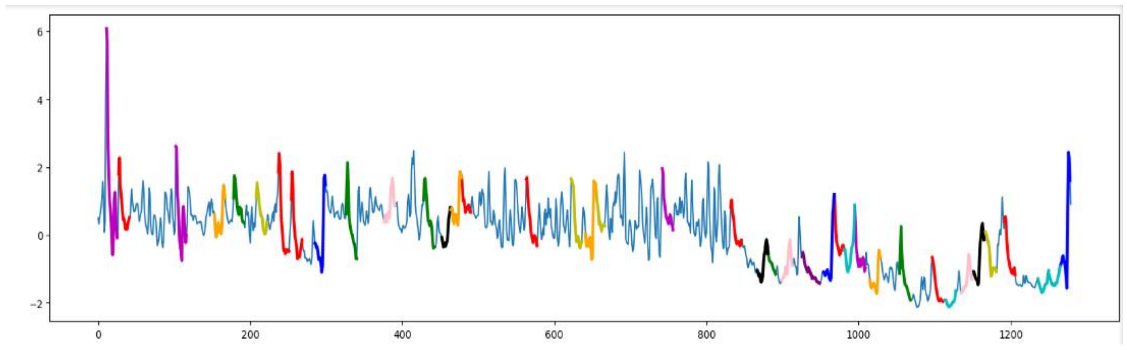
Figure 2.11 TS Motifs with w=15

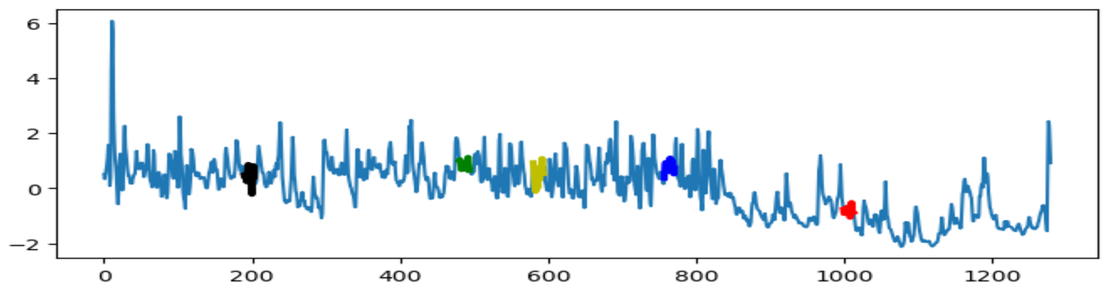
Figure 2.12 TS Discords with w=15

By working on the original dataset, we obtained analyses and information that were certainly more faithful, but as the TS is long and complex, it is easier to work on approximations. In a second step, we decided to perform the analysis on the *SAX* approximation. Working on a reduced dataset, it was not possible to test all the amplitudes proposed for the original dataset.

In the following Figures we show the *motifs, discords* and *matrix profiles* of the best amplitude w = 6. In this case we have decided to reduce the maximum number of motifs to 5.

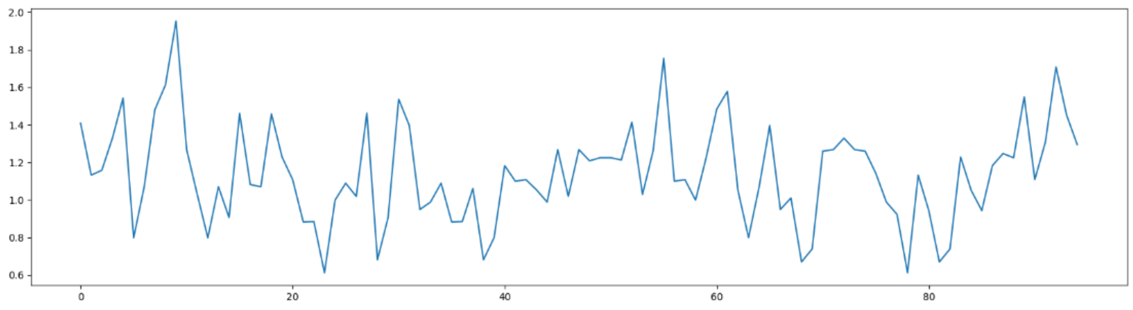
Figure 2.13 MP with w= 6

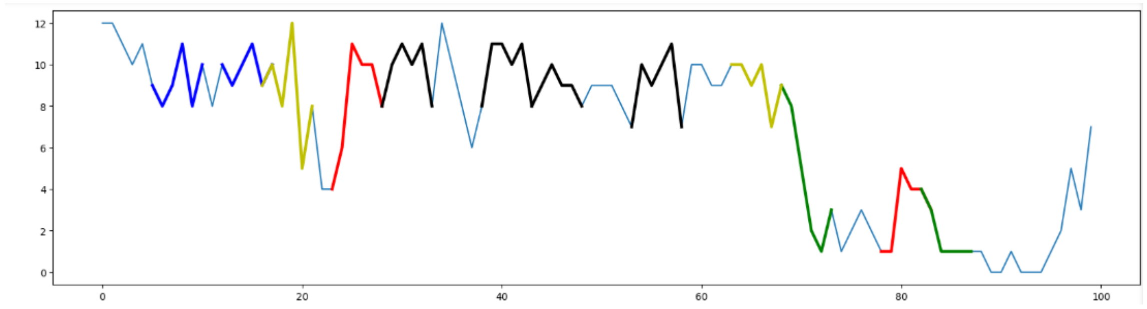
Figure 2.14 TS Motifs with w= 6

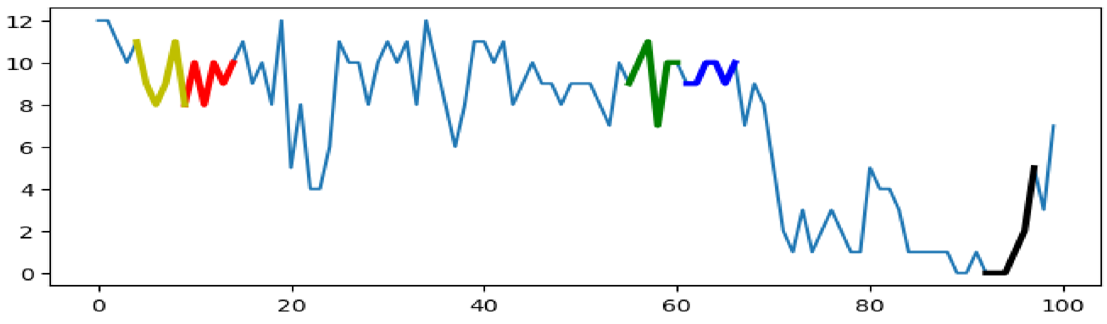
Figure 2.15 TS Discords with w= 6

2.4 Classification

To perform this task, we decided to work using a *KNN* trained with both *Euclidian/Manatthan* and *DTW* applied to different approximations of the original data, i.e. *SAX* and *DFT.*

SAX (Symbolic Aggregate approXimation) – genre

In this case, we trained the *KNN* model by tuning the parameters using a *Random Search* with 10 iterations and 5-*fold-cross-validation*; the table 2.4 shows the parameters tested.

parameters	Values tested	best
P n neighbors weights	'1 (Manhattan)', '2 (Euclidean)'] (1,3,5,10,15,30,50) ['uniform', 'distance']	1 5 distance
accuracy		0.1245

Table 2.4 Tested parameters and best SAX value with Euclidean/Manhattan distance

The parameters found are: 'weights': 'distance', 'p': 1, 'n_neighbors': 5. We can say that the performance is not satisfactory as the accuracy value is 0.1245 turns out to be very low, with a precision ranging from a minimum of 0.05 to a maximum of 0.33.
Shown is the *ROC Curve* graph with a *ROC AUC* value of 0.56.

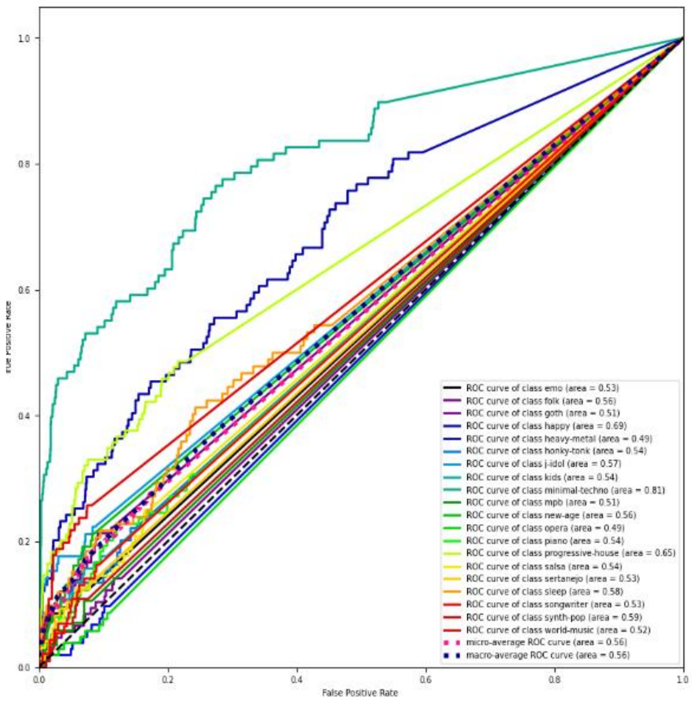

Figure 2.16 ROC Curve Manhattan

We then performed a further Random Search for parameter tuning, but using *DTW* as the distance, testing both the *dtw_fast* and *dtw_sakiechiba* metric versions. By training the model with the parameters found (weights = 'uniform', n neighbours = 3 and metric = 'dtw sakoechiba'), the accuracy rises to 0.1305. The model turns out to be slightly better than the previous one, but still not satisfactory, with precision values ranging from 0.09 to 0.40.
Looking at the *ROC Curve*, we note that the happy and minimal-techno genres are the only ones to stand out from the others in terms of higher prediction ability, with a value of 0.72 for the former and 0.83 for the latter.

