Data Mining 2

Corrado Baccheschi, Isabel Santucci, ~~Noemi Boni~~ :-)

1 Introduction

The following study aims to analyze, explore and use different kinds of advanced data mining techniques on two different types of datasets. A first tabular dataset in which there are data related to music tracks of artists and their related genre, while a second dataset contains 10000 time series rapresenting the spectral centroids of the song mp3 audio files of the genres.

This report is structured as follows.
Section 2.1 it's about the data understanding and preparation, here we discuss the data semantics and explain each attribute for the tabular dataset (Section 2.1 and Section 2.2), then the first analyses are delved: on the distributions of the variables (Section 2.2.1), checks of similar attributes and variable transformations (Section 2.2.2), assessment of data quality exhibying the possible presence of outliers and finally the correlations found and their relations each other. The last Sections 2.3 and 2.3.1 are about understanding time series dataset and the preprocessing steps performed on before clustering them, the latter which is explained in Section 3 .
Section 3 is about clustering on the time series dataset, it presents different algorithms used and the results obtained.
Section 4 is related on time series dataset, in particular about motif analysis, which may be useful for classification.
Section 5 explains the two classification tasks we decided to perform. One regard multiclassification using the time series dataset and the other binary classification on the tabular one.
Section 6 serves as an introduction for the next series of studies on the tabular dataset which include: Outliers detection (Section 7), Imbalanced learning (Section 8), Advanced Classification (Section 9) , Advanced Regression (Section 10) and Explainability (Section 11).

2 Data understanding and preparation

2.1 Data Semantics

The track dataset contains different types of attributes. Each track has its own *ID* which is useful for implementing links to time series. There is the name of the track (*name*), *artists* is the name of the artist(s) possibly separated by semicolon, the duration in milliseconds (*duration_ms*), the same information found in *features_duration_ms*. *Popularity* which specifies the popularity value between 0 and 100, meaning as a song played a lot in the past. The type of the album, *album_type*, where the type of track (album, single, compilation) is found and the name of the album (*album_name*), and the number of tracks in the album as *album_total_tracks*. The number of the track, *track_number*, the disc number, *disc_number*, the date the album was first released (*album_release_date*), *album_release_date_precision* the precision with which *release_date* value is known (year, month, day). If the track has explicit lyrics then it has value True for Explicit, False otherwise. In addition, there is more technical information about the track such as *energy*, a measure ranging from 0.0 to 1.0, *danceability* with a value from 0.0 to 1.0 that describes how suitable a song is for dancing based on a combination of musical elements including tempo, rhythm stability, beat strength, and overall regularity. *Key* values, which are whole numbers and correspond to the pitches in standard Pitch Class notation. *Mode* which indicates the mode (major or minor) of a song. *Speechiness* detects the presence of spoken words in a song; values above 0.66 describe songs that are likely to consist entirely of spoken words, whereas values between 0.33 and 0.66 describe songs that may contain both music and speech, and values below 0.33 most likely represent music and other nonspeech-like songs. *Acousticness*, a confidence measure from 0.0 to

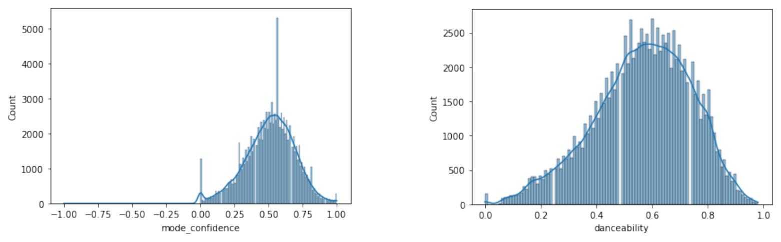
1.0 of whether the track is acoustic. *Instrumentalness* that predicts whether a track contains no vocals and the closer the instrumentalness value is to 1.0. *Liveness* detects the presence of an audience in the recording and values above 0.8 provides strong likelihood that the track is live. *Valence* measured from 0.0 to 1.0 describing the musical positiveness conveyed by a track. *Loudness* specifies the overall loudness of a track in decibels (dB). There are also have information about the time of the tracks with the attribute *tempo*, the estimated total time of a track in beats per minute (BPM), *time_signature* an estimated time signature. *N_beats* is the total number of beat time intervals throughout the track and *n_bars* is the total number of beat time intervals throughout the track. *Start_of_fade_out* indicates the time, in seconds, at which the fade-out period of the track begins. *Processing* is the major/minor mode ration aggregated by *key* and *genre* to which the track belongs. Finally, there are different confidence measures for some attributes: *popularity_confidence*, *time_confidence*, *time_signature_confidence*, *key_confidence*, and *mode_confidence*.

2.2 Tabular dataset

The tabular dataset is obtained by merging two datasets. One dataset containing information about the artists' tracks and another dataset related to additional information about the artists including popularity, number of followers, and a list with all genres they belong to. In the artist dataset there are some artists present more than once with different popularity value, so to handle this problem all the different popularity values for the artist were considered and the maximum value is taken as the reference value for popularity. In contrast, for artists with only one value for popularity, the single value is taken as the reference. On the other hand if the artist present in the tracks dataset is not in the artists dataset it is assigned 0 as the popularity value. In the tracks dataset there are tracks made by individual artists, but also by various collaborations. Therefore, the additional column that reports popularity, called *popularity_artist*, is obtained by making a list that contains a single popularity value if the track was made by a single artist, while you have a list with several popularity values if several collaborations are made, and the order of the popularity values in the list is reported according to the order found in the column where the artists' names are separated by semicolons.

2.2.1 Distribution of the variables and statistics

We first selected and tested the most relevant variables for the presence of normal distributions[1] in the continuous variables. While we didn't find any, we discovered some attributes that are close to meeting this criterion. In the next figure, you can see the histograms for the *mode confidence* and *danceability* attributes, which are the closest to a normal distribution.



(a) Histogram for *mode_confidence* attribute.

(b) Histogram for *danceability* attribute.

Figure 1: Distribution of *mode_confidence* and *danceability* shown by histograms. Note the blue line that limits and draws the curves.

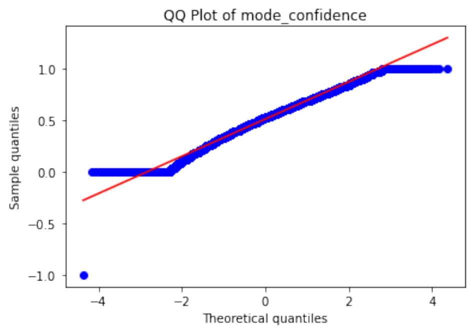
The distributions shown in the plots 1 are *not* normal, as indicated by the blue dashed line. The *mode confidence* has a skewness[2] of -0.33 and kurtosis[3] of 0.17, while the *danceability* has a skewness of -0.40

[1]In statistics, a normal distribution or *Gaussian* distribution is a type of continuous probability distribution for a real-valued random variable.

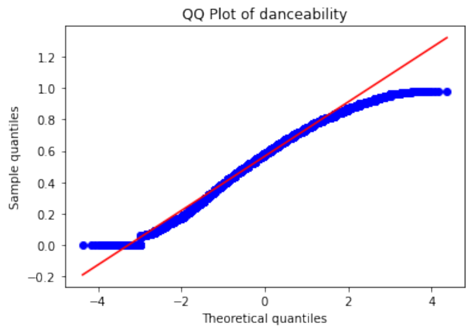
[2]The skewness value measures the asymmetry of the probability distribution of a real-valued random variable about its mean. For a unimodal distribution (a distribution with a single peak), negative skew commonly indicates that the tail is on the left side of the distribution, and positive skew indicates that the tail is on the right.

[3]The Kurtosis value measures how many values lies around the mean. A curve with a high value of Kurtosis shows

with kurtosis of -0.18. In order to test the hypothesis of normality, the Shapiro test was not used due to potential inaccuracies with large data sets (N >5000). Instead, the Kolmogorov-Smirnov test[4] is applied, which is not affected by this issue. The resulting p values[5] were 0.0, using a *threshold* of 0.05, leading to the rejection of the normality hypothesis. Additionally, the QQPlot is used to identify the variable closest to a normal distribution, following the rule that if the blue line closely follows the red line, the data is likely to be normally distributed.



(a) QQPlot for *mode confidence* attribute.



(b) QQPlot for *danceability* attribute.

Figure 2: *mode confidence* and *danceability* on a QQPlot. It's easy to see that the blue line of *mode confidence* (on the left) is one that follows better the red line.

At this level of analysis, outliers are still present, as can be seen in Figure 2 (the separate lines or points at the bottom). The distribution may become closer to a normal distribution after removing these outliers. For more deep discussion on outlier detection and their dealings, see the Section 6.

2.2.2 Variable transformations

The first analysis is performed on the tracks dataset checking for similar attributes. Through cosine similarity it is possible to identify considerable similarity between *features_duration* and *duration_ms*, *n_bars* and *n_beats*, *start_of_fade_out* and *duration_ms*, pairs of attributes with a similarity of about 99%. This made it possible to eliminate one attribute per pair by considering only *duration_ms*, and *n_bars*. Since there are some attributes in integer and float format, in order to standardize the data as best as possible, it was decided to convert the attributes to float format where possible. These include *duration_ms*, converted from milliseconds to seconds, *disc_number*, *popularity*, *track_number*, *album_total_tracks*, *key* and *time_signature*. While *mode* is left in integer format since it can be considered as a categorical attribute because it has possible two outcomes (0 and 1). Next, all float values in the dataset are approximated to two values after the comma.

2.2.3 Assessing data quality

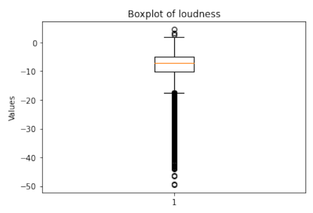
When analyzing the initial data, it is important to identify outliers, which are values that are significantly different from the majority of the data. Outliers can affect the analysis and cause data quality issues. To identify outliers, box plots can be used to visually display the distribution of data for continuous variables.

Figure 3 shows the graphs depicting the distribution of the variables *loudness*, *n_bars* and *valence*. These graphs help us easily understand the location of numerous outliers in relation to the majority of data.

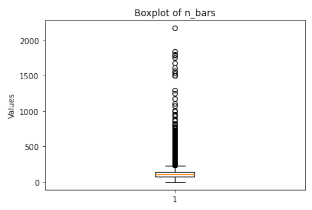
a peak around the mean and a steep nearby the tails, while with a low value the shape of the peak is more flat and the decrease around the tails is slower. Like the skewness, the Kurtosis value is normalized on 0.

[4]The Kolmogorov-Smirnov is a statistical test, which was developed by Andrey Kolmogorov and Nikolai Smirnov. It is used to determine whether a sample comes from a particular probability distribution.

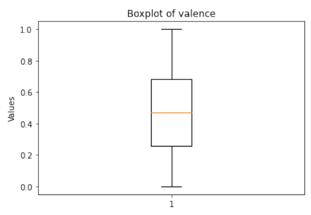
[5]In statistic, the p-value provides a way to assess whether the observed data supports the null hypothesis ($p > \alpha$) or contradicts it in favor of the alternative hypothesis ($p \leq \alpha$)



(a) Box plot for loudness



(b) Box plot for n_bars



(c) Box plot for valence

Figure 3: Box plots made with seaborn for the attributes *loudness*, *n_bars* and *valence*.

From these plots (Figure 3), one can see that only *valence* does not exhibit outliers in its distribution, whereas *loudness* primarily shows outliers at the lower end of the box, and *n_bars* predominantly has outliers at the upper end. Unsurprisingly, the distribution of *valence* demonstrates a very low standard deviation of 0.25, while *n_bars*, for instance, shows a considerable deviation from its mean with a value of 59.

2.2.4 Pairwise correlations

Another important analysis to consider is the potential correlation between variables. It measures the linear relationship between two random variables and it is always between -1 and 1. A value of -1 indicates an inverse linear relationship, 1 indicates a direct linear relationship, and 0 indicates no linear trend between the two variables. The correlation can be easily visualized through a correlation matrix, which is a square and symmetrical matrix with all elements on the main diagonal equal to 1.

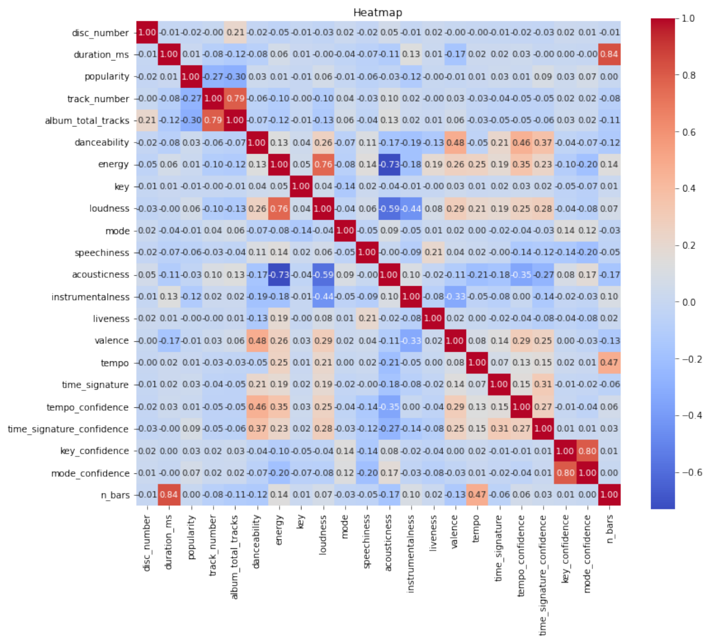


Figure 4: Heatmap made with the seaborn library. In each square is shown the numerical value of a correlation, and a colour that displays in a more rapid way how much the attributes are correlated: the more warm the more closer to 1, the more cool the more closer to -1.

