Report DM2

Mattia Di Sante — 676763
Emanuele Moscuzza — 677879
Emanuele Sciancalepore — 678521

Contents

1 Introduction **3**

2 Data Understanding **3**
2.1 Tracks . . . 3
2.1.1 Duplicates . . . 5
2.2 Artists . . . 6
2.2.1 Duplicates . . . 7
2.3 Time Series . . . 7

3 Time Series Analysis **8**
3.1 Classification . . . 8
3.1.1 KNN . . . 8
3.1.2 Shapelets . . . 9
3.1.3 mini-ROCKET . . . 11
3.1.4 Conclusions on classification . . . 11
3.2 Clustering . . . 12
3.2.1 Features-based . . . 12
3.2.2 K-Means . . . 13
3.2.3 DBSCAN . . . 14
3.3 Motifs and discords . . . 15
3.3.1 Motifs . . . 15
3.3.2 Anomalies . . . 16
3.4 Conclusions on Time Series . . . 17

4 Advanced Data-Preprocessing **18**
4.1 Outliers detection . . . 18
4.2 Imbalanced learning . . . 19

5 Advanced Learning Techniques **20**
5.1 Advanced classification . . . 20
5.1.1 Advanced Imbalanced Learning . . . 21
5.1.2 50 classes . . . 22
5.1.3 20 classes . . . 25
5.2 Advanced regression . . . 26

5.3 DL-based Models for TS Classification 27
5.4 Explainable AI 28
5.4.1 TREPAN 28
5.4.2 LIME 29

1 Introduction

The objective of this report is to analyse **three** datasets, all containing data concerning audio **tracks** extracted from the Spotify database.

The first dataset contains information about **tracks** extracted from Spotify; it is very similar to the one used in the first part but differs in that it contains much more data than the previous one (110 thousand rows compared to 15 thousand). In addition, twelve columns were added to the already existing ones, which brought information about album specifications and the confidence of variables already present.

The second dataset we will discuss is the smallest: it contains about 30 thousand records and gives us all the information about an **artist**, such as their name and popularity.

The last dataset we will discuss is the one containing the **time series**. In this dataset, we have 10 thousand data, each of which is spread over 1280 columns (each column represents an instant of time in which the value is recorded) and each row represents a song. The only two attributes this dataset has are 'id' and 'genre'.

2 Data Understanding

2.1 Tracks

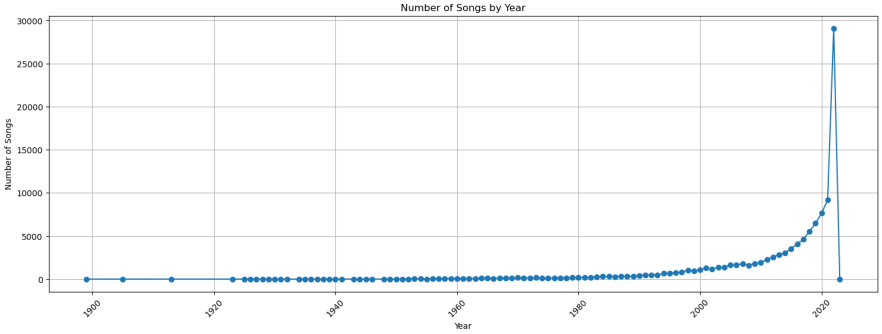
The columns that our dataset has in addition to the previous one are the following: "*id*", "*album_type*", "*disc_number*", "*track_number*", "*album_release_date*", "*album_release_date_precision*", "*album_total_tracks*", "*start_of_fade_out*", "*time _confidence*", "*time_signature_confidence*", "*key_confidence*", "*mode_confidence*"; below we analyse those that we found most interesting and those on which we made some considerations.

- **id**: this variable represents a code that allows us to **uniquely** recognise a song. Furthermore, we thought it might be useful in the event that future analyses require **merging** this dataset with the one containing the time series.

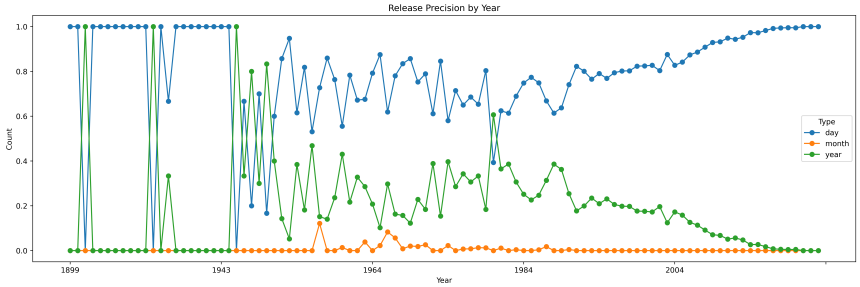
- **album_type and album_total_tracks**: at this point, we will analyse two variables together as they are very closely **related**, in fact "*album_type*" indicates the type of publication the song is in, it can take on three possible values: "single", "album" and "compilation". While "*album_total_tracks*" tells us how many tracks are present within that specific publication. The analysis was carried out on the two variables together as we found numerous **errors** on a semantic level, for example we found 10043 records, all classified as 'singles' but which have more than one track within them, when we would expect only one track within such a publication. Similarly, we also noted that 13 records, 7 of which were classified as "albums" and 6 as "compilations", have only one track within them, when we would expect these to be collections of songs and thus contain more than one track.

- **disc_number**: this variable indicates the number of **discs** in the album, although it is **not** very informative, in fact the **98%** of the dataset has a value of 1, we decided to keep it because it might be useful in future tasks.

- **album_release_date_precision and album_release_date**: here, too, we will analyse the variables in pairs, "*album_release_date_precision*" indicates how much information we have about the **publication date** of the album, it can take values "year" if only the publication year is available, "month" if in addition to the year there is also the month and finally "day" if the complete date is known. While "*album_release_date*" contains the actual publication date of the album based on the level of precision described in the previous variable, in the format "yyyy-mm-dd". As can be seen in Figure 1a, **90%** of songs was released **after 2000**. In our dataset 94% of the records have a precision level of 'day', however we noticed that, especially in the case of older releases, the date often coincides with the first of January (e.g. "yyyy-01-01"), which makes us think that many of these are **placeholders** and not the actual publication dates. Furthermore, the accuracy in the release date decreases going back in time, the normalised count of accuracies per year can be observed in Figure 1b.



(a) Number of songs released by year



(b) Normalized counts of release precision by year

Figure 1: Release date related distributions

- **start_to_fade_out**: this variable indicates, when present, the point in the song at which the fade-out **begins**, however for those songs that do not have it, the variable has a value equal to the duration of the entire track. On this feature we have made a modification in order to make it more **readable**, we have turned all values for songs that have fade-out into **percentages** [0,1] (e.g. a 100-second song with a 90-second start-to-fade-out will have a new value of 0.1), so that we have all songs without fade-out with a value equal to 0.

- **Confidence attributes**: among the new variables we found in the dataset, there are four in particular that can be treated together as they are of similar meaning

"*tempo_confidence*" "*time_signature_confidence*" "*key_confidence*" "*mode_confidence*", each of them indicating the level of **trustworthiness** of the information to which it is dedicated (e.g. "*key_confidence*" indicates how trustworthy the value of the "*key*" column is in that record). At this early stage of the project, we have not yet worked out exactly what they might be used for, but we are keeping them in our data as they might be useful to use as **weights** in some future task.

Furthermore, with regard to the variables already known to us in the dataset:

- **duration_ms**: we decided to transform this variable by changing its unit of measurement from milliseconds to seconds to improve **readability**, and we also decided to drop "*feature_duration_ms*" from the dataset as it correlates **perfectly** with the variable in question.

- **genre**: there are **114** genres in this dataset, of which 97 are equally populated with between 980 and 1001 records for each. The following plot (2) represents the distribution of the **least** present genres within the dataset:

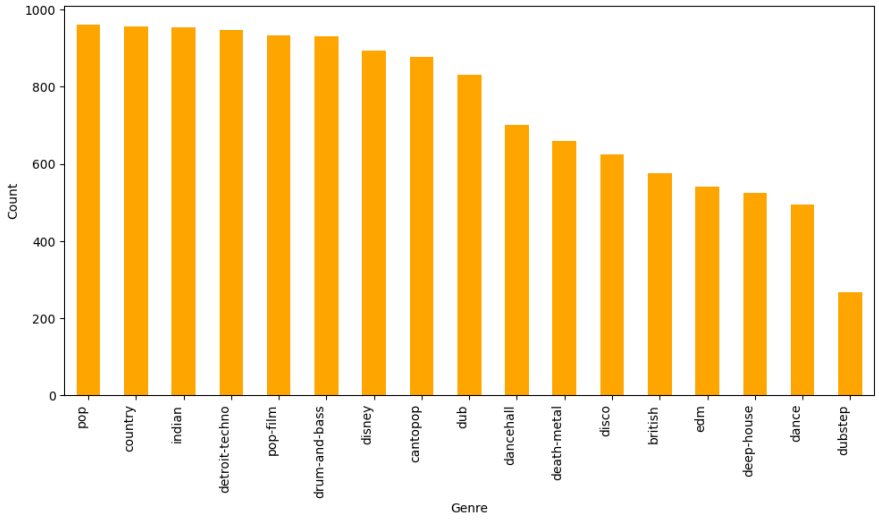


Figure 2: Less present genres distribution

- **n_beats, n_bars, time_signature**: about these three variables we can extract interesting informations, in fact in 156 cases they have, jointly, a value equal to **0**; while in the majority of these cases (140) the value can take on meaning, for example songs belonging to the "sleep" or "ambient" genres, which have no rhythm and are mostly composed of **background noise**, in some of them they seem to represent **missing values**, since having a value of 0 in the "*n_bars*" column in genres such as "show-tunes" seems unrealistic, given that this genre is often associated with pop.