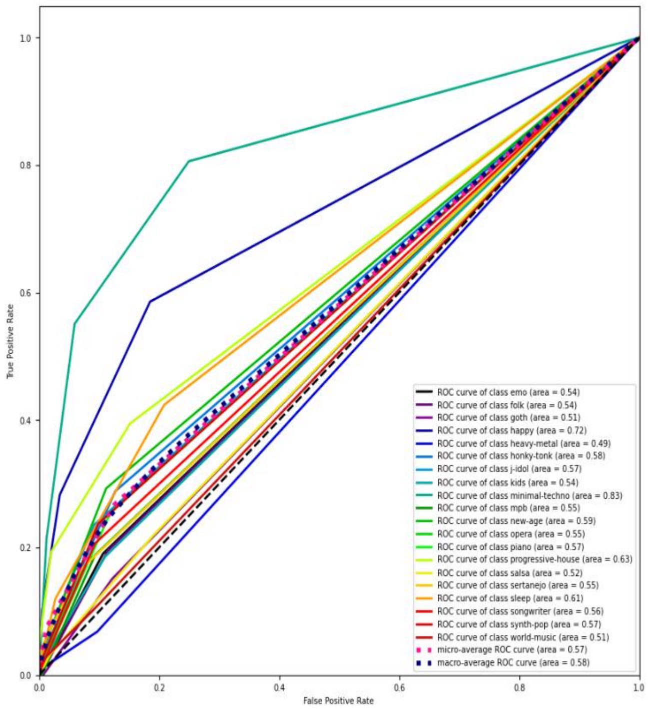

Figure 2.17 ROC Curve DTW

DFT (Discrete Fourier Transform) – genre

We performed the classification task on the target genre variable also using the *DFT* approximation, in order to see if the results could be improved. Again, we proceeded similarly to the previous case, using a *KNN* model and tuning the parameters using a *Random Search* with 10 iterations and 5 *fold cross validation*. Again, we performed two tunings using first the best distance between Euclidean and manatthan and a second tuning using *DTW*.

The results of the random search are shown in the table 2.5.

parameters	Tuning 1	Tuning 2
distance n neighbours weights	Euclidean 5 'uniform'	'dtw sakoechiba' 30 distance
accuracy	0.112	0.132

Table 2.5 Comparison of different parameter values for the two model tunings

Focusing on the first tuning, we can see that the accuracy value of 0.112 is slightly worse than previously obtained with *SAX* and especially in this case we do not have a precision value for the *world-musi*c class. The highest precision value is reached for the *j-idol* class with a value of 0.20.

After training the model according to the parameters obtained from the second tuning, we obtain an *accuracy* value of 0.132.
In this case, the best predicted classes are *minimal-techno* with a *precision* of 0.27 and *new-age* with a *precision* of 0.22, while the worst class is goth with a *precision* = 0.01. The *ROC Curve* of the classifier just described is shown in the figure 2.18.

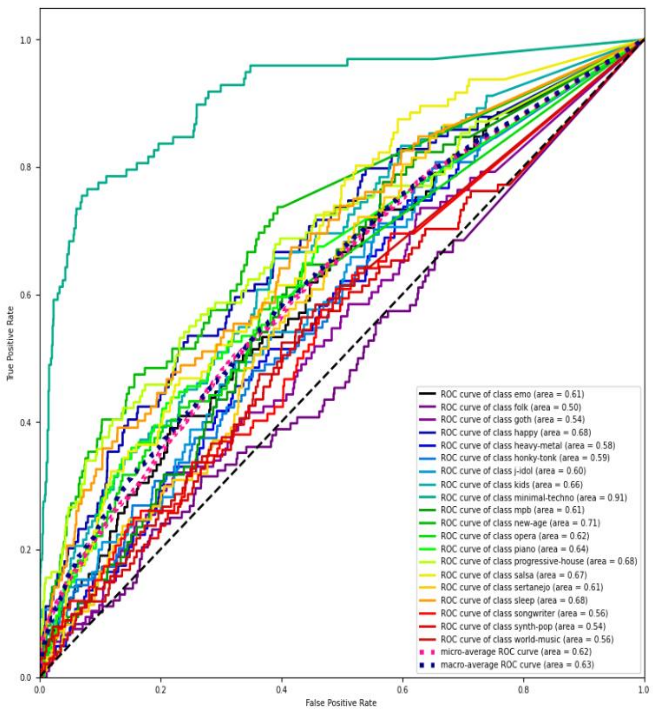

Figure 2. 18 ROC Curve DTW

ROCKET

To conclude the task, we decided to test the classification performance using the *ROCKET* algorithm applied to our approximate data, so that we could compare the results.
Once the model was built using the *Rocket()* function, we then tested its performance on the approximate dataset, obtaining the results shown below.

	accuracy
SAX	0.06
DFT	0.11

Table 2.6 Comparison of Rocket results with SAX and DFT approximation on 'genre' variable

In terms of accuracy, we can immediately see that *DFT* achieves a value of 0.11, almost double the accuracy achieved by *SAX*, which has a value of 0.06. In addition, looking at the classes predicted by *SAX*, we can see that only the *goth*, *happy*, *honky-tonk, j-idol*, *mpb*, *sentanejo* genres are predicted by the classifier, with precision values very far apart. In the case of DFT, all classes are predicted by the classifier, with a precision reaching a maximum value of 0.24. Although the results obtained by this classifier are not satisfactory for our classification task, we can conclude our analysis by stating that in general, the results obtained using the *DFT* approximation are better than those with the *SAX* approximation.

2.5 Shapelets

Shapelets are sub-sequences of Time Series that are highly discriminating for the recognition of different classes. In our case, we employed shapelets in a classification task on the target genre variable, using the dataset obtained from the *SAX* approximation as the original data was too computationally demanding. First, we trained a basic *DummyClassifier* classifier in order to have a starting point for comparison. As could be expected, the results are not satisfactory as only the *j-idol* class is predicted with a precision value of 0.04 and an *accuracy* of 0.0425.
We then moved on to the search for shapelets within the data. We extracted 7 shapelets of length 10, this was obtained via the function *grabocka params to shapelet size dict*, using as parameters n ts = n ts, ts sz = ts sz, i.e. number and size of Time Series within *X train*, n classes = n classes, i.e. different classes within *y train*, l = 0.1, r = 1. As for the construction of the model via ShapeletModel, we set the parameters optimiser = 'sgd', weight regularizer = 0.1 and max iter = 100. The different shapelets obtained are shown in the figure 2.19.

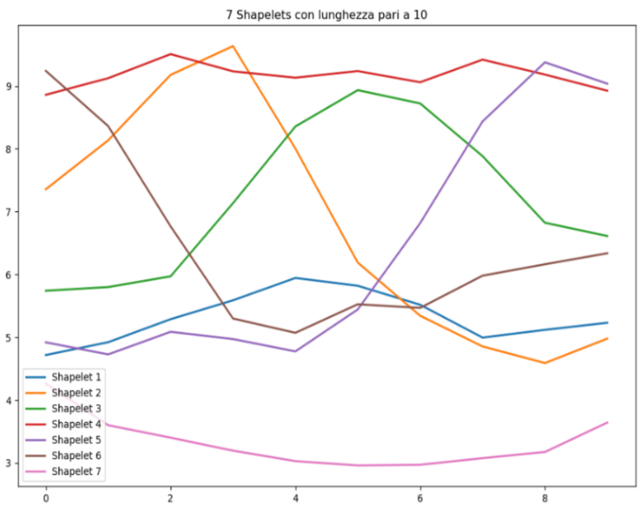

Figure 2.19 Shapelets with length=10

Despite the use of this model, the results obtained are not satisfactory even though they are significantly better than the basic model, as only the heavy-metal and honky-tonk classes were not predicted. The *accuracy* of the model is 0.1085 and the *precision* values range from a minimum value of 0.06 for the *world-music* class to a maximum value of 0.15 for the *songwriter* class.
We exploited shapelets by applying them to the *KNN* and *Decision Tree* classifiers as well, hoping to obtain better results for the classification task. We used the transform function, which generates the shapelet (n_ts, n_shapelets) transformation on *X_train*. For tuning the parameters, we again performed a *Random Serach*. We trained the *KNN* using the following values as a set of parameters: 'weights': 'distance', 'p': 2, 'n_neighbors': 50, 'metric': Euclidian, resulting in an *accuracy* value = 0.109, which is better than in the previous case, and unlike before, all classes are now predicted.
For the *Decision Tree*, the best parameters resulting after the tuning operations are as follows: 'max_depth': 6, 'criterion': 'gini'. We obtain an accuracy value = 0.088 and a non-prediction of the emo and world-music classes.
The *ROC curves* of both models are shown in the next page.