Figure 4 shows a strong positive correlation of 0.84 between *duration_ms* and *n_bars*. There is also a positive correlation between *energy* and *loudness*, meaning that more loud songs are also more energetic. On the other hand, there are strong negative correlations between *energy* and *acousticness*, indicating that more acoustic songs are less energetic. Additionally, there is a positive correlation between *key_confidence* and *mode_confidence*. There is also a soft negative correlation between *loudness* and *acousticness*, meaning that acoustic songs tend to have lower loudness. All values in the squares

are calculated using the Pearson coefficient[6].
Furthermore, we analyze the relationship between *duration_ms* and *n_bars*, these are compared and plotted using the seaborn library, as shown in Figure 5.

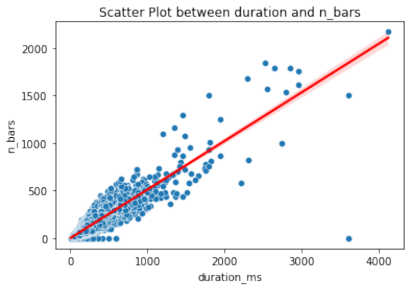


Figure 5: The obtained scatter-plot for *n_bars* and *duration_ms* attribute. Note the red line that is fitted with respect to the positive correlation between both such variables.

From the plot 5, it's possible to state that more number of intervals of beats a track contains, consequently more would be its duration in time. These correlations provided a starting point for more in-depth analyses on the time and signal domains.

2.3 Time Series dataset

In order to perform analyses in the time domain, there is also a dataset containing 10,000 time series in NumPy format. This dataset includes 500 time series for each of the 20 selected genres, providing information about the songs. Since the names of the files containing such time series are composed of a random alphanumeric string followed by the genre to which they belong, an algorithm is run to extract the specific genres. These genres include: 'sertanejo', 'folk', 'j-idol', 'world-music', 'songwriter', 'progressive-house', 'heavy-metal', 'mpb', 'opera', 'minimal-techno', 'new-age', 'sleep', 'emo', 'kids', 'honky-tonk', 'synth-pop', 'goth', 'happy', 'salsa', 'piano'. Each time series is long 1280 and represent the spectral centroids of the 10,000 songs mp3 audio files chosen. In the following sections, we will delve into the preprocessing steps, transformations, and approximations that have been applied to the time series data.

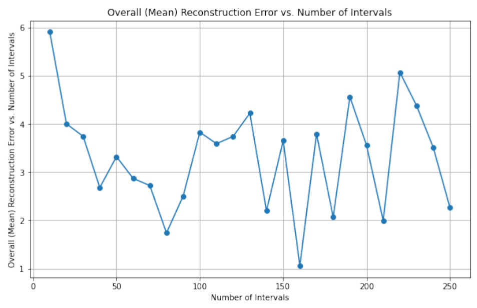
2.3.1 Time series transformations and approximations

Firstly, our focus was on cleaning the data to eliminate any inconsistencies or anomalies that might impact the analysis. This involved reducing noise, and normalizing the data to maintain consistency across various time series. We initiated the process by applying amplitude scaling to all time series. This consists in scaling each series substrating from each one its mean and dividing by its standard deviation, facilitating also comparison between different features. Then, we removed noise applying a window rolling mean of 150, obtaining more smoothed time series. After that, since the time series were too long and their computational cost was too high, the Piecewise Aggregate Approximation (PAA)[7] algorithm is run to approximate them. PAA reduces the dimensionality of the input time series by splitting them into equal-sized segments (or number of intervals, or frames) and in order to determine the optimal value for them, we reconstructed the time series by applying the inverse PAA trasformation, then assessed the similarity between the original-no-approximated and reconstructed

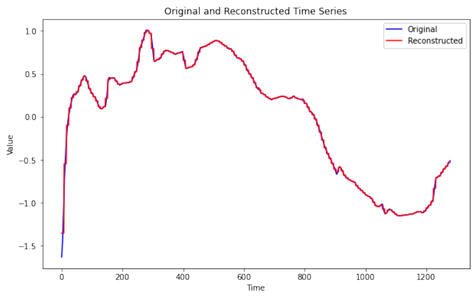
[6]In statistic, a coefficient that evaluates the amount of the (linear) correlation. In particular, closer -1 means negative correlation whereas nearer to +1 means positive correlation. 0 indicates absence of linear relationship (but there may be a *non*-linear one)

[7]The Piercewise Aggregate Approximation, called PAA, is a technique consisting in taking the mean over back-to-back points. This decreases the number of points and reduces noise while preserving the trend of the time series.

versions using the euclidean distance metric. 1.06 was the minimum distance obtained with 160 as the best value, Figure 6 depicts the plot which drove us to the right value (a) and the approximated-and-reconstructed time series, overlapped on the original one (b).



(a) The progress of the overall (mean) euclidean error with respect to different values of number of intervals. One can observe that the lowest error is reached with 160 obtaining 1.06 as euclidean error



(b) The original and the approximated-and-reconstructed time series, in red the reconstructed version and in blue the original one.

Figure 6: The progress of the overall (mean) euclidean error with respect to different values of number of intervals (on the left) and the comparison between the original and the approximated-and-reconstructed time series (on the right)

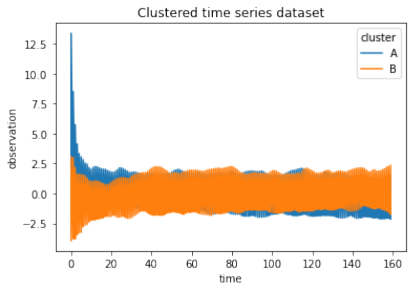
Once we preprocessed and approximated time series, everything is ready for the clustering explained in the next Section 3.

3 Clustering of the time series dataset

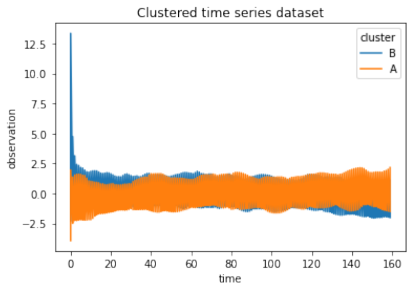
We performed clustering with TimeSeriesKmeans and DBSCAN algorithms on our time series dataset to detect possible clusters in order to inspect them. We weren't able to perform hierarchical clustering due the high computational resources required, hence we tried using 100 time series but since such number was too small to draw conclusions, we chose to not report it. Since we approximated the time series using PAA trasformations (see Section 2.3.1) to avoid too high computational costs, it was necessary to use another type of distance different from the euclidean, that is Dynamic Time Warping (DTW).[8] However, since DTW is computationally very expensive, we opted to utilize a sample from the 10,000 time series, running an algorithm to extract 100 time series for each of the 20 genres, resulting in a total of 2,000 time series. For the same reason, we implemented itakura[9] max slope of 1.0. The 2000 time series appeared us as a highly representative number, as the discrepancy with respect to the total 10,000 clustered time series did not appear substantial, as represented in Figure 7 on a simple example with k=2. We from now refer to this representative number of time series approximated with PAA as "reduced dataset" whereas as "full dataset" for the original one (with 10000 time series approximated).

[8]Dynamic Time Warping (DTW) is a time series analysis technique used for measuring similarity between two temporal sequence. Basically, the difference with the euclidean distance is that while the former takes into account time shifts, the latter requires both the time series are in sync.

[9]The Itakura parallelogram constraint defines a region in the time warping space as a parallelogram, based on a spectral representation of the sequences. This constraint limits the possible alignments to those within this region, significantly reducing the number of alignments that need to be considered.



(a) Clustering of full dataset of time series



(b) Clustering of reduced dataset of time series

Figure 7: Series of plots of a toy example to show the differences between clustering (with k=2) comparing TimeSeriesKMeans using 10,000 time series (on the left) or 2,000 (on the right).

Notably, the clustering depicted in Figure 7 (a) took only 3 minutes whereas for (b) took the half that is 1 minute. Although these values appear not very high, one should note that it depends a lot of on value of k, for which, if equal to 3 we observed a substantial increment for the (a) case to more than two hours, leading to a task computationally expensive.

3.1 TimeSeries KMeans with Dynamic Time Warping (DTW)

In order to perform a more representative clustering using the reduced dataset, we selected the k value (=number of clusters) on the basis of the elbow method evaluating the inertia, which represents the sum of squared distances of samples to their closest cluster center. Figure 8 depicts the method we adopted.

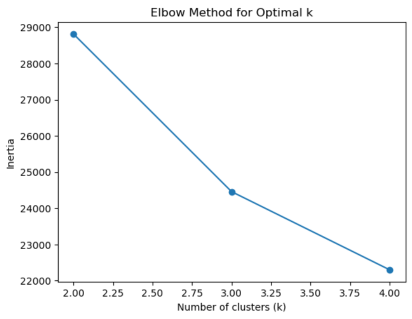


Figure 8: The progress of the inertia. Note the blue point in the middle which indicates the best trade-off between k and inertia.

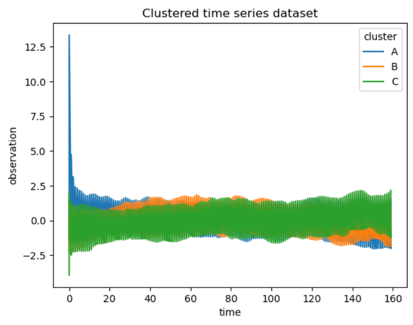


Figure 9: Clustering of the reduced dataset of time series fixing k=3 and using DTW distance. A, B, C are the clusters found.

According to the plot, the best number of clusters i.e the k value is 3, cause it's the value which presents lower inertia. Consequently, we performed clustering with k=3 and the results of this clustering are illustrated in the Figure 9.
Figure 9 above doesn't show substantial differences if compared to Figure 7 (b). The algorithm finds another cluster which, however, basically overlaps the orange and blue ones found before. One can observe the difficulties for the centroids to keep each other distant, we could infer this is a consequence of the dataset containing similar values.

3.2 Density-based clustering with DBSCAN

The second type of clustering is performed on the features [10] of the time series extracted using SummaryTransformer()[11] from the Sktime library. Before running DBSCAN on the reduced dataset containing 2000 time series, it is necessary to choose the best epsilon by calculating the average DTW distance implemented with itakura max slope of 2.0, through the library pyts, between each point, taking into account its 4 nearest neighbors. The point of maximum slope (Figure 10), the 'knee', indicates the best epsilon to use, in our case 0.5. The algorithm is then run with a minimum number of points equal to 5.

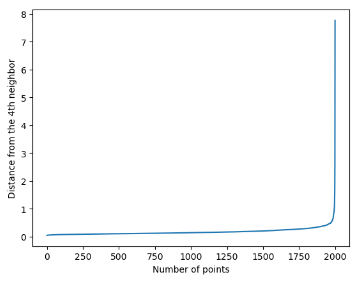


Figure 10: The average DTW distance for 4 nearest neighbors

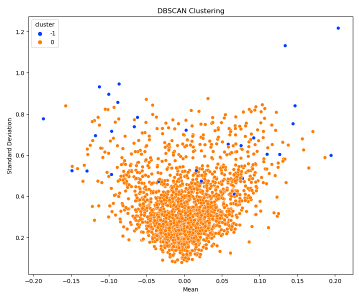


Figure 11: DBSCAN Clustering on the reduced Dataset using Standard Deviation and Mean

In Figure 11, a single cluster consisting of core (orange points) and noise points (blue ones) can be observed.

3.3 Dimensionality Reduction(PCA, TSNE)

Once we obtained clusters in previous Sections 3.1 and 3.2 through TimeSeriesKMeans and DBSCAN, we moved on inspect them through different dimensionality reduction techniques.

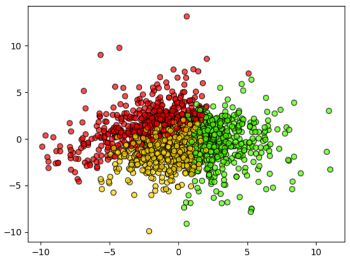


Figure 12: PCA for clustering KMeans

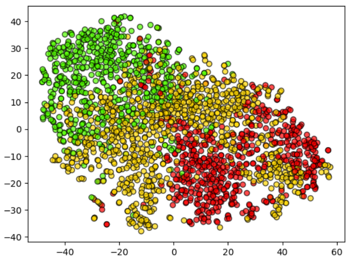


Figure 13: t-SNE for clustering KMeans

[10]Summary statistics of the time series, such as mean, standard deviation, maximum, minimum and quantiles
[11]Calculate summary value of a time series

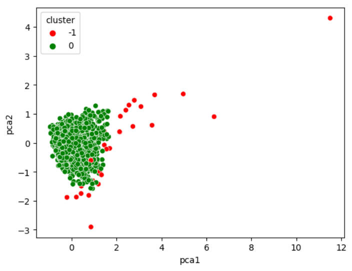


Figure 14: PCA for DBSCAN clustering. In red, the noise points.