2.1.1 Duplicates

In order to check for **duplicates** within the track dataset, we first checked the values of the "*id*" column, which showed that there were 33674 rows in our data with **non-unique** values for this variable. We then decided to eliminate all redundant rows by

keeping only one row for each "*id*" within our dataset, so as to make it unique as specified in the variable description, at that point our dataset went from 109547 to 89562 records.

We then extended the control to a first subset of columns: ["*album_name*", "*track_number*", "*artists*", "*disc_number*"], from this we realised that our dataset still contained 600 rows with **repeated** values between them, so by proceeding with the same logic as before, i.e. keeping only the first occurrence of each subset combination, we were able to eliminate a further 305 rows, bringing the total to 89257.

We then differentiated the search by using a second subset, larger than the first, which contained ["*name*", "*artists*", "*duration_ms*", "*danceability*", "*energy*", "*loudness*", "*key*", "*mode*", "*acousticness*", "*speechiness*", "*liveness*", "*instrumentalness*", "*valence*", "*time*", "*time_signature*"], with these parameters we were able to find another 8248 **duplicate** rows, which once analysed and eliminated allowed us to obtain the final dataset, containing **83612** records.

2.2 Artists

This second dataset we will discuss is **smaller** than the previous one, both in terms of the number of records (in this dataset we have 30141 rows) and the number of **dimensions**, in fact we only find five columns: "*id*", "*name*", "*popularity*", "*followers*", "*genres*".

- **id**: also in this dataset we find this column, in this case the code **identifies** an artist but semantically is identical to the variable in the track dataset.
- **name**: it's the name that the artist uses for its publications. Every artist in the track dataset, even those who appear only once as a featuring, is present in this dataset, except some minor **syntactic-related** error.
- **popularity**: value from 0 to 100 that describes the popularity of the artist, the value is extracted from the artist's songs.
- **followers**: number of followers that an artist has on Spotify.
- **genres**: the **list** of genres of songs in which an artist appears. In this dataset, unlike the tracks dataset, the column has more than one value, so each record has an **array** of genres characterising each artist. We first noticed that the genres in the two tabular did **not** coincide, so we opted to replace those in the artists dataset to make them match and thus make much more sense of the content. To do this, we **cross-referenced** the data from the two datasets so that we could also fill in those rows where an artist's genre array was empty. Once all the rows were filled in, we performed **transactional clustering** (using hierarchical clustering with the Jaccard similarity and Complete Linkage) on the lists of genres as, for future tasks, it might be very useful to **reduce** the total number of genres; from Figure 3, one can clearly identify some clusters especially in the **extremes**, while in the middle one cannot quite grasp the various connections.

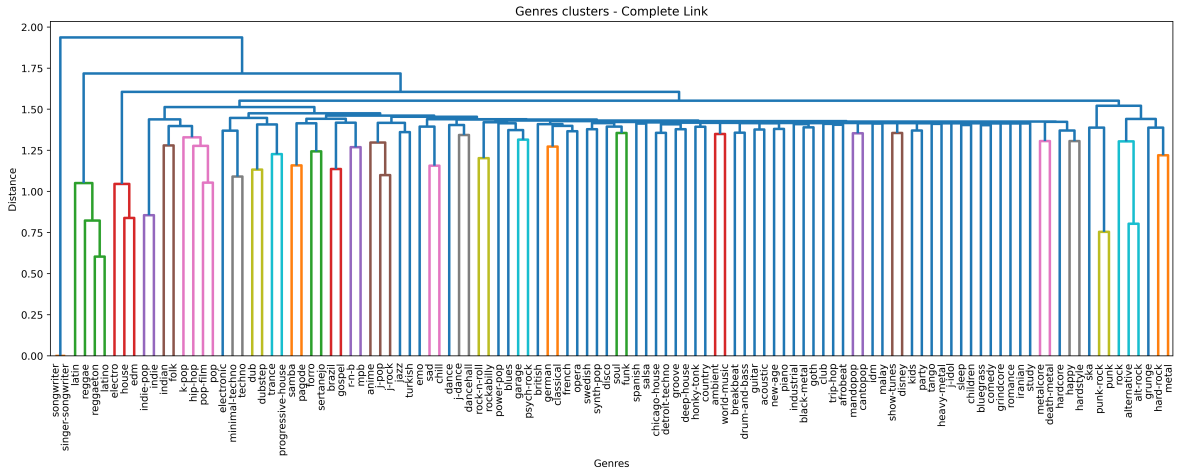


Figure 3: Genre transactional clustering

2.2.1 Duplicates

In order to check for duplicates, we followed a very similar procedure to that of the previous dataset, in fact we first checked for redundant values in '*id*', finding only **two** rows that shared the same value, so we eliminated the one too many; we then went on to check whether the same thing occurred for the names, however we found 215 matches, which we decided to keep because we do not know whether they are actually duplicates or are simply cases of **homonymy**, also because the other columns had **different** values.

2.3 Time Series

As far as time series are concerned, we proceed with the analysis starting with **duplicates**. We noticed that there are **271** duplicates for the Id, which probably originate from songs already duplicated from the tabular dataset. In addition, there are 3 pairs of time series that despite different Ids contain identical values. Therefore, after noticing the existence of all these duplicates, the size of the time series dataset was reduced to **9861** rows.

The **genres** are proportionally distributed among the various time series, obviously after the elimination of duplicates the proportion changed slightly but still remained **balanced**.

As can be seen in Figure 4, which represents the **average** time series for each genre, the numbers have a **semantic** value as genres such as 'piano' and 'new-age' are those with the lowest average and instead other genres such as 'happy' and 'j-idol' are those with the highest average values. Although most genres have similar values between them, we still consider it correct **not** to normalise the time series as the distances between them can have an impactful **meaning**. In this way, we can proceed with our analysis focusing mainly on this differences in **values**, while shapes and patterns will be taken into account when discussing shapelets and motifs.

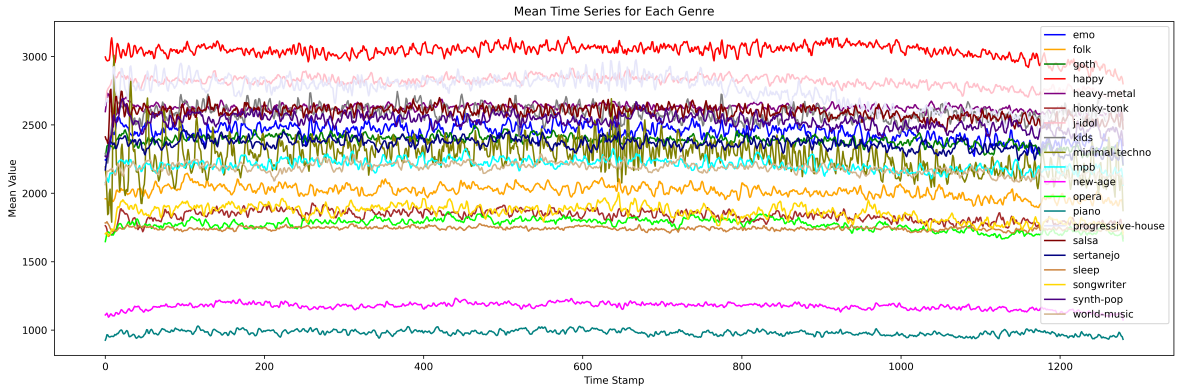


Figure 4: Mean time series for each genre

As the time series dataset is very **voluminous** in size, we found it necessary to **approximate** the time series with PAA down to 1/5 of their original length, thus bringing them from a length of 1280 to a length of 256, for all operations of: K-Means, DBSCAN, Silhouette Score calculation, KNN and Shapelets Extraction. For mini-ROCKET only, given its low time complexity, we could approximate the time series by half, reducing them to 640 timestamps. Given the **time complexity** of DTW, KNN and the shapelet extraction algorithm, applying a finer approximation would be too time-consuming and would bring negligible gain in results. Whenever possible, to speed up hyper-parameter tuning processes, it was useful to use a **precomputed** distance matrix between all time series. Instead, motifs and anomalies were extracted from the **original** time series length. For most operations requiring the calculation of distances, we used a **DTW** with a Sakoe-Chiba constraint of 10%.

3 Time Series Analysis

3.1 Classification

In the following section, we will see how various **classification** models behave: KNN, Shapelets Classification and Rocket. The analysis will be led using "*genre*" as **target** variable.

3.1.1 KNN

For the first classification task we decided to **split** our dataset, using 70% of it as train set and the remaining 30% as test set. Then we've done our work using two different metrics: Dynamic Time Warping (DTW) and Euclidean distance. In both cases we performed a 10-fold cross validation on the train set only to identify the correct number of **K** to use.

DTW First thing we've observed is how the accuracy of the model changes when the value of K increases, from that we got that the best parameter is set at K = 14 and the **highest** accuracy is returned by the use of the metric that **weighs** distances (weights='distance'). In this case, the classification report returned an accuracy of **0.30**.

Euclidean For Euclidean metric we've done the same type of research, getting that K = 1 was the best parameter, which had an accuracy of **0.12**. In our opinion, the performance of the model using Euclidean distance is not satisfactory. Also we have observed that the accuracy goes down **significantly** by choosing a value of K greater than 1, and the overall model results increase slightly if a **uniform** weight is used for the nearest neighbours.

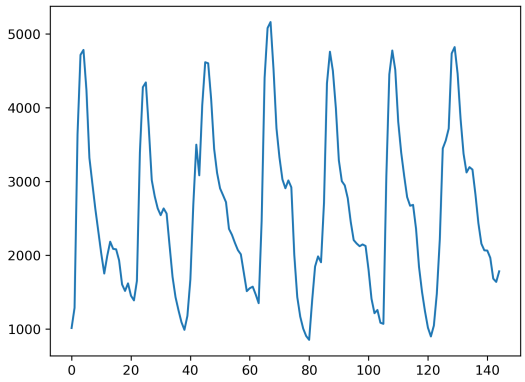
Given the results of the two KNN-based models, we decided to give more consideration to the classifier implemented with **DTW**. The results and considerations below are the result of an analysis using **f1-score**. Among the best-classified genres, the following stand out in **descending** order of f1-score: "sleep", "piano", "minimal-techno", "new-age" and "happy". Of these five, all except "minimal-techno" are distributed **marginally** more than the other time series, as can be seen in Figure 4. Therefore, it can be seen that the f1-score is not constant from genre to genre, in fact there are very **poorly** ranked genres such as the following in **ascending** order: "goth", "mbp" and "emo".

