UNIVERSITÀ DI PISA

Data Mining 2 - Project

R. Anello, M. Poiani

A.Y. 2023/2024

Contents

Chapter 1

Data Understanding and Preparation

1.1 Introduction

This report presents an overview of our investigative endeavours and discoveries. We conducted an in-depth examination of our datasets to achieve a holistic comprehension of their components. The first dataset, which pertains to Spotify tracks, comprises 109547 entities and 34 features.The second dataset concerns artists associated with songs documented in the first dataset and comprises 30141 records and 5 features. Our initial task involved cleaning the data in both datasets, employing the approaches that we will describe in the following paragraphs. Once both datasets were prepared we merged them.

1.2 Data Semantic

In Table 1.1, we provide details about the tabular datasets variables, including their names, concise descriptions, and types.

Attribute name	Description	Type
id	The Spotify ID for the track.	Categorical
name	Name of the track.	Categorical
duration_ms	The track length in milliseconds.	Discrete
explicit	Whether the track has explicit lyrics or not.	Categorical
popularity	Quantifies the popularity of a track within a ranging from 0 to 100.	Discrete
artists	The artists' names who performed the track.	Categorical
album_Type	Specifies if the tracks appear in an album, a compilation or in a single.	Categorical
album_name	Specifies the album name in which the track is featured.	Categorical
danceability	Indicates how suitable a track is for dancing on a scale from 0.0 to 1.0.	Continuous
energy	Measures the level of energy on a scale from 0.0 to 1.0, serving as a way to evaluate intensity and activity levels.	Continuous
key	The key the track is in.	Categorical
loudness	The overall loudness of a track in decibels (dB).	Continuous
mode	The modality of a track, with 1 representing major and 0 representing minor.	Categorical
speechiness	Detects the presence of spoken words within a track.	Continuous
acousticness	A confidence measure from 0.0 to 1.0 of whether the track is acoustic.	Continuous

instrumentalness	Predicts the absence of vocals in a track. The closer to 1.0, the greater the likelihood the track contains no vocal content.	Continuous
liveness	Detects the presence of an audience in the recording, measured on a scale from 0.0 to 1.0.	Continuous
valence	A measure from 0.0 to 1.0 describing the musical positiveness	Continuous
tempo	The overall estimated tempo of a track in beats per minute (BPM).	Continuous
features_duration_ms	Denotes the duration of the track in milliseconds.	Discrete
time_signature	An estimated time signature. The time signature is a convention to specify how many beats are in each bar.	Categorical
n_beats	The total number of time intervals corresponding to beats across the duration of the track.	Discrete
n_bars	The total number of time intervals corresponding to bars across the duration of the track.	Discrete
popularity_confidence	The confidence in the song's popularity level, on a scale from 0.0 to 1.0.	Continuous
genre	The genre to which the track belongs.	Categorical
disc_number	The disc number (usually 1 unless the album consists of more than one disc).	Discrete
track_number	The number of the track. If an album has several discs, the track number is the number on the specified disc.	Discrete
album_release_date	The date the album was first released.	Datetime
album_release_date_precision	The precision with which album_release_date value is known ("year", "month", "day").	Categorical
album_total_tracks	The total number of tracks into an album	Discrete
start_of_fade_out	The time, in seconds, at which the track's fade-out period starts. If the track has no fade-out, this should match the track's length.	Continuous
tempo_confidence	The confidence, from 0.0 to 1.0, of the reliability of the tempo.	Continuous
time_signature_confidence	The confidence, from 0.0 to 1.0, of the reliability of the time_signature.	Continuous
key_confidence	The confidence, from 0.0 to 1.0, of the reliability of the key.	Continuous
mode_confidence	The confidence, from 0.0 to 1.0, of the reliability of the mode.	Continuous

Table 1.1: Tracks dataset variables description

In Table 1.2, we provide details about the second dataset, the one related to artists.

Attribute name	**Description**	**Type**
id	The Spotify ID for the artist.	Categorical
name	The name of the artist.	Categorical
popularity	The popularity of the artists in a range between 0 and 100. It is calculated from the popularity of all the artist's tracks.	Discrete
followers	Information about the artists' number of followers.	Discrete
genres	A list of the genres the artist is associated with. If not yet classified, the array is empty.	Categorical

Table 1.2: Artist dataset variables description

1.3 Data Quality

1.3.1 Duplicate rows assessment

We begin with the phase of identifying and eliminating duplicates within the track dataset. Initially, we uncovered 398 duplicate records, which were exact copies of each other. Subsequently, we utilized the track ID as the primary identifier to detect duplicate entries. This method effectively pinpointed 19985 instances of duplicated primary keys, all of which were promptly removed. We opted against using the song name as the primary identifier for two main reasons. Firstly, different songs can share the same title. Secondly, the same song can appear in different album types, such as compilations, standalone albums or single. In such cases, we treated them as distinct entities. This approach allowed us to retain more tracks and facilitated comparisons regarding the impact of album type on song popularity. Additionally, following our entity concept, we also eliminated songs where the ID was different but the title, artist's name, album type, and album name were identical. Only 317 such entries were identified. Ultimately, we were left with 89245 tracks. Moving on to the artist dataset, we encountered and removed two instances of duplicate artist IDs. Consistent with our methodology, we consider the artist ID as the primary key of the dataset due to the possibility of multiple artists sharing the same name.

1.3.2 Missing Values

We examined both datasets for missing values and found that the dataset regarding tracks had no missing values. However, in the dataset pertaining to artists, our analysis revealed two missing values in the 'name' column. Given the relevance of that information, we promptly removed both rows. The issue became more pronounced in the 'genres' columns for artists, where 7556 lists were empty, effectively constituting missing values. We addressed the presence of these missing values by leveraging the option to merge the two datasets. This enabled us to directly extract the artist's genre from the associated song, thereby resolving the issue.

1.4 Variable transformations

During the Data Preparation phase, significant attention was devoted to the "album_release_date" attribute, whose values were originally represented as strings. To facilitate further analysis, they were converted into datetime format. Subsequently, in anticipation of more granular analyses, feature extraction was performed on the aforementioned variable, resulting in the creation of "album_release_year" and "album_release_month". Another significant transformation focused on the "start_of_fade_out" variable, which presented two challenges: it was measured in seconds, complicating comparisons with "duration_ms", and its analytical significance was limited. To address these issues, the seconds were converted into milliseconds. Afterwards, this transformed variable was subtracted from "duration_ms", resulting in a new attribute named "fade_out_duration". This attribute denotes the duration of the fade out, with an intended minimum value of 0, signifying tracks without a fade out. However, 39 records erroneously showed values slightly lower than 0 due to measurement errors. These values were adjusted to 0 to ensure data consistency. Three new variables were introduced to the "tracks" dataset through merging with the "artists" dataset. "Primary_artist_popularity" reflects the popularity of the primary artist listed in the "artists" attribute, as the featuring artists are typically listed after the main one. On the other hand, "average_artists_popularity" considers the popularity of all artists contributing to a track and presents the average value. Lastly, "sum_of_followers" represents the total sum of followers for all artists featured in a track. These transformations enhance the dataset's usability and set the stage for more insightful analyses of the music tracks data.

1.5 Pairwise correlations and eventual variables removal

In Figure 1.1, we present the correlation matrix for all numerical variables in our dataset. Beginning with the most substantial correlation coefficients, a strong correlation of 0.98 is evident between "n_bars" and "n_beats". This association is easily explained by the inherent connection between the number of beats and bars in a song, with the latter forming the structural framework for beats. Another noteworthy correlation arises between a track's duration in milliseconds and its count of beats or bars, yielding a coefficient of 0.84. This relationship is straightforward, with longer songs naturally containing more beats and bars, while shorter tracks exhibit fewer musical elements. An insightful observation reveals a significant correlation (0.76) between "loudness" and

"energy". This result is entirely anticipated, emphasizing the interplay between a track's intensity and activity and its overall loudness. An evident correlation is observed between "primary_artists_popularity" and "average_artists_popularity", given the substantial influence of the former on the latter's resulting value. Correlations regarding "feature_duration_ms" won't be mentioned, as it essentially mirrors "duration_ms" with a correlation coefficient of 1. Additionally, a notable correlation exists between "album_total_tracks" and "track_number" (0.79), as album size increases, there is a tendency for track numbers to correspondingly rise. Finally, two noteworthy negative correlations are observed within the dataset. These correlations link the "acousticness" variable with "loudness" (-0.58) and "energy" (-0.73). These negative relationships imply that as a track's acoustic characteristics become more prominent, its loudness decreases, and its energy level diminishes.

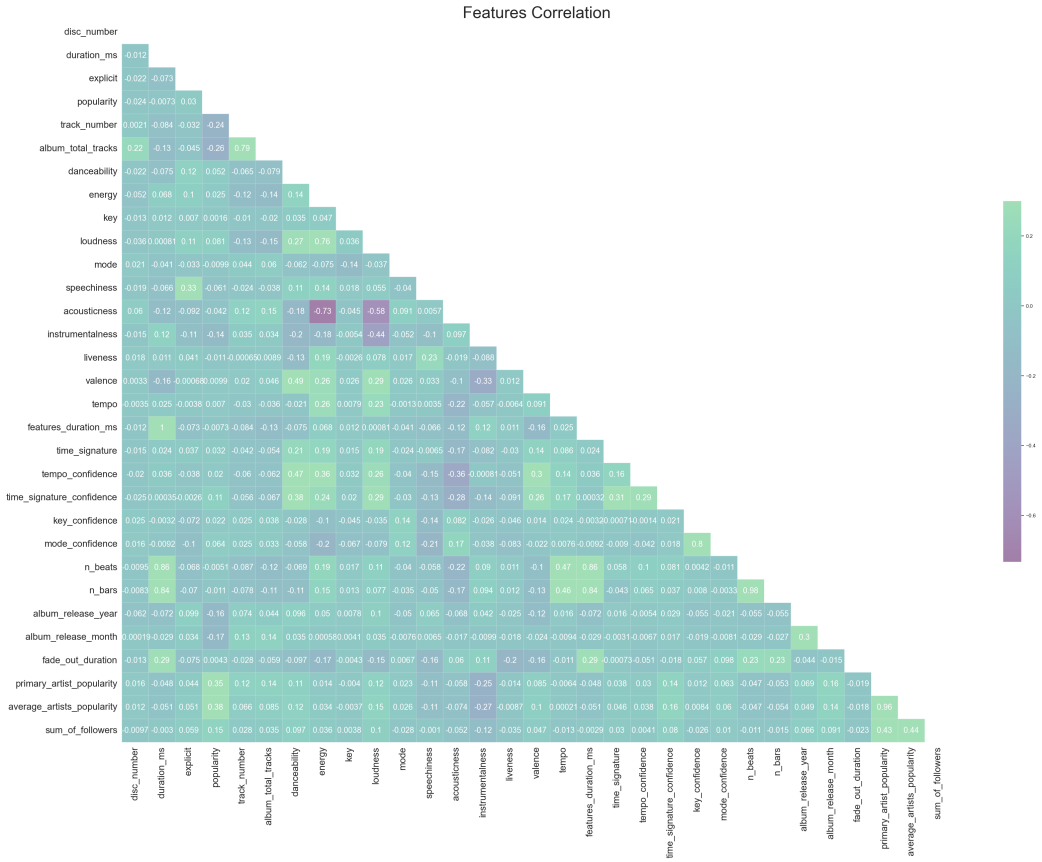


Figure 1.1: Triangular Matrix for Features Correlation

Table 1.3: Variables Removed from Dataset

Variable	**Reason for Removal**
features_duration_ms	Substantial redundancy with "duration_ms" attribute
n_bars	High correlation coefficient (0.86) with "duration_ms"
acousticness	High negative correlation coefficient (-0.73) with "energy"
loudness	High correlation coefficient (0.76) with "energy"
disc_number	High imbalance and lack of useful insights (98% of values were equal to 1)
track_number	High correlation coefficient (0.79) with "album_total_tracks"
tempo_confidence	Mean and median below 0.5, indicating lack of reliability for "tempo" attribute
key_confidence	Mean below 0.5, indicating lack of reliability for "key" attribute
time_signature_confidence	Values consistently close to 1, redundant for reliability of "time signature"
mode_confidence	Mean almost 0.51, indicating lack of reliability for "key" attribute
tempo, mode, key	Correspondent confidence levels not reliable enough

In Table 1.3, there is a comprehensive list of the variables that were removed from the dataset during the variable elimination process. Each variable is accompanied by a brief explanation detailing the reason for its removal.

1.6 Distribution of the variables and statistics

In this segment, we explore the distribution of variables and associated statistics to unveil patterns, central tendencies, and variability within the dataset. In Figure 1.2, we explore the frequency distributions of numerical variables within the dataset. In constructing the histograms, we adopted 17 bins based on the guidance provided by Sturges' rule. Beginning with the top-left subplot, we delve into the analysis of the "popularity" attribute. Despite a slight positive skew, we also observe hints of a bimodal distribution. This phenomenon arises from a substantial portion of records characterized by a value of zero or close to it in the popularity attribute, as evidenced by the 13th percentile being exactly zero. However, upon excluding these records, the distribution appears relatively balanced around the mean and median values. Notably, the distribution is sparse at the upper end, with the 95th percentile capped at 67. The distribution of the "danceability" feature stands out as the only one resembling a Gaussian distribution, likely due to the even distribution of tracks across genres. A similar pattern would be expected for the "energy" attribute, but it exhibits a negative skew, indicating a prevalence of high-energy tracks in the dataset. Conversely, the distributions of "instrumentalness," "liveness," and "speechiness" are positively skewed, indicating a concentration of data points towards lower values, with a few extreme values pulling the mean to the right. Of particular interest is the plot for "instrumentalness," which hints at a potential bimodal distribution. This aligns with the characteristic features of many songs, which present either an extremely low instrumentalness value, when vocals are prominent, or an extremely high one, typical of many electronic genres like IDM, techno, or house. Among categorical variables, the distribution of album types in Figure 1.3 is notable. The majority of tracks

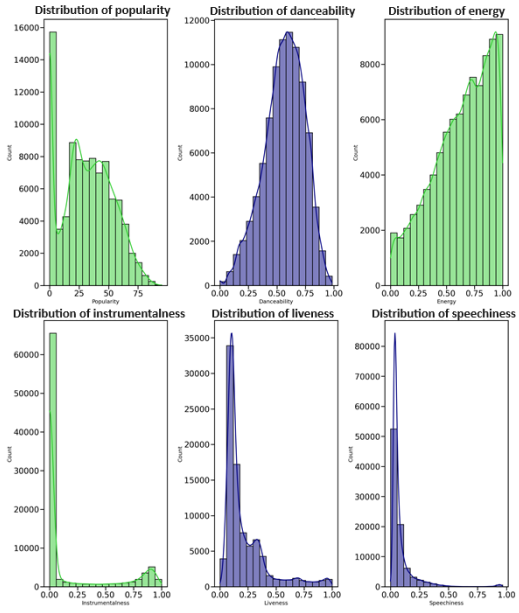


Figure 1.2: Numerical attributes distribution

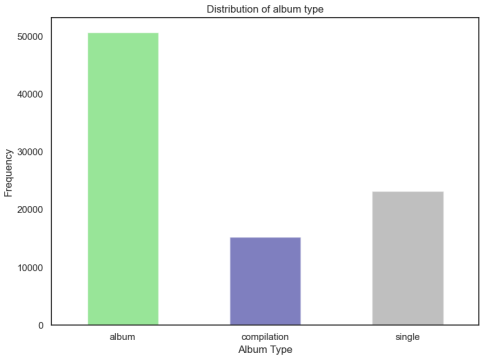


Figure 1.3: Album Type distribution

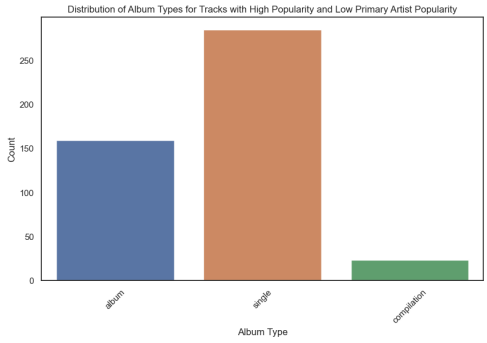


Figure 1.4: Conditional Album Type distribution

belong to the "album" category, followed by "single" and "compilation." This finding aligns with the common perception that albums are the primary form of track release. However, as depicted in Figure 1.4, exceptions exist, particularly when examining popular tracks with low primary artist popularity. In this scenario, it is not surprising to find a higher proportion of "single" releases compared to "album" releases. This observation may stem from lesser-known artists producing one-hit wonders, where popularity is associated more with the

individual track than with the artist's complete album offerings.