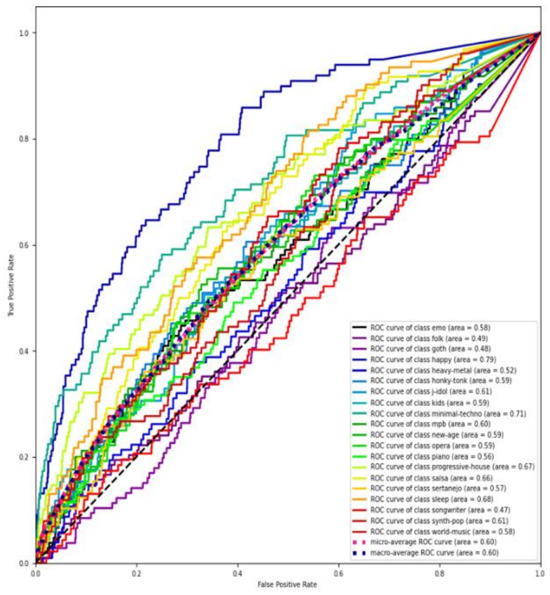

Figure 2.20 KNN ROC Curve

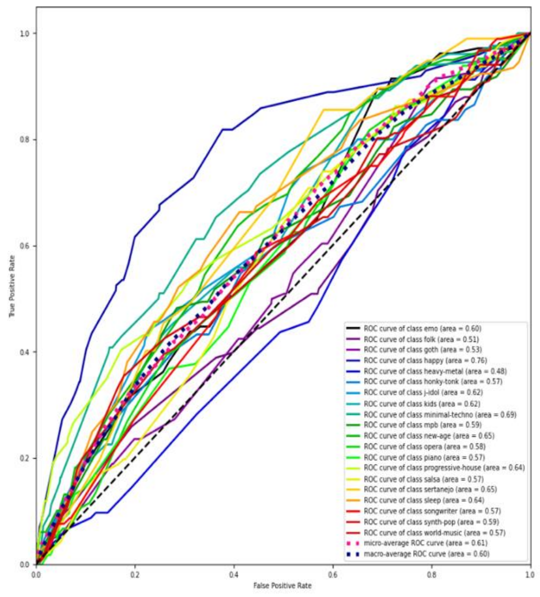

Figure 2.21 Decision Tree ROC Curve

In conclusion, we can say that despite the search for the best parameters, the performance of the *KNN* is slightly better than that of the *Decision Tree* in terms of accuracy. No classifier is satisfactory for the correct classification of the target genre variable.

2.6 Comparison of Shapelets and Motifs

To make this comparison, we decided to further reduce the sample on which we had calculated the *motifs* from 100 to 20 time series, 1 for each genre, and compared their indices in the dataset approximated with *SAX*. Using the indices as position indicators in the data, we calculated the Euclidean distance between the arrays of shapelets and those of motifs. This analysis showed that the shapelets do not coincide with the motifs, this could be due to the simplicity of the *Euclidean distance*, which may not be optimal for capturing similarities between the indices.

Chapter 3: Outliers Detection and Imbalanced Learning

3.1 Outlier Detection

The next part of our analysis regards the identification of the top 1% outliers. We decided to apply different techniques of outlier detection, all methods from different families: *Local Outlier Factor* (LOF), *Lightweight On-line Detector of Anomalies* (LODA) and *Isolation Forest.* For all the methods we selected a *contamination factor* of 0.01, because we wanted to identify the top 1% outliers.

3.1.1 LOF (Local Outlier Factor)

The *Local Outlier Factor* is a density-based algorithm. LOF compares the local density of a single point with the densities of its neighbours. The specific point is considered an outlier if the density around it, is significantly different from the density around its neighbours. We applied this algorithm on the *Artists* dataset. First, we removed the categorical features of *name* and *genres*, so we were left with only *popularity* (of the artist) and *followers*, we than normalized the data using a *Standard Scaler*. As parameters we selected *Euclidean distance*, *contamination* of 0.01 and 50 as *number of neighbours*. To visualize the outliers, we adopted the *Principal Component Analysis* (PCA) and we plotted it in a bidimensional space to observe the inliers and the outliers (Figure 3.1). In the plot in the image, we can observe how a cluster has formed, that has this line shaped form. Given that we are just using *popularity* and *followers* as features, it may be caused by a relation between the two variables, higher the popularity, higher the followers and vice versa. The total number of outliers that the algorithm found was 287, and we decided to remove them all.

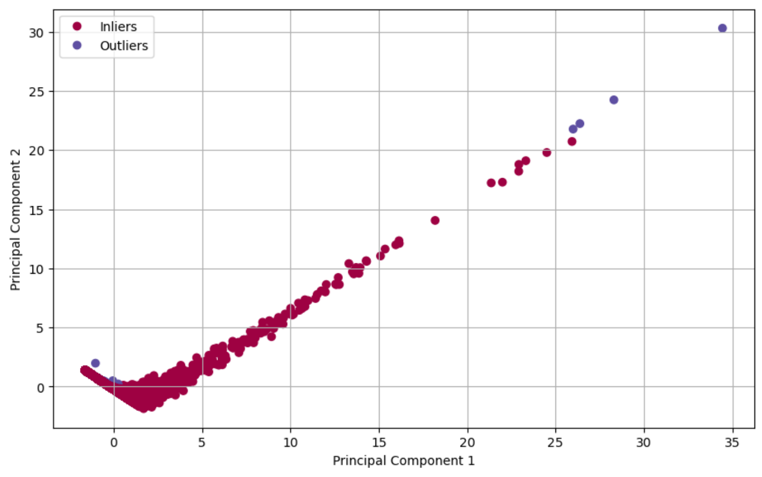

Figure 3.1: PCA of Artists with LOF

Then we applied two different algorithms on the Tracks dataset, in this way we were able to compare them. To begin with, we remove all the attributes that were categorical and other attributes that we thought were not so important for our analysis we remained with the following features:
duration_ms, explicit, popularity, danceability, energy, loudness, speechiness, acousticness ,instrumentalness, liveness, valence, tempo, tempo_confidence, time_signature_confidence, key_confidence, n_beats.

3.1.2 LODA

LODA is an ensemble method; it combines multiple iterations of the Histogram-Based Outlier Score (HBOS) algorithm. By leveraging multiple one-dimensional histograms, LODA can efficiently analyse large datasets and detect patterns and anomalies that single histograms might miss. We applied the algorithm with a *contamination score* of 0.01, and we plotted it in bidimensional space using PCA (Figure 3.2). The algorithm was able to identify 896 outliers.

3.1.3 Isolation Forests

Isolation forests is a model-based algorithm. The idea is to build a forest of trees by randomly selecting dimensions and values to split the dataset. These were the parameters we chose to run the algorithm: (*'n_estimators'*: 100, *'max_samples'*: 'auto', '*contamination'*: 0.01, '*max_features'*: 1.0). Also, in this case we got 896 as number of outliers. We plotted in bidimentional space using PCA (Figure 3.3)

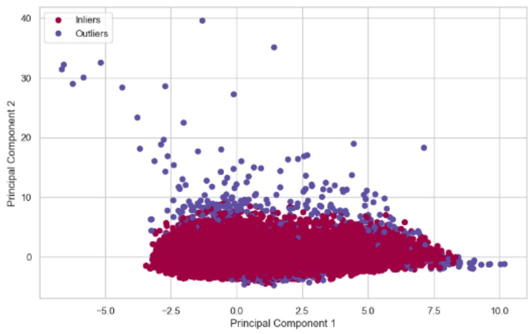

Figure 3.2: PCA of Tracks with LODA

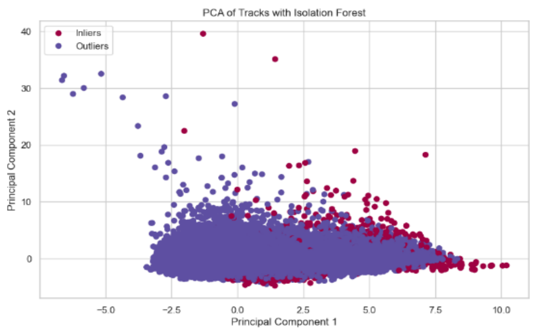

Figure 3.3: PCA of Tracks with Isolation Forest

We added two columns on the original dataset where the values of the Loda and Isolation Forest classified a certain record as outlier or inlier, we compared them, and found that that out of the 896 detected by both the algorithm, only 527 records were classified as outliers by both Loda and Isolation Forest, so we decided only to drop this common records.

3.2 Imbalanced Learning

In this paragraph we have analyzed our dataset and the variable *explicit* as a target to find if it is unbalanced. The variable presents a considerable imbalance in favor of class 0 which represents non-explicit songs and at the expense of class 1 which represents explicit ones. In particular, the classes account for 92% and 8% of the songs in the dataset, respectively.
We divided our dataset into 80% training set and 20% test set and we used a decision tree to solve the unbalanced classification task. We found as best parameters, after performing a Grid search*, {'max_depth': 20, 'min_samples_leaf': 3, 'min_samples_split': 2}.*
The results obtained by the classifier were satisfactory with regard to class 0, as can be seen from Table 3.1, we found a good accuracy value of 0.89, at the same time the precision and f1-score values were excellent as same as the recall's value.
On the contrary, the values of the metrics for class 1 were low, this is due to the severe imbalance in our dataset.
In this case, the classifier is biased towards the majority class 0 because it is much easier to achieve high accuracy by predicting the majority class most of the time. This leads to the classifier almost always predicting the majority class and essentially ignoring the minority class 1.
We have used additional techniques, which we will delve into in the following subparagraphs, such as undersampling and oversampling to get a more in-depth analysis and to try to get better results.

