UNIVERSITÀ DI PISA

Data Mining 2 2023/2024 project report

The Spotify Dataset

Professor:
Riccardo Guidotti

Erica Neri
Ilaria Ritelli
Virginia Marletta

May 2024

Contents

Introduction

The goal of this project is to analyze the three datasets provided, two representing tabular data, the third containing time series data. The tabular datasets are called *artists* and *tracks*, and contain information provided by Spotify, respectively about individual artists, and on musical tracks. The time series dataset contains spectral centroids calculated from *.mp3* files corresponding to some of the musical tracks.
This report summarizes our analysis of the three datasets. It is organized in 6 chapters:

- Data Understanding and Preparation;
- Time Series Analysis;
- Outlier Detection;
- Imbalanced Learning;
- Advanced Classification and Regression;
- Explainability.

1. Data Understanding and Preparation

1.1 Artists Dataset

The *Artists* dataset has 30141 entries, described by 5 attributes: *id*, *name*, *popularity*, *followers*, and *genres*. Attributes *popularity* and *followers* are numeric, while *id*, *name* and *genres* are categorical. Specifically, each value of *genres* is a list of strings representing the different musical genres of the corresponding artist.
We analyzed the correlation between the two numeric attributes: the resulting coefficient was very low (0.32), so they do not present any significant correlation.

1.1.1 Data Distribution

We plotted histograms for the two numeric variables (Fig. 1.1): *popularity* follows a slightly left-skewed Gaussian distribution, while *followers* has an extremely left-skewed distribution, with 90% of the records having a number of followers less than approximately 6.5e5, and the remaining 10% with values going up to 1e8.

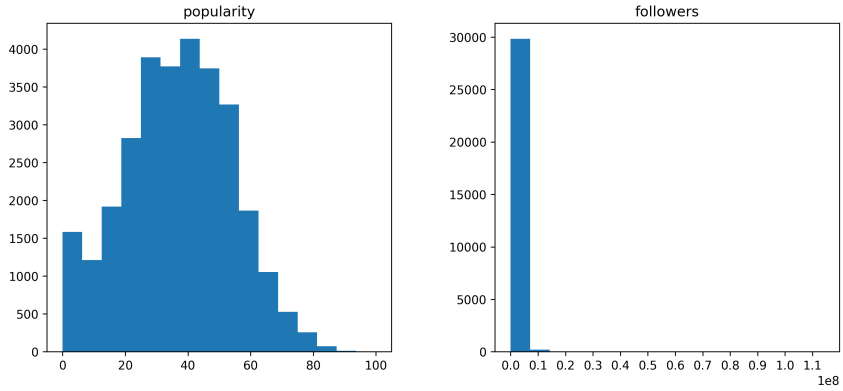

Figure 1.1: Histograms for attributes *popularity* and *followers*.

1.1.2 Data Cleaning

Missing Values Only 6 values were recorded as missing: 1 for *id*, 2 for *name*, 1 for *popularity*, 1 for *followers*, and 1 for *genres*. There is one record with all missing values in all variables, and one record with a missing *name*. Both records were simply removed from the dataset.

Inconsistencies We observed no inconsistencies in the attribute values, as they all fall within the acceptable domain ranges.

Duplicate Data There are only two duplicate records in the entire dataset, out of which one of the copies was removed. There are very few artists with the same *name*, but they are assumed to be homonyms and so are kept as they are.

1.2 Tracks Dataset

The second part of the analysis focuses on the *Tracks* dataset, which is composed by 109547 records described by 34 variables, of which 13 are categorical and 21 are numerical. Tables 1.1 and 1.2 show which attributes are numeric and which are categorical, respectively.

Attributes
duration_ms, popularity, track_number, album_total_tracks, danceability, energy, loudness, speechiness, acousticness, instrumentalness, liveness, valence, tempo, features_duration_ms, start_of_fade_out, tempo_confidence, time_signature_confidence, key_confidence, mode_confidence, n_beats, n_bars

Table 1.1: Numerical attributes.

Attributes
id, name, explicit, artists, album_type, album_name, album_release_date, album_release_date_precision, genre, key, mode, time_signature
disc_number

Table 1.2: Categorical attributes.

1.2.1 Distribution Analysis

The next step was to analyze the distribution of categorical and numerical features. Most tracks belong to albums, as opposed to being part of collections or being singles, and over 100.000 records are marked as the 1^{st} disk of the album. The tracks are mostly not explicit, with a 9/1 ratio over the whole set. The most common mode is 1 (major), and the most common time signature value is 4. The most recorded keys were *7* and *0*, with key *0* being the least used by a considerable margin.

Almost all continuous attributes follow right or left skewed distributions, with the exception of *popularity* and *tempo_confidence*, both of which follow a bimodal distribution.

1.2.2 Correlation Analysis and Variable Elimination

We analyzed the correlation between attributes using Pearson's coefficient and visualizing it on an heatmap (Fig.1.2). Several pairs displayed high correlation. In particular, variables *n_beats*, *n_bars*, *features_duration_ms*, and *start_of_fade_out* are highly positively correlated with *duration_ms*. Furthermore, the pairs *track_number*-*album_total_tracks* and *mode_confidence*-*key_confidence* both show strong correlation with each other.

We decided to drop variables whose correlation coefficients were greater than 0.8, keeping only *duration_ms*, *key_confidence*, and *album_total_tracks* out of the ones mentioned above. The number of features was reduced from 34 to 28. The correlation matrix for the remaining attributes is shown in Fig.1.3.

1.2.3 Data Cleaning

Missing Values The dataset has no missing values.

Inconsistencies We found two records with inconsistent values, both with incorrect values for the attribute *key_confidence*: one with a value greater than 1, the other with a value smaller than 0. For both records, those values were simply clamped to the $[0, 1]$ range.

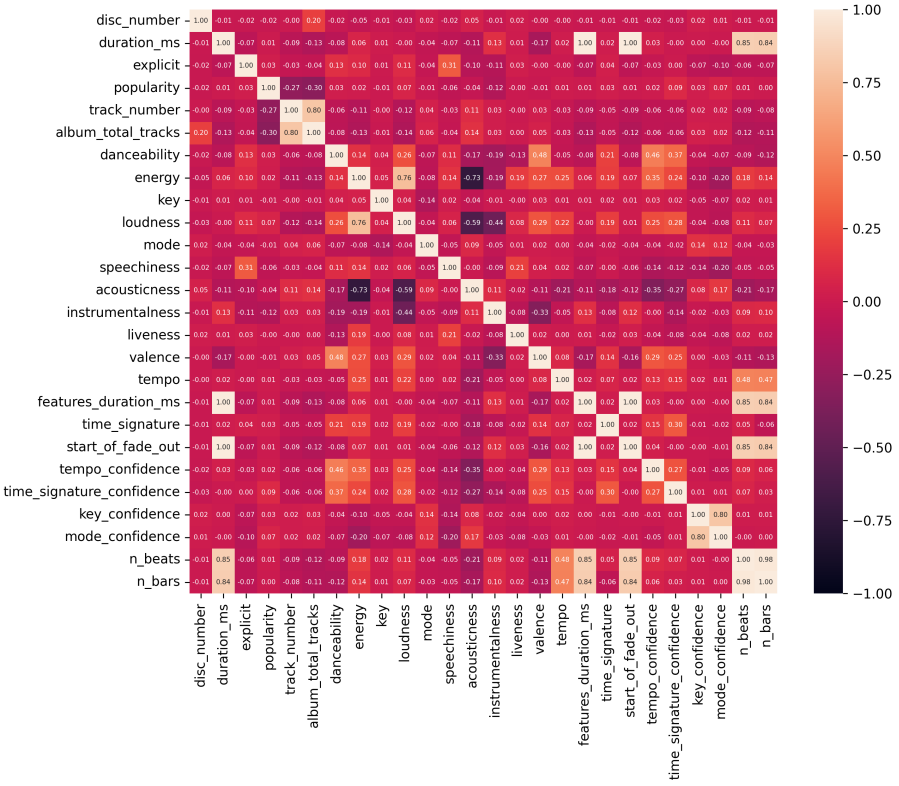

Figure 1.2: Correlation matrix of the *Tracks* dataset.

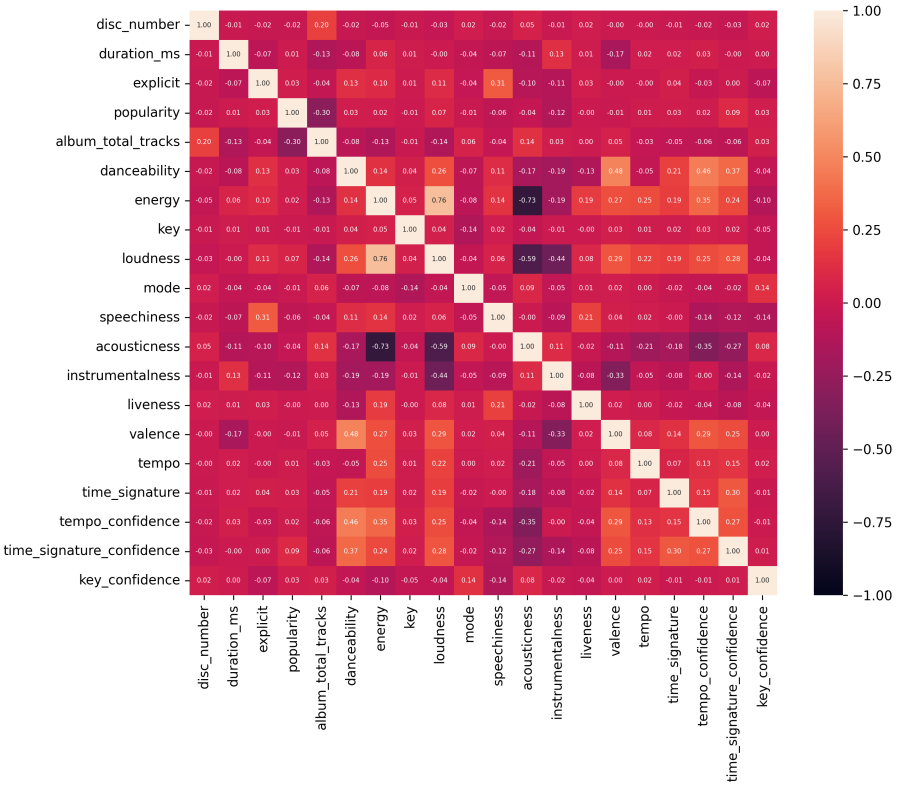

Figure 1.3: Correlation matrix of the *Tracks* dataset, after removing highly correlated variable.