3.1.2 Shapelets

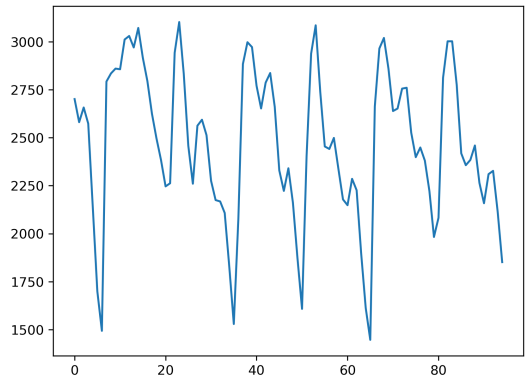
For the shapelets, we **approximated** the time series as previously mentioned, i.e. compressed to a length of 256. Once the time series were compressed, we split the dataset into train and test sets with a ratio of 70/30. Then from the train set we randomly extracted a maximum of **1000** shapelets from a pool of 10000 shapelets. We then transformed both the train and test set and applied **KNN** and **Decision Tree** to them.

The extracted shapelets are **998** and have an average length of 71 (355 when compared to time series of original length) and a standard deviation of 50 (250). However, the highest **Information Gain** is only 0.079, with an average of 0.012. The shapelets with a higher average Information Gain (0.06) are from the 'minimal-techno' genre, followed by shapelets from the 'happy' genre (0.033). On the other hand, the **least** represented genres are 'folk', with shapelets having an average Information Gain of 0.002, and 'goth,' also with an average of 0.002.

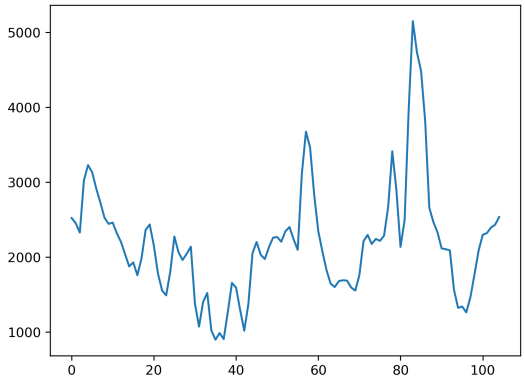
This discrepancy could also be observed directly from the **shapes** of the shapelets. We took as an example, in Figure 5, the shapelet with highest Information Gain for the genres mentioned before. In order not to report the approximate shapelets, we **manually** extracted them from the original length time series, using the information provided by the shapelet extraction algorithm. According to our observations, the shapelets in Figures 5a and 5b, have a much more **discriminating** shape than those in Figures 5c and 5d, which seem much more **broad**.



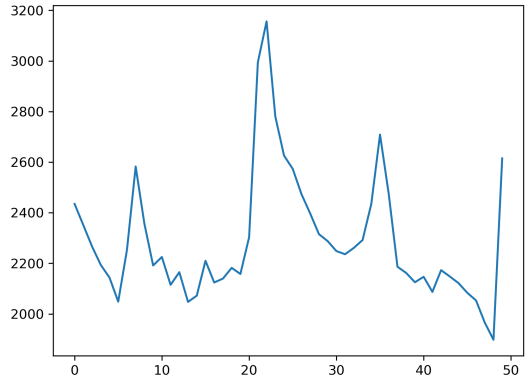
(a) 'Minimal-techno' shapelet



(b) 'Happy' shapelet



(c) 'Folk' shapelet



(d) 'Goth' shapelet

Figure 5: Shapelets examples

KNN Again, as with the KNN used previously, we opted for 10-fold cross validation on the transformed train set. The accuracy this time is **oscillating** as K changes. We therefore set this parameter to 20, which is one of the lowest K but with best accuracy. The performance on the test set reports an accuracy of **0.26**.

Decision Tree For this classifier, we performed a **grid-search** on a wide range of parameters. Then for each parameter set, a 10-fold cross validation was applied on the train set. The result obtained suggests that the best parameters are the following: 'max_depth'=10, 'min_samples_split'=100, 'min_samples_leaf'=30 and 'criterion'=gini. After applying these parameters on the test set, the accuracy is found to be **0.19**, thus **lower** than KNN.

Regarding Shapelets' KNN, the genres that were classified better and worse are more or less the **same** as in the paragraph on the simple KNN. An interesting observation in our opinion is that model's performance on the various genres are **amplified** compared to the simple KNN, getting worse for genres that had lower f1-score and better for those that had higher f1-score. For example, "minimal-techno" has an f1-score of 0.68 and "goth" of 0.03. In fact, "minimal-techno" and "goth" not surprisingly feature sets of shapelets with higher and lower Information Gains respectively. This also applies in a **gradual** manner to all shapelets of the other genres.

On the other hand, for the Decision Tree, having a very poor performance, the values are much **lower** than for the KNN. The f1-score values are generally lower for all genres, in particular "folk" and "mpb", which score 0.00 and 0.01, respectively.

3.1.3 mini-ROCKET

As the last **classifier**, we decided to use the "MiniROCKET". In this algorithm the most important parameter required is the number of **Kernels**, to choose the value that is optimal for us we performed a 10-fold cross validation using a list of possible Kernel values ranging from 100 to 15000, testing for each of them the performance of the model. From the analysis we extracted that as the number of Kernels increased, the model **slowly** improved, however, also the time cost was gradually **increasing**, so we decided to opt for 3000 Kernels, which returned us an accuracy of 0.43, a fair **compromise** noting that using 15000 we would get an accuracy of 0.44, but in much more time.

3.1.4 Conclusions on classification

In summary, after testing different classifiers with different train inputs, we may claim the one that performs best in classifying time series in genres is **mini-ROCKET**, comparable with the others in Table 1.

Method	**Parameters**	**Accuracy**
KNN with DTW	k=14, weights='distance'	0.30
KNN with Euclidean	k=1, weights='uniform'	0.12
Shapelets KNN	k=20, weights='distance'	0.26
Shapelets DT	md=10, mss=100, msl=30, gini	0.19
mini-ROCKET	kernels=3000	0.43

Table 1: Classification performance summary

In addition, mini-ROCKET is also extremely **efficient** in terms of time complexity, it is extremely fast in computation especially when compared with the others, because the KNN ($O(n^2)$) only performs well if the distances are calculated with DTW (also $O(n^2)$), which becomes **extremely slow** overall, and the shapelet extraction algorithm needs to calculate the information gain for each shapelet, which moreover are extracted **randomly**, making it **less reliable**.

To analyze the performance of the various classifiers a little further, we tried to calculate the **ROC curve** for each of them. Since the curve requires a **binary** classification task, we opted for the 'minimal-techno' genre against all others. We chose this particular genre because it is one of those that is always classified best by the various models, but unlike others such as 'piano' or 'happy,' its time series do not have an average value too **dissimilar** from the rest (Figure 4), which means it is easily classified probably because of its **distinctive shape**.

The curves can be observed in Figure 6. Also examined is the 'Anomalies' classifier, which is a Decision Tree similar to that used for shapelets, but with a set of **anomalies** instead. We will discuss this more in Section 3.3.2.

In general, it can be seen from the ROC curves in Figure 6 that most of the models effectively manage to correctly classify the 'minimal-techno' genre, and that the results obtained on this genre individually do not precisely **reflect** the overall performance of the various models. For example, the KNN based on shapelets has the largest AUC despite being just the third best model in terms of overall accuracy. In our opinion, the results are also quite biased by the **unbalanced** distribution of the genres (1/20).

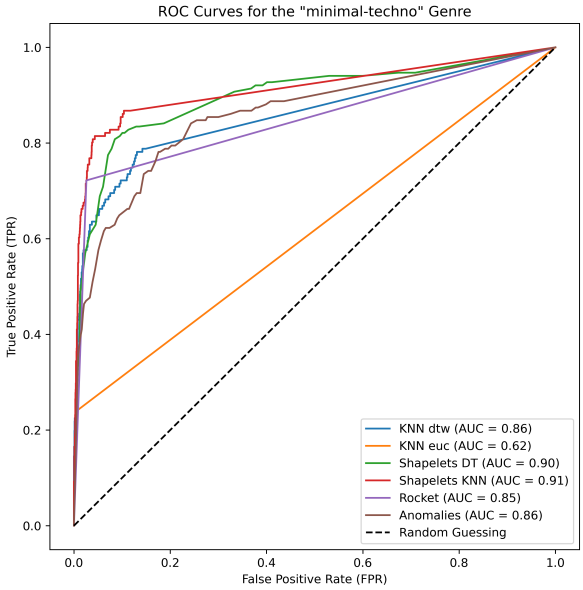


Figure 6: ROC curve for the 'minimal-techno' genre

3.2 Clustering

3.2.1 Features-based

For clustering time series, we made several attempts. The first of these is **Features-based** Clustering, for which we transformed the dataset into a feature set for each time series. The features we considered are: '*max*', '*min*', '*mean*', '*variance*', '*std*', '*autocorrelation*' and '*kurtosis*'. With these variables we implemented the **K-means**. We then searched for the most appropriate number of k using a classical elbow method, which suggested setting k to 6. The Silhouette Score in this case is equal to **0.27**.