3.2.1 Undersampling Techniques

Undersampling is a technique used to address class imbalance by reducing the number of instances in the majority class to match the number of instances in the minority class. The methods of Random Under Sampler, Tomek link and Edited Nearest Neighbours were used for our analyses.

Random Under Sampler

The first equalized the samples of the classes at 5564 and obtained an accuracy of 0.73, lower than that found previously, as far as class 0 is concerned the other metrics decreased except for the precision which increased to 0.99. About class 1 the metrics have all increased apart from precision as can be seen in Table 3.1.

Tomek Link

The second method used decreased the class 0 samples to 59732 from 62315 while the class 1 samples of 5564 remained unchanged. We have obtained an excellent accuracy value of 0.93, of the high values for the other metrics for class 0, as far as class 1 is concerned the precision is good, equal to 0.75 but the recall and the F1-score remain low as can be seen from Table 3.1.

Edited Nearest Neighbour

The third method further reduced class 0 samples to 49358 and left class 1 samples unchanged. We have thus obtained an accuracy equal to that of the previous method, 0.93, moreover the metrics of class 0 had satisfactory values while those of class 1 remain poor except for the precision of 0.84.

3.2.2 Oversampling Techniques

Oversampling is a technique used to address class imbalance by increasing the number of instances in the minority class to match the number of instances in the majority class. For our analyses, we used the methods of Random Over Sampler, SMOTE, and ADASYN.

Random Over Sampler

The first method increased the samples of class 1 to 62315 while leaving the samples of class 0 unchanged and obtained an accuracy of 0.88. For class 0, the recall was 0.93, the precision was 0.94, and the F1-score was 0.94. As for class 1, the metrics were unsatisfactory with a recall of 0.29, a precision of 0.28, and an F1-score of 0.28.

SMOTE

The second method maintained the number of class 0 and class 1 samples equal to the previous method. This resulted in an accuracy of 0.80. For class 0, the recall decreased, while precision remained high. For class 1, there was a significant increase in recall, although precision did not improve at all.

ADASYN

The third method increased the class 1 samples to 60,737 from 5,564 while keeping the class 0 samples unchanged. This led to an accuracy of 0.81. For class 0 and class 1, the metrics were similar to those obtained with SMOTE.

3.3 Results

In this analysis, we addressed the class imbalance in our dataset, where 92% of the songs were non-explicit and only 8% were explicit. Using a decision tree classifier with optimal parameters, we achieved high accuracy, but metrics for class 1 were all low due to the imbalance so the value of accuracy was not a useful metric as it was greatly influenced by the majority class.
We then implemented various undersampling and oversampling techniques.
Among undersampling methods, Tomek Link and Edited Nearest Neighbours obtained the highest values for precision while the Random Under Sampler achieved the highest recall for class 1. Tomek Link improved class 1 precision to 0.75, and Edited Nearest Neighbours increased it to 0.84, but both methods had low recall and F1-scores for class 1.
Oversampling techniques like Random Over Sampler, SMOTE, and ADASYN increased class 1 recall but did not significantly improve precision or F1-scores. The last two techniques achieved the highest recall for class among oversampling methods.
In conclusion, we obtain slightly better performances using Tomek link and Edited Nearest Neighbour compared to the initial data, but since the values of the metrics were already very high, we were not able to significantly improve them even if as regards minority class 1 we obtained much higher results although not entirely satisfactory.

As shown in Table 3.1, the results of various metrics for both classes demonstrate the impact of each technique. Additionally, Figure 3.4 illustrates the PCA results after applying methods of undersampling and oversampling, providing a visual representation of how the data distribution changes.

Technique	**Accuracy**	**Recall (class 0)**	**Recall (class 1)**	**Precision (class 0)**	**Precision (class 1)**	**F1 Score (class 0)**	**F1 Score (class 1)**
Initial unbalanced data	0.89	0.94	0.27	0.93	0.30	0.94	0.29
Random Undersampler	0.73	0.71	0.94	0.99	0.22	0.83	0.36
Tomek Link	0.93	1.00	0.16	0.93	0.75	0.96	0.26
Edited Nearest Neighbour	0.93	0.99	0.34	0.93	0.84	0.96	0.49
Random Oversampler	0.88	0.93	0.29	0.94	0.28	0.94	0.28
SMOTE	0.80	0.83	0.48	0.95	0.20	0.89	0.29
ADASYN	0.81	0.84	0.47	0.95	0.21	0.89	0.29

Table 3.1: Decision Tree performances on the unbalanced *explicit* variable.

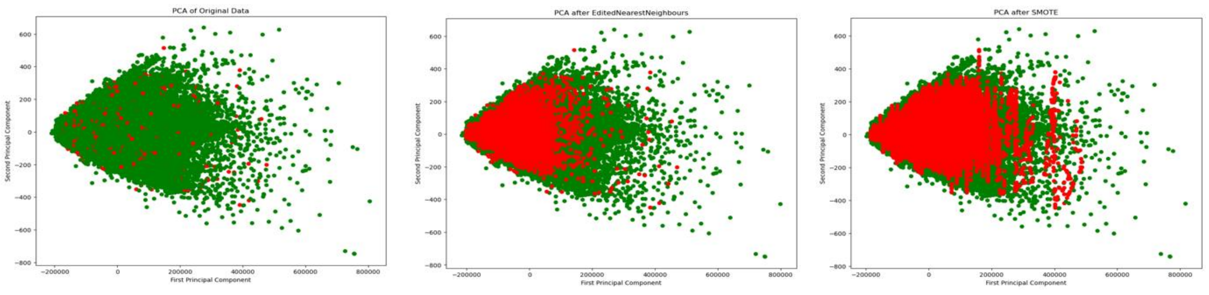

Figure 3.4: Comparing PCA of original data, the undersampling algorithm ENN and the oversampling algorithm SMOTE. Green points represent the majority class 0 and red ones the minority class 1.

Chapter 4: Advanced Classification and Regression

In this chapter we are going to present the results obtained on advanced classification and regression task. We used the pre-processed Tracks tabular dataset for classification and a joined version of the Tracks and Artists Tabular dataset for Regression. We removed all the categorical features, and we normalised using the StandardScaler.

4.1 Advanced Classification

In this section we are going to present the advanced classification we performed with *Explicit* as target variable. We know that the variable is highly unbalanced and for this reason we can't rely just on the accuracy of the model but we must look at all the results of the classification of the 2 classes.
The following algorithms will be presented: *Logistic Regression, Support Vector Machine, Neural Networks, Ensemble Methods and Gradient Boosting Machines.*

4.1.1 Logistic regression

For our task, we first used logistic regression as the classifier, which aims to produce an output that can be interpreted as the probability of a given data point belonging to a certain class, with this probability ranging from 0 to 1.
Initially we performed the logistic regression with the default parameters of the classifier on the explicit variable, we got high results regarding the accuracy of 0.9149 and the metrics regarding the majority class 0, on the contrary we did not get good results for the class 1 metrics, this due to the imbalance of our dataset.
We then tried to use different configurations of the parameters, since our dataset is unbalanced as seen in chapter 3, we set the class weight parameter to *balanced* to handle it. The parameter adjusts the weights inversely proportional to the class frequencies in the input data. This means that the logistic regression algorithm pays more attention to the minority class by increasing its weight, thus balancing the influence of each class during training. We achieved good results for the metrics of class 0 but lower than the previous ones on the contrary, the values of the metrics for class 1 have increased with the exception of precision, the accuracy decreased significantly to 0.74.
Another configuration used was with a high C, the parameter regulates regularization by penalizing large coefficients to prevent overfitting. A smaller C increases regularization, favouring simpler models with higher bias and lower variance. In contrast, a larger C reduces regularization, enabling the model to fit the training data more closely, potentially increasing variance while reducing bias. This adjustment can help capture more signal in the data, particularly beneficial given logistic regression's stability with standard features. We obtained results equal to those with the default parameters this may be since the model is already well regularized with the default parameters making the increase in C insignificant.
Finally, we performed a GridSearch to find the best parameters to get the highest recall because it is a better metric to analyse when the dataset is highly unbalanced, the parameters found, *{'C': 1, 'class_weight': 'balanced', 'penalty': 'l2', 'solver': 'saga'}*

, led to a very minimal increase in the value compared to the use of the class_weight 'balanced' parameter, obtaining an accuracy of 0.73 and substantially going to match the values of the other metrics as you can see in Table 4.1.
We evaluated our model, with the parameters found by GridSearch, also through ROC curve represented in Figure 4.1. The classifier showed promising performance based on the graph. With an AUC of 0.81 for both classes, it demonstrates good discriminative ability, significantly outperforming random guessing. The curve's steep initial rise indicates the model excels at identifying true positives with a low false positive rate and the almost perfect overlap of curves for both classes suggest balanced performance, treating both classes equally well. This is a positive sign, as it indicates the model isn't biased towards one class.