Duplicate Data The dataset has 398 simple duplicate records, which were removed. However, we noticed that many tracks had duplicates with the same *id* and *name*, but weren't marked as duplicates since they had different values only for some attributes (*explicit*, *artists*, *popularity*, and *genre*). We decided to aggregate the information of each group of duplicates into a single record.

Two records were almost identical except for the value of attribute *explicit*: we assumed the record with value *False* to have a missing value, and kept the one with value *True*. In another group of duplicates, one of the records had a different spelling of the artists involved: we replaced it with the correct one, checking that the artist existed in the *Artists* table. For each group, we also calculated the mean popularity, and randomly selected one genre from them. The number of records decreased from 109547 to 89563.

1.3 Time Series Dataset

The provided time series dataset contains 10.000 series, each with 1280 observations. The dataset also contains a label attribute, called *genre*. Time series are equally distributed across labels, with a total of 500 instances per genre. We extracted a sample of the dataset and visualized it to check if any transformations are needed. Some of the time series present in the sample are visualized in Fig. 1.4.

As evidenced by the sample, the time series are unaligned on the y axis, have different amplitudes, and are noisy. We used amplitude scaling to transform all the instances, aligning them and adjusting their amplitude; since the dataset was too big to carry the required tasks, the time series were also approximated using Piecewise Aggregate Approximation, reducing the number of observations from 1280 to 100. The same time series shown above can be seen again in Fig. 1.5, after all transformations have been applied.

This version of the dataset was used for motif and discord discovery, clustering, and classification. We generated another version, transformed with Symbolic Aggregate Approximation, to be used for sequential pattern mining; more details are provided in the dedicated section.

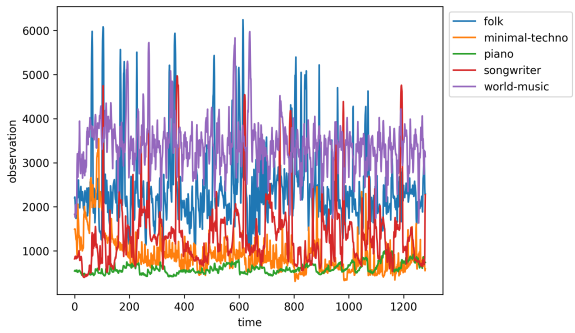

Figure 1.4: A selection of time series belonging to different classes.

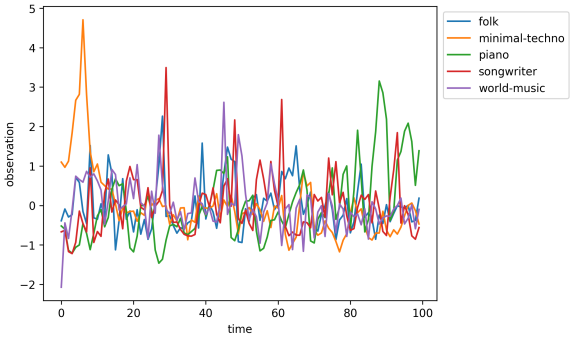

Figure 1.5: A selection of time series belonging to different classes, transformed and approximated.

2. Time Series Analysis

This chapter illustrates how we carried out clustering, classification, motif and discord discovery, and sequential pattern mining on the time series dataset.

2.1 Clustering

We used two clustering methods: (whole) distance-based clustering, and feature-based clustering.

2.1.1 Distance-Based Clustering

First, we tried to cluster the time series using a simple K-Means algorithm. We scaled the dataset using standardization. We used DTW to calculate the distances between time series, constrained using the Sakoe-Chiba band by setting the parameter $window = 0.05$ (which sets the radius of the band to 5% of the original size of the time series). To find a good value for the number of clusters k, we repeated the execution of the algorithm with different values of k and plotting their SSE and silhouette scores (Fig 2.1).

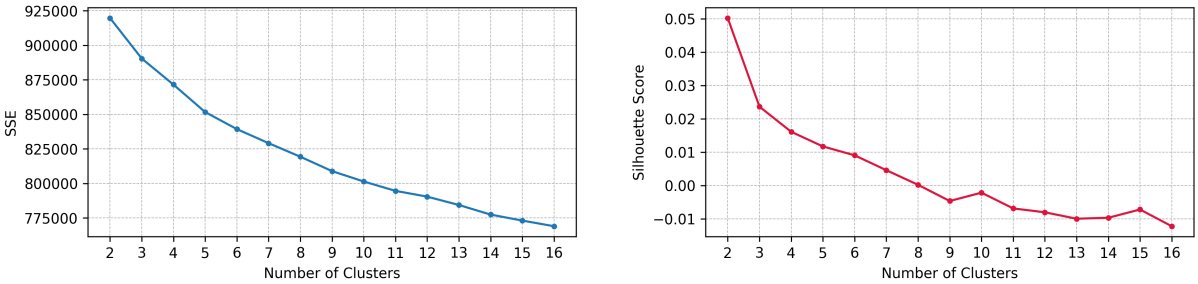

Figure 2.1: SSE and silhouette score for K-Means.

There is no clear "good" choice to make, since both measures simply decrease as the number of clusters increase with no clear elbow in the SSE plot and no peaks in the silhouette score plot. We chose $k = 5$ since it represents a relatively reasonable trade-off between SSE and silhouette score (although the latter is still very low). The result of the clustering is visualized via dimensionality reduction techniques in Fig. 2.2 and 2.3. The SSE and silhouette score of the clustering are 853965 and 0.012, respectively.

As evidenced from the visualizations, the clusters are not well-separated at all. Inspecting the distribution of the *genre* class labels (Fig. 2.4), we can see that each cluster contains the same ratio of each label as the other ones.

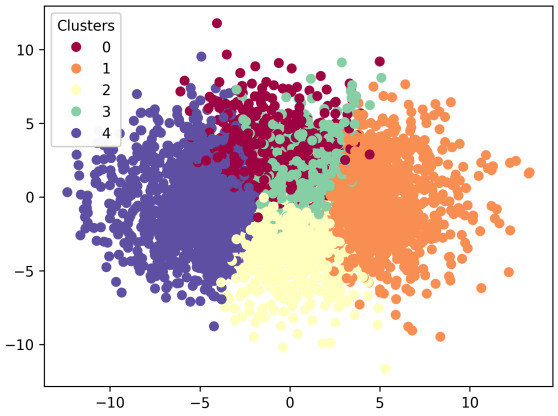

Figure 2.2: PCA visualization of the K-Means clustering.

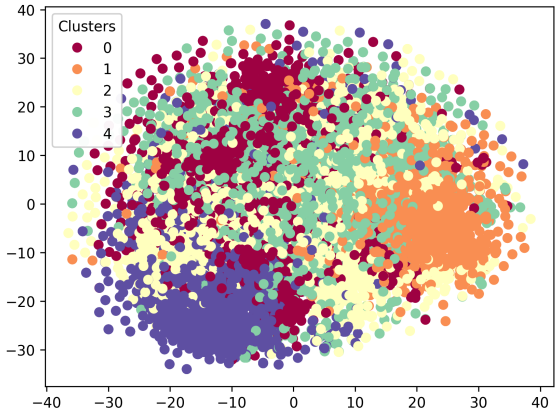

Figure 2.3: t-SNE visualization of the K-Means clustering.

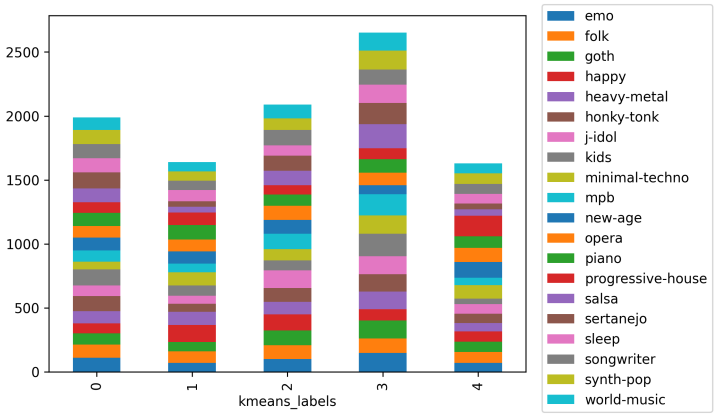

Figure 2.4: Distribution of genres across clusters produced by K-Means.

2.1.2 Feature-Based Clustering

For this method, we extracted the following set of features from each row of the dataset: mean, standard deviation, minimum value, maximum value, and 0.1, 0.25, 0.5, 0.75, and 0.9 quantiles; the transformed dataset was then clustered with DBSCAN, using Euclidean distance. We first ran K-Nearest Neighbors, calculating each point's distance to its k neighbors, and plotting those distances to find a good value of the hyperparameter *eps* (Fig. 2.5 reports the plot obtained for $k = 16$).

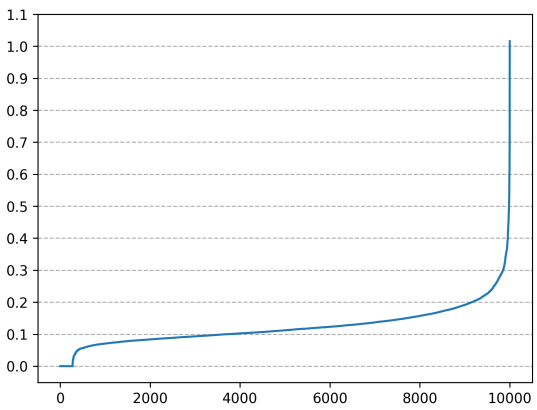

Figure 2.5: Plot of the k-distances of the points ($k = 16$).

We chose $eps = 0.25$, as it corresponds to the elbow in the plot.

To choose a value for *minPts*, we ran DBSCAN multiple times, varying the value of *minPts* between 2 and 1024, increasing in powers of 2. However, only values 2, 4, and 16 actually produced clusters, and only $minPts = 16$ actually yielded a good enough silhouette score. The clustering produced with parameters $eps = 0.25$ and $minPts = 16$ are visualized in Fig. 2.6 and 2.7 (using the original time series dataset); the distribution of the class labels is visualized in Fig. 2.8. The silhouette score is 0.19.

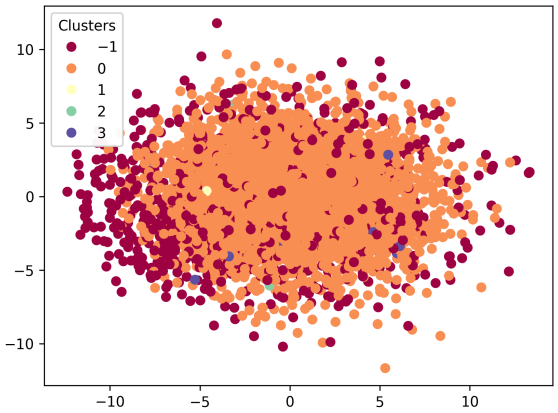

Figure 2.6: PCA visualization of the DBSCAN clustering.