We then calculated the **distribution** of genres for each cluster, which can be seen in Figure 7. It can immediately be seen that in cluster 3 there are almost exclusively songs from 'new-age', 'piano', 'opera' and 'sleep'. This is not surprising, as it can be seen from Figure 4 that the average values of these three genres are **isolated** from the rest. Furthermore, within cluster 0, there are only a portion of songs from the 'sleep' genre. Subsequently, it can be noticed that clusters 1, 2 and 4 are the most populated of all, but possess a very **heterogeneous** distribution of genres. Finally, cluster 5 has a high concentration of only 3 genres, which, however, do not have much in common semantically.

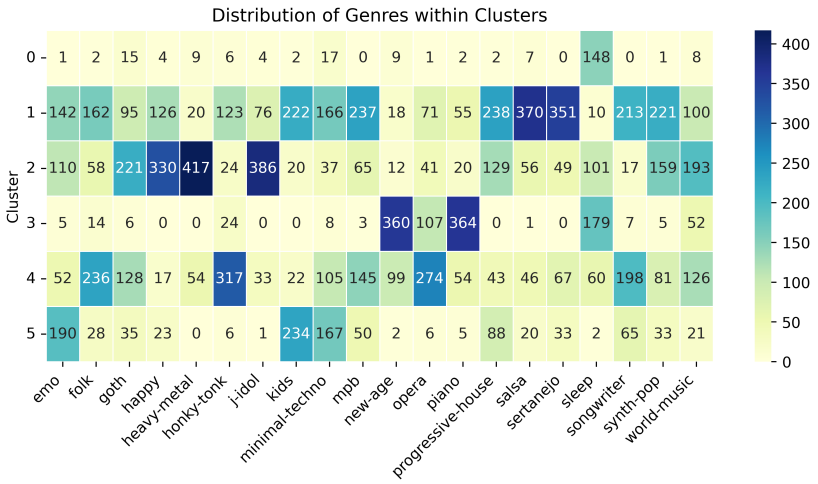


Figure 7: Distribution of Genres within Features-based Clusters

3.2.2 K-Means

After testing an implementation of K-Means through a Features-Based approach, we tried to apply it on time series using **DTW** as already mentioned in Section 2.3. We re-used the same approach as in the previous section, so the best k is the one set to 7. The Silhouette Score in this case is **0.13**, far worse than the previous K-Means.

We again calculated the distribution of genres by cluster, which can be seen in Figure 8. In general, we can observe that, despite the lower Silhouette Score, the distributions are very **similar** to the previous ones. One substantial difference is that this time it is a portion of the genre 'happy' that is isolated in cluster 4.

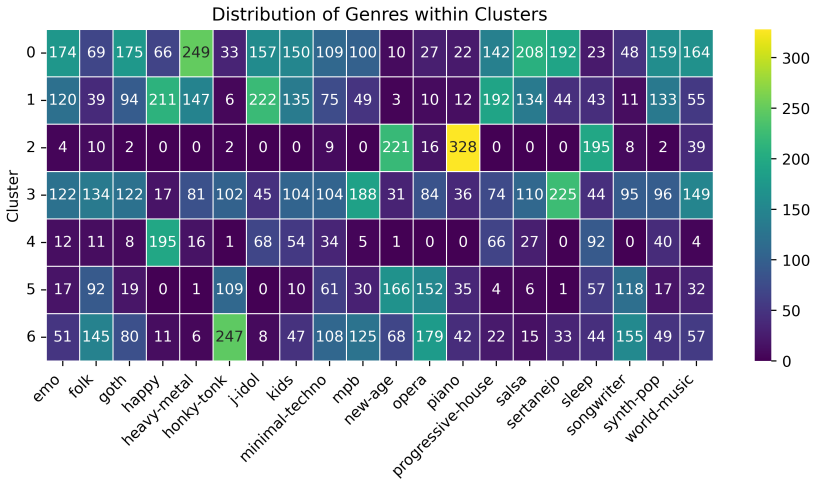
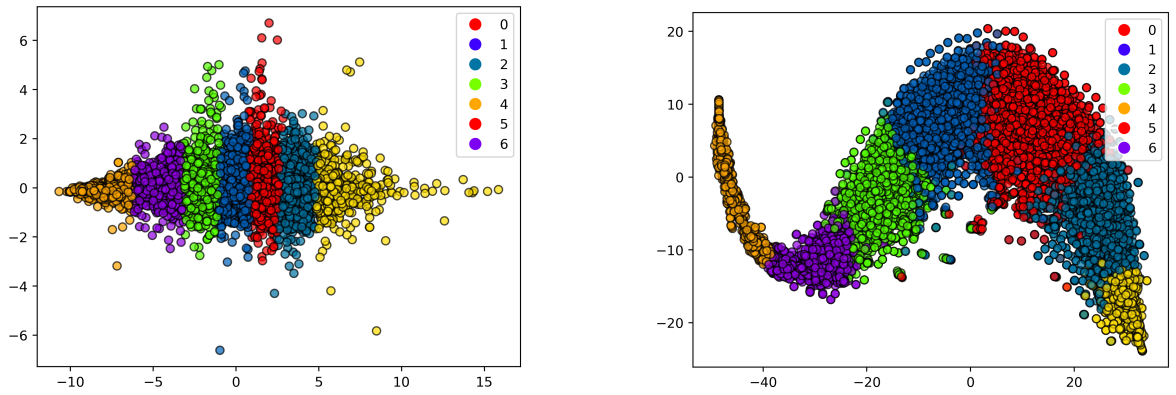


Figure 8: Distribution of Genres within K-Means Clusters

Finally, in order to visualise the dataset in **two dimensions**, we used two dimensionality reduction techniques, namely **PCA** and **t-SNE** (observed in Figure 9a and 9b). As can be seen from these images, the shape of the clusters is completely different. However, both techniques manage to create a certain **separation** between clusters while avoiding the presence of **overlaps**.



(a) PCA (b) t-SNE

Figure 9: K-Means 2D visualization

3.2.3 DBSCAN

With regard to **DBSCAN**, we applied a large grid-search to find the best 'eps' and 'min_samples' parameters. However, we noticed that for any pair of values there are a large number of **noise points**, so we opted for a compromise between the Silhouette Score and the noise ratio, which should be when 'eps' is equal to 3500 and 'min_samples' to 200. In this particular case, the Silhouette score is equal to **0.67**. However, **73%** of all time series are classified as noise points.

Looking at the distribution of genres in Figures 10 (where we removed the cluster with noise), we can see that the distribution of genres in both clusters is very **heterogeneous**. However, we can again see in cluster 0 a high concentration of genres that, in our opinion, might **resemble** each other.

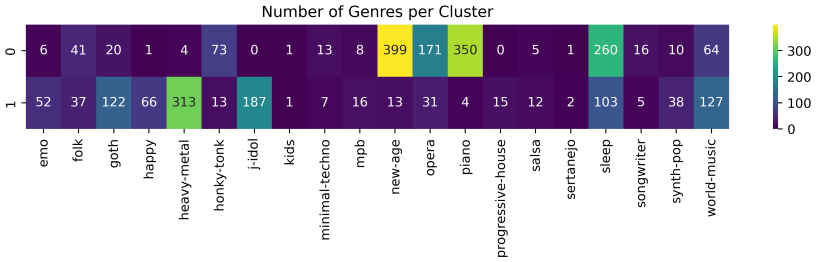
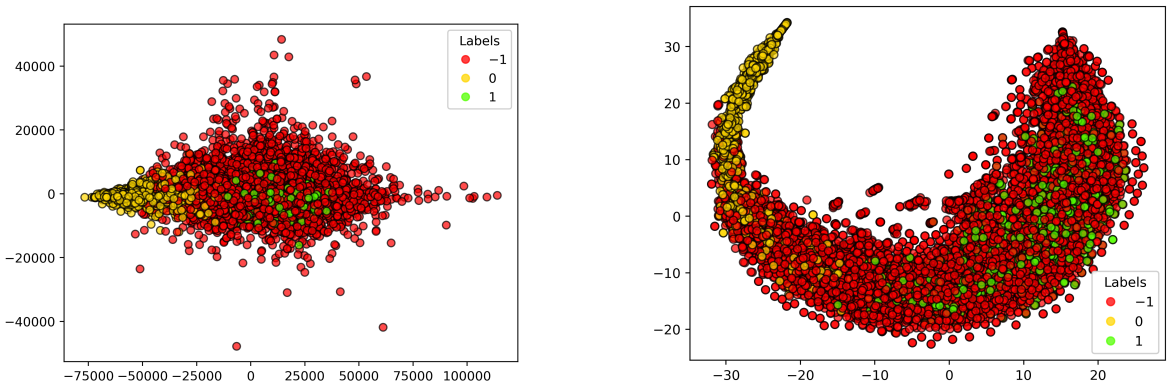


Figure 10: Distribution of Genres within DBSCAN Clusters

As with K-Means we used both PCA and t-SNE to visualize the dataset, observable in Figures 11a and 11b. In this case, in our opinion, the results are **less descriptive** because many noise points are still present. Also, according to us, the concept of density loses '**significance**' when moving from a multi-dimensional to a two-dimensional representation.



(a) PCA

(b) t-SNE

Figure 11: DBSCAN 2D visualization

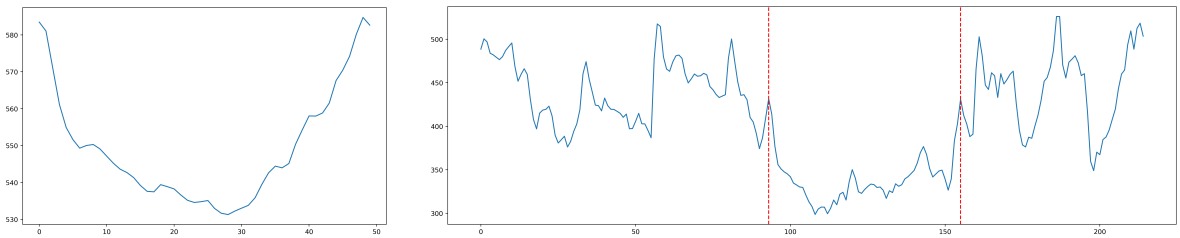
3.3 Motifs and discords

3.3.1 Motifs

Moving on to motifs discovery, we implemented two similar methods but with different inputs to find **representative** motifs. In the beginning, we extracted the top-3 motifs of length 50 from each time series of original length, after which, in order to find a small set of motifs representative enough of one or more genres, we tried applying K-Means clustering on all motifs with k=20: ideally, if one genre is easily **distinguishable** from the rest using the motifs, we should have a totally **pure** cluster with respect to the genres contained within. Then, we calculated the **impurity** of each cluster with **Misclassification Error**, and the result is that, although the majority of clusters are almost totally impure, there are some clusters with few genres inside. Among them, a cluster with a support of 0.08, and an impurity of **0.41**, definitely stands out: the majority of motifs within come from time series of genre 'piano,' but the rest are still from similar genres (value-wise), such as 'sleep,' 'new-age,' and 'world-music.

