UNIVERSITÀ DI PISA

Department of Computer Science

Master in Data Science and Business Informatics

DM2 – Data Mining, Advanced Topics and Applications

Professors:

Prof. Riccardo Guidotti

Dott. Andrea Fedele

Students:

Martin Miccoli

Elisa Lucarelli

Academic Year 2023/24

Contents

1 Data Understanding

In this report, we will analyze three datasets: two tabular datasets and one time series dataset. The ”tracks” dataset consists of 109547 rows and 34 columns, while the ”artists” dataset comprises 30141 rows and 5 columns. The third dataset consists of 10.000 time series representing musical tracks. We convert the time series dataset into a tabular format called ”ts_to_tabular”, which consists of 10.000 rows and 1282 columns.

These datasets are correlated and provide different information regarding musical tracks. The ”tracks” dataset offers descriptive features corresponding to individual tracks, the ”artists” dataset provides information about the artists who created them and ”ts_to_tabular” gives us audio feature characteristics. With these three datasets, we can analyze various aspects of our data in depth. We will explore patterns, correlations, trends, and insights that emerge.

1.1 Track and artist data semantic

The **artists dataset** contains the columns: ”id”, ”name”, ”popularity”, ”followers”, and ”genres”. These columns provide additional information about the artists. The ”genres” column contains lists of genres corresponding to a single artist. We decided to change the names of these columns by adding the suffix ”_artists” to make the analysis clearer, as the tracks dataset also has a ”popularity” column.

The column **”id_artist”** represents the identifier of each artist, which corresponds to the ”id” column in the track dataset. The **”name_artist”** column contains the names of the artists, which is useful for searching and retrieving specific information about each artist present in the track dataset.

The **”popularity_artist”** column highlights an artist’s popularity in terms of the total listens of all their songs, with values ranging from 0 to 100. Observing the related boxplot, we noted that the 75th percentile reaches 49, indicating varying popularity levels across the artist spectrum in our dataset. The average popularity of the artists is 37.

The **”follower_artist”** column represents the number of followers an artist has on Spotify. Examining the correlation between the variables ’popularity_artists’ and ’follower_artists’, we expected a high correlation. However, it turned out to be 0.32, indicating a positive but not very strong correlation. The lower-than-expected correlation might be due to the fact that an artist’s popularity reflects the popularity of all their tracks. Some tracks may be played less frequently than others, as not all followers necessarily play the artist’s tracks (e.g. bots), and not all followers listen to all the artist’s tracks.

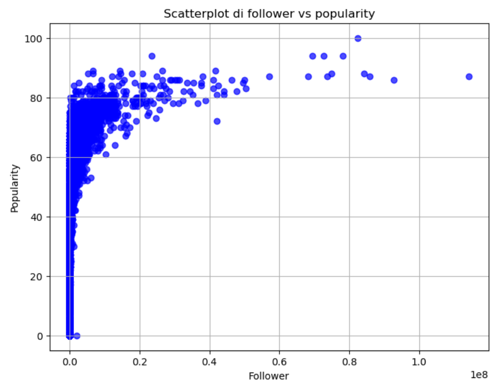

Figure 1: Popularity follower

In the scatterplot, we can see that there is a general trend where artists with a higher number of followers tend to have greater popularity. However, this is not a perfectly linear correlation. Most of the data points are concentrated in the region with a low number of followers (near 0), yet the popularity values vary significantly. This suggests that there are many artists with few followers, and some of them have very high popularity.

The tracks dataset contains information about the songs in the following columns: id, name, disc_number, duration_ms, explicit, popularity, track_number, artists, album_type, album_name, album_release_date, album_release_date_precision, album_total_tracks, danceability, energy, key, loudness, mode, speechiness, acousticness, instrumentalness, liveness, valence, tempo, features_duration_ms, time_signature, start_of_fade_out, tempo_confidence, time_signature_confidence, key_confidence, mode_confidence, n_beats, n_bars, genre.

These columns provide various types of information about the tracks. We can find details that identify the track itself, the album, or the artist. Additionally, we can find musical and temporal information about a specific track. Below, we have decided to describe the most significant columns:

- **ID and Name**: These columns are useful for identifying the track. Id provides a numerical identifier, and Name gives the name of the track. These two variables are very useful for identifying songs and particularly for recognizing any duplicates.
- **Disc Number**: Represents the number of discs per album. We observed that each track has a disc_number value ranging from 1 to 14, with most tracks (98%) being on disc 1.
- **Duration_ms**: we observed an average duration of 3.79 minutes, a maximum of 68.67 minutes and a minimum of 0.14 minutes. We identified 582 tracks that exceed 10 minutes in duration and 835 tracks that are less than 1 minute long. This suggests the presence of outliers in the column, which we will analyze more thoroughly in the report.

- **Explicit**: This variable shows a tendency towards one side of the distribution, with the majority of tracks (91%) being non-explicit. We consider this variable to be very interesting for future analyses.
- **Album release dates**: represents the precise date on which a song was released, with the earliest release date being in 1899 and the most recent in 2023. We visualized a chart containing all release dates and observed a peak in 2022. Zooming in, we found that the majority of songs in our dataset were released during October 1st to October 21st (see Figure 2). This might suggest a strategic period for marketing song releases. We further investigated the timing between 2010 and 2024 to understand which days of the week releases were concentrated on, discovering that it was Friday (see Figure 2). This confirmed the hypothesis that releases were strategically timed to maximize listenership.

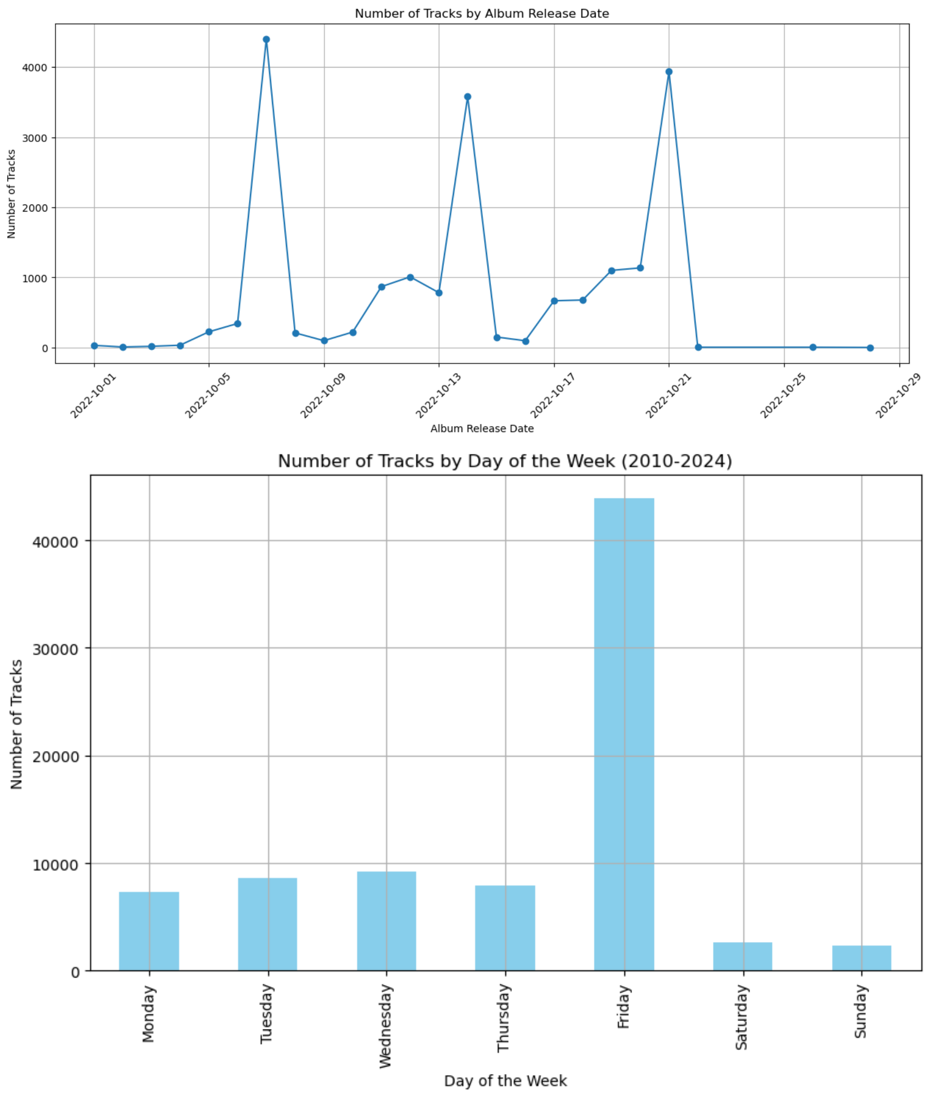

Figure 2: Release Date & Number of Tracks of the Week

- **Popularity**: Each track has a popularity score that ranges from 0 to 95, the average popularity of tracks is 31.
- **Album Type**: we observed that 53.58% of the tracks belong to an album, 25.15% are singles and 21.27% compilation. Below, we can see a bar chart that shows the popularity corresponding to each album type, and we discovered that singles are the most popular.

  We can observe a bar chart that highlights the correlation between popularity and album type, revealing that singles are the most listened-to album type.