1.6.1 Genre Analysis of Top 1000 Artists and Tracks ordered by popularity

Before merging the datasets, we analyzed each independently. Our focus was on the top 1000 tracks and artists ranked by popularity, aiming to explore any potential connections between them, particularly concerning genres. To achieve this, we first processed the dataset containing artist information and we verified that the first 1000 artists by popularity have a valid genre, since many of them had the "genre" field empty. The "genre" column originally contained genre lists in string format, which we converted into actual lists. For the tracks dataset, we directly sorted all rows by descending track popularity and extracted the first 1000 rows. After this preparation, we analyzed the most common genres among both artists and tracks. The most popular genres were pop, K-pop, alternative-rock, rock, hip-hop, reggaeton, house, dance, indie, and indie-pop. In the visualization 1.5, we grouped similar genres under broader labels, such as combining pop and K-pop, alternative rock and rock, indie and indie-pop. In the end, we discovered that K-pop/pop was the most popular macro genre with respect to both artists and tracks.

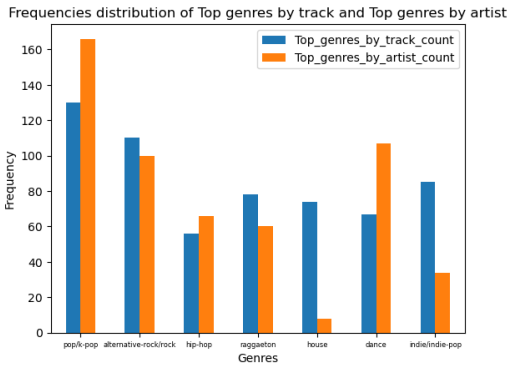


Figure 1.5: Frequencies distribution of Top genres by track and Top genres by artist'

1.6.2 Seasonality

We endeavoured to utilize the "album_release_date" feature to uncover potential seasonal trends followed by artists when releasing songs throughout the year. However, the data proved somewhat inadequate for this task due to the presence of three different types of release date precisions for songs. The majority of songs have daily precision, while some have yearly precision, and a few have monthly precision. As a result, we can exclusively rely on the year component of the album release date to gain some insight into the temporal distribution of releases, as depicted in picture 1.6, but cannot draw meaningful conclusions since our dataset is heavily imbalanced towards 2022 songs. Indeed, we observed that approximately 25% of the songs were released in 2022. Specifically, the majority of these releases occurred on the 7th, 14th, and 21st of October 2022.

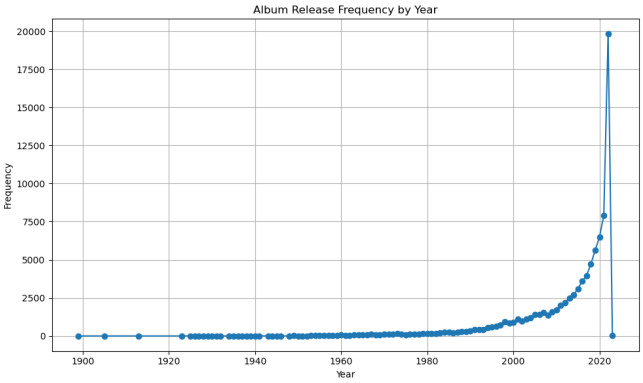


Figure 1.6: Track Release date by year

1.7 Clustering

In preparation for advanced classification analysis, we conducted a K-Means clustering on our dataset aiming to group the 114 different genres into clusters that encapsulate their distinct characteristics. Our approach was guided by the selection of attributes known to be most "discriminative" for the diverse range of genres, followed by a Principal Components Analysis (PCA), in order to reduce dimensionality. With 5 principal components explaining over 90% of the cumulative variance, we proceeded to determine the optimal number of clusters through experimentation, ultimately selecting 5 clusters. Figure 1.7 showcases the distribution of data points in the principal component space, while Figure 1.8 illustrates how the clusters are delineated based on "energy" and "instrumentalness" attributes, revealing clear distinctions among the clusters. Examining the cluster characteristics, we made insightful observations, shown in Figure 1.9. Cluster 3, denoted by the purple color, was labeled as "High_insLow_en" and encompasses genres like sleep, new-age, ambient, and classical, characterized by minimal vocals and a slow rhythm. Differently, Cluster 4, designated as "High_insHigh_en" and

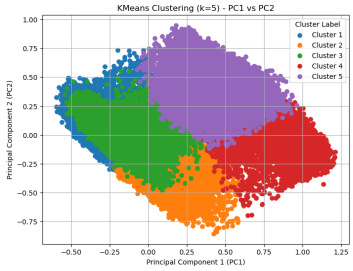


Figure 1.7: PCA visualization of k=5 clustering

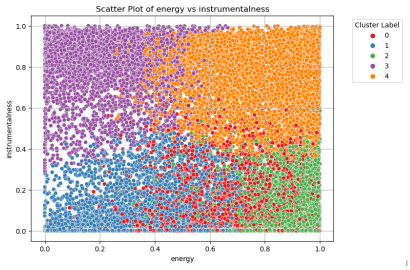


Figure 1.8: "energy" vs "instrumentalness" clustering visualization

represented in orange, comprises techno and house sub-genres, known for their instrumental nature and high energy levels. The clusters below exhibit higher degrees of vocal presence with varying energy levels. Cluster 0, labeled "Low_insMid_en," includes romantic and mandopop songs; Cluster 1, "Low_insLow_en", consists primarily of kids songs and salsa, requiring some energy for children engagement and danceability. Lastly, Cluster 2, identified as "Low_insHigh_en" and depicted in green, predominantly comprises metal songs and sub-genres, notoriously characterized by high energy levels and vocal intensity.

High_insHigh_en:		High_insLow_en:		Low_insHigh_en:		Low_insMid_en:		Low_insLow_en:	
genre		genre		genre		genre		genre	
detroit-techno	659	new-age	744	comedy	765	salsa	876	mandopop	770
grindcore	614	ambient	691	heavy-metal	737	forro	782	romance	697
techno	550	sleep	588	metalcore	720	kids	765	show-tunes	687
chicago-house	509	study	509	hardstyle	651	children	675	cantopop	646
minimal-techno	502	classical	449	grunge	573	party	638	opera	636
...		...		...		...		...	

Figure 1.9: Distribution of genres in the five clusters

The clustering analysis successfully grouped genres based on their distinctive attributes, laying the foundation for further analysis and exploration in the realm of advanced classification.

1.8 Time Series dataset

In our Time Series Analysis, we analyzed a dataset consisting of 10000 unique time series records, each representing the spectral centroids of MP3 audio files. Each record contained the associated track ID and genre information, covering selections from twenty unique genres. The dataset contained 9998 time series with a length of 1280 data points, one series with a length of 1263, which was adjusted to 1280 by averaging the last five observations, and one last series with a length of 830 which was excluded from the dataset.

1.8.1 Preprocessing steps: PAA Approximation and Normalization

Recognizing the computational challenges posed by the dataset's size, we opted to perform Piecewise Aggregate Approximation (PAA) to reduce the time series length from 1280 to 256. We chose PAA over alternative techniques due to its capability to retain crucial information regarding diverse trends and patterns, while also exhibiting robustness to noise. PAA achieves this by averaging values within segments to approximate the data, a process that aids in mitigating the impact of outliers, as shown in picture 1.10. In preparation for clustering and classification tasks, two distinct datasets were created with different scaling methods applied: MinMax scaling for clustering and standard scaling for classification. These scaling techniques were chosen not only for their ability to enhance the performance of the analyses but also for their capacity to ensure consistency and comparability across the dataset features.

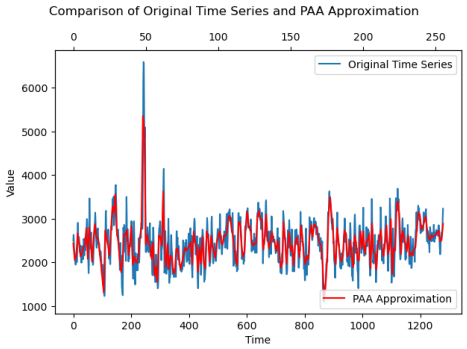


Figure 1.10: Comparison of the first original time series and its PAA approximation

Chapter 2

Time Series Analysis

2.1 Time Series Similarity

In our Time Series analysis, we adopted two distinct approaches to measure similarity: Structural-based similarity and Shape-based similarity. The former was integral to our analysis for custom feature extraction in time series clustering, as detailed in Section 2.2.3. On the other hand, Shape-based similarity was central to our methodology, where we utilized two primary distance metrics: Euclidean and Dynamic Time Warping (DTW).

Euclidean distance emerged as a valuable choice due to its straightforward interpretability and computational efficiency. Given that our Time Series data exhibited minimal distortion and were reasonably well-aligned, Euclidean distance facilitated rapid computations. Additionally, our prior application of Piecewise Aggregate Approximation (PAA) contributed to the smoother representation of our Time Series data, which complemented Euclidean distance's performance.

Conversely, DTW excels in capturing temporal distortions, shifts, and non-linear alignments between Time Series. However, in our specific context, DTW did not yield optimal results. This could be attributed to the relatively low prevalence of distortion in our Time Series data and the uniform length of all records, mitigating the need for DTW's capabilities.

2.2 Clustering

In our Time Series Clustering analysis, we employed various clustering techniques to explore the structure and patterns within the data. These included K-Means clustering with both Euclidean distance and Dynamic Time Warping (DTW), as well as Custom Feature Extraction, Hierarchical Clustering using different linkage methods, and DBSCAN.

2.2.1 K-Means with Euclidean distance

The initial approach we undertook involved applying Time Series K-Means with Euclidean distance to our preprocessed dataset, which had been subjected to PAA approximation and MinMax normalization. Initially, we experimented with different values of k, including k=7 and k=11, but after analyzing the knee graph, we determined that the optimal configuration for K-Means was k=3. This choice yielded a silhouette score of nearly 0.25, indicating a reasonable degree of separation between the clusters. Upon examining the cluster sizes, we observed that Cluster 0 and Cluster 2 contained slightly over 4000 records each, while Cluster 1 comprised around 1500 records. Figure 2.1 illustrates the distribution of the clustered dataset based on its mean and standard deviation. Notably, we found distinct differences in the mean and standard deviation values across the clusters. For instance, Cluster 1 exhibited the lowest mean distribution coupled with the lowest standard deviation, resulting in minimal spikes. These observations led us to investigate the potential relationship between cluster characteristics and track genres. Since the time series represented the spectral centroids of the tracks, variations in the average frequency content of audio signals among clusters could be linked to different genres. Genres associated with higher energy levels typically exhibit higher mean frequencies, since they tend to contain more high-frequency components, such as cymbals, high-pitched vocals, and sharp transients, which contribute to a sense of excitement and intensity. Figure 2.2 provides insight into the genre distribution within two contrasting

clusters: Cluster 0, characterized by higher mean frequency values, and Cluster 1 the opposite. Cluster 0 predominantly comprised energetic genres such as happy, j-idol, progressive house and heavy metal. In contrast, Cluster 1 was primarily composed of slower genres like piano, new age, sleep and opera. These findings were pivotal in affirming the efficacy of our clustering approach and provided valuable insights into the underlying structure of the dataset.

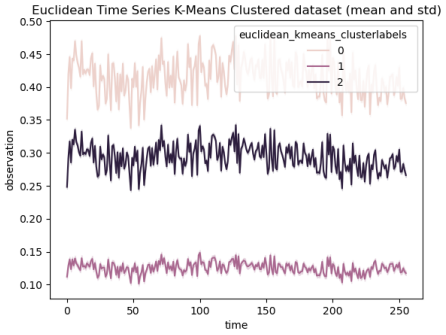


Figure 2.1: Euclidean TS K-Means clustered dataset (mean and std)

Cluster0:		Cluster1:	
happy	468	piano	383
j-idol	431	new-age	369
progressive-house	378	sleep	242
heavy-metal	375	opera	119
salsa	335	songwriter	107
kids	312	folk	81
synth-pop	309	honky-tonk	73

Figure 2.2: Genre distribution in Cluster 0 and Cluster 1

Recognizing the suitability of our clustering approach for our requirements, we opted to visualize the clusters using three distinct techniques. Initially, we conducted Principal Components Analysis (PCA) with five principal components and visualized the clusters in the first two dimensions (Figure 2.3), followed by a three-dimensional representation (Figure 2.4). The results demonstrated clear differentiation among the three clusters, with minimal overlap between them. This indicates that the clusters capture distinct patterns and characteristics within the dataset.

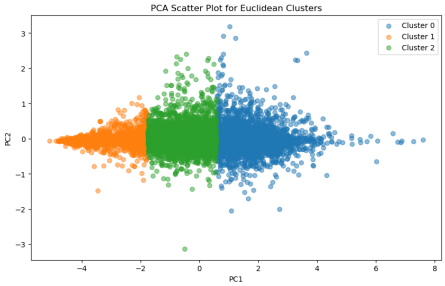


Figure 2.3: 2-dimensional PCA Scatter Plot

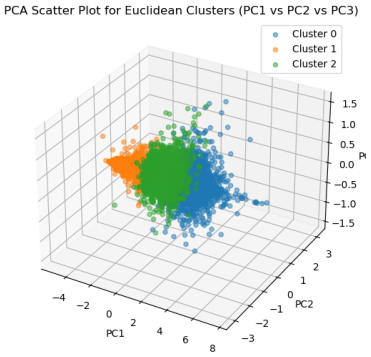


Figure 2.4: 3-dimensional PCA Scatter Plot

Subsequently, we employed another dimensionality reduction technique, Variance Thresholding. After experimenting with different threshold values, we settled on a threshold equal to the mean variance plus twice the standard deviation variance. We visualized the clusters in the first two dimensions, as shown in Figure 2.5, although the choice of dimensions had minimal impact due to their similar variances. While the clusters remained distinguishable and separable, the visual separation was slightly less pronounced compared to PCA. This could be attributed to the nature of the Variance Thresholding technique, which may not capture complex relationships as effectively as PCA. Lastly, we merged the Time Series dataset with the tabular dataset containing track information, introduced in chapter 1, by matching TrackIDs. Subsequently, we selected "energy" and "instrumentalness" as the two most representative continuous features to represent the clusters. Figure 2.6 illustrates the relationship between these features and the clusters. Notably, the cluster with the lowest mean frequency and fewer spikes predominantly comprised tracks with high instrumentalness and low energy. In contrast, Cluster 0 exhibited a mix of tracks with both high and low instrumentalness but uniformly high energy values, aligning with our expectations based on genre characteristics.