Next, we calculated the **centroid** of this cluster (Figure 12a) since it should be quite a representative motif of the 'piano' genre. Then, we tried to compare it with the **shapelet** with higher Information Gain from a time series of genre 'piano'. This shapelet, in Figure 12b, just as was done at the beginning of Section 3.1.2, was manually extracted from the original length time series to eliminate the effect of approximation and to **compare** it more accurately with the motif.

Obviously the two time series in Figure 12 have two different lengths, but if we only consider a section of roughly the same length as the motif for the shapelet, we can see a certain **similarity**. In addition, the shapelet also appears to be much more detailed and with a wider range of values than the motif, but this is due to the nature of the latter, being a centroid and therefore an average of many more time series.



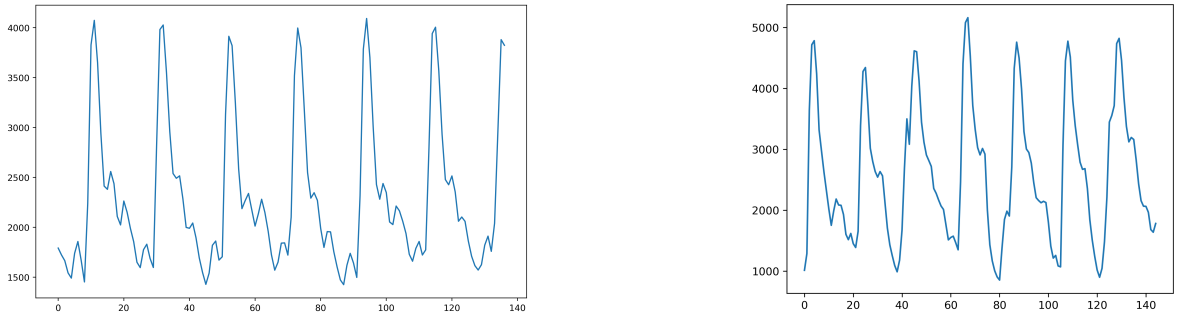
(a) Motif (b) Shapelet

Figure 12: 'piano' genre comparison

In order to find another example of a representative motif, we tried the same method, but this time we performed K-Means clustering with K=20 on the **unapproximated** time series, calculating distances using DTW with a Sakoe-Chiba constraint of 10%. Then, as done before, we calculated the **impurity** for each cluster and decided to consider not one, but two small clusters with a total support of only 0.005 (58 time series out of a total of almost 10,000), but with an impurity of **0.03** and 0.2. In both, only two genres are present: 'minimal-techno' for the most part, and 'progressive-house' for the rest.

Again, we calculated the centroid of the cluster resulting from a **merge** of the two small clusters. The result is thus another time series, from which we have extracted the top motif, in Figure 13a, possibly **representative** of the 'minimal-techno' genre. To compare it, we took the shapelet with the highest Information Gain from a time series of the same genre, which fortunately also has the same original length as the motif. They are compared in Figure 13.

Although the range of values of the shapelet is again slightly wider, the two time series are undoubtedly very **similar**, suggesting the fact that genres such as 'minimal-techno' and 'progressive-house' are easily **recognisable** from the shape and **patterns**.



(a) Motif (b) Shapelet

Figure 13: 'minimal-techno' genre comparison

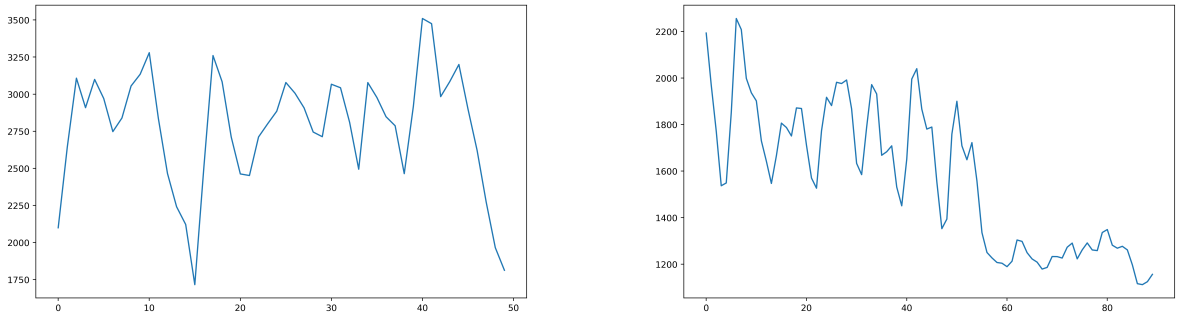
3.3.2 Anomalies

With regard to anomalies, however, we used a different approach to try to find one or more **discriminative** anomalies of a genre. First, we selected a stratified sample of 1000 time series (1/10 of the total), and from each of these we extracted the top anomaly. Then, as is done with shapeles, we transformed the dataset of time series

into a dataset of **best-matching** distances from each time series to each anomaly (10,000x1,000 approximately).

In order to compare anomalies with shapelets in a different way than with motifs, we used this new **transformed** dataset for a **classification** task using a Decision Tree. We found the best parameters in the same way as we did in Section 3.1.2, and according to our analysis they are: 'max_depth'=10, 'min_samples_split'=15, 'min_samples_leaf'=30 and 'criterion'=gini. The classification results are not bad, especially when compared to the Decision Tree using shapelets, as the accuracy here is **0.30** (compared to 0.19). Moreover, the **ROC curve** of this classifier with respect to the 'minimal-techno' genre can be observed in Figure 6, at the end of Section 3.1.

To find one or more of the most discriminative anomalies, we used the Decision Tree function that returns the **importance** of each feature, and since our features, in this case, correspond to anomalies, we considered the anomaly that had the highest importance (i.e. the most **discriminant**). This anomaly comes from a time series of genre 'new-age', and it is compared with the 'new-age' shapelet with higher Information Gain, and both are observed in Figures 14. Again, as with the motifs, there is a slight **similarity** in the shapes between these two time series, especially if we consider only the first half of the shapelet in Figure 14b, that is, only the part of the same length as the anomaly in Figure 14a.



(a) Anomaly

(b) Shapelet

Figure 14: 'new-age' genre comparison

3.4 Conclusions on Time Series

In conclusion, we can say that, as a result of our analyses focused primarily on the genres of each song, the results were **not** particularly satisfactory: despite the absence of normalizations, which allowed us to gain some **information** from the original values, there are only a few genres that are easily **distinguishable** from the rest (such as 'happy,' 'new-age,' or 'piano') or that are related to others in an understandable way. Approximating the Time Series meant that a lot of information was **lost**, but without it, the operations would have taken far more time but with **negligible** gains in the results; few genres are also easily distinguishable from the shapes and patterns even **visually** (such as 'minimal-techno'), and thanks to calculating the distances with DTW they manage to maximize these results. In any case, in our opinion, it is likely that, with many **more** instances, very efficient classifiers such as MiniROCKET would have achieved far more **satisfactory** results, and, possibly, some other **representative** motif of a genre would have been found.

4 Advanced Data-Preprocessing

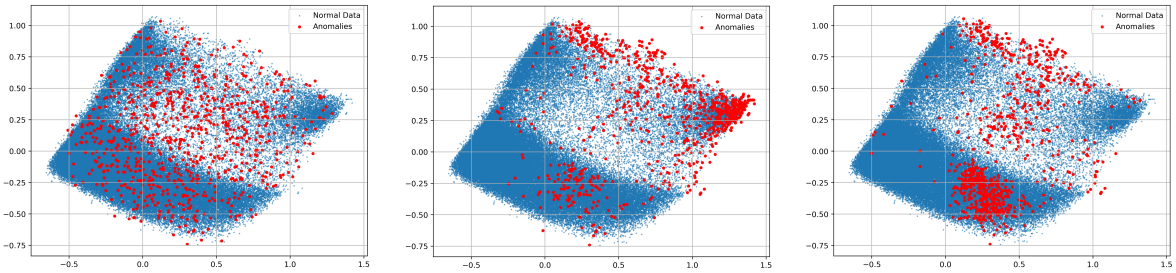
4.1 Outliers detection

To identify outliers possibly present in the track dataset, we considered a **subset** of variables. This is because attributes such as '*album_total_tracks*' and '*disc_number*' certainly have many outliers (e.g., albums with hundreds of songs or with dozens of discs), but in our opinion, it is more interesting to try to identify outliers based on more **meaningful** variables. Therefore, we considered: '*duration_sec*', '*popularity*', '*danceability*', '*energy*', '*loudness*', '*speechiness*', '*acousticness*', '*instrumentalness*', '*liveness*', '*valence*', '*tempo*', '*fade_out_%*' and '*n_beats*'. Of these, four variables ('*duration_sec*', '*loudness*', '*tempo*', and '*n_beats*') were normalized with a min-max scaler, since they are the only ones with values not between 0 and 1, while '*popularity*' values were simply divided by 100.

We used three methods, chosen because they are among the most **suitable** for identifying outliers in a **high-dimensional** dataset: ABOD, Isolation Forest and K-Means. Recall that, following the removal of duplicates, our dataset contains **83612** records, and since our goal here is to identify the top 1% outliers, we will consider only **836** records for each method.

