Data Mining II Project

A Comprehensive Data Mining Analysis on a (larger) Spotify Dataset

Fabio Melasi, Giulio Leonardi, Matilde Contestabile

A.A. 2023/24

UNIVERSITÀ DI PISA

Contents

1 Data Understanding and Preparation

In this section, we introduce the three datasets utilized in our study: *Artists*, *Tracks* and *Time Series*. First, we delve into the two tabular datasets, offering either a broad overview and detailed insights into the optimizations we have implemented to enhance future performance. Our efforts extended beyond mere removal of missing values and outliers: we focused keenly on feature selection and addressed duplicate entries – two crucial aspects within the Tracks dataset. Moreover, we constructed a consolidated dataset by merging information from Artists and Tracks datasets, aiming to leverage the interplay of data across both sources.

Finally, we present the Time Series dataset, providing detailed insights into the strategic approximation decisions made to strike an optimal balance between efficiency and performance.

1.1 Artists Dataset

1.1.1 Description

The artists dataset comprises 30.141 records across five dimensions. Each row represents an individual artist, characterized by their name, popularity, follower count, produced music genres, and a unique identifier. Table 1 provides a more detailed description of each feature, while Table 2 provides general statistics about the numerical features.

Attribute	Description	Data type	Example values
id	Unique artist identifier	String	*4bs90uhVxZFrNaqUoajuuQ*
name	Name of the artist	String	*Jack White, Cage the Elephant*
popularity	Popularity score given by Spotify in a range 0-100	Integer	*64, 95*
followers	Number of followers on Spotify	Integer	*139, 2890131*
genres	Typical musical genres of the artist	List of Strings	*['alternative rock', 'blues rock']*

Table 1: Feature description of the artists dataset.

Attribute	Type	Mean	Median	Mode	STD	Range
popularity	Discrete	36.66	37	41	17.19	0 - 100
followers	Discrete	4.16e+05	1.6e+04	0	2.49e+06	0 - 1.14e+08

Table 2: Statistics on numerical features of the artists dataset

Based on the mean, median, mode, and standard deviation values from the preceding table, it seems that `popularity` follows a distribution resembling normality, whereas `followers` does not. Intuitively, `followers` may follow a power law. To gain further insights, we have plotted both distributions.

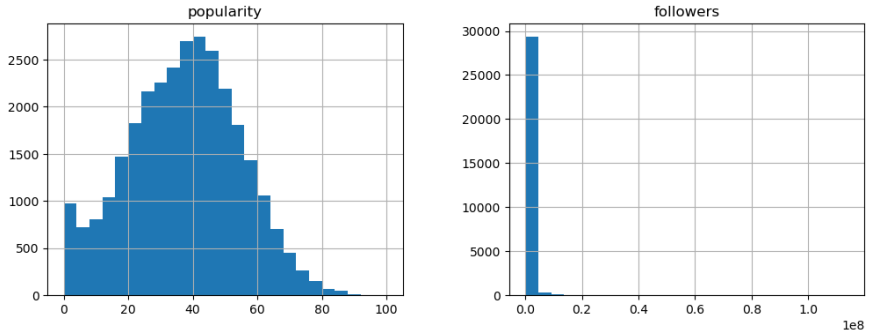


Figure 1: Histograms of `popularity` and `followers` features

.

The findings from Figure 1 validate our initial hypothesis regarding the variable `popularity`. Additionally, the analysis of `followers` aligns well with our expectations. Further validation will be sought in the upcoming section, where boxplots will be employed.

1.1.2 Missing values, duplicates and outliers

In the Artists dataset, there exists only one missing value, which is promptly removed from the analysis.

Although there are no instances of fully duplicated records within the dataset, a preliminary analysis revealed instances where artists, despite having different IDs, share identical names. While this could potentially signify cases of homonymy among artists, the occurrence is relatively infrequent, totaling 435 instances. Moreover, given that the name serves as the primary key facilitating the linkage between this dataset and the tracks dataset, we have decided to eliminate all records containing duplicate values in the `name` feature, for a total of 223 instances.

In order to handle outliers, we plotted the boxplots of the two numerical features of our dataset.

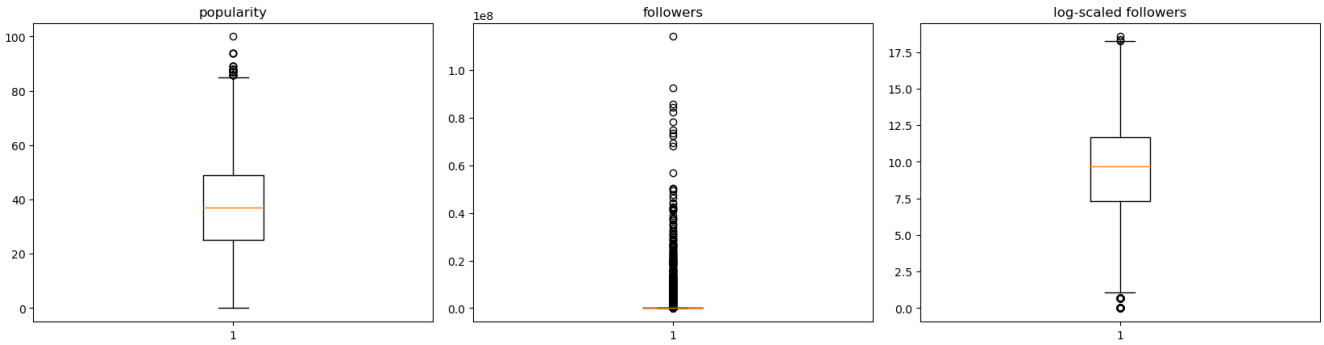


Figure 2: Boxplots of `popularity` and `followers` features

The presence of outliers in the `popularity` feature is notably minimal, predominantly attributed to a select few top-performing artists. Furthermore, the semblance of a normal distribution in the log-scaled distribution of the `followers` feature lends credence to the hypothesis that it follows a power law distribution. In light of these observations, we have opted to retain all records in both cases.

1.1.3 Final considerations

This dataset can be combined with the Tracks dataset (outlined in the following section) to address tasks requiring both artist and track information. To merge the datasets, we'll utilize the `name` attribute of the artists and the `artists` attribute of the tracks. It's worth noting that the Artists dataset contains 4,072 unique genres, whereas the Tracks dataset only includes 114 distinct music genres. This divergence stems from the Artists dataset offering more nuanced genre categorizations.

Further details on the manipulation of the Artists dataset, including normalization, will depend on the specific task to handle and will be discussed in the relative sections.

1.2 Tracks Dataset

1.2.1 Description

The Tracks dataset comprises 109.547 records across 34 dimensions, with each row representing a distinct music track. These dimensions capture various facets of the tracks, many of which have been previously discussed in the Data Mining I report. Therefore, Table 3 exclusively presents the additional features introduced in the expanded dataset.

Attribute	Description	Data type	Example values
id	Unique track identifier	String	*3dPQuX8Gs42Y 7b454ybpMR*
disc_number	Number of discs of the album the track belongs to	Integer	*1, 3*
track_number	Position of the track in the album	Integer	*1, 6*
album_type	Type of the album the track belongs to	String	*album, single, compilation*
album_name	Name of the album the track belongs to	String	*Elephant, Melophobia*
album_release_date	Date in which the album the track belongs to was released	String	*2013-10-08, 1987-08*
album_release_date_precision	Precision with which release_date value is known	String	*day, month, year*
album_total_tracks	Number of tracks in the album the track belongs to	Integer	*14, 9*
start_of_fade_out	Time in seconds at which the track's fade-out period starts. If the track has no fade-out, it's the track's length.	Float	*229.79, 170.97*
tempo_confidence	Score of reliability of the tempo feature, in a 0 - 1 range.	Float	*0.393, 0.766*
time_signature_confidence	Score of reliability of the time_signature feature, in a 0 - 1 range.	Float	*1, 0.7*
key_confidence	Score of reliability of the key feature, in a 0 - 1 range.	Float	*0.264, 0.808*
mode_confidence	Score of reliability of the mode feature, in a 0 - 1 range.	Float	*0.565, 0.040*

Table 3: Feature description of the tracks dataset.

Before delving into both numerical and graphical summaries of the dataset's numeric features, it's important to note that many distinct tracks are associated with multiple records. This occurs for several reasons, for example when a track spans different music genres, resulting in an instance for each genre while maintaining consistent the other features. Other examples of duplicate tracks are discussed in Section 1.2.3. To ensure meaningful statistics and prevent double counting of tracks, our

analysis is conducted on a condensed dataset where each track appears only once: the remaining total number of records stands at 84.547.

For readability and space reasons, we don't show the statistics related to the features that will be removed, which will be discussed in the next section.

Attribute	Type	Mean	Median	Mode	STD	Range
duration_ms	Continuous	230374.93	215026.0	180000	103445.91	8586 - 4120258
popularity	Discrete	32.85	32.0	0	20.85	0 - 95
danceability	Continuous	0.56	0.573	0.598	0.177	0.0 - 0.985
energy	Continuous	0.636	0.679	0.961	0.258	0.0 - 1.0
key	Discrete	5.28	5.0	7	3.56	0 - 11
loudness	Continuous	-8.56	-7.236	-5.879	5.275	-49.531 - 4.532
mode	Discrete	0.634	1.0	1	0.482	0 - 1
speechiness	Continuous	0.088	0.049	0.0324	0.115	0.0 - 0.965
acousticness	Continuous	0.328	0.188	0.995	0.34	0.0 - 0.996
instrumentalness	Continuous	0.181	0.000079	0.0	0.329	0.0 - 1.0
liveness	Continuous	0.219	0.133	0.111	0.197	0.0 - 1.0
valence	Continuous	0.465	0.452	0.961	0.263	0.0 - 0.995
tempo	Continuous	122.15	122.023	0.0	30.08	0.0 - 243.372
time_signature	Discrete	3.898	4.0	4	0.453	0 - 5
tempo_confidence	Continuous	0.444	0.412	0.0	0.302	0.0 - 1.0
time_signature_confidence	Continuous	0.874	0.987	1.0	0.22	0.0 - 1.0
key_confidence	Continuous	0.491	0.509	0.0	0.249	-1.0 - 1.454
mode_confidence	Continuous	0.511	0.522	0.0	0.183	-1.0 - 1.0

Table 4: Statistics on numerical features of the tracks dataset.

The numerical attributes exhibit vastly different distributions, as illustrated in the following figure.

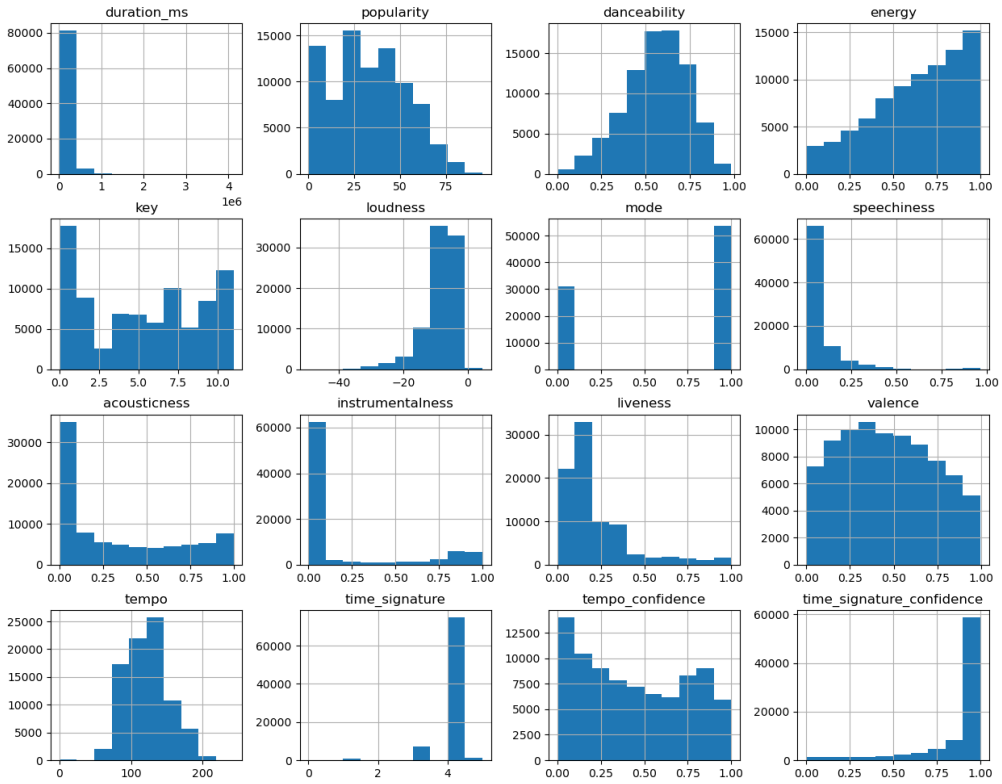


Figure 3: Histograms of the numerical attributes on the tracks dataset.

Instead of conducting an exhaustive analysis on the distribution of features, Figure 3 is presented simply to highlight the diverse distribution patterns exhibited by different attributes. It is evident that various attributes follow distinct distribution shapes, including power law distributions, normal distributions, and bimodal distributions. In summary, there is no overarching trend, and each attribute should be handled according to its unique nature.

1.2.2 Features Selection

Consistent with the observations detailed in the Data Mining I report, we have decided to exclude certain features from our dataset. Specifically, we have removed the `features_duration_ms`, `n_beats`, and `n_bars` attributes due to their significant correlation. Additionally, we have omitted the `processing` feature due to its reliance on the `key` attribute. However, unlike

the previous study, we have opted to retain the `popularity_confidence` feature this time, as no missing values were detected within this dataset.

To determine which new features could be pruned, we generated a correlation matrix. Our analysis revealed an exceptionally high correlation (0.999) between the `start_of_fade_out` and `duration_ms` attributes. Consequently, we made the decision to eliminate the former from our dataset.

We also observed a correlation of 0.800 between the `track_number` and `album_total_tracks` attributes. However, beyond their correlation, we chose to eliminate these features—and several others—due to their lack of significant information contribution. This decision was particularly motivated by the nature of the research tasks at hand, which primarily revolve around music genre identification. Our hypothesis suggests that such features, when not detrimental, hold negligible relevance in genre identification tasks. Hence, we pruned the following features from our dataset: `disc_number`, `track_number`, `album_type`, `album_name`, `album_release_date`, `album_release_date_precision`, and `album_total_tracks`[1]. The dataset now counts 23 dimensions.

1.2.3 Missing values, duplicates and outliers

No missing values are present in any of the attributes of the dataset.

Regarding duplicates, a few hundred identical tracks with different IDs were removed. However, the vast majority of duplicate tracks were associated with two cases:

- tracks belonging to different albums;
- tracks related to different music genres;

The first issue is automatically resolved upon removing the `album_name` feature from our dataset, as discussed earlier. By doing so, we can confidently eliminate duplicated tracks without sacrificing any pertinent information, given that the album name holds no relevance in this context.

Regarding the second issue, there are some considerations to weigh. In principle, retaining all records could be justified, as each provides insights into a specific genre, even if overlapping across different genres. However, to prevent overloading the models with redundant information, we opted to remove duplicate entries at this stage of analysis.

An immediate concern arises: when dealing with distinct tracks associated with multiple genres, how should we determine which genre label to preserve? Our approach involved employing a KNN algorithm to classify the most representative genre for each track based on its features among the genres it is affiliated with.

Following this procedure, the remaining number of records is 84.547. We assessed whether the distribution of the 114 genres underwent significant alterations, which, upon examination, proved not to be the case.

Concerning outliers, we utilized five outlier detection algorithms to cleanse our dataset. A comprehensive overview of this analysis will be provided in the next section.

1.2.4 Outliers Detection

This section delves into outlier detection analysis. Initially, we will explore the various techniques employed to identify outliers within the tracks dataset, accompanied by visual representations showcasing their distribution. Subsequently, we will delve into our strategy for effectively eliminating these outliers, thus concluding the preparation phase of our dataset.