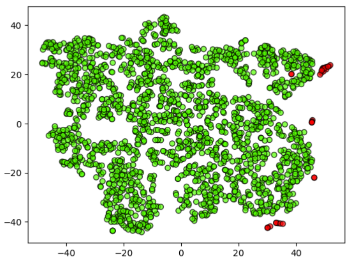


Figure 15: t-SNE for DBSCAN clustering

Looking at the different clusters obtained by two dimensionality reduction techniques such as PCA [12] and t-SNE [13], one can see significant differences. In Figure 12 in which PCA was applied for clustering achieved with KMeans using DTW as distance, 3 distinctly separate clusters can be seen, but when going to see local information with t-SNE (Figure 13) it can be seen instead that there is no clear separation between the clusters, but points of the different clusters that tend to blend together. Since t-SNE is much more precise than PCA this allows for more information about the internal distrubution of clusters and possibly highlight particular inconsistencies that could not be observed at a first glance of clusters with PCA. Whereas in the case of DBSCAN, both Figures 14, 15 tend to look similar. This is due to the fact that you only have one cluster compared to KMeans, and also DBSCAN can handle the noise points better and this is also reflected in the different dimensionality reduction techniques.

3.4 Clustering evaluation

After understanding the different clustering algorithms, it is important to identify the best one. It is possible to compare the Silhouette coefficient, which measures how well the clusters are separated and the best value is one that approaches 1.

Clustering algorithm	Silhouette Coefficient
DBSCAN	0.70
TimeSeriesK-Means	0.08

Table 1: Comparison of clustering algorithms using the silhouette coefficient

In Table 1, the varying values of the Silhouette coefficient are presented in relation to the various clustering algorithms implemented. Based on the values, it can be stated that, for our data, DBSCAN is the best algorithm for data division, as it has the highest value among the 2, while K-Means algorithm is the worst one.
Note that, since DBSCAN identified a single cluster, the silhouette calculation is particularly irrelevant in this context, cause there is only one cluster. Moreover, it's important to note that the dataset contains outliers which particularly affects the results in the case of K-Means. On the other hand, DBSCAN which can deal with outliers, showed that the dataset contains similar values since it recognizes only one cluster.

[12] Principal Component Analiysis (PCA) allows for dimensionality reduction by selecting only a subset of the principal components. Typically, the top k eigenvectors associated with the largest eigenvalues are retained, while the rest are discarded. This reduces the dimensionality of the data from the original number of features to k, where k is typically much smaller than the original dimensionality.

[13] t-SNE stands for t-Distributed Stochastic Neighbor Embedding. It firstly computes pairwise similarities between data points in the high-dimensional space then aims to find a low-dimensional representation (usually 2D or 3D) of the data points that preserves the pairwise similarities as much as possible. It achieves this by defining a similar probability distribution in the lower-dimensional space and minimizing the Kullback-Leibler divergence (KL divergence) between the two distributions.

4 Motif analysis on the time series dataset

For the analysis related to motif extraction, we again used the reduced dataset (details in Section 3). A time window of length 5 is considered for extraction and a maximum of 3 motifs per time series is taken. Once the motifs are obtained in the form of arrays, for reasons of computational efficiency, we decided to take only the first ones, corresponding to the first element of the array of each time series, since the further values turn out to be completely similar to each other. To arrive at discriminating the most significant motifs, clustering is performed on the obtained motifs, choosing the correct k via the elbow method, obtaining a k equal to 3.

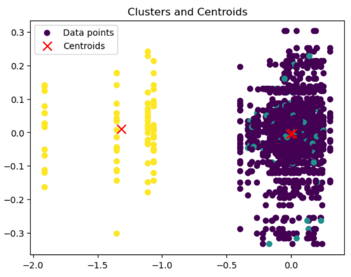


Figure 16: Clusters and centroids obtained with K-Means on the 2000 time series motifs

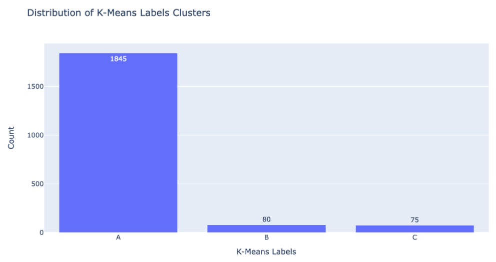


Figure 17: The label frequency of the different clusters extracted from the motifs

As can be seen in Figure 16, there is an overlap between 2 clusters while the remaining cluster differs significantly from the others and is sparsely populated. To further investigate the amount of elements in the clusters, a bar chart is made in relation to frequency.
From the Figure 17 above, it can be stated that cluster A is the densely populated one compared to clusters B and C whose quantity differs by 5 elements.

To identify possible discriminatory motifs for a given genre, the presence of each genre in each cluster is analyzed.

Cluster A	
Genre	Frequency
Progressive-house	97
New-age	96
Salsa	95
Goth	94
Songwriter	94
Happy	94
Folk	93
Opera	92
Synth-pop	92
J-idol	92
Piano	92
Minimal-techno	91
World-music	91
Heavy-metal	91
Emo	91
Honky-tonk	91
Kids	90
Sertanejo	90
Mpb	90
Sleep	89

Table 2: Genres Frequencies in cluster A

Cluster B	
Genre	Frequency
Heavy-metal	8
Opera	7
Emo	5
Goth	5
Sleep	5
J-idol	5
Synth-pop	5
Sertanejo	4
Minimal-techno	4
World-music	4
Piano	4
Folk	4
Mpb	4
Honky-tonk	4
Happy	3
Salsa	3
Songwriter	2
Kids	2
Progressive-house	1
New-age	1

Table 3: Genres Frequencies in cluster B

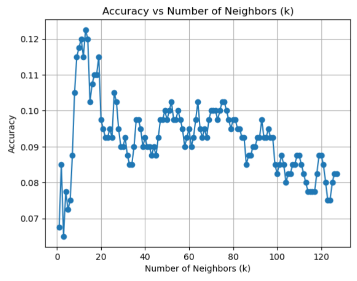
Cluster C	
Genre	Frequency
Kids	8
Mpb	6
Sertanejo	6
Sleep	6
Minimal-techno	5
World-music	5
Honky-tonk	5
Piano	4
Emo	4
Songwriter	4
Folk	3
J-idol	3
Happy	3
Synth-pop	3
New-age	3
Progressive-house	2
Salsa	2
Heavy-metal	1
Goth	1
Opera	1

Table 4: Genres Frequencies in cluster C

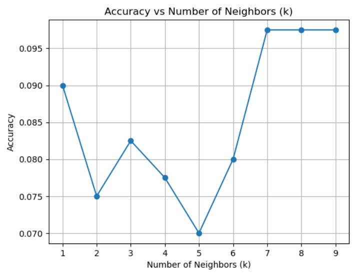
It can be seen that genres are present in each cluster in a balanced manner, not allowing for any detailed identification as to which genre might differ most from others. The only information this analysis provides us with is any similarity between the genres, not allowing any discrimination between them. For example, it is inferred that progressive-house is most similar to new-age, minimal-techno to world-music, as they recur close together in all three clusters. Thus, compared to an initial analysis, it was not possible to discriminate one motif of one genre from another.

5 Classification

We used the reduced dataset to perform a classification task on their 20 genres, using KNeighborsTimeSeriesClassifier (KN-TS). In order to compare the results on the basis of the difference distances used (DTW or euclidean) we took the same 2000 times series of the reduced dataset but, in case of euclidean, not approximated. We refer to them as "reduced no-approximated dataset". Moreover, again using KN-TS we defined another task on the full dataset about the classification of *mode* attribute belonging to the tabular dataset, and which is discussed in the Section 5.2. In Section 5.3 Deep Learning (DL) models are also employed for the multilclassification task on the 20 genres.
Firstly, we searched for the best trade off between k (number of neighbors) and accuracy obtained in the euclidean case and in the DTW case.



(a) Accuracy wrt. number of neighbors in range (1,128) using euclidean distance on the reduced no-approximated dataset.



(b) Accuracy wrt. number of neighbors in range (1,9) using DTW distance on the reduced dataset.

Figure 18: Accuracy wrt. number of neighbors. One can note that the best k is equal to 13 in the euclidean case (on the left). Note that for the higher computational cost in DTW case (on the right), we reduced to a range of 1,9 and the best k was equal to 7.

Hence, classification is performed for both cases.

	Precision	**Recall**	**F1-score**	**Support**
0	0.09	0.20	0.12	20
1	0.14	0.20	0.16	20
2	0.15	0.20	0.17	20
3	0.05	0.05	0.05	20
4	0.05	0.05	0.05	20
5	0.00	0.00	0.00	20
6	0.13	0.15	0.14	20
7	0.13	0.30	0.18	20
8	0.15	0.45	0.23	20
9	0.13	0.10	0.11	20
10	0.17	0.05	0.08	20
11	0.33	0.05	0.09	20
12	0.50	0.10	0.17	20
13	0.20	0.15	0.17	20
14	0.09	0.15	0.11	20
15	0.00	0.00	0.00	20
16	0.50	0.05	0.09	20
17	0.15	0.10	0.12	20
18	0.00	0.00	0.00	20
19	0.50	0.10	0.17	20
Accuracy			0.12	400
Macro avg	0.17	0.12	0.11	400
Weighted avg	0.17	0.12	0.11	400

Table 5: KN-TS 20 genres classification report with euclidean distance and k=13

	Precision	**Recall**	**F1-score**	**Support**
0	0.08	0.15	0.11	20
1	0.04	0.10	0.06	20
2	0.00	0.00	0.00	20
3	0.13	0.10	0.11	20
4	0.00	0.00	0.00	20
5	0.05	0.05	0.05	20
6	0.16	0.20	0.18	20
7	0.11	0.15	0.12	20
8	0.25	0.30	0.27	20
9	0.08	0.05	0.06	20
10	0.10	0.05	0.07	20
11	0.50	0.15	0.23	20
12	0.17	0.10	0.12	20
13	0.08	0.05	0.06	20
14	0.06	0.05	0.05	20
15	0.05	0.05	0.05	20
16	0.27	0.15	0.19	20
17	0.10	0.10	0.10	20
18	0.12	0.10	0.11	20
19	0.12	0.05	0.07	20
Accuracy			0.10	400
Macro avg	0.12	0.10	0.10	400
Weighted avg	0.12	0.10	0.10	400

Table 6: KN-TS 20 genres classification report with DTW and k=7

In both cases the accuracy is particularly low. In the case of Table 5 the classifier performs markedly worse, since for some genres it has performance of 0, as in the case of genre 5, 15 and 18. Whereas DTW (Table 6) still has better performance, recognizing most genres unlike genre 2 and 4. This could all be due to the fact that there are genres that are very similar to each other, as seen before in motif identification (see Section 4), so it might be difficult for a model to distinguish between them.

5.1 Multi classification task and shapelets analysis

We obtained the shapelets of the reduced dataset using LearningShapelets algorithm of tslearn library. Since the cost was computationally expensive, we restricted the algorithm to find only 150 shapelets and each one of length 20. With the shapelets retrieved, we then performed Cross-Validation (CV)[14] with *k=5* using grid search on the following hyperparameters in Table 7.

Hyperparameters	**Values**
criterion	[gini, entropy]
max_depth	[10, 20]
min_samples_split	[10, 15, 20]
min_samples_leaf	[2, 4, 6]

Table 7: Parameter Grid for the Grid search

At the end of the process, best values found were 'criterion': 'gini', 'max_depth': 10, 'min_samples_leaf': 4, 'min_samples_split': 20, which we used to train our DecisionTree to classify the 20 genres using the shapelets extracted by LearningShapelets algorithm. We obtained the classification performance showed in the following report in Table 8.

Class	**Precision**	**Recall**	**F1-score**	**Support**
0	0.24	0.25	0.24	20
1	0.08	0.05	0.06	20
2	0.00	0.00	0.00	20
3	0.09	0.10	0.09	20
4	0.40	0.10	0.16	20
5	0.10	0.10	0.10	20
6	0.04	0.05	0.04	20
7	0.11	0.25	0.15	20
8	0.32	0.40	0.36	20
9	0.09	0.05	0.06	20
10	0.17	0.15	0.16	20
11	0.00	0.00	0.00	20
12	0.20	0.10	0.13	20
13	0.21	0.15	0.18	20
14	0.06	0.10	0.07	20
15	0.00	0.00	0.00	20
16	0.38	0.30	0.33	20
17	0.00	0.00	0.00	20
18	0.10	0.25	0.14	20
19	0.00	0.00	0.00	20
Accuracy			0.12	400
Macro avg	0.13	0.12	0.11	400
Weighted avg	0.13	0.12	0.11	400

Table 8: Classification Report

From this report one can visualize the better performance of the tree on class (genre) 8 and class 4, which present 0.32 of precision for the former and 0.40 for the latter. Notice that, for the recall the best classified is the class 8 even if we compare it with all the other classes' metrics and values.[15]

[14]Cross-validation (CV) is a technique used to assess the performance of a model on unseen data. It's a method for estimating how well a model will generalize to new data by partitioning the available data into *k* subsets, training the model on some of these subsets, and evaluating it on the remaining subset(s).