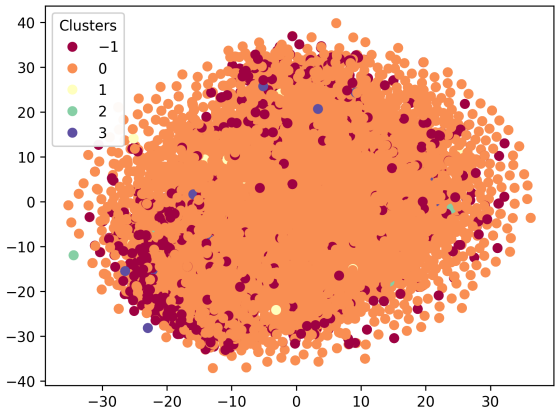

Figure 2.7: t-SNE visualization of the DBSCAN clustering.

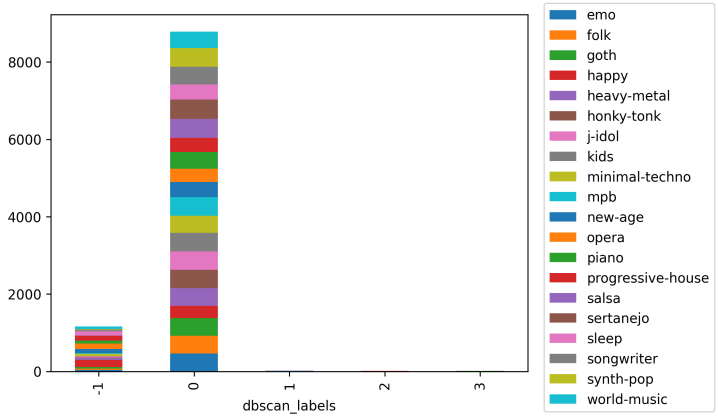

Figure 2.8: Distribution of genres across clusters produced by DBSCAN.

The clustering is made up of a bigger cluster with about 8786 points, three tiny clusters of only about 20 points each (belonging to different genres), and the remaining 1161 points are noise points. Even using this method, the result does not contain any well-separated or meaningful clusters.

2.2 Classification

We tried to solve the task of multi-class classification using the variable *genre* (meaning a total of 20 classes). In this section we compare the following models: K-Nearest Neighbors, shapelet-based classification, and deep learning models (convolutional neural networks, ROCKET). To train and evaluate the performance of each model, we split the dataset into training set (80%) and test set (20%).

2.2.1 K-Nearest Neighbors

We classified the data using K-Nearest Neighbors with both Euclidean distance and DTW. In both cases, we started with a grid search, exploring the following hyperparameters:

- ***n_neighbors***: varying in powers of 2 between 4 and 256;
- ***weights***: *distance* or *uniform*,

and validating each combination using 4-Fold Cross Validation. For DTW, the calculation was constrained again using the Sakoe-Chiba band, setting $window = 0.05$. The best combination for Euclidean distance is $n\_neighbors = 8$, $weights = distance$, with test accuracy equal to 0.12; the best combination for DTW is $n\_neighbors = 256$, $weights = distance$, with test accuracy equal to 0.15. Their respective confusion matrices are found in Fig. 2.9 and 2.10.

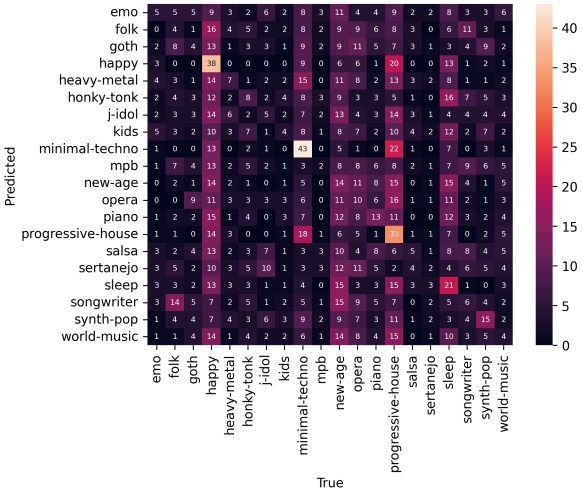

Figure 2.9: Confusion matrix of the K-NN classifier, using Euclidean distance.

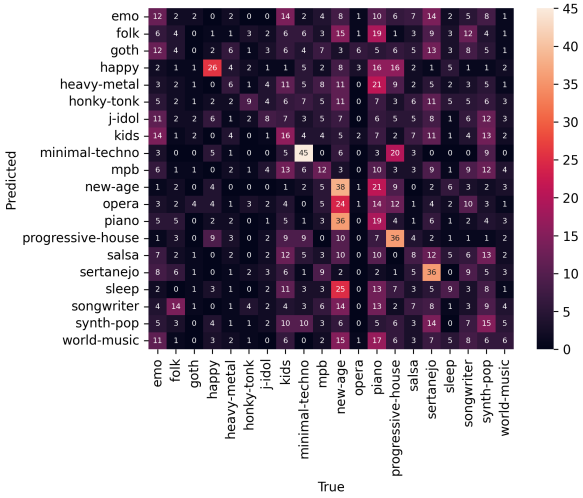

Figure 2.10: Confusion matrix of the K-NN classifier, using DTW.

Even if the improvement in accuracy is not exceptional, DTW is better than Euclidean distance at estimating closeness between time series even if constrained, and produces better predictions overall. However, this general improvement is at the expense of the ability to identify certain classes (such as *goth*, *piano*, *opera*), which report lower precision, recall, and f1 scores than in the Euclidean distance classifier (or even equal to 0), while the same measures increase for the others.

2.2.2 Shapelet-Based Classification

We first extracted shapelets from the time series, setting *max_shapelets* to 200 (which would produce 10 shapelets per class), *min_shapelet_length* to 4, and *max_shapelet_length* to 16. Then, we extracted the distances between training and test set from the set of shapelets found, and used them to train and evaluate a Decision Tree classifier. We again performed a grid search, with 4-Fold CV, and the following hyperparameter grid:

- ***criterion***: *gini* or *entropy*;
- ***max_depth***: varying in powers of 2 between 8 and 512, plus *None* (no limit);
- ***min_samples_split***: varying in powers of 2 between 4 and 256;
- ***min_samples_leaf***: same as the above.

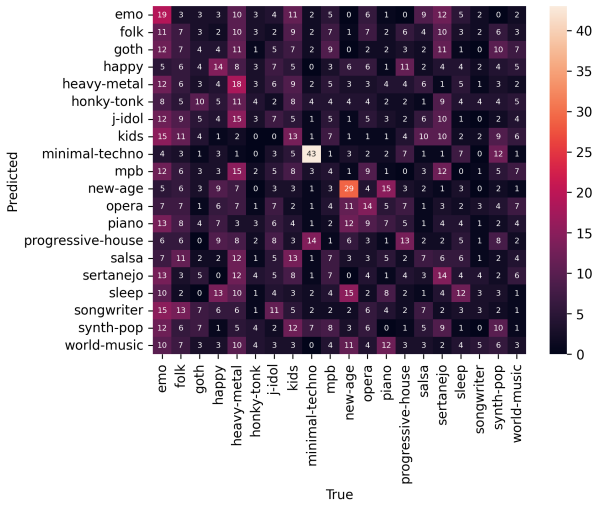

Figure 2.11: Confusion matrix of the shapelet-based classifier (Decision Tree).

The best combination is $max\_depth = 16$, $min\_samples\_leaf = 4$, $min\_samples\_split = 128$, and $criterion = gini$. The accuracy on the test set is equal to 0.12 (the same as the K-NN classifier with Euclidean distance). The confusion matrix of the model is in Fig. 2.11.
As evidenced by the confusion matrix itself, precision, recall, and f1-score are lower for most classes compared to the previous models, and slightly higher for a couple of them (*new age* and *minimal techno*). More analysis about the shapelets themselves can be found in the later section about motif and discord discovery.

2.2.3 Deep Learning Models

The two final models used for classification are a convolutional neural network and ROCKET. Both models use convolutional kernels applied to the time series, but while the first one is an actual neural network with several layers of trainable filters (which can also take a lot of time to be trained, depending on the size of the network), ROCKET has only a single layer with random filters applied to the time series to transform them, and training is done on a linear classifier. We considered it interesting to compare the two and see if there is any big difference in performance.

Convolutional Neural Network

To train this model, the input was reshaped to the appropriate format, and the class labels were one-hot-encoded. We fixed the number of layers and units per layer, and ran a randomized search to find the best activation function, the regularization type and penalty term, the learning rate, and the number of epochs to train the model for, chosen between:

- ***activation_function***: *sigmoid*, *ReLU*;
- ***learning_rate***: 0.1, 0.5, 0.01, 0.05, 0.001;
- ***reg_type***: L1, L2;
- ***reg_lambda***: 0.1, 0.5, 0.01, 0.05, 0.001;
- ***epochs***: 100, 250, 500.

The model uses the Adam-based optimizer. As an ulterior regularization technique, we set dropout with rate 0.3 for each convolutional layer, as well as batch normalization. We also set a decaying learning rate schedule, with parameters $decay\_steps = 100$, $decay\_rate = 0.95$. The

learning rate chosen by the search is 0.001, and the number of epochs is 100. The chosen network has the following architecture:

Layer type	# Units	Activation function	Regularization
Convolutional	16	sigmoid	L2 ($\lambda = 0.01$)
Convolutional	32	sigmoid	L2 ($\lambda = 0.01$)
Convolutional	16	sigmoid	L2 ($\lambda = 0.01$)
Dense	20	softmax	None

Table 2.1: Description of the architecture of the convolutional neural network.

We retrained the model again, this time setting the early stopping callback with parameters *patience* = 10 and *min_delta* = 0.001, and indeed training stopped at 64 epochs instead of the full 100, avoiding eventual overfitting. The test accuracy is 0.17. The learning curves of loss and accuracy of the neural network are reported in Fig. 2.12, and its confusion matrix is reported in Fig. 2.12.