Detection

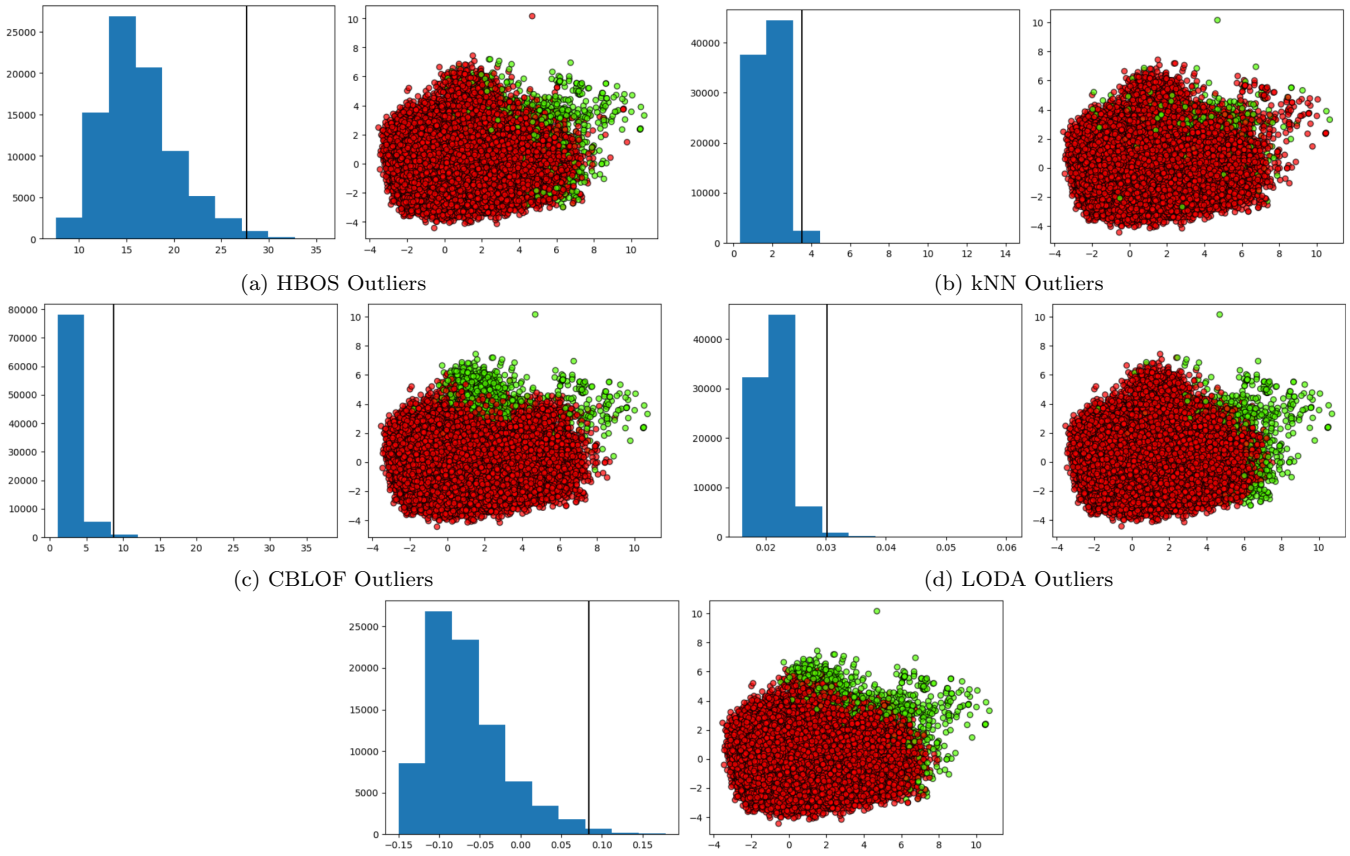
We explored five distinct outlier detection algorithms, each belonging to a different family. Beginning with a Visual approach, we employed *Histogram-based outlier score* (HBOS). Following this, we pursued a Distance approach utilizing *k-Nearest Neighbors* (KNN). Additionally, we experimented with a Clustering approach, implementing *Clustering Based Local Outlier Factor* (CBLOF). Transitioning to an Ensemble approach, we evaluated *Lightweight on-line detector of anomalies* (LODA). Finally, we investigated a Model approach, employing *Isolation Forests*.

Each algorithm generated an outlier score for each track in the dataset. Using these scores, we identified the top 1% of outliers. To visualize the distribution of these outliers, we reduced the dimensionality of our dataset using PCA and plotted all points, emphasizing those identified as outliers. The results are showed in Figure 4.

These plots vividly demonstrate how different algorithms identify distinct points as outliers. Specifically, in each plot, a dense cluster of outliers is noticeable in the upper-right quadrant, yet this cluster varies significantly from one algorithm to another. Notably, kNN exhibits the least uniformity, whereas CBLOF displays the least overlap. However, it's essential to recognize that these plots don't accurately reflect the spatial distribution of the data but rather represent a dimensionally reduced version of the real space. Consequently, outliers aren't necessarily distant points from the main distribution.

In the next section we illustrate the outlier elimination process.

[1] We get rid of these features aware that we may take back some of them in case of specific analysis not regarding the music genre.



(a) HBOS Outliers

(b) kNN Outliers

(c) CBLOF Outliers

(d) LODA Outliers

(e) Isolation Forest Outliers

Figure 4: The histogram on the left depicts the distribution of outlier scores generated by the algorithm. The vertical line denotes the threshold beyond which points are categorized as outliers. The scatterplot on the right illustrates the dataset's PCA representation, with outliers highlighted in green.

Outlier elimination

To determine which outliers to remove from the dataset, we examined the outcomes of each previously discussed approach. Interestingly, only 42 tracks were consistently identified as outliers by every algorithm. Conversely, when considering the combined set of outliers identified by each algorithm, the count amounted to 2401 instances. Seeking a balanced approach, we opted to remove only the tracks flagged as outliers by at least three out of the five algorithms, totaling 498 instances. The ensuing plot illustrates the varying distributions of outliers resulting from these combinations.

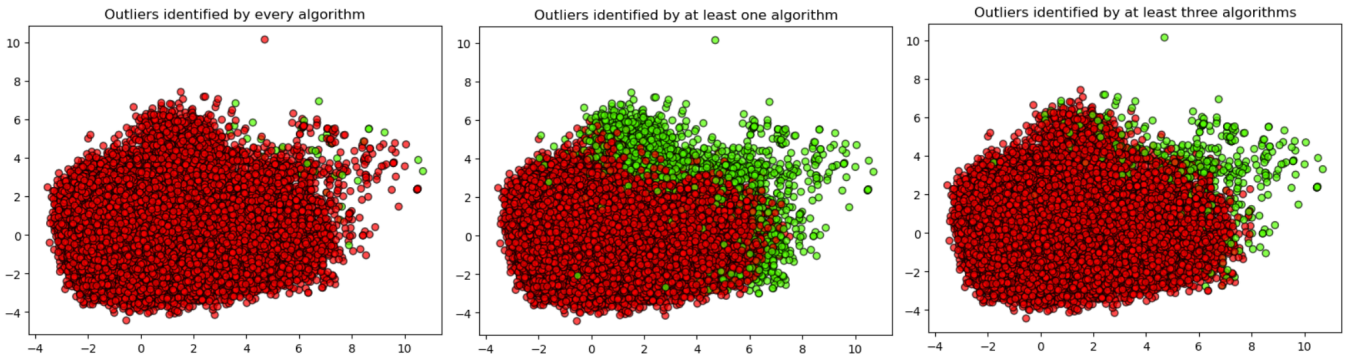


Figure 5: Outliers resulting from different combinations of the five previous methods.

1.2.5 Artists and Tracks datasets merging

In order to perform tasks based on the cross-datasets information between the Artists dataset and the Tracks dataset, we decided to merge them. This integration process inevitably led to some loss of information:

- artists present in the Artists dataset but lacking any associated tracks in the Tracks dataset;
- tracks existing in the Tracks dataset but without corresponding artists in the Artists dataset;
- tracks with multiple authors were ignored;

The merging operation was conducted using the `name` column from the artist dataset and the `artists` column from the tracks dataset. As a result, the merged dataset comprises 81.478 records spanning 35 dimensions.

1.2.6 Data splitting and normalization

In order to be used in machine learning contexts, our datasets (including the merged one) has been splitted into training and test set, equally distributing the records according to the `genre` attribute. Then, all of them were normalized using z-score standardization technique.

1.3 Time Series Datasets

1.3.1 Description and approximation

Our Time Series dataset comprises 10.000 sequences, each consisting of 1.280 data points. Each sequence represents the spectral centroids of a distinct track. Spectral centroids are computed using the `librosa` python package having in input the mp3 audio files of the songs. Each time series is linked to a specific genre, with 500 sequences assigned to each of the 20 genres.

Section 2 is entirely dedicated to Time Series Analysis. To achieve tasks within a reasonable timeframe while preserving essential information, we experimented with various approximation configurations. Following several attempts, we settled on utilizing PAA (Piecewise Aggregate Approximation) with 128 final data points as the most balanced approach. This process involves segmenting each time series into equal-length intervals and capturing the average value of the data points within each segment. By doing so, the 1280 dimensions of a track are condensed to 128, achieved through the division of the time series into uniform frames. Figure 6 displays an exemplary time series before and after the PAA approximation.

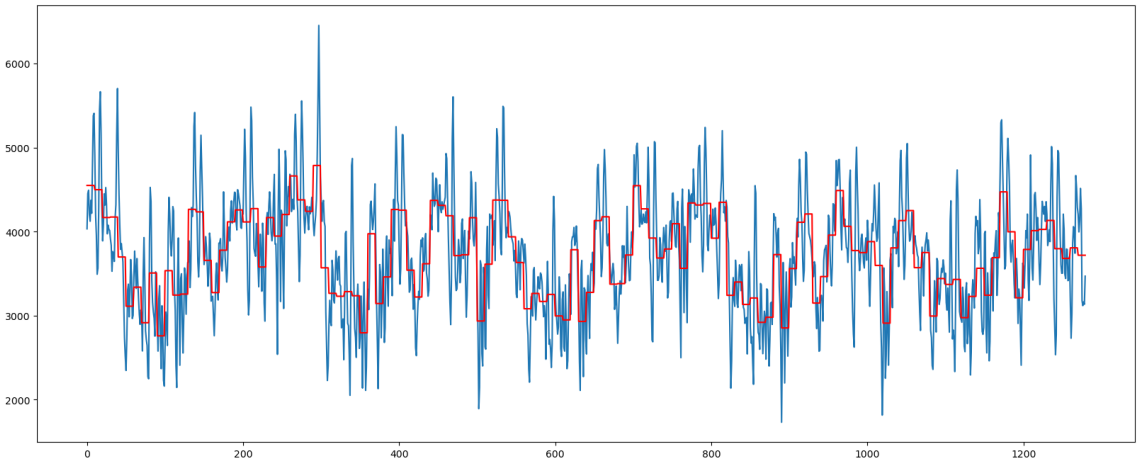


Figure 6: Example time series before (in blue) and after (in red) PAA approximation.

Lastly, to enhance possible comparability between datasets and prepare the data for future analyses, we employed amplitude scaling techniques, specifically leveraging z-score standardization. This adjustment aids in aligning the magnitudes of values within the time series data, ensuring more accurate comparisons across diverse datasets and optimizing their suitability for subsequent analyses.

1.4 Conclusions

In this section, we outlined our approach to handling and preparing the datasets slated for future analysis. Through a combination of qualitative narratives and statistical analyses, we provided an overview of the data, accompanied by justifications for any modifications made. We also decided to merge the two tabular datasets, laying the groundwork for integrated studies in subsequent chapters.

While we detailed our strategy for approximating and representing sequences in the Time Series dataset, it's important to acknowledge that not all our choices in this domain will prove correct. In the forthcoming chapter dedicated solely to time series analysis, we will revisit and refine our methodologies, aiming for a more robust and insightful exploration of the data.

2 Time Series Analysis

2.1 Clustering

In our time series analysis, we employed two distinct clustering methodologies. First, we utilized a Shape-based approach utilizing euclidean distance. Subsequently, we extracted multiple structural features from the dataset and implemented a Feature-based algorithm employing hierarchical clustering techniques. Through this analysis, we uncovered a significant characteristic within the time series of tracks that prompted us to reconsider our initial approach to this dataset preparation for improved outcomes. Furthermore, we will present a comparison of the two strategies implemented, highlighting the genre as a key discriminative feature for clustering.

2.1.1 Shape-based Clustering

The shape-based algorithm we decided to implement is the **Time Series K-Means** algorithm. While Dynamic Time Warping distance tends to yield superior results compared to Euclidean distance, we chose to employ the latter. This decision was influenced by the considerable computational overhead of DTW, even when employing global constraints. Additionally, given the relatively short length of our time series (due to approximation), we found Euclidean distance a valid alternative. Our initial task involved determining the optimal number of clusters, denoted by the parameter K.

Find the best K

To determine the optimal parameter K for K-Means clustering, we employed the elbow method. This technique involves plotting the Sum of Squared Error (SSE) explained as a function of the number of clusters. Below is the corresponding plot showcasing the elbow curve.

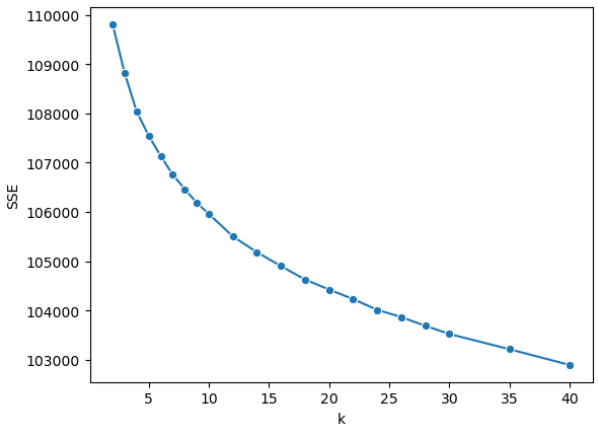


Figure 7: Plot based on elbow method intuition to find the best K.

Two closely linked observations arise from this visualization:

- The absence of a sharp elbow, as typically seen in similar analyses, complicates the selection of a specific value for K.
- A consistently high SSE is evident across various potential cluster numbers, indicating that no particular K stands out as notably superior.

These findings led us to restrict our focus to a small number of clusters, specifically three, so that they can be easier to analyze and probably more discriminative. Despite the initial discouragement from the results, we tried to explore whether these clusters exhibited any meaningful structure.

Visualization and evaluation

The three identified clusters have different numbers of components:

Cluster 0: 4213 Cluster 1: 1609 Cluster 3: 4178

Analyzing the results, we noted that these clusters exhibit a modest distinctiveness: rather than being randomly dispersed, music genres display associations with each cluster. This observation suggests that a song's genre is discernible by its track shape. A comprehensive breakdown of genre distributions will be provided in Section 2.1.3, where we juxtapose Shape-based clustering against Feature-based clustering. This comparison will illustrate how various genres are distributed among clusters derived from these two distinct methodologies. We wanted to gain further insights visually, so we used three different visualization techniques aimed at elucidating the cluster nature. The plots are showed below.