For ABOD, since its time complexity is $O(n^3)$, it was **necessary** to use the 'fast' algorithm, which approximates using k-nearest-neighbors, which we set to 5. For Isolation Forest, we trained and tested 100 decision trees over the **entire** dataset, while for K-Means, we clustered the dataset into an optimal K of **clusters** (7), which we identified previously based on the SSE. The first two methods, in their own way, return a **decision score**; for K-Means, we considered, for each record, the **distance** between the point and the nearest centroid (the higher, the more **abnormal**).

We reduced the number of variables to 2 using PCA to **visualize** the outliers in a two-dimensional scatterplot, observable in Figure 15. In our opinion, the concept of outliers loses some **meaning** when the number of variables is reduced; in fact, we could say that only in the 15b graph of the Isolation Forest are the outliers somewhat **visible**.



(a) ABOD (b) Isolation Forest (c) K-means

Figure 15: Outliers in 2D using PCA

Regarding outliers as **songs**, they mainly come from genres such as 'sleep', 'comedy' and 'iranian', but unfortunately, following much analysis, we could not identify why they are considered anomalies. We also tried to conduct **pattern mining** analysis on both the set of inliers and the union of the three sets of outliers, to look for possible **differences** in patterns. At the end of the analysis, we found that the itemsets with

higher support in the outliers were **also** present among the inliers with relatively high support.

Considering that we have 3 sets of 836 records each, only **61** songs are considered outliers by **all** three methods. Among them, except for a few non-songs (i.e. 'sleep', 'ambient' or 'world-music' songs with no time signature, no beats and extremely low DB) and a few anomalous mismatches among the variables (i.e. 'iranian' songs with very low '*loudness*' and very high '*energy*', or 'sleep' songs with extremely high '*danceability*'), we did **not** find a strong and/or general **reason** why they were calculated as outliers. In any case, we consider it more appropriate to **remove** from the dataset only these 61 records on which all three models **agree**, as the number is so low that it will not impact future analyses (0.0007%)

4.2 Imbalanced learning

Our track dataset is already very **unbalanced** if we consider one genre versus all the others. For example, if we were to consider only the 'minimal-techno' genre (for consistency, since it was used in the previous section), the percentage of songs from this genre would be **0.8%**, which is perhaps a bit too few samples. Therefore, using the transactional clustering calculated in Section 2.2 and displayed in Figure 3, we decided to **merge** 'minimal-techno' with 'techno'. In this way, the percentage of samples for the minority class is **1.85%**.

Thus, our classification task is to be able to predict the macro-genre 'techno'. For this, we took into consideration the following variables: '*popularity*', '*danceability*', '*energy*', '*speechiness*', '*acousticness*', '*instrumentalness*', '*liveness*', '*valence*', '*fade_out_%*', '*tempo*', '*loudness*', '*duration_sec*' and '*n_beats*'; and normalized with min-max scaler the last four in the list.

We split the dataset into train and test set with a ratio of 70/30, so as to apply the **Undersampling** and **Oversampling** algorithms on the **train** set, and **verify** the performance on the original test set. For all models, we fitted a 10-fold cross validation on the train set to look for the optimal parameters for the Decision Tree. First, we tested the performance with the **original** (imbalanced) train set in order to compare the results of Imbalanced Learning models.

For the Undersampling methods, the only one we tested was RandomUnderSampling, as Tomek Links, EditedNN and CondensedNN removed **too few** samples from the population, leaving the dataset still too **imbalanced**. Among the Oversampling methods, we tested RandomOverSampling, ADASYN and SMOTE, but the latter reported very **similar** results to ADASYN, so we did not consider it. Finally, we tried changing the minority class **weights** in the same Decision Tree used for imbalanced classification. The parameters and results of each classifier can be observed in Table 2.

Method	Parameters	Precision	Recall	f1-score
Imbalanced	md=8, mss=10, msl=65, entropy	0.62	0.35	0.45
RUnS	md=20, mss=5, msl=50, entropy	0.08	0.89	0.15
ROvS	md=50, mss=65, msl=5, entropy	0.33	0.51	0.40
ADASYN	md=20, mss=30, msl=5, entropy	0.17	0.67	0.28
Class Weight	md=8, mss=10, msl=65, entropy	0.48	0.45	0.47

Table 2: Imbalanced Learning performance summary

We also calculated **ROC curves** for the various models, and compared them with the imbalanced model in Figure 16. From the comparison of the ROC curves it can be seen that the model with the **highest** AUC is RandomUnderSampling, however despite managing to achieve a remarkably high TPR, the resulting precision is very low, showing that the model tends to **missclassify** the negative (non-techno) class too much. Also, it is important to specify that, due to the nature of RandomUnderSampling, this Decision Tree was trained on an **extremely small** train set, making it probably much **less reliable**. RandomOverSampling and ADASYN are the ones that report the **worst** AUCs and, in general, show the fact that, in our case, Oversampling models are **not** very effective in resolving this binary classification. In the last method used, it was not necessary to balance the train set distribution, as we **increased** the error weight for the positive class to 3, thus finding a compromise between an excessively high weight that maximizes Recall at the expense of low Precision.

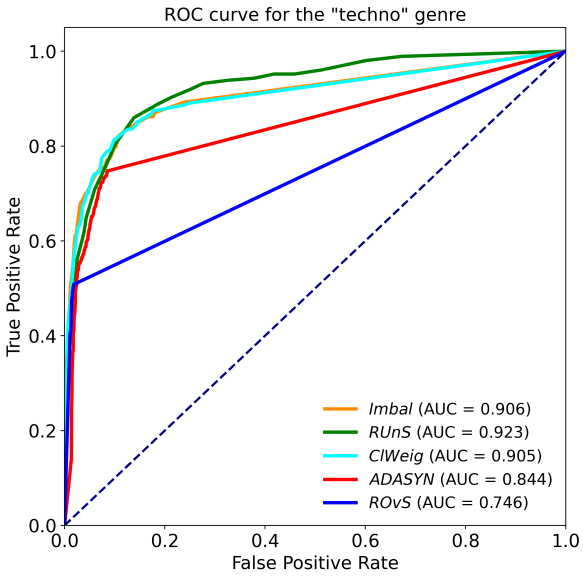


Figure 16: ROC curve for the 'techno' genre

5 Advanced Learning Techniques

5.1 Advanced classification

Regards to the Advanced Classification section, we performed several tests on different classifiers and observed the results based on the features we give as input. In general, for all classifiers, we split the dataset in 70/30, and we took into account all variables in the dataset, normalising continuous variables with **MinMax Scaler** and transforming categorical variables with **One-Hot Encoding**. First, we tried solving the **Imbalanced Learning** task defined in Section 4.2. Next, we tried targeting song genres by grouping them into **macro-genres** (see Section 5.1.2). Finally, we tried targeting songs linked to their **time series**.

5.1.1 Advanced Imbalanced Learning

Recall that the Imbalanced Learning task was to target the macro-genre 'techno', which includes the genres 'techno' and 'minimal-techno', against all others, with a minority class percentage of **1.85%**. The difference here is that we will not use any Imbalanced Learning techniques but train the classifiers on the original **imbalanced** dataset.

Ensemble Methods As a first classifier, we used Random Forest, trying to increase the number of **estimators** and the number of **variables** used. The best result (f1-score = 0.38 for the positive class) is achieved with 150 **Decision Trees** and with 50 features. AdaBoost achieves an f1-score of 0.37 with 200 **stumps**, while the result increases **very little** if more complex Decision Trees are used (f1-score = 0.38). Bagging, on the other hand, achieves the best performance of all (f1-score = 0.49) with 150 Decision Trees and 50 features per tree.

Support Vector Machine As far as SVM is concerned, the results are **less satisfactory** compared to the other classifiers. Consequently, we increased the regularization parameters up to 100 to see what the best results the model could achieve, even at the cost of **overfitting**. Therefore, with an 'rbf' kernel and a gamma set to 1, the f1-score of the positive class reaches 0.27. By contrast, with a **more regularized** model (C = 1), the model reaches an f1-score of 0.11.

Neural Network With regard to Neural Networks, we tested many structures relying on various combinations of parameters. However, the model we find **most efficient** is the one with: an input layer consisting of as many neurons as there are variables, three hidden layers with 128 neurons per layer and 'relu' as an activation function and an l2 regularisation of 0.0001, an output layer of 2 neurons (1 per class) and a Dropout of 0.5. In this case the result seems satisfactory, f1-score = 0.45.

LightGBM We used the LightGBM classifier from the GradientBoosting family because it is extremely fast during the training phase and can handle large datasets well. We noticed that using more than 100 GBDT and more samples per bin than there are in the train set does not increase performance, in fact they remain stable with an f1-score = 0.47.

Logistic Regression In this case, we noticed that the best performance is obtained with 'sag' as a solver and **not using** any kind of **penalty**, but very **strongly** regularising the model (C = 0.0001). However, the model in our opinion needs at least 300 iterations to converge, although in this case the model returns an f1-score of 0.33.

Table 3 shows a **summary** of the results of all top classifiers and their respective **parameters**. We can immediately see that the best classifiers are Bagging and LightGBM, but although the first manages to achieve higher precision, LightGBM is extremely **faster** and more memory **efficient**. We can also see that SVM, despite attempts to maximise performaces, remains the **worst**, as well as being extremely **slow**. As far as the neural network is concerned, despite all the structures tested, we could not find a more efficient model.