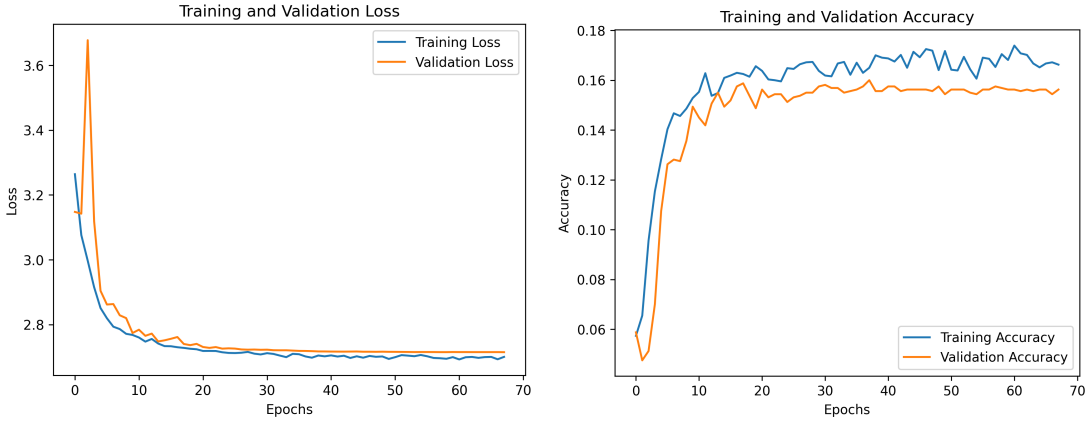

Figure 2.12: Learning curves of the convolutional neural network.

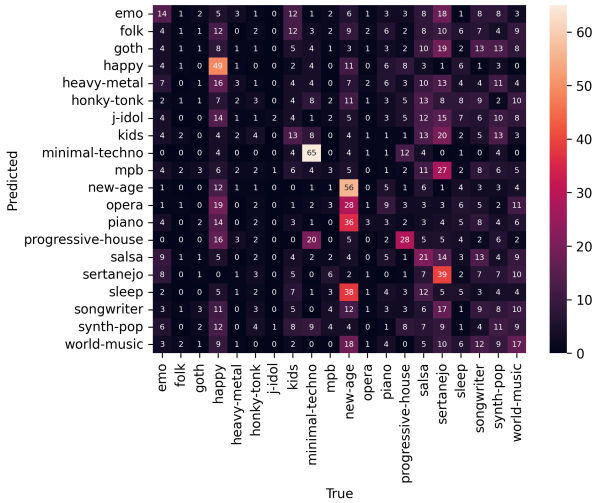

Figure 2.13: Confusion matrix of the convolutional neural network.

Compared to the model with the highest accuracy so far (K-NN with DTW), the neural network has similar metrics for each class, but none of them are zero-valued, meaning that it is at least capable of partially recognizing all of them.

ROCKET

As for all the models above, we started with a grid search with 4-Fold CV, exploring the following hyperparameter grid:

- ***num_kernels***: either 1000, 2000, 4000, or 8000;
- ***max_dilations_per_kernel***: 8, 16, 32, or 64;

The transformer used was *minirocket*, which is much faster than the default *rocket* transformer while still capable of maintaining comparable results. The best hyperparameter combination found is *num_kernels* = 1000 and *max_dilations_per_kernel* = 64; the accuracy of this model is 0.21 (the highest accuracy recorded among all models), and its confusion matrix is in Fig. 2.14. By comparing the confusion matrix with the previous ones, it is clear that ROCKET is the best-performing model; it is also very fast to train, taking less than a minute on average, while the convolutional neural network takes over 3 minutes for worse results, and the others also require much longer training times (or, in the case of shapelets, a long time to extract the shapelets themselves).

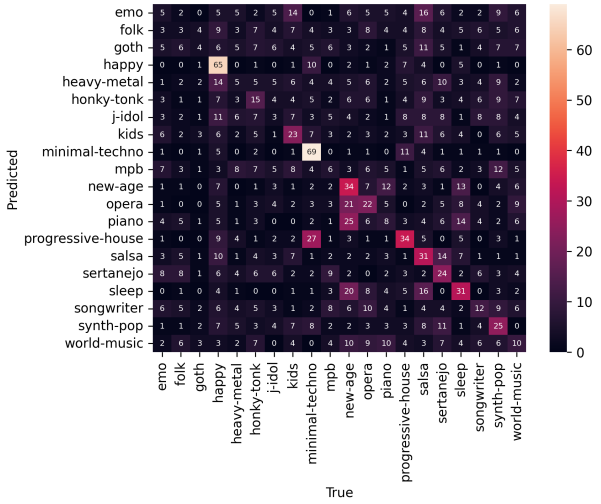

Figure 2.14: Confusion matrix of ROCKET (with MINIROCKET transformer).

2.3 Motif and Discord Discovery

In this section we describe how we extracted motifs and discords from the time series, and compare them to the shapelets found for classification. For each time series, its matrix profile was calculated setting the sliding window to 6, after trying out a few different values (between 3 and 16). The matrix profile was then used to extract 3 motifs and 3 discords from each series.

To visualize them, we first selected the time series from which shapelets were extracted in the previous section, isolating 10 time series per class, and plotted those shapelets along with the motifs and discord of the corresponding series. Some interesting plots are reported in Fig. 2.15 and 2.16: for some time series like those in the figures, the shapelets did have some total or partial overlap (highlighted in purple) with both motifs and discords, and in some cases the shapelet would include multiple motifs/discords at once (such as in Fig 2.16). However, for most of the others there was no overlap at all, and we could not observe any class-specific behaviour relating to the relationship between shapelets and motifs/discords.

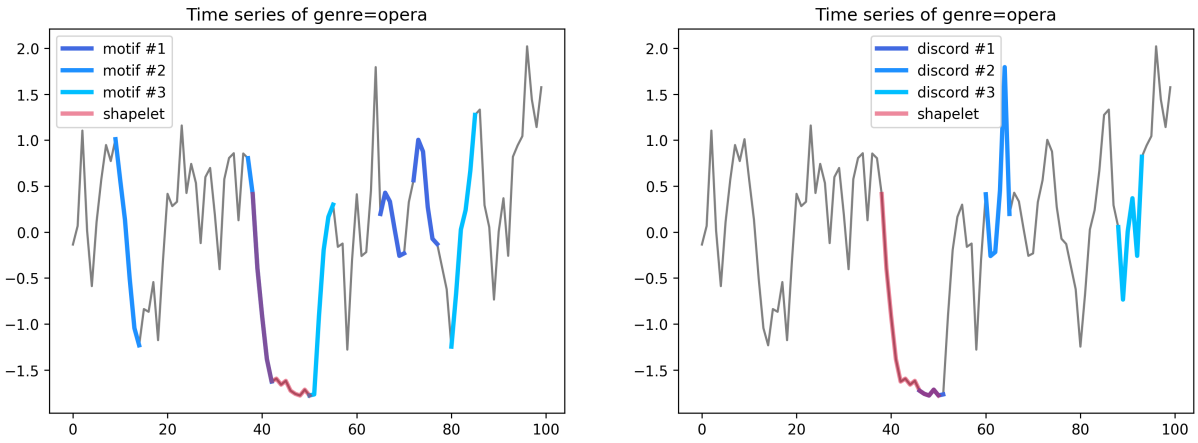

Figure 2.15: Motifs and discords compared to the shapelet extracted from a time series of *genre* "*opera*".

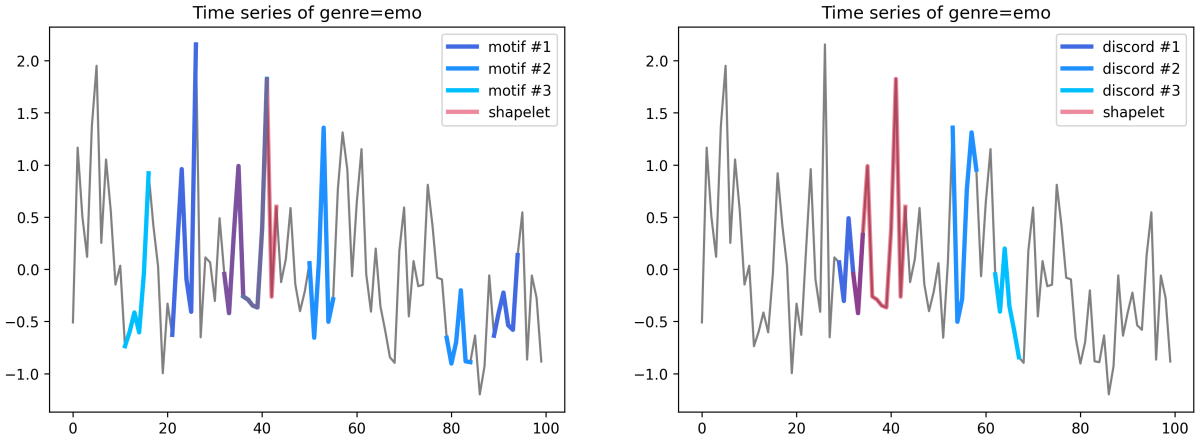

Figure 2.16: Motifs and discords compared to the shapelet extracted from a time series of *genre* "*emo*".

2.4 Sequential Pattern Mining

To transform the dataset into an appropriate format for this task, we used SAX on the time series with scaled amplitudes, setting the alphabet size to 4, and the number of observations to 64. This resulted into a sequential dataset of 10.000 sequences, each with exactly 64 elements of 1 event each. We also tried to produce larger datasets with bigger alphabets and higher number of observations, but the execution of the algorithm used (even if constrained) would have taken too long to obtain any results in a reasonable amount of time.

Frequent sequences were extracted using the Generalized Sequential Patterns algorithm, using time constraints $maxgap = 2$, $mingap = 1$, and $maxspan = 8$. We repeated the execution multiple times, varying the value of *minsup* between 0.1 and 0.5, and recorded the amount of frequent sequences found (Fig. 2.17).
We chose to execute the algorithm with $minsup = 0.2$ since it produces an intermediate amount of sequences. Some of the frequent sequences found (the three most frequent ones for each length, with the exception of 6, since only one sequence was found with that length) are reported in Table 2.2.

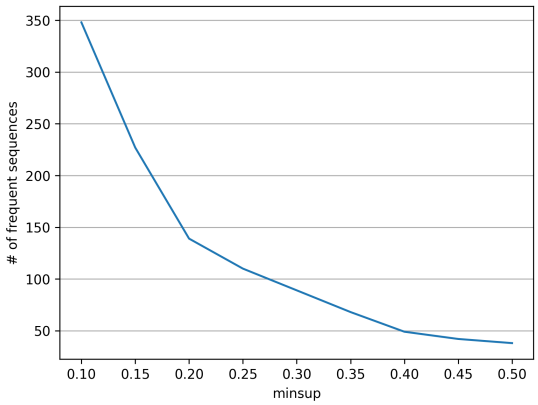

Figure 2.17: Frequent sequences found by GSP for different values of *minsup*.