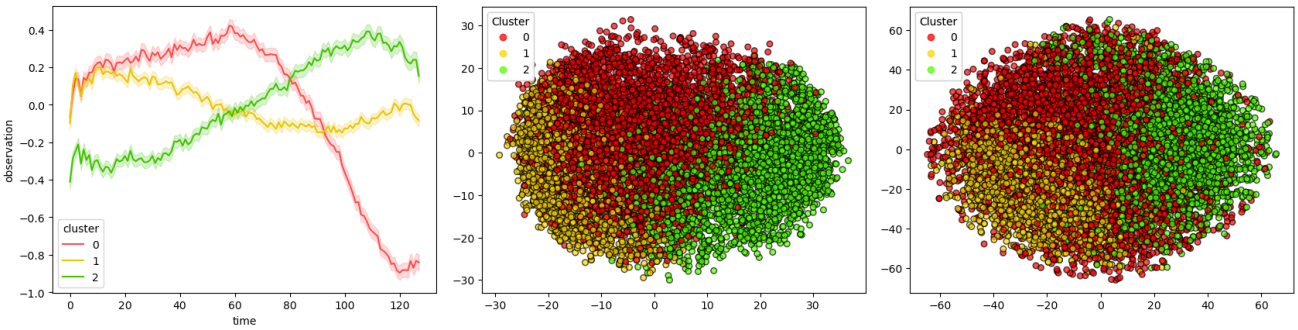


Figure 8: Shape-based clusters. From left to right: time series centroids of the three clusters; IsoMap representation; t-SNE representation.

The first plot provides a condensed time series for each cluster, that is a sort of centroid, revealing distinct trends within each group. Subsequent visual representations, depicted in the second and third figures, illustrate the results of applying dimensionality reduction techniques: IsoMap and t-SNE. IsoMap, known as Isometric Feature Mapping, aims to maintain the inherent geometric structure of high-dimensional data in a lower-dimensional space by utilizing geodesic distances. Instead, t-SNE, or t-Distributed Stochastic Neighbor Embedding, focuses on preserving both local and global relationships among data points. Consequently, nearby points in the original space tend to remain close together in the reduced-dimensional space.

This visualization, coupled with the notably discouraging silhouette value (0,03), left us perplexed. While the first plot effectively displays three distinct curves representing differently shaped clusters, the subsequent plots reveal a concerning number of overlapping points and clusters dispersed in various directions. Regrettably, the results obtained with the Feature-based strategy, which we are unable to present here due to space constraints, were even worse, with clusters appearing nearly indistinguishable.

These outcomes prompted us to explore the possibility of achieving improved clustering. An intuition arose: during the preparation phase (Section 1.3), we had assumed that the varying amplitudes of tracks were detrimental to our analyses and thus we applied amplitude scaling. However, we began to suspect that amplitude might actually play a significant role in cluster discrimination. Consequently, we reran the same algorithm using the time series dataset without amplitude scaling. The results we obtained were markedly superior, a fact corroborated by the silhouette value (0,28) and the subsequent visualizations.

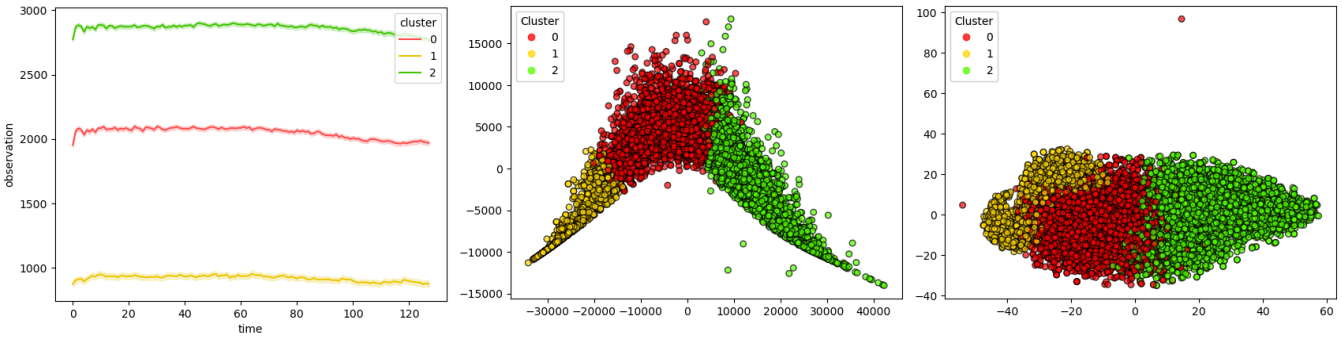


Figure 9: Shape-based clusters, without amplitude scaling applied to time series. From left to right: time series centroids of the three clusters; IsoMap representation; t-SNE representation.

The dimensionality reduction techniques have yielded plots with distinctly separate clusters. Moreover, there's a notable improvement in the clusters' ability to discriminate between different genres (further elaborated in Section 2.1.3). Particularly, the first plot, featuring three similar sequences with different amplitudes, convinced us that the improved performance is attributed to the fact that that different genres are deeply recognizable on the base of the amplitude of the tracks. This is an important result, as it provides a significantly powerful tool for genre identification.

2.1.2 Structural-based Clustering

In employing a features-based strategy for clustering time series data, the approach entails extracting structural features from the sequences and utilizing them as the dataset for clustering through a standard algorithm (in our case, Hierarchical Clustering was chosen). Thus, the initial phase involved extracting these structural features. The results presented are all based on the non-amplitude scaled time series, as they consistently deliver the best performance.

Feature Extraction

We derived fundamental statistical information from the time series data. Specifically, for each track, we computed the mean value, standard deviation, minimum, maximum, and five quantiles, reducing the dimension of the dataset from 128 dimensions to only nine. By engaging in this process, we unavoidably sacrificed certain information, notably the temporal aspect, with the expectation that the newly acquired structural insights would compensate for this loss and ensure effective clustering. The following section is devoted to determining the optimal model configuration to utilize.

Find the best linkage method

Hierarchical Clustering's effectiveness hinges significantly on the method used to compute inter-cluster similarity. Varying linkage choices can yield vastly divergent results. To identify the most optimal approach, we generated dendrograms for each of the four standard possibilities, emphasizing the delineation of clusters to determine an appropriate "cut".

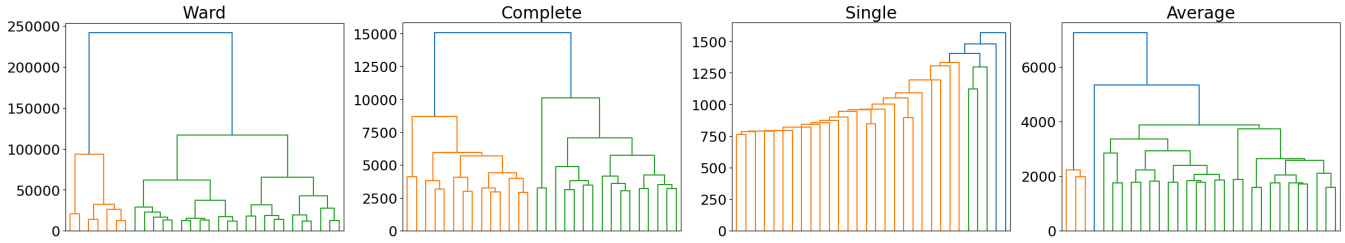


Figure 10: Dendrograms with different linkage methods for feature based clustering.

The dendrograms indicate that the Ward and Complete linkage methods perform better, in the sense that they both identify two well-separated clusters. Also silhouette suggests a good cluster quality for both strategies (0,51 and 0,28). Therefore, we delved into the specifics of these two clusters.

Visualization and evaluation

In this case, points in clusters are distributed as follows:

Ward

Cluster 0: 2296 Cluster 1: 7704

Complete

Cluster 0: 6425 Cluster 1: 3575

The visualization readily reveals the varying proportions of different clusters, each distinctly discernible.

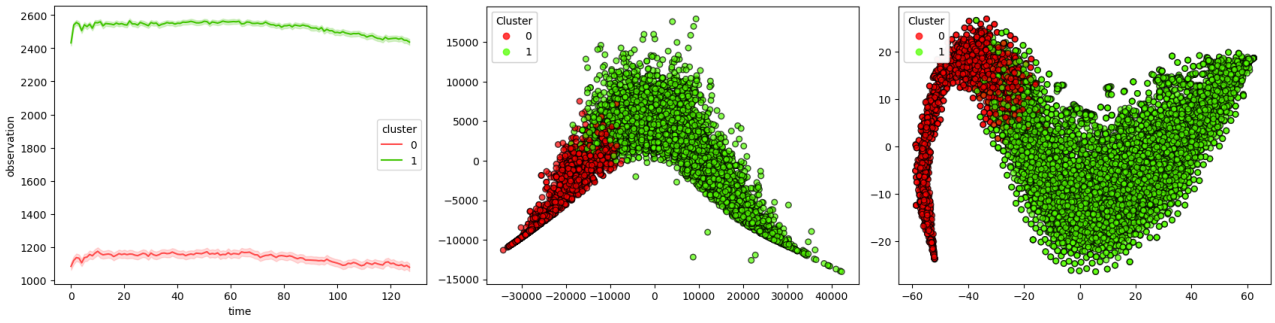


Figure 11: Feature-based clusters with Ward linkage method. From left to right: time series centroids of the three clusters; IsoMap representation; t-SNE representation.

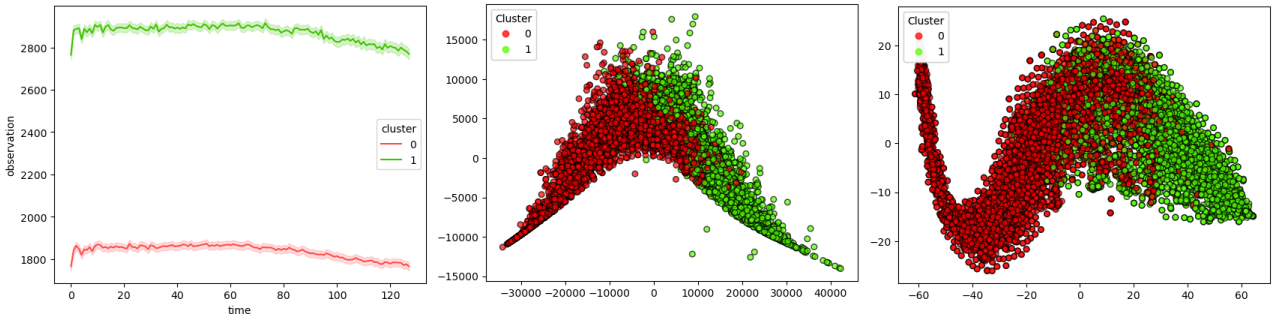


Figure 12: Feature-based clusters with Complete linkage method. From left to right: time series centroids of the three clusters; IsoMap representation; t-SNE representation.

The initial plots in both Figure 11 and 12 bear a striking resemblance to each other, as well as to the plot obtained via K-Means, further reinforcing the hypothesis that genres are segmented into clusters based on varying amplitudes. The visual overlap between the two clusters is more pronounced in this scenario. This is likely due to the fact that we are now considering only two clusters instead of three, which may result in some tracks falling in-between the clusters.

In the following section, we will conduct a comparative analysis of the two clustering methodologies discussed so far. Additionally, we will delve into the specifics of how various genres are distributed across the clusters we have defined.

2.1.3 Comparison

In this section, we undertake a comparative analysis between Shape-based clustering and Features-based clustering, specifically examining how music genres are distributed among various clusters. Within each cluster, certain genres emerge as predominant while others are notably scarce. To illustrate this, we will present a tabulated representation.

	SHAPE BASED			FEATURES BASED			
	Euclidean Distance			**Ward Linkage**		**Complete Linkage**	
Genre	C0	C1	C2	C0	C1	C0	C1
honky-tonk	77.6 (9.21)	16.6 (5.16)	5.8 (0.69)	38.4 (8.36)	61.6 (3.40)	95.6 (7.44)	4.4 (0.62)
mpb	72.0 (8.54)	3.6 (1.12)	24.4 (2.92)	11.4 (2.48)	88.6 (5.75)	73.2 (5.70)	26.8 (3.75)
songwriter	66.6 (7.90)	22.2 (6.90)	11.2 (1.34)	37.8 (8.23)	62.2 (4.04)	87.2 (6.79)	12.8 (1.79)
opera	66.2 (7.86)	25.6 (7.96)	8.2 (0.98)	50.4 (10.98)	49.6 (3.22)	95.0 (7.39)	5.0 (0.70)
sertanejo	64.0 (7.60)	0.2 (0.06)	35.8 (4.28)	1.0 (0.22)	99.0 (6.43)	68.2 (5.31)	31.8 (4.45)
folk	63.2 (7.50)	16.8 (5.22)	20.0 (2.39)	31.2 (6.79)	68.8 (4.47)	82.4 (6.41)	17.6 (2.46)
world-music	50.8 (6.03)	13.2 (4.10)	36.0 (4.31)	16.4 (3.57)	83.6 (5.43)	73.4 (5.71)	26.6 (3.72)
minimal-techno	49.0 (5.82)	12.4 (3.85)	38.6 (4.62)	22.2 (4.83)	77.8 (5.05)	64.6 (5.03)	35.4 (4.95)
goth	48.6 (5.77)	3.4 (1.06)	48.0 (5.74)	8.2 (1.79)	91.8 (5.96)	66.6 (5.18)	33.4 (4.67)
emo	43.4 (5.15)	2.8 (0.87)	53.8 (6.44)	5.8 (1.26)	94.2 (6.11)	44.2 (3.44)	55.8 (7.80)
kids	37.2 (4.41)	1.2 (0.37)	61.6 (7.37)	3.2 (0.70)	96.8 (6.28)	35.6 (2.77)	64.4 (9.01)
synth-pop	36.4 (4.32)	2.6 (0.81)	61.0 (7.30)	5.6 (1.22)	94.4 (6.13)	44.4 (3.46)	55.6 (7.78)
salsa	33.2 (3.94)	1.0 (0.31)	65.8 (7.87)	2.8 (0.61)	97.2 (6.31)	53.4 (4.16)	46.6 (6.52)
progressive-house	26.0 (3.09)	0.2 (0.06)	73.8 (8.83)	1.6 (0.35)	98.4 (6.39)	29.8 (2.32)	70.2 (9.82)
heavy-metal	26.0 (3.09)	0.0	74.0 (8.86)	0.6 (0.13)	99.4 (6.45)	57.4 (4.47)	42.6 (5.96)
new-age	23.2 (2.75)	74.8 (23.24)	2.0 (0.24)	85.8 (18.68)	14.2 (0.92)	98.2 (7.64)	1.8 (0.25)
sleep	21.4 (2.54)	48.4 (15.04)	30.2 (3.61)	56.2 (12.24)	43.8 (2.84)	73.2 (5.70)	26.8 (3.75)
piano	16.8 (1.99)	76.8 (23.87)	6.4 (0.77)	79.8 (17.38)	20.2 (1.31)	94.6 (7.36)	5.4 (0.76)
j-idol	14.2 (1.69)	0.0	85.8 (10.27)	0.2 (0.04)	99.8 (6.48)	32.6 (2.54)	67.4 (9.43)
happy	6.8 (0.81)	0.0	93.2 (11.15)	0.6 (0.13)	99.4 (6.45)	15.4 (1.20)	84.6 (11.83)

Table 5: Genre distribution over clusters. For each genre, the numbers indicate the percentage of tracks belonging to that genre that fall into a specific cluster. Instead, in parenthesis is indicated, for each cluster, the percentage of tracks of a certain genre.

This table may seem a bit intricate at first glance, but it serves as a powerful tool for examining the behavior of specific genres within the clusters we've identified. For instance, we can observe that the majority of `honky-tonk` tracks were grouped in Cluster 0 of the Complete Linkage clustering, whereas they are predominant in Cluster 0 of the Shape-based clustering, comprising almost one-tenth of all tracks. Similarly, `happy` tracks consistently cluster closely together across all approaches, indicating their distinct recognizability.

Moreover, an intriguing observation emerges when we notice that certain clusters tend to group similar genres. For example, in Shape-based Cluster 1, nearly half of the tracks consist of `new-age` and `piano` genres, both characterized by relaxing sounds, which may also justify the presence of another 15% of `sleep` tracks.