[15]We tried also to use a ShapeletTransformer (sktime library) keeping its default hyperparameters and it took us 20 hours to run, although the result wasn't bad showing an accuracy of 0.19. Even in this case the most representative genres were class 8 and class 4.

That's not true for the precision, since class 16 presents an higher value. We identified the genre of class 8 as "minimal-techno", class 4 as "heavy-metal" and class 16 as "sleep". We will delve deeper on the former and the latter since they are strongly opposite each other respect the type of songs which contain. To thoroughly examine them, we analyzed the shapelets obtained through the LearningShapelets algorithm. We then computed the Euclidean distance between the average of the time series for each genre and all the shapelets. This process helped us identify the shapelet that has the shortest distance from a specific genre, thus being as the most representative shapelet for that genre. Comparing the shapelets of a given musical genre with its extracted motifs can provide important insight into the distinctive characteristics of that genre's tracks and the recurring motifs that characterize it. In relation to the results obtained, this type of analysis can be particularly relevant on genres on which the classifier performs better, as in the case of Minimal-techno and Sleep, which at the musical level also exhibit contrasting characteristics.

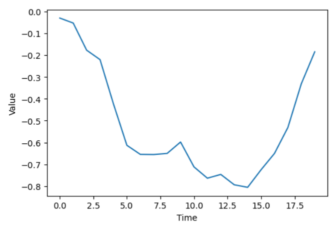


Figure 19: Sleep Shapelet at minimum distance 1.26

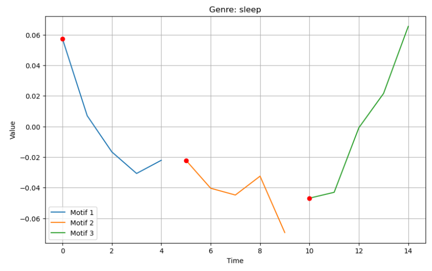


Figure 20: Sleep Motifs

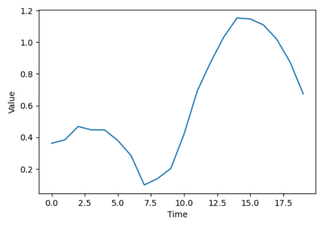


Figure 21: Minimal-techno Shapelet at minimum distance 1.33

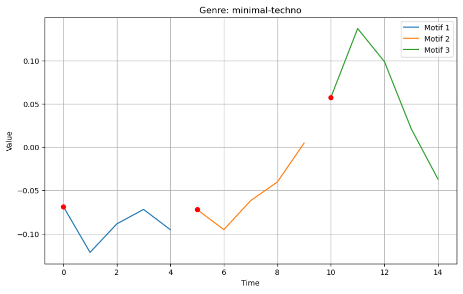


Figure 22: Minimal-techno Motifs

Figures 19, 20, 21, 22 provide further confirmation that the extracted shapelets are of the associated genre due the motifs seem separated parts of each shapelet. It is possible to show a significant difference between the two genres from the very shape of the shapelets. This can be distinctly significant in identifying Time Series belonging to that particular genre. It is believed that this was possible because the two genres are musically different, and this allows for their discrimination. If in the previous analysis concerning the extraction of motifs and the realization of clusters did not allow any kind of relevant identification among them, this type of analysis related to the extraction of shapelets allowed the identification of relevant motifs of the genre.

5.2 Binary classification task

As previously discussed (see Section 5), a binary classification task is also defined on the *mode* attribute (see Section 2.1 for more details on it and others attibutes) and in this case it was decided to consider the full dataset i.e all 10000 PAA-approximated time series in case of DTW, whereas the full no-approximated dataset for the euclidean one. Target class labels are obtained by doing a merge with the track dataset via the time series ids, and successively taking all values of *mode* attribute. A standard subdivision of both datasets is used, keeping 80% percent of the data for the training and the remaining 20% for the testing.

In the euclidean and DTW cases to identify the optimal neighbor, we searched for the best number of neighbors using a range of values of k (1,128) for the euclidean case, and (1,15) for the DTW one.

	Precision	**Recall**	**F1-score**	**Support**
0	1.00	0.00	0.01	635
1	0.68	1.00	0.81	1365
Accuracy			0.68	2000
Macro avg	0.84	0.50	0.41	2000
Weighted avg	0.78	0.68	0.56	2000

Table 9: KN-TS classification report for mode with unbalanced training set and best k=114, in the euclidean case

The Confusion Matrix in Table 9 shows inconsistencies in the recognition of the two classes, as it obtained a very good precision for class 0 but a recall of 0, which indicates that the model correctly classifies a sample as belonging to class 0, however fails to identify any of the other instances of such class. Whereas in the case of class 1, there is 68% precision and 100% recall, so most of the positive predictions made by the model are correct, and the maximum recall value indicates that it is able to identify all the instances. These somewhat inconsistent results are due to an imbalance between the data in the training data: 5461 in the case of class 1 and 2539 in the case of class 0, reason why the model performed better on class 1 than on class 0. To solve this issue, it was decided to adopt the Condensed Nearest Neighbor (Condensed NN) technique[16] obtaining 2539 examples for class 0 and 2433 examples for class 1.

	Precision	**Recall**	**F1-score**	**Support**
0	0.33	0.51	0.40	635
1	0.69	0.51	0.59	1365
Accuracy			0.51	2000
Macro avg	0.51	0.51	0.49	2000
Weighted avg	0.58	0.51	0.53	2000

Table 10: KN-TS classification report for mode with balanced training set and best k=1, in the euclidean case

The Confusion Matrix in Table 10 presents more balanced data and there are more congruent results, despite a decrease in accuracy of 51%. The model correctly predicts one-third of the instances in the case of class 0, compared to the Table 9, but there is a marked improvement for recall as 51%. Consequently there is also an improvement in the case of the F1-score.
Below, in Table 11, classification is carried out considering the DTW as a distance metric.

	Precision	**Recall**	**F1-score**	**Support**
0	0.36	0.14	0.20	635
1	0.69	0.89	0.78	1365
Accuracy			0.65	2000
Macro avg	0.53	0.51	0.49	2000
Weighted avg	0.59	0.65	0.59	2000

Table 11: KN-TS classification report for mode with unbalanced training set, DTW distance and best k=13

Despite the unbalance present, the classifier nevertheless has better performance than the Table 10 considered with euclidean distance. However, the model still has better results in the case of the

[16]Condensed Nearest Neighbor (Condensed NN) is a type of instance selection technique used to reduce the size of a dataset by selecting a representative subset of instances. The idea is to retain only the most informative instances while discarding redundant or noisy ones.

majority class and relatively low results for minority class. In relation to the results obtained, it is possible to state that DTW has better classification performance, this is due to the fact that it takes into account temporal variations in the time series, resulting in a more robust distance metric. In addition, DTW is able to adapt to complex patterns and this can lead to improvements in classification performance. Note that in the balanced case after Condensed NN is applied, the KN-TS classifier with DTW distance has showed no improvements in accuracy, therefore we didn't report it.

5.3 Multiclassification task using Conv.NeuralNetwork in Keras

We also employed a Convolutional Neural Network (CNN) in Keras to classify the 20 genres. For this task, we utilized 70% of the dataset for training (TR), 10% for validation (VL), and reserved 20% for testing. Owing to the high computational costs, we abtained from employing GridSearch and cross-validation. Instead, we manually tuned the hyperparameters by evaluating performance on the VL set and proceeded empirically through various trials. The CNN was trained for 300 epochs with early stopping and a patience of 20.
In Table 12 below, we list the network's architecture alongside the different hyperparameters used:

Parameter	**Value**
Layers type and number	Conv1D, 3
Number of Filters (Conv1D layer 1,2,3)	16,32,64
Kernel Size (Conv1D layer 1,2,3)	8,5,3
Activation Function (Conv1D layer 1,2,3)	ReLU
Kernel Regularization (Conv1D layer 1,2,3)	L2 0.01, 0.01, 0.05
Dropout (after Conv1D layer 1,2,3)	0.3
Batch Normalization (after Conv1D layer 1,2,3)	Yes
Pooling Layer (after Conv1Dlayer 3)	Global Average Pooling1D
Activation Function (Dense layer)	Softmax
Kernel Regularization (Dense layer)	L2 (0.05)
Loss Function	Sparse Categorical Crossentropy
Optimizer	Adam
Learning Rate	0.0002

Table 12: Parameters and structure of the CNN model

Additionally, the loss curves for the TR (blue) and VL (orange) losses are presented below, demonstrating robust convergence and effective management of overfitting.

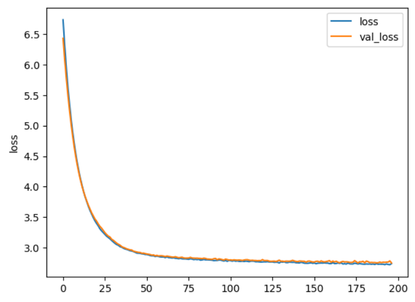


Figure 23

The TR accuracy reached 0.18, while the VL accuracy attained 0.16, and the test set yielded an accuracy of 0.14 as reported in Table 13.

	Precision	Recall	F1-score	Support
0	0.00	0.00	0.00	20
1	0.00	0.00	0.00	20
2	0.21	0.15	0.18	20
3	0.33	0.10	0.15	20
4	0.00	0.00	0.00	20
5	0.00	0.00	0.00	20
6	0.00	0.00	0.00	20
7	0.09	0.40	0.15	20
8	0.28	0.60	0.38	20
9	0.00	0.00	0.00	20
10	0.22	0.65	0.33	20
11	0.08	0.05	0.06	20
12	0.00	0.00	0.00	20
13	0.21	0.20	0.21	20
14	0.00	0.00	0.00	20
15	0.08	0.35	0.13	20
16	0.00	0.00	0.00	20
17	0.00	0.00	0.00	20
18	0.00	0.00	0.00	20
19	0.11	0.20	0.14	20
Accuracy			0.14	400
Macro Avg	0.08	0.14	0.09	400
Weighted Avg	0.08	0.14	0.09	400

Table 13: Classification Report on the test set

The overall accuracy appears to be better than in the previous case. However, both the Macro and Weighted accuracy are lower compared to before. This suggests that the CNN might not have been optimally tuned.

6 Series of studies on the tabular dataset

The tabular dataset described in Section 2.2 is used for different types of analysis and considering different approaches even for the same study. For ease of reference we will call the original tabular dataset as *original dataset*. In this dataset the variable *popularity_artists*, which was presented as a list of artist popularities in Section 2.2, is converted to a single value obtained from the average of all popularities.
The studies on the tabular dataset are structured as follows.
The Section 7 discusses Outliers Detection, different methodologies are used to identify outliers and their treatment.
The Section 8 takes the dataset obtained from the treatment of outliers for application studies to imbalanced learning, understand how to deal with any imbalance issues in the dataset.
The Section 9 deals with advanced classification, different classifications of the original dataset through different models and techniques.
The Section 10 deals mainly with advanced regression on 'popularity' values.
The last Section 11 is about using one explanation methods to illustrate the reasons for the classification in one of the steps of the previous tasks.

7 Outliers detection

As already discussed in the introduction, in this chapter we present the methodologies adopted to identify the top 1% of outliers. The referenced dataset is the tabular one, on which we ran different kind of algorithms to catch them. We present three methods, and the strategy was to remove from the dataset the common outliers identified by each method.

The first method we present is a clustering-based technique, that is DBSCAN. DBSCAN can deal ”natively” with outliers and for this reason it's a good approach in this context. First of all, we identified the best epsilon i.e best value for the radius in order to consider a number of points belonging to a cluster and we found it equal to 0.1, which we used together with minimum number of points equal to 5 to run DBSCAN on the tabular dataset. Another method used is Local Outlier Factor (LOF) which identifies outliers by considering neighborhood density, using default k equal to 20. In addition, Isolation Forest was used which isolates outliers within a dataset by exploiting the concept that outliers are few and different making them easier to isolate than normal data.

Method	**Number of outliers detected**
DBSCAN	105073
Isolation Forest	11753
LOF	1027

Table 14: Outliers detection by the 3 different methods

The Table 14 shows the number of outliers identified by the different algorithms, and it can be said that LOF that manages to capture fewer of them than the other two. This may be due to the fact that LOF is a density-based algorithm for classifying a point as an outlier, but if densities are not marked, LOF may have problems, and in addition, identification also depends on the choice of k.