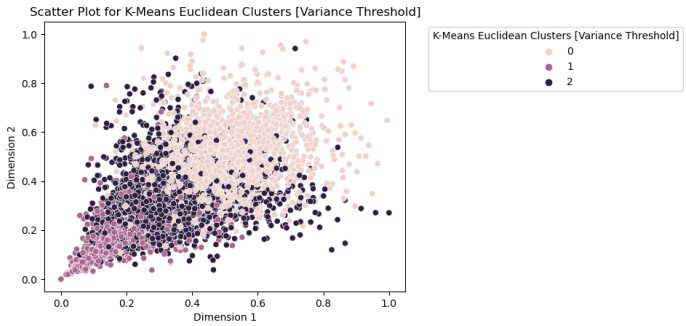


Figure 2.5: 2-dimensional Variance Threshold SP

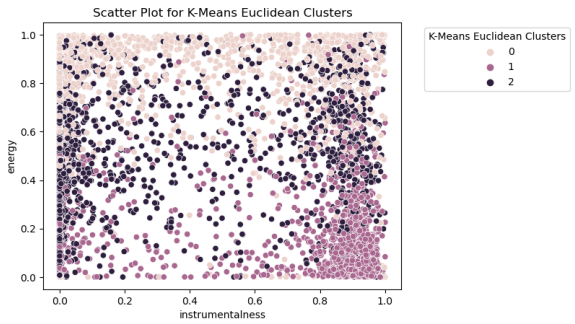


Figure 2.6: energy vs instrumentalness SP

2.2.2 K-Means with Dynamic Time Warping

An analogous analysis was conducted utilizing K-Means with Dynamic Time Warping (DTW) as the distance metric. Performing the same preparation steps, we maintained a consistent choice of k-value at k=3 for both approaches. However, the outcomes were marginally less satisfying. While the silhouette score slightly improved to 0.26, a mere 0.01 higher than that of the Euclidean approach, and the visual representation of the clustered dataset based on mean and standard deviation closely resembled the one depicted in Figure 2.1, notable disparities arose in terms of cluster size and genre distribution within the clusters. In this instance, Cluster 0 encompassed approximately 5000 records, Cluster 2 contained around 4000 records, and Cluster 1, associated with "slow" genres, comprised just under 1000 records. As a result, significant consequences arose, especially regarding the composition of Cluster 0, as shown in Figure 2.7. Whereas previously, with the Euclidean distance metric, this cluster predominantly comprised energetic songs, now it exhibited a less discerning profile, with almost one-fifth of its components stemming from genres such as sleep, emo, or opera. This shift also affected Cluster 1, which, despite still hosting a substantial portion of the aforementioned genres, lost some of its distinctive character. This divergence in outcomes between the Euclidean and DTW approaches can be attributed to the nature of DTW itself. Unlike Euclidean distance, which measures the straight-line distance between two points, DTW accounts for the temporal dynamics inherent in time series data. While this makes DTW well-suited for capturing similarities between sequences that may vary in length or exhibit temporal distortions, it may be overly complex for our dataset, where all time series have uniform length and exhibit few distortions, thanks in part to the smoothing effect of PAA approximation. Moreover , it's plausible that tracks of similar genres show distinct musical patterns like higher-pitched sections, crescendos, and slower parts, which are reflected in the spectral centroid time series. By utilizing Euclidean distance, we effectively capture the overall similarity in patterns within the spectral centroid time series, which are indicative of genre characteristics: this allows for a more straightforward and interpretable clustering outcome.

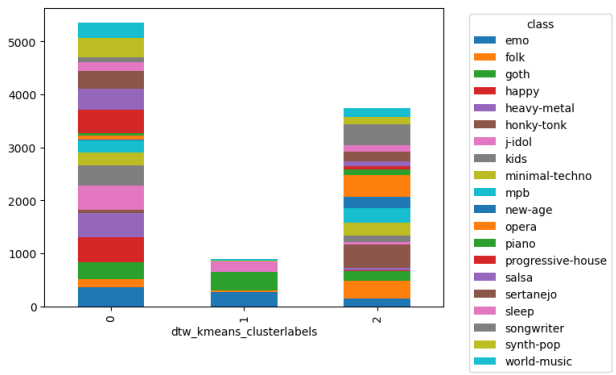


Figure 2.7: DTW K-Means cluster sizing with "genre" distribution

2.2.3 Feature Extraction

We pursued a feature extraction approach to explore an alternative clustering method based on statistical features extracted from the time series data. Our preprocessing steps, including approximation and normalization, mirrored those discussed in the Euclidean K-Means clustering approach. Specifically, the utilization of MinMaxScaler led to the reduction of significant statistical features available for extraction. Notably, features like variance and standard deviation became redundant due to the constrained range of values (0 to 1), while Skew and Kurtosis lost their relevance. We extracted the following features from each time series: mean, median, 10th percentile, 25th percentile, 50th percentile, 75th percentile, and 90th percentile. This resulted in a dataset

comprising all time series and 7 features, which were utilized for clustering. The knee graph analysis indicated that the optimal values for K were 3 and 4, with silhouette scores of 0.465 and 0.446 respectively. Initially, we selected 4 clusters, but the results were unsatisfactory, while clustering with 3 clusters yielded results resembling those of the previous Euclidean K-Means clustering.

Euclidean K-Means vs Feature Extraction Clustering Comparison

In the comparison presented in Figure 2.8, the distribution of genres between Cluster 0, predominantly energetic songs, and Cluster 1, featuring less energetic ones, exhibited minor disparities. Specifically, Cluster 0 allocation appeared slightly inferior compared to the Euclidean K-Means clustering, with some tracks being incorrectly assigned to Cluster 2. Conversely, the clustering of slower-paced songs demonstrated slightly improved accuracy in Feature Extraction compared to its counterpart. While the Feature Extraction K-Means method boasted a higher silhouette score than the alternative technique, our comparative analysis underscores the multifaceted nature of evaluating clustering methods. It's noteworthy that the feature extraction-based approach, with its reliance on statistical summaries and preprocessing techniques like PAA-approximation, MinMax scaling, and feature engineering, may have simplified data representation and distorted feature distributions, potentially influencing clustering outcomes and interpretability. Conversely, the Euclidean K-Means method offered a transparent approach by utilizing the entire time series as features for clustering. Our domain expertise in music genres did not favor one method over the other, as both produced clusters consistent with expectations. However, given the transparency and preservation of detail inherent in the Euclidean K-Means method, we found it to be the preferred approach.

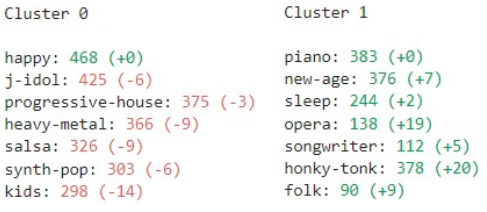


Figure 2.8: Feature Extraction Clustering: genre distribution in Cluster 0 and Cluster 1 [with difference from genre distribution in corresponding Time Series Euclidean K-Means clusters]

2.2.4 Exploration of Alternative Techniques

Following the methodologies detailed previously, we ventured into exploring alternative approaches. Hierarchical Clustering was among the techniques investigated, with experimentation conducted on single, complete and average linkage. However, despite exhaustive attempts to refine the dendrogram's cut, none of the resulting clusterings proved satisfactory. Invariably, a predominant cluster would encompass the majority of data points, with smaller clusters offering limited insight, irrespective of the chosen number of clusters or cutting distance. Similar challenges arose when exploring Density-based clustering algorithms, such as DBSCAN. Despite thorough exploration of various combinations of radius and MinPts parameters, the outcomes consistently fell short of expectations. Results typically manifested as either a single cluster with a few small clusters or singletons, or an overwhelming amount of noise accompanied by clusters of limited utility. This suggests that the inherent structure of the data may not be conducive to effective clustering using these techniques.

2.2.5 Conclusions

In summary, our exploration of clustering techniques for time series analysis revealed the effectiveness of Time Series K-Means with Euclidean distance in delineating distinct genre profiles based on spectral centroid patterns. Dynamic Time Warping, while capable of capturing temporal dynamics, proved less optimal for our dataset with uniform length and few distortions. Furthermore, the feature extraction-based approach demonstrated satisfactory results, comparable to those of K-Means Euclidean clustering. However, some feature engineering choices may have impacted its performance, while the Euclidean K-Means method, operating directly on the entire time series data, offered transparency and detail preservation, aligning with our preference. Finally, our exploration of alternative techniques such as Hierarchical Clustering and DBSCAN proved to be very ineffective.

2.3 Motifs/Discords and Shapelets

Extracting motifs and discords is a valuable endeavour as it enables the identification of recurring patterns within each time series. A motif represents a subsequence that recurs throughout the time series, and we focused our

attention on them since we were interested in understanding the underlying structure and recurring patterns in each track, without considering anomalies or discords. For this task, we opted to extract the top motif from each time series along with its closest nearest neighbour. To commence this task, we first generated the matrix profile. Leveraging the matrix profile facilitates the identification and extraction of motifs with greater efficiency as shown in figure 2.9. We have chosen to set the fixed length of each motif to 8. By utilizing the indices of both the first and second occurrences of the motif, we constructed a dataset comprising 19998 rows and 8 features, representing the 8 values (window) assumed by the motif.

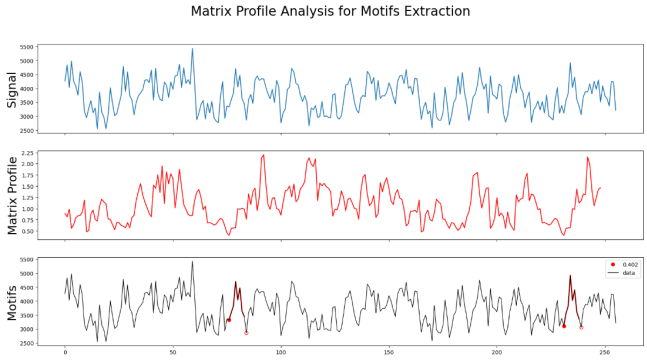


Figure 2.9: Signal: Represents the first approximated time series of the dataset; Matrix profile: it is the matrix profile of the first time series of the dataset; Motifs: it is the representation of the top motif detected in the first time series

2.3.1 Clustering on motifs

At this stage, we endeavoured to conduct clustering using motifs, a method particularly beneficial for lengthy time series datasets. This approach allows for the exploration of the underlying structure of the time series by analyzing its most significant segments. In our case, we employed this technique to assess whether the resulting clusters would resemble those derived from clustering the entire approximated time series using Euclidean distance. The outcome proved quite surprising. Clustering based on time series motifs yielded results strikingly similar to those obtained from clustering the entire time series, as we can notice from the distribution of the tracks in the different clusters in figure 2.10. Once again, we observed the emergence of three distinct clusters. A smaller cluster, characterized by less energetic genres, exhibited increased discriminative power, with a higher proportion of songs falling under categories such as sleep and piano. The first larger cluster comprised more energetic songs, including happy, j-idol, and heavy metal. Lastly, the third cluster, though less discriminative, encompassed all genres, with a minimal presence of less energetic songs that were of negligible significance.

cluster	genre	
0	happy	415
	j-idol	393
	progressive-house	312
	heavy-metal	303
	salsa	291
	kids	256
	synth-pop	247

cluster	genre	
1	piano	416
	new-age	358
	sleep	255
	songwriter	181
	opera	164
	honky-tonk	133
	folk	92

Figure 2.10: Genres distribution in motifs clustering (cluster 0 and cluster 1)

2.3.2 Shapelets extraction

We began by extracting shapelets from the approximated time series, aiming to capture representative subsequences that could effectively classify tracks based on their genre. Our focus was on identifying the most discriminative shapelets to enhance classification accuracy. Initially, we extracted the top 1000 shapelets. However, upon evaluation, we observed diminishing returns in terms of information gain beyond the top 100 shapelets. These top 100 shapelets exhibited information gains ranging from 0.07 (for the first) to 0.009 (for the 100th). To ensure consistency and effectiveness, we set a minimum length of 8 for the shapelets and left the maximum length unrestricted. Subsequently, we calculated the minimum distance between each time series and each shapelet. This process yielded a dataset comprising 9999 rows and 100 columns, with each column representing the minimum distance between a given shapelet and each time series in the dataset.

2.3.3 Motifs vs Shapelets

We established a connection between shapelets and motifs by selecting the most discriminative motifs based on qualitative assessment. These selected motifs were then repurposed as shapelets, thereby enriching the pool of shapelets in a non-randomized way. Subsequently, this augmented set of shapelets was leveraged to enhance the performance of the classifier, as will be later illustrated in Section 2.4.2. This symbiotic relationship between motifs and shapelets illustrates their potential for mutual reinforcement within time series analysis.

2.4 Classification

For the Classification phase of our Time Series analysis, we employed various techniques including K-NN with both Euclidean distance and DTW, Shapelets-based classification and the ROCKET algorithm. The primary objective was to predict the 20 different genres associated with the Time Series data. Additionally, we pursued a secondary task, aligning with the scale of our dataset, which involved classifying the three clusters obtained from our best clustering approach: Time Series K-Means using Euclidean distance. Preprocessing steps for classification included applying PAA approximation, akin to the Clustering phase, and normalizing the dataset using the Standard Scaler, which is preferred over MinMax normalization for classification tasks. Furthermore, our dataset, comprising 9999 records, was divided into a training set consisting of 9000 records and a testing set comprising the remaining 999 records. Additionally, to ensure the robustness of our model and mitigate potential biases, we validated the 9000 training set records using 3-fold cross-validation.

2.4.1 K-NN

Predicting 20 genres

Initially, K-NN classification with the Euclidean distance metric was employed for predicting the 20 different genres. However, despite efforts to optimize the model by selecting the appropriate number of nearest neighbors using the knee point method, it became evident that this task posed significant challenges. Even after thorough validation of the training data, predicting the remaining 999 test instances proved to be extremely difficult. An attempt to address this by rebalancing the train and test sets, including 500 additional test records in the training set, did not yield satisfactory results. The classification report in Figure 2.11 illustrates the poor performance of the classifier, with notable struggles across most genres. Of particular note is the class "sleep", which exhibited a high recall but low precision, indicating many false positives. Conversely, the class "synth-pop" demonstrated a very high precision but extremely low recall, suggesting that while the classifier correctly identified very few instances of "synth-pop", it failed to capture the majority of them. Despite these challenges, an alternative approach using DTW as the distance metric was also explored, yielding similar outcomes as depicted in Figure 2.12. While the change in distance metric from Euclidean to DTW had some nuanced effects on specific class predictions, the overall performance of the K-NN classifier remained unsatisfactory. These findings suggest that the inherent complexity of the task may present significant obstacles for this approach, regardless of the distance metric utilized.