While many other insights can be gleaned from this table, they fall beyond the scope of our current study. Our primary aim was to demonstrate that meaningful clusters could be identified from tracks' time series using various techniques.

The diversity among the clusters we discovered makes it challenging to definitively select one clustering strategy as superior to another. However, this in itself is a significant conclusion, as it suggests that Shape-based approaches are not inherently better than Feature-based ones, and vice versa. Of course, our exploration only scratches the surface of the possibilities within this field. For instance, employing Shape-based approaches with DTW distance can be substantially better, as well as extracting different features from the time series could potentially enhance Feature-based performances. The realm of possibilities is large, but the space in this report is very limited. So, we will now proceed to delve into Motifs and Discords analysis.

It's important to highlight that, based on our findings and the optimal outcomes achieved thus far, we will proceed with our future experiments on the Time Series dataset without employing amplitude scaling, diverging from our initial approach (described in Section 1.3).

2.2 Motifs and Discords

In preparation for the upcoming classification task and to offer a glimpse into the subsequent chapter's content, the detection process of relevant patterns and anomalies within the data was carried out directly on the training set derived from the original dataset (we leave the details to section 2.3). In order to identify motifs and discords within the dataset under consideration, we concatenated the time series of all songs within each genre, resulting in 20 time series of length 48,000 units each. However, considering the musical nature of our dataset – consisting of songs (each being 128 units long) –, we employed a window size of 20 to extract the most frequent motifs and discords. As expected, the Matrix Profiles of these long time series resulted unreadable due to the small window size. Therefore, we opted to partition again each long time series into segments, corresponding to the original 128-unit-songs. This partitioning facilitated the identification and visualization of the motifs and discords in the time series, based on their indices. In order to have a reasonable number of data to elaborate on, the maximum number of motifs and discords we decided to extract is 5.

Upon visual and qualitative evaluation of the matrix profiles of each time series by genre, we uncovered intriguing and distinctive motifs and discords within the time series of *minimal-techno*. Below, we present the segments extracted from the time series, showcasing the motifs and discords identified. To provide a comprehensive overview, we also include some correspondent matrix profiles. The consistent patterns of motifs suggest that *minimal-techno* stands out as the most structured and repetitive genre. This is likely attributed to its predominant features, including the prevalent use of characteristic instruments such as synthesizers and drum machines, coupled with a focus on repetitive rhythmic patterns and sparse arrangements.

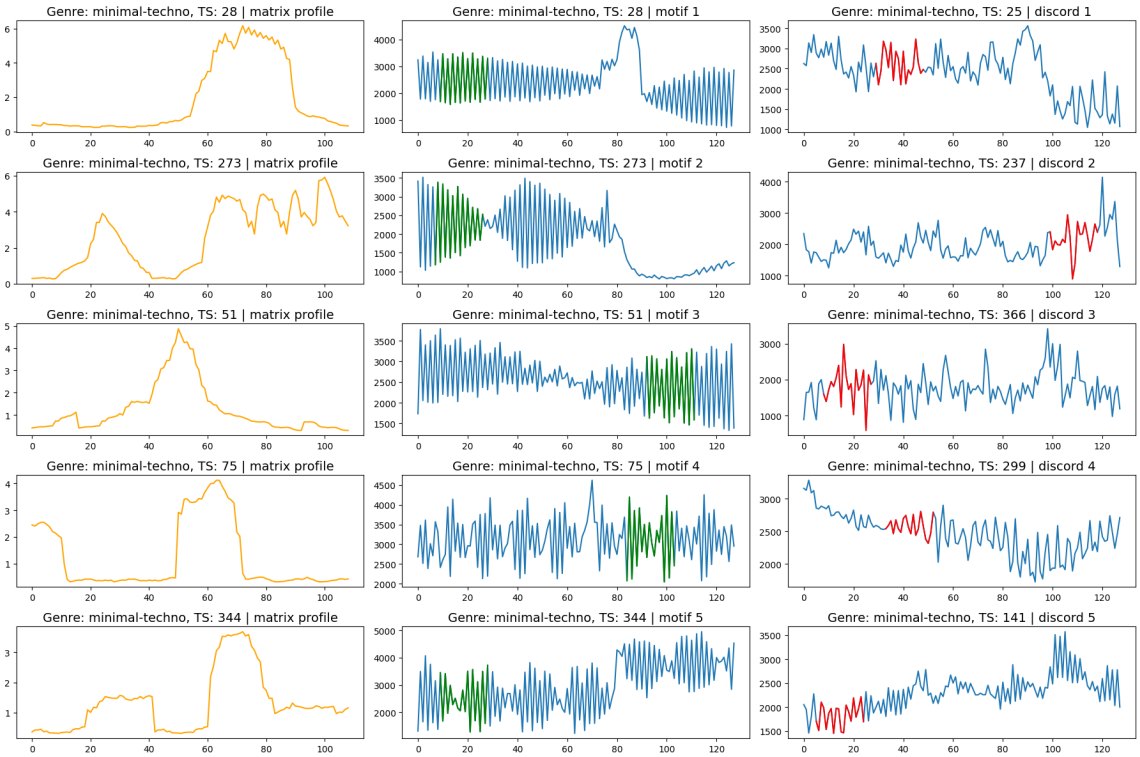


Figure 13: Matrix Profiles, Motifs and Discords within *minimal-techno* time series

2.3 Classification

In this chapter, time series will be used to address a multiclass classification problem: predicting the genre of the songs. For this task, the original dataset – consisting of 10000 time series, balanced across the 20 different genres – was divided into a training set and a test set, comprising respectively 7500 and 2500 records, ensuring that the classes distribution is maintained in the test set. Additionally, for each method that derives a feature dataset from the time series (whether structure-based or distance-based), MinMaxScaling was applied to normalize the feature range.

The first section reports various methods based on the shape of the time series utilizing distance measures for classification. In the second section, a structure-based approach is presented, focusing on classification using features extracted from statistics of the time series structure. Finally, the third section describes the results of deep learning algorithms for classification.

For each model, a hyperparameter tuning phase was conducted either through 5-fold cross-validation or by deriving a validation set from the training set. The performance of each model is reported in terms of accuracy and F1-score. The performance of selected models, will be further investigated to understand the error distribution and the strengths and weaknesses of the classifiers. The aim is to consider also the potential and simplicity of each approach, not only the performances.

2.3.1 Shape-based Classification

K-Nearest Neighbors

As initial approach to classification, K-Nearest Neighbors was employed. For time series data, the choice of distance function is crucial. Both the simple Euclidean distance and the Dynamic Time Warping (DTW) distance were utilized. DTW is expected to be more robust to distortions present in time series data and to better capture the concept of similarity, which is naturally more complex to express with conventional metrics like Euclidean distance. Therefore, the aim of this section is also to compare the performance of these two different metrics, to test whether the theoretical premises are reflected in empirical data.

Before proceeding with testing, it was necessary to choose the parameter k for both models. A grid search was conducted on this parameter, selecting the model with the highest accuracy on the validation set. For the model using DTW distance $k = 31$ was chosen, while for the model using Euclidean distance, $k = 111$ was selected. Below are the validation results for both models.

	Precision	**Recall**	**F1-score**	**Accuracy**
Euclidean	0.26	0.17	0.12	0.17
DTW	0.30	0.27	0.25	0.27

Table 6: Evaluation measures on **validation set** for **KNN**

Top 5 genres	**Worst 5 genres**
new-age	emo
piano	mpb
sleep	goth
happy	world-music
minimal-techno	folk

Table 7: Classes f1-score on validation set for KNN

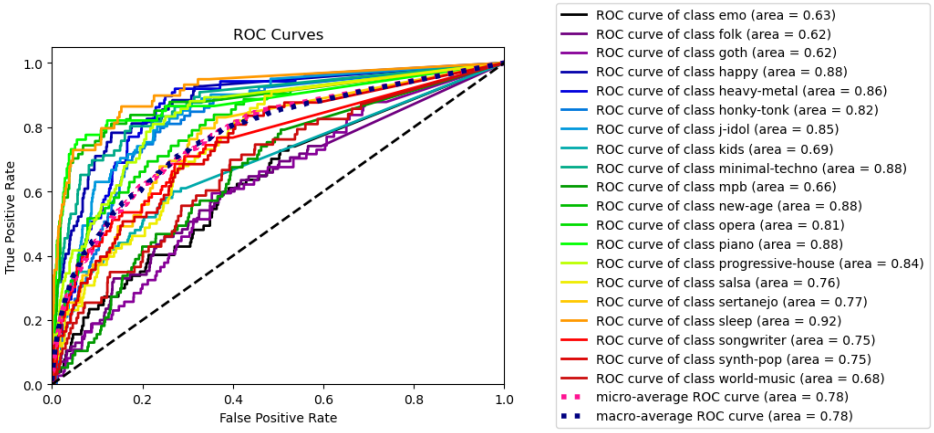


Figure 14: ROC Curves of KNN with DTW distance

As we can see, on the validation set, the model using DTW distance appears to outperform the model using Euclidean distance in terms of accuracy and F1-score.

Delving into the results of the DTW model, we observe varied performance across different genres. Table 7 lists the 5 genres with the highest f1-score and the 5 genres with the lowest. The classifier achieves an F1-score above 0.5 for certain genres, such as *new-age* and *piano*, which is a commendable result considering the 20 classes classification task. Conversely, for other genres like *emo* and *mpb*, the f1-score is below 0.1. The classifier using Euclidean distance struggles with the same genres, even reporting an f1-score of 0 for 7 genres, indicating complete misclassification of these genres.

	Precision	**Recall**	**F1-score**	**Accuracy**
Euclidean	0.38	0.17	0.12	0.17
DTW	0.30	0.28	0.26	0.28

Table 8: Evaluation measures on **test set** for **KNN**

The performances on the test set confirms the hypothesis at the beginning of the section: the Euclidean distance struggles to represent the distances between time series, as evidenced by its poor performance. Probably, the very high precision value (0.38) for the model using Euclidean distance is misleading: several genres have maximum precision, but with a recall below 0.05; this implies that the classifier assigns these genres only very rarely, only in cases where the record clearly belongs to that class.

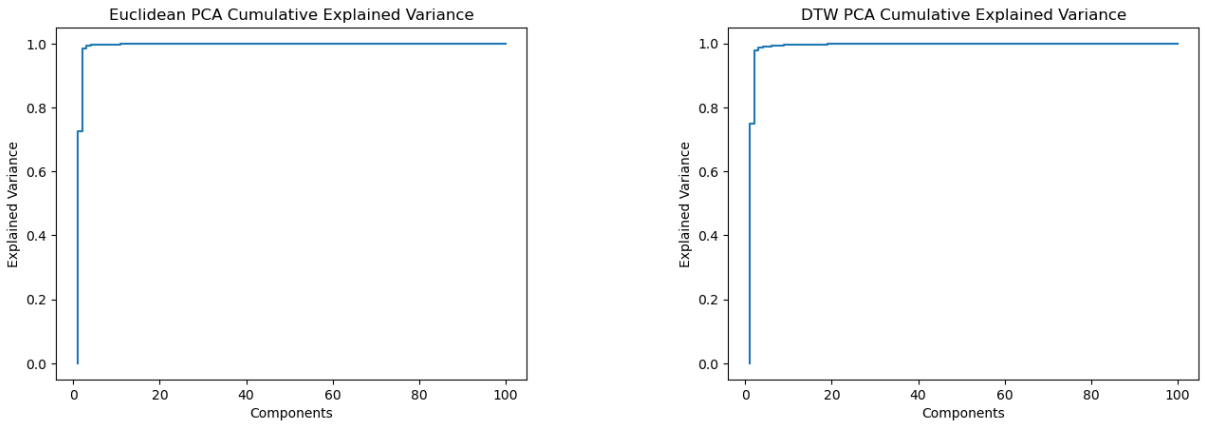
In conclusion, KNN used with DTW distance appears to offer good classification performance; it can be a solid choice given the algorithm's simplicity and the relative ease of hyperparameter tuning, while acknowledging the problems of the instance-based approach. KNN with Euclidean distance shows significantly lower performance but benefits from computational efficiency: computing the DTW between two time series of equal length has quadratic complexity, whereas computing the Euclidean distance has linear complexity. Therefore, DTW could be computationally expensive to apply on datasets with long time series, unless significant approximations are made.

PCA on the distance matrix

In this section, we introduce a straightforward method to extract features for time series classification: we compute the distance matrix between the time series in the dataset, ensuring to only consider distances to the training set time series; subsequently, Principal Component Analysis (PCA) is applied to reduce dimensionality. So, each time series will be represented by a feature vector based on distance, enabling the dataset to be used to train any machine learning model.

In our case, this approach was applied to both the Euclidean and DTW distance matrices. Furthermore, classifiers such as KNN, Decision Tree, and Naive Bayes were employed.

The first step is dimensionality reduction, so to determine the number of components to use, plots were created showing the cumulative variance explained by the PCA components.



(a) Euclidean distance matrix

(b) DTW distance matrix

Figure 15: Cumulative explained variance ratio for the first 100 components

As we can see from figure 15, only a few components are sufficient to explain the variance in the data from the distance matrices. Therefore, it was decided to retain only the first 10 components for both cases. Each time series will thus be represented by just 10 features, which also ensures excellent execution time for classification algorithms.

Subsequently, a hyperparameter tuning phase was conducted for the KNN and Decision Tree algorithms, employing a 5-fold cross-validation on the training set. For the dataset based on the Euclidean distance matrix, the following parameters were selected: KNN with *k=150* and cosine similarity; Decision Tree with *criterion=entropy*, *max_depth=8*, and *min_samples_split=100* (only pre-pruning techniques were adopted). For the dataset derived from the DTW distance: KNN

with *k=150* and cosine similarity; Decision Tree with *criterion=entropy*, *max_depth=8*, *min_samples_leaf=6*, and *min_samples_split=100*. Below are the validation results.

	Precision	**Recall**	**F1-score**	**Accuracy**
KNN	0.27	0.26	0.24	0.26
Decision Tree	0.23	0.24	0.22	0.24
Naive Bayes	0.21	0.24	0.19	0.24

Table 9: Evaluation measures on **validation set** for classifiers with the dataset obtained from **Euclidean distance matrix**

	Precision	**Recall**	**F1-score**	**Accuracy**
KNN	0.29	0.28	0.26	0.28
Decision Tree	0.22	0.25	0.23	0.25
Naive Bayes	0.24	0.25	0.22	0.25

Table 10: Evaluation measures on **validation set** for classifiers with the dataset obtained from **DTW distance matrix**

From the validation results, we first notice that the best-performing classifier among those tested is KNN. Additionally, the results on the dataset based on the DTW distance matrix do not seem to show a significant improvement compared to the straightforward application of KNN (as shown in table 6). However, the results obtained from PCA on the Euclidean distance matrix show a great improvement over the simple application of KNN with Euclidean distance (see table 6). Let's see if these observations are true also on the test set.

	Precision	**Recall**	**F1-score**	**Accuracy**
KNN	0.28	0.27	0.25	0.27
Decision Tree	0.20	0.23	0.20	0.23
Naive Bayes	0.22	0.24	0.19	0.24

Table 11: Evaluation measures on **test set** for classifiers with the dataset obtained from **Euclidean distance matrix**

	Precision	**Recall**	**F1-score**	**Accuracy**
KNN	0.29	0.28	0.26	0.28
Decision Tree	0.22	0.25	0.22	0.25
Naive Bayes	0.24	0.27	0.22	0.27

Table 12: Evaluation measures on **test set** for classifiers with the dataset obtained from **DTW distance matrix**

In conclusion, we can say that this approach doesn't seem to have a significant impact compared to directly using KNN with DTW distance: the performances, as shown in table 12, is virtually identical to that in table 8. However, when calculating the DTW distance isn't feasible due to dataset size, this method significantly boosts performance using the Euclidean distance: the results are very close to KNN with DTW distance but with a much faster execution time. It could, therefore, be a cost-effective approach to time series classification, though these observations should, of course, be confirmed with tests on multiple datasets.