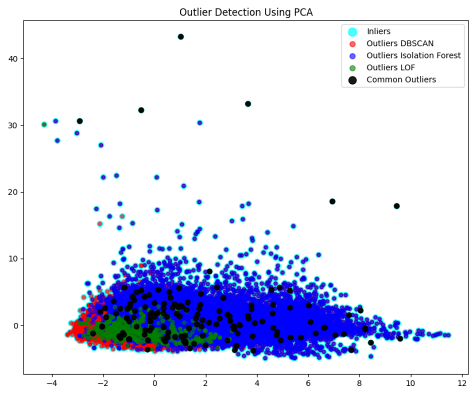


Figure 24: Outliers detection with PCA

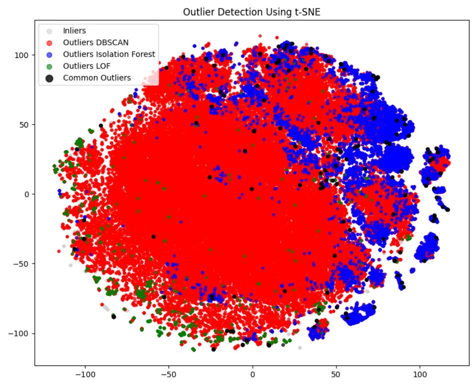


Figure 25: Outliers detection with t-SNE

Visualization of the outliers identified by the three different methods is carried out with two dimensionality reduction techniques: PCA and t-SNE (Figures 24 and 25), which allow different results to be detected on the same data. Regarding PCA, it can be observed that DBSCAN detects outliers mainly around the edges of the main data aggregate, while Isolation Forest detects most outliers and LOF identifies outliers similar to DBSCAN, mainly at the edges of the cluster. Common outliers (black dots) are those on which all methods agree and are distributed most in the main cluster. In the case of t-SNE we highlight the fact that DBSCAN detects a large number of outliers distributed mostly uniformly throughout the graph, Isolation Forest detects outliers that are more concentrated and close together, and LOF identifies outliers similarly to DBSCAN, but with some differences in specific locations. Common outliers are fewer in number and are found in various scattered locations throughout the graph. In conclusion, DBSCAN detects many outliers, especially with t-SNE, while Isolation Forest tends to identify more concentrated outliers, both with PCA and with t-SNE, and LOF identifies outliers mainly at the edges of clusters, similar to DBSCAN but fewer in number. As for common outliers, they are relatively few, indicating that each method has its own detection criteria that do not always coincide.
To identify potential outliers, those detected jointly by the three different methods were considered. Since the number of such outliers was very small, amounting to 200, they were removed from the dataset.

8 Imbalanced Learning

For the study related to imbalanced learning, the dataset obtained from the removal of outliers was taken into analysis by performing a classification task for the *mode* variable with KNN. Since the outliers shared by the three different methods were found to be essentially few, this did not lead to a strong imbalance within dataset. Therefore, to treat this analysis anyway, it was decided to create an imbalance. The *mode* variable is a dichotomous variable whose value can be either 0 or 1. Once the split between train and test was implemented, elements from the minority class 0 were removed to promote the imbalance, so that class 1 accounts for 96%, with 69779 instances, while class 0 is 4%, with 2907 instances of the total. This imbalance is treated through two different approaches involving Undersampling and Oversampling. After applying the relevant methods, 5-fold-cross validation is used to find the best K for KNN.

8.1 Undersampling

Regarding undersampling the two methods implemented are Tomek Links[17] and Edited Nearest Neighbors[18].

Class	F1-score	support
0	0.0	596
1	0.98	13739

Table 15: results for KNN classification after implementing Tomek Links

Class	F1-score	support
0	0.0	558
1	0.98	12625

Table 16: results for KNN classification after implementing Edited Nearest Neighbors

As can be seen, the performance for both methods is mostly similar. Reducing the majority class does not lead to any kind of performance improvement since the imbalance persists to the point that the minority class is clearly disregarded resulting in an F1-score of 0. Undersampling is not an effective method to implement balancing when there is a strong imbalance within the dataset as the case taken under analysis.

8.2 Oversampling

The oversampling methods used are Smote[19] and ADASYN[20].

Class	F1-score	support
0	0.94	13913
1	0.93	13999

Table 17: results for KNN classification after implementing SMOTE

Class	F1-score	support
0	0.94	14057
1	0.93	14014

Table 18: results for KNN classification after implementing ADASYN

In this case the results turn out to be better as the following algorithms leading to generating new examples for the minority class leading to a clear balance, improving performance especially for the minority class with an F1-score of 94% in relation to both algorithms.

[17]Tomek Links are a technique used to reduce the imbalance in imbalanced datasets by removing instances from the majority class that are close to instances from the minority class.

[18]Edited Nearest Neighbors is an undersampling method technique that remove the majority class to match the minority class. It works by removing samples whose class label differs from the class of the majority of their k nearest neighbors.

[19]Smote is an oversampling technique that generates synthetic samples from the minority class to match the majority class.

[20]ADASYN is a technique used to address imbalanced datasets in machine learning, improving classification performance for underrepresented classes. It focuses on the minority instances that are difficult to classify correctly, rather than oversampling all minority instances uniformly. It assigns a different weight to each minority instance based on its level of difficulty in classification.

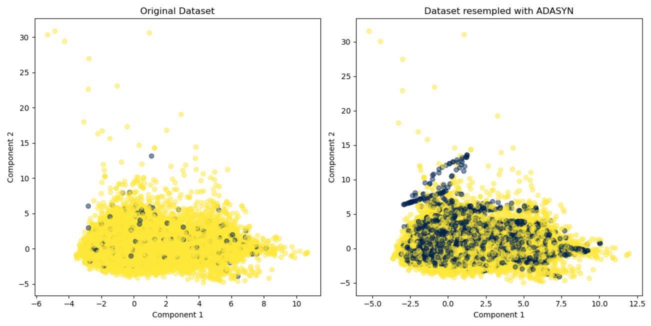


Figure 26: Comparison with PCA between the unbalanced dataset and the resampled dataset with ADASYN

In the Figure 26, a comparison is made between the unbalanced dataset and the dataset obtained from the generation of examples for the minority class using ADASYN, and it is identified how its application leads to greater balance within data.

9 Advanced Classification

We repeated the multi-classification task described in Section 5, this time using the original dataset described in Section 6. For this task, we employed advanced classification models, including Support Vector Machines (SVM) with both linear and nonlinear kernels, Logistic Regression, and Neural Networks (NN). Additionally, we utilized advanced techniques such as bagging[21] and boosting[22]. For each model a standard suddivision of 80% training and 20% test is applied. Moreover, stratification is employed. Regarding the model selection phase, we used cross-validation with k=5 and GridSearch for all models. Due to their higher computational cost, we opted for RandomizedSearch for the NN, Gradient Boosting Machine(GBM) and XGBoost.
In our hyperparameter space for GridSearch for each model, we considered the following aspects.
Controlling model complexity: this involved tuning regularization parameters to avoid overfitting and achieve a balance between complexity and accuracy, which is crucial. For models like SVMs and Logistic Regression, the strength of the regularization is inversely proportional to the C parameter.
Optimizing algorithms: to maximize the performance of the classifiers, we experimented with different algorithms and varied the maximum number of iterations.
Monitoring optimizer progression: we implemented early stopping to halt the optimizer when performance improvements plateaued or when the error did not reduce beyond a certain threshold (specified by the *tol* parameter).
Firstly, we established a baseline for our 20-genres classification task. Using the DummyClassifier() from sklearn with the 'most frequent' strategy, we obtained an accuracy of 0.05 (essentially 1/number of classes). The ROC curves for each genre are depicted in the Figure 27.

[21] Bagging, short for Bootstrap Aggregating, is a popular ensemble learning technique used to improve the stability and accuracy of machine learning algorithms, especially those that are prone to overfitting.

[22] Boosting is another ensemble learning technique, like bagging, aimed at improving the performance of machine learning algorithms, particularly those with high bias, such as weak learners. Unlike bagging, boosting focuses on sequentially training multiple models, each attempting to correct the errors of its predecessor.

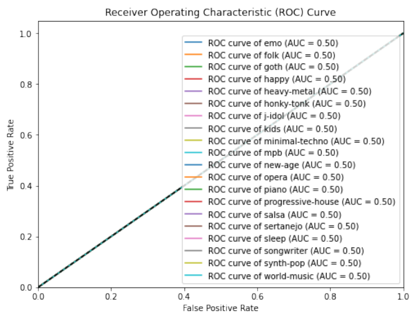


Figure 27: ROC Curves and AUC score of each genre for a DummyClassifier. Basically, a random model.

With the models discussed in the upcoming sections, we will explore how to surpass this baseline and improve the AUC scores.

9.1 Support Vector Machines

We employed linear SVM such as *LinearSVC* model of sklearn and non linear such as *SVC* from the same library. For the first one, Table 19 shows the hyperparameters for the GridSearch and cv=5.

Parameter	**Values**
`C`	[1, 10, 100]
`dual`	[True, False]
`max_iter`	[2000, 3000]
`tol`	[0.001, 0.1, 0.01]

Table 19: Grid Search Parameters. The default penalty is L2(Tikhonov).

As seen in Section 9, for the SVM cases, C is a regularization parameter, and *tol* refers to the stopping criterion. We also opted to experiment with different types of problem formulations (*dual* parameter) and varying numbers of optimization runs (e.g., *max_iter* parameter). The training accuracy is 0.54 and again 0.54 for the best cross-validation score using these hyperparameters: C: 10, dual: True, max_iter: 2000 and tol: 0.01. Hence, we trained our LinearSVC with these values and in the Table 20 below the classification report on the test set is depicted.

	Precision	**Recall**	**F1-Score**	**Support**
emo	0.57	0.45	0.50	200
folk	0.54	0.07	0.13	197
goth	0.22	0.03	0.05	200
happy	0.59	0.68	0.63	200
heavy-metal	0.44	0.66	0.53	200
honky-tonk	0.62	0.80	0.70	196
j-idol	0.54	0.57	0.56	200
kids	0.70	0.81	0.75	198
minimal-techno	0.73	0.88	0.80	199
mpb	0.29	0.14	0.18	200
new-age	0.55	0.66	0.60	199
opera	0.49	0.60	0.54	198
piano	0.57	0.32	0.41	200
progressive-house	0.49	0.61	0.55	200
salsa	0.53	0.79	0.64	199
sertanejo	0.51	0.79	0.62	200
sleep	0.82	0.84	0.83	200
songwriter	0.32	0.24	0.28	197
synth-pop	0.37	0.26	0.30	200
world-music	0.42	0.57	0.49	200
accuracy			0.54	
macro avg	0.52	0.54	0.50	3983
weighted avg	0.52	0.54	0.50	3983

Table 20: Classification report, note the many improvements respect to the baseline in Figure 27.

Despite the low accuracy on the test set, one can note that on the genre 'minimal-techno' and 'sleep' (class 8 and 16 respectively) the classifier performes better. We already seen this behaviour in Section 5.1, Table 8 when the classification was performed using the time series dataset instead of the tabular one. On the other hand 'folk' and 'goth' have the lowest F1 Score. Since this behavior will be systematic for all models, we henceforth only report the F1 score for the most interesting genres. Furthermore, we employed also non linear SVM such as *SVC* model of sklearn library. Table 21 shows the hyperparameters for the GridSearch and cv=5.

Parameter	**Values**
`C`	[1, 10, 100]
`kernel`	['poly', 'rbf']
`gamma`	['scale']
`degree`	[2, 3]

Table 21: Grid Search Parameters for non-linear SVM case.

Genre	**F1-Score**
folk	0.35
goth	0.40
minimal-techno	0.84
sleep	0.89

Table 22: F1 scores for the best and the worst classified genres, using best hyperparameters found by the GridSearch.

The changes here include the omission of some hyperparameters due to the higher computational cost. However, new parameters have been added, related to the kernel used and its complexity (such as gamma and the *degree* of the polynomial, the latter only applicable when the *kernel* is set to 'poly'). The training accuracy is 0.86 and the best cross-validation score is 0.64 accuracy using these hyperparameters: C: 10, degree: 2, gamma: scale, kernel: 'rbf'. Hence, we trained our SVC (non-linear) with these values and in the Table 22 above the report on the test set is illustrated.
The overall accuracy on test set is 0.64 and one can see once again the strong performance of the classifier on 'minimal-techno' and 'sleep,' as previously observed. Although 'goth' and 'folk' remain particularly low, even these genres show better results compared to their F1-score values in the previous Table 20.
The introduction of non-linearity through the Radial Basis Function (RBF) kernel[23] has made the

[23]In SVMs, the RBF kernel allows the algorithm to operate in a higher-dimensional space without explicitly calculating

classifier's decision boundary more flexible. Consequently, this table shows improvements in F1-score values, including the general accuracy.
Moreover, it could be intriguing to perform permutation feature importance, that is based on measuring the change in model performance when feature values are randomly permuted. This approach is robust to non-linear relationships between features and the target variable, which is particularly relevant for non-linear SVM models that can capture complex patterns in the data.
Furthermore, SVC is more complex than LinearSVM, requiring greater attention to regularization. Therefore, it could be intriguing to investigate the role of the C parameter in relation to validation set accuracy more deeply, while keeping the other parameters the same as those chosen by GridSearch, and yet using cv=5.

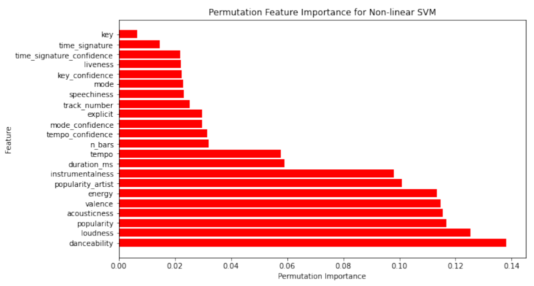


Figure 28: Permutation feature importance for an SVC with an RBF kernel.