Classifier	Parameters	Precision	Recall	f1-score
Bagging	estimator=None, max_samples=1.0, n_estimators=150, max_features=50	0.80	0.36	0.49
SVM	kernel='rbf', gamma=1, C=100	0.36	0.21	0.27
Neural Network	hl=3, n_neurons=(128,128,128), activation='relu', kernel_regularizer=l2(0.0001), Dropout=0.5, input_dim=n_feature, output=n_classes	0.66	0.34	0.45
LightGBM	boosting_type='gbdt', n_estimators=100, subsample_for_bin=n_samples	0.65	0.37	0.47
Logistic Regression	C=0.0001, penalty=None, solver='sag'	0.58	0.23	0.33

Table 3: Advanced Imbalanced Learning performance summary

5.1.2 50 classes

For this classification task, we tried to **reduce** the number of genres, as many of them seemed **redundant**. To do this, we took into account the results of the hierarchical **clustering** performed in Section 2.2, as well as merging some genres based on musical **similarities** not captured by clustering. As a result, we went from a total of 114 genres to **50**.

Ensemble Methods For this type of classifier, we tried to use a Random Forest, looking for the best number of Decision Trees to maximise accuracy. Given the high number of features, we considered 50 features per tree and a total of 150 estimators. This model already allows us to arrive at an accuracy of 0.47. We calculated the **importance** of each feature, which can be seen in Figure 17, and tried to train the same model with only the **first 15** features in order of importance, considering only 'sqrt' features per tree, and this type of model manages to achieve the same accuracy (0.47). So, also for the next classifiers, we will try to verify the performance **both** with all features and with only these 15.

In addition, we also tried to use the techniques of Bagging and Boosting, but although we tried to **increase** the number of estimators to 200, they failed to achieve the results of the Random Forest. Boosting, in particular, even when using **more complex** Decision Trees, performs worse, which is due, in our opinion, to the fact that we target 50 classes, so each estimator missclassifies **many** genres, making Boosting probably lose **effectiveness**.

After these analyses, we tried to employ the model that works best (the first one, i.e. Random Forest) to perform a **binary** classification for each genre (*ovr*). We trained a Random Forest of 100 trees for **each class**, and then assigned each test sample the genre of the forest that returned the **highest probability** of belonging to that genre.

Despite its high complexity in terms of memory and time, it only achieved 0.49 accuracy, making it **less worth** than a simple Random Forest.

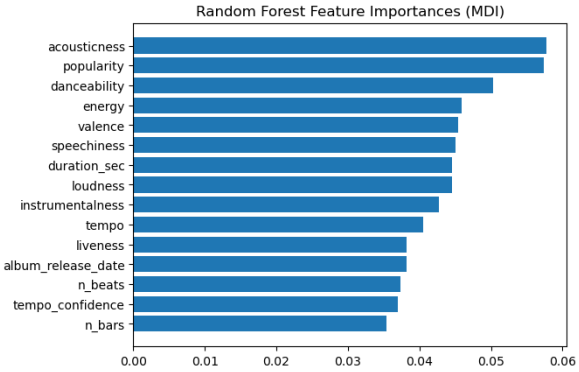


Figure 17: Feature importance of random forest

Support Vector Machine SVM is the **worst** of all classifiers, after trying several parameter combination tests we opted for gamma = 0.002 and an 'rbf' kernel. In this case unfortunately the accuracy is **not** satisfactory, being equal to 0.23.

Neural Network For Neural Networks, as in the previous section, we tested many structures to arrive at an **optimal** model. After several attempts, we arrived at a network comprising three hidden layers with 256, 128 and 64 neurons respectively, using the hyperbolic tangent activation function (tanh). Each hidden layer applies an L2 regularisation with a parameter of 0.0001 to prevent **overfitting**. In addition, we applied Dropout at 0.3 to increase **generalization**. In addition to the three hidden layers, the model has an input layer with a number of neurons equal to the number of features, and an output layer with a number of neurons equal to the number of classes. Therefore, by **increasing** the numbers of layers or the numbers of neurons per layer, the results **worsened** slightly while remaining relatively high. The best accuracy we were able to achieve with this model is 0.43. All attempts to construct a neural network were subjected to an **Early Stopping** with patience = 50, and we noticed that most of them managed to achieve a validation accuracy of around 0.40 within a **few tens** of epochs. Whereas, as can be seen from the Loss Curve of the selected model in Figure 18, our NN model manages to **stabilise** after approximately 20/25 epochs, on a validation loss of just over 2.2 .

For this model and for NNs in general, having a larger number of variables **increases** the performance of the model; we observed this when we tried to train the neural network **only** on the features found to be most important in the Random Forest classification, where we obtained a 0.05 **worsening** of the result compared to the model with **all features**.

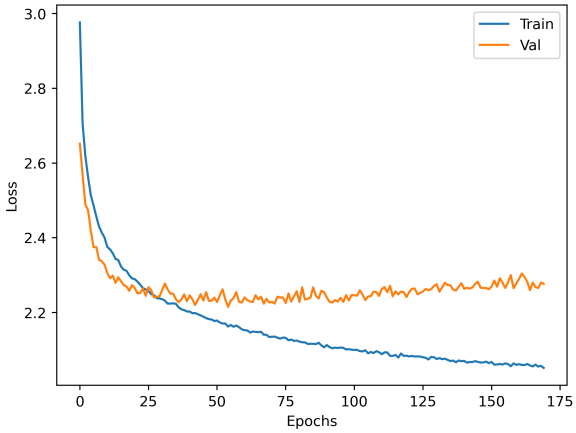


Figure 18: Neural Network loss curves

LightGBM In using LGMB, we noticed that, even leaving the **default** hyperparameters almost unchanged, the model proved to be **very efficient** in both classification and execution time. We were able to slightly increase the performance of the model by **increasing** the number of estimators from 100 to 150 and reducing the number of subsamples per bin to the number of samples in our train set, yielding a final result of 0.49. We then tested different combinations of parameters, observing that the changes, both increasing and decreasing, do **not impact** the model so decisively as to justify its choice.

Logistic Regression Finally, we used logistic regression, which in the context of this experiment with 50 classes, is the **worst**, along with SVC. Since this model has a very fast computation we were able to test a **wide** combination of hyperparameters, trying to find the best one. Using the most appropriate solvers for multi-class problems, we obtained very similar values between them with the exception of the solver 'newton-cg' which returned the **worst performance**, prompting us to overlook it. The best solver, 'lbfgs', achieves an accuracy of 0.33 with a penalty of type l2 and moderate regularization, with C = 1, chosen following some experiments that showed that performance **worsened** if regularization was increased ($C \leq 0.5$) while it remained mostly **stable** for milder levels of regularization ($C \geq 5$).

In order to draw conclusions from this classification experiment, we have grouped the results of the best classifiers in Table 4, for a more agile comparison, also plotting the **ROC curve** for the performance that the classifiers returned for the macro-genre 'techno' (in Figure 19). From these two representations of the results, we can see that, although the classification task was **complex**, since it had 50 different classes to target, the results we obtained are, in our opinion, **acceptable**, especially in the cases of classification with Random Forest and LightGBM; as far as the ROC curve is concerned, we can see that, among the different methods, the macro-genre is characterised by a good **discriminating capacity**, since we obtain values all greater than 0.88.

Classifier	Parameters	Accuracy	Macro avg	Weighted avg
Random Forest	estimator=Decision Tree, n_estimators=150, max_features=50	0.47	0.38	0.44
SVM	kernel='rbf', gamma=0.002, C=1	0.23	0.09	0.15
Neural Network	hl=3, n_neurons=(256,128,64), activation='tanh', kernel_regularizer=l2(0.0001), Dropout=0.3, input_dim=n_feature, output=n_classes	0.43	0.37	0.42
LightGBM	boosting_type='gbdt', n_estimators=150, subsample_for_bin=n_samples	0.49	0.43	0.48
Logistic Regression	C=1, penalty=l2, solver='lbfgs'	0.33	0.25	0.30

Table 4: Advanced Classification performance summary

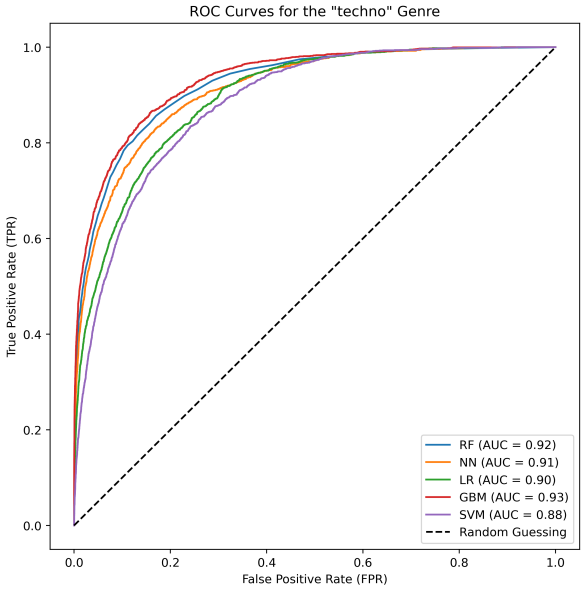


Figure 19: ROC curve for the 'techno' macro-genre

5.1.3 20 classes

As a last classification task, we tried to apply Neural Networks, LightGBM and Random Forest models on a **subset** of samples. We only considered instances of the song dataset that also had a **temporal representation**, so the genres to be targeted are only 20, as the time series only feature these. Then we tried to do some **feature engineering**: we extracted a total of 77 variables from each time series by calculating the following features from both the **total** time series and 10 **sliding windows** of length 128 (1/10 of the total length): '*max*', '*min*', '*mean*', '*variance*', '*std*', '*autocorrelation*' and '*kurtosis*'.

We tried to train these three models on some **combination** of 3 subsets of variables: the 15 **continuous** variables in Figure 17 (Cont), the **rest** of the columns of the

tabular dataset of songs (others) and the 77 features extracted from the **time series** (TS). This was done to get a measure of how much **each group** of variables **impacted** on the classification. The accuracy results can be observed in the following table (5).