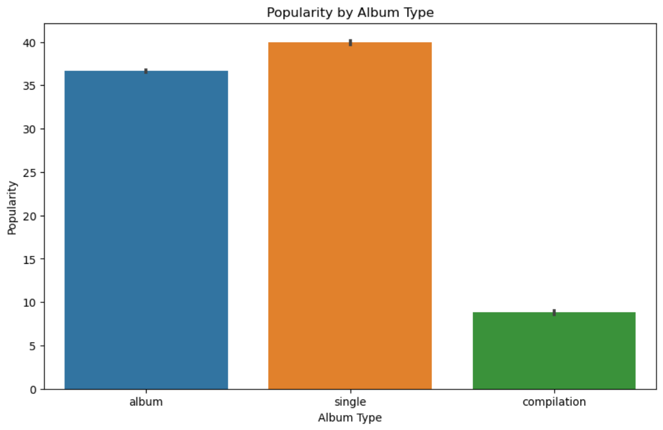

Figure 3: Popularity album

- **Album Release Date Precision:** This variable gives us more in-depth information about the variable "album_release_date", letting us know if the dates correspond to specific days (94.6%), months (0.05%), or years (5.35%) in which the track was released.
- **Genre**: represents a diversity of 114 genres, providing us with very specific information about which genre a song belongs to. Most genres contain roughly the same number of tracks (1000), while 8 genres contain fewer than 700 tracks, highlighting a relatively balanced distribution of tracks across genres. Additionally, we observed that the 3 most popular genres are: "pop-film", "k-pop" and "grunge".

1.2 Data cleaning and Data quality

1.2.1 Artists data cleaning

We searched for duplicates by referring to the columns 'id_artist' and 'name_artist', identifying 57 duplicates which we then removed, ensuring the uniqueness of the IDs. For the missing values, our initial analysis revealed the presence of missing values in only two rows, which we subsequently removed. We conducted a further analysis for missing values by searching for empty lists in the 'genre' column, identifying 7.556 such instances. We decided to remove these rows to improve the quality of our analysis.

1.2.2 Track data cleaning

As with the artist dataset, we searched for duplicates in the 'id' and 'name' columns and removed them. From an initial analysis, no missing values were found. We then conducted a more in-depth analysis, which also did not reveal any missing values. We did observe many values corresponding to 0, but we did not interpret them as missing values (e.g., in the 'popularity' column, there are tracks with a popularity of 0, which we interpreted as unpopular tracks). Additionally, we examined the correlations between variables and removed those that were highly correlated to simplify future analyses. We found that the variables 'n_beat', 'n_bars', 'features_duration_ms', and 'start_of_fade_out' were highly correlated with 'duration_ms'. Therefore, we decided to remove these variables and consider only 'duration_ms'.

1.3 Introduction of ts dataset

The time series dataset will be analyzed more in depth in Chapter 3; for now, we will describe its main characteristics. Each row in the dataset represents a time series, with 1280 columns representing the numerical data that depict the structure of the song, while the last two columns are 'id_ts' and 'genre'. The 'genre' column categorizes the time series into 20 genres. In Figure 4 , we can observe the average time series for each genre. Some genres, such as progressive-house, happy, and j-idol, exhibit very high values, indicating a more energetic rhythm. Conversely, genres like piano and new age correspond to a slower rhythm. Additionally, we can observe that some variations have more stable average values over time, while others exhibit more fluctuations. For example, heavy-metal shows a line with relatively high variations, whereas new-age appears to be more stable.

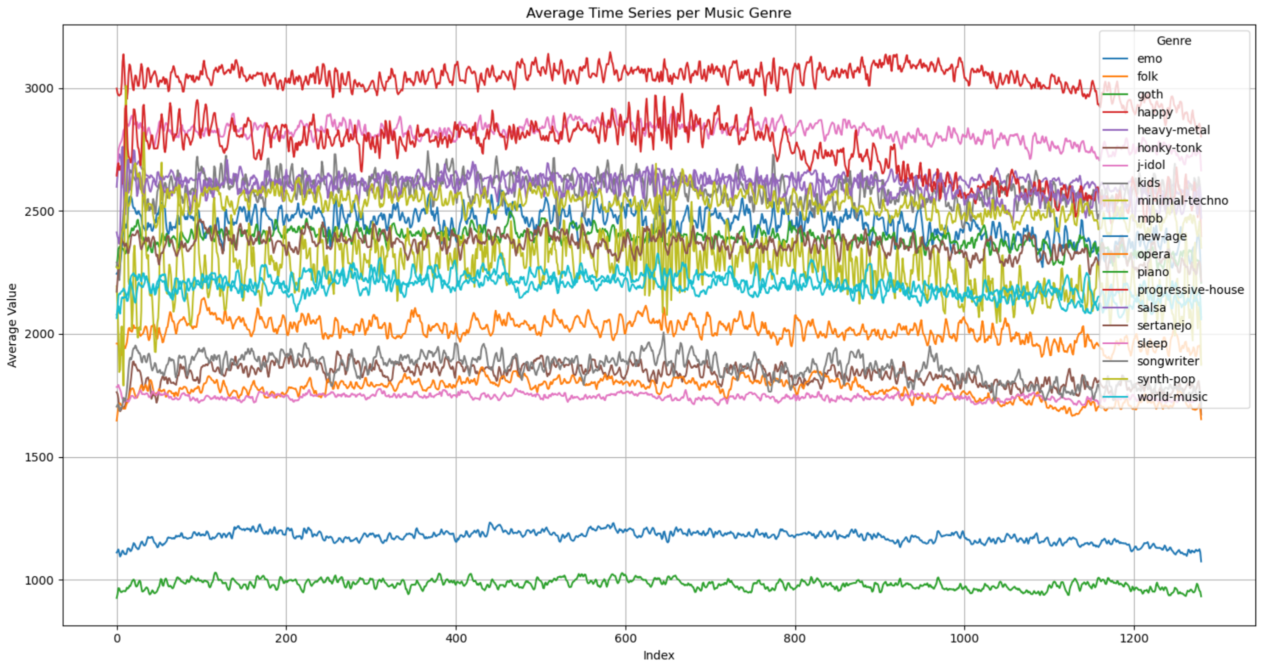

Figure 4: Time Series medie per genere

The 'id_ts' column is used to identify individual time series and to detect any duplicates. We identified 136 duplicates and removed them, reducing the dataset to 9864 time series. Additionally, we observed a time window of 1280 for all the time series. Finally, we approximated the time series using PAA, reducing the length from 1280 to a length of 100. We used this approximation technique for all clustering and classification operations.

2 Time Series

Building on what we discussed in Chapter 2, this section will analyze our time series data. We will introduce two main clustering techniques: DBSCAN and K-means. We will identify the motifs and discords present in our tracks. The KNN algorithm will be employed for the classification part, using two distance measures: Euclidean and DTW. This will also help us assess the similarity of the time series. Finally, by analyzing shapelets, we will identify the subsequences representative of the classes.

2.1 Clustering

2.1.1 DBSCAN

Regarding DB-Scan, we set the following values: an epsilon of 3500, min samples of 180, and the metric to DTW. This allowed us to achieve a Silhouette score of 0.65 with 14% of points classified as noise. Here we visualize the distribution of genres for each class with a heatmap. The color ranges from purple (low values) to yellow (high values), representing the number of tracks of a particular genre present in each cluster. Each cell contains a number that represents the actual count of tracks of a specific genre in a given cluster.

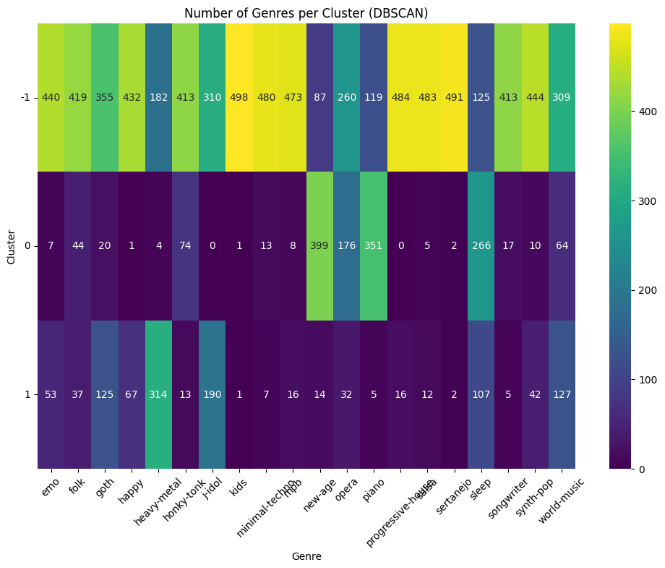

Figure 5: Number of genres per cluster dbscan

We can notice that the outliers cluster is highly populated, which might indicate the presence of many noisy data points or points that do not fit well with the clustering models. For the main clusters (0 and 1), the genres are distributed differently. Cluster 0 has a strong representation of new-age, piano, and sleep, while Cluster 1 is more representative of j-idol and happy. DBSCAN managed to identify clusters that are representative of certain genres but also many outliers.

Now we visualize the 3 clusters detected by DBSCAN using PCA and t-SNE.