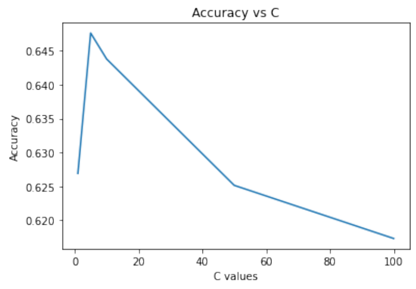


Figure 29: The role of C on the validation accuracy for SVC non linear.

It can be seen from Figure 28 above that the most important feature is 'danceability', which significantly influences the SVM's decisions. Note also that the most important features are those related to the song, specifically continuous attributes such as danceability, loudness, and popularity. The role of the C parameter is well explained by the Figure 29, which shows that the optimal trade-off between accuracy and complexity (also known as the *bias-variance* trade-off) is around C=5 and C=10. Increasing the C value (weaker regularization) leads to a deterioration in performance (leading to a more complex model that may struggle to generalize), as indicated by the downward slope starting from C=20 and continuing to C=100. In conclusion, for this task and dataset, selecting the appropriate value for the C penalty in the SVC case is a crucial point.

9.2 Logistic Regression

Logistic regression was also employed, and Table 23 presents the hyperparameters used for the GridSearch. Unlike in the SVM case, where we didn't include different types of penalties in the hyperparameter search space, we found it interesting to do so here. Specifically, we experimented with the "l1" penalty (despite that, GridSearch still chose l2 penalty).

Parameter	**Values**
penalty	[l1, l2]
C	[1, 10, 100]
solver	[liblinear, lbfgs]
max_iter	[2000]
tol	[0.001, 0.01]

Table 23: Parameter Grid for Logistic Regression for multiclassification task. *solver* refers to the different optimization algorithms.

Genre	**F1-Score**
folk	0.20
goth	0.23
minimal-techno	0.84
sleep	0.83

Table 24: F1 score of the relevant genres for Logistic Regression, using best hyperparameters found by the GridSearch.

The training accuracy is 0.56 and the best cross-validation score is 0.55 accuracy using these hyperparameters: 'C': 100, 'max_iter': 2000, 'penalty': 'l2', 'solver': 'lbfgs', 'tol': 0.001. Hence, we trained our Logistic Regression with these values and in the Table 24 above the report on the test set is depicted.

the coordinates of the data points. This is known as the "kernel trick," which enables SVMs to efficiently handle nonlinear decision boundaries.

The overall test accuracy is 0.55 and once again, it is evident that even this classifier struggles to accurately classify the 'folk' and 'goth' genres. Despite the decision of the GridSearch, as discussed before we experimented with the "l1" penalty with respect to different possible values of C, in this case it was necessary to use 'liblinear' as optimization algorithm for comparison purpose, because 'lbfgs' only supports 'l2' penalty. Once again, for Figure 30 we used cv=5 and took the performance on the mean across folds.
Additionally, the ROC curves obtained on the test set for all genres are depicted in Figure 31.

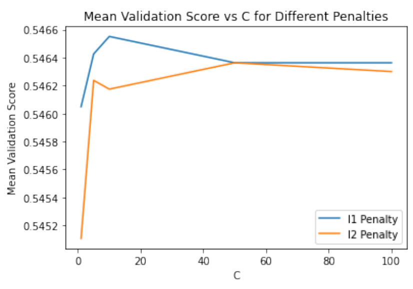


Figure 30: The role of the different penalties with respect to values of C on the validation accuracy.

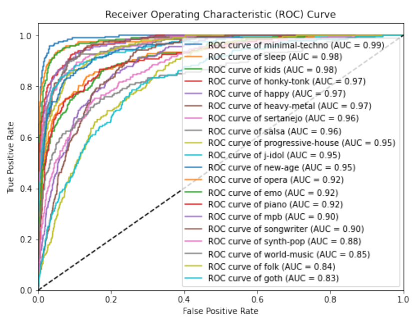


Figure 31: The ROC curves for Logistic Regression on all genres.

For lower values of C (strong regularization), approximately from C=1 to C=40, the l2 penalty appears to reduce complexity and accuracy more than the l1 penalty, although the differences are very minimal. For higher values of C, from C=50 (where the two lines intersect perfectly) to C=100, both penalties exhibit the same behavior. In conclusion, for this task and dataset, using l1 or l2 penalties in Logistic Regression with different values is not expected to significantly change the overall validation accuracy.

9.3 Neural Networks - sklearn

Neural Network are also explored, employing Multi Layer Perceptron (MLP) of sklearn library, and Table 25 below presents the hyperparameters used for the Grid- Search. Note that early_stopping param is set on 'True'.

Parameter	**Values**
`hidden_layer_sizes`	[(100,), (50, 50), (50, 100, 50)]
`activation`	['logistic', 'tanh', 'relu']
`solver`	['adam', 'sgd', 'lbfgs']
`alpha`	[0.0001, 0.001, 0.01]
`learning_rate`	['constant', 'adaptive']
`momentum`	[0.1, 0.01, 0.001]
`n_iter_no_change`	[20, 30]
`max_iter`	[300, 400, 500]

Table 25: Parameter Grid for MLPClassifier hyperparameters tuning

Genre	**F1-Score**
folk	0.32
goth	0.33
minimal-techno	0.86
sleep	0.88

Table 26: F1 scores for selected genres using best hyperparameters found by the RandomizedSearch.

This extensive GridSearch highlights the substantial effort required to achieve good tuning for a Neural Network. Unlike previous models, the number of hyperparameters is considerably larger. Neural Networks are indeed challenging to tune and expensive to train. Therefore, we decided to use RandomizedSearchCV (with 30 iterations) instead of GridSearchCV. This approach allows us to explore more hyperparameters while avoiding an overwhelming number of combinations.
In this context, regularization is controlled by the 'alpha' parameter and uses L2 regularization. Early stopping monitors the validation loss during epochs (max_iter) and waits for a specified number of epochs (patience or n_iter_no_change) before halting training if the validation loss doesn't decrease (in a threshold *tol*), indicating potential overfitting.
The 'momentum' parameter helps avoid the local minima problem of the gradient descent, favoring a global minimum. The solver includes two new options: 'adam', which is effective for large datasets, and

'sgd', which stands for stochastic gradient descent using batches, while 'lbfgs' is faster and preferred when the dataset is smaller.
The training accuracy is 0.73 and the best cross-validation score is 0.65 accuracy using these hyperparameters: 'solver': 'adam', 'n_iter_no_change': 30, 'momentum': 0.001, 'max_iter': 300, 'learning_rate': 'adaptive', 'hidden_layer_sizes': (100,), 'alpha': 0.0001, 'activation': 'tanh'. Hence, we trained our NN with these values and in the Table 26 above the report on the test set is depicted. The overall test accuracy is 0.65 and in the Figure 32 the progress of the loss on the TR is depicted, and in Figure 38 also ROC Curves on the test set for all genres are illustrated.

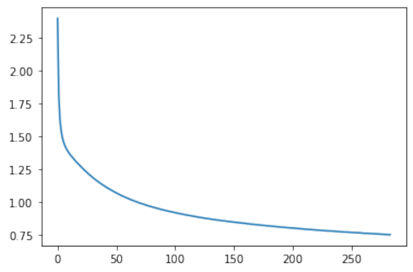


Figure 32: The descending of the TR loss with respect to number of epochs.

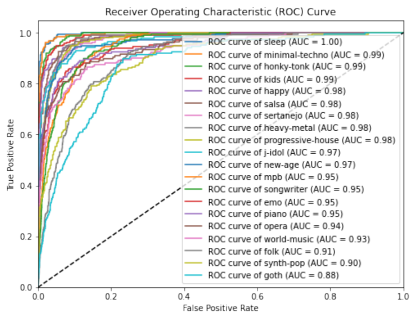


Figure 33: The ROC Curves of the NN.

From the ROC Curves one can see that in general the model is far to be a random estimator as the baseline in Figure 27 was. Once again, 'sleep' has the best AUC together with 'minimal-techno' and emerges also 'honky-tonky'.
It's important to note that the complexity of the NN increases during training, but effective regularization and early stopping can manage this behavior effectively. It's challenging to pinpoint the exact parameters that contribute most to the "complexity" of a NN, but it certainly depends (though not exclusively) on the number of hidden layers and units. It's interesting to observe in the following images 34, 35 the various configurations of the NN in relation to validation accuracy and the role of early stopping in handling overfitting. Once again, the accuracy is measured using cv=5, applying the best hyperparameters already chosen by the GridSearch above, and focusing on whether or not early stopping is used.

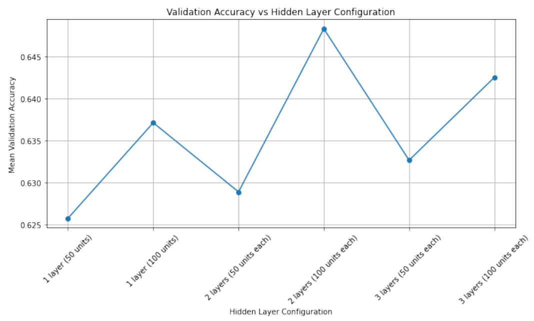


Figure 34: The role of the different configurations of the NN with respect to the accuracy when early stopping is used.

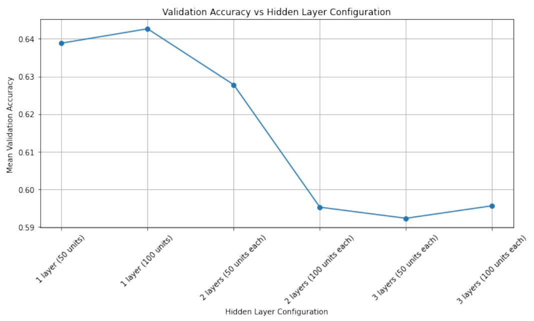


Figure 35: The role of the different configurations of the NN with respect to the accuracy when early stopping is *not* used.

In Figure 34, where early stopping is implemented, the validation loss shows very little change among the different configurations. However, in Figure 35, where early stopping is not implemented, it is evident that increasing the hidden layers from 2 to 3 and the units from 50 to 100 leads to a more complex model that is unable to generalize, as indicated by the reduction in validation accuracy. As previously mentioned, the complexity and the appropriate regularization do not depend on a single parameter alone. Therefore, it could be interesting to deeply investigate the behavior of the 'alpha' (l2 regularization) parameter. Initially, as already discussed 80% of the dataset was allocated to training.

To conduct the experiments described below, we further split off 10% from this training portion for validation purposes. This resulted in a final distribution of 70% for training (TR), 10% for validation (VL), and the remaining 20% for testing. This approach allowed us to observe the behavior of 'alpha' on the VL set without affecting the test set, which will provide an unbiased evaluation of our model's performance. Consequently, we plotted the TR loss and VL loss using the best hyperparameters previously found through RandomizedSearch.

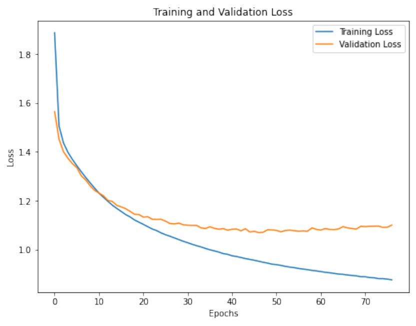


Figure 36: The role of the 'alpha'. The curves are obtained using the best parameters suggested by the RandomizedSearch which include for 'alpha' a value equal to 0.0001.

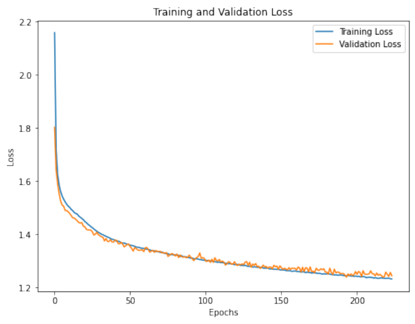


Figure 37: The role of the 'alpha'. The curves are obtained using the best parameters suggested by the RandomizedSearch but increasing 'alpha' = 0.01.

While in Figure 36 the VL loss curve rises at a certain point, indicating overfitting, in Figure 37 increasing the 'alpha' parameter results in the VL loss following the TR loss more closely, thus avoiding overfitting. However, it is important to note that in Figure 36, the accuracy obtained on the test set is 0.65, while in Figure 37, the accuracy is 0.62. Note that the loss values in Figure 37 are higher than those in Figure 36. The reasons for this discrepancy are difficult to pinpoint; it may be that the learning rate needs to be increased due to the higher regularization being used. Furthermore, Figure 36 also illustrates how early stopping works; it shows that the training stops after a few epochs once the model begins to overfit. In contrast, Figure 37 shows that the training continues for 250 epochs, as no overfitting is encountered.
In conclusion, for this task and dataset, using the early stopping criterion in the NN, is crucial to avoid overly complex models that can easily overfit and might reduce performance on unseen data. On the other hand, it is not possible to pinpoint the exact reasons for overfitting, and focusing solely on the 'alpha' parameter can be misleading. While tuning this param is very important, it is also necessary to be aware of other parameters, as increasing 'alpha' can reduce the accuracy.

9.4 Bagging - Random forest

For bagging, Random Forest (RF) model that is an ensemble of DecisionTree (DT) is employed and in the Table 27 below the values for the tried hyperparameters are depicted.