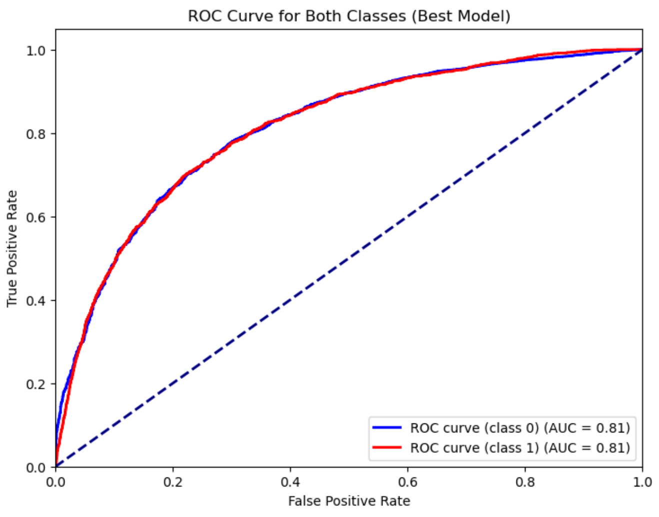

Figure 4.1 : ROC curve of Majority and Minority class.

Parameters	**Accuracy**	**Recall (class 0)**	**Recall (class 1)**	**Precision (class 0)**	**Precision (class 1)**	**F1 Score (class 0)**	**F1 Score (class 1)**
Default	0.91	0.99	0.06	0.92	0.45	0.96	0.11
Class-weight=balanced	0.74	0.74	0.74	0.97	0.21	0.84	0.32
C=100	0.91	0.99	0.06	0.92	0.45	0.96	0.11
GridSearch	0.74	0.74	0.75	0.97	0.22	0.83	0.32

Table 4.1: Performance of the logistic regressor as parameters vary.

4.1.2 Support Vector Machine

The second approach we used was with the Support Vector Machine (SVM). The method operates by identifying the optimal hyperplane that separates data points of different classes in a high-dimensional space. The selected hyperplane maximizes the margin, which is the distance between the nearest points of each class to the hyperplane, enhancing the model's classification performance.

We chose Linear Support Vector Classifier for the classification task. This decision was based on several factors. Firstly, Linear SVC provides a clear and interpretable model by creating a linear decision boundary, which is advantageous for understanding the impact of various features on the classification of explicit content in songs. Secondly, Linear SVC is well-suited for high-dimensional data, such as our dataset, where each song has numerous features. The algorithm handles these high-dimensional spaces effectively without needing complex transformations. Additionally, Linear SVC is computationally efficient, an essential aspect given the thousands of samples and many features involved.

The results obtained with the default parameters showed an overall accuracy of 0.92 indicating strong performance. However, the classifier struggled to accurately identify the minority class 1, reflected in significantly lower F1-score and recall for it. This discrepancy is due to the class imbalance in our dataset.

This imbalance means that accuracy, although high, could be a misleading metric, as a model that always predicts the majority class would still achieve a high accuracy without being useful in identifying the minority class.

To address this challenge, we used GridSearch to optimize for recall. Recall, defined as the proportion of true positives among all actual positives, is particularly critical in this context as it ensures the model effectively identifies as many explicit tracks as possible, reducing the number of false negatives. Moreover, GridSearch allowed us to explore various model parameters, such as class weight balancing (`class_weight='balanced'`), to mitigate the class frequency inequality and improve the model's ability to identify the minority class and this can be seen from the significant increase, compared to the use of default parameters, of the class 1 recall going from 0.02 to 0.75 and from the f1-score going from 0.04 to 0.32, on the other hand the general accuracy was slightly reduced to 0.73, a necessary trade-off to increase the metrics for class 1. Finally, the technique discovered as best parameters: *{'C': 0.001, 'class_weight': 'balanced', 'max_iter': 1000}.*

All the results are visible in Table 4.2 below, even further down, it is possible to visualize Figure 4.2, which is the ROC curve obtained using the previous parameters found by GridSearch. It can be noted that the area under the curve is relatively large and the ROC curve has a value for both classes of 0.81, thus demonstrating the goodness of the model used.

Parameters	**Accuracy**	**Recall (class 0)**	**Recall (class 1)**	**Precision (class 0)**	**Precision (class 1)**	**F1 Score (class 0)**	**F1 Score (class 1)**
Default	0.92	1.00	0.02	0.92	0.42	0.96	0.04
GridSearch	0.73	0.73	0.75	0.97	0.20	0.83	0.32

Table 4.2: Performance of the Linear SVM as parameters vary.

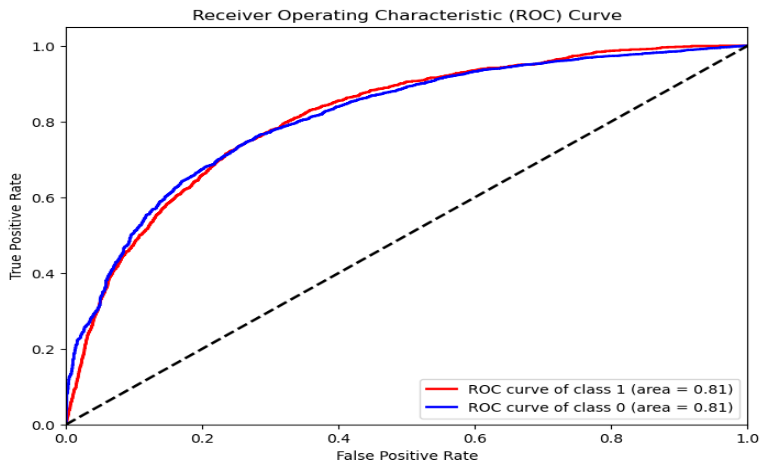

Figure 4.2: ROC curve of Majority and Minority class.

4.1.3 Neural Networks

The next approach for our classification task was Neural Networks. The first thing we tried was analysing the base performance of the base algorithm of a Neural Network on classifying the *Explicit* class. The model reached an accuracy of 0.918 and an F1-score of 0.957. The results on the test set are reported in Table 4.3, alongside the ROC curve results. The model performs well in recognising the class 0, we can' say the same for class 1. Then, a 5-fold cross validation *Randomized Search* was performed to fine-tune the hyperparameters. The search considered the following values {*'hidden_layer_sizes'*: [(128, 64, 32), (128, 32), (64, 32), (100, 100, 100)], learning rate *'alpha'*: [0.1, 0.01, 0.001, 0.002]}. The best configuration found was: {'*hidden_layer_sizes*': (64, 32), '*alpha*': 0.1}. This model achieved an accuracy of 0.919 and F1-score of 0.957, similar to the base model. However, the performance in classifying class 1 was slightly worse. We tried introducing early stopping ('Early_stopping'=True) to mitigate overfitting. The model was able to reach again an accuracy of 0.918 and an F1-score of 0.957, with improved performance in classifying class 1 compared to the tuned model without early stopping. However, it did non surpass the performance of the base model.

Model MLPC	**Class**	**Precision**	**Recall**	**F1-score**	**ROC Curve**
Base Neural Network	0	0.93	0.99	0.96	0.84
	1	0.56	0.14	0.22	0.84
MLPC tuned Early stopping=False	0	0.92	1.00	0.96	0.84
	1	0.64	0.08	0.14	0.84
MLPC tuned Early stopping=True	0	0.92	0.99	0.96	0.83
	1	0.47	0.11	0.17	0.83

Table 4.3 Classification results MLPC

Keras Deep Neural Networks

For Keras Deep Neural Networks, the training set was divided in train set (80%) and validation set (20%), to better understand overfitting effect of the model. We tested various implementation. First, we trained a Neural Network model with 3 dense layers:

- Layer 1: 128 neurons, ReLU activation function
- Layer 2: 64 neurons, ReLU activation function
- Output layer: 2 neurons, Softmax activation function

The model was trained for 200 epochs, with a batch size of 32. The loss function used was *sparse_categorical_crossentropy*, and the optimizer chosen was *'adam'*. During the training of the 200 epochs the *loss* reduced from 0.2449 to 0.0972 and *accuracy* increased from 0.9148 to 0.9619. We plotted the results to visually better understand. In the first plot (Figure 4.3) we can see how there is a consistent decrease, so the model is learning with more accurate predictions during the training, as also visually confirmed by the accuracy plot (Figure 4.4), which shows that the model is becoming better at classifying.

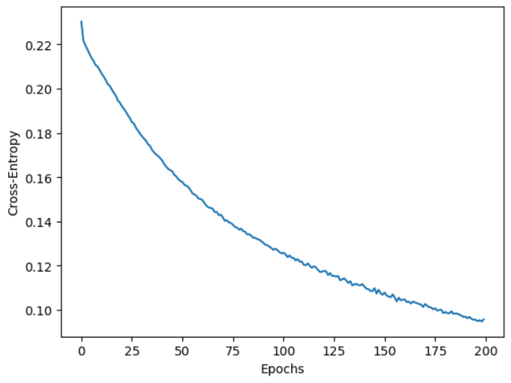

Figure 4.3 Loss through Epochs

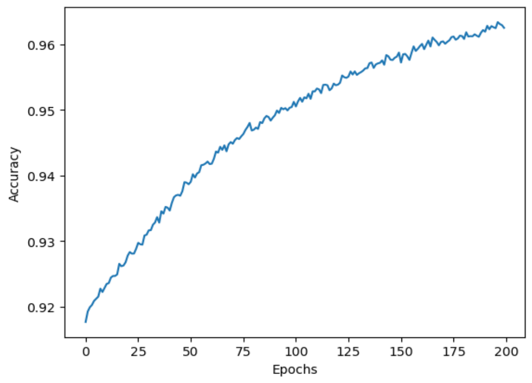

Figure 4.4 Accuracy through Epochs

Then, we evaluated the performance of the model on the test set, and it achieved a *loss* of 0.5132 and an accuracy of 0.8991, which is lower than the training accuracy, suggesting some overfitting.
We did a hyper parameter tuning with a 5-folds *randomized search* that gave us the best parameters: {'optimizer__learning_rate': 0.01, 'optimizer': 'sgd', 'model_ hidden_layer_si zes': (64, 64, 64), 'model__activation': 'tanh', 'epochs': 100}. We trained the model for 100 epochs, with 3 hidden layers of 64 neurons, activation function 'tanh' and 'sgd' as o ptimizer. In the Table 4.4 you can see the results.