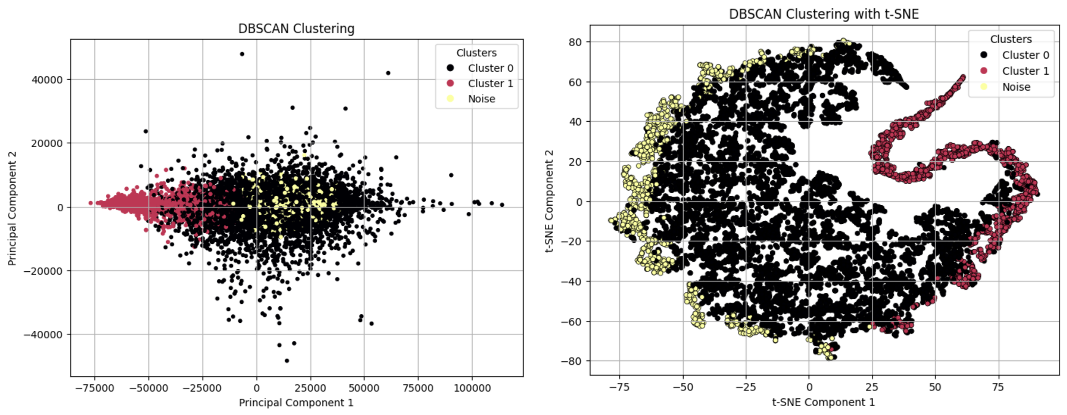

Figure 6: Cluster - K-means & t-SNE DBSCNA

2.1.2 K-means

Regarding the clustering with the K-means algorithm, we used DTW as the distance metric. To choose k, we observed a range of k from 2 to 10, evaluating the silhouette score and the elbow plot. The elbow method suggested k = 4, while the silhouette score suggested k = 2. For the purposes of our analysis, we preferred k = 4, as it represents a good compromise between cluster compactness and their separation. Additionally, for K-means, we visualize with a heatmap how the various genres are distributed across the clusters (see Figure 7).

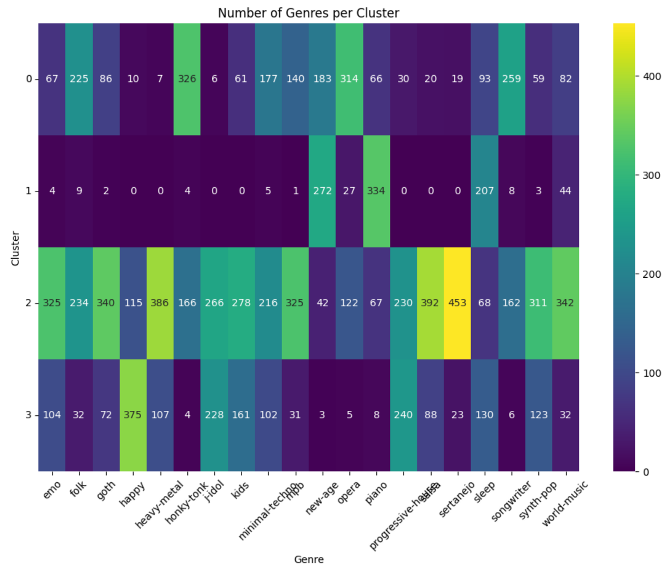

Figure 7: Number of genres per cluster k-means

We can see that Cluster 0 is representative of the genres: honky-tonk and songwriter. Cluster 1 includes the genres: new-age, piano, and sleep. Cluster 2 is characterized by the genres: heavy-metal, MPB, and salsa. Finally, Cluster 3 is representative of the genres: happy, j-idol, and progressive-house. The distribution of genres in the K-means clusters is better compared to that of DBSCAN, which grouped many tracks within the outlier cluster.

For better interpretation of the results, we reduced the dimensionality of the data using PCA and t-SNE, which allow us to visualize the clusters in two-dimensional space (see image 8). With K-means, we achieved a silhouette score of 0.44. Considering that the silhouette score ranges from -1 to 1, we can say that this is a fairly good result. This indicates that our clusters are well-grouped, which is evident in the images.

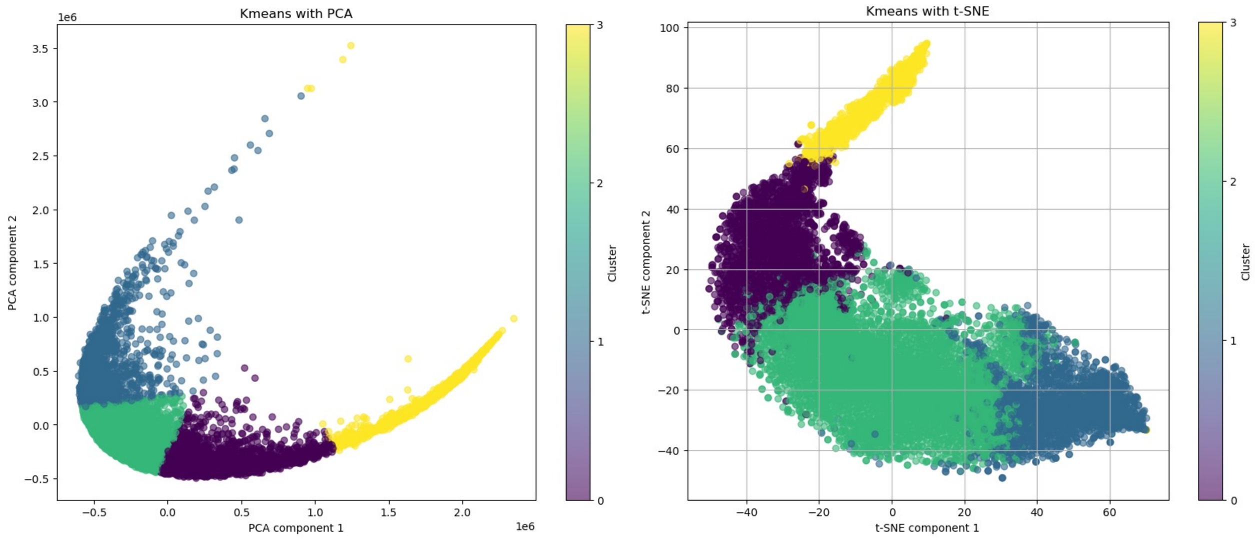

Figure 8: PCA & t-SNE K-Means

2.2 Motif and Discords

In this chapter, we will explore the analysis of motifs and discords in time series, using a clustering-based approach. Our methodology is divided into three main phases:

1. Application of clustering techniques to identify significant groups within the data (already done in chapter 3.1)
2. Identification of centroids for each cluster obtained
3. Search for motifs and discord for each identified centroid.

Through this methodology, we aim to reveal meaningful insights into the underlying structures of the data by identifying both typical patterns and notable deviations within each cluster. The image 9 shows the distribution of clusters in the PCA space. Additionally, we wanted to show the centroids of each cluster.

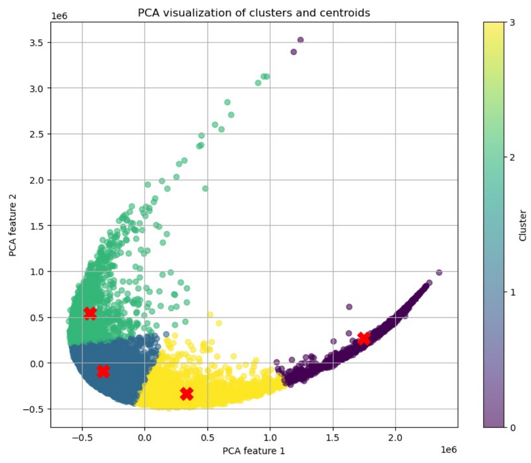

Figure 9: Clusters

Essentially, we extracted the Time Series from the centroids and applied motif and discord discovery on those Time Series. We identified those using the STUMPY algorithm, the parameter choosed was 50 for the length of the motif/discord and we trasformed our time series into an array 1D. The rationale behind choosing a motif/discord length of 50 for the STUMPY algorithm likely involves a balance between capturing significant patterns and computational efficiency. Here's a breakdown of the considerations:

- Motif/Discord Granularity: A length of 50 allows the algorithm to identify patterns or deviations that span 50 data points in the time series. This provides a balance between capturing specific details and avoiding overly granular patterns that might be due to noise.
- Computational Cost: STUMPY's running time can increase with motif/discord length. A length of 50 offers a reasonable compromise between capturing informative patterns and maintaining computational feasibility, especially for larger datasets.

This approach allowed us to visually represent segments of interest within the time series, providing insights into the data's structure and behavior. Note: In the following images, we only show 2 time series due to space constraints. The first one with class "happy" and the second one with class "heavy-metal".

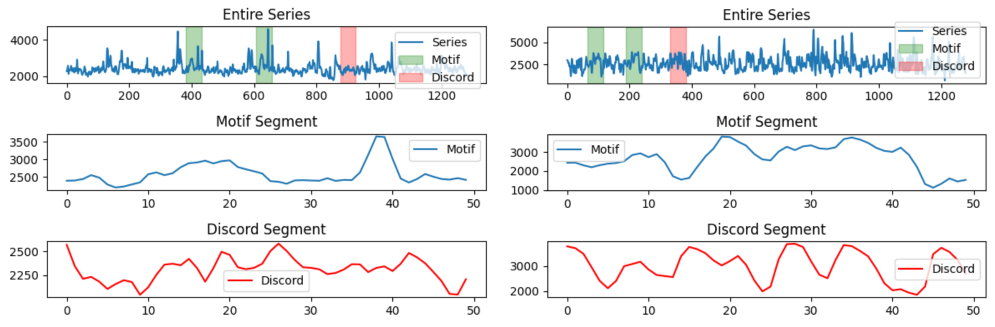

Figure 10: Time Series - Motif & Discords

Then, we applied a series of transformations to the time series of the centroids. This included using Local Outlier Factor (with n_neighbors = 20) to remove anomalies, scaling the amplitude using StandardScaler to standardize the data scales, and adding Gaussian noise to simulate unstructured variations in the data. Finally, we searched for motifs and discords to visualize their differences.