Sequence Length	**Sequence**	**Support Count**
1	$< \{3\} >$	9992
	$< \{0\} >$	9956
	$< \{1\} >$	8141
2	$< \{3\}, \{3\} >$	8504
	$< \{1\}, \{1\} >$	8039
	$< \{1\}, \{0\} >$	7870
3	$< \{1\}, \{1\}, \{1\} >$	7304
	$< \{1\}, \{1\}, \{0\} >$	6122
	$< \{3\}, \{1\}, \{1\} >$	5819
4	$< \{1\}, \{1\}, \{1\}, \{1\} >$	5488
	$< \{2\}, \{1\}, \{1\}, \{1\} >$	4178
	$< \{3\}, \{1\}, \{1\}, \{1\} >$	4060
5	$< \{1\}, \{1\}, \{1\}, \{1\}, \{1\} >$	3583
	$< \{3\}, \{1\}, \{1\}, \{1\}, \{1\} >$	2698
	$< \{1\}, \{1\}, \{1\}, \{1\}, \{3\} >$	2637
6	$< \{1\}, \{1\}, \{1\}, \{1\}, \{1\}, \{1\} >$	2161

Table 2.2: Top 3 most frequent sequences of each length.

3. Outlier Detection

In this chapter we explain how we identified the top 1% outliers in the tabular datasets, and how we dealt with them. We used three different methods: LOF, LODA, and Isolation Forest.

3.1 LOF

Since LOF is a density-based outlier detection method, its result is sensitive to the chosen hyper-parameter values. After fixing the *contamination* parameter to 0.01 (which sets the appropriate threshold on the decision function to find the top 1% of outliers), we tried different values of *n_neighbors*, since we noticed that the default value would return too few points. The value we chose is $n\_neighbors = 256$, which found 302 outliers in *artists*, and 877 in *tracks*; these outliers are visualized via PCA in Fig. 3.1 and 3.2, respectively.

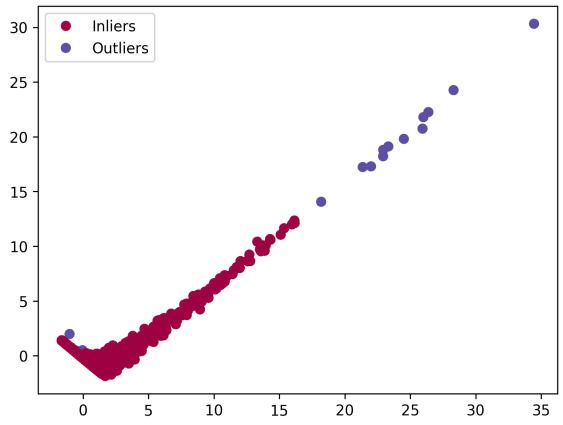

Figure 3.1: PCA visualization of the outliers and inliers found by LOF in the *artists* dataset.

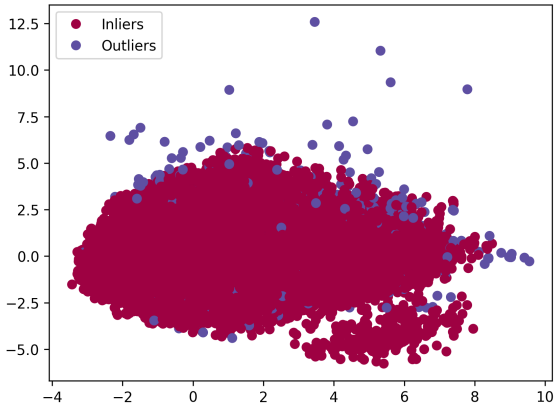

Figure 3.2: PCA visualization of the outliers and inliers found by LOF in the *tracks* dataset.

In the first dataset, the method identified outliers among the points straying from the rest both in the "tail" of the scatter plot, and among some of the further ones closer to the mass in the lower left area. As for the second dataset, the method did manage to select the few points scattered far away from the center, but mostly ignored the small group in the lower right corner.

3.2 LODA

For LODA, we set parameters $bins = auto$, which automatically determines the numbers of bins to discretize the data into, $n\_random\_cuts = 500$, and $contamination = 0.01$. The result found 293 outliers in *artists* and 896 in *tracks*, visualized in Fig. 3.3 and 3.4.
From looking at these representations, we can see how in the *tracks* dataset the method declared as outliers all points outside of the bigger mass in the center, including the small group on the bottom-right (which LOF failed to do). In the *artists* dataset, the outliers are only found among those in the "tail" of the scatter plot, while the points in the bottom-left group are all considered inliers.

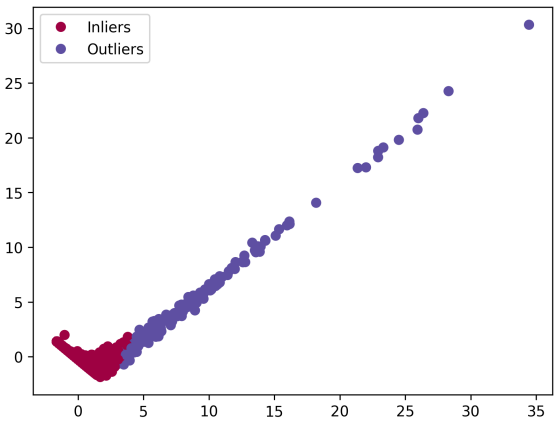

Figure 3.3: PCA visualization of the outliers and inliers found by LODA in the *artists* dataset.

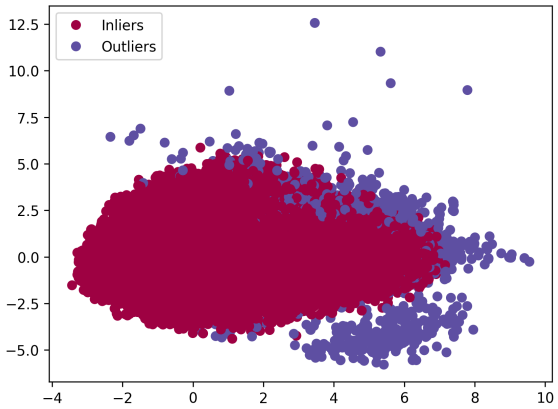

Figure 3.4: PCA visualization of the outliers and inliers found by LODA in the *tracks* dataset.

3.3 Isolation Forest

The final method used is Isolation Forest. We tried different numbers of estimators and compared the results: the points identified as outliers with the default value ($n\_estimators = 100$) do not change as the number is increased, so we kept it as the default. The Isolation Forest found 302 outliers in *artists* and 896 in *tracks*, visualized in Fig. 3.5 and 3.6.

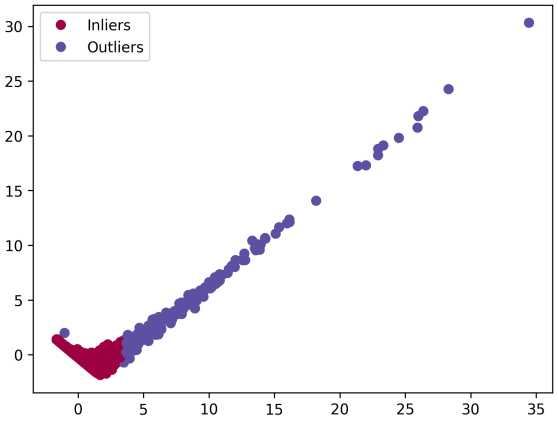

Figure 3.5: PCA visualization of the outliers and inliers found by Isolation Forest in the *artists* dataset.

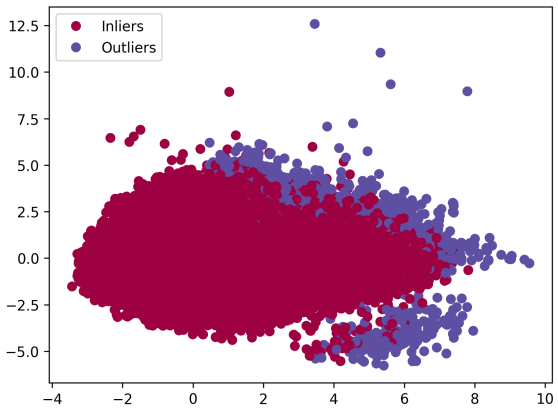

Figure 3.6: PCA visualization of the outliers and inliers found by Isolation Forest in the *tracks* dataset.

The result for the *artists* dataset is similar to that of LODA, but also includes some of the points in the bottom-left found by LOF. For the *tracks* dataset, while this method did not isolate outliers as drastically as LODA, it also seems to identify more outliers on the outer edge of the bigger mass at the center of the plot.

3.4 Handling of Outliers

For the *artists* dataset, we decided to remove all the outliers identified by LOF, since the method was able to find outliers in areas with lower density. For the *tracks* dataset, we instead decided to remove all the points that were identified as outliers by both LODA and Isolation Forest, since they are more appropriate for higher-dimensional datasets: the number of points removed from this dataset amounts to 629.

4. Imbalanced Learning

In this chapter we defined an unbalanced classification task and solved it using both Undersampling and Oversampling techniques.

4.1 Task Definition

As we conducted the Data Understanding task we realized, by visualizing the bar charts for the categorical features, that the dataset was significantly unbalanced with regards to the variable *explicit*, in particular:

- 81209 records had *explicit = False*
- 7725 records had *explicit = True*

Since the ratio between the two classes sat approximately at 92%-8% we decided not to perform any further unbalancing. Before continuing with the task we removed the categorical features from the dataset and encoded *explicit*, replacing value False with 0 and True with 1; then we proceeded with splitting the dataset into training (80%) and test (20%) and initialized the classification task.

We chose to train a Decision Tree on the imbalanced dataset to provide a baseline model for comparing the performance obtained by using the balancing techniques. The Decision Tree's hyperparameters were tuned according to a grid search out of the following alternatives:

- ***criterion***: *gini*, *entropy*;
- ***max_depth***: 8, 16, 32, 64, 128 and *None*;
- ***min_samples_leaf***: 4, 8, 16, 32, 64 or 128;
- ***min_samples_split***: 4, 8, 16, 32, 64 or 128;

The best configuration found by the grid search was *criterion = entropy*, *max_depth* = 8, *min_samples_leaf* = 32, *min_samples_split* = 4. The performance of the model was, apparently, optimal with an accuracy equal to 0.92, which was to be expected given how large the size of class *not explicit* was, and an AUROC amounting to 0.815; however the real picture was painted by looking at the Precision, Recall and F1-Score for both classes, summarized by Table 4.1.

Class	**Precision**	**Recall**	**F1-Score**
Explicit	0.62	0.13	0.21
Not Explicit	0.92	0.99	0.96

Table 4.1: Precision-Recall-F1-Score values of the Decision Tree on the imbalanced set.

4.2 Undersampling

In the following we delve deep into the techniques used to undersample the majority class.

4.2.1 Random Under Sampler

This technique drastically reduced the size of the dataset into an equal split for both classes, with 5407 records each. We trained the Decision Tree on the dataset using the hyperparameters found by the grid search, and obtained an accuracy of 0.69 and an AUROC of 0.814. Table 4.2 summarizes the results obtained for the Precision, Recall and F1-Score.