Classification Report:

	precision	recall	f1-score	support
emo	0.08	0.04	0.05	25
folk	0.31	0.17	0.22	30
goth	0.03	0.04	0.03	25
happy	0.22	0.30	0.25	23
heavy-metal	0.03	0.05	0.03	22
honky-tonk	0.60	0.12	0.21	24
j-idol	0.07	0.04	0.05	25
kids	0.00	0.00	0.00	26
minimal-techno	0.58	0.27	0.37	26
mpb	0.00	0.00	0.00	24
new-age	0.16	0.27	0.20	22
opera	0.19	0.21	0.20	19
piano	0.50	0.12	0.19	25
progressive-house	0.57	0.20	0.30	20
salsa	0.00	0.00	0.00	25
sertanejo	0.00	0.00	0.00	19
sleep	0.08	0.95	0.14	21
songwriter	0.00	0.00	0.00	32
synth-pop	1.00	0.03	0.06	30
world-music	0.00	0.00	0.00	36
accuracy			0.13	499
macro avg	0.22	0.14	0.12	499
weighted avg	0.22	0.13	0.11	499

Figure 2.11: K-NN classification report with Euclidean distance

Classification Report:

	precision	recall	f1-score	support
emo	0.05	0.02	0.03	46
folk	0.28	0.12	0.17	57
goth	0.02	0.02	0.02	44
happy	0.24	0.27	0.25	59
heavy-metal	0.01	0.02	0.02	41
honky-tonk	0.32	0.13	0.19	45
j-idol	0.12	0.09	0.10	46
kids	1.00	0.02	0.04	55
minimal-techno	0.62	0.34	0.44	53
mpb	0.00	0.00	0.00	49
new-age	0.17	0.26	0.21	54
opera	0.16	0.17	0.17	40
piano	0.42	0.16	0.24	49
progressive-house	0.41	0.15	0.22	47
salsa	0.00	0.00	0.00	55
sertanejo	0.00	0.00	0.00	36
sleep	0.08	0.88	0.14	43
songwriter	0.00	0.00	0.00	60
synth-pop	1.00	0.02	0.03	60
world-music	0.00	0.00	0.00	60
accuracy			0.13	999
macro avg	0.25	0.13	0.11	999
weighted avg	0.26	0.13	0.11	999

Figure 2.12: K-NN classification report with DTW distance

Predicting 3 classes

Following the challenging classification task involving 20 genres, our subsequent exploration into K-NN classification of the three clusters retrieved from time series K-Means clustering with Euclidean distance revealed noteworthy insights, as shown in Figure 2.13. Upon validation, the K-NN classifier employing Euclidean distance demonstrated robust performance, achieving high precision, recall, and F1-scores across all clusters (0, 1, 2) in each fold. Conversely, employing DTW distance led to a notable decline in classification performance for the same clusters. Precision, recall, and F1-score metrics for all classes experienced a decrease, resulting in a validation accuracy of only 42%.

The macro-average F1-score similarly diminished, indicating an overall deterioration in model efficacy compared to the Euclidean-based approach. Testing results echoed the trends observed during validation. The Euclidean-based model maintained its strong performance on the test set, achieving an accuracy of over 96%. In contrast, the DTW-based model struggled, with a lower accuracy of 43%, underscoring its difficulty in generalizing learned patterns to unseen data. The effectiveness of the Euclidean K-NN model in predicting the three clusters resulting from Time Series K-Means Euclidean clustering can be attributed to several factors. Firstly, for the nature of spectral centroids, as discussed in clustering [Section 2.2.2], tracks of similar genres exhibit specific patterns of musical elements, which are reflected in the spectral centroid time series data. Additionally, since the clustering was performed using Euclidean distance, the resulting clusters are naturally more suited for classification tasks based on the same distance metric. This consistency ensures that the Euclidean K-NN model performs well across validation folds and test sets, indicating effective generalization.

K-NN with Euclidean distance

Classification Report for fold 1

	precision	recall	f1-score	support
0	0.96	0.98	0.97	1258
1	0.95	0.97	0.96	467
2	0.97	0.94	0.95	1275
accuracy			0.96	3000
macro avg	0.96	0.96	0.96	3000
weighted avg	0.96	0.96	0.96	3000

Classification Report for fold 2

	precision	recall	f1-score	support
0	0.95	0.98	0.96	1297
1	0.96	0.98	0.97	464
2	0.97	0.93	0.95	1239
accuracy			0.96	3000
macro avg	0.96	0.96	0.96	3000
weighted avg	0.96	0.96	0.96	3000

Classification Report for fold 3

	precision	recall	f1-score	support
0	0.98	0.97	0.97	1272
1	0.95	0.99	0.97	477
2	0.96	0.96	0.96	1251
accuracy			0.97	3000
macro avg	0.96	0.97	0.97	3000
weighted avg	0.97	0.97	0.97	3000

Classification Report for Test Set:

	precision	recall	f1-score	support
0	0.97	0.97	0.97	429
1	0.95	0.99	0.97	160
2	0.96	0.94	0.95	410
accuracy			0.96	999
macro avg	0.96	0.97	0.96	999
weighted avg	0.96	0.96	0.96	999

K-NN with DTW distance

Classification Report for fold 1

	precision	recall	f1-score	support
0	0.42	0.58	0.49	1258
1	0.20	0.06	0.09	467
2	0.44	0.40	0.42	1275
accuracy			0.42	3000
macro avg	0.35	0.34	0.33	3000
weighted avg	0.40	0.42	0.40	3000

Classification Report for fold 2

	precision	recall	f1-score	support
0	0.44	0.54	0.49	1297
1	0.20	0.05	0.08	464
2	0.42	0.44	0.43	1239
accuracy			0.42	3000
macro avg	0.36	0.35	0.34	3000
weighted avg	0.40	0.42	0.40	3000

Classification Report for fold 3

	precision	recall	f1-score	support
0	0.42	0.59	0.49	1272
1	0.24	0.05	0.08	477
2	0.41	0.37	0.39	1251
accuracy			0.41	3000
macro avg	0.36	0.34	0.32	3000
weighted avg	0.39	0.41	0.38	3000

Classification Report for the test set

	precision	recall	f1-score	support
0	0.44	0.58	0.50	429
1	0.22	0.06	0.09	160
2	0.43	0.40	0.42	410
accuracy			0.43	999
macro avg	0.36	0.35	0.34	999
weighted avg	0.40	0.43	0.40	999

Figure 2.13: K-NN for 3 classes, Euclidean vs DTW

2.4.2 Shapelets-based classification

After defining the dataset containing the minimum distance of each time series from each shapelet, we applied a decision tree to classify the 20 genres. We executed the split introduced at the beginning of this paragraph, resulting in a training set consisting of 9000 records and a test set comprising 999 records. Initially, we evaluated the model without conducting any hyperparameter tuning, yielding poor performances. The overall accuracy of the model was 0.14, with the best-classified genre being "minimal-techno" with precision of 0.50. Subsequently, as shown in figure 2.14 we conducted a random search to determine the optimal parameters for the model. As a result, the overall accuracy slightly improved to 0.16 and "minimal-techno" remained the best-classified genre, achieving a precision of 0.56. Other genres, such as "happy", "progressive-house" and "sleep" also demonstrated improved precision. Despite the efforts in hyperparameter tuning, the results remained unsatisfactory. At this point, instead of incorporating more random shapelets, we chose to leverage motifs as shapelets. Utilising a qualitative approach, we carefully selected the most discriminative motifs. Specifically, we extracted all motifs from cluster 1, predominantly representing less energetic

	precision	recall	f1-score	support
emo	0.11	0.09	0.10	46
folk	0.06	0.04	0.04	57
goth	0.00	0.00	0.00	44
happy	0.48	0.41	0.44	59
heavy-metal	0.07	0.07	0.07	41
honky-tonk	0.11	0.09	0.10	45
j-idol	0.07	0.24	0.11	46
kids	0.19	0.18	0.19	55
minimal-techno	0.56	0.53	0.54	53
mpb	0.00	0.00	0.00	49
new-age	0.26	0.35	0.30	54
opera	0.17	0.20	0.18	40
piano	0.10	0.08	0.09	49
progressive-house	0.38	0.28	0.32	47
salsa	0.08	0.09	0.09	55
sertanejo	0.08	0.25	0.12	36
sleep	0.27	0.30	0.29	43
songwriter	0.11	0.05	0.07	60
synth-pop	0.04	0.05	0.04	60
world-music	0.00	0.00	0.00	60
accuracy			0.16	999
macro avg	0.16	0.16	0.15	999
weighted avg	0.16	0.16	0.16	999

	precision	recall	f1-score	support
emo	0.20	0.04	0.07	46
folk	0.12	0.04	0.05	57
goth	0.00	0.00	0.00	44
happy	0.60	0.51	0.55	59
heavy-metal	0.19	0.32	0.23	41
honky-tonk	0.20	0.56	0.30	45
j-idol	0.26	0.24	0.25	46
kids	0.24	0.18	0.21	55
minimal-techno	0.64	0.40	0.49	53
mpb	0.12	0.18	0.15	49
new-age	0.42	0.52	0.46	54
opera	0.44	0.28	0.34	40
piano	0.58	0.39	0.46	49
progressive-house	0.19	0.15	0.17	47
salsa	0.17	0.38	0.23	55
sertanejo	0.13	0.39	0.19	36
sleep	0.53	0.63	0.57	43
songwriter	0.21	0.10	0.13	60
synth-pop	0.29	0.07	0.11	60
world-music	0.16	0.12	0.14	60
accuracy			0.27	999
macro avg	0.28	0.27	0.26	999
weighted avg	0.29	0.27	0.25	999

Figure 2.14: Performance comparison between shapelets-bassed classification (left) and shapelet-based classification adding motifs (right)

songs. Our rationale was that these motifs would effectively identify less energetic tracks and, dichotomously, those with higher energy. We randomly selected 20 motifs and computed the minimum distance between each motif and every time series in our dataset, similar to the procedure we followed for the shapelets. This procedure was repeated 5 times to mitigate bias, selecting at every iteration 20 different motifs. The decision tree classification was then performed again using these updated train and test sets, now both with 120 features. We achieved notable improvement, with the decision tree model achieving an overall accuracy of 0.27, a significant improvement from the previous best model's accuracy of 0.16. Furthermore, there was an enhancement in precision for the majority of the genres, in particular, "minimal-techno" reached a precision of 0.64, "happy" achieved 0.60, "piano" attained 0.58, and "sleep" reached 0.53. As imagined, the classifier exhibited higher precision in classifying all genres, as depicted in figure 2.14.

2.4.3 ROCKET

In this study, we utilize the ROCKET algorithm for time series classification, leveraging its efficacy in handling high-dimensional data. Selected for its capacity to efficiently capture temporal patterns, particularly well-suited in music genre prediction tasks, ROCKET enhances computational efficiency. The inclusion of "mini-rocket" further boosts efficiency while mantaining robust predictive capabilities.

Predicting 20 genres

In the primary classification task, where we predict the 20 genres, ROCKET exhibited exceptional performance, surpassing the best K-NN model. Specifically, as shown in Figure 2.15, the ROCKET classification achieved an impressive overall accuracy of 0.43. Notably, the highest precision was attained in predicting the class "sleep," with a value of 0.75. Additionally, several other classes were accurately predicted; for instance, "minimal-techno" achieved a precision of 0.73, while "piano" reached a precision of 0.67. These precision values significantly outperform those obtained with K-NN using Euclidean distance. Based on these results, we can confidently conclude that ROCKET stands out as the superior classification model for this task.

Classification Report:

	precision	recall	f1-score	support
emo	0.42	0.30	0.35	46
folk	0.16	0.09	0.11	57
goth	0.10	0.05	0.06	44
happy	0.68	0.69	0.69	59
heavy-metal	0.38	0.56	0.46	41
honky-tonk	0.39	0.60	0.47	45
j-idol	0.35	0.39	0.37	46
kids	0.45	0.47	0.46	55
minimal-techno	0.73	0.70	0.71	53
mpb	0.15	0.08	0.11	49
new-age	0.51	0.57	0.54	54
opera	0.40	0.57	0.47	40
piano	0.67	0.53	0.59	49
progressive-house	0.41	0.60	0.48	47
salsa	0.38	0.51	0.44	55
sertanejo	0.22	0.42	0.29	36
sleep	0.75	0.77	0.76	43
songwriter	0.20	0.18	0.19	60
synth-pop	0.37	0.27	0.31	60
world-music	0.36	0.17	0.23	60
accuracy			0.42	999
macro avg	0.40	0.43	0.40	999
weighted avg	0.41	0.42	0.40	999

Figure 2.15: Rocket classification report for 20 classes

Predicting 3 classes

In the classification task involving the three clusters derived from Time Series K-Means with Euclidean distance, ROCKET demonstrates exceptional performance. ROCKET's accuracy closely rivals that of K-NN with Euclidean distance. This parity highlights the suitability of ROCKET for classification tasks involving time series, affirming its efficacy in capturing temporal patterns and extracting meaningful features from the data. Furthermore, ROCKET outperformed the Euclidean K-NN classifier in predicting instances belonging to Cluster 1, characterized by slow songs, with higher precision (0.99) and slightly lower recall (0.96) compared to the Euclidean KNN's precision of 0.95 and recall of 0.99. This indicates ROCKET's superior ability to accurately classify instances within this discriminative cluster while minimizing false positives, emphasizing its efficacy in capturing subtle patterns within time series data.

2.4.4 Conclusions

In our Time Series analysis, despite extensive trial and error, K-NN faced significant challenges in predicting 20 genres, mainly due to task complexity. However, Euclidean K-NN excelled in classifying three clusters derived from Time Series K-Means, benefitting from algorithmic compatibility and effective generalization. However, DTW-based K-NN did not yield satisfactory results, suggesting limitations in capturing the nuanced patterns within the data. Shapelets-based classification performance was initially not satisfactory, but improved significantly with motif selection, highlighting the impact of feature engineering on model performance. ROCKET outperformed K-NN in predicting 20 genres, demonstrating superior efficiency and accuracy in capturing temporal patterns. However, in the task involving 3 clusters, both ROCKET and Euclidean K-NN achieved similar performance, showcasing comparable capabilities. Overall, ROCKET emerged as the most effective classifier, underscoring the importance of thoughtful algorithm selection in Time Series analysis for robust model outcomes.

Chapter 3

Advanced Data-Processing

3.1 Outlier Detection

3.1.1 Introduction

Up to this point, we have addressed duplicates in our dataset but have not considered outliers. Our task was to detect and handle the top 1% outliers and we performed it using three techniques, which we analyze in detail in the following paragraphs. The first technique was Local Outlier Factor (LOF), an unsupervised, local, and density-based method, which we initially deemed suitable for our dataset. The second technique was HBOS, an unsupervised, global method that we enhanced with LODA, an ensemble approach. The third technique was Isolation Forest, a global, model-based method, along with one of its state-of-the-art variants, Deep Isolation Forest. For this task, we focused on 15 features that could potentially identify outliers such as "duration_ms", "valence" and "energy" and disregarded categorical variables with high cardinality such as "id" and "album name". Features like "album type" were also excluded as they were deemed less helpful in outlier detection. Before applying any technique, all values were standardized for consistency.

3.1.2 Local Outlier Factor (LOF)

In this section, we are going to explore the LOF method. Since our dataset is condensed, we attempted to isolate points that were far from the dense regions. However, instead of labeling these points as outliers a priori, we considered their relative density using the LOF method. This local approach assigns labels to the points, 0 if a point is an inlier and 1 if a point is an outlier. When compared with the results of other techniques, the outliers detected by the other methods were mostly consistent among them, while the anomalies detected by LOF were diverse. Consequently, we considered the results less reliable. We experimented with various configurations, changing the value of K from 3 to 25 and the contamination, representing the proportion of the dataset expected to be outliers, from 0.01 to 0.02. By visualizing the LOF results using PCA, as shown in Figure 3.1, we understood why the outliers were different from those detected by the other methods. The visualization showed that the relative density in this context was a drawback rather than an advantage. Points in denser areas that were slightly different from others were considered outliers, while points in less dense areas were not considered outliers. We concluded that in this context, a different density-based technique that considers global density rather than local density might be more suitable.