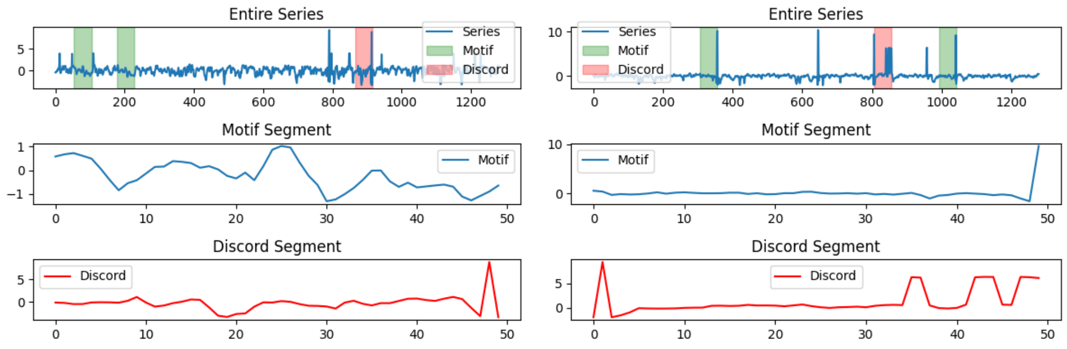

Figure 11: Time Series - Motif & Discords after transformation

In conclusion, we wrapped up our analysis by revisualizing the clusters using PCA. We observed that the behavior of our data was completely different compared to the previous analysis, see Figure 12.

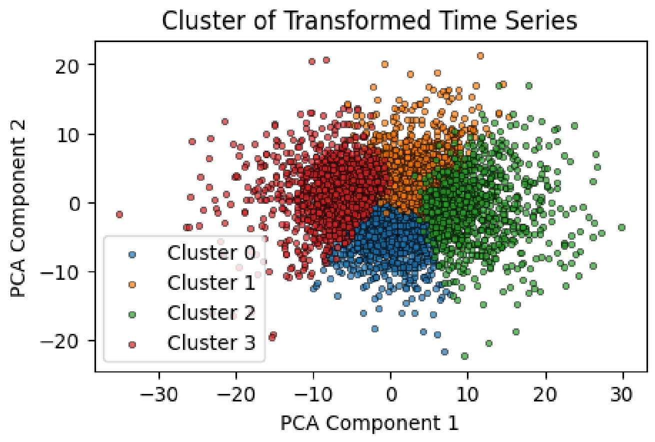

Figure 12: Cluster after transformation

2.3 Classification

The K-Nearest Neighbors (KNN) has proven to be an effective method for the classification of this type of data, thanks to its ability to adapt to different distance metrics. In this part of the report, we explore the application of KNN on time series (TS), comparing the effectiveness of two distance metrics: Euclidean and Dynamic Time Warping (DTW). We will also see the main difference between Rocket e Minirocket.

The entire preprocessing part is exactly the same as that done previously on motifs and discords. In addition to the preprocessing phase, PAA (Piecewise Aggregate Approximation) was applied, reducing the segments to 100, purely for computational reasons. The implementation of KNN was carried out using the KNeighborsTimeSeriesClassifier class from the tslearn library, specifically designed to work with time series. At this point, to find the most optimal value of K possible, we set a range from 1 to 30, and for each value, we performed cross-validation (see Figure 13).

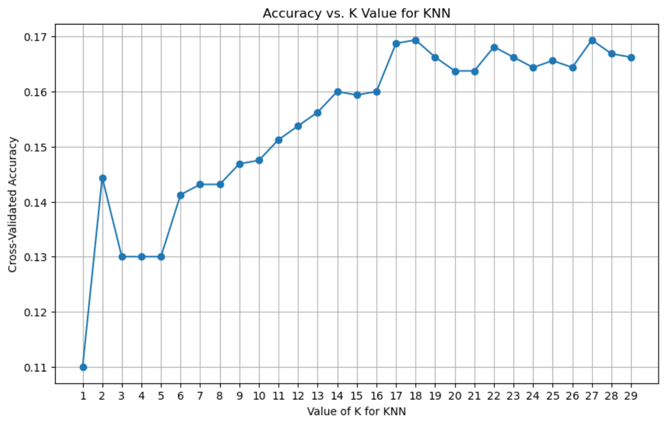

Figure 13: The best K

To visualize the actual differences, we chose 4 values of K: 3, 19, 25, and 27. Below we can observe the ROC Curve, showing the performance of our K-Means with k = 3 and k = 25 (see Table 1). As demonstrated, the best K values were 25 and 27, with practically identical results.

K	**accuracy**	**Recall**	**F1-Score**
3	0.155	0.15	0.12
19	0.17	0.18	0.12
25	0.18	0.19	0.13
27	0.18	0.19	0.13

Table 1: Classification Report k-Means euclidean distance

With n_neighbors = 3 and DTW as the distance metric, the KNN model achieved an accuracy of 16.75% on the test series, demonstrating poor ability to distinguish between different classes of time series. Increasing n_neighbors to 25 slightly improved accuracy to 22.08%. This suggests that a higher n_neighbors value might be preferable for this dataset, allowing the model to focus on nearest neighbors and reduce the influence of outliers or noise.

Regarding Euclidean distance, the accuracy obtained was lower compared to DTW, at 15% with n_neighbors = 3 and 18% with n_neighbors = 25. These results indicate that DTW is a more suitable distance metric for this type of time series, as it can better capture similarities between patterns with temporal distortions. We also calculated the Silhouette score for knn with the two distance, and we achive a very bad result etheir for DTW and Euclidean: 0.07 and 0.05.

The behavior of the ROC Curve confirms the models' performance for some classes and contradicts it for others. As shown in Figure 14, some classes exhibit extremely high AUC with Euclidean distance, whereas with DTW, AUC consistently remains around 40-60%. This indicates that while the DTW model performs better in classification, the Euclidean model is still fairly effective in distinguishing between classes.

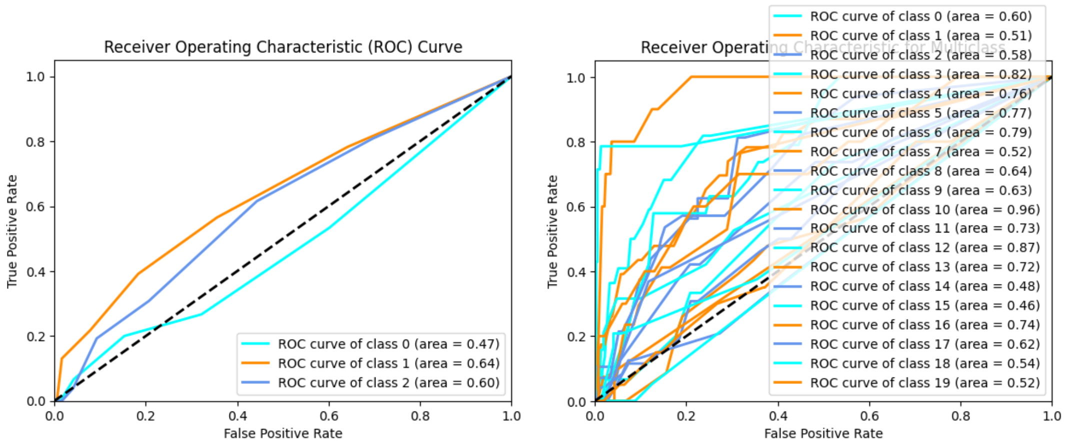

Figure 14: ROC KNN - DTW & Euclidean with k = 25

K-Nearest Neighbors (KNN) was explored for classification with two distance metrics: Euclidean and Dynamic Time Warping (DTW). DTW achieved better accuracy (22.08%) compared to Euclidean distance (18%) for KNN classification, indicating its suitability for capturing similarities in time series with temporal variations.

However, the ROC Curve analysis revealed inconsistencies. While DTW showed overall better classification performance, Euclidean distance remained effective for distinguishing specific classes.

2.3.1 Rocket e Mini-Rocket

We conducted data classification first using the Rocket algorithm and then with the Mini-Rocket algorithm.

To ensure optimal performance of Mini-Rocket, we implemented a GridSearch procedure, which allowed us to identify the best parameter as 1000. Rocket achieved an accuracy of 0.19, while Mini-Rocket demonstrated significantly better performance with an accuracy of 0.32.

These results confirm our initial hypothesis: Mini-Rocket offers superior performance compared to Rocket, due to its enhanced ability to capture relevant features from the data, thereby improving classification precision.

2.4 Shapelet

In this section of the report, we will analyze the shapelets to gain a better understanding of the most representative subsequences for each class. We will compare two approaches: ShapeletModel and DecisionTreeClassifier. The model was trained using ShapeletTransformClassifier, which achieved an accuracy of 21%. The parameters used were as follows:

- ShapeletTransformClassifier: configured to extract up to 100 shapelets, sampling 500 candidates.
- ShapeletModel: set to extract shapelets of lengths 10, 20, and 30.

The models were evaluated using accuracy and classification reports. Visualizations of the extracted shapelets and their importance were generated.

The ShapeletModel is configured to extract shapelets of three different lengths: 10, 20, and 30 points. For each length, 5 shapelets are extracted, for a total of 15 shapelets. The optimizer used is SGD (Stochastic Gradient Descent) with a weight regularizer of 0.01. The model is trained for a maximum of 500 iterations and a GridSearch was also performed to optimize the parameters, including the number of shapelets per size, the weight of the regularizer, and the maximum number of iterations. With this model, we achieved an accuracy of 22%.

