UNIVERSITÀ DI PISA

Department of Computer Science
Master program in Data Science and Business Informatics

Data Mining: Advanced Topics and Applications

Project Report

Minh Duc Pham
Thomas Gonzo
Giuliano Gelsomino

Academic Year 2023-2034

Contents

1 Data Understanding

1.1 Tracks file

The Spotify Tracks Dataset used in this study represents the information of available audio tracks on the Spotify catalog. Each track is described by essential details *id, name, artist, album* and several audio-derived features representing various aspects such as *energy*, *tempo*, *loudness*, etc. The dataset contains 109,547 records and 34 attributes. There are 22 numeric attributes, 11 categorical attributes, and a date time column.

We implemented checks to find null values, and the data has no missing values. However, there were 398 records duplicated, so we removed them. Furthermore, we detected duplicated values based on the *id* column, where some entries had different *genre* values despite having the same values for other attributes. We decided to remove these duplicates as well. Outlier detection was skipped at this stage and will be conducted again in a later phase with more advanced techniques. Finally, the dataset was reduced to 89,560 records. Some descriptive statistics were subsequently conducted, as shown below.

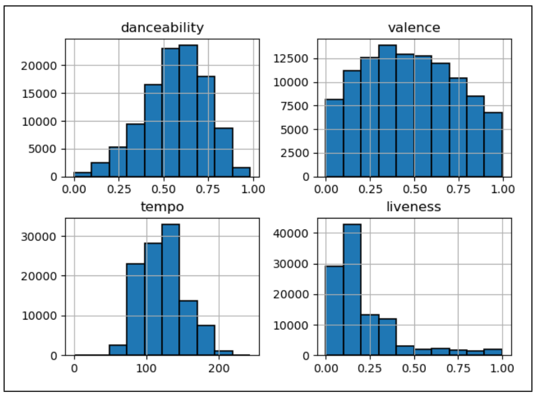


Figure 1: The distribution of the continous variable

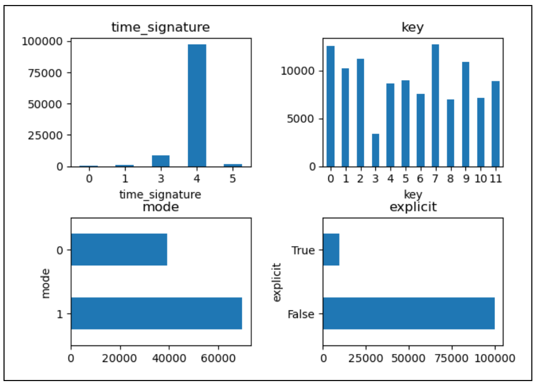


Figure 2: The distribution of the discrete and binary variables

This section explores the distribution of selected numerical attributes through histograms and count plots, along with the corresponding statistics. Figure 1 illustrates the visualization of randomly chosen attributes having continuous variables, including *tempo*, which shows a distribution resembling a standard normal distribution. However, the *valence* and *liveness* data display a leftward skew, while *danceability* has a rightward skew.
In the figure 2, there is the distribution of the discrete and binary variables. It is easy to see unbalanced data at *mode* and *explicit* side. In *time_ signature*, most of the tracks are in 44. The *key* attribute also exhibits imbalance, with the most frequent values being 7, 0, and 9, while 3 is the least frequent. Therefore, those problems would be solved in the outlier detection and imbalanced learning phase.

1.2 Artists file

The Spotify Artist shows the information of each specific artist, including their *id*, *name*, *popularity*, *followers* and *genres*. The *id*, *name*, and *genre* attributes are categorical types, the remains are numerical. We checked the duplicate data at the *id* and there are no duplicate. The dataset has also no missing value. We implemented z-score technique at numerical attributes to find the outliers having more than 3 scores, and no outlier is detected. Finally, we have 30,139 records in the dataset.

1.3 Merging two datasets

In this step, we merged both *tracks_file* and *artists_file*. Artist names in the tracks data were matched to corresponding entries in the artists data. This process yielded two new features in the tracks data: *artist_popularity* and *artist_followers*. For a track having a single artist, the *artist_popularity* and *artist_followers* values directly correspond to the respective values in the artists data. However, in tracks with multiple artists, these features represent the average popularity and follower count across all contributing artists.
Before implementing the next step, we removed some unnecessary attributes, such as *id*, *name*, *disc_number*, *duration_ms*, *track_number*, *artists*, *album_release_date*, *album _name*, , *album_release_date_precision*, *album_total_tracks* , to avoid the wrong results. We encoded the categorical data having small unique value like *album_type*, *genre*, *explicit*. In the end of this step, we have totally 89,560 records and 35 attributes.

1.4 Initial classification

Prior to addressing outliers and data imbalance, we conducted two classification techniques: Decision Tree and KNN. The results obtained from these initial models would be used to make the comparison with subsequent models trained after outlier removal and data balancing are implemented. In this case, we used the *genre* column like a target attribute. Before deployment, we split the data into 0.75-0.25 train-test partition and then used StandardScaler to implement the normalization. At Decision Tree, we applied the GridSearch technique to identify the best value, and we apply it with '*min_samples_split*': 14, '*min_samples_leaf*': 5, '*max*': 29 and '*criterion*': '*entropy*'. At KNN technique, we identified the best parameters at '*metric*': '*manhattan*', '*n_neighbors*': 24. The results of both are quite low around 35%. In particular, the performance is showed as below.

	Precision	**Recall**	**F1 Score**	**Accuracy**
Decision Tree	0.34	0.33	0.33	0.35
KNN	0.35	0.35	0.32	0.35

Table 1: Performance metrics for Decision Tree and KNN models

2 Advanced Data-Preprocessing

2.1 Anomaly Detection

This section discusses the detection and handling of outliers. We employed 9 different outlier detection techniques. Since we didn't want to rely on just one method, we constructed an ensemble which provided us a more accurate outlier labelling (using majority voting). Each algorithm provides an outlier score for each point in the dataset. Accordingly, the 1% of outliers with the highest score was extracted for each algorithm used.
Firstly, we did the various visualizations by analyzing each feature individually using boxplots. The numerical variables are plotted in charts considering IQR with thresholds of $Q1 - 1.5 \times IQR$ and $Q3 + 1.5 \times IQR$. We detected a total of 7777 outliers. However, this approach did not return to the top 1% threshold. Therefore, we ignored this result and implemented further analysis then.