Furthermore, this method, after transforming the distance matrix, allows for the potential use of any classifier. It's worth noting, however, that the entire classification process remains instance-based: for each new test record, the distance from every record in the training set must still be computed, regardless of the classifier used after the PCA transformation.

Shapelets

In this section, we present the results of shapelet-based classification of time series. The strategy involves using an algorithm to extract representative subsequences (shapelets) for distinguishing target classes. Once the shapelets are extracted, each time series in the dataset is represented by its minimum distance to each shapelet.

Several algorithms can extract shapelets from a dataset, in this study, we used the gradient-based approach. The optimization algorithm employed was Adam, with a learning rate of 5e-4, on batches of 256 records, for 800 epochs. The training set accuracy at the end of training was 0.30. Subsequently, the extracted shapelets were used to compute the distance dataset for each time series. From this distance dataset, KNN and Decision Tree were used for classification.

Subsequently, a hyperparameter tuning phase was conducted using a 5-fold cross-validation, leading to the selection of two models: KNN with Euclidean distance and $k = 30$; Decision Tree with $criterion = entropy$, $min\_samples\_leaf = 40$, and $min\_samples\_split = 128$.

Additionally, given the high dimensionality of the dataset (1000 extracted shapelets), PCA was applied, retaining only the first 10 components. On this reduced dataset, the models selected were: KNN with Manhattan distance and $k = 60$; Decision Tree with $criterion = entropy$, $min\_samples\_leaf = 18$, and $min\_samples\_split = 32$. Below are the validation results.

	Precision	Recall	F1-score	Accuracy
KNN	0.27	0.26	0.24	0.26
Decision Tree	0.24	0.25	0.24	0.25
KNN-PCA	0.31	0.29	0.28	0.29
Decision Tree-PCA	0.25	0.25	0.25	0.25

Table 13: Evaluation measures on **validation set** for **shapelet-based** classification

The best result is from KNN on the reduced-dimensionality dataset, which closely approaches the accuracy achieved on the training set during training, suggesting that the shapelets are not "overfitted". The Decision Tree algorithm yields lower results in both cases compared to KNN, however, this model allows us to extract features importance to analyze which shapelets were most significant for classification.

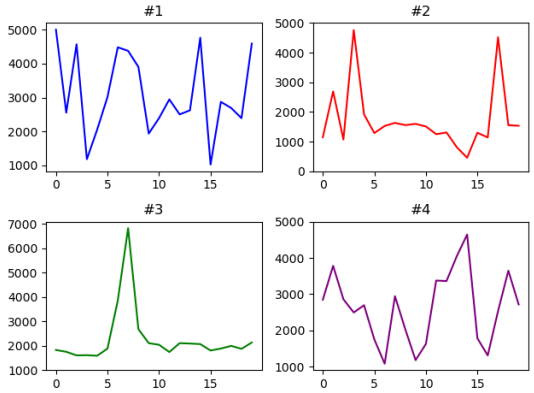


Figure 16: Top 4 shapelets based on feature importance

	Precision	Recall	F1-score	Accuracy
KNN	0.28	0.26	0.25	0.26
Decision Tree	0.24	0.25	0.23	0.25
KNN-PCA	0.30	0.29	0.28	0.29
Decision Tree-PCA	0.25	0.25	0.24	0.25

Table 14: Evaluation measures on **test set** for **shapelet-based** classification

Analyzing the results on the test set (tab. 14), we can see that the best-performing model (KNN-PCA) has slightly better results (+0.01 acc. and +0.02 f1) compared to the direct use of KNN with DTW (tab. 8). Moreover, this method allows us to use any classifier after transforming the dataset, so it's possible that using more refined methods could yield even better results.

Motifs and Discords Distances

In this section, the motifs and discords analyzed in section 2.2 will serve as significant sequences to represent the time series, similar to shapelets. To reiterate the method used to detect these patterns, initially, the time series in the training set were grouped and concatenated by genre, resulting in 20 genre-identified extended time series. Subsequently, matrix-profiles were computed for each of them to extract the patterns. This time, our goal was to extract 20 motifs and 20 discords.

Next, the transformation of the original dataset was performed: for each time series, the minimum distance to each motif and discord for each genre was computed, similar to treating them as shapelets. Then, for instance, from the 20 distances to the motifs of a genre, only the 5 smallest were retained and their average was used as feature for the finale dataset. This final step was carried out to keep the dimensionality of the final dataset low.

So, in the end, three datasets were derived: one with distances to the motifs (shape 10000x20), one with distances to the discords (shape 10000x20), and finally a combined dataset (shape 10000x40).

For the experiments, only KNN was used as the classification algorithm. Below are the validation results.

	Precision	Recall	F1-score	Accuracy
Motifs	0.31	0.29	0.28	0.29
Discords	0.27	0.26	0.24	0.26
Total	0.30	0.28	0.27	0.28

Table 15: Evaluation measures on **validation set** for **motifs and discords** classification

The validation results are promising, with the dataset based on distances to the motifs displaying superior performance, suggesting a better representation of this data for genres classification.

	Precision	**Recall**	**F1-score**	**Accuracy**
Motifs	0.32	0.30	0.28	0.30
Discords	0.28	0.26	0.25	0.26
Total	0.31	0.29	0.28	0.29

Table 16: Evaluation measures on **test set** for **motifs and discords** classification

The test set results confirm that on this dataset, this strategy appears to be more effective with motifs. In conclusion, this method, which can be viewed as a semi-supervised version of shapelets, has yielded good results in these experiments, outperforming the other distance-based techniques employed.

2.3.2 Structural-based Classification

In this brief section, a trivial structure-based classification technique is presented: 18 statistics are extracted from the raw time series, such as mean, standard deviation, maximum, and minimum. These features are then used to represent the time series.

KNN and Decision Tree algorithms were employed for classification. After tuning the hyperparameters using a 5-fold cross-validation, the chosen models were: KNN with $k = 100$ and cosine similarity; Decision Tree with $criterion = entropy$, $max\_depth = 10$, $min\_samples\_leaf = 30$, and $min\_samples\_split = 75$. Below are the validation results.

	Precision	**Recall**	**F1-score**	**Accuracy**
KNN	0.33	0.34	0.32	0.34
Decision Tree	0.28	0.29	0.28	0.29

Table 17: Evaluation measures on **validation set** for **structure based** classification

Top 5 genres	**Worst 5 genres**
sleep	goth
piano	folk
minimal-techno	mpb
new-age	synth-pop
happy	emo

Table 18: Classes f1-score on validation set for KNN structure based classification

Additionally, feature importance analysis was conducted for the Decision Tree classifier to understand which time series statistics were most significant for genre classification. The top 5 features by importance were: *Mean Absolute Deviation*, *90th percentile*, *maximum*, *skewness*, and *25th percentile*.

	Precision	**Recall**	**F1-score**	**Accuracy**
KNN	0.34	0.35	0.33	0.35
Decision Tree	0.28	0.29	0.27	0.29

Table 19: Evaluation measures on **test set** for **structure based** classification

As we can see from the results in table 19, this trivial structure based approach outperforms all the distance based methods presented in the previous section. This suggests that, for genre classification, in this dataset, the most valuable information lies in the inherent structure of the time series. Moreover, the ease of computing these statistics allowed us to directly extract them from the raw time series; in contrast, the distance-based approaches required an approximation technique (PAA), which might have led to an information loss.

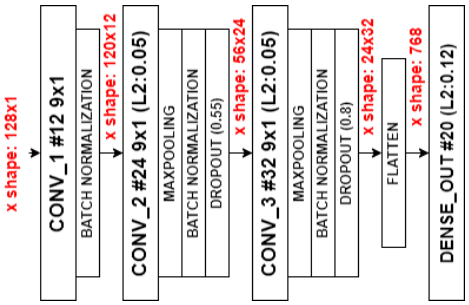
In conclusion, examining the f1-score for individual genres shown in table 18, we observe a recurrent pattern both in structure based and distance based approaches: some genres are easier to classify than others. For instance, genres like *sleep*, *piano*, and *new-age* appear to be well-identified by the classifiers. In contrast, genres such as *goth*, *mpb*, and *folk* prove to be more challenging to classify.

2.3.3 Deep Learning Classification

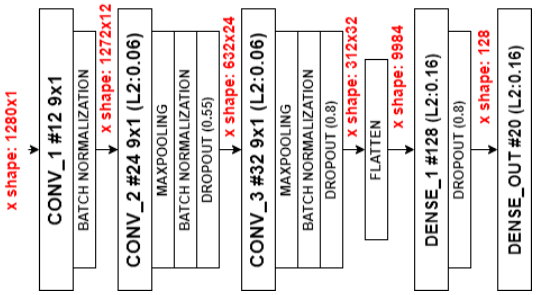
In this section, a deep learning method for time series classification will be presented. We chose to design a simple convolutional neural network, which will be described in detail later. We decided to evaluate the ROCKET and MINIROCKET classification

algorithms on the test set as well, without performing a hyperparameter tuning phase and without delving into the models. They will be considered only as a strong baseline to understand if our model, at least for this dataset, is competitive with state-of-the-art methods. Initially, the results of the experiments using the time series approximated with PAA will be presented. Subsequently, the same methods will be applied directly to the original time series.

Regarding preprocessing, initial experiments were conducted to understand whether to apply amplitude scaling to the time series or leave them raw. In the case of ROCKET, minimal differences were observed between the two approaches. However, with our convolutional neural network, better results were obtained using the time series without amplitude scaling. Therefore, we opted for the latter option, thus maintaining consistency with the rest of the chapter.



(a) *Model A*: approximated time series

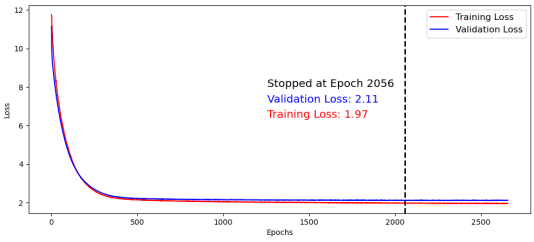


(b) *Model B*: raw time series

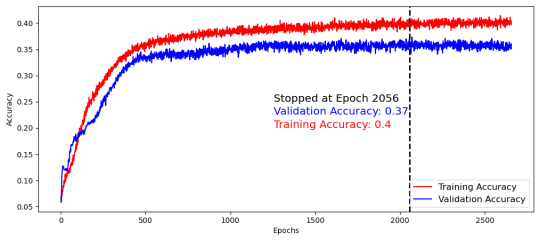
Figure 17: Structure of the two neural networks

To design our convolutional neural network, we employed three convolutional layers, with the second and third followed by max-pooling layers, progressively increasing the number of filters from one layer to the next. The concept is to extract few simple patterns in the first layer, then refine them and arrive at a greater number of high-level patterns in the second and third convolutional layers. To prevent overfitting, we applied *L2 regularization* and *dropout*, which were implemented on all layers except the first convolutional layer. This decision was made to predominantly regulate the layers with more parameters, which are responsible for the model's complexity. Additionally, batch normalization was applied at the output of the convolutional layers. *ReLU* activation function was used for all hidden layers, while *Softmax* was applied to the output layer.

This basic architecture, designed after several experiments on the approximated time series, was also adapted for the original time series by adding a dense layer before the output layer, because after some attempts with the basic structure, we noticed unsatisfactory performance. Our hypothesis was that the model was too simple to capture the complexity of longer time series. Therefore, the model for approximated time series is very simple (*Model A*), with just over 25000 parameters; conversely, mainly due to the additional dense layer, the model for raw time series (*Model B*) has more than 1229000 parameters.



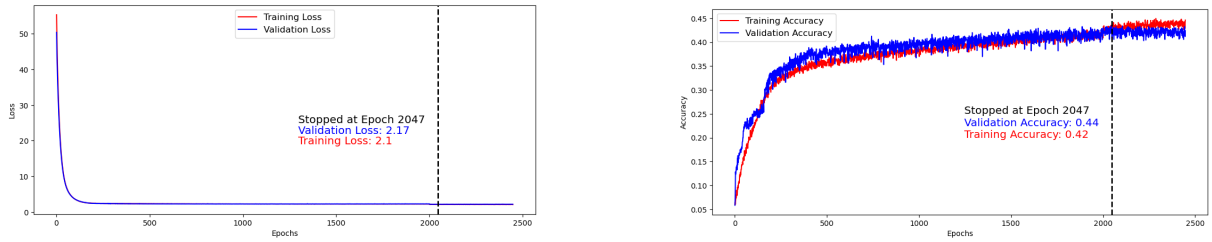
(a) Loss



(b) Accuracy

Figure 18: Plot of the loss and accuracy per epoch for Model A

Regarding the training of *Model A*, after various experiments, we opted to use the *ADAM* optimization algorithm with a learning rate of 0.00005 for 3000 epochs, and the *Categorical Cross Entropy* loss function (same for *Model B*). Additionally, we employed an early stopping criterion based on minimal accuracy gain on the validation set, with a patience of 600 epochs. The choice of such a high patience is due to the heavy regularization applied to our model, which prevents a fast overfitting. Hence, we allowed for a high patience to enable the model to capture any potential accuracy gains during validation. Convergence, as depicted in the plot 18, was achieved by epoch 2056. We observe a slight difference (0.003) between the training and validation set accuracies, which is quite natural; indeed, in that epoch the trajectories of both curves appear parallel, indicating that the model is not overfitted, and that it is a good stop.



(a) Loss

(b) Accuracy

Figure 19: Plot of the loss and accuracy per epoch for Model B

Regarding the training of *Model B*, we again utilized the *ADAM* optimizer, but this time for 2000 epochs with a learning rate of 0.0001, followed by an additional 1000 epochs with a reduced learning rate of 0.00005. This dynamic reduction in learning rate was applied to enable more precise optimization in the final phase. Similarly, we employed the same early stopping criterion, with a patience of 500 epochs. Convergence, in this case, was reached by epoch 2047. As observed from the plot, this configuration led to a stop at a phase of perfect performance parity between the training and validation sets, while in subsequent epochs the model began to exhibit signs of overfitting.

	Precision	**Recall**	**F1-score**	**Accuracy**
MODEL A (PAA)	**0.35**	**0.36**	0.33	**0.36**
ROCKET (PAA)	0.21	0.21	0.21	0.21
MINIROCKET (PAA)	0.35	0.35	**0.34**	0.35
MODEL B (raw)	0.41	0.43	0.40	0.43
ROCKET (raw)	0.38	0.40	0.39	0.40
MINIROCKET (raw)	**0.46**	**0.48**	**0.46**	**0.48**

Table 20: Evaluation measures on **test set** for **Deep Learning** classification

From the results in table 20, we observe that concerning the models trained on the approximated time series, our model (*Model A*) outperforms others in terms of accuracy, yielding very similar results to MINIROCKET and significantly surpassing ROCKET. Hence, we can be satisfied with *Model A*, which, on this data, proves that it is comparable to state-of-the-art classifiers.

However, for the models trained on the raw time series, we observe that MINIROCKET outperforms the other two classifiers. Nevertheless, our *Model B* still manages to achieve better performance than ROCKET (+0.03 accuracy). These results might be attributed to the fact that *Model B* was designed as an adaptation, aiming to maintain the structure of *Model A*. It's plausible that the convolutional part of the network is too simple to process time series with 1280 timesteps, potentially requiring at least another convolutional layer and/or a greater number of filters for data of this size.

2.4 Conclusions

In this section, we conducted a series of experiments on the Time Series dataset. Initially, we employed both Shape-based and Structural-based clustering algorithms, illustrating how different methodologies delineated clusters representing distinct genres within the tracks. We further provided a comparative analysis of these clusters. Following this, we delved into exploring the significance of Motifs and Discords within our sequences, particularly within a specific music genre. The motifs and discords identified were subsequently integrated into our classification framework, alongside other strategies, focusing on both the shape and structure of the sequences. Overall, we are pleased with the results obtained, which demonstrate the efficacy of classifying and clustering track data represented as time series.