4.2.2 Tomek Links

Tomek Links technique reshaped the dataset into 54463 records for class *Not Explicit* and 5407 for class *Explicit*. As per the performance of the Decision Tree its accuracy sat at 0.92, improving thus the result produced by the Random Under Sampler method, while its AUROC at 0.812. Table 4.3 records the classification results.

4.2.3 Cluster Centroids

Cluster Centroids repeated the same reshaping as Random Under Sampler, with 5407 records for each class; the performance of the trained Decision Tree was by far the worst out of all the techniques performed, with a 0.50 accuracy and an AUROC equal to 0.761. Table 4.4 reports the results obtained for the Precision, Recall and F1-Score of the model.

4.2.4 Edited Nearest Neighbors

Edited Nearest Neighbor produced a split of 44925 for class *Not Explicit* and 5407 for class *Explicit*. The accuracy scored by the Decision Tree was 0.92 and its AUROC was equal to 0.816, while Table 4.5 summarizes the results obtained for the Precision, Recall and F1-Score.

Class	**Precision**	**Recall**	**F1-Score**
Explicit	0.19	0.80	0.31
Not Explicit	0.97	0.68	0.80

Table 4.2: Precision-Recall-F1-Score values using Random Under Sampler.

Class	**Precision**	**Recall**	**F1-Score**
Explicit	0.62	0.13	0.21
Not Explicit	0.92	0.99	0.96

Table 4.3: Precision-Recall-F1-Score values using Tomek Links.

Class	**Precision**	**Recall**	**F1-Score**
Explicit	0.13	0.87	0.23
Not Explicit	0.97	0.46	0.63

Table 4.4: Precision-Recall-F1-Score values using Cluster Centroids.

Class	**Precision**	**Recall**	**F1-Score**
Explicit	0.58	0.15	0.24
Not Explicit	0.92	0.99	0.96

Table 4.5: Precision-Recall-F1-Score values using ENN.

4.3 Oversampling

In this section we describe the methods used to oversample the minority class.

4.3.1 Random Over Sampler

The resampling operated by Random Over Sampler was the total opposite of the one applied by Random Under Sampler, hence resulting into an equal number of 56846 records for each class. The accuracy scored by the Decision Tree was equal to 0.69, while the values for Precision, Recall and F1-Score are shown in Table 4.6. We noted that, although the results of the classification

were exactly the same as the Random Under Sampling instance, the AUROC for Random Over Sampler was higher, sitting at 0.812.

Class	Precision	Recall	F1-Score
Explicit	0.19	0.80	0.31
Not Explicit	0.97	0.68	0.80

Table 4.6: Precision-Recall-F1-Score values using Random Over Sampler.

4.3.2 SMOTE

SMOTE performed the exact same reshaping as Random Over Sampler on the dataset, although producing much better results with respect to most of the metrics, as shown in Table 4.7, with an accuracy of 0.78 and an AUROC score of 0.798.

4.3.3 ADASYN

ADASYN reshaped the dataset into 56846 and 57134 records each for, respectively, class *Not Explicit* and *Explicit*. The Decision Tree trained on the balanced dataset produced an accuracy equal to 0.77 and an AUROC of 0.790, while the report on the Precision, Recall and F1-Score is shown by Table 4.8.

Class	Precision	Recall	F1-Score
Explicit	0.23	0.64	0.34
Not Explicit	0.96	0.80	0.87

Table 4.7: Precision-Recall-F1-Score values using SMOTE.

Class	Precision	Recall	F1-Score
Explicit	0.22	0.63	0.32
Not Explicit	0.96	0.78	0.86

Table 4.8: Precision-Recall-F1-Score values using ADASYN.

4.4 Conclusions

The results obtained by applying the Undersampling technique to the unbalanced data do not paint a clear picture in terms of pointing to the optimal method, as both Tomek Links and Edited Nearest Neighbours produced good results for the precision, while significantly sacrificing the sensitivity of the model, with Random Under Sampler and Cluster Centroid performing poorly on the other end due to the number of records removed on the majority class. Oversampling produced, as expected, satisfactory results regarding the sensitivity of the Decision Tree, although with a clear drop in the precision score across all models. SMOTE showcased narrowly better metrics than ADASYN, with Random Over Sampler producing the least balanced results overall. After taking into consideration both the classification metrics and the AUROC, we conclude that the best performing method for our analysis was, although narrowly, undersampling by Edited Nearest Neighbor.

5. Advanced Classification and Regression

This chapter illustrates our analysis regarding classification and regression using advanced models. The classification task is the same as the one defined in the previous chapter, choosing *explicit* as the target variable. The dataset was kept as is, without using any under/oversampling techniques.

5.1 Advanced Classification

The models used for the classification task are Logistic Regression, Support Vector Machines, Neural Networks, and Ensemble Methods. For all these tasks, we split the data into training (80%) and test set (20%), stratifying on the target variable. We also excluded all unique identifiers and similarly irrelevant variables (*id*, *name*, *artists*, *album_release_date*).

5.1.1 Logistic Regression

In order to have a starting point with which to compare subsequent models, we performed the classification using all available features as independent variables for the prediction of the class *explicit*, using the default parameters of the classifier. The results obtained from this initial model are, on average, satisfactory, especially for the majority class. The area under the ROC has a value of 0.82. The initial model was utilized as a foundation for subsequent parameter tuning, which was conducted through a grid search with 4-fold cross-validation. This process involved consideration of three hyperparameters: C, which represents a regularization parameter that controls the penalty for classification errors; *penalty*, which denotes the type of penalty applied during model optimization; and *solver*, which specifies the solver to be used to optimize the model weights. The hyperparameter values where chosen out of the following:

- ***C***: 0.001, 0.01, 0.1, 1, 10, 100;
- ***penalty***: L1, L2;
- ***solver***: *lbfgs*, *liblinear*, *newton-cholesky*, *sag*, *saga*.

The best combination is the following: $C = 1$, $penalty = L2$, and $solver = newton\text{-}cholesky$. The hyperparameters yielded the same AUROC, equal to 0.82, but slightly better result in terms of precision, recall and f1-score. Those values are summarized by Tables 5.1 and 5.2.

Class	Precision	Recall	F1-Score
Explicit	0.49	0.11	0.18
Not Explicit	0.92	0.99	0.95

Table 5.1: Precision-Recall-F1-Score values of the first logistic regressor.

Class	Precision	Recall	F1-Score
Explicit	0.50	0.11	0.18
Not Explicit	0.92	0.99	0.95

Table 5.2: Precision-Recall-F1-Score values of the logistic regressor with the best hyperparameters.

In the context of logistic regression, the features utilized to predict the dependent variable play a pivotal role. The selection of the most important features was done by searching those capable of making the greatest contribution to the classification. As first approach, a new model was trained with the optimal grid search parameters on all features. The coefficients of the features were

then obtained using the $coef\_$ attribute, and the features were sorted according to their absolute value. The top k most significant features were selected for use in making new predictions. After conducting a series of experiments, we were unable to identify a value below 15 that would yield improved results, and in fact, showed a decline. The value of the AUROC curve was 0.63.

In the second approach, we supposed that L1 regularization might improve the outcomes, since it is more suitable when feature selection is required. A grid search was conducted using the complete dataset, with the regularization parameter "L1" held constant and the model trained in accordance with the previously described parameters. A comparison of the results obtained with the previous model revealed that there was no appreciable change in performance with respect to the AUROC that is, in fact, lower: with a value of 0.80. The values are summarized by Table 5.3.

Class	**Precision**	**Recall**	**F1-Score**
Explicit	0.54	0.07	0.12
Not Explicit	0.92	0.99	0.95

Table 5.3: Precision-Recall-F1-Score values of the logistic regressor with L1 regularization.

In conclusion, the best model retrieved is the one captured by the grid search, with $C = 1$, $penalty = L2$, $solver = newton\text{-}cholesky$ and $AUROC = 0.82$.

5.1.2 Support Vector Machines

The initial phase of the analysis used a linear SVM; next, we proceeded to use a non-linear model.

Linear SVM

Once more, our investigation started with the default model for linear SVMs, with the objective of obtaining a comprehensive understanding. The hyperparameters considered were: $C = 1.0$ and $penalty = L2$. The initial results yielded an AUROC of 0.82, indicating a relatively high degree of precision and predictive power. To examine the full range of potential values and to enhance the outcomes, we conducted a random search, concentrating on the alternative values illustrated in the following table for the parameters C and *penalty*.

Hyperparameter	**Values**	**Best**
C	0.001, 0.01, 0.1, 1, 10, 100	0.01
Penalty	L1, L2	L2

The optimal combination yielded an AUROC value of 0.82, which remained unchanged. In consideration of the overall outcome, these hyperparameters yield a marginally superior result. The precision, recall, and f1-score of the models are summarized by Tables 5.4 and 5.5.

Class	**Precision**	**Recall**	**F1-Score**
Explicit	0.53	0.08	0.13
Not Explicit	0.92	0.99	0.95

Table 5.4: Precision-Recall-F1-Score values of the SVM with the default hyperparameters.

Class	**Precision**	**Recall**	**F1-Score**
Explicit	0.54	0.08	0.13
Not Explicit	0.92	0.99	0.95

Table 5.5: Precision-Recall-F1-Score values of the SVM with the best hyperparameters.

Non Linear SVM

Once more, we started the process from the baseline SVM classifier, with the following parameters: $kernel =$ *"rbf"*, $C = 1.0$, and $gamma =$ *"auto"*.

Due to the occurrence of multiple kernel deaths, it was not feasible to implement a grid search or random search approach. Consequently, we fixed the *kernel* values for *"poly", "rbf",* and *"sigmoid"* and varied the values of $C \in [0.001, 0.01, 0.1, 1]$. In the case of the *kernel="rbf"*, gamma was set to "auto" for each iteration.
While the results were largely similar, the following combination was identified as optimal: *kernel="poly"*, $C = 1.0$. It presented a slight advantage in terms of $AUROC = 0.76$.
In the Table 5.6 we can find the value retrieved for precision, recall and F1-score for that model.
In conclusion, the best result for the Support Vector Machine is the linear model with $C = 0.01$

Class	**Precision**	**Recall**	**F1-Score**
Explicit	0.62	0.07	0.12
Not Explicit	0.92	1.00	0.96

Table 5.6: Precision-Recall-F1-Score values of the Non Linear SVM with the best hyperparameters.

and $penalty = L2$.