The algorithms adopted for detecting the outliers that we applied in this study are: IsolationForest, LOF, DBSCAN, CBLOF, KNN, HBOS, Elliptic Envelope, LODA. Anomaly detections were only applied on training dataset to avoid the data leakage. The results of implementation are described as the table 2 with the number of outliers. Figure 3 show the scatter plots of various outliers detection methods. In yellow are represented inliers, whereas in red are pointed the outliers. The PCA representation is used for plotting the data in 3 dimensions.

	IF	**LOF**	**DBSCAN**	**CBLOF**	**KNN**	**HBOS**	**EE**	**LODA**
Outliers	896	692	612	896	851	896	896	896

Table 2: Number of outliers detected by each method.

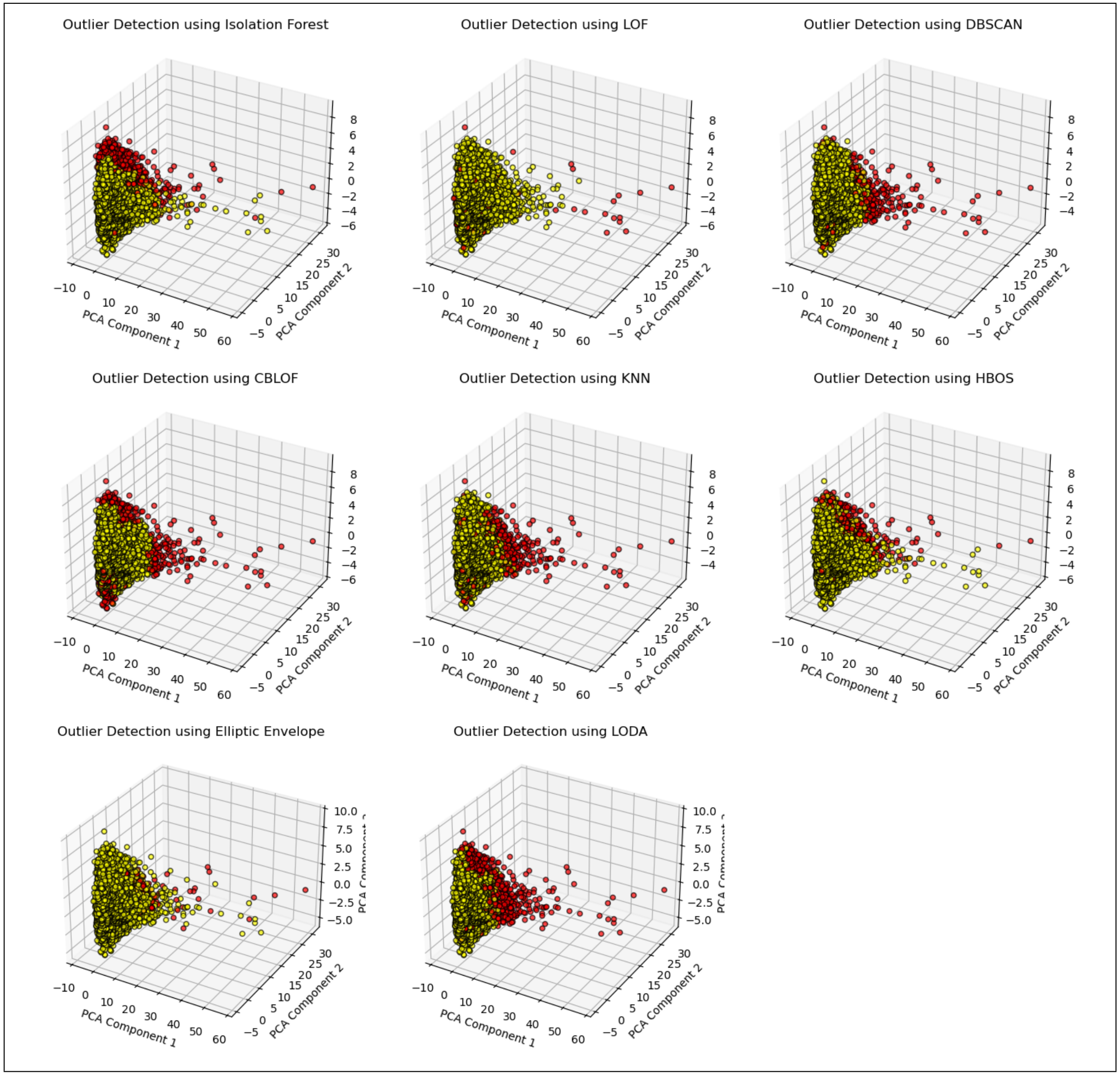


Figure 3: Outliers detection using various algorithms.

The outcome of the anomaly detection was obtained through a majority voting schema. After evaluating the proposed methods, we assigned a label to each row of the dataset, indicating if the track was an outlier or not. In 8 methods, if each records containing