3 Imbalanced Learning

In this chapter, we tackled the imbalanced learning case of classifying two classes with uneven distributions. Specifically, we focused on distinguishing between explicit tracks (the positive class), which make up 8% of the data, and non-explicit tracks (the negative class), which account for the remaining 92%. Initially, we attempted a common classification approach, ignoring the class imbalance. Subsequently, we employed an undersampling method (*Edited Nearest Neighbors*), an oversampling approach (*ADASYN*), and finally the combination of both. We will display the results at the end of the chapter.

Additionally, in this scenario, it is important to pay attention to evaluation metrics for the task. Accuracy is certainly not very meaningful, so we will analyze the performance of various methods using: precision and recall of the positive class, macro average f1-score of the classes, and AUC-ROC Curves; thus simulating a situation where the focus is on identifying the positive class, rather than overall accuracy.

The classification algorithm used for each experiment is Decision Tree, utilizing pre-pruning techniques to avoid overfitting. The hyperparameters tuning phase for each classifier was conducted using a randomized search 5-fold cross-validation, using the macro average f1-score as the evaluation metric for selecting the best model. This phase will not be extensively discussed in the chapter; instead, the focus will be on comparing the different methods for handling the imbalanced dataset.

All the remaining features in the *Tracks* dataset were used after the data preparation phase, including the four nominal features (`key`, `mode`, `time_signature`, `genre`). This decision was made for two reasons: genre intuitively carries important information for identifying explicit tracks, and by using the Decision Tree algorithm, even if we have less useful features, the algorithm should naturally ignore them for splits. The training set consists of 57600 instances of the negative class and 5436 instances of the positive class, while the test set consists of 19201 and 1812 instances respectively.

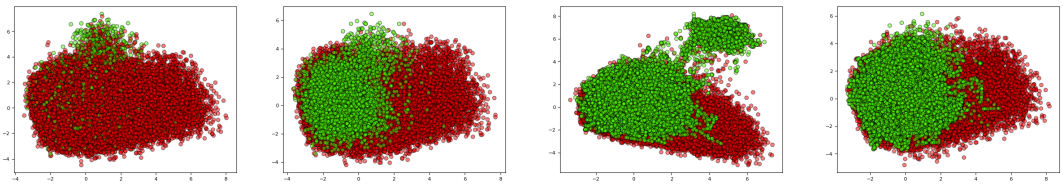
-	**None**	**ENN**	**ADASYN**	**Both**
Negatives	57600	37991	57600	37991
Positives	5436	5436	58221	38039

Table 21: Training set composition for each experiment.

Edited Nearest Neighbors (ENN) is an undersampling method based on instance neighborhood: for each observation, if the output class of the neighborhood vote differs from that of the target point (i.e., if KNN is misclassifying), then instances of the majority class will be removed from the dataset. Choosing the hyperparameter k (number of neighbors to consider) is crucial for this algorithm; indeed, a low k would define a more conservative approach, while increasing k would result in more instances of the majority class being eliminated. In our case, the first experiment was with $k = 3$, which led to a training set with 49129 negative instances (-8471) and 5436 positive instances. Subsequently, we tried increasing the k parameter, making the model less conservative; based on the validation results and visual analysis of the instance distribution in two dimensions (using PCA), we found a good compromise in choosing $k = 10$, which defines a training set with 37991 negative instances (-19609) and 5436 positive instances.

Adaptive Synthetic (ADASYN) is an oversampling method that builds upon the SMOTE algorithm, which adds instances to the minority class by interpolating between the target point and one of its neighbors. Specifically, with ADASYN, the number of points generated from an instance is based on the ratio of the majority class in the neighborhood of that point. Therefore, the number of synthetic instances generated in an area is proportional to its density. The utilization of ADASYN resulted in a training set comprising 57600 instances of the negative class and 58221 instances of the positive class. One potential drawback of ADASYN is the generation of entirely new instances, which could be undesirable in certain real-world scenarios. However, in our case, the task is not as sensitive, so we did not consider this a concern.

In the final experiment, we attempted to combine the two previously discussed methods. The concept was to first use ENN to eliminate instances of the majority class that overlap with instances of the minority class in the space; subsequently, we employed ADASYN to balance the number of instances between the two classes. The goal was to create a training set where not only the two classes are balanced (primarily thanks to ADASYN), but where they are also more easily separable in the space (thanks to ENN). The resulting dataset comprises 37991 negative instances and 38039 positive instances.



(a) Base Dataset (b) ENN (c) ADASYN (d) ENN and ADASYN

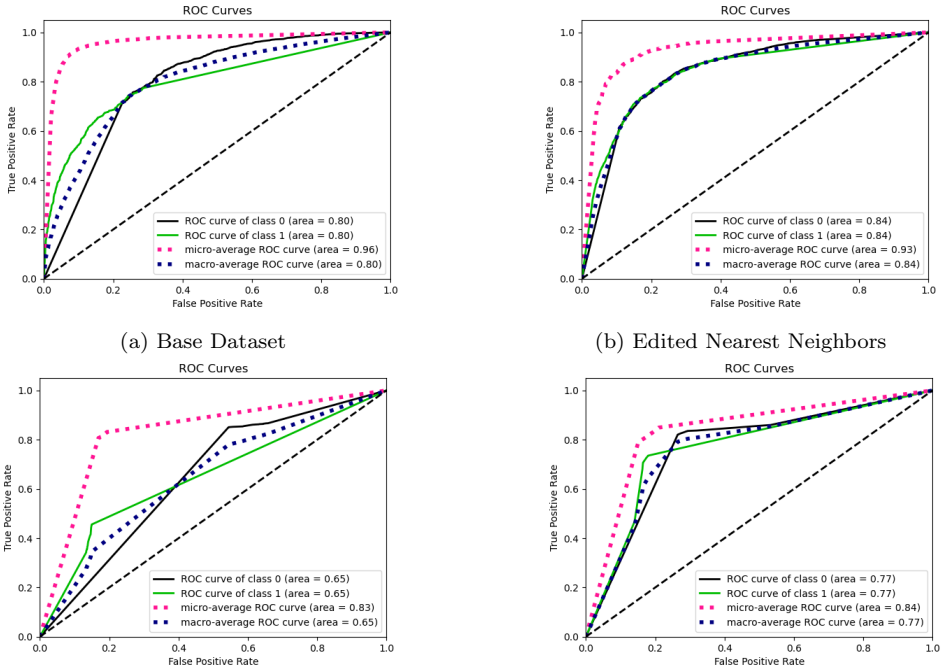
Figure 20: The distribution of instances in the training set in a two dimensions space obtained with PCA; greens are positives, reds are negatives.

In Figure 20, we observe the different distribution of instances in the space for the various techniques. In the base dataset, positive instances largely overlap with negative ones, and due to their greater number, the latter "hide" the positive points

almost entirely; this dominant overlap of the negative class could render the classifier unable to predict the positive class. After using ENN, however, we observe that all negative instances overlapped with positive ones have indeed been eliminated: the area at the center-left of the plot is now predominantly composed of positive class instances; this may assist the classifier in choosing a simpler and more generalizable decision boundary for the two classes. Following the use of ADASYN, we observe that numerous positive class points have been generated, mainly in the center-left area of the plot; however, it should be noted that in this case, negative instances have not been removed, so they are still widely present in that area but "covered" by synthetic records. Finally, after the combined use of both techniques, again in the center-left area of the plot, we have both the removal of overlapping negative instances and a reinforcement of synthetic positive records.

-	None	ENN	ADASYN	Both
f1-score macro avg	0.67	0.69	0.60	0.65
f1-score (class 1)	0.39	0.44	0.30	0.41
Precision (class 1)	0.54	0.37	0.22	0.29
Recall (class 1)	0.30	0.56	0.46	0.72

Table 22: Evaluation on the test set for each experiment.



(a) Base Dataset

(b) Edited Nearest Neighbors

(c) ADASYN

(d) ENN and ADASYN

Figure 21: Roc curves plots for each experiment.

From the results presented in table 22 and figure 26, we observe that the experiment with the base dataset still yielded decent results, despite the rather low recall of the positive class, which could be problematic in certain real-world scenarios.

The experiment with ENN succeeded in improving the results, with an increase of +0.02 in macro average f1-score, +0.05 in f1-score of the positive class, and a higher Area Under the Curve for both classes. Additionally, with ENN, we notice a much higher recall of the positive class compared to precision, indicating an opposite imbalance with respect to the base dataset experiment.

However, the use of ADASYN worsened the results in all aspects, suggesting that for this case, the algorithm may not be suitable, or in general, direct usage of an oversampling method may not be suitable.

Finally, the combined use of both techniques led to results comparable to the base case, but it is interesting to note the precision and recall of the positive class: the recall is quite high (0.72), with low precision (0.29). This could be a more or less desired effect depending on real-world scenarios.

In conclusion, only the Edited Nearest Neighbors undersampling approach led to objective improvements for the imbalanced classification task.

4 Advanced Classification

In this section, we explored a range of advanced classification methods to categorize the feature `genre`. With a dataset boasting 114 unique genres, this classification task presented significant complexity. Despite this, some classifiers exhibited good performance, while others struggled. We'll now delve into each classifier's performance and characteristics in detail.

4.1 Logistic Regression

Logistic Regression isn't well-suited for our objectives due to its linear nature, which attempts to model the relationship between input features and the log-odds of a binary outcome using a linear function. Given the evident non-linearity of our task, it's understandable why it might perform poorly.

However, despite these limitations, we decided to conduct some tests. We employed the one-vs-rest scheme, training individual binary classifiers for each class, considering one class as positive and the rest as negative.

The results were far from promising. Across the board, precision, recall, and f1-score were strikingly low, with most classes registering scores of 0. Only a few genres occasionally exhibited slightly different, yet consistently dismal, performance metrics. The table below succinctly summarizes the overall outcomes across the 114 genres.

	Precision	**Recall**	**F1-score**
macro average	0	0.02	0
weighted average	0	0.03	0.01
accuracy		0.03	

Table 23: Logistic Regression performance.

4.2 Support Vector Machines

After visually examining a PCA-generated visualization to reduce the data's dimensionality, it became apparent that our dataset is non-linearly separable. This idea was reinforced by the dataset's nature and previous analyses. Given that we have 114 genres to classify, the dataset is likely quite complex, with potentially overlapping characteristics between genres. Yet, despite this understanding, we attempted to employ a Soft Linear SVM for the classification task, taking into account the regularization parameter C. As anticipated, the results were unsatisfactory: many genres could not be classified correctly. Among the values of C that we tried (0.1, 1.0, 10, 100), the best one proved to be C=1.0 with an overall accuracy of 0.22, while the worst one was C=100, with an overall accuracy of 0.06. The Non-Linear SVM, on the other hand, yielded significantly improved performance in every couple of hyper-parameters. We tried again the same C parameters (0.1, 1.0, 10, 100) and different kernel functions (RBF, Linear, Polynomial) to find the best for our model. The best configuration in the end proved to be `C=10, kernel=rbf`, which granted an overall accuracy of 0.31.

	Precision	**Recall**	**F1-score**
macro average	0.27	0.28	0.27
weighted average	0.30	0.31	0.30
accuracy		0.31	

Table 24: SVM tuned to `C=10, kernel=rbf`

The superiority of the Non-Linear SVM with an RBF kernel over the Soft Linear SVM in classifying songs by genre likely stems from the inherent complexity and non-linearity of the relationship between the features (attributes of the songs) and their respective genres. The RBF kernel allows the SVM to model intricate decision boundaries, capturing the nuances and complexities of this relationship more effectively compared to a linear kernel and making it well-suited for high-dimensional data.

We were surprised that some genres, such as `comedy`, `grindcore`, `honky-tonk`, `sleep` and `study`, exhibited F1-scores higher than 0.65. However, some genres were consistently misclassified, yielding F1-scores equal to 0, namely `singer-songwriter`, `punk` and `end`. To provide a comprehensive understanding of the model's performance, in the conclusions we'll display the ROC Curves for the best-performing genres, focusing on those with F1 scores exceeding 0.65 and equal to 0.

4.3 Neural Network

In this section, we introduce our neural network, designed from scratch to tackle the genre classification task. The basic architecture is a Multilayer Perceptron (MLP) featuring four hidden layers. These layers include 128, 256, 512, and 256 neurons, respectively, all with ReLU activation function. The output layer comprises 114 neurons with softmax activation. To mitigate overfitting, dropout was implemented after each hidden layer, and L2 regularization was applied to all layers, including the output layer. Furthermore, batch normalization was introduced after each hidden layer to stabilize their outputs.

Subsequently, after several validation experiments, we implemented residual connections between the layers, observing that they allowed for faster training with more stable results (reduced oscillations in accuracy and loss between epochs) and slightly higher performance. Three residual connections were used, which combine in two different ways: concatenation, by concatenating the residual vector to the input of a more advanced layer; and summation, by adding the residual vector to the input vector of an advanced layer. The entire structure is depicted in the diagram below.

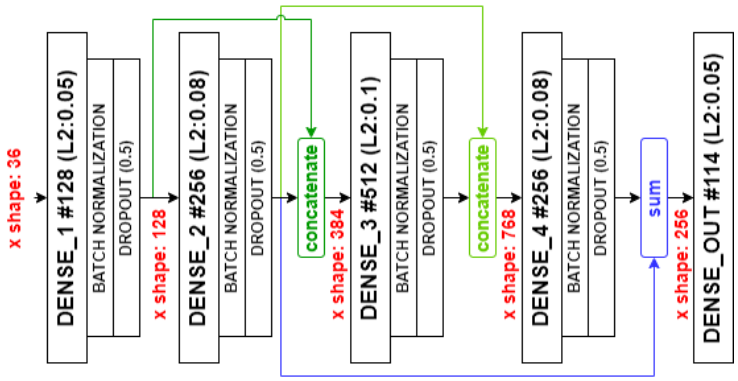
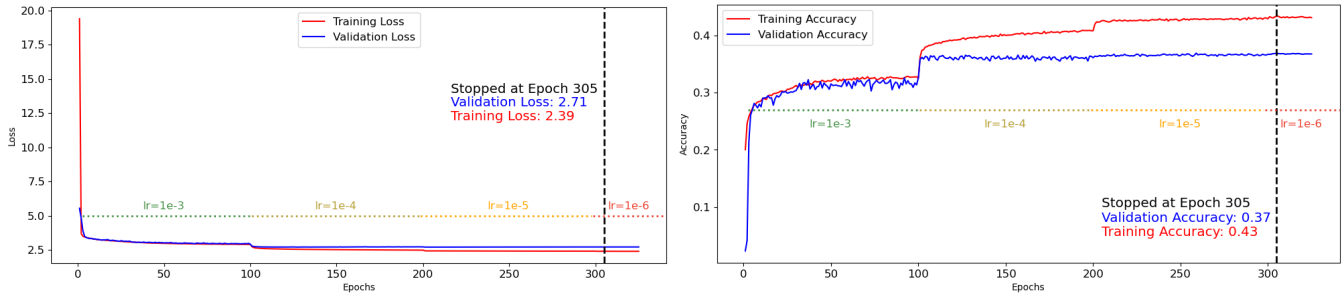


Figure 22: Structure of our neural network.

For the training phase, we employed the Categorical Cross Entropy loss function and utilized the ADAM optimization algorithm. To achieve both fast and precise training, we initially set a relatively high learning rate (1e-3), then decayed it every 100 epochs, multiplying it by 1/10. This approach allowed us to prevent stagnation during the training when the rate was too high, and ensuring more precise optimization, we can observe the effects in plot 23. Additionally, early stopping was implemented based on the accuracy in the validation set, with a patience of 25 epochs.



(a) Loss

(b) Accuracy

Figure 23: Plot of the loss and accuracy per epoch for our MLP, "lr" stands for learning rate.