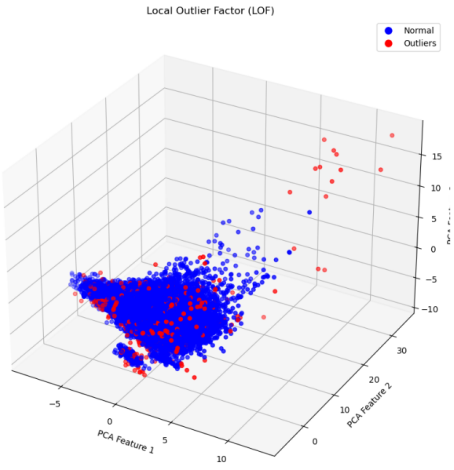


Figure 3.1: Outliers detected by LOF

3.1.3 Histogram-based Outlier Score (HBOS) and Lightweight On-line Detector of Anomalies (LODA)

In this section, we explored Histogram-based Outlier Score (HBOS) and Lightweight On-line Detector of Anomalies (LODA) for detecting outliers in our dataset. HBOS constructs individual histograms for each feature, assuming feature independence. While feature independence can't be guaranteed, we mitigated this by removing all highly correlated feature in Section 1.5, increasing our confidence in this assumption.

The output of HBOS is an outlier score, which is the inverse of the estimated density. We experimented with various parameters, such as the "alpha" parameter for regularizing the histogram and the contamination level, trying values between 0.01 and 0.02. We started with 17 bins, according to Sturges' Rule, and increased the number of bins to add more detail to our results. The best results were achieved with 17 bins and an alpha of 0.05. We normalized the outlier score frequency distribution, as seen in Figure 3.2 to improve comparability. With a contamination of 0.01, the threshold for detecting outliers was 0.74.

To enhance our results, we used an ensemble-based approach called Lightweight On-line Detector of Anomalies (LODA). LODA is essentially an ensemble of HBOS, using multiple one-dimensional histograms constructed from input space projected onto randomly generated vectors. The main parameter we tuned for LODA, unlike HBOS, was the number of random cuts. We experimented with 50, 75, 100, which is the default value, and 125 random cuts. We found that 75 random cuts provided the best balance of accuracy and robustness, as shown in Figure 3.3. We also used a contamination level of 0.02 to further evaluate the discovered outliers with different techniques.

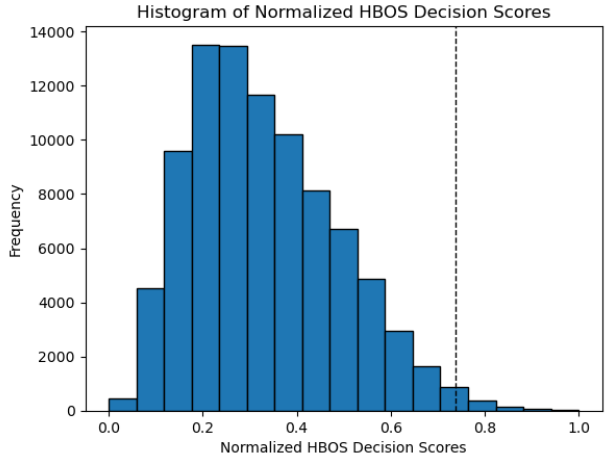


Figure 3.2: Histogram of Normalized HBOS Decision Scores

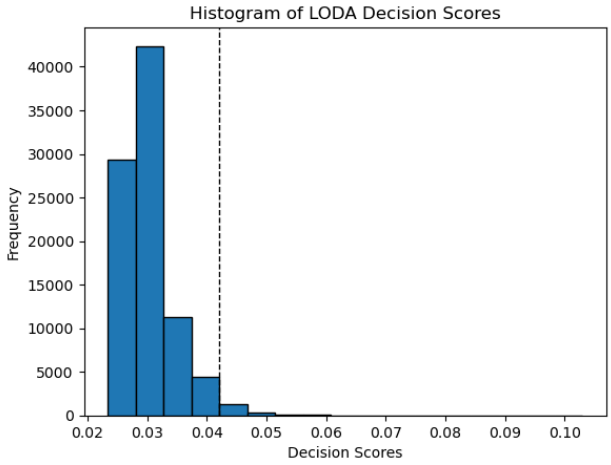


Figure 3.3: Histogram of LOFA Decision Scores

Finally, we computed the similarity metrics of HBOS and LODA the results. Unfortunately, they exhibited very low F1-score, precision, and recall, rendering HBOS results unsatisfactory. As for LODA outliers, we reserve judgement until comparing them with the results of more sophisticated Outlier Detection techniques.

3.1.4 Isolation Forest

The first model-based approach we utilized was the Isolation Forest algorithm. We chose a contamination value of 0.02, resulting in the identification of 1,785 outliers. To compare these results with those from previous techniques, we standardized the output by mapping the default Isolation Forest values of 1 for inliers and -1 for outliers to 0 for inliers and 1 for outliers. To identify a preliminary set of "pure outliers," we cross-referenced the outliers detected by the Isolation Forest with those identified by the previously applied LODA technique, finding 978 matching records. However, our assessment of their effectiveness was inconclusive upon visualizing the outliers using Principal Component Analysis (PCA). We found that the identified outliers did not exhibit satisfactory characteristics that would make them effective outliers. Consequently, we opted to extend our analysis beyond this model in pursuit of identifying the most accurate set of outliers possible.

3.1.5 Deep Isolation Forest

To evaluate our previous results and establish a form of "ground truth", we needed an extremely sophisticated outlier detection method. The decision was between Extended Isolation Forest (EIF) and Deep Isolation Forest (DIF), both state-of-the-art extensions of Isolation Forest. Given the composition and high dimensionality of our dataset, we opted for DIF, which effectively combines deep learning, through autoencoders, with Isolation Forest, transforming data into a lower-dimensional space before applying the Isolation Forest algorithm. This approach helps capture complex patterns and non-linear relationships within the data. We chose to use the default values for most parameters in DIF, specifically the hidden activation function, tanh, batch size of 1000, max_samples equal to 256, number of ensembles equal to 50, and number of estimators equal to 6. These parameters were selected to balance computational efficiency and model performance. The only parameter we adjusted was contamination, setting it at 0.02 to facilitate comparison with the results from previous techniques, LODA and Isolation Forest. The DIF model identified 123 outliers. Upon comparing these with the outliers detected by the other two methods, we found that 113 outliers were consistently identified across all three methods. These were immediately removed from the dataset, as we deemed them "pure outliers".

3.1.6 Dealing with outliers

At this point, we returned to the matching outliers between LODA and Isolation Forest, now consisting of 865 records. Since these were not yet considered "pure outliers", we aimed to reclassify them as inliers. We began by plotting the frequency distribution of these outliers across all their features. From this analysis, we identified four features (popularity, danceability, energy, and valence) that showed the most significant differences between the mean of the "candidate" outliers' distribution and the mean of the entire dataset. To address the most anomalous values, we established upper and lower bounds for each of these four features. The bounds were set by multiplying the mean of the entire dataset by 1.25 for the upper bound and dividing by 1.25 for the lower bound. For any feature where an outlier's value fell outside this interval, we substituted that value with the dataset's mean. This approach helped mitigate extremely anomalous values in the features with the greatest discrepancies between the outliers and the total distribution. The results of this process are illustrated in Figure 3.4. In the graph, the mean of the distribution of outliers before this process is indicated by a red vertical line, the new mean is represented by a green vertical line, and the mean of the entire dataset is shown as a black vertical line. We can observe that the new mean is now much closer to that of the entire distribution. Up to this point, we had only removed 113 records. Our goal was to remove around 1% of the top outliers, so we needed to identify more outliers and check whether the candidate outliers we attempted to transform into inliers survived subsequent outlier detection runs. We iteratively ran Deep Isolation Forest until successfully removing around 1000 outliers. Our total number of records was now 88284 (down from 89270 before outlier detection). After verifying whether our previous candidate outliers were removed from the dataset, we found that only about 20% of the 865 records had been removed.

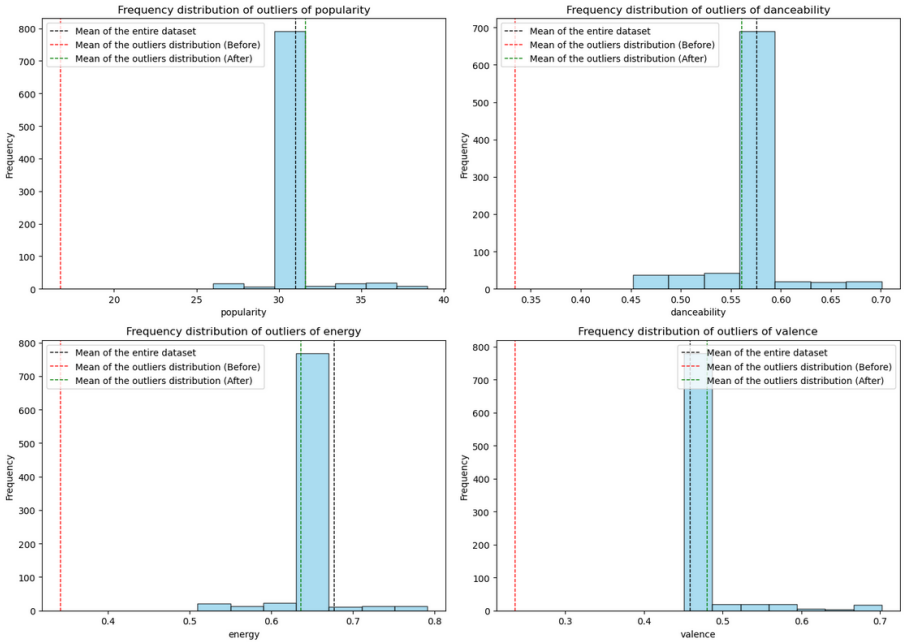


Figure 3.4: "Candidate" Outliers transformation

3.1.7 Another approach: "Top-down" Outlier Detection

Up until now, our method could be described as "bottom-up". We initially removed the smallest number of pure outliers, tried to fix the candidate outliers that were not detected by Deep Isolation Forest in its first run, and then iteratively ran DIF until we reached our desired number of outliers to remove. We used this approach based on a mix of intuition and domain knowledge, assuming that there were not many outliers in our dataset and that we had to be careful in detecting them. This consideration was applied through a consistently small value of the contamination level across all the techniques we utilized. By always setting it at 0.01 or 0.02, we declared that our dataset was "contaminated" by outliers in only a small percentage of its total records. However, since there was no decisive hint or proof that this was the case, we opted to pursue another approach, which we described as "top-down". In this approach, we assumed that our initial dataset was highly contaminated by outliers, making it better to first detect a large number of them and then remove those which proved to be the "top 1% outliers." For this method, the choice of the contamination parameter was key. We decided to set it at 0.1 for every employed technique, but apart from that, we maintained all other parameters unchanged from before. This time, we began with LODA, using 75 random cuts, and discovered 8921 outliers. Then, we applied Isolation Forest and still found 8921 outliers. Lastly, we ran Deep Isolation Forest with default parameters and obtained 718 outliers. To select the purest outliers among these three lists, we merged the results by picking the matching records, and in the end, we successfully detected 677 outliers. We visualized our findings using Principal Components Analysis (PCA) with 7 components and a 3D scatter plot, as shown in Figure 3.5. We instantly noticed similarities between this visualization and Figure 3.1, but this time, the detected outliers appeared more coherent, and the overall visualization was better. Ultimately, we opted for removing all of these outliers, resulting in a dataset of 88530 records.

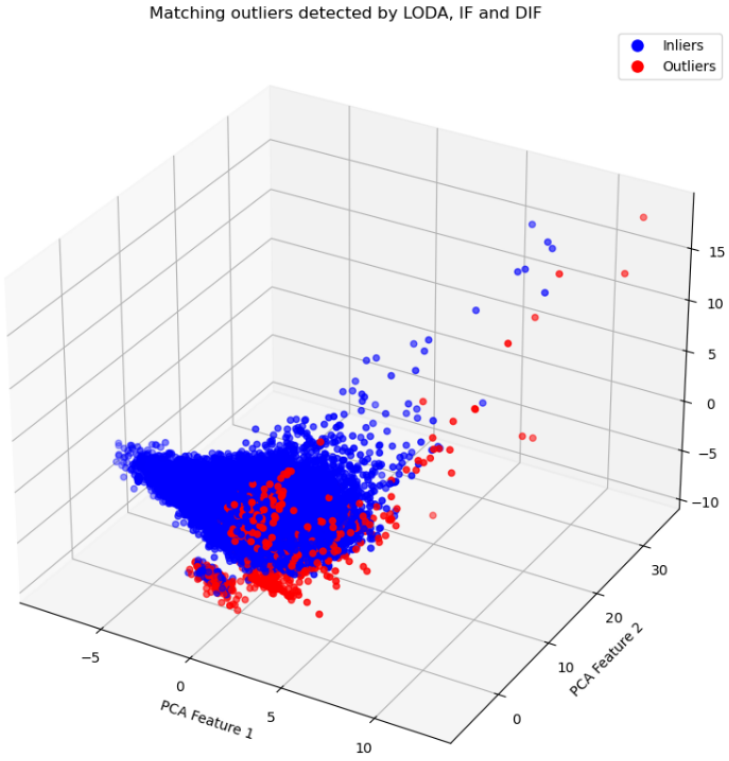


Figure 3.5: Matching Outliers LODA, IF and DIF ("top-down approach")

3.1.8 Merging the two approaches and Conclusions

After pursuing both the "bottom-up" and "top-down" approaches, we were left with two "almost-clean" datasets: one with 88284 rows and the other with 88530 rows. We compared these datasets and found that exactly 88000 records were consistent across both, while 814 records did not match. These 814 records were outliers detected by one approach but not the other. To merge the two approaches, we decided to remove these outliers, resulting in a final dataset of 88000 rows. This outcome reflects the efforts of both our "bottom-up" and "top-down" outlier detection strategies. In total, we removed 1270 records, and the modifications applied in Section 3.1.6 have been maintained for approximately 600 records in the final dataset.