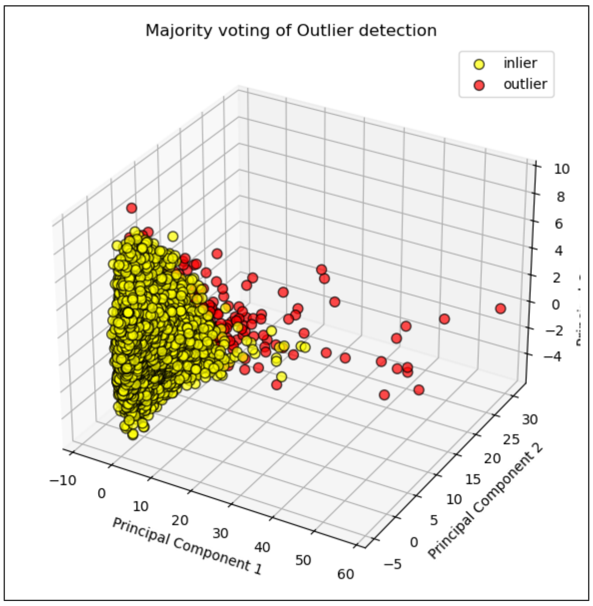


Figure 4: Outliers after Majority Voting

“*outlier*” from more than 4 methods will be considered as outlier. Consequence, we have totally 215 records are outliers. Figure 4 shows the PCA presentation in 3D, the red points (outliers) reside mostly in a sparse region compared to the denser area (inliers). To assess the impact of outliers on classification performance, we removed these identified outliers and compared the classification results before and after outlier removal. The *genre* column was chosen as the target variable for classification. Two classification techniques, Decision Tree and KNN, were used for this comparison. After testing, we have the results between before and after outlier removal as table 3. In particular, we identified the accuracy of Decision Tree was same but the result of KNN after outlier removal was improved. Based on this observation, we opted to remove all outliers identified by the majority voting schema for subsequent analyses.

	Before Outlier Removal				**After Outlier Removal**			
	Precision	Recall	F1 Score	Accuracy	Precision	Recall	F1 Score	Accuracy
Decision Tree	0.34	0.33	0.33	0.35	0.34	0.34	0.34	0.35
KNN	0.35	0.35	0.32	0.35	0.37	0.36	0.35	0.38

Table 3: Comparison of classifiers before and after outlier removal

2.2 Imbalanced Learning

In this section, another classification task was performed. The `explicit` column contains two classes: "TRUE" (explicit lyrics) and "FALSE" (non-explicit lyrics). The target values were encoded by class “TRUE” = class 1 and class “FALSE” = class 0. The data shows a significant class imbalance, with "FALSE" constituting 90.5% of the data and "FALSE" constituting only 9.5%. The dataset was split into ratio at 75% - 25%. We applied the classification with unbalanced dataset initially. Then we implemented both two data balancing techniques – oversampling and undersampling – to address the class imbalance and assess its potential influence on the model’s performance.

First of all, after parameter tuning, K-NN (*'metric'*: *'manhattan'*, *'n_neighbors'*: 20, *'weights'*: *'distance'*) and Decision Tree (*criterion* = *'entropy'*, *max_depth*=8, *min_samples_leaf*=17, *min_samples_split*=15) classifiers are deployed. The results of classification are shown in table 4.

	Decision Tree		**KNN**	
Class	**TRUE**	**FALSE**	**TRUE**	**FALSE**
Precision	0.60	0.93	0.71	0.93
Recall	0.18	0.99	0.24	0.99
F-1 score	0.27	0.96	0.36	0.96
Accuracy	0.92		0.93	

Table 4: Performance Metrics of Decision Tree and KNN Classifiers

Several imbalance techniques were applied in this section. In order to undersample the majority class, we applied Random Under Sampler, Tomek's Link, Edited Nearest Neighbors, Cluster Centroids. Besides that, we also oversample the minority class, with some techniques like Random Over Sampler, SMOTE and ADASYN.

2.2.1 Undersampling

The first presented technique will be Undersampling, the methods that we used including Random Under Sampler, Tomek's Link, Edited Nearest Neighbors, Cluster Centroids. Plotting the first and second eigenvectors of the PCA can help appreciate how different methods affected the distribution of the dataset's principal component; such plots are shown in Figure 5.

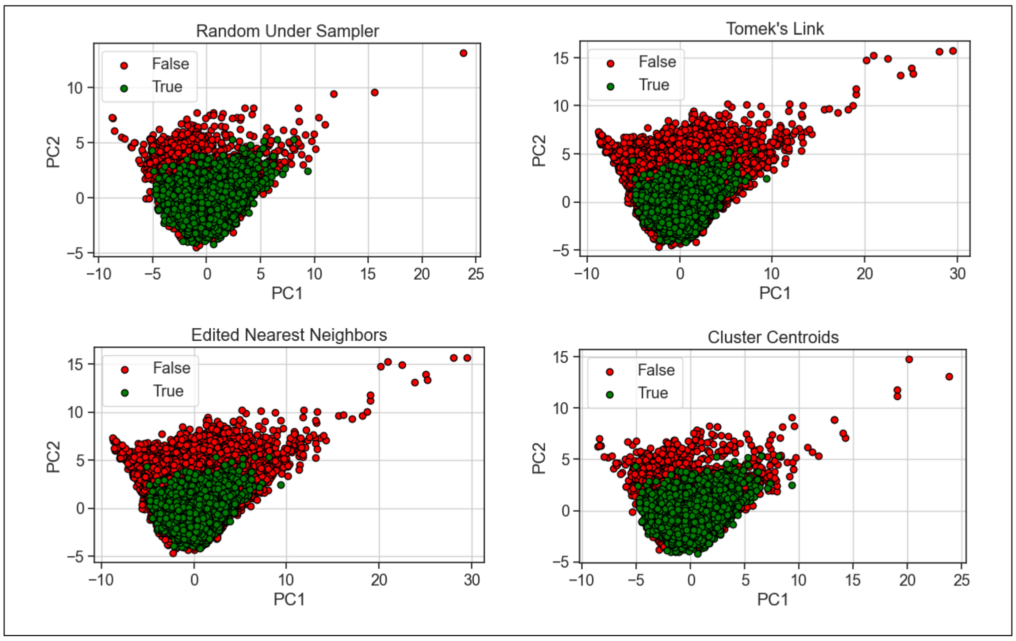


Figure 5: 2D PCA of Undersampling techniques

Ater applying the undersampling methods, we made the classification by Decision Tree and compared them to the results at imbalanced dataset. For each balancing method, we did the GridSearch techniques separately to find the optimal parameters of both KNN and Decision Tree model, then we conducted the classification to obtain the performance. The comparison is shown in table 4. For undersampling accuracy, we see there are not improved the results significantly between before and after applying balancing techniques.