Training was stopped at epoch 305 due to a stall of the accuracy on the validation set, even after the learning rate had been adjusted. Overall, we are satisfied with the training of the network, as it shows promising performance for the task. Despite naturally higher accuracy on the training set (+0.06), it appears to generalize well on the validation set; below are the results of the model on the test set.

	Precision	**Recall**	**F1-score**
macro average	0.30	0.31	0.30
weighted average	0.34	0.36	0.34
accuracy		0.36	

Table 25: MLP performances on test set.

4.4 Random Forests

Random Forests represent a state of the art in classification techniques. Embracing them entails acknowledging their potential for delivering highly accurate results, albeit with the challenge of fine-tuning numerous hyperparameters. To initiate our exploration, given its ensemble nature, we opted for 100 estimators—a parameter we'll delve into later regarding its impact on model performance. To refine the remaining hyperparameters, we employed a Randomized Search across a curated selection. This approach yielded a satisfactory configuration, culminating in the outcomes showcased in Table 26.

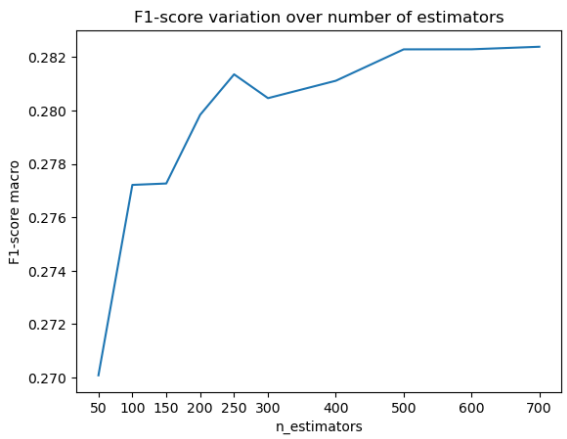
	Precision	Recall	F1-score
macro average	0.32	0.30	0.28
weighted average	0.34	0.35	0.32
accuracy		0.35	

Table 26: Random Forest performance obtained with hyperparameter configuration: `'min_weight_fraction_leaf': 0.0, 'min_samples_split': 20, 'min_samples_leaf': 10, 'max_features': 'log2', 'max_depth': 20, 'criterion': 'gini', 'bootstrap': True`. In addition to this configuration, which was determined through a randomized search, we also explored other potential setups through manual testing. However, none of these alternatives yielded significantly better results than this one.

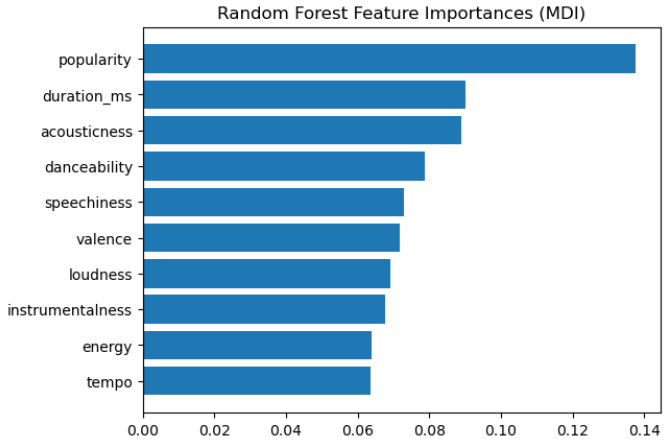
Before delving into the specifics of the results attained with our current configuration, we sought to evaluate the influence of the number of estimators on overall performance. Not surprisingly, the trend indicates that as the number of estimators increases, prediction accuracy likewise improves. However, the benefits derived from a substantial escalation in the number of estimators are marginal and diminish as the count grows. Consequently, we opted to cap our selection at 100 estimators, a quantity that has already demonstrated adequacy in achieving satisfactory results. Image 24a shows the tendency we discussed here.

Our model's performance is generally satisfactory, especially given the dataset's extensive number of classes. However, it's noteworthy that the model's performance varies significantly across different classes. For instance, genres such as `sleep` exhibit a high F1-score of 0.77, indicating excellent recognition by the model. Conversely, genres like `swedish` or `punk` are consistently misclassified, yielding an F1-score of 0. To provide a comprehensive understanding of the model's performance, we will present an overview of genres with particularly high or low performance within the context of the ROC Curve. This visualization will selectively display classes with exceptional F1-scores. In the conclusions, we will showcase the ROC Curve for the Random Forest classifier alongside those of other classifiers for comparison.

An essential analysis for Random Forests involves exploring feature relevance, which entails discerning which features hold greater significance in the decision-making process of the estimators. We measured feature importance using the metric called Mean Decrease in Impurity (MDI). Figure 24b, illustrates the top ten most influential features.



(a) Effect of the number of estimators on the performance of the model.



(b) Top ten important features in the decision process of the estimators.

Figure 24

It's interesting to observe that the primary distinguishing factor for identifying genres is their `popularity`: the estimators prioritize the popularity of a song over its intrinsic characteristics. Furthermore, it's noteworthy that none of the top ten features include `confidence` features, indicating their minimal impact on classification.

4.5 Gradient Boosting Machines

To explore Gradient Boosting Machines, we opted for LightGBM due to its fast training speed, excellent performance, and compatibility with large and complex datasets like ours. These qualities enabled us to experiment with different parameters and find the best configuration for our model.

Initially, we trained the model with its default hyperparameters, but the results were unsatisfactory, displaying extremely low accuracy. To address this issue, we adjusted the learning rate, testing values of 0.0001, 0.001, and 0.01. The best learning

rate was 0.01, which resulted in an accuracy of 0.35.

Next, we experimented with the number of estimators, keeping it within the range of 50 to 700, which led to even better results. We observed a continuous increase in performance, more so than with the Random Forest model. We provide a graph illustrating this significant improvement.

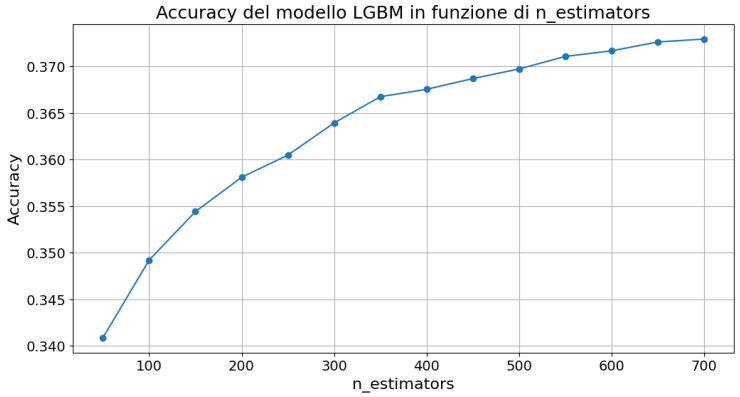


Figure 25: LGBM classifier performance with `n_estimators` values

We then employed Randomized Search to fine-tune the number of leaves and the learning rate. We set the number of leaves between 31 and 41 and the learning rate between 0.01 and 0.1. This further improved our results, achieving an outstanding accuracy of 0.38.

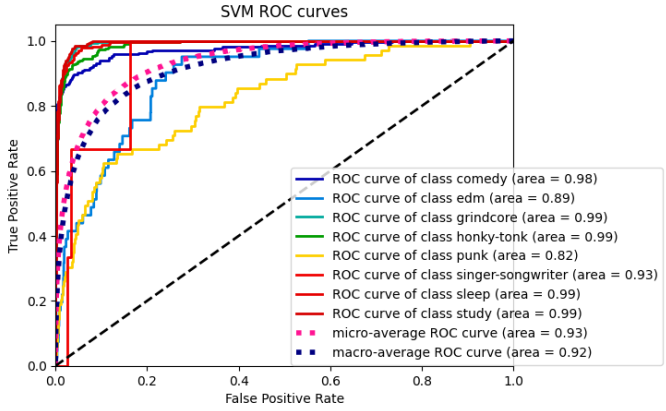
	Precision	**Recall**	**F1-score**
macro average	0.34	0.33	0.33
weighted average	0.37	0.38	0.37
accuracy		0.38	

Table 27: LightGBM with hyperparameter configuration: `num_leaves`: 37, `learning_rate`: 0.02, `n_estimators`: 700

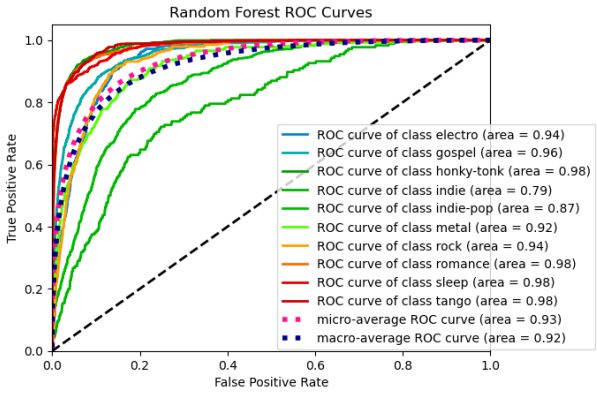
As with previous models, some genres were recognized better than others. However, this time, we raised the bar higher, as the best genres exhibited F1-scores above 0.70. The exceptions were the singer-songwriter and punk genres, which still had F1-scores of 0. The ROC curve in the conclusion section will display the best and worst-performing genres.

4.6 Conclusions

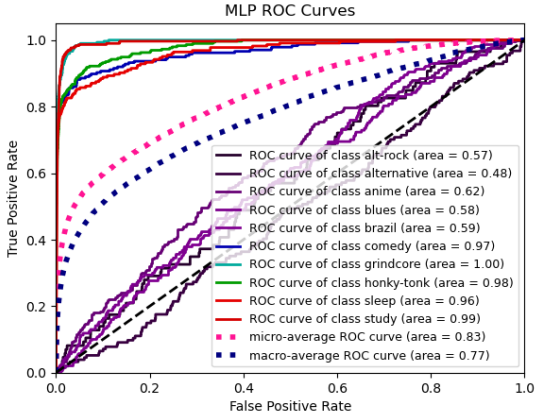
We now present the ROC curves corresponding to each of the models constructed in the preceding sections and a table summarizing the performance results. To avoid making the plots too dense, we only display the best and worst classified genres.



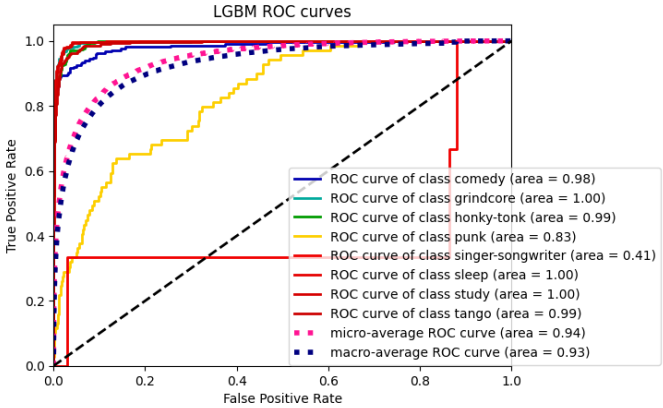
(a) SVM



(b) Random Forest



(c) MLP



(d) GBM

Figure 26: Roc curves plots for each Classifier.

	SVM	**MLP**	**RF**	**GBM**
Accuracy	0.31	0.36	0.35	0.38
F1 (weighted)	0.30	0.34	0.32	0.37

Table 28: Models performance summary.

In summary, all our models demonstrate comparable performance, achieving scores of over 0.30 for both accuracy and F1-score. This performance is quite satisfactory given the complexity of the classification task, which involves 114 classes. Specifically, the SVM exhibits the weakest performance, followed by the ensemble methods. The Gradient Boosting Machine (GBM) ultimately achieved the best results, while our custom-designed MLP was the second-best performer. This suggests that while our MLP design is with no doubt highly effective for this type of task, the GBM's capabilities are even more suited to handling the complexities involved.

5 Advanced Regression

In this chapter, we will focus on solving a regression task aiming to predict song popularity. To approach this task, we have opted to utilize the combined dataset of tracks and artists, because intuitively the artist's popularity and number of followers play a significant role in predicting the popularity of their song.

The continuous features of the dataset were normalized using a StandardScaler, while the categorical ones were treated differently based on the regression algorithm used. Specifically, the first method we will present involves Boosting with LightGBM, where categorical features were not encoded. However, for the second approach involving a neural network, categorical attributes were one-hot encoded. Lastly, in all experiments, the target attribute was scaled by dividing it by 100, to convert the range from $[0, 100]$ to $[0, 1]$.

The regressors will be evaluated based on MSE, MAE, R2 coefficient, and linear correlation with the vector of ground truth. Additionally, we'll attempt to interpret the Boosting model by analyzing the impact of features on popularity regression.

5.1 Boosting

The Light Gradient-Boosting Machine (LGBM) is a state-of-the-art boosting algorithm commonly employed for tabular data classification and regression tasks. Being histogram-based, it offers faster training times compared to other boosting algorithms, hence, we opted to employ this algorithm as our initial approach to predict song popularity.

Initially, we conducted hyperparameter tuning focusing on the regularization weights (L1 and L2) and the number of estimators in the ensemble. While the results were comparable across various configurations, we ultimately selected the setup yielding the highest R2 coefficient on the validation set. This configuration featured regularization weights set to 0.001 each and 300 estimators, resulting in an R2 value of 0.47. Below are the results of the model on the test set.

	MSE	**MAE**	**R2**
LGBM	0.0225	0.1040	0.47

Table 29: LGBM performances on test set.

The regressor results appear quite positive, with the R2 coefficient significantly greater than zero, so the model has a clear improvement in explaining the popularity of tracks compared to their mean. Additionally, the MAE is sufficiently low, suggesting that, on average, the prediction error of the tracks popularity is 10 points on a scale from 0 to 100; therefore, this error is seems acceptable for this type of task.

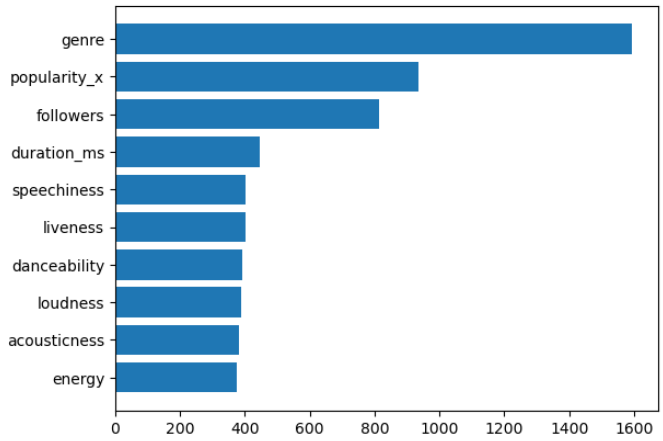


Figure 27: LGBM feature importance (`popularity_x` stands for the popularity of the artist).

Lastly, examining the feature importance of the model, we observed that the most important attributes for predicting the popularity of a track are the genre, the artist's popularity (`popularity_x`), and the number of artist followers, as we intuitively anticipated during the task definition.

5.1.1 Let's find a better explanation

Through the feature importance of our LGBM model, we have identified the most important features for regression. However, in a real-world scenario, this is often not sufficient. For example, it is quite obvious that the genre has a significant influence on the popularity of the song, but how do individual genres behave? Which genres does the model associate with an increase in popularity? And which genres have the opposite effect? The same questions could be posed for any feature in the dataset.

To attempt to answer these questions, we decided to train an Explainable Gradient Boosting Machine (EBM), in order to analyze the global and local impact of features on the regression. Below are the results on the test set.

	MSE	MAE	R2
EBM	0.0236	0.1096	0.44

Table 30: EBM performances on test set.

The regression results are slightly lower compared to those obtained with LGBM, but they are still comparable. Therefore, the model appears to be reliable enough to accurately analyze the influence of features.