Classifier	Cont	Cont + TS	Cont + others	All
Random Forest	0.67	0.63	0.69	0.66
Neural Network	0.63	0.61	0.60	0.64
LightGBM	0.69	0.70	0.72	0.72

Table 5: Variables combinations results

In general, one can immediately see from the table that the models perform very well despite the **dimensionality** of the datasets on which they are trained on. However, it stands out that only in the case of the LightGBM does the addition of columns (both TS and others) actually lead to a **very slight** improvement in performance. In the case of the neural network, adding only one set of variables to the continuous ones leads to a **lowering** of the accuracy, which however only returns to a higher value than the original when all the columns are considered. Finally, in the Random Forest, the model is only **negatively affected** by the columns derived from the TS, while it has a slight benefit from the others variable. We can say that, although all these models should in theory benefit from the addition of variables, this does **not always** occur and, from our analysis, it emerges that these features are **not significant enough** to improve performance.

5.2 Advanced regression

In this section we dealt with nonlinear regression. On this occasion we decided to use a model among ensembles (AdaBoost) and a Gradient Boosting (LightGBM), using in both models '*danceability*' as the **dependent** variable. Both regressors were trained on **multiple combinations** of independent variables to test how many and which variables could best explain the target.

AdaBoost Among the various ensemble methods, we chose this one since, after a few attempts, we could get a better performance. The first model we present is the one with 3 regressors, which were chosen from the **most important** variables, already visualized previously in Figure 17. Among them, those that best explain Y are *'energy'*, *'valence'* and *'loudness'*, and they are also the ones that at the **semantic level**, in our opinion, can best explain the dependent variable. Using these three independent variables, the model achieves a **very low** R^2 value of 0.106. Next, we tried to **enrich** the model by adding one continuous variable at a time as long as the gain of R^2 justified it. This led us to consider a total of 12 variables: '*duration_sec*', '*popularity*', '*energy*', '*loudness*', '*speechiness*', '*acousticness*', '*instrumentalness*', '*liveness*', '*valence*', '*tempo*', '*fade_out_%*' and '*n_beats*'. With the addition of these variables, the **regressors** were able to explain 38% of the total **variability** in the model.

LightGBM We therefore tried to use a more **complex** model such as LightGBM, as it uses **deeper** Gradient Boosting Decision Trees than stumps, because we expected that, all the conditions unchanged, it would be able to explain '*danceability*' better. Indeed, even using **only three** variables, the regressors achieve an R^2 of 0.33, slightly lower than the best result obtained by AdaBoost. When, on the other hand, the independent variables become 12, they demonstrate an **excellent ability** to capture the target's variability, returning an R^2 of 0.65

All the above results are shown in Table 6 below.

Regression Model	**N. of regressors**	R^2
AdaBoost	3	0.106
AdaBoost	12	0.381
LightGBM	3	0.336
LightGBM	12	0.655

Table 6: Advanced Regression's performance summary

5.3 DL-based Models for TS Classification

In addition to the classification section of the time series, we also opted for a Deep Learning classification. Our aim is to see how the time series behave and react to the application of certain classification techniques used in Section 5.1. For each classifier used, we initially performed a **PAA approximation** of the time series: for CNN, we reduced the length size to 1/10 of the original size (from 1280 to 128), while for mini-ROCKET we wanted to keep the same application structure as the one used in Section 3.1.3, so we reduced the size of the time series to half of the original size (from 1280 to 640).

Convolutional Neural Network As a first classifier, we opted for a Convolutional Neural Network. After trying out various CNN structures, we managed to create a model that produced results we found satisfactory. The model in fact returns an accuracy of 0.37. With regard to the parameters and structure of the model, it has 3 layers with respectively 16, 32 and 64 filters and with values of 8, 5, 3 for the kernel size. After each layer, a Batch Normalization layer was applied and then 'relu' was assigned to the activation function. Finally, we set the Dropout to 0.2. However, **before** approximating the time series, we applied several CNN structures to the time series of **original length**. Unfortunately, The performance in this case is **not satisfactory**, as the results are **poorer** than when using the approximate time series.

Bagging with Mini Rocket In this case we used a combination of a Bagging approach with the Mini Rocket Classifier, so we thought it appropriate to create this interesting combination to observe how the time series might react. We therefore opted for the **same model** used in Section 3.1.3 about the Time Series Classification, which has a 3000 kernels. As for Bagging, after trying various numbers of estimators, the best performance we obtained sees the number of estimators set to 100. This model returns

an accuracy of 0.48.

Referring to Table 1, which shows all the methods we used for the Time Series Classification, it is clear that the mini-ROCKET classifier with 3000 kernels achieved the **highest performance** with an accuracy of 0.43. This demonstrates the effectiveness of the mini-ROCKET method in capturing and exploiting time series features for classification. Bagging classifiers with Rocket and CNN showed satisfactory performance, with accuracies of 0.48 and 0.37, respectively. In particular, we are particularly satisfied with the Bagging + mini-ROCKET classifier, as it was able to exceed the accuracy of the mini-ROCKET taken individually. As for CNN, on the other hand, although its results are quite satisfactory and good (it returns better performance than KNN and Shapelets-based models), it fails to outperform mini-ROCKET + Bagging and single mini-ROCKET.

5.4 Explainable AI

In this section, we will try to explore the **explainability** of some of the models used in the analysis so far. We will explore two different approaches to performing this task, one **local**, using LIME, which focuses on providing explanations of how a specific prediction is performed; and another **global**, using TREPAN, which attempts to faithfully reproduce the entire model with the intention of replicating its performance and providing an easily interpretable representation.

5.4.1 TREPAN

Here, we sought to explain the behavior of the Neural Network used to solve the **binary** imbalanced classification task presented in Section 5.1.1, namely classifying the genre 'techno' vs all others.

Recall that the NN result is an f1-score of 0.45. We constructed an **interpretable** Decision Tree, thus with a maximum depth of 4 splits, looking for the best parameters of 'criterion', 'min_sample_split' and 'min_samples_leaf', which after some analysis should correspond to: entropy, 13 and 9, respectively. The Decision Tree in question can be observed in Figure 20.

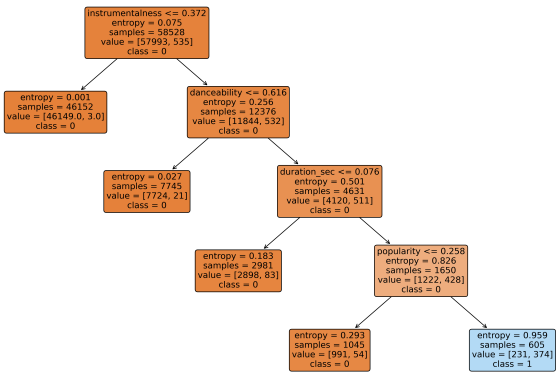
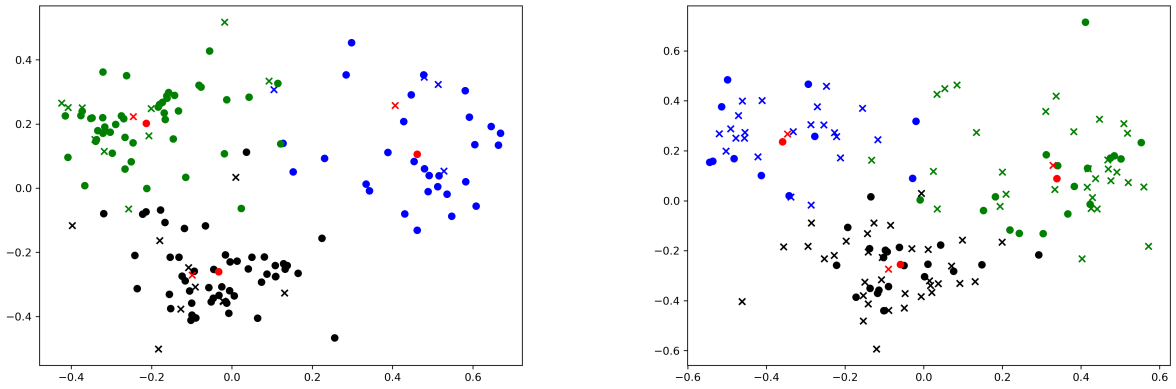


Figure 20: Neural Network representation by TREPAN

Our opinion is that the model is quite **faithful** to the original NN, returning an f1-score on the positive class test samples of 0.38 and a fidelity of **99.3%**, demonstrating that the model created by TREPAN manages to replicate the logical connections operating within the original model **almost perfectly**, even though the performance slightly drops (may be symptom of **overfitting**).

5.4.2 LIME

As done in Section 5.1.3, we used the 10000 songs from the dataframe obtained from the union with the time series, considering only the **continuous** variables of the tabular dataset. Of the various models tested in this section, we chose to explain the **RandomForest**. For each test instance we calculated, using **LIME**, the **contribution** of each feature to the final prediction of the model, **normalising** the results using the MinMax Scaler. We then focused our analysis on the **two** genres that returned the two extreme results of the classification: the best, 'honky-tonk' (f1 = 0.84), and the worst, 'goth' (f1 = 0.34); using then the feature contributions only of the test instances **classified** as one of the two genres. Next, we clustered these two subsets into **3 clusters** each using KMeans, plotted them reducing their size to 2 with **IsoMap**. The two scatterplots for the two genres can be seen in Figure 21.



(a) 'honky-tonk' (b) 'goth'

Figure 21: LIME clusters scatterplots

In the graphs, the three clusters are recognisable by their different **colouring**; in addition, each cluster has been further subdivided into **misclassified** samples (crosses) and **correctly** classified samples (dots), with the respective **centroids** highlighted in red.

We can see that in the 21a plot, the dots are **denser** than in the 21b plot, which in our opinion means that, in the explained model, the characteristic features of the 'honky-tonk' samples are more **discriminating** than those of the 'goth' samples, so the scatterplot confirms the fact that the model is very **robust** in the estimation of 'honky-tonk' and less so for 'goth'. Unfortunately, there is no clear **distinction** in the features that contribute to one or the other classification, and it can be seen that correctly predicted samples and negatively predicted samples are highly **overlapped**.