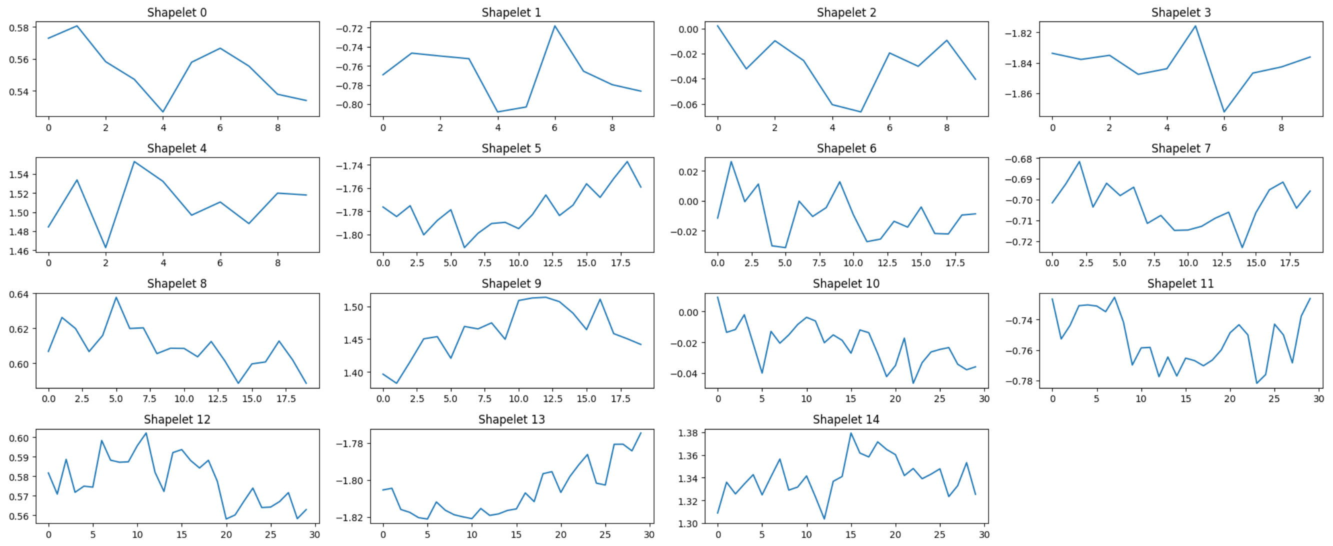

Figure 15: Shapelet estratte

The image 15 shows 15 graphs, each representing an extracted shapelet. Each shapelet represents a distinctive pattern that the model identified as relevant for classification. Shapelets 0-3 show shorter patterns (10 points) with rapid variations. Shapelets 4-8 represent medium-length patterns (20 points) with more details. Shapelets 9-14 are the longest (30 points) and capture more complex and extended patterns over time. Some shapelets show clear trends (e.g., Shapelet 13 with an upward trend). Other shapelets capture oscillatory patterns (e.g., Shapelets 6 and 7).

The diversity in the shapes of the shapelets suggests that the model has identified a variety of discriminative features in the time series.

We also calculated the importance of each shapelet to understand which ones were the most impactful, and as a result, we had: 5-9-12-13-14. Shapelet 5 (Estimated importance: 2246.2218 - class: new-age): has a length of 20 and shows an undulating pattern with a general upward trend, having relatively wide oscillations. Shapelet 9 (Estimated importance: 3128.9610 - class: happy), also with a length of 20, shows an upward trend with some fluctuations and a pronounced peak towards the end of the sequence.

Shapelet 12 (Estimated importance: 3747.5220 - class: heavy-metal), with a length of 30, shows more frequent and irregular oscillations with a notable peak around point 10. Shapelet 13 (Estimated importance: 5472.0021 - class: new-age), with a length of 30, starts with a downward trend, followed by an upward trend in the second half and shows more subtle oscillations compared to other shapelets. Finally, shapelet 14 (Estimated importance: 6670.6208 - class: happy): has a length of 30 points and shows the most pronounced oscillations among all the shapelets, with several distinct peaks and valleys along the sequence.

We can observe how the estimated importance increases with the length of the shapelet and the complexity of the pattern. Longer shapelets (30 points) tend to have higher estimated importance. More complex patterns with more oscillations seem to be considered more important by the model. Each shapelet shows a unique pattern, suggesting that they represent different distinctive features of musical genres.

The shapelets for ”new-age” (5 and 13) and ”happy” (9 and 14) tend to have broader and less irregular patterns compared to that of ”heavy-metal” (12), perfectly representing the differences in the musical characteristics of these genres.

As mentioned in the introduction of this chapter, we also used the DecisionTreeClassifier with this parameter: n_estimators = 100, random_state = 42 but its results were not satisfactory. We achieved an accuracy of 13.8%, which is lower than the first model used. We had a slight improvement with optimization during the hyperparameter tuning phase, reaching 15%. We also tried to further improve the results by using model ensemble methods, first with Random Forest and then with Bagging. The former returned more promising results, with an accuracy of 25.4%, while with Bagging we reached 18.5%, which are still better results than previously obtained.

2.4.1 Relation between Motif and Shapelet

Shapelets and motifs were extracted in different ways: shapelets represent the most discriminative subsequences for each class, while motifs are the centroids of clusters. Even though they are identified in different contexts, they can be compared to check if the patterns in the clusters are reflected in the shapelets used for classification.

For example, shapelet 9 and shapelet 12 show distinctive patterns for the ”happy” and ”heavy-metal” genres. As we can see in image 10, the behavior of our time series is comparable to that of shapelet 9 in image 15. This suggests that they have a similar pattern, and it can be concluded that the shapelets indeed capture the distinctive characteristics of this genre.

To further support this, we have confirmation with the second time series in image 10 and with shapelet 12 in image 15. The behavior of both time series is characterized by many peaks, and the corresponding genre in both cases is heavy-metal.

The coincidence between shapelets and motifs confirms the validity of the identified patterns. This integration of approaches provides a more comprehensive and accurate view of the data, improving both the clustering and classification phases.

3 Advanced Data-Preprocessing

For the following analysis, we consider the Track dataset after data understanding, which involved removing columns deemed insignificant for analysis, handling duplicates, and addressing missing values.

3.1 Outliers detection

By performing a preliminary analysis of the truck dataset we detected the presence of outliers. We focused on significant variables, excluding columns that have outliers but are not relevant for our analysis, such as 'disc_number', which indicates that some albums correspond to dozens of discs. We considered the features: 'duration_-min', 'popularity', 'danceability', 'energy', 'loudness', 'speechiness', 'acousticness', 'instrumentalness', 'liveness', and 'valence'. We displayed their respective boxplots and observed that only some of them contained outliers, so exclude those without outliers. Therefore, we created a subset with the features: 'duration_min', 'danceability', 'loudness', 'speechiness', 'instrumentalness', and liveness'. As we said before, we removed duplicates and reduced our dataset to 89.562 records, which we analyzed to find 1% outliers, totaling 895 records. Another step we took was to normalize and standardize the data.

We used the following methods for outlier detection: K-Nearest Neighbors (KNN), Histogram-based Outlier Score (HBOS), and Isolation Forest. Subsequently, we represented these four techniques in 3D plots using PCA as the dimensionality reduction technique. In the images below, we can see the three principal components derived from PCA on the axes, and the outliers represented in red, showing how they differ depending on the technique used. In the KNN plot, we can observe that there is no clear distinction between outliers and normal points. This indicates that we may have found local outliers, points that are anomalous relative to their immediate neighbors. This pattern suggests that KNN is effective at identifying anomalies that are not globally isolated but differ significantly from their surrounding data points. However, it should be considered that the separation between outliers and normal points is not always clear and could lead to the identification of false positives, that is, points erroneously classified as outliers. Regarding the Isolation Forest, the outliers are concentrated in the upper corners of the plot and it creates a clearer separation between them and the normal points, compared to the other two algorithms. This technique excels at identifying anomalies that are significantly different from the majority of the data, making the separation between normal points and outliers more distinct and evident. However, it may not be very effective at capturing local contexts. The HBOS algorithm is graphically represented as a middle ground between the first two. It identifies both local outliers, close to normal points, and global outliers at the edges of the distribution. This dual capability highlights HBOS's strength in detecting a variety of outliers, from those subtly differing from their neighbors to those markedly distant from the core data distribution. However, it does not present a clear visual distinction like Isolation Forest and, similar to KNN, struggles to isolate the outliers.

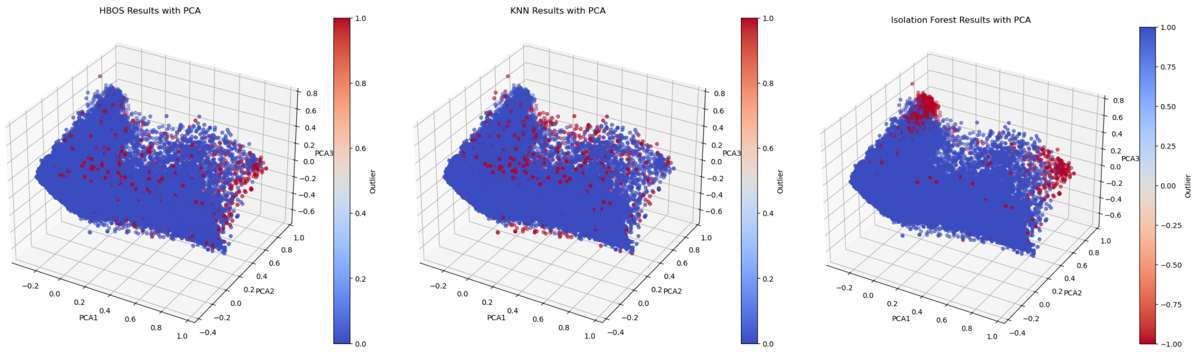

Figure 16: Enter Caption

Based on our analysis of the plots, Isolation Forest appears to be the best technique for global analysis. It provides a clear visualization of how data points are separated due to its ability to distinctly differentiate between outliers and normal points. KNN is useful for detecting local outliers, but as observed in the plots, it tends to mix outliers with normal points, making the separation less clear. HBOS represents a good compromise, identifying both global and local outliers. For our analysis, we preferred using Isolation Forest for outlier detection as we focused on the global analysis of the dataset. We decided to remove all outliers. This can be attributed to various goals such as dimensionality reduction and variance reduction to facilitate the generalization ability of certain predictive models. This second goal could also be achieved by modifying the detected outliers with the mean of the respective features, but we believe that replacing outliers with the mean can mask the unique and potentially significant characteristics of the data. Furthermore, outliers can represent rare but valid events that should be studied rather than modified. By removing them, we acknowledge that these observations do not fit the general pattern of the dataset without artificially altering the remaining data. We removed the outliers and verified that the removal was successful: the dataset decreased from 89.562 instances to 88.667 .