Classification techniques	**Indicator**	**Random Under Sampler**	**Tomek's Link**	**Edited Nearest Neighbors**	**Cluster Centroids**
Decision Tree	F1-Score of class 0	0.84	0.96	0.95	0.82
	F1-Score of class 1	0.32	0.36	0.38	0.32
	Accuracy	0.84	0.92	0.95	0.82
KNN	F1-Score of class 0	0.84	0.96	0.96	0.83
	F1-Score of class 1	0.36	0.39	0.44	0.35
	Accuracy	0.75	0.93	0.92	0.74

Table 5: Performance metrics for Decision Tree and KNN classifiers using different Undersampling techniques.

The best performance is Tomek's Link, it increases the indicator of F1-Score for both classes, but the accuracy was not changed. The remains have lower values than the ordinal dataset. In conclusion, no undersampling methods gave the better performance than the original result.

2.2.2 Oversampling

The second presented technique will be oversampling, which could be preferred over undersampling since the training set was already small and so it would be better instead of shrinking it, to avoid underfitting. We applied Random Over Sampler, SMOTE and ADASYN to oversample the minority class. Figure 6 describes the distribution of "explicit" variables by PCA 2D plotting.

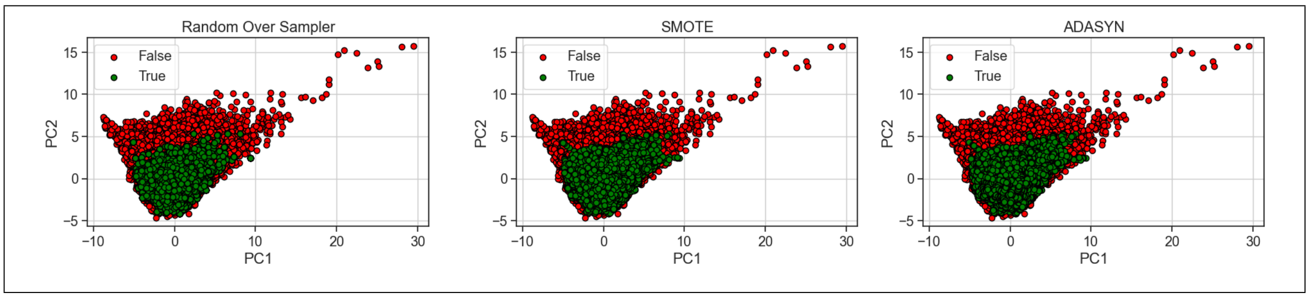


Figure 6: 2D PCA of Oversampling techniques

Following the application of those techniques, we performed classification using a Decision Tree model. The results of this classification task are presented in Table 6. We realized the Random Over Sample had the best results but less than the performance of imbalanced dataset. Figure 6 shows the ROC of the Random Over Sample
In conclusion, all classification performance metrics achieved with the balanced datasets were lower compared to those obtained using the imbalanced dataset. Because both techniques in undersampling and oversampling gave the low performance, so we decided not to apply them. The dataset would keep after this step.

Classification	Indicator	Random Over Sampling	SMOTE	ADASYN
Decision Tree	F1-Score of class 0	0.92	0.88	0.88
	F1-Score of class 1	0.40	0.35	0.35
	Accuracy	0.85	0.80	0.95
KNN	F1-Score of class 0	0.88	0.87	0.86
	F1-Score of class 1	0.37	0.38	0.36
	Accuracy	0.80	0.79	0.77

Table 6: Performance metrics for Decision Tree and KNN classifiers using different Oversampling techniques.

3 Advanced Classification

In this section, we did some advanced techniques for classification such as: Linear Regression, Support Vector Machine, Ensembled Method, Gradient Boosting. We applied these techniques to a dataset that was carefully cleaned and preprocessed in the previous module. We selected only useful numerical attributes and scaled them through the StandardScaler. The split algorithm was used to divide the dataset into training set (70%) and testing set (30%). We specifically targeted an attribute for prediction: "*popularity*". In "*popularity*", we encoded them as table 7 below:

Range of Values	New Label	Number of Variables
<40	Low	57182
40 - 70	Medium	29159
>70	High	3000

Table 7: Number of variables in popularity class

3.1 Logistic Regression

In "*popularity*" class, we did the GridSearch we found the best set of hyper parameters among: classweight: none, balanced; C: 0.01, 0.01, 0.1, 1, 10 ; penalty: "*l1*", "*l2*", "none"; solver: "*lbfgs*", "*liblinear*", "*saga*". The best set of parameters was: "*balanced*", 0.001, "*l2*", "*liblinear*". The algorithm gave the good ROC_AUC score at 0.79 but the accuracy score is only 0.64. This indicates that the model is good at distinguishing between classes, but the class imbalance has a negative impact on accuracy and f1-score. The figure shown the ROC curves and AUC scores at each class, class of high performed better than others. It indicates a good performance in distinguishing between positive and negative classes

Class	Precision	Recall	F1-score
High	0.19	0.50	0.28
Medium	0.50	0.43	0.46
Low	0.76	0.75	0.76

Table 8: Performances of the Logistic Regression model, Accuracy = 0.64

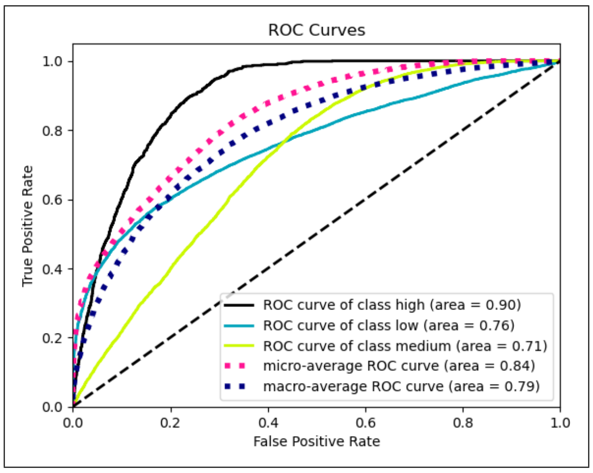


Figure 7: ROC curves and AUC scores for each class after performing Logistic Regression model

3.2 Support Vector Machine

In this section, we did both Linear and Non-Linear SVM for the task of classifying Popularity class of the tracks. The models were implemented with Scikit-learn and the GridSearch technique was applied to find the best parameters for the dataset.