5.1.3 Neural Networks

We tested three networks: a small net with two hidden layers, of 4 and 8 units, a medium net with three hidden layers, of 32, 64, and 32 units, and a bigger net with three hidden layers of 64, 128, and 64 units. For all of them we ran a randomized search to find the best hyperparameters out of the following:

- ***activation_function***: *sigmoid*, *ReLU*;
- ***optimizer***: *adam*, *sgd*;
- ***learning_rate***: 0.1, 0.5, 0.01, 0.05, 0.001;
- ***reg_type***: L1, L2;
- ***reg_lambda***: 0.1, 0.5, 0.01, 0.05, 0.001;
- ***momentum***: 0.9, 0.5, 0.1;
- ***epochs***: 100, 500, 1000.

The combinations found by the search are reported in Table 5.7, 5.8, and 5.9.

Hidden units	**4, 8**
Activation fun.	ReLU
Optimizer	SGD
Learning rate	0.01
Reg. type	L1
Reg. lambda	0.01
Momentum	0.9
Epochs	100

Table 5.7: Best hyperparameter combination for the smaller network.

Hidden units	**32, 64, 32**
Activation fun.	ReLU
Optimizer	SGD
Learning rate	0.001
Reg. type	L2
Reg. lambda	0.05
Momentum	0.9
Epochs	500

Table 5.8: Best hyperparameter combination for the medium-sized network.

Hidden units	**64, 128, 64**
Activation fun.	Sigmoid
Optimizer	SGD
Learning rate	0.005
Reg. type	L2
Reg. lambda	0.05
Momentum	0.5
Epochs	1000

Table 5.9: Best hyperparameter combination for the larger network.

Afterwards, all networks were retrained, setting the number of epochs to 1000, and using early stopping, with *patience* = 10 and *min_delta* = 0.0001: the training of the first model stopped at 96 epochs, the second model stopped at 659, and the third model stopped at 764. Below we

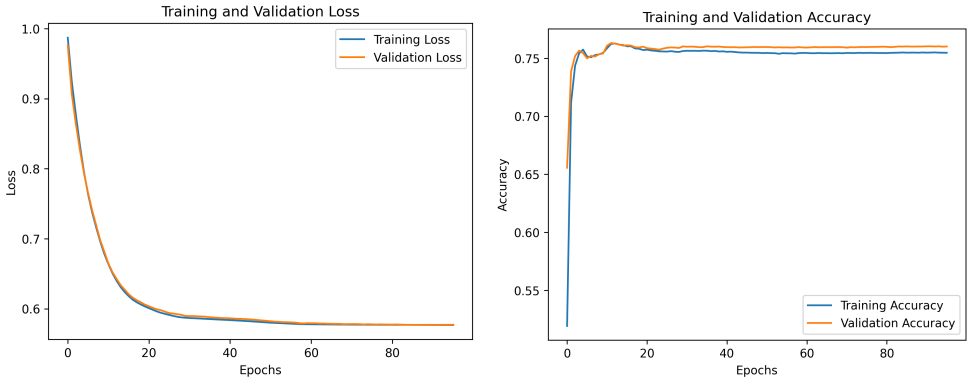

Figure 5.1: Learning curves of the first neural network (two hidden layers).

report the learning curves of the first model, in Fig. 5.1; the others have similar behavior. Despite having different sizes, the nets have no big difference in performance: the main difference is that, since the larger nets have more units (so they are more complex), they need a higher *lambda* than the first net. Also, since the first net uses a higher learning rate, it requires a lower number of epochs to converge to a solution. Their precision, recall, and f1-score are summarized by Tables 5.10, 5.11, and 5.12 where it can be seen how their performances evaluated on the test set are nearly identical. Their AUROC are all 0.80.

Class	**Precision**	**Recall**	**F1-Score**
Explicit	0.23	0.74	0.35
Not Explicit	0.97	0.76	0.85

Table 5.10: Precision-Recall-F1-Score values of the first neural network.

Class	**Precision**	**Recall**	**F1-Score**
Explicit	0.22	0.75	0.34
Not Explicit	0.97	0.75	0.85

Table 5.11: Precision-Recall-F1-Score values of the second neural network.

Class	**Precision**	**Recall**	**F1-Score**
Explicit	0.22	0.74	0.34
Not Explicit	0.97	0.75	0.85

Table 5.12: Precision-Recall-F1-Score values of the third neural network.

5.1.4 Ensemble Methods

Random Forests

The first ensemble method we implemented for our analysis was Random Forest, whose tuning was performed by a grid search with 4-fold CV on the following grid:

- ***criterion***: *gini*, *entropy*;
- ***max_depth***: 30, 40, 50;
- ***max_features***: *sqrt*, *log2*;

- ***min_samples_split***: 2, 4, 8;
- ***n_estimators***: 200, 300, 400.

The best configuration found by the grid search was *criterion* = *entropy*, *max_depth* = 30, *max_features* = *sqrt*, *min_samples_split* = 2 and *n_estimators* = 300. The results of the classification are summarized by Table 5.13.

Class	**Precision**	**Recall**	**F1-Score**
Explicit	0.83	0.22	0.34
Not Explicit	0.93	1.00	0.96

Table 5.13: Precision-Recall-F1-Score values of Random Forest Classifier.

We also report the Feature Importance plot, as per Figure 5.2, which provides a ranking of the importance of each feature in the dataset, with respect to predicting feature *explicit*, as well as the ROC curve plot in Figure 5.3, whose AUC was equal to 0.88. As highlighted by the Feature Importance plot, the most influential feature is *speechiness* with a score equal to 0.157.

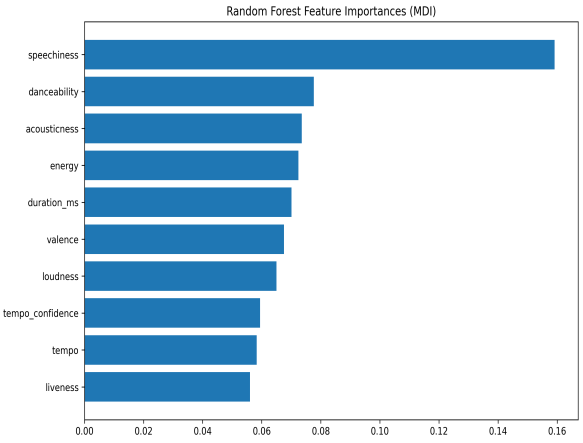

Figure 5.2: Feature Importance Plot for Random Forest Classifier

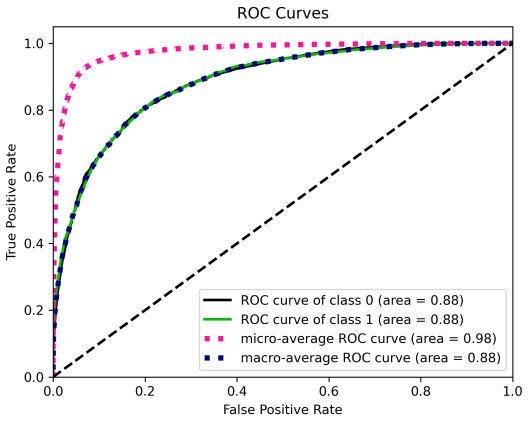

Figure 5.3: ROC curve plot for Random Forest Classifier

Gradient Boosting Machines

We tested how well gradient boosting could solve the task using both "basic" Gradient Boosting Machines and the regularized variant offered by XGBoost. Through a grid search with 4-Fold CV, we explored the following hyperparameters for gradient boosting:

- ***n_estimators***: 100, 500, 1000;
- ***learning_rate***: 0.1, 0.01, 0.001;
- ***max_depth***: 3, 5, 7;
- ***min_samples_split***: 2, 4, 8;
- ***min_samples_leaf***: 1, 2, 4.

For XGBoost, we explored the same first three hyperparameters as gradient boosting, as well as:

- ***gamma***: 0, 0.1, 0.2, 0.3, 0.4;

- ***reg_alpha***: 0.1, 0.01, 0.001;
- ***reg_lambda***: 0.1, 0.01, 0.001;

which are all regularization hyperparameters offered by the library.

The best combination found for GBM is *n_estimators* = 1000, *learning_rate* = 0.01, *max_depth* = 7, *min_samples_split* = 2, and *min_samples_leaf* = 1. For XGBoost, it is *n_estimators* = 1000, *learning_rate* = 0.1, *max_depth* = 7, *gamma* = 0.1, *reg_alpha* = 0.01, and *reg_lambda* = 0.01. Their respective precision, recall, and f1-score are in Tables 5.14 and 5.15. Their AUROC are, respectively, 0.87 and 0.86.

Class	**Precision**	**Recall**	**F1-Score**
Explicit	0.30	0.73	0.42
Not Explicit	0.97	0.84	0.90

Table 5.14: Precision-Recall-F1-Score values of the GBM classifier.

Class	**Precision**	**Recall**	**F1-Score**
Explicit	0.51	0.45	0.48
Not Explicit	0.95	0.96	0.95

Table 5.15: Precision-Recall-F1-Score values of the XGBoost classifier.

5.1.5 Comparison of Models

By taking in consideration AUROC values only (collected in Table 5.16), the best model would be the Random Forest, closely followed by regularized Gradient Boosting Machines, while the worst would be the Neural Network. In general, however, all models have high values of AUROC.

	Log. Regression	**SVM**	**NN**	**Random Forest**	**GBM**
AUROC	0.82	0.82	0.80	0.88	0.86/0.87

Table 5.16: AUROC values of the models applied.

If instead we compare precision, recall, and f1-scores of the positive class, we can observe that:

- Logistic regression, linear and non-linear SVM all have very low recall and a medium precision, resulting in an also low f1-score;
- Neural Networks and non regularized GBM have high recall and low precision (although much higher than the previous models), while Random Forest is the opposite;
- XGBoost has similar, medium values for both precision and recall, and also has the highest f1-score (0.48).

5.2 Advanced Regression

For regression, we defined a task using data contained in both the *tracks* dataset and the *artists* dataset. For each track, we collected the popularity of its artist (or average popularity, in case of multiple artists); any track which did not have its artist listed in the other dataset was simply removed, bringing the training data to a total of 87259 records.

We chose try and solve the task using two types of models: Neural Networks, and Random Forest.

5.2.1 Neural Networks

Similarly to the classification task, we tried comparing the performance of different architectures: two hidden layers with a medium amount of units (16 and 32 units), four hidden layers with few units (2, 4, 4, and 2 units), and three hidden layers with many units (128, 256, and 128 units). We ran a randomized search to find the best hyperparameter combination for each of them (using the same grid as seen for the neural networks in the previous section). For all of them we also ran an additional training with early stopping to make sure they wouldn't overfit. Out of the three, the biggest one seemed to perform much better, with the hyperparameter combination in Table 5.17. We also report its learning curves in Fig. 5.4, and its metrics in Table 5.18.

Hidden layers	**[128, 256, 128]**
Activation function	ReLU
Optimizer	Adam
Learning rate	0.001
Reg. type	L1
Reg. lambda	0.01
Epochs	100

Table 5.17: Best hyperparameter combination for the neural network.