By analyzing the global feature importance, we observed behaviors consistent with those of LGBM, with artist popularity and genre being the most important features. With this model, we could perform an analysis of individual genres to understand which genres correspond to an increase in popularity according to the model.

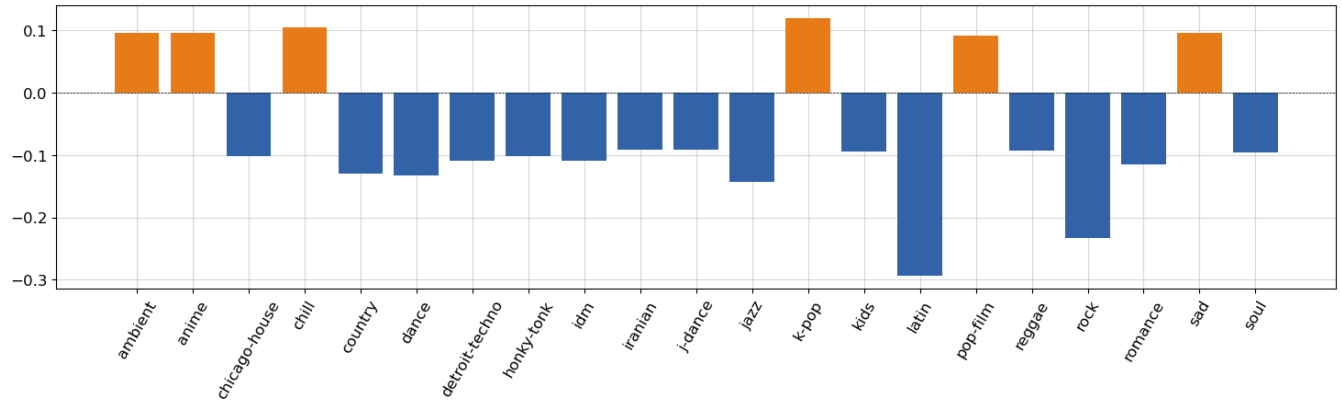


Figure 28: EBM impact of the most important genres for popularity regression.

As we can observe, genres like *latin*, *rock*, and *jazz* lead the model to significantly lower the predicted popularity of a song compared to the intercept (i.e., the average popularity in the training set). Thus, the model associates these genres with the lack of success of the track. Conversely, genres such as *k-pop*, *chill*, *anime*, and *pop-film* are associated with an increase in popularity relative to the average.

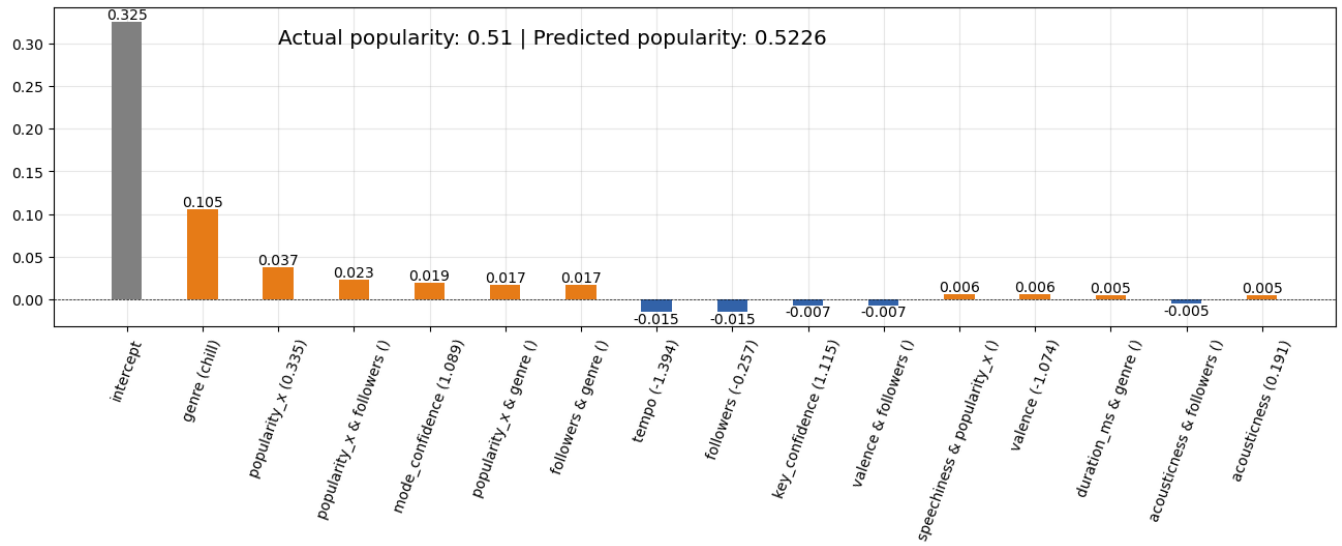


Figure 29: EBM local explanation of a track.

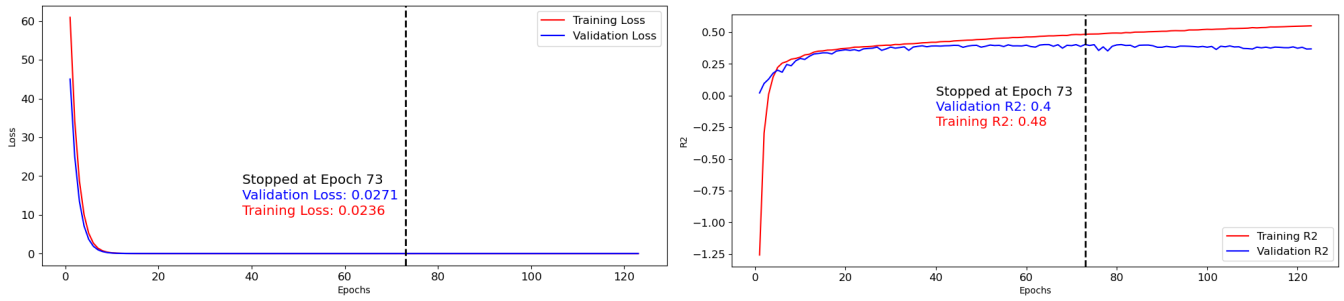
By analyzing the local explanation of a single record with a low prediction error, having utilized a glass-box model, we can observe the model's exact behavior in predicting the track's popularity. As shown in Figure 29, starting from an intercept value of 0.32, the model increases the predicted popularity by 0.10 for the *chill* genre and by 0.04 for the artist's popularity

being above average (the mean of continuous attributes is 0). Conversely, for example, the follower value of -0.26 causes the model to decrease the prediction by 0.02.

5.2 Neural Network

A completely different approach to predicting popularity was implemented using deep learning. We decided to utilize a neural network based on the structure used for genre classification (shown in Figure 22), adapted for the regression task. Consequently, the output layer was modified to have a single neuron with a sigmoid activation function, as the popularity ranges from $[0, 1]$, allowing for optimal modeling of the network's output.

The chosen loss function for training was Mean Squared Error (MSE), and the optimization algorithm was ADAM with a learning rate of 0.0001. Additionally, an early stopping criterion was employed, based on the R2 coefficient in the validation set, with a patience of 50 epochs.



(a) Loss

(b) R2

Figure 30: Plot of the loss and R2 coefficient per epoch for our MLP for regression.

As we can see from Figure 30, convergence was reached very quickly: the early stopping criterion was triggered at epoch 73. After this point, while the R2 score on the training set was still increasing, the score on the validation set began to decline. The results of the model on the test set are shown below.

	MSE	**MAE**	**R2**
MLP	0.0253	0.1135	0.40

Table 31: MLP performances on test set.

5.3 Conclusions

	MSE	**MAE**	**R2**
LGBM	0.0225	0.1040	0.47
EBM	0.0236	0.1096	0.44
MLP	0.0253	0.1135	0.40

Table 32: Total performances on test set.

In conclusion, we observe that the best results come from LGBM, which is also the fastest model to train among the three. The EBM still achieved good results, allowing us to conduct insightful analyses on the behavior of the regressor: we identified which genres are associated with an increase in track popularity (such as *k-pop*, *chill*, *anime*, etc.), and which ones are associated with the track's failure (like *latin*, *rock*, *jazz*, etc.). Additionally, the EBM enabled us to analyze the exact behavior of the model in predicting the popularity of individual records.

On the other hand, our neural network, besides being naturally more complex and computationally intensive to train, yielded the worst results (though still acceptable); moreover, the model is inherently challenging to explain, being a black-box. Hence, there seem to be no compelling reasons to prefer this neural network; instead, both boosting approaches deliver good performance and allowed for interesting observations on the models' behavior.

6 Explainability

6.1 Analysis on two classification cases

We conclude our report with some experiments focused on explainability. Specifically, we tried to open the Random Forest Classifier "box" detailed in Section 4.4. To achieve this, we employed the `SHAP` explanation strategy. `SHAP` is a model-agnostic, local method: being model-agnostic means it can theoretically be applied to any model, although we chose Random Forest to utilize the feature importance derived from the trees (see Figure 24b); local means it provides explanations for specific instances. In our experiments, we analyzed two different classification cases, both pertaining to the genre identified as the easiest to classify: the first case is a correctly classified `sleep` track with a high prediction probability of 0.76, while the second case is a `sleep` track misclassified as `comedy` with a high prediction probability of 0.74.

Let's start with the former case, the correct classification of a `sleep` track. The way the different features influenced the prediction is summarized in the image below.

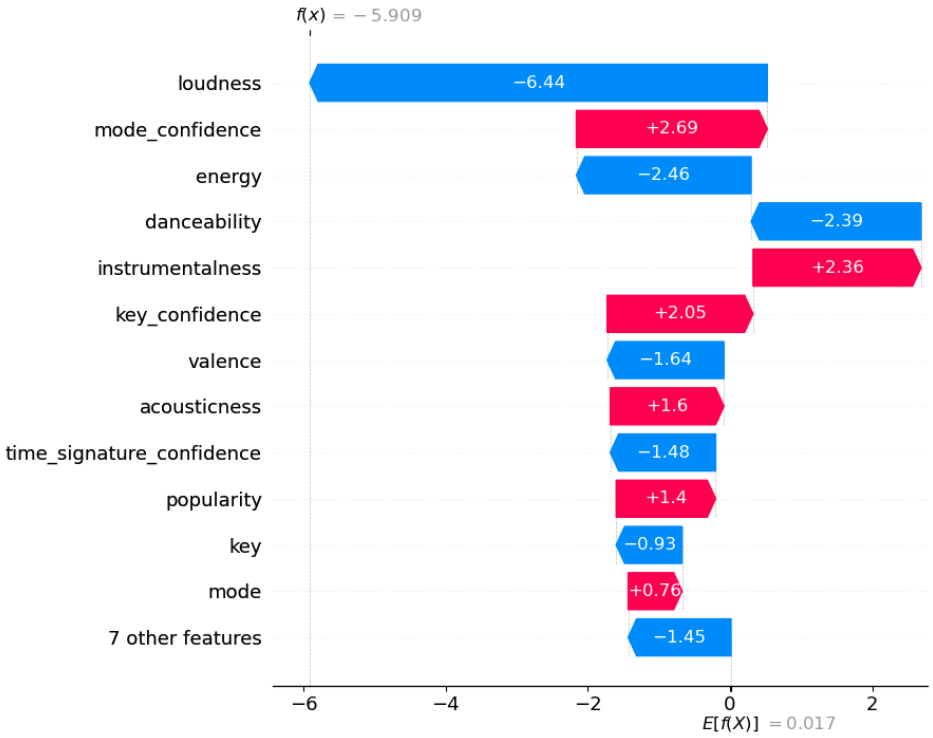


Figure 31: Features impact on the correct classification of a `sleep` track, with predict probability of 0.76.

The results obtained suggest several important considerations. First, in classifying this particular record—and likely all other `sleep` instances—the `loudness` feature plays a crucial role. The low value of this feature strongly influences its classification. Additionally, the significant impact of the `energy` and `danceability` features is not surprising. However, the presence of confidence measures among the influential features is more puzzling. Throughout our work with this dataset, the potential impact of these confidence features was unclear. Now, at the conclusion of our study, we reveal that they indeed significantly influence the predictions, at least for the instances we analyzed. Nevertheless, it remains difficult to understand why features like `mode_confidence` and `key_confidence` have a greater impact on classification than the `mode` and `key` features themselves.

Another noteworthy consideration is the difference between the impactful features identified here and those with high importance in Figure 24b. While it's expected that there wouldn't be a complete overlap between the two—given they represent different types of metrics—it's still significant that `popularity` and `duration_ms`, which were the most important features according to our Random Forest model, play a secondary role in the context of interpretability.

Let's now explore the impact of features on the second classification case.

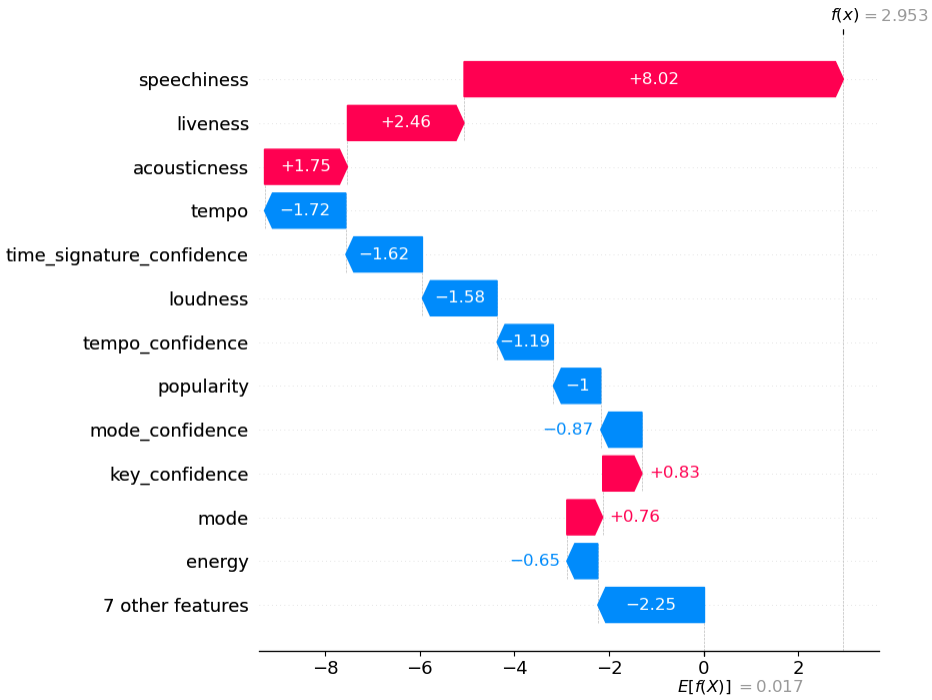


Figure 32: Features impact on the incorrect classification of a `sleep` track in a `comedy` track, with predict probability of 0.74.

The results differ significantly from the previous ones (indeed also the class identified is different). Specifically, a high value of `speechiness` heavily influences the classifier towards the `comedy` class, along with `liveness` and `acousticness`. The features that previously indicated the correct `sleep` class now play a minor role and do not significantly impact the prediction. As a result, the record is misclassified with a high level of certainty. This suggests that this instance might be an outlier or potentially mislabeled in the ground truth, warranting its removal from the dataset.

6.2 Conclusions

In this section, as well as in Section 5.1.1, we describe approaches to interpret the output of the models, specifically the reasons behind certain predictions. Our study on the Spotify dataset involved developing and analyzing several different models in terms of their performance. Despite the novelty of these analyses, we believe that incorporating explainability methods is essential for thoroughly understanding the behavior of the models we built. These methods are fundamental tools that provided us with unexpected results, offering new insights not only into the models themselves but also into the nature of the dataset.

With these latest results in hand, we are now satisfied and ready to conclude both the report and the research it details.