3.2.1 Linear SVM

We used the LinearSVC to implement. The input parameters for the GridSearch were: C: 0.001, 0.01, 0.1, 1; loss: "*hinge*", "*squared_hinge*"; tol: 0.01, 0.1, 1; class_weight: "*None*", "*balanced*". After searching, the optimal setting for the C parameter is 1, class_weight: "*balanced*"; loss = "*squared_hinge*" and tol is 1. In popularity class, we obtained the results with ROC_AUC score at 0.79 and accuracy score at 0.64 . We presented in table. For the results obtained were similar to those achieved with logistic regression.

Class	**Precision**	**Recall**	**F1-score**
High	0.19	0.57	0.28
Medium	0.77	0.74	0.76
Low	0.51	0.43	0.47

Table 9: Performances of the Linear SVM model, Accuracy = 0.64

3.2.2 Non-linear SVM

We found the best parameters for Non-linear SVM with C: 0.01, 0.1, 1; tol: 0.01, 0.1, 1; class_weight: "*none*", "*balanced*"; kernel: "*poly", "rbf", "sigmoid*"; gamma: "*scale*", "*auto*". For each kernel, we found the optimal set of parameters as C = 1, Class_weight = "*balanced*", gamma= "*scale*", tol = 1. Table presented the specific indicator with each kernel for both 2 target variables. The best SVC model is in kernel ="*rbf*". The table 10 shows the performance of each model of Kernel.

Model	Accuracy	Precision	Recall	F1-score	ROC
kernel = "sigmoid"	0.44	0.41	0.54	0.37	0.72
kernel = "poly"	0.66	0.53	0.66	0.54	0.82
kernel = "rbf"	0.69	0.56	0.70	0.58	0.85

Table 10: Performances of the Non-linear SVM model

From the table 10, the kernel "*sigmoid*" gave the worst performance with low accuracy (0.44) and ROC score (0.72). The "*poly*" kernel had the better results than. The best performance was "*rbf*", it achieved accuracy up to 0.69 and the high ROC score at 0.85. Besides that, the other results of "*rbf*" were also higher than "*sigmoid*" and "*poly*" kernel.

3.2.3 Final Evaluation

We compared two models a Linear SVM and a Non-Linear SVM. The performance of Non-Linear SVM gave the better result for ROC_AUC and accuracy score. It increased the accuracy from 0.64 (Linear SVM) to 0.60 (Non-Linear SVM), the ROC_AUC score was improved up to 0.85 at Non-Linear model.
Figure 8 shows the confusion matrices for both models. The Non-Linear SVM with the RBF kernel significantly increased the number of correctly predicted values in the "high" class, from 22 to 517, and in the "medium" class, from 3168 to 4287. While the number of correct predictions for the "low" class decreased a little bit, however this decrease was not significant.

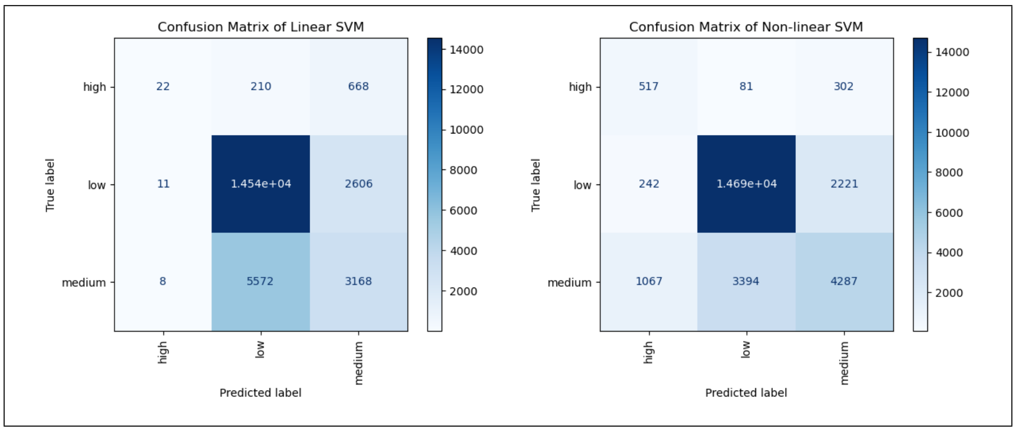


Figure 8: Confusion Matrix of Linear SVM (left) and Non-Linear SVM (right)

3.3 Ensemble Method

Ensemble learning improves the accuracy by aggregating the predictions of multiple classifiers in order to predict class labels of test records by combining the predictions made by multiple weak classifiers. In this section, we deployed the flowing algorithm in ensembled method such as: Random Forest, Bagging and Adaboosting. All models were implemented by Scikit-learn and the GridSearch technique was used to find the best parameters for the dataset

3.3.1 Random Forest

For the RandomForest classifer, we used the Gridsearch to find the best parameters with n_estimators: 10, 100, 200, criterion: "*gini*", "*entropy*", "*log_loss*", max_depth in range (5, 35), min_samples_split in range (1,20), min_samples_leaf in range (1,10). After searching we have the optimal parameters with n_estimator = 200, criterion = "*gini*", max_depth = 30, min_samples_split = 2, min_samples_split = 2. The result is in table with the accuracy score was up to 0.79 and ROC_AUC score was 0.9 According to the table '11, the model performed better for classes with higher recall values (Medium and Low). However, the results of "high" class were lower, it indicated the model might be missing some relevant instances. The result of confusion matrix and ROC curve of this model are shown in figure 9

Class	Precision	Recall	F1-score
High	0.66	0.13	0.23
Medium	0.68	0.68	0.68
Low	0.84	0.88	0.86