Parameter	**Values**
`n_estimators`	[100, 200]
`max_depth`	[10, 20]
`min_samples_split`	[2, 5, 10]
`min_samples_leaf`	[1, 2, 4]

Table 27: Parameters of the GridSearch for the RF

Genre	**F1-score**
folk	0.30
goth	0.43
minimal-techno	0.85
sleep	0.92

Table 28: F1-score of the relevant genres for RF, using best parameters found by the GridSearch.

The GridSearch above is notably distinct from those previously encountered. In this scenario, there isn't a mathematical penalty to incorporate such as 'alpha' (NN) or 'C' (SVM, Logistic R). Instead,

regularization operates in a more sparse way on parameters such as max_depth, which restricts the depth of the DT and min_samples_split, which aids in controlling tree growth and mitigating overfitting by ensuring nodes possess sufficient samples for reliable decisions. Similarly, min_samples_leaf serves a comparable role to min_samples_split, helping prevent overfitting by establishing a threshold for the minimum data required at each leaf node. The training accuracy is 0.98 and the best cross-validation score is 0.69 accuracy using these hyperparameters: 'max_depth': 20, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200. Hence, we trained our Random Forest with these values and in the Table 28 above the report on the test set is depicted.
The overall accuracy on test set is 0.69 and the report above confirms that 'minimal-techno' and 'sleep' are the easiest genres for the model to classify. Notably, 'sleep' is the most distinctive genre, as it has the highest F1 Score of 0.92. The ensemble of DT significantly improves the general performance on the 'folk' and 'goth' genres, both achieving their best score so far, exceeding 0.40 for the latter.
Note that, the DT are highly efficient but also more prone to overfitting. In fact, the RF achieves the highest accuracy of 0.98 on the TR compared to previous models. However, its performance on the test set of 0.69, while still good and better than previous models, is slightly lower.
The DT offer also greater interpretability, allowing for in-depth investigation of their decision paths. Below, Figure 39 shows the feature importance obtained and Figure 38 shows the ROC curves for each of 20 genres.

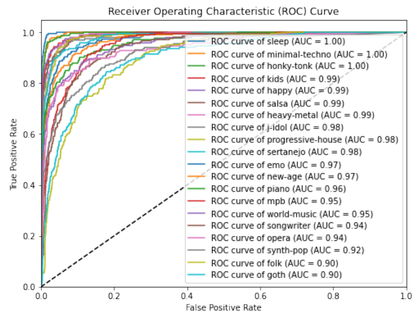


Figure 38: ROC curves and AUC score. In this context, for every genres.

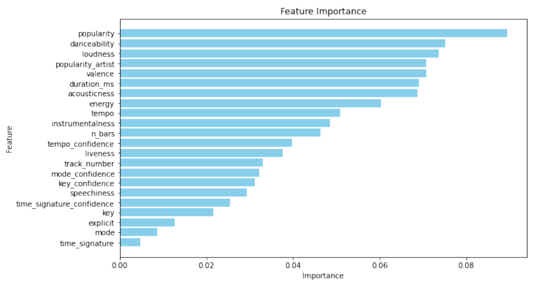


Figure 39: Features importance obtained investigating the RF.

The observations are consistent with those seen in the non-linear SVM case, as shown in Figure 28. It is intriguing to note that 'popularity' is the most important feature, suggesting that many genres can be distinguished by their popularity. The top-ranked features are continuous, song-related attributes such as popularity, loudness and acousticness. On the other hand, binary attributes like mode, energy, and various types of confidence appear in the lower ranks. Notably, the position of 'popularity_artist' is noteworthy; this feature was added by us and represents the combined popularity of artists collaborating on a song (see Section 2.2 and Section 6 for details).
Moreover, ROC Curves in the Figure 38 above confirms the previous observations: 'sleep' and 'minimal-techno' have the highest AUC of 1.0, in stark contrast to 'folk' and 'goth', which are at the lower end of the ordered list. However, it is worth noting that 'folk' and 'goth' still achieve a good AUC score of 0.90.
Furthermore, it could be intriguing to compare this model with a variation of it, employing a different number of trees as estimators, or what could be referred to as a *wise crowd*. Specifically, let's consider a lower number, such as 100, 50 or 20, keeping the same parameters values as already chose by the GridSearch and again employing cv=5.

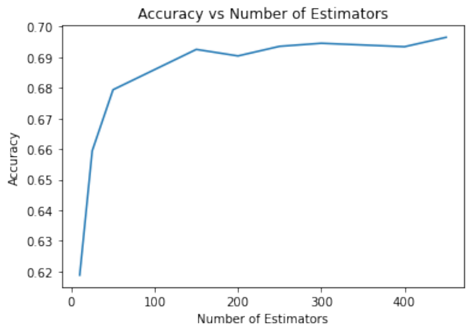


Figure 40: The progress of the VL Accuracy with respect to number of estimators.

From this figure, one can observe lower validation accuracy with 10-50 estimators, however with a significant increase in performance from 10 to 100 estimators. After 200 estimators, the accuracy reaches a plateau, and further increasing the number of estimators to 400 raises the risk of creating an overly complex model without improving significantly the accuracy. In conclusion, for this task and dataset, the number of estimators in a RF context, is crucial for smaller quantities. However, increasing their number beyond a certain point only leads to a computationally expensive model without any substantial improvement in accuracy.

9.5 Boosting

9.5.1 Gradient Boosting Machine - GBM e XGBoost

Parameter	Values
Learning_rate	Uniform(0.01, 1)
Max_depth	randint(1, 10)
N_estimators	randint(50, 60)
Min_samples_split	randint(2, 10)
Min_samples_leaf	randint(1, 10)
Max_features	[sqrt, log2, None]
Subsample	Uniform(0.0, 1)

Table 29: Hyperparameter values

Genre	F1-score
Folk	0.29
Goth	0.48
Minimal techno	0.87
Sleep	0.88

Table 30: Best Genres F1-scores with Gradient Boosting

Parameter	Values
Learning_rate	Uniform(0.3)
Max_depth	randint(1, 10)
Gamma	Uniform(0.0, 1.0)
Reg_lambda	Uniform(0.0, 1.0)
Tree_method	[auto, hist, exact, approx]
Early_stopping_rounds	2

Table 31: Hyperparameter values for XGBoost

Genre	F1-score
Folk	0.36
Goth	0.47
Minimal techno	0.87
Sleep	0.89

Table 32: Best Genres F1-scores with XGBoost

The best parameters for the implementation of both models were obtained through Random Search by providing the hyperparameters found in the Tables 29, 31 and also implementing early stopping. The learning rate returned for Gradient Boosting is 0.21 while for XGboost has a default value of 0.35. XGBoost can use a higher learning rate without risk of overfitting since it is a model that has additional regularizations such as gamma which controls the minimum loss reduction required to make a split in the tree node, in our case equal to 0.59, and reg_lambda, to control the complexity of the model, with a value of 0.68. These regularizations are not present in the Gradient boosting case so it may require a lower learning rate to obtain a more stable model. Moreover, the additional regularizations allow XGboost to build shallower trees in fact its max_depth is equal to 3 while for Gradient boosting it is 8. In the case of Gradient boosting there are additional parameters for model building such as min_sample leaf equal to 3, min_sample_split of 2, and the number of phases to be executed, n_estimators, with value 57, a high number allows more robustness to overfitting ensuring better performance. The best results obtained for the classification concern the same genres but with slightly different values for F1 score, especially for Folk, a genre for which Gradient Boosting has worse perfomances, but despite not having multiple regularizations it still obtains good perfomances extremely comparable to XGboost. The two VL accuracies differ slightly, amounting to 0.67 for Gradient Boosting and 0.71 for XGBoost. Regarding the evaluation of the models on the TR and test there are few differences. The accuracy for the TR in the case of Gradient Boosting is 0.93 while for XGboost it is 0.89, it is noticeable that XGBoost depends less on the training data such that it presents better perfomances for the test with an accuracy of 0.70 while for Gradient Boosting it is equivalent to 0.68.

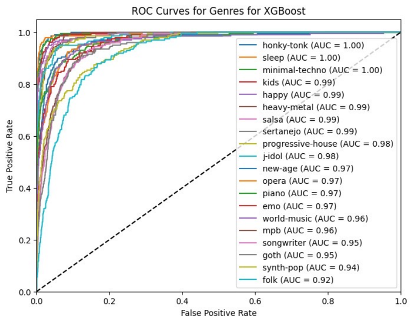


Figure 41: ROC Curve for Gradient Boosting

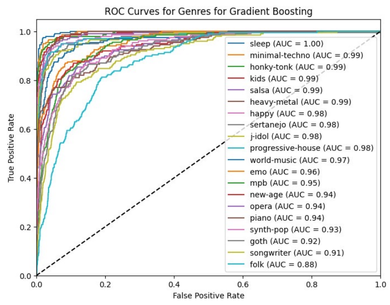


Figure 42: ROC Curve for XG-Boost

In relation to the ROC Curves for genre classification, however, little difference can be seen, both still achieve good performance on the same genres except that Gradient Boosting implements a very good classification for sleep, followed by minimal-techno and honky-tonk, while in the case of XGBoost there is honky-tonk followed by sleep and minimal techno.

9.6 Final Comparison

In the following Table 33 our experiments with the models and evaluations are resumed.

Model	TR Accuracy	VL Accuracy	Test Accuracy
XGBoost (Boosting)	0.85	0.71	0.70
Random Forest (Bagging)	0.98	0.69	0.69
GBM (Boosting)	0.95	0.67	0.68
Sklearn - NN	0.73	0.65	0.65
Non - Linear SVM	0.86	0.64	0.64
Logistic Regression	0.56	0.55	0.55
Linear SVM	0.54	0.54	0.54
DummyClassifier (baseline)	0.05	0.05	0.05

Table 33: Training (TR), Validation (VL) and test accuracy for the different models on the multi-classification task. The model are in descent order based on VL accuracy. Note that the model more prone to overfitting are those having the higher disparities between the TR and VL accuracies.

In this section, starting from the baseline shown in Figure 27, we found it straightforward to improve the results. Even using the ”weaker” models, such as Linear SVM and Logistic Regression (as shown by their VL accuracy in the Table 33), it was possible to achieve respectable AUC scores and ROC curves. In fact, it’s important to consider that this task is not simple, as it involves multiclass classification with 20 target classes, resulting in a very low baseline accuracy of 0.05. Despite their lower performance, the ”weaker” models appeared easier to tune because they contain fewer parameters and seem more resistant to overfitting due to their less complexity. On the other hand, more complex models such as NonLinear SVM, Neural Networks, Random Forests, and boosting models ”compensate” for their higher accuracy with greater difficulty in tuning and regularization. These models, particularly GBM and Random Forest, exhibit a tendency to overfit, as indicated by their substantial decrease in accuracy from the TR set to the VL and test sets. However, despite this propensity for overfitting, XGBoost emerges as the top-performing model on both the validation and test sets. Additionally, the disparities between the TR set accuracy and the VL set accuracies are relatively lower for XGBoost compared to the second and third-best models, which are Random Forest and GBM Boosting. It might appear somewhat unusual that the neural network (NN) model didn't secure a top position. We suspect this could be attributed to the fact that an exhaustive GridSearch wasn't employed due to computational constraints. Consequently, the random factor might have led the model to converge to a local minimum, or perhaps the chosen parameters weren't optimal (although GridSearch tuning would be necessary for confirmation). However, the early stopping and regularization effectively mitigated the risk of overfitting, as evidenced by the marginal decrease in accuracies between the TR (0.73) and the VL and test sets (0.65). Moreover, it's worth considering that the top three models are derived from ensemble techniques (bagging, boosting), leading one to view the neural network as the top performer among individual models.

10 Advanced Regression

We employed advanced regression techniques to predict the 'popularity' variable using the columns of our tabular dataset discussed in Section 9 as input. To achieve this, we utilized models capable of capturing non-linear patterns, such as MLRegressor (a neural network for regression) and Random Forest Regressor (an ensemble of trees for regression tasks). To evaluate the performance of each model, we utilized metrics including R-squared (R^2) and Mean Squared Error (MSE). Moreover, we inspected the residuals distribution and composition through residual plot and the histogram of the errors. No cross-validation is used due to the computational expensive resources required to run a GridSearch or even a RandomizedSearch; 70% of the dataset is destinated to the TR and 10% for the VL with the remaining 20% for the test. We tuned the hyperparameters by manually and empirically looking at the R^2 score on the VL set. Firstly, as before for advanced classification, we established a baseline for our multi-regression task on the 'popularity' variable. We achieved this by utilizing DummyRegressor() from sklearn with the 'median' strategy, chosen for its robustness against outliers. This baseline approach yielded an R^2 score of -0.0003 and an MSE of 524.65. Below, in Figure 43 a residual plot for the DummyRegressor is depicted and an histogram for the errors distributions are also illustrated.

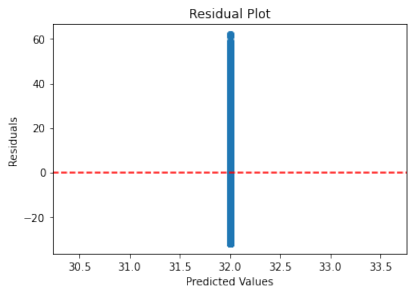


Figure 43: Residual plot.The solid blue vertical line at around 32.0 on the x-axis represents the median of the predicted values.