To limit overfitting and improve the performance of the neural network model, three different Regularization methods were applied: *Early Stopping*, *L2 Regularization*, and *Dropout*. All the results are in the Table 4.4. After trying different parameters, the best one resulted to be *patience* of 50 epochs for Early Stopping, *kernel regularizer* = l2 (0.005) for the hidden layers of the *L2 Regularization* and a *Dropout* rate of 0.1 between each pair of hidden layers. For the *L2 Regularization* and *Dropout* we applied also an Early Stopping regularization, the L2 Regularization stopped training at epoch 68 and Dropout at epoch 50.

All the models showed a value of Precision and Recall for class 1 much lower compared to class 0. The Roc Curve values remained stable across different models, indicating a similar performance. This consideration plus the low Recall and F1-scores for class 1 suggest that the models are good at identifying true negative, but it struggles with the true positives for class 1.

Model	Accuracy	Class	Precision	Recall	F1-score	Roc
KerasClassifier, 200 epochs, Validation split=0.2	0.8957	0	0.93	0.95	0.94	0.77
		1	0.34	0.27	0.30	0.77
KerasClassifier, 500 epochs, Validation split=0.2, EARLY STOPPING	0.9063	0	0.93	0.97	0.95	0.80
		1	0.40	0.23	0.30	0.80
KerasClassifier, 500 epochs, Validation split=0.2, L2 regularization	0.9177	0	0.92	0.99	0.96	0.84
		1	0.56	0.11	0.18	0.84
KerasClassifier, 500 epochs, Validation split=0.2,, Dropout	0.918	0	0.92	1.00	0.96	0.84
		1	0.66	0.05	0.09	0.84
KerasClassifier 100 epochs, tuned	0.918	0	0.93	0.99	0.96	0.84
		1	0.55	0.16	0.25	0.84

Table 4.4 Classification results Keras Neural Network

4.1.4. Ensemble Methods

Ensemble methods are based on the so-called principle of “Wisdom of the Crowds”, which says that the collective knowledge of a diverse and independent body of people typically exceeds the knowledge of any single individual. In this section we are going to present 3 different Ensemble Methods: *Random Forest, Adaboost and Bagging.*

Random Forest

The first method applied was *Random Forest,* which combines the predictions of multiple decision trees and outputs the class that is the mode of the individual trees’ outputs. We performed hyperparameters tuning using a GridSearch, with 5-fold cross-validation, resulting in the best parameters: *{‘min samples split’: 2, ‘min samples leaf’: 2, ‘max depth’ :30 , ‘criterion’ : ‘entropy, ‘n_estimators: 200}*. In the table 4.5 you can see the tested parameters.

	Tested parameters
N_estimators (n)	[200, 300, 400]
Max_depth	[20, 30, 40]
Min_samples_split	[2, 4, 5]
Criterion	[gini, entropy]

Table 4.5 Tested Parameters for Random Forest

In the Table 4.6 we reported the results of the model on the test set. The overall accuracy is very high, with a score of 0.924 and F1-score of 0.960. However, due to the very unbalanced class distribution, *precision*, *recall* and *f1 score* for the class 1 (Explicit=True) are significantly lower.

Model	Class	Precision	Recall	F1-score	ROC Curve
Random Forest tuned	0	0.93	0.99	0.96	0.86
	1	0.72	0.17	0.28	0.86
ADABOOST	0	0.93	0.99	0.96	0.84
	1	0.53	0.13	0.21	0.84
Bagging	0	0.93	0.99	0.96	0.85
	1	0.65	0.17	0.26	0.85

Table 4.6 Classification results for Ensemble Classification

In the Figure 4.5we can see the impact of different features on the classification, with *speechiness* being the most influential feature, followed by *acousticness* and *popularity*.

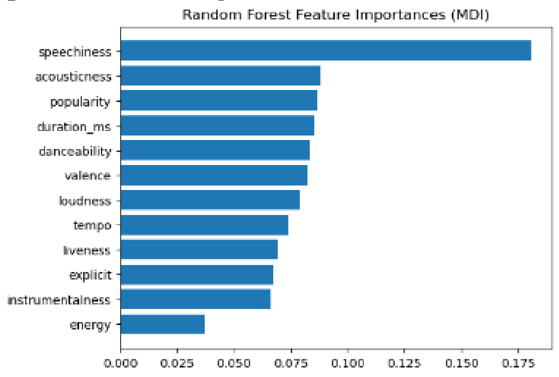

Figure 4.5 Feature Importance Random Forest

The roc curve (Figure 4.6) indicates good performance for both classes, with an area under the curve of 0.86 for each class. The confusion matrix (figure 4.7) shows how the model performs very good on non-explicit tracks (class 0), correctly classifying 15480 instances, but it struggles with explicit tracks (class 1), correctly classifying only 183 instances out of 1412.

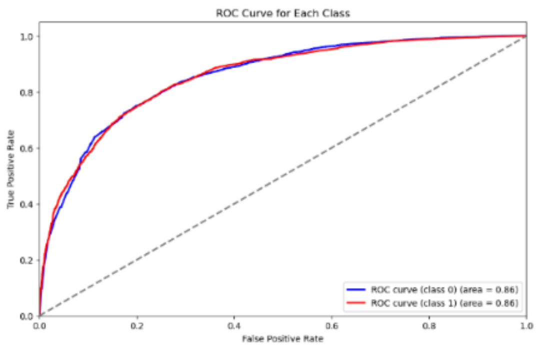

Figure 4.6 Roc Curve Random Forest

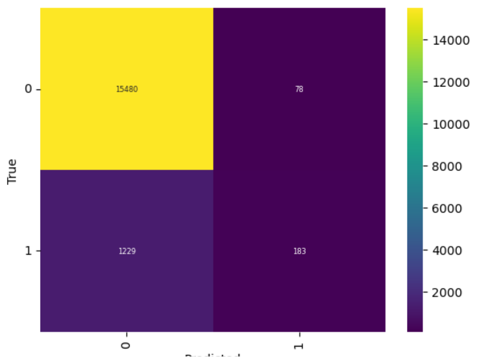

Figure 4.7 Confusion Matrix Random Forest

ADABOOST

The second Ensemble method we analysed was ADABOOST (ADAptive BOOSTing), that, differently from Random Forest, it is an adaptive method, that focuses more on previously misclassified records. We trained the model using a DecisionTree as base classifier and performed a GridSearch with 5-fold cross-validation to find the best hyperparameters. In the table 4.7 you can see the tested parameters.

	Tested parameters
N_estimators (n)	[100, 200, 300]
Learning_rate	[0.01, 0.1, 0.5, 1.0]

Table 4.7 Tested parameters for AdaBoost

The best parameters we obtained from the gridsearch were {'n_estimators'=100, 'learning rate'=1.0}, we used them to train the model. In the Table 4.6 you can see the report of the results, the confusion matrix (Figure 4.8) and the roc curve (Figure 4.9) that show similar results to *Random Forest.*

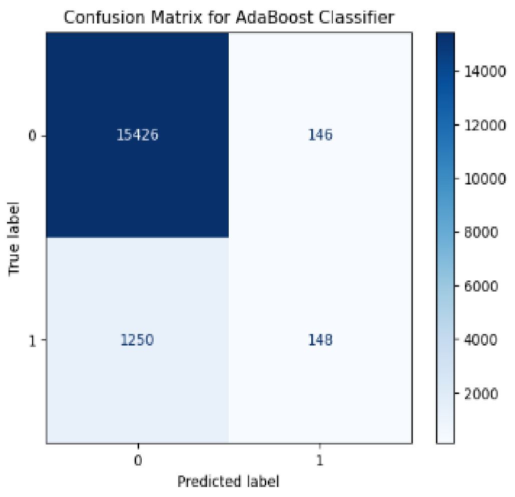

Figure 4.8 Confusion Matrix Adaboost

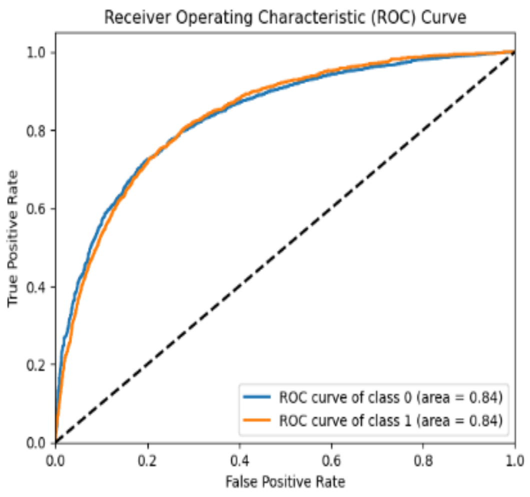

Figure 4.9 Roc Curve Adaboost

Bagging (Bootstrap AGGregatING)

Bagging is the third and last algorithm of the ensemble methods we performed, it generates multiple subsets of the original dataset through random sampling with replace ment. Independent models are trained on the subsets, and their predictions are combined to form the final classification. Using a Decision Tree as the base classifier, we did a GridSearch to tune the hyperparameters (Table 4.8), resulting in {'bootstrap': True, 'boot strap_features': False, 'estimator': None, 'max_features': 1.0, 'max_samples': 1.0, 'n_estim ators': 100, 'n_jobs': None, 'oob_score': False, 'random_state': 0, 'verbose': 0, 'warm_start ': False}.
The model achieved an accuracy of 0.923 and an F1-score of 0.960, detailed results are reported in the Table 4.6. Also, in this case the results of the roc curve and the confusion matrix (Figure 4.10) show similar results to the previous Ensemble methods, with an high accuracy but with consistent issue in classifying class 0.