Table 11: Performances of the Random Forest model, Accuracy = 0.79

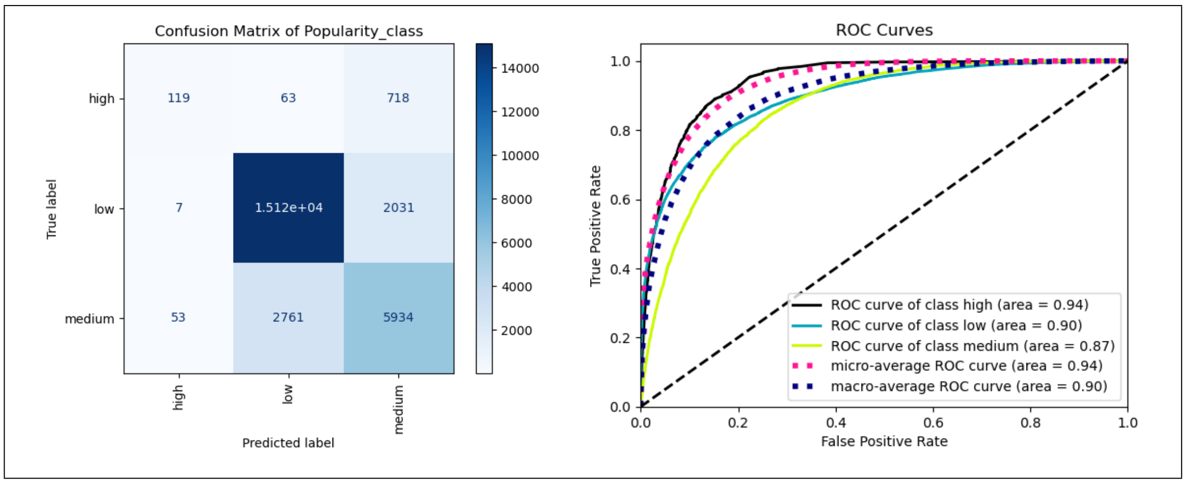


Figure 9: Confusion Matrix (left) and ROC curve (right) of Random Forest model

3.3.2 Bagging

For the Bagging approach, we selected the estimators of None, SVC, RandomForest, Decision Tree, Logistic Regression. The GridSearchCV was used to tune the hyper parameters of the corresponding model. The table 12 give the results after deploying.

In popularity_class, "None" estimator gave the best accuracy (ROC_AUC, Precision). This suggests the data might be well-clustered or the metrics aren't ideal. Bagging improved SVC performance, but Random Forest and Logistic Regression showed minimal change.

Estimator	Accuracy	Precision	Recall	F1-score	ROC
None	0.80	0.74	0.59	0.62	0.90
SVC	0.76	0.70	0.53	0.55	0.74
Random Forest	0.79	0.74	0.55	0.57	0.90
Decision Tree	0.78	0.74	0.55	0.57	0.90
Logistic Regression	0.64	0.49	0.57	0.50	0.79

Table 12: Performances of the Bagging model with several estimators

3.3.3 Adaboost

The Adaboost algorithm is known for providing improvement in the performance of classiers through combining multiple weak learner into the stronger learner. For this approach, we chose to use as estimators None, RandomForest, Decision Tree and Logistic Regression. The GridSearchCV was deployed to find the optimal parameters of the corresponding model

Estimator	Accuracy	Precision	Recall	F1-score	ROC
None	0.74	0.65	0.54	0.56	0.70
Random Forest	0.78	0.67	0.53	0.55	0.89
Decision Tree	0.76	0.67	0.51	0.52	0.87
Logistic Regression	0.59	0.47	0.59	0.44	0.76

Table 13: Performances of the Adaboost model with several estimators

In the table 13, the best performance is Random Forest, with the highest score of all indicators. However, it's important to note that these scores were lower than the performance observed before deploying the boosting technique (applicable to both Random Forest and Logistic Regression).

3.4 Neural Networks

In this section, we implemented the Neural Networks Classification.The classification was performed by training the multilayer perceptron classifier from the Scikit learn library. After testing numerous network configurations, parameter tuning was conducted using RandomizedSearchCV among the following:

- Solver: '*sgd*', '*adam*'
- Hidden_layer_sizes: (20,), (128, 64, 32), (64, 128, 32), (64, 32,), (32, 16)
- Alpha: 0.01, 0.01, 0.001
- Activation: '*tanh*', '*relu*', '*logistic*'
- Validation_fraction: 0.0, 0.1, 0.2

The architecture and parameters that performed most satisfactorily are: 'sgd', (128, 64, 32), True, 0.01, 'tanh', '0.2'. The performance obtained is 76% accuracy. The results of model are shown in table 14.

It is noteworthy that the most performance configuration has a pyramidal structure, characterized by 3 layers of neurons arranged in decreasing order based on the number of

Model	Precision	Precision	Recall	F1-score	ROC
MLPClassifier	0.76	0.71	0.57	0.59	0.86