3.2 Imbalance learning

We begin by referring to the same dataset defined at the beginning of the study, from which we removed the outliers. For the classification task, we used KNN with 'explicit' as our target variable. The KNN parameters were optimized using Grid Search, exploring a range of neighbors from 3 to 10, with weights options of 'uniform' or 'distance', and distance metrics 'euclidean' or 'manhattan'. The best parameters identified were: 'metric': 'manhattan', 'n_neighbors': 9, 'weights': 'distance'. We are dealing with an imbalanced classification problem in the 'explicit' column used to train the model. This variable was chosen because 91.32% of the values are non-explicit, and only 8.68% are explicit. Consequently, during classification, most values were classified as non-explicit due to the significant disparity in the presence of the two values in the training data. To address this issue, we employed undersampling (Random undersampling) and oversampling (SMOTE) algorithms. For both techniques, we rebalanced the dataset to achieve a ratio of 60 (non-explicit) to 40 (explicit). By employing these techniques, we aimed to mitigate the imbalance and improve the performance and generalization of the KNN classifier. To evaluate the impact of Random OverSampler and SMOTE on performance, we implemented KNN on the rebalanced dataset.

Models	**accuracy**	**Recall**	**F1-Score**
KNN	0.92	0.94	0.90
Radom undersampler	0.80	0.67	0.37
SMOTE	0.79	0.67	0.35

Table 2: Classification Decision Tree

From the table, it is evident that KNN on the imbalanced dataset shows artificially high performance metrics due to overfitting. This is not a favorable result despite the high values, as it indicates poor generalization to new data. In contrast, both Random Undersampling followed by KNN and SMOTE followed by KNN provide more realistic performance metrics, albeit lower. These techniques aim to balance the dataset but come with trade-offs in terms of potential information loss (undersampling) or noise introduction (SMOTE). Random Undersampling followed by KNN performs slightly better than SMOTE followed by KNN. The higher accuracy and F1-score indicate that Random Undersampling, despite the potential for information loss, provides a better balance and overall performance for this particular imbalanced classification task.

4 Advanced Learning Techniques

In this chapter, we will see and analyze: Logistic Regression, Support Vector Machines, Neural Networks, Ensemble Methods, and Gradient Boosting, which will allow us to take a closer look at classification. Additionally, there will be advanced regression tasks and an explanation part.

4.1 Logistic Regression

We have examined the application of logistic regression to predict the popularity of music tracks based on predictive variables extracted from the provided dataset. The dataset used contains information on various characteristics of music tracks, including duration, acoustic features, and time metrics. After a careful data exploration, the following preprocessing operations were performed:

- Removal of columns not relevant to the analysis.
- Transformation of the target variable 'popularity' into a binary variable using a threshold of 0.4.
- Normalization of features using MinMaxScaler.
- Encoding of categorical variables through **one-hot encoding**.

The logistic regression model was trained using 70% of the data for training, while the remaining 30% was used for testing. The following steps were carried out:

1. Creation of the model using the LogisticRegression class from scikit-learn.
2. Training the model on the training data.
3. Evaluating the model's performance on the test data using the following metrics:

After training and evaluating the model, the following results were obtained:

Classifier	**Parameters**	**Accuracy**	**Recall**	F1-**score**
LogisticRegression	penalty='l2', C=1.0, solver='sag', max_iter=100	0.62	0.11	0.18

Table 3: Logistic Regression

In summary, the logistic regression model shows decent accuracy in predicting the popularity of music tracks but highlights significant issues in predicting tracks with high popularity. The low recall and F1-score for the high-popularity class indicate that the model struggles to correctly identify the most popular tracks.

4.2 Support Vector Machines

We are examining the application of Support Vector Machines (SVM) to classify music tracks into two categories based on the 'energy' feature: 'happy' and 'sad'. After a careful exploration of the data, the following preprocessing operations were performed:

- Removal of rows with missing values and duplicates.
- Creation of the target variable 'genre_happy_sad' based on the 'energy' feature.
- Selection of the relevant features for training.

The data was split into a training set (80%) and a test set (20%), maintaining class proportions using the stratify option. We standardized the data using StandardScaler to make the feature scales consistent and applied PCA to reduce dimensionality.

4.2.1 LinearSVC

Even with this model, the preprocessing phase remained the same. After preparation, the trained linear model achieved an accuracy of 86.8%, demonstrating its ability to effectively classify music tracks based on their "energy". We conducted another analysis using 1000 random samples and optimized the regularization parameter C with Grid Search to see if the results would change. Of course, all the preparation steps were the same as before: feature selection, normalization, and PCA application. The analysis resulted in excellent outcomes: 100% accuracy, and F1-score and recall scores of 1. As we mentioned, the accuracy achieved is 86.6% so we applied the PCA and after that, it drops to 76%. This might be due to overfitting, which caused the accuracy to be high initially and we cannot consider these results valid.

4.3 Neural Networks

In this section, we explored the use of neural networks for classification. We used a Multilayer Perceptron (MLP), a type of feedforward neural network, to predict the music genre category based on various audio features. We removed rows with missing values and duplicates to ensure data quality.

The selected columns were: 'duration_ms', 'popularity', 'danceability', 'energy', 'loudness', 'speechiness', 'acousticness', 'instrumentalness', 'liveness', 'valence' and 'tempo'. The data was split into training and test sets with a 70-30 ratio using stratification to maintain class distribution, and the data was standardized using StandardScaler to enhance model performance.

We decided to create a value map to categorize all genres into 22 groups, which helps give more consistency. The all new genres are: 'Electronic', 'Hip-Hop/Rap', 'Country', 'World', 'Indie', 'Pop', 'Rock', 'Jazz', 'Metal', 'Blues', 'R&B/Soul', 'Folk', 'Reggae', 'Reggaeton', 'Latino', 'Musical', 'New Age', 'Children', 'Classical', 'Comedy' and 'Ambient'. Every new categorization is based on the parent category, for example the genres 'kids' and 'disney' can be grouped in 'children' this because of the nature of the song.

We used MLPClassifier with a neural network structure consisting of a single hidden layer of 100 neurons, max_iter set to 500, regularization parameter set to 0.0001, and the solver used was "adam". These settings led to the following results:

These results show an accuracy of 46% and an F1-score of 0.44, indicating that the

Classifier	Parameters	Accuracy	Recall	F1-score
MLPClassifier	hidden_layer=(100,), max_iter=500, alpha=0.0001, solver='adam'	0.46	0.35	0.44

Table 4: Neural Networks

model performs moderately in classifying music genres. However, the ROC AUC of 0.79 (see Figure 17) suggests that the model has a good ability to distinguish between classes. To improve the model performance, a Randomized Search was conducted to optimize parameters. The optimized parameters included the size of the hidden layers and the regularization parameter alpha.

hidden_layer_sizes	alpha
(128, 64, 32), (128, 32), (64, 32), (100, 100, 100)	[0.1, 0.01, 0.001, 0.002]

Table 5: Parameter NN

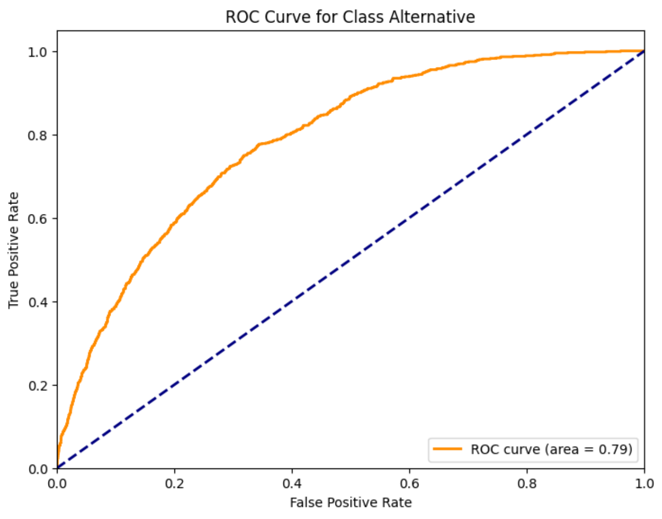

Figure 17: ROC NN

In a subsequent evaluation, we assessed two configurations of the MLPClassifier for music genre classification based on various audio features. In the first model, configured without early stopping, we achieved an accuracy of 47.65% and a weighted F1-score of 44.00%. This model exhibited varied performance across different classes, with F1-scores significantly varying depending on the music genre.

In the second model, we introduced early stopping to improve model regularization. However, this led to a slight decrease in overall performance, with an accuracy of 46.02% and a weighted F1-score of 43.89%. Again, we observed notable variation in F1-scores among classes, indicating the challenge of adapting the model to the heterogeneous characteristics of different music genres.

Accuracy	F1 score
0.4764	[0.05921053 0.63470032 0.16727273 0.44814341 ...]

Table 6: 1° Modello

Accuracy	F1 score
0.46015	[0.02881152 0.63741007 0.44366608 0.47568988 ...]

Table 7: 2° Modello

An interesting aspect to analyze is the Hyperparameter tuning performed with RandomizedSearchCV. From the search, the best parameters found are hidden_layer_sizes = (64, 32) and alpha = 0.001. These parameters indicate that the neural network has two hidden layers with 64 and 32 neurons respectively, and a regularization parameter (alpha) set to 0.001. As a result, the accuracy is 0.63 and the F1-score is 0.55, demonstrating a noticeable improvement in performance.