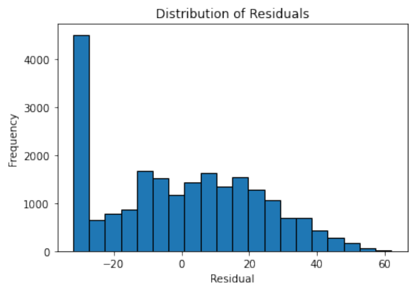


Figure 44: Histogram of the residuals. The long tails on both sides indicate the presence of many large residuals, where the model is making sizable errors in its predictions.

Figure 43 shows the residuals (differences between true and predicted values) on the y-axis and the predicted values or the true values on the x-axis. Ideally, residuals should be randomly distributed around zero, indicating no patterns in the errors. Figure 44 shows the distribution of the residuals, which ideally should be symmetrical, suggesting that the model errors are balanced. Also, the distribution should have a peak around 0, indicating that most predictions are accurate.
With the models discussed in the upcoming sections, we will explore how to surpass this baseline and improve the distribution of residuals and the quality of the predictions.

10.1 Random Forest Regressor

Random Forest Regressor (RFR) is implemented, an ensemble method of DT for regression tasks. We tried 50, 100, 200 and 300 estimators and the best trade-off between precision of the model and its complexity was of 150 estimators. Table 34 below shows the MSE and R^2 score for the different sets achieved by the RFR.

Metric	TR set	VL set	Test set
MSE	22.65	160.30	164.70
R^2	0.96	0.70	0.69

Table 34: Mean Squared Error (MSE) and R^2 Score for Training (TR), Validation (VL), and Test Sets

From Table 34, it is evident that the RFR model overfits the TR data, as indicated by the decrease in R^2 from TR (0.96) to VL (0.70) and test (0.69) sets. Additionally, the MSE significantly increases, with a good MSE of 22 on the TR set increasing to 164.70 on the test set. However, these results are certainly better than those achieved by the baseline, which had an R^2 of -0.0003 and MSE of 524.65. Below, the figures show the residual plot and the histogram of their distribution on the VL set.

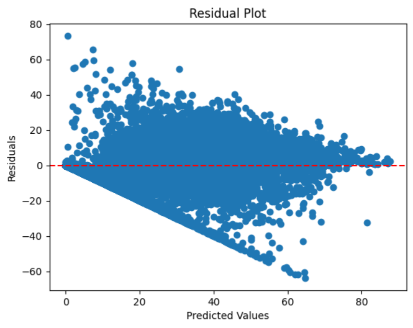


Figure 45: Residual plot. Most points are above the red dotted line, indicating that the model generally underestimates when predicts.

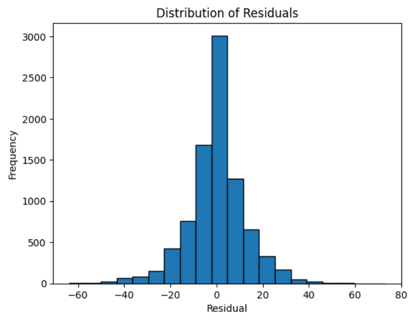


Figure 46: Histogram of the residuals.

Significant improvements can be observed from the baseline in Figure 43 and Figure 44. Figure 45 shows the points distributed around 0 along the red dotted line, indicating that the model makes various types of errors and predictions. Additionally, in Figure 46, the distribution appears more symmetrical compared to the baseline, with a peak around 0, suggesting that the model more frequently predicts accurate values, in fact the model shows a good R^2 according to the Table 34.

10.2 Neural Network - Keras

Neural networks were also explored, utilizing a Keras model trained for 300 epochs. Table 35 below presents the hyperparameters employed after several empirical trials, alongside the evaluation metrics obtained. Once again, early stopping was implemented.

Parameter	**Value**
Num.of Hidd. Layers	3
Neurons per Layer	64, 32, 16
Activation Function	ReLU
Regularization	L2 (0.01)
Learning Rate	0.0001
Optimizer	Adam
Loss Function	MSE
Patience (for Ear.Stopp.)	20

Metric	**TR set**	**VL set**	**Test set**
MSE	227.18	241.50	243.50
R^2	0.57	0.54	0.54

Table 35: Hyperparameters used (Left) and Mean Squared Error (MSE) along with R^2 Score (Right)

Overfitting appears to be more mitigated compared to the RFR case, as shown by the TR and VL loss curves in the Figure 47 below. Since the residual plot is essentially the same as that of the RFR, only the histograms of the residuals for the NN on the VL set are depicted in Figure 48

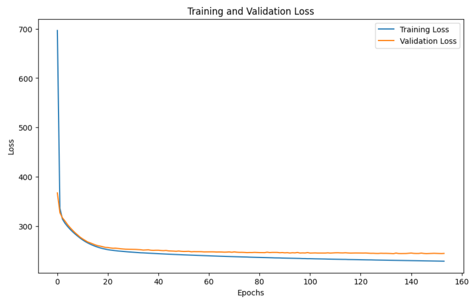


Figure 47: TR and VL loss. The VL loss follows the training loss (TR) more closely, thus avoiding overfitting.

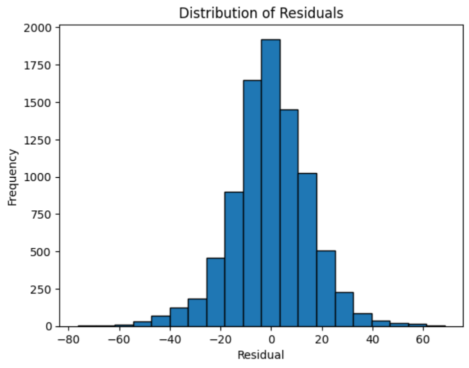


Figure 48: Histogram of the residuals.

Figure 48 shows more symmetry compared to the RFR case in Figure 45. Despite this, it is evident that the frequency of accurate values (2000) is lower than in the RFR case (3000), although a peak around 0 is still present. Consequently, the R^2 is better than the baseline but lower than the RFR case. The MSE is about half that of the baseline but higher than in the previous case.

10.3 Final Comparison

In the following Table 36 our experiments with the models and evaluations are resumed.

Model	MSE TR	MSE VL	MSE Test	R^2 TR	R^2 VL	R^2 Test
RFR	22.65	160.30	164.70	0.96	0.70	0.69
Keras NN	227.18	241.50	243.50	0.57	0.54	0.54
DummyRegressor (baseline)		524.65			-0.0003	

Table 36: Comparison of Mean Squared Error (MSE) and R^2 Score for different models

In this section, starting from the baseline R^2 of -0.0003 shown in the table above, we found it straightforward to improve the R^2 results. Even with the 'worst' model, the neural network (NN) in this context, it was possible to achieve respectable R^2 scores. However, the situation is different for the MSE. Although the RFR shows the best R^2 and is thus the best model, it exhibits clear overfitting, as the VL MSE is much higher than on the TR. For the Keras NN, the MSE is nearly half that of the baseline, which is not particularly good given the complexity of the model with 3 hidden layers. The R^2 score for the NN, at 0.54, is slightly better than random. Nevertheless, overfitting is reduced in the NN case respect of RFR. It could be interesting to inspect the number of residuals for the best model, particularly with respect to the genre whose popularity is being predicted.

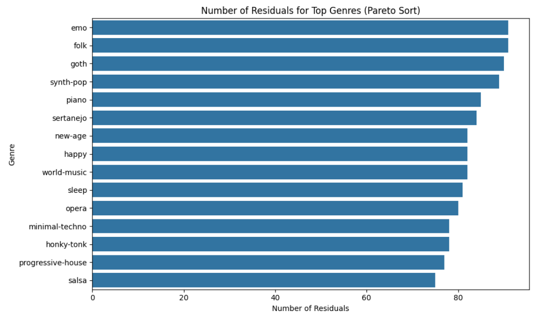


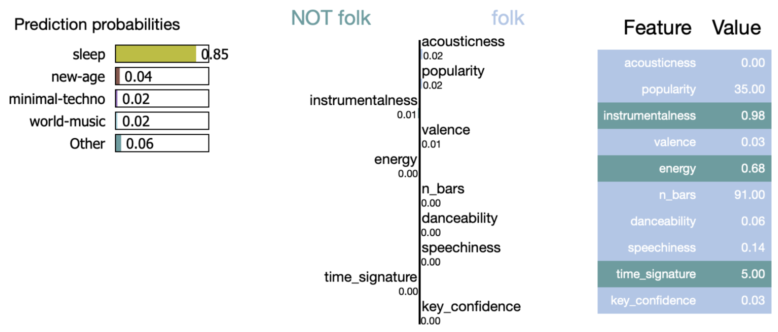
Figure 49: Pareto sort of the genres with the most residuals.

The observations in Figure 49 are consistent with those seen in Section 9 and in the ROC curves. The models, even in the case of regression, struggle to predict the popularity of genres like 'folk' and 'goth', while performing better with 'sleep', 'minimal-techno', and 'honky-tonky'.

11 Explainability

Regarding the explainability analysis, we use LIME and SHAP two model-agnostic tools applied to the random forest model, which had the best perfomances for the classification of the 20 genres realized in the previous tasks. In particular, we focused on explainability for the music genre *sleep* for which we obtained the best results.

11.1 LIME



Feature	Value
acousticness	0.00
popularity	35.00
instrumentalness	0.98
valence	0.03
energy	0.68
n_bars	91.00
danceability	0.06
speechiness	0.14
time_signature	5.00
key_confidence	0.03

Figure 50: Explainability with LIME

Figure 50 shows the explanation with LIME[24] with the relative prediction probabilities for sleep and for the various genres of music, and the characteristics that most influenced the model's prediction. The model assigned a very high probability for the genre "sleep" of 0.85, indicating that the model is very confident of the prediction unlike the other genres which have very low probabilities. The most relevant features in gender prediction are *instrumentalness* (0.98), *energy*(0.68) and *time_signature* (5.00). In contrast, other features such as valence, popularity and acousticness determine less influence. In addition, Folk genre is taken as a reference genre to show the influence of features in the prediction of any genre compared to sleep. The features that have greater influence are useful in discriminating any genre with respect to sleep, features that then contribute to classifying the genre as 'not folk' with respect to folk. This result suggests that the Random Forest model relies predominantly on song instrumentality, energy, and time_signature to determine whether or not a song belongs to the sleep genre.

11.2 SHAP

SHAP[25] is a methodology for interpreting machine learning models that is based on Shapley values. The goal of SHAP is to explain the prediction of a model by showing the importance of each feature for a specific prediction.

[24]Local Interpretable Model-agnostic Explanations (LIME): LIME is a technique used to explain complex machine learning models, often considered "black boxes." It focuses on explaining predictions for individual data points, thus providing local rather than global explanations. To make the explanations easily understandable, LIME uses simple, interpretable models, such as linear models, that approximate the behavior of the complex model in a limited area. One of the most important features of LIME is its model-agnostic nature, that is, its ability to work with any type of machine learning model, regardless of its complexity or structure.

[25]SHAP (SHapley Additive exPlanations): SHAP is a technique for explaining machine learning models, based on the Shapley value theory of game theory. SHAP assigns an importance value to each input feature, representing the contribution of each feature to the prediction of the model. These importance values are calculated by considering all possible combinations of features, thus ensuring consistent and interpretable explanations. SHAP not only provides local explanations for individual predictions, but can also provide global explanations that help to understand the overall behavior of the model.

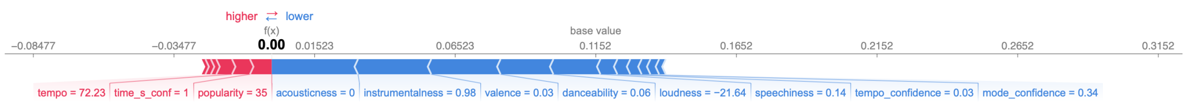


Figure 51: Explainability with SHAP

Figure 51 shows the SHAP explanation plot for the prediction of the music genre ”sleep” which highlights how each feature contributes to the prediction. The base value (0.1152) indicates the average prediction of the model compared to the training set. There are features that increase the probability of the genre such as the case of tempo, time_signature_confidence, and popularity. A tempo of 72.23 BPM is relatively slow and it is relevant in the classification of the genre, time_signature_confidence suggests that the song may have a well-defined rhythmical structure, which may contribute to a steady and predictable rhythm typical of relaxing music, and a low value of popularity may indicate that the song is not overly stimulating or commercial typical just of sleep songs. While attributes that decrease the odds in genre classification are predominantly acousticness, instrumentalness, valence, and danceability. A track with a high instrumentalness decreases the probability of the track being classified as sleep, in fact these tracks will usually tend to have a very low instrumentalness value. A low acousticness of 0, indicating that the track is not acoustic, and low values of valence and danceability indicating tracks that do not report positivity, such as tracks that suggest sad emotions and little danceability, lead to a reduced likelihood in classifying the genre as sleep.

Both methods allowed us to identify the most relevant features for predicting sleep genre. LIME allowed us to have a generic identification of features that can implement a contribution, while SHAP allowed us to get a more specific view by then looking at features that contribute positively or negatively. In both cases instrumentalness is highlighted as a relevant feature.