Table 14: Performances of the Multilayer Perceptron model, Accuracy = 0.79

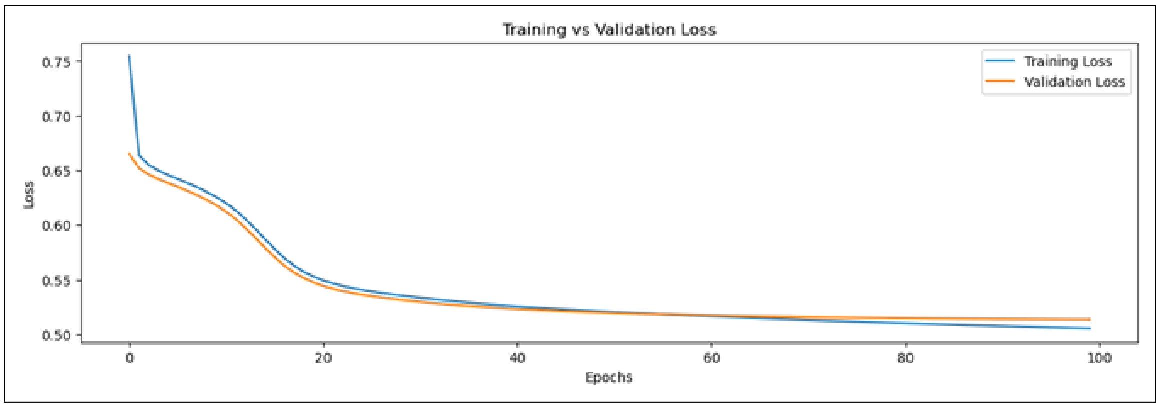


Figure 10: Training and validation loss per epoch

nodes. Interestingly, the most suitable learning rate is neither the highest nor the lowest; probably due to the structure, a moderate gradient descent allows reaching the minimum efficiently while limiting the risk of escaping the minimum by performing steps that are not too large. The decisive factor for the model's performance is the combination of early stopping and validation fraction. Thanks to these two parameters, it was possible to stop the model's training when the training error curve started to decrease and the validation error started to increase, set at a size of 0.2 on the training set. The model performs this operation automatically; however, to understand the relationship between overfitting and the number of epochs, the model's functioning was manually replicated on a validation dataset with a size of 20% of the training set.

The figure 10 shows that after around 60 epochs, the training error keeps decreasing, whereas the validation error starts increasing after an initial decrease. This indicates the onset of overfitting.

The ROC curve was illustrated at the figure 11 shows the performance of the classifier

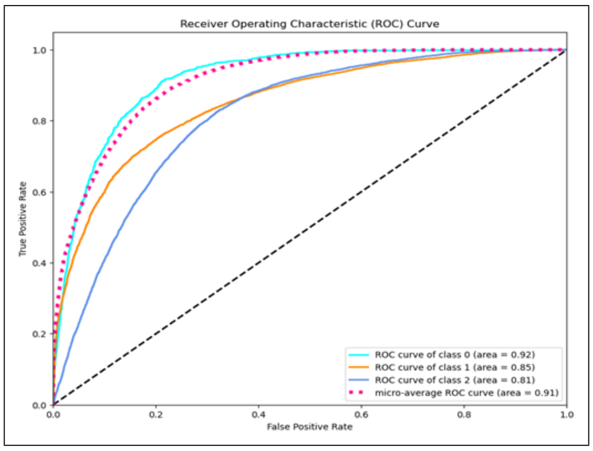


Figure 11: ROC curves and AUC scores for each class after performing Multilayer Perceptron model

across three classes, with the area under the curve (AUC) being 0.92, 0.85, and 0.81 for class 0 (high popularity), 1 (medium popularity), and 2 (low popularity), respectively. The micro-average ROC curve has an AUC of 0.91, indicating overall good classification performance, particularly for class 0.

3.5 Gradient Boosting

This paragraph aims to compare the classification techniques related to gradient boosting. The classification problem is defined as predicting the target variable 'popularity' using the models Gradient Boosting, XGBoost, and LightGBM. The dataset was initially divided into a training set (70%) and a test set (30%). The selection of optimal parameters in terms of performance on the test set was done using the RandomizedSearchCV operator. Given the large number of instances, the models were trained on a sample equal to 50% of the dataset and then applied to the test set. The table 15 compares the performance of the various models used.

Classifier	**Accuracy**	**Precision**	**Recall**	**F1-Score**
Gradient Boost	0.83	0.82	0.82	0.82
XGBoost	0.86	0.86	0.86	0.86
LightGBM	0.82	0.80	0.81	0.81

Table 15: Performances of different estimators for Boosting models

Training was also performed on the complete training dataset, but the best performance was obtained with the sampled dataset. Perhaps the smaller number of training instances proved to be a better technique for reducing complexity and thus preventing overfitting.

4 Advanced Regression

In this section, two advanced regression models were analyzed: Multilayer Perceptron Regressor and Random Forest Regressor. From the result of advanced data preparation, we selected the numerical attributes to implement the Spearman correlation. After sorting the feature pairs based on the absolute value of their Spearman correlation coefficients, we filtered the pairs to only include those with a correlation coefficient greater than 0.4 or less than -0.4. After analyzing, we decided to select the "*acoustics*" as a target for prediction, the other columns were chosen to as variables for computation such as "*n_ beats*", "*n_ bars*", "*energy*", "*loudness*", "*start_ of_fade_ out*", "*tempo*'", "*tempo_ confidence*", "*speechiness*". Next, we splitted the dataset into training set (70%) and testing test (30%). All the data were standardized by StandardScaler before doing the RandomSearchCV.
For both models, a tuning phase of the hyperparameters was performed, with the same methods used for the classification. For the Multilayer Perceptron Regressor, the following parameters were tested:

- Hidden layer size: with a single hidden layer with number of neurons of 32, 64, 128, 256.
- Activation: "*logistic*", "*tanh*", "*relu*"
- Learning rate: "*constant*" and "*adaptive*"

- Alpha parameters: 0.0001, 0.001, 0.01, 0.1

After searching, we identified the best parameters: hidden layer size 256, activation "*relu*", learning rate "*constant*", and alpha 0.001. The results of this model are shown in table 16

Model	R^2	MAE	MSE
Multilayer Perceptron Regression	0.614	0.165	0.04
Random Forest Regression	0.646	0.146	0.004

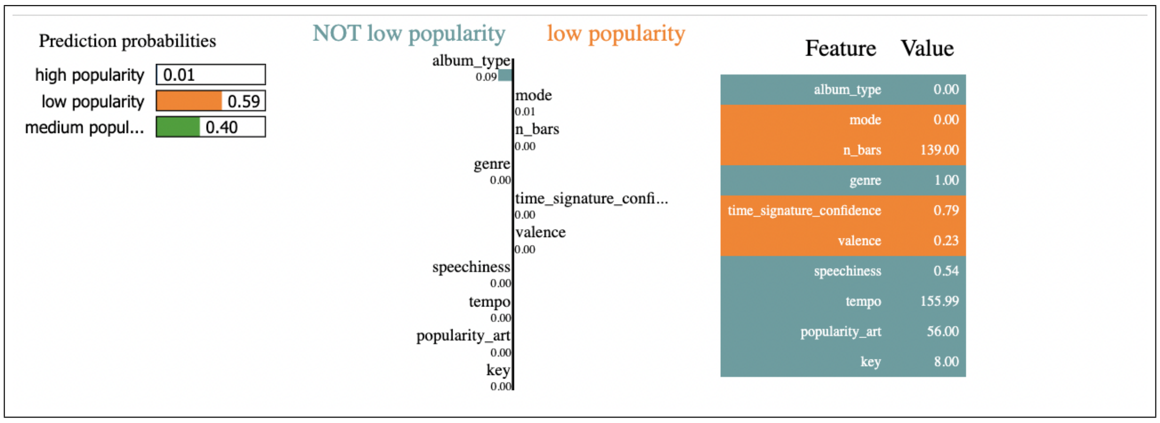
Table 16: Performances of the Advanced Regression models