4.4 Ensemble Methods

In this subsection of the report, we will examine the application of ensemble methods, specifically the RandomForestClassifier, for classifying music genres based on audio features. As always, the preprocessing phase was conducted in the same manner as the previous techniques. The model was trained using 100 trees, each using the **Gini** splitting criterion and without limits on maximum depth. We set the minimum number of samples required to split an internal node to 2, and the minimum number of samples required to be at a leaf node to 1, aiming to ensure good model generalization without excessive overfitting.

After training and evaluating the model, the following results were obtained: The model achieved an accuracy of 53.7%. The F1-score varies significantly across different classes (see Table 8). After conducting this analysis, we attempted to modify various parameters (see Table 9)

Accuracy	F1 score
0.537	[0.25711275 0.68529607 0.35087719 0.53164557 ...]

Table 8: Ensemble Methods

Option	max_features	max_depth	Accuracy	F1-score
1	4	None	0.537	from 0.12 to 0.88
2	log2	None	0.537	from 0.09 to 0.88
3	log2	4	0.33	0.22

Table 9: Ensemble Methods differences

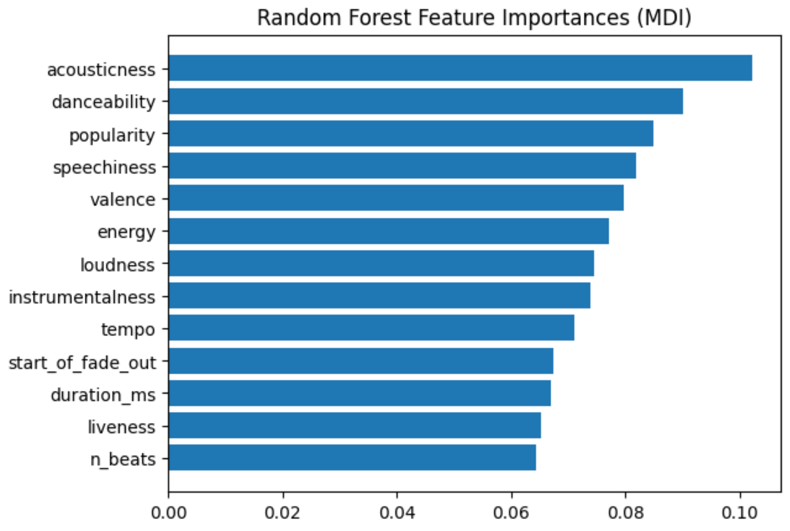

Figure 18: Random Forest

The image (see Figure 18) indicates that the most important features for the model are acousticness, danceability, popularity, speechiness, and valence. This suggests that these features are crucial for distinguishing between different types of music, unlike duration_ms and liveness.

We then used a decision tree with the same parameters as the random forest to highlight the potential differences we encountered. With this model, we achieved an accuracy of 39.2%, which is not exactly a strong result (see Figure 19).

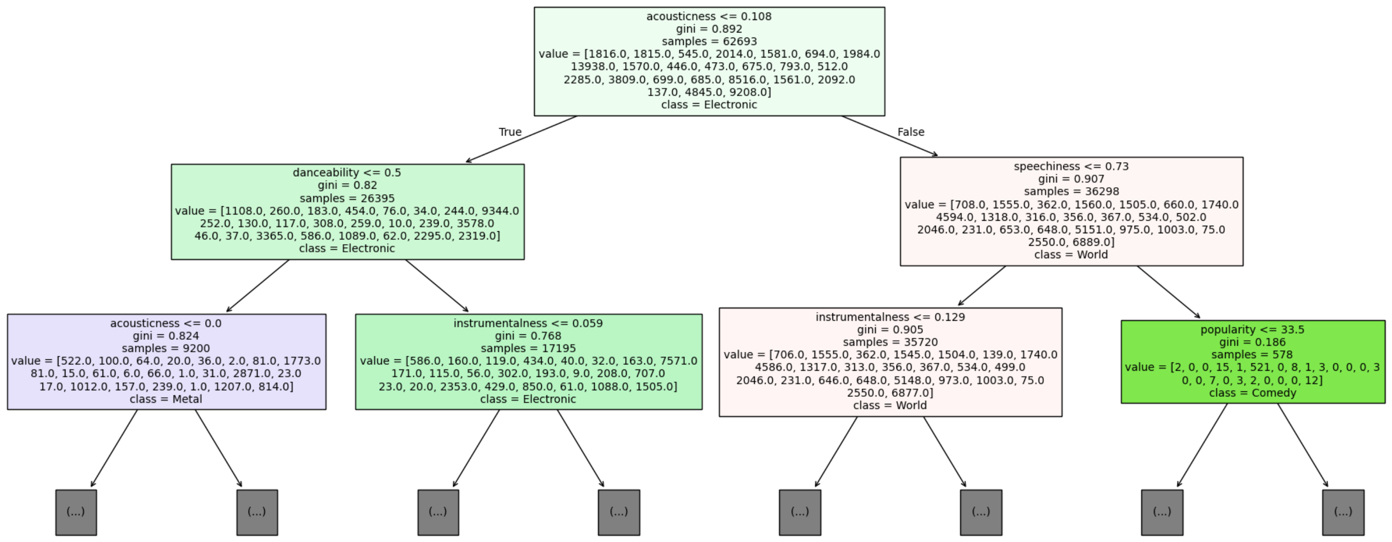

Figure 19: Decision Tree

4.4.1 Bagging

Different approaches were then explored using the BaggingClassifier with variable estimators. Two distinct models were trained: the first using RandomForestClassifier and the second using SVC with a parameter C = 100. The results obtained indicate a similar performance between the two models. Both achieved an overall accuracy of approximately 52.8%, with F1-scores ranging from 0.09 to 0.88 and a weighted average of 0.51. These findings suggest that the use of BaggingClassifier, regardless of the base estimator used, led to comparable results in terms of predicting various music genre classes.

4.4.2 Boosting

We also examined the effectiveness of AdaBoost using different types of estimators. Three distinct models were trained: the first using a generic base estimator, the second using RandomForestClassifier with 30 estimators, and the third using RandomForestClassifier with 10 estimators.

The generic base estimator achieved an accuracy of 23% and an F1-score of 0.24. Regarding the model with 30 estimators, it showed improved predictive capability with an overall accuracy of 53% and a weighted average F1-score of 0.51. Lastly, the model with 10 estimators yielded comparable results to the 30 estimators model, although there was a slight reduction in performance. This suggests that a lower number of trees can still provide good predictive capability without significantly compromising performance.

In conclusion, the AdaBoostClassifier has proven to be an effective method for enhancing the performance of weak classifiers, as demonstrated by the results obtained with various estimators. The use of a RandomForestClassifier as the base estimator led to a significant improvement compared to using a generic base estimator, suggesting that the choice of base estimator is crucial for achieving optimal performance in complex classification applications.

4.5 Gradient Boosting

In the field of music analysis, automatic classification of music genres plays a fundamental role in various applications, including personalized song recommendation, automated playlist creation, and music trend analysis. The primary goal of this chapter is to use and evaluate a gradient boosting-based classification model to predict the musical genre of an audio track based on acoustic and temporal features. As in previous subsections, the preprocessing phase remained consistent.

The model achieved an overall accuracy of 46%, indicating it correctly predicted approximately 46% of instances in the test set. Below are the precision, recall, and F1-score metrics for some of the most representative genre classes in our dataset:

These metrics demonstrate significant variations in the model's performance across different genre classes. For example, the model achieved high precision and recall for the "Electronic" genre, indicating a strong ability to distinguish this class. In contrast, the "Pop" genre showed lower performance in terms of precision and recall.

Genre	Accuracy	Recall	F1-score
Ambient	0.67	0.61	0.64
Metal	0.62	0.67	0.65
Electronic	0.53	0.73	0.61
Pop	0.37	0.39	0.38

Table 10: Gradient Boosting

The model was trained using parameters such as n_estimators = 100, learning_rate = 0.1, and max_depth = 3.

4.5.1 XGBoost

We extended our analysis using XGBoost, which achieved an accuracy of 44%. As we can notice, the model performed slightly worse on these genres compared to the previous model. Of course, the numbers are not excessively negative, so the results remain acceptable.

Genre	Accuracy	Recall	F1-score
Ambient	0.63	0.59	0.61
Metal	0.60	0.68	0.64
Electronic	0.50	0.73	0.59
Pop	0.35	0.38	0.36

Table 11: XGBoost

4.5.2 LightGBM

Another highly efficient boosting technique is LightGBM. The LightGBM model achieved an overall accuracy of 46% on the test set, aligning with the results obtained using Gradient Boosting. These metrics indicate that LightGBM has achieved slightly better performance compared to previous models so far, with some variations in performance across individual classes.

Genre	Accuracy	Recall	F1-score
Ambient	0.67	0.62	0.64
Metal	0.61	0.68	0.64
Electronic	0.52	0.73	0.61
Pop	0.37	0.39	0.38

Table 12: LightGBM

4.5.3 Cat Boost

The CatBoost model achieved an overall accuracy of 41% on the test set. The metrics are as follows (see Table 13). These metrics indicate that although Cat-

Boost achieved lower accuracy compared to other models, it still showed decent performance for some classes.