3.2 Imbalanced Learning

3.2.1 Minority class selection and task introduction

In this section, we explore strategies to handle imbalanced learning within the context of binary classification. Specifically, we focused on creating a minority class consisting of tracks from catchy Japanese genres: 'J-idol', 'J-dance', 'Anime', and 'J-pop'; the majority class, instead, comprised all other genres. This resulted in a dataset with 3660 tracks in the minority class and 84340 in the majority, representing 4% and 96% of the dataset, respectively. Our personal goal is to develop a model that, given a track with an unknown genre, can classify it as a "J-song" if it belongs to one of the specified Japanese genres, with the highest possible precision. We anticipate scenarios where large volumes of data are added to Spotify databases, with possible corrupted or incorrect genre labels. Accurate genre classification, specifically for Japanese songs, is essential as an auxiliary task for Recommendation Systems applications such as Smart Shuffle, which caters to the niche audience of Japanese music fans. Therefore, our primary evaluation metric is precision, prioritizing quality over quantity. Ensuring that every track labelled as a "J-song" is indeed Japanese, even if it means the model may miss many such tracks. This priority stems from established domain knowledge, emphasizing that Recommendation Systems must maintain high precision to avoid losing credibility and reliability. Given the complexity of the task, we optimized our approach by selecting 15 key features and discarding those deemed unnecessary for classification. Subsequently, we standardized these features and split our dataset into training (70%, consisting of 61600 records) and testing (30%, consisting of 26400 records) sets. This split was used consistently throughout the process, resampling X_train and y_train as needed. The trained models were tested using the initial X_test to ensure robustness and comparability. Before applying any undersampling or oversampling techniques, we classified the records using a trivial decision tree without hyperparameter tuning. As expected, class 0 was predicted quite well, while the values for class 1 were very low. The precision in predicting class 1 was 0,42, the recall 0,38 and the F1-score 0,40. Then, we tuned the model to improve its predictions. However, while the AUC increased from 0.707 in the first model, to 0.730 in the second model, precision, recall, and F1-score all worsened. Investigating this behaviour, we discovered that ROC curves and AUC are less sensitive to class imbalance compared to classification metrics like precision and recall. This means that even if the classifier performs poorly on the minority class, this might not be evident from the AUC. This observation is relevant not only in this primal context but also in the scenarios discussed in the following paragraphs. Essentially, a higher AUC for the minority class does not necessarily indicate better classification performance for the minority class, especially in a highly imbalanced context.

3.2.2 Undersampling

Initially, we addressed the imbalance problem in our training set by employing different undersampling techniques, then we compared the performance of each classifier, trained with these resampled data, against those of the baseline approach mentioned earlier. The first technique employed to mitigate the effects of imbalanced data involved random sampling, the most popular undersampling strategy. This resampling resulted in a training set of 5168 records, with 2584 belonging to class 0 and 2584 to class 1. We trained two decision trees with this reduced training set: one without hyperparameter tuning and one using a random search for optimal parameters. The results were almost identical. The model without hyperparameter tuning returned a precision of 0.10, a recall of 0.71, and an F1-score of 0.17 for the minority class. The relatively high recall and poor precision indicate that while the model can recognize records belonging to class 1, it generates many false positives. The tuned model only slightly differed in recall, with a value of 0.70 instead of 0.71 and achieved a higher AUC of 0.783 compared to 0.742 for the non-tuned decision tree. Both AUCs are superior to the one generated by the decision tree trained on unbalanced data, which had a value of 0.707. The second strategy we actuated was Cluster Centroid (CC) undersampling, which led to 2548 records for each class. We tuned and trained a decision tree as before and the model achieved the highest recall in the undersampling setting, 0.83, but the lowest precision, 0.06 and the F1-score was 0.11. The AUC was 0.713, close to the value of the decision tree trained on unbalanced data. Finally, we used Condensed Nearest Neighbor (CNN), a more advanced undersampling strategy. This method aims to retain the samples most informative for classification, typically those near the decision boundary. By focusing on these critical samples, CNN often provides better performance by ensuring the reduced dataset still represents the original decision boundaries well. Indeed, the decision tree tuned and trained on these data reached the highest AUC, 0.787. However, in terms of classification, neither this method combined with a properly tuned decision tree returned satisfactory results, with a precision of 0.13, a recall of

0.51, and an F1-score of 0.23. Up to now, CNN has been the most effective strategy among those we attempted. However, the decision tree trained on unbalanced data remains one of the top performers.

3.2.3 Oversampling

In this section, we explored various techniques of oversampling to address class imbalance within our dataset. To ensure reproducibility and comparability between different techniques, we utilized a specific random_state parameter during data processing. The dataset we employed has been resampled and exhibits class 0: 59016, class 1: 59016. For each undersampling and oversampling technique, we conducted Principal Component Analysis (PCA) with 7 principal components, to examine their impact on the data. When applying ROS with a Decision Tree (DT) classifier without parameter tuning, the results were unsatisfactory. In fact, they were worse than our unsampled classification with DT across all metrics. The precision, recall, and F1-score of the minority class were lower, and even the area under the curve (AUC) was diminished. Even after parameter tuning, the performance of the DT classifier did not improve significantly, prompting us to explore other techniques. SMOTE, when coupled with a Decision Tree classifier, only marginally enhanced the performance of the base unsampled classifier, primarily in terms of recall and AUC of the minority class. However, as our task's main focus did not prioritize these metrics, this improvement was not deemed significant. Upon hyperparameter tuning of the DT classifier with SMOTE, there was a further increase in the recall of the minority class, reaching 0.52. While this represented an improvement, it still fell short of our expectations. Similar to SMOTE, ADASYN demonstrated modest improvements over the base Decision Tree classifier. While it achieved a better recall and higher AUC than ROS and SMOTE, the precision and F1-score of the minority class were inferior to the base Decision Tree. Despite ADASYN yielding the best results among the oversampling techniques tested, the performance was still not satisfactory for our task, as illustrated in Table 3.1.

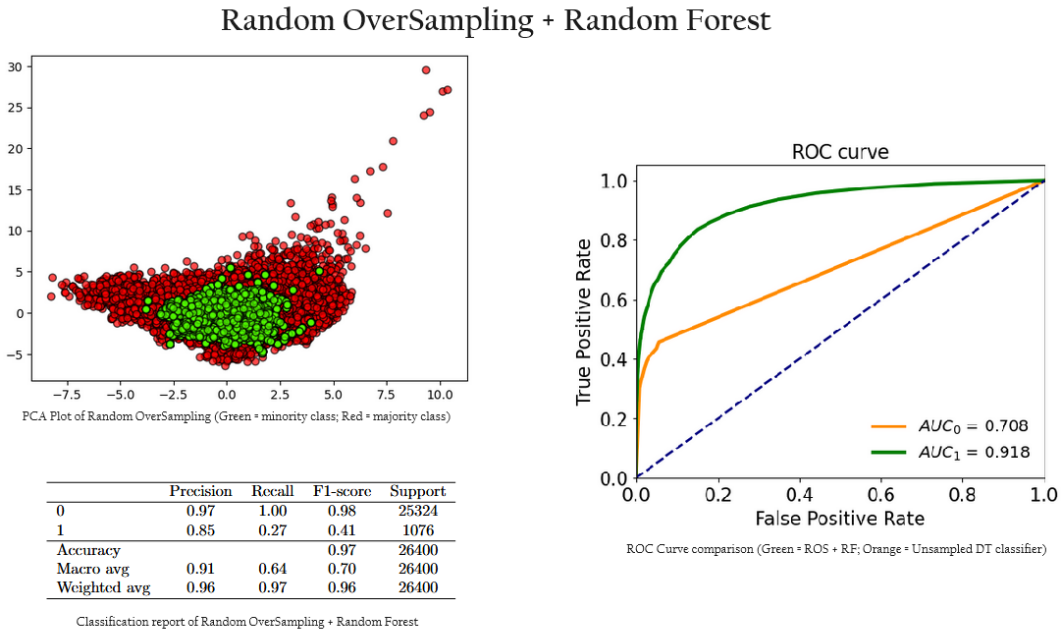
Metric	**Class 0**	**Class 1**	**Overall**
Precision	0.98	0.19	-
Recall	0.91	0.52	-
F1-score	0.94	0.28	-
Support	25324	1076	26400
Accuracy: 0.8911			

Table 3.1: Adasyn - Classification Metrics

Overall, our experiments with oversampling techniques underscored the challenge of effectively addressing class imbalance in our dataset. Further exploration and potentially alternative methods may be necessary to achieve the desired classification performance.

3.2.4 Advanced Methods Exploration

In our quest for results tailored to our specific task requirements, we delved into more sophisticated techniques. Instead of relying on individual models like Decision Trees, which come with inherent limitations, we opted for an ensemble approach. Specifically, we turned to BalancedRandomForest (BRF), a variant of Random Forest that distinguishes itself by drawing a bootstrap sample from the minority class and sampling with replacement the same number of samples from the majority class. Initially, when we applied BRF without balancing data with external techniques, we achieved an unprecedented recall level for the minority class, reaching 0.72. However, this boost in recall did not translate into an improvement in precision. Recognizing this discrepancy, we decided to explore combining Random Forest with oversampling techniques. Given that oversampling essentially nullifies the balancing mechanism of BRF, Subsequently, we conducted experiments with both SMOTE and ADASYN in conjunction with RF. The results yielded good performance, with SMOTE edging out slightly as the better performer, with notable improvements in the precision and F1-score of the minority class, reaching 0.49 and 0.48, respectively, while achieving an outstanding AUC of 0.907. Despite these advancements, we remained unsatisfied, as the results, though decent, fell short of our goal of attaining a high precision level. In pursuit of further enhancements, we explored various approaches, including utilizing alternative ensemble classifiers such as Easy Ensemble, but without significant improvements. After several iterations, we turned to Random Oversampling with Random Forest. The outcomes proved satisfactory for our task, as depicted in the figure 3.6



	Precision	Recall	F1-score	Support
0	0.97	1.00	0.98	25324
1	0.85	0.27	0.41	1076
Accuracy			0.97	26400
Macro avg	0.91	0.64	0.70	26400
Weighted avg	0.96	0.97	0.96	26400

Classification report of Random OverSampling + Random Forest

Figure 3.6: Random OverSampling + Random Forest (Summary)

Despite the relatively low recall metric for the minority class observed in the graph, the precision remarkably stands at 0.85, effectively solving our problem. While it's true that many Japanese tracks may remain undetected and categorized into the majority class, any track labeled as "j-song" is highly likely to be a True Positive, and that is what we were interested in. Furthermore, the AUC attained its peak value, indicating a strong performance at 0.916, much higher than the Decision Tree classifier trained on the unsample data and all the other methods investigated so far.

Finally, we explored hybrid combinations, consisting of both Oversampling on the minority class and Undersampling on the majority class, such as SMOTEENN (SMOTE + Edited Nearest Neighbor), RandomOverSampler + NearMiss, and ROS + ENN, trying different "sampling_strategy" configurations. While some of these combinations yielded balanced values across minority class metrics, hovering around 0.5 in precision, recall, and F1-score, they did not align with our specific task requirements. However, it's worth noting that these combinations would have been quite effective if our task did not rely so heavily on precision.

3.2.5 Conclusions

In our study on handling imbalanced learning for binary classification, we aimed to achieve high precision in identifying Japanese genre tracks. The initial trials with the decision trees trained on unsampled data showed modest performance, given the unbalanced setting. Undersampling techniques did not lead to a significant improvement in precision. Even the model trained on the undersampled data using a CNN only achieved a precision of 0.13, although the recall was not too low. Despite our efforts to train the best possible classifier, the classification results were worse than those obtained without resampling the training set. This may be attributed to the limited size of the training set, which was insufficient to effectively train a robust model. We then tried oversampling techniques like Random OverSampling (ROS), SMOTE, and ADASYN, with ADASYN showing the best results but still not meeting our precision requirements. Ensemble methods, particularly BalancedRandomForest (BRF), improved recall but not precision. The most promising results came from combining ROS with Random Forest, achieving a precision of 0.85 for the minority class and an AUC of 0.916. This combination ensured that any track labelled as a "J-song" was highly likely to be a True Positive, aligning with our primary objective. Hybrid methods like ROS + ENN and SMOTEENN provided balanced metrics which are quite satisfactory for a generic task but didn't meet our precision needs. Thus, ROS with Random Forest was chosen as the optimal solution for its high precision and overall performance reliability.

Chapter 4

Advanced Classification and Regression

4.1 Binary classification task: "explicit" vs "not explicit" tracks

4.1.1 Introduction

In this section, we detail our methodology and results for a binary classification task aimed at predicting whether tracks are "explicit" or "non-explicit". This dataset exhibits a significant class imbalance, with the majority class, "non-explicit" (class 0), comprising 80420 records, approximately 91.4% of the total dataset, and the minority class, "explicit" (class 1), comprising the remaining 7580 records, or about 8.6%. Our primary models for this task are Logistic Regression and Support Vector Machines (SVM). Prior to training our models, we pre-processed the data by selecting the same features as described in Chapter 3 and standardizing their values. The dataset was then split into training (70%) and testing (30%) sets.

4.1.2 Logistic Regression

The initial Logistic Regression model was trained with the default solver 'lbfgs' and the parameter max_iter set to 1000. As expected, the classifier showed high performance for class 0 (majority class) but performed poorly for class 1 (minority class), as shown in Test 1 of Table 4.1. To improve the model, we one-hot encoded

Test	Balanced	Encoded Genre	Decision Threshold	Class	Precision	Recall	F1	Supp	Accuracy
1	No	No	NaN	0	0.92	0.99	0.96	24164	0.92
				1	0.56	0.10	0.17	2236	
2	No	Yes	NaN	0	0.93	0.99	0.96	24164	0.92
				1	0.65	0.23	0.34	2236	
3	Yes	Yes	NaN	0	0.98	0.78	0.87	24164	0.78
				1	0.26	0.84	0.40	2236	
4	Yes	Yes	0.72	0	0.96	0.91	0.93	24164	0.88
				1	0.38	0.62	0.48	2236	
5	SMOTE-Tomek	Yes	0.78	0	0.96	0.93	0.94	24164	0.90
				1	0.43	0.54	0.48	2236	

Table 4.1: Models Performances under Different Configurations

the 'cluster_labels' (section 1.7) and 'genre' features and included them in the feature set. This step improved the model's output, as evidenced by Test 2 in Table 4.1. Next, we attempted to oversample the training set using Random Oversampling, but the results were unsatisfactory. We then set the class_weight parameter to 'balanced', which automatically adjusts the weights of the classes to account for the imbalance. This adjustment improved recall but further decreased the already low precision, as reported in Test 3. Our goal for this task was to balance precision and recall, so we focused on the F1 score. To improve this last classification, we tuned another parameter, the decision threshold, which increased the F1 score from 0.40 to 0.48, as shown in Test

4. We continued our experiments by applying hybrid oversampling techniques, specifically SMOTE-Tomek and SMOTEENN. These techniques combine oversampling (SMOTE) with undersampling methods (Tomek links for SMOTE-Tomek or Edited Nearest Neighbors for SMOTEENN) to improve class balance. The combination of SMOTE-Tomek, the tuned Logistic Regression model, and the adjusted decision threshold provided the best balance with an F1 score of 0.48, as shown in Test 5 of Tab 4.1. Contrarily, the results obtained with SMOTEENN were worse than our previous best. Similarly, an attempt to map 'genre' and 'cluster_labels' instead of one-hot encoding them yielded inferior results.

4.1.3 Support Vector Machine (SVM)

We began with LinearSVM and tested different configurations of the regularization parameter C to find the optimal model setup. Notably, C values of 1 and 0.001 showed promise by delivering satisfactory results for the majority class but fell short for the minority class. To address this imbalance, we employed Random OverSampling and conducted a Random Search for the C parameter. This search converged around C = 0.16, resulting in slightly improved performance compared to earlier attempts. However, achieving an F1-score of 0.34 for the minority class still did not meet our expectations, particularly when compared to our previous use of Logistic Regression. Consequently, we proceeded to delve deeper by exploring Non-Linear SVM. Utilizing Grid Search, we sought to determine the optimal parameter configuration, focusing particularly on C values and the choice of kernel (including γ for RBF). Specifically, the evaluation process involved testing 12 different candidate configurations through 5-fold cross-validation, which resulted in a total of 60 model fits. The optimal parameters identified were C=100, the Radial Basis Function (RBF) kernel, and γ='auto'. The performance metrics for this configuration demonstrated an overall accuracy of 0.92. The model excelled in predicting Class 0, with a precision of 0.93, a recall of 0.99, and an F1-score of 0.96. However, for Class 1 (explicit songs), the precision was 0.60, the recall was 0.20, and the F1-score was 0.30. Despite making a final attempt to address the imbalance using Random OverSampling on the minority class and testing various combinations of kernels and C parameters, the results remained unsatisfactory. The highest recorded F1-score for the minority class was only 0.35. Comparing these results to those obtained with Logistic Regression, the latter achieved a more balanced performance in the minority class, with higher recall and a more acceptable F1-score, despite SVM maximizing the precision metric. Given our objective to achieve balanced performance across both classes, the Logistic Regression model is the preferable choice.