MSE	**MAE**	**R2 Score**
154.9	9.81	0.47

Table 5.18: MSE, MAE, and R2 score of the neural network.

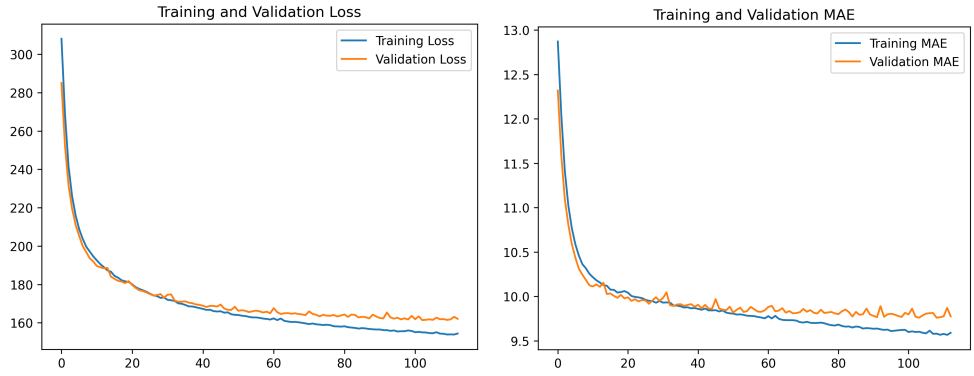

Figure 5.4: Learning curves of the neural network.

5.2.2 Random Forests

As we previously did in the classification task, we ran a 4-Fold Grid Search in order to find the best hyperparameter configuration for the Random Forest Regressor, resulting in:

- ***criterion***: *friedman_mse*;
- ***max_depth***: 30;
- ***max_features***: *sqrt*;
- ***n_estimators***: 500;
- ***min_samples_split***: 2;

We report the metrics as per Table 5.19 and the Feature Importance graph produced by the Random Forest in Figure 5.5, which highlighted the most influential feature by far in predicting *artist_popularity* being *popularity* with a score equal to 0.28.

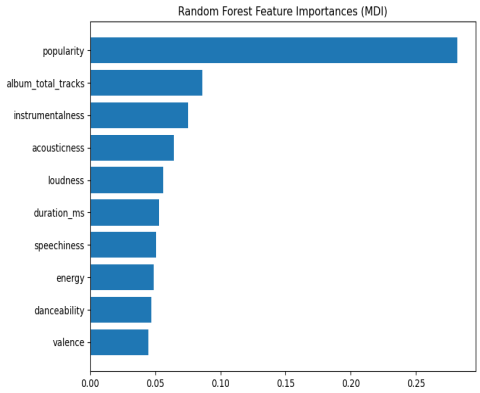

Figure 5.5: Feature Importance plot of Random Forest Regression.

MSE	**MAE**	**R2 Score**
139.38	9.21	0.53

Table 5.19: MSE, MAE, and R2 score of the Random Forest.

5.2.3 Comparison of Models

By comparing their respective scores, we can conclude that both models have similar performances, with the random forest having slightly better metrics. While they don't solve the task exceptionally well, they both have positive R^2 scores, which indicate that can explain the target variable reasonably well.

6. Explainability

This last chapter is dedicated to our analysis using XAI methods to explain the predictions done for the classification task of the previous chapter. We chose the neural network, specifically the smaller one with two layers, since it's faster to train and has similar performances to the other ones.

6.1 Global Method

After training the model, we obtained the predictions for all test instances, and trained a Decision Tree (as an approximation of TREPAN) using the test inputs and their corresponding predictions as target values, and limiting its *max_depth* to 4, so that it wouldn't get too deep to easily read. The pruned tree is in Fig. 6.1.

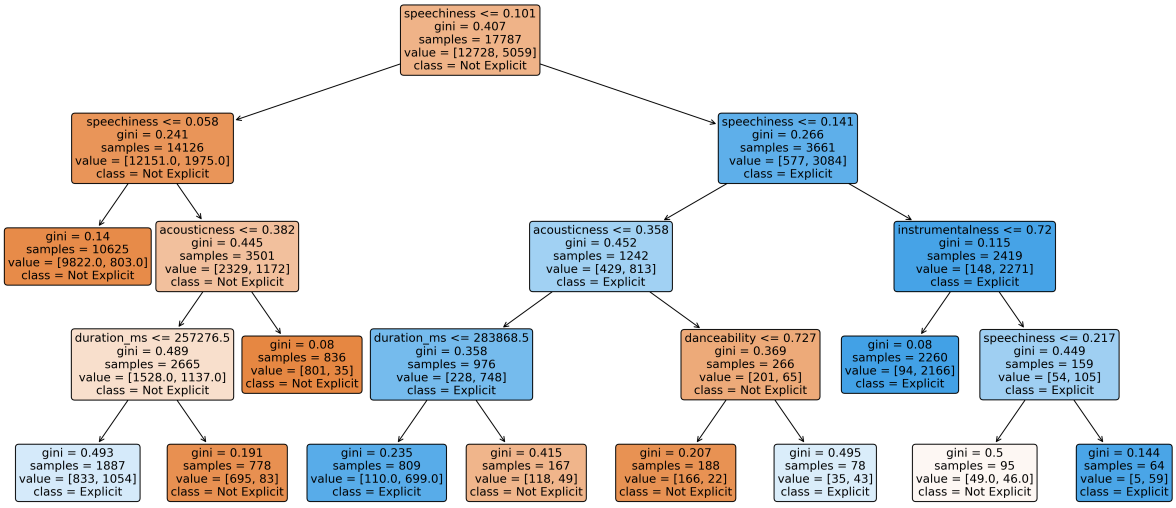

Figure 6.1: Decision Tree obtained by training on the test inputs and their predictions by the neural network.

The most significant splits are done on the *speechiness* attribute, which appears both in the root node and its two children, showing that tracks with higher values for this attribute are classified as *explicit*. The other splits use attributes *acousticness*, *instrumentalness*, *danceability*, and *duration_ms*. Specifically, since *acousticness* and *duration_ms* appear in both the root's subtrees, it can be seen how low values of both attributes tend to classify the instance as *explicit*, while high values tend to classify it as *not_explicit*. The attributes chosen for the splits are also some of the ones with the highest feature importance assigned by the Random Forest (Fig. 5.2).

6.2 Local Method

As a local method, we used SHAP, finding the shapley values of a few instances. We isolated four different instances in total: two misclassified ones and two correctly classified ones, both groups with one positive and one negative. The waterfall charts with each instance's shapley values are in Fig. 6.2, 6.3, 6.4, and 6.5. Coherently with what is shown by the decision tree above, *speechiness* is the feature that influences predictions the most: by inspecting the actual values of the instances, we could confirm that those with low *speechiness* are classified as belonging to the negative class, while those with high *speechiness* as part of the positive class. Most of the

other features at the top of the charts also match the ones in the decision tree.

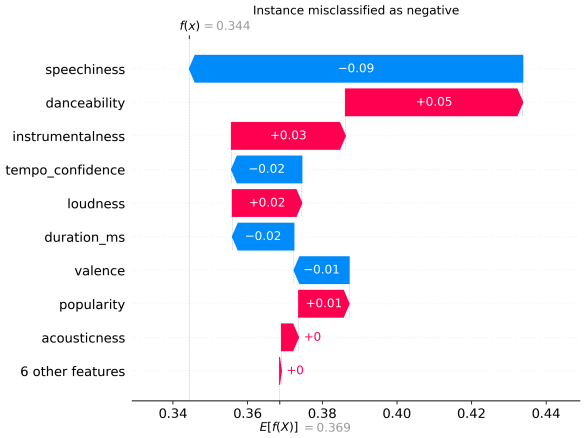

Figure 6.2: Shapley values of a positive instance incorrectly classified as negative.

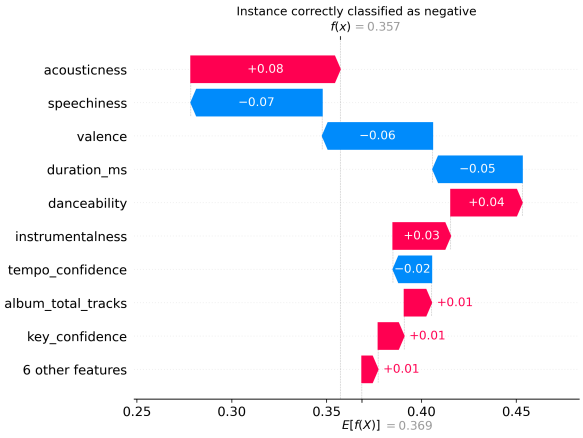

Figure 6.3: Shapley values of a correctly classified negative instance.

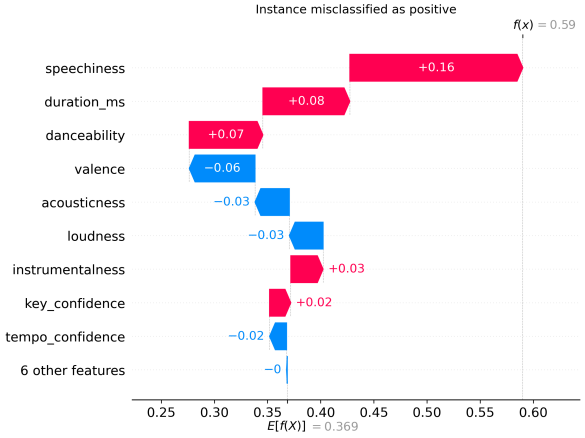

Figure 6.4: Shapley values of a negative instance incorrectly classified as positive.

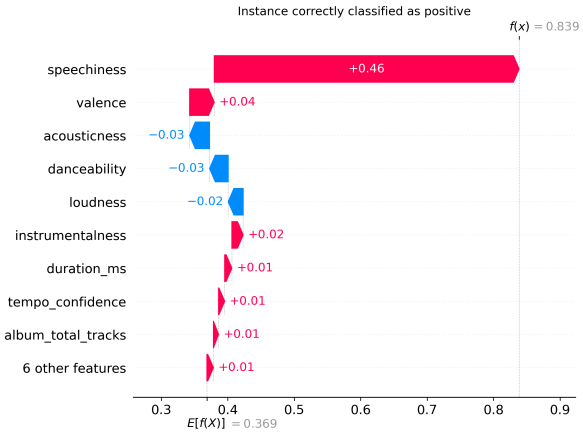

Figure 6.5: Shapley values of a correctly classified positive instance.

Another feature that is among the top-5 most influential ones in *valence*. The first three instances all have medium-high *valence* (≥ 0.5), while the last one has a low value (≈ 0.2). The waterfall charts suggest that lower values of this attribute also marginally contribute to classify the instance as *explicit*.