	Tested parameters
N_estimators (n)	[100, 200, 300]
Max_samples	[0.75, 1,0]
Min_samples_split	[1, 2, 4]

Table 4.8 Tested parameters for Bagging

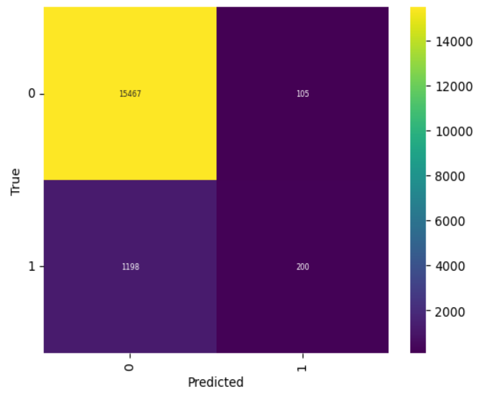

Figure 4.10 Confusion Matrix Bagging

4.1.4 Gradient Boosting Machines

The next group of classifiers used was *Gradient Boosting Machines (GBM*) and their improvements, specifically *eXtreme Gradient Boost (XGBoost*) and *LightGBM.*
We initially implemented the classifier (GBC) with the default parameters of the algorithm and later optimized through a *Randomized search* of parameters. Initially the model achieved an accuracy of 0.918 and F1-score of 0.957, however it is necessary to analyse in detail all the classification results, which will be reported in the Table 4.9 below. The best parameters resulted from the Randomize search were: {'min_samples_leaf': 30, 'max_iter': 200, 'max_depth': 9, 'max_bins': 255, 'learning_rate': 0.1, 'l2_regularization': 0.5}. The optimized model was trained and was able to reach an accuracy of 0.920 and an F1-score of 0.958. Again, detailed results are reported in the Table 4.9. The model with the tuned parameters showed improved performance. The poor results on the class 1 (explicit=True) might be due to the very unbalanced class.

XGBoost

We implemented an XGBoost Classifier with a randomized search of the parameters. The optimal parameters found were: {objective='binary:logistic', reg_lambda=1.0, reg_alpha=1.0, n_estimators=50, max_depth=5, learning_rate=0.3, booster='gbtree', use_label_encoder=False, random_state=42 }. The accuracy achieved was 0f 0.920 with an F1-score of 0.958, which is quite similar to the tuned GBC model. In the figure 4.11 we wanted to compare how the accuracy of the models changes with the number of estimators. The two models seem to have a very similar steady improvement as the number of estimators increases, however the XGBoost demonstrates an advantage.

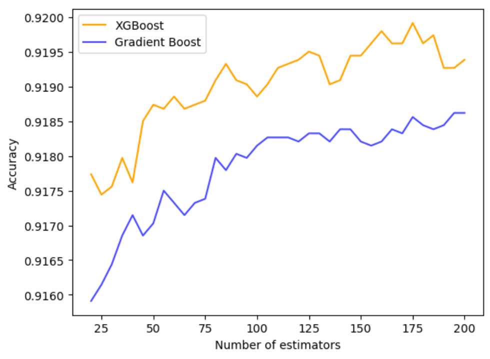

Figure 4.11 Accuracy, XGBoost vs Gradient Boosting

LightGBM

Finally, we implemented a LightGbM Classifier with a Randomized search of the parameters. The model was trained using the optimal parameters found: { 'num_leaves': 50, 'max_depth': 10, 'learning_rate': 0.1, 'n_estimators': 200, 'subsample_for_bin': 200000, 'reg_alpha': 0.0, 'reg_lambda': 0.1, 'min_child_samples': 20 }. The model achieved an accuracy of 0.922 and an F1-Score of 0.959, detailed results can be seen in table 4.9.

Model	Class	Precision	Recall	F1-Score	ROC AUC
GBC	0	0.92	0.99	0.96	0.84
	1	0.57	0.11	0.18	0.84
GBC tuned	0	0.93	0.99	0.96	0.86
	1	0.60	0.16	0.25	0.86
XGBoost tuned	0	0.93	0.99	0.96	0.85
	1	0.58	0.15	0.24	0.85
LightGBM tuned	0	0.93	0.99	0.96	0.86
	1	0.62	0.19	0.29	0.86

Table 4.9 Classification results for Gradient Boosting Machines

In the following images you can see the Roc Curve of the three models.

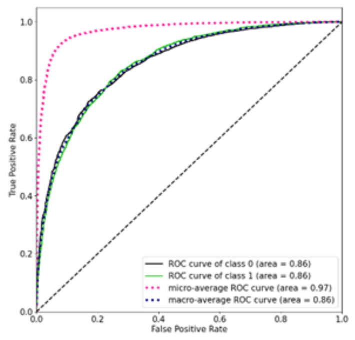

Figure 4.12 Roc Curve GMB

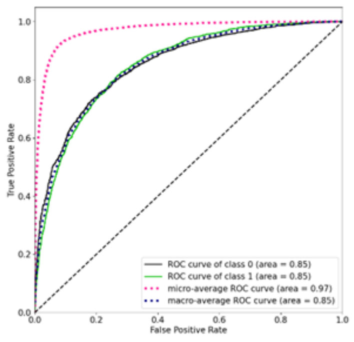

Figure 4.13 Roc Curve XGBoost

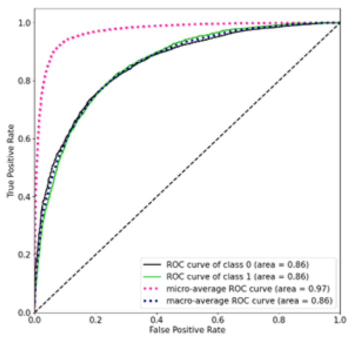

Figure 4.14 Roc Curve Light Boost

Observing the ROC curves, we notice that all models present a high area under the curve (AUC) but observing the results in the Table 4.9 we can see still a significant difference in performance between class 0 and class 1. The Light GMB model is the algorithm that achieves the better results in identifying the *Explicit* class, with a precision of 0.62 and an F1-score of 0.29. This may suggest that Light GBM might be more robust in handling imbalanced datasets of that the optimized parameters are better in this specific classification.

4.1.5 Conclusions Advanced Classification

We can affirm that, due to the very unbalanced class we chose as target variable, the results of the classification task are quite disappointing for most of the algorithms. The best performance was obtained by the *Logistic Regression* algorithm when we set the class weight parameter to *balanced*. In conclusion all the algorithm performed in a very similar way.

4.2 Advanced Regression

Introduction

In this section we are going to present the advanced regression task on the variable *Popularity* from the Tracks Tabular Dataset. To perform this task, we incorporated information from the Artists Tabular Dataset by joining the pre-processed Tracks and Artists datasets using the *name* feature from the Artists dataset and the *artist* feature from the Tracks dataset. This allowed us to integrate track-related data with artist information, resulting in a more comprehensive analysis for our regression task. We obtained a dataset of 66725 observations with the Artists features *followers* and *popularity* added to the features of tracks. To avoid confusion, we renamed the feature *popularity* from the Artist dataset as *popularity_artist* to distinguish it from the popularity of the track. We performed the regression task with *Track_popularity* as our target variable using two non-linear regression algorithms: *RandomForestRegressor* and *SupportVectorRegressor*.

4.2.1 Random Forest Regressor

The first non-linear regression method we used was *Random Forest Regressor*.
We performed a 3-cross-validatiom Randomized Search to find the best hyperparameters. As a result, we obtained {*'n_estimators'*: 300, *'min_samples_split'*: 2, *'max_features'*: 'sqrt', *'max_depth'*: 40, '*criterion*': 'friedman_mse'}. Using these parameters, we trained a Random Forest Regressor model which resulted in a R^2 value of 0.896 for the training and 0.387 for the test set. The Mean Squared Error (MSE) was 47.66 for the training and 281.46 for the test set. The results can be seen in the Table 4.10.

Random Forest Training MSE	47.66
Random Forest Training R^2	0.896
Random Forest Test MSE	281.46
Random Forest Test R^2	0.387

Table 4.10: Random Forest Regression results

We also found quite interesting to extract the importance of the different features for the regression model. In the Figure 4.15 you can see how the variables *follower* and *popularity_artist* were the most important for the regression.