4.2 Multi-class classification task: five-class classification

In this section, we transitioned to simple multi-class classification task. This phase aimed to predict five distinct classes ("Low_insLow_en", "Low_insMid_en", "Low_insHigh_en", "High_insLow_en", "High_insHigh_en") derived from our previous clustering analysis, detailed thoroughly in Section 1.7. These clusters categorized music genres based on instrumentalness and energy levels.

For this multi-class classification task, Logistic Regression was chosen as the primary model due to its effective performance in the preceding binary classification task. The model was trained using a fixed parameter setting, specifically 'max_iterations = 1000'. Upon evaluation, the Logistic Regression model exhibited exceptional performance on the test set, achieving an accuracy of 1.00. Precision scores were 1.00 for three classes and 0.99 for the remaining two, with recall rates similarly high, scoring 0.99 for two classes and 1.00 for the other three. These results underscored the model's robust predictive capability across all classes, leading us to conclude this phase without further exploration of alternative models.

4.3 Complex multi-class classification task: 113 genres classification

4.3.1 Introduction

In this section, we embark on a highly complex classification task aimed at predicting the genre of music tracks across 113 distinct categories. The preprocessing steps applied here follow the same methodology introduced in subsection 4.1.1. Given the intricacies involved in classifying across such a large number of classes, our approach begins with exploring traditional machine learning algorithms. Specifically, we initially employ Logistic Regression and Support Vector Machines (SVM) to establish a baseline performance. Following this, we will delve into Ensemble-based methods to potentially enhance predictive accuracy by leveraging the strength of multiple

models. Finally, recognizing the potential of Neural Networks in capturing intricate patterns within data, we will develop and fine-tune a Neural Network tailored specifically for this classification task. This sequential approach aims to progressively refine our models, ultimately striving to achieve optimal performance in accurately classifying music tracks into their respective genres. Each stage of the experimentation will be meticulously evaluated and compared to identify the most effective model for this challenging multi-class classification problem.

4.3.2 Logistic Regression

We began our investigation using Logistic Regression, without hyperparameter tuning, the model achieved an accuracy of 0.23 for genre prediction, varying in effectiveness across different genres. Establishing this as a baseline, we proceeded with a Random Search to optimize key parameters: regularization parameter C, solver type, and maximum iterations. The Random Search identified C equal to 7.320, "newton-cg" as the optimal solver, and set max iterations to 100. The model's performance plateaued at an accuracy of 0.25 on train, validation and testing sets, suggesting that Logistic Regression might not be suitable for this task. As a final effort, we conducted a Grid Search, yielding a configuration similar to the Random Search and the accuracy remaining at 0.25 across all three sets.

4.3.3 Gradient Boosting Machine

We tested various Gradient Boosting Machine (GBM) algorithms for predicting 113 genre-related classes. The variants we used included Extreme Gradient Boosting (XGBoost), Light Gradient Boosting Machine (LGBM), Histogram-based Gradient Boosting, and CatBoost. While we also considered the original Gradient Boosting Machine, its training time was significantly longer compared to these variants, limiting the number of configurations we could test effectively. Additionally, XGBM offers more parameters for tuning and is recommended for solving complex tasks. All of them are ensemble methods based on boosting, meaning that predictions are generated sequentially rather than in parallel, as each prediction influences subsequent ones. The first model that we implemented was XGB Machine and in this section, we explore the most important parameters for this model and how we tuned them. First, we focused on the learning rate, which controls the contribution of each tree to the final model, and the number of estimators. A convention is to maintain a small learning rate and a large number of estimators because this generally yields more accurate models. Secondly, we adjusted the maximum depth of each base learner, as deeper decision trees can capture more intricate patterns but also increase the risk of overfitting. The first XGBM we trained has a learning rate of 0.1, 100 estimators, a maximum depth of 6, and an L2 reg equal to 1. The resulting model was too simple, with a training accuracy of 0.83 and a test accuracy of 0.56. To better understand the parameters, we tuned the model manually, we increased the maximum depth to 10 and the number of estimators to 500. As predicted, we ended up with overfitting, achieving a training accuracy of 0.998 and a test accuracy of 0.56. After a RandomizedSearch, we arrived at our best configuration with 1000 estimators, max depth 5, minimum child weight 4, learning rate 0.076, alpha reg 2 (L1 reg, adds a penalty for the absolute value of coefficients, helping to enforce sparsity), tree method 'exact'(the most precise algorithm), gamma 0 (allowing the algorithm to create partitions freely), and early stopping of 15 rounds (it stops training if the model doesn't improve on the validation set for 15 consecutive rounds). This configuration yielded a validation set loss function of 1.7274, a training accuracy of 0.8156 and a validation and test accuracy of 0.56, evidently the model generated overfitting and despite all the regularization that we performed the configuration above resulted in the most balanced. In Picture 4.1 we reported the values for the first for the best 9 classes and the last 2 sorted by F1.

Figure 4.1: Classification Results of Final XGB Model (11 out of 113)

Genre	Precision	Recall	F1-score	Support
sleep	0.92	0.98	0.95	48
study	0.90	0.95	0.93	59
iranian	0.83	0.93	0.87	41
honky-tonk	0.79	0.95	0.86	59
comedy	0.78	0.90	0.83	
children	0.78	0.85	0.81	59
grindcore	0.76	0.86	0.81	58
romance	0.75	0.86	0.80	49
dance	0.75	0.86	0.80	49
...	...	...	...	...
punk	0.25	0.08	0.12	13
metal	0.14	0.07	0.10	14

HGB, LGBM and CatBoost

The second-best model we used was the Histogram Gradient Boosting classifier. We adapted parameters from the previous model to fit those required by HGB. The loss function on the validation set was 1.667, the training accuracy was 0.949, the validation accuracy was 0.59 and the test accuracy was 0.58. While this performance was slightly worse than XGB, the training process was significantly faster. We concluded testing LGBM and CatBoost to check if they could outperform the previous models. As LGBM was very fast we used RandomizedSearch and the optimal values were: L2 reg 1, L1 reg 0.44, 20 leaves per tree, 150 estimators, a subsample for bin of 2000, a max depth of 7, a learning rate of 0.3, and boosting type 'dart'. This configuration yielded a training accuracy of 0.92 and a test accuracy of 0.56. We couldn't prevent overfitting in any of the models, we could only mitigate the training accuracy, making the model simpler and reaching an accuracy between 0.90 and 0.94, but every time the performances on the test set underperformed those on the training. Finally, knowing that the CatBoost is designed not only for handling more accurately categorical features but also for mitigating situations where overfitting is complex to avoid, we trained and tested it but with scarce results, indeed it was aligned with all the other models with a training accuracy of 0.93 and a test accuracy of 0.56. To determine whether the challenges we faced with the initial models were due to the task's complexity or suboptimal configurations, we trained an XGBoost model to classify the five classes identified from our clustering analysis. By manually tuning a few configurations, we achieved remarkable results: a training accuracy of 1.00, a test accuracy of 0.99, a precision of 0.99 for each predicted class, and a loss function value of 0.036 on the validation set. The same results have been obtained varying the size of the three sets and slightly changing parameters. This behaviour suggests that our difficulties with the original 113 classes problem were more likely due to the inherent complexity of the task.

4.3.4 Deep Neural Networks

In our exploration of Neural Networks, we followed a preprocessing approach similar to earlier sections. Initially, we utilized 15 standardized features and split the dataset into training (70%) and test (30%) sets. Furthermore, the test set was further split into validation (80%) and test (20%) subsets. This division aimed to provide ample training examples crucial for Deep Neural Networks, in particular when the task is to classify 113 classes with 88000 records. Our first neural network model featured two hidden layers with 128 and 64 neurons, using Rectified Linear Unit (ReLU) activations for transitions between layers and softmax for the output layer. Despite these settings, the model achieved only a validation accuracy of 0.22 and a training accuracy of 0.32, indicating inadequate performance. To enhance performance, we explored various regularization techniques such as early stopping, L2 regularization, and Dropout. Despite these efforts, subsequent models achieved modest improvements with maximum validation accuracies of 0.23, 0.25, and 0.26, respectively, but with high loss values exceeding 4. Recognizing the need for further refinement, we turned to more systematic approaches. Due to computational constraints, we initially applied a Randomized Search on a representative 10% sample of the dataset, validated by Kolmogorov-Smirnov tests to ensure its representativeness. However, parameters derived from this search did not generalize well to the entire dataset, yielding results similar to previous models.

This led us to a hands-on approach, where we designed and tested 30 custom models. These custom models varied in architecture and hyperparameters, focusing on optimizing dropout rates, layer configurations (2 to 4 layers with neuron counts ranging from 512 to 64), learning rates (0.01, 0.001, 0.0005), batch sizes (32 and 64), and activation functions (primarily ReLU and Tanh). We also experimented with L2 regularization (finding 0.0001 effective) and early stopping with different patience levels, all while using the Adam optimizer. After approximately 10 custom models, the best-performing model achieved a training accuracy of 0.23 and a validation accuracy of 0.22. This indicated that while the models were not overfitting, they were not effectively learning complex patterns from the data. This insight prompted a shift in strategy towards balancing regularization with learning capacity more effectively.

Our subsequent efforts focused on expanding model complexity to capture intricate patterns. Initially, we explored larger input sizes and varying numbers of layers. We found a learning rate of 0.001 optimal compared to 0.0005, which showed slower convergence. Batch sizes of 32 proved better despite potential noise, acting as a regularization mechanism, particularly as models became more intricate.

One standout model, "CustomModel23", surpassed previous benchmarks despite slight overfitting. It featured three hidden layers with Tanh activation in the first layer for data centering and complex pattern capture, followed by dropout layers (dropout probability 0.1) and ReLU activations. The layer sizes were 256, 128, and 128, respectively, with other parameters including a learning rate of 0.001, batch size of 32, and early stopping patience set to 20 epochs. "CustomModel23" achieved a training accuracy of 0.52 and a validation

accuracy of 0.41 after 200 epochs, demonstrating robust learning despite overfitting. Attempts to mitigate overfitting by increasing dropout probability to 0.3 led to inferior results, prompting adjustments. Subsequently, "CustomModel25" emerged with increased complexity, featuring four layers and Leaky ReLU (with default "negative_slope" parameter) activations across all layers, often better than traditional ReLU for detecting complex patterns. This model included some dropout layers (dropout probability 0.3) strategically placed to prevent overfitting. The layer sizes were 1024, 512, 256, and 128, respectively, with a learning rate of 0.001, weight decay of 0.0001 (compensating for a few removed dropout layers), and early stopping patience set to 10 epochs. While "CustomModel25" achieved a higher training accuracy of 0.53, it exhibited increased overfitting compared to "CustomModel23", with a validation accuracy of 0.41 and higher validation loss. Further refinements, including additional dropouts, layer adjustments, and hyperparameter tuning, led us to our final model.

The final model represented a culmination of effective strategies, carefully balancing regularization and performance. It featured four hidden layers with a dropout probability of 0.2, all using Leaky ReLU activations. The layer sizes were 1024, 512, 256, and 128, progressively reducing to aid convergence in deep networks. Key hyperparameters remained consistent with successful configurations: a learning rate of 0.001, 200 epochs with early stopping patience set to 10 epochs, batch size of 32, and L2 weight decay set to 0.0001, all managed by the Adam optimizer. This model achieved the highest validation accuracy of 0.42, with a training accuracy of 0.50, suggesting minimal overfitting, as shown in Figure 4.2. Analyzing the classification report, in Table 4.2, it's evident that the model excels in certain genres like "comedy" with precision, recall, and F1-score all around 0.90, indicating robust classification performance. Genres such as "honky-tonk" and "iranian" also show high precision and recall scores, reflecting the model's ability to accurately predict these categories. However, some genres like "alt-rock", "dubstep", "punk" and "edm" have lower scores across precision, recall, and F1-score, suggesting challenges in accurately classifying these genres compared to others. The results on the test set confirmed consistent performance, although they did not surpass those of Gradient Boosting methods like XGBoost, known for their effectiveness with tabular data. The challenges posed by limited data availability for both training and testing were evident, impacting the neural network models' optimization.

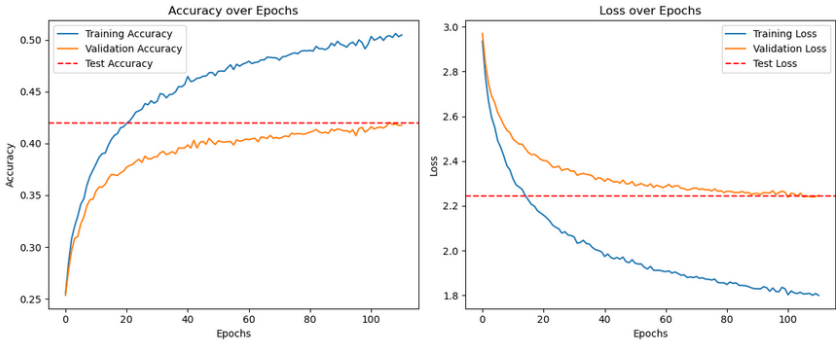


Figure 4.2: Accuracy and Loss over Epochs in Final DNN Model

Table 4.2: Classification Results of Final DNN Model (12 out of 113)

Genre	Precision	Recall	F1-score	Support
comedy	0.9333	0.8750	0.9032	48
honky-tonk	0.8136	0.8136	0.8136	59
iranian	0.7609	0.8537	0.8046	41
romance	0.7193	0.8367	0.7736	49
sleep	0.6885	0.8750	0.7706	48
study	0.6543	0.8983	0.7571	59
grindcore	0.7213	0.7586	0.7395	58
jazz	0.5862	0.7907	0.6733	43
opera	0.5606	0.7551	0.6435	49
...	...	...	...	...
dubstep	0.0000	0.0000	0.0000	16
punk	0.0000	0.0000	0.0000	13
edm	0.0000	0.0000	0.0000	9

4.3.5 Conclusions

In this section, we explored advanced classification tasks, evaluating binary and multi-class classification models. For binary classification (explicit vs. non-explicit tracks), Logistic Regression with SMOTE-Tomek oversampling achieved a balanced F1 score of 0.48, demonstrating effective handling of imbalanced data. In 5-class classification, Logistic Regression delivered strong precision and recall metrics across categories, proving its suitability for simpler multi-class scenarios. The complex multi-class classification, encompassing 113 genres, highlighted the effectiveness of ensemble methods like XGBoost and Histogram Gradient Boosting, achieving test accuracies of 0.56. Deep neural networks showed promise but faced challenges in achieving competitive accuracies (around 0.42), emphasising its need for a huge amount of data for improving model performance and the complexity of multi-label classification in music genres. Overall, ensemble methods excel in handling complex classification tasks with tabular data and numerous output classes, while Logistic Regression remains robust for simpler scenarios. In Figure 4.3, we present the AUC curves corresponding to the Micro-Average generated by each model, using One-Vs-Rest. Despite low accuracy in precise classification settings, all models demonstrate exceptional performance in distinguishing whether a track belongs to a specific class versus others, as reflected by their ROC curve, while the models can distinguish between classes, their ability to accurately predict specific genres remains limited.