Genere	Accuracy	Recall	F1-score
Ambient	0.52	0.53	0.53
Metal	0.56	0.67	0.67
Electronic	0.32	0.36	0.34
Pop	0.46	0.72	0.56

Table 13: Cat Boost

4.6 Conclusion

Comparing the four boosting models used, we can observe that the importance of features continues to be a crucial point for all models. Each model has shown strengths and limitations, with variations in performance across different genre classes. To determine the best model among those used, it is very useful to take a look at the table (Table 14). Considering the performance metrics, Gradient Boosting appears to be the best model.

Model	Accuracy	Recall	F1-score
Gradient Boosting	0.46	0.35	0.38
XGBoost	0.44	0.31	0.33
LightGBM	0.46	0.35	0.37
Cat Boost	0.41	0.24	0.36

Table 14: Models

5 Advanced Regression

In this section of the report, we will employ non-linear regression techniques. Specifically, we have chosen to use the lightGBM and AdaBoost models for their robust performance in handling complex relationships and improving predictive accuracy.

We defined ”popularity” as the target variable because we consider it the most significant numeric feature. Initially, we analyzed the correlation of the target column with 5 selected independent variables: ”danceability”, ”loudness”, ”liveness”, ”valence”, and ”tempo”. We chose these features for their perceived importance in predicting popularity. Subsequently, we extended the analysis to include 9 independent variables: ”duration_min”, ”danceability”, ”loudness”, ”liveness”, ”acousticness”, ”instrumentalness”, ”speechiness”, ”valence”, and ”tempo”. This was done to assess whether adding more variables could improve the performance of the two regression algorithms, which it did. Both algorithms, lightGBM and AdaBoost, were first applied to the 5 variables and then to all 9, and the results were presented in a subsequent table.

Regression Algorithms	N° regressors	MAE	MSE	R2
LightGBM	9	16,49	413,09	0,1158
LightGBM	5	17,34	439	0,0604
AdaBoost	9	17,67	449,36	0.0382
AdaBoost	5	17.91	458,47	0,0187

Table 15: LightGBM & AdaBoost

From the evaluation of the performance of the individual models, it emerges that LightGBM with 9 regressors is the model with the best performance. LightGBM shows the lowest MAE and MSE values and the highest R^2 value. However, an R^2 value of 0.1158 is not very positive, as it indicates that the model explains only about 11.58% of the variance in the target variable. This suggests that while LightGBM with 9 regressors performs better than the other configurations, here is still significant room for improvement in explaining the variance in popularity.

6 Explainable AI

In this section, we will use explainability tools to clarify a classification model introduced in the previous chapter: the SVM model distinguishing between happy and sad classes. The target variable is defined as in the previous chapter: if the energy value is greater than a certain threshold (0.6), it is classified as happy; otherwise, it is classified as sad. We will examine this model using two different methods: a local approach (LIME) and a global approach (TREPAN). In both analyses, we refer only to the continuous variables of the dataset 'track' after data understanding: 'duration_min', 'popularity', 'danceability', 'energy', 'loudness', 'speechiness', 'acousticness', 'instrumentalness', 'liveness', and 'valence'. Additionally, we used the MinMax scaler to normalize our data.

6.1 LIME

We selected a subset of 600 observations representative of the data. To interpret the SVM model introduced earlier in the chapter, we employed LIME (Local Interpretable Model-agnostic Explanations). It was found that the model has a 63% probability of predicting a song as 'happy'. The feature that significantly contributes to classifying a song as 'happy' is loudness, with a SHAP value of 0.58. If a song has a loudness below 0.37, there is a higher likelihood that it will be classified as 'happy'. Similarly, danceability and duration_ms also play significant roles. Conversely, the variable 'acousticness' tends to classify a song as 'sad'.

After conducting this preliminary analysis to assess the role of selected features in classification, we opted to use Isomap (see Figure 20) as our data visualization tool. In the bidimensional graph, the two classes are visualized: each point represents an instance of the dataset reduced to two dimensions. Yellow points depict instances classified as 'happy', purple points represent 'sad', and the red point signifies a specific instance selected for explanation using LIME. The two classes show a lack of complete separability in the two principal components identified by Isomap. Additionally, areas of high point density in either yellow or purple highlight regions where each class is concentrated. Regarding the red point selected for LIME explanation, its isolated placement from the main cluster suggests it could be an outlier or a mispredicted instance, this is because it belongs to a region in the feature space where the 'happy' class is underrepresented.

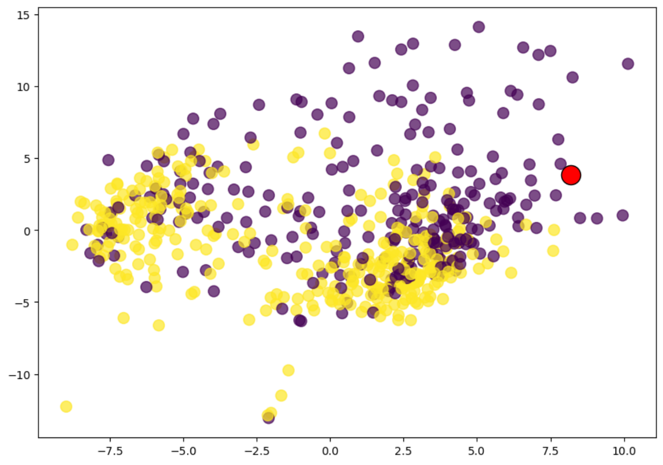

Figure 20: Lime

In conclusion, LIME revealed limitation in the model's ability to effectively classify instances as 'happy' or 'sad', indicating potential overfitting where the model memorizes training data rather than learning a general separation between classes.

6.2 TREPAN

We decided to use the TREPAN model to extract more information from the classification performed in Chapter 7 using a neural network (Section 7.3). This approach helps us better explain how the classification of the 21 genres described in the previous chapter was achieved. The decision tree provides good interpretability of the neural network model, clearly showing how decisions are made based on various features. For better visibility, we set the variable max_leaf_nodes to 4, as we believe this choice offers a clearer representation of the results. In the various leaf nodes, we can see: the attribute and threshold used to split the data, the Gini index of each node, the number of samples, the distribution of samples and the predominant class in the node. Focusing on the first node, we notice a Gini value of 0.89, indicating high impurity, meaning a fairly uniform distribution of different classes within the node. This could suggest that the neural network initially has difficulty clearly distinguishing between the classes. However, it is not only the first leaf that has a high Gini value, even in the second node on the right and left no improvements are seen, particularly in the right node, which seems to have worse performance than the root. Focusing on the root split feature, we notice that the first distinction is based on acousticness being less than or greater than 0.107, followed by splits on danceability and instrumentalness A significant improvement in the Gini index is shown in the bottom left leaf with a value of 0.41. In this leaf, the data has acousticness <= 0.1, danceability <= 0.5, and instrumentalness

<= 0.05, with the predominant class being "Electronic" with 4086 samples.The progressive reduction of the Gini index in the final nodes suggests that the decision tree is successfully identifying more precise rules for classification.

In conclusion, the distribution of classes in the various nodes shows that the neural network correctly classifies samples as the splits progress. The predominant class in many nodes is "Jazz," indicating that this class is well represented and perhaps better classified than others. Certain features, such as instrumentalness and danceability, are more effective in reducing impurity and improving classification.

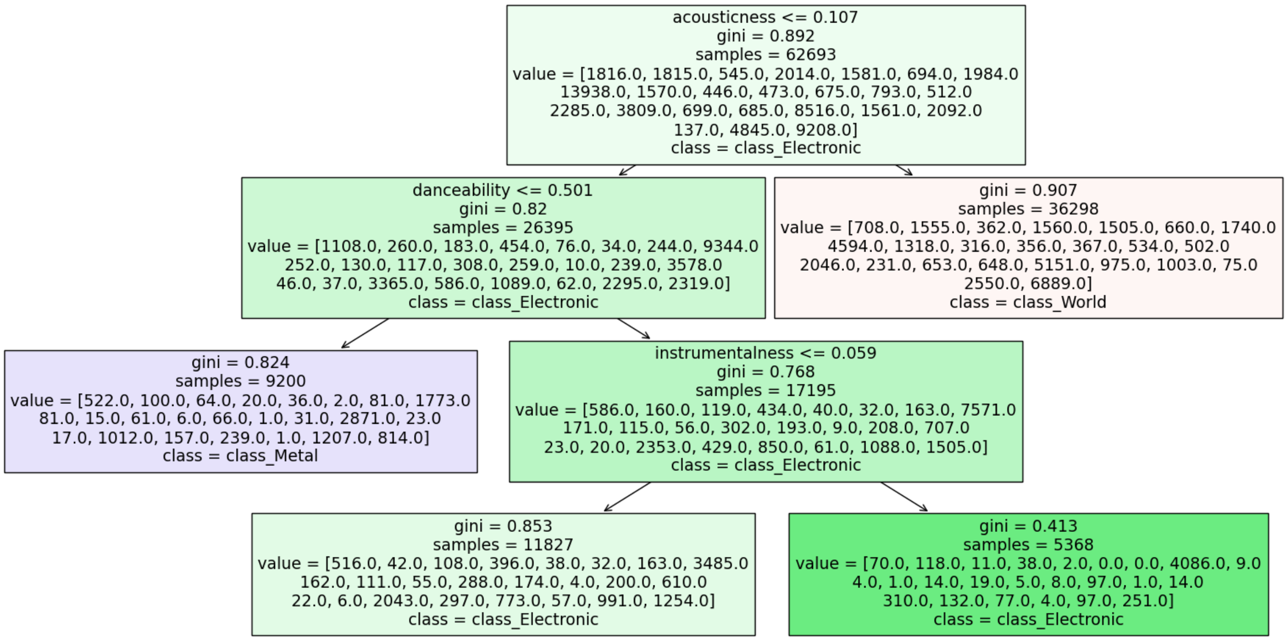

Figure 21: Trapan