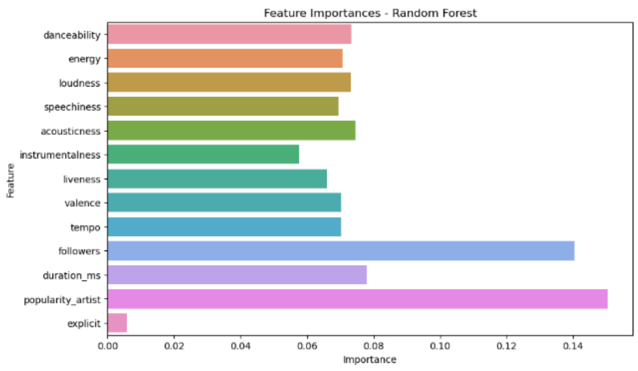

Figure 4.15 Feature importance Random Forest Regressor

4.2.2 Support Vector Regressor

The second model we used was Support Vector Regressor (SVR). Also, in this case we performed a 3-fold cross-validation Randomized Search testing the parameters in the Table 4.11:

Parameters	Tested Values
Kernel	[‘poly, ‘rbf’, ‘sigmoid’]
C	[0.01, 1, 10]
Epsilon	[0.1, 1, 10]

Table 4.11 Tested Parameters SVR

SVR Training MSE	355.94
SVR Training R^2	0.221
SVR Test MSE	365.01
SVR Test R^2	0.204

Table 4.12 1 SVR results

With the best parameters obtained (kernel=’rbf, C=10, epsilon=0.01) we trained the model and obtained an R^2 value of 0.221 for the training and 0.204 for the test set, and a value of the MSE of 335.94 for the training and 365.01 for the test set (Table 4.12).
As you can see in Figure 4.16, in this case the most important features for the model were *popularity_artist.*, while *followers* did not have as impact as important as in the Random Forest Regression.

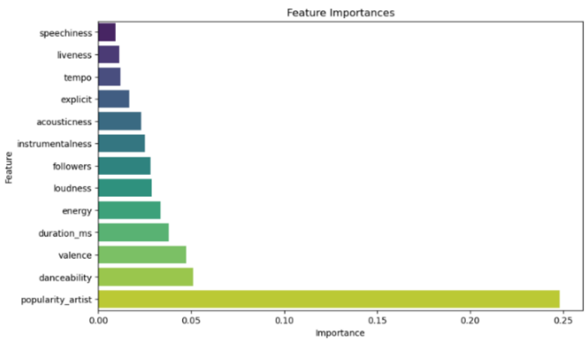

Figure 4.16 Feature Importance SVR

4.2.3 Conclusions

We were quite satisfied with the results of both regressors because both are quite efficient in the prediction of the target variable *Popularity_track.* However *The Random Forest Regressor* showed significantly better results, which can be also visually confirmed in the scatterplots (image). The plot of the Random Forest Regressor’s results shows how the data point appear slightly closer to the prediction line, compared to the Support Vector Regressor.

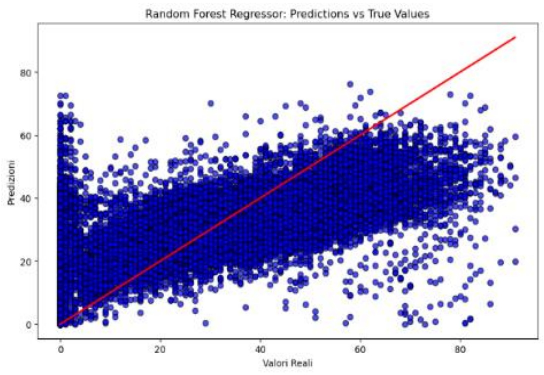

Figure 4.17 Scatterplot Predicted vs True Values for Random Forest Regressor

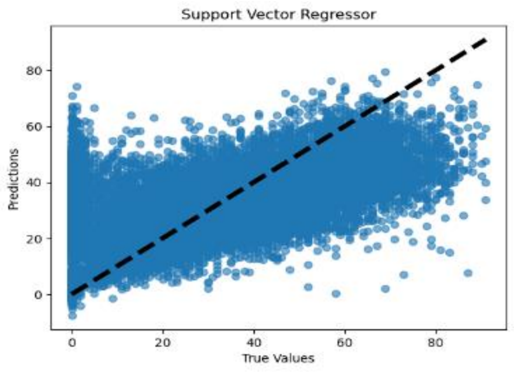

Figure 4.18 Scatterplot Predicted vs True Values SVR

Chapter 5: Explainability

In the final part of our analysis, we concentrated on explaining the results from one of the classification tasks discussed in Chapter 4. We selected the logistic regression classifier with the optimal parameters: *{'C': 0.1, 'class_weight': None, 'penalty': 'l1', 'solver': 'saga'}*. This choice was based on the balance between the classifier's excellent performance and its efficiency in training and prediction. For the analysis we used two different methods: LIME and SHAP.

5.1 LIME

LIME, Local Interpretable Model-agnostic Explanations, works by approximating the model locally with an interpretable model, such as linear regression, to provide insights into how the features contribute to the prediction for a particular instance.
LIME has been applied to explain our model's prediction regarding whether a song is explicit or not. The prediction probabilities can be seen in Figure 5.1, they indicate a high confidence of 95% for the majority class 0, Not explicit songs, and 5% for the minority class 1, explicit songs.

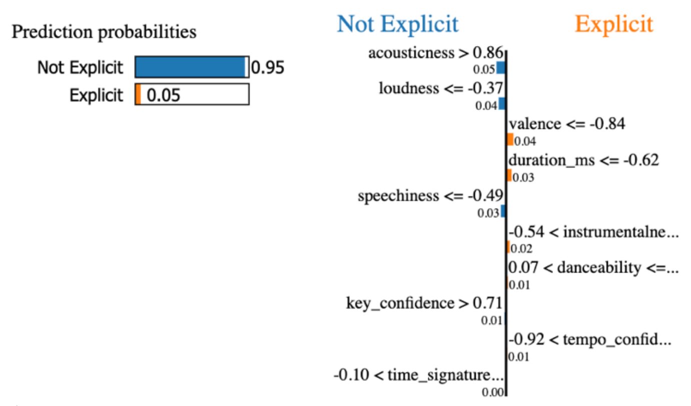

Figure 5.1, LIME output

Moreover, on the right side of it, the Figure 5.1 breaks down the features influencing the prediction. Features pushing the prediction towards "Not Explicit" are listed on the left in blue, while those pushing towards "Explicit" are on the right in orange. Each feature's impact is measured, with stronger values indicating a more significant effect on the prediction.
Key observations from the LIME explanation indicate that certain features significantly influence the prediction of whether a song is explicit or not. High acousticness, greater than 0.86, strongly suggests that a song is "Not Explicit." Additionally, lower loudness, less than or equal to -0.3, also leans towards the "Not Explicit" classification. On the other hand, features like low valence, less than or equal to -0.84, and shorter duration, less than or equal to -0.62, push the prediction towards the "Explicit" class.
These insights from LIME help to understand the logistic regression classifier's decision-making process, enabling better interpretability and trust in the model's outputs.

5.2 SHAP

SHAP, SHapley Additive exPlanations, is a technique designed to explain the predictions of individual instances or observations. It achieves this by calculating the contribution of each feature to the overall prediction, thereby offering a clear understanding of how each feature influences the model's output.

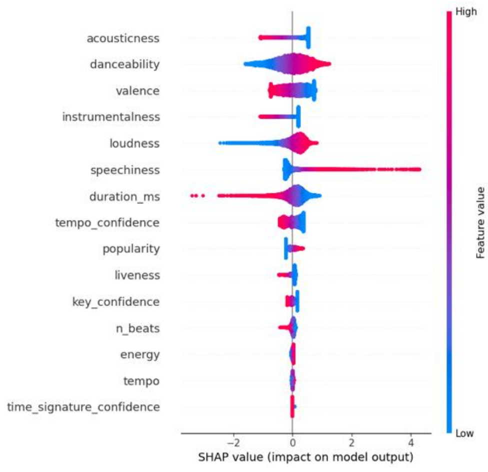

Figure 5.2: SHAP output

The SHAP summary plot, Figure 5.2, illustrates the impact of various features on the predictions of our model designed to classify whether a song is explicit or not. The SHAP values on the X-axis indicate the influence of each feature on the model's output, where positive values push the prediction towards explicit and negative values push it towards non-explicit. Each feature is listed on the Y-axis, and the color gradient from red to blue represents high to low feature values, respectively. For instance, high acousticness, in red, tends to decrease the likelihood of a song being explicit, while higher loudness and speechiness increase it. Danceability, valence, and instrumentalness show varying degrees of influence, with high valence and instrumentalness generally reducing the probability of explicit content. Other features like duration_ms and tempo_confidence also play a role, with longer song durations and higher tempo confidence slightly favoring explicit predictions. Popularity has a mixed impact but slightly leans towards increasing the likelihood of a song being explicit.

5.3 Conclusion

In analyzing the prediction of whether a song is explicit or not, LIME and SHAP offer complementary insights. LIME provides local explanations by approximating the model for a specific instance, highlighting features like high acousticness and lower loudness as indicators of non-explicit songs, while low valence and shorter duration suggest explicit content. SHAP, on the other hand, offers a global perspective by calculating the contribution of each feature across the entire dataset, showing that high acousticness generally reduces explicitness likelihood, whereas higher loudness and speechiness increase it. While LIME focuses on specific instances and SHAP on overall feature influence, both methods identify key features like acousticness and loudness, enhancing our understanding and trust in the model's predictions.