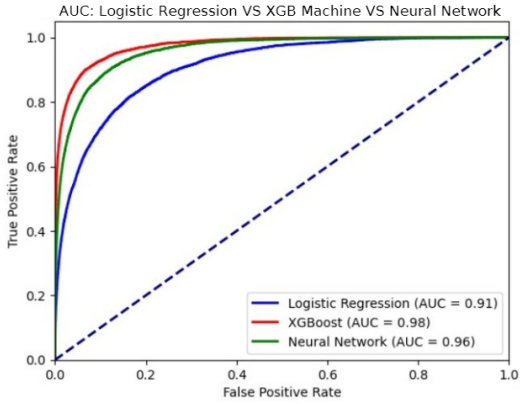


Figure 4.3: All the models recognize if the track belongs to the current class or to the other 112 (OVR) but not the precise one

4.4 Advanced Regression

4.4.1 Random Forest Regressor and XGB Regressor

The task we defined for this section is the same one we attempted to solve in DM1's project without great success: predicting a track's popularity.

This time, we had more powerful models, more data and more features, some of them, like "average_artist_popularity", "primary_artist_popularity" and "sum_of_followers", derived from merging the track's dataset with the artist's dataset. We pre-processed our dataset to select the most relevant features for the task, which are those reported in Figure 4.4. As expected, the most important feature was "average_artist_popularity," which often correlates positively with the track's popularity. To capture the impact of each feature, we performed a multiple regression with all of them using a Random Forest. We then checked for multicollinearity and identified it between "average_artist_popularity" and "primary_artist_popularity." Consequently, we removed the second as well as the two least important features from the set of independent variables and trained the Random Forest, with discrete success but also slight overfitting, as shown in Model 1 in Tab 4.3. We continued our investigation using only the top three most important features to train and validate our Random Forest Regressor

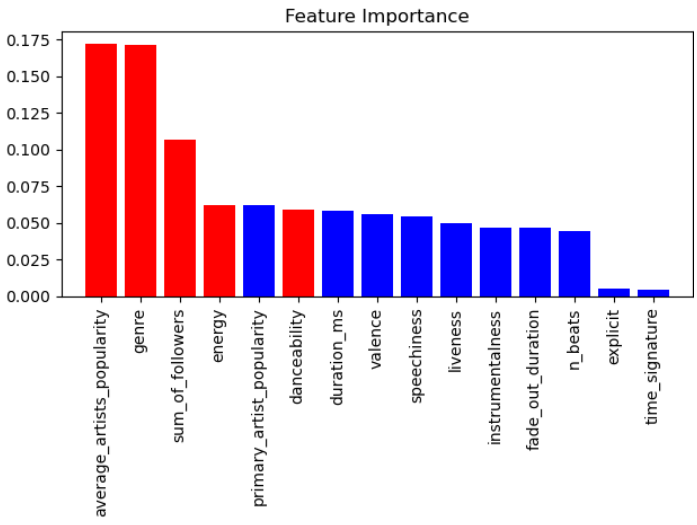


Figure 4.4: Features sorted from the most to the least important for the task. Red bars indicate those that return the best results.

and the result in terms of MSE and R^2 were slightly better than the previous model, as reported in Model 2 of Table 4.3. We then attempted to further tune the model but did not achieve satisfactory results. Hence, we opted to increase the number of independent features, avoiding those with a Variance Inflation Factor above 5 (i.e., high levels of multicollinearity). Our regression model's results improved as we added more features. After a trial-and-error phase, we settled on five features that exhibited no multicollinearity among them: "average_artist_popularity", "genre", "sum_of_followers", "energy" and "danceability". The resulting model, Model 3 in Tab 4.3, displayed the highest R^2 score among all trained models during validation, indicating its effectiveness in explaining and predicting the target variable. Furthermore, when evaluated on the test set, it maintained strong performance with an R^2 of 0.49. This suggests that the regression model excels in predictive accuracy and generalizes well to new, unseen data.

Table 4.3: Random Forest Regression: Results for Different Configurations

Model	**Features Utilized**	**MSE Train**	R^2 **Train**	**MSE Validation**	R^2 **Validation**
1	Twelve	122.4651	0.7313	239.0590	0.4802
2	Top-Three	179.3404	0.6068	230.1017	0.4994
3	Five	138.5636	0.6964	216.2979	0.5128
Model 3, the best on the Validation Set, on the Test Set shows an MSE of 233.681 and an R^2 of 0.4919.					

As a secondary model for regression, we tested the XGB Regressor, by paying close attention at the tuning of the hyperparameters we explained in subsection 4.3.3. Table 4.4 presents the results of the XGBoost model, trained using an increasing number of features, similar to our approach with the Random Forest regression. After extensive hyperparameter tuning, the configuration chosen for the best-perfoming model (Model 3 in Tab 4.4) included n_estimators = 100, learning_rate = 0.06, max_depth = 15, and min_child_weight = 1. However, the performance of this XGBoost model closely resembled that of the Random Forest regression. Both models reached a performance plateau, indicating that further improvements were limited by the dataset used. This suggests that, given our current data, both XGBoost and Random Forest have comparable predictive capabilities.

Table 4.4: XGB Regression: Results for Different Configurations

Model	**Features Utilized**	**MSE Train**	R^2 **Train**	**MSE Validation**	R^2 **Validation**
1	Twelve	63.1954	0.8613	235.5607	0.4878
2	Top Three	103.3019	0.7733	234.2175	0.4907
3	Five	156.6976	0.6562	225.7373	0.5091
Model 3, the best on the Validation Set, on the Test Set shows an MSE of 234.5145 and an R^2 of 0.4808.					

In conclusion, a positive aspect of this multiple regression task is the significant improvement in performance compared to our previous attempt in the DM1 Project. In that project, we achieved unsatisfactory results with a maximum R^2 of 0.109, largely due to limitations inherent in the model used at the time. However, in the current context, leveraging advanced regression models such as XGBoost and Random Forest, coupled with a larger dataset, has markedly enhanced our predictive accuracy and overall outcomes.

4.5 Explainability

In this section, our aim was to delve into the intricate workings of an Extreme Gradient Boosting Machine (XGB) model used for binary classification between "explicit" (1.0) and "non-explicit" (0.0) tracks. Initially, we considered using a Support Vector Machine (SVM), but its performance on the minority class was suboptimal, with an F1-score of 0.30. Consequently, we opted for XGB due to its superior performance in previous tasks. The dataset utilized for training and testing comprised features used initially with SVM, augmented by the inclusion of encoded "genre" information. We standardize the values and split the dataset into 80% training and

20% testing records. On the testing set, XGB demonstrated decent performance with an accuracy of 93% and an F1-score of 0.51 for the minority class. Then, we focused on building an explainability model to shed light on how XGB makes predictions on unseen data. Despite the limited dataset size, we still utilized the X_test

Feature Id	Importances
speechiness	15.35
genre	15.10
danceability	10.05
energy	8.38
valence	6.93
instrumentalness	6.04
duration_ms	5.71
average_artists_popularity	5.69
fade_out_duration	5.28
n_beats	5.27
liveness	4.33
sum_of_followers	4.06
popularity	4.03
primary_artist_popularity	3.26
time_signature	0.52

Figure 4.5: Feature Importances

dataset to ensure insights into generalization and model behavior. The CatBoostClassifier was selected for its ability to provide explainable outputs, especially after hyperparameter tuning (depth=5, learning_rate=0.3, iterations=500, l2_leaf_reg=6), achieving an impressive accuracy of 97%. However, some overfitting was noted due to the dataset's inherent imbalance, although the F1-score for the minority class remained adequate at 0.66.

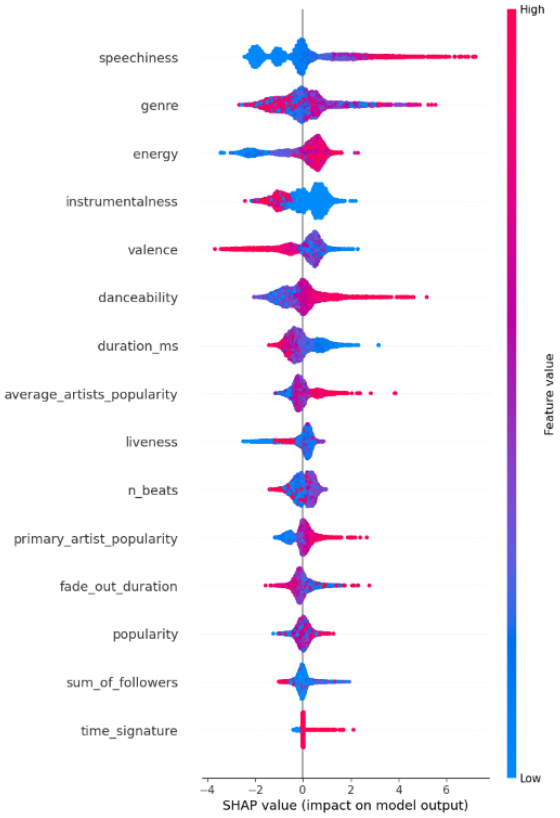


Figure 4.6: SHAP summary plot

A pivotal aspect of our analysis was the feature importance assessment, shown in Table 4.5, which revealed "speechiness" and "genre" as primary influencers in the classification task. Songs categorized as explicit typically exhibited higher levels of speechiness (reflecting explicit lyrics) and tended to belong to specific genres identifiable through encoding.

Further insights were obtained from SHAP (SHapley Additive exPlanations) plots, shown in Figure 4.6, elucidating how variations in feature values impacted predictions. Higher speechiness values consistently correlated with increased likelihoods of "explicit" classification. The feature "genre" also proved instrumental, reflecting genre-specific characteristics influencing classification outcomes. Notably, higher valence values, indicative of positive emotional tones, leaned towards "non-explicit" classifications, aligning with our assumptions. Exploring interactions among feature pairs highlighted speechiness and genre as the most intertwined, underscoring their joint influence on model predictions. This synergy reinforced their individual importance while emphasizing their combined impact on classification accuracy.

We then employed LIME (Local Interpretable Model-agnostic Explanations) to delve deeper into the XGB model's behavior. Analysis of instances, like Instance 16 in Figure 4.7, highlighted

scenarios where high danceability and speechiness increased the likelihood of "explicit" classification. Conversely, low instrumentalness and high valence tended to balance this effect towards "non-explicit" classification, although less prominently. However, LIME also highlighted the inherent limitations of local surrogate models in fully capturing the complexity of the original XGB model. In Instance 10, depicted in Figure 4.7, the XGB model predicted a high probability (0.90) of it being "non-explicit". However, when examining the local surrogate model (LIME), it emphasized that features such as danceability and speechiness significantly influenced predictions towards explicitness, whereas features like valence had a weaker impact. This discrepancy underscores the challenge in LIME's interpretation: while it effectively identified local effects such as valence and instrumentalness suggesting "non-explicit" classification, it struggled to reconcile why the overall model leaned towards "non-explicit" when features like speechiness and danceability were predominant in the decision-making process.

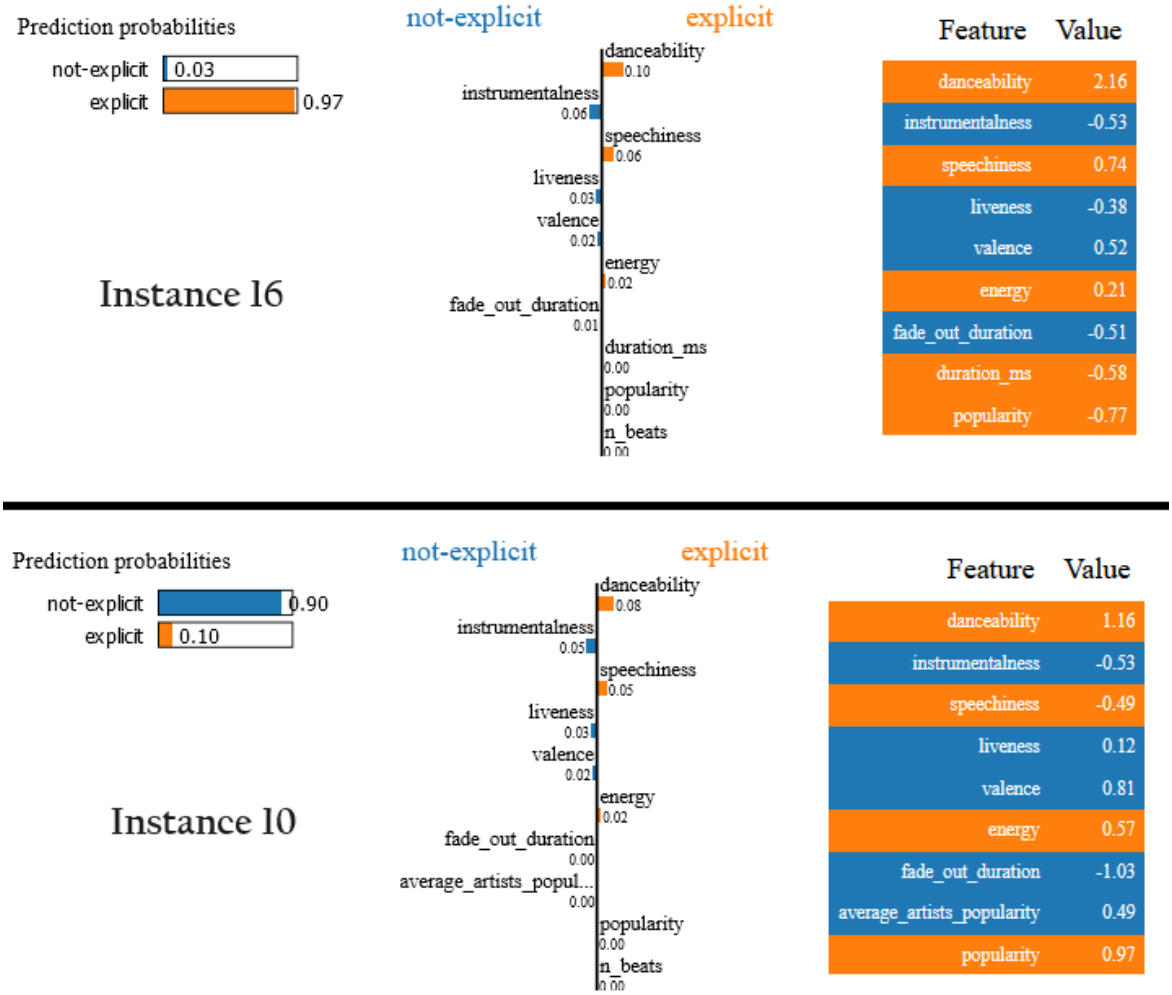


Figure 4.7: LIME Explainer (Instances 10 and 16)

In conclusion, our study utilized XGB for robust binary classification and CatBoostClassifier for explainability, revealing speechiness and genre as pivotal features in predicting explicitness. Insights from SHAP and LIME deepened our understanding of model behavior, highlighting challenges in fully interpreting complex machine learning models.