For Random Forest Regressor, we set the input for parameters as n_estimator with values of 10, 100, 200. After implementing the RandomizedSearchCV, we defined the best optimal n_estinator of 200.
As shown in the table 16, both Multilayer Perceptron Regression and Random Forest Regression achieved good results. Their low Mean Absolute Error (MAE) and Mean Squared Error (MSE) indicate a good ability to capture the underlying pattern of the target variable. In conclusion, the Random Forest Regression provided slightly better results than the Multilayer Perceptron Regression.

5 Explainable AI

5.0.1 LIME

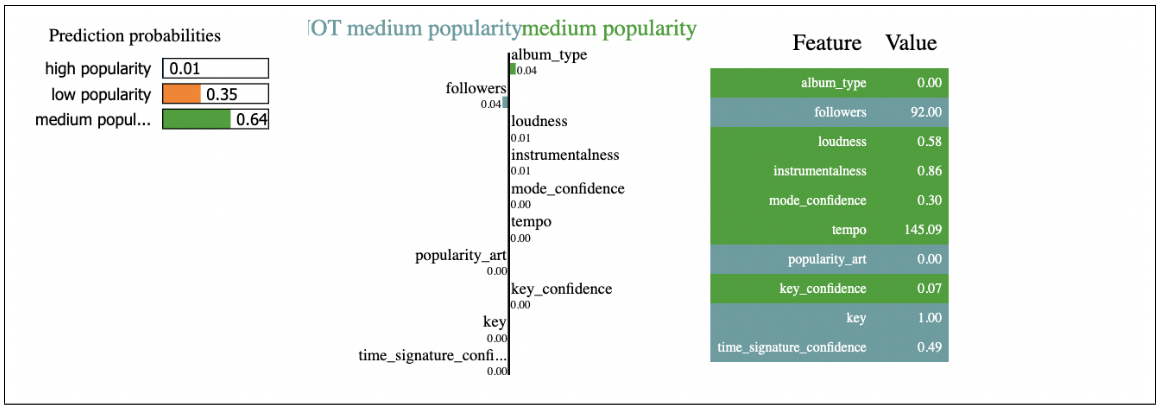
In the following paragraph, we presented the classification performed in module 3 using LIME. We decided to explain the classification results obtained with bagging to understand which features the model used to classify various instances. We did the explanation for each of the three classes so that it would be easier to understand the module.



Feature	Value
album_type	0.00
mode	0.00
n_bars	139.00
genre	1.00
time_signature_confidence	0.79
valence	0.23
speechiness	0.54
tempo	155.99
popularity_art	56.00
key	8.00

Figure 12: LIME explainer for low popularity class

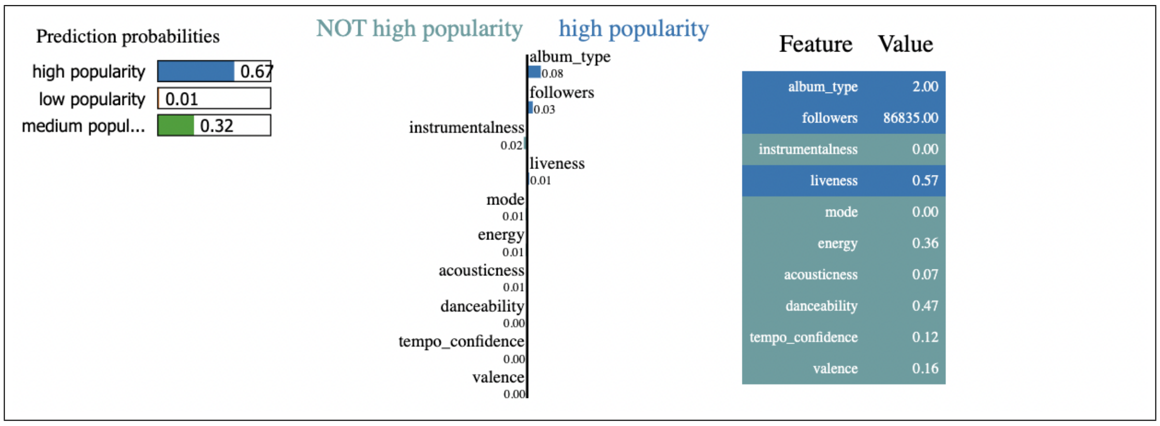
We have an observation classified as low class along with its explanation. The left bars represent features that negatively impact the instance being classified as low, while the right bars represent features that positively impact it. It is interesting to note that, despite the left bars (indicating characteristics that push the observation away from being classified as 'low popularity') being more prominent than the green ones (which positively influence the classification), this observation still has approximately a 60% probability of being predicted as 'low popularity'.



Feature	Value
album_type	0.00
followers	92.00
loudness	0.58
instrumentalness	0.86
mode_confidence	0.30
tempo	145.09
popularity_art	0.00
key_confidence	0.07
key	1.00
time_signature_confidence	0.49

Figure 13: LIME explainer for medium popularity class

In this case of medium class, '*album type*' is the most influential feature on the classification, while track characteristics like '*danceability*,' '*loudness*,' and '*instrumentalness*' seem to have less impact. It's interesting to see how '*followers*' negatively impacts the classification, possibly because the number of followers for this particular observation makes it closer to one of the other two classes.



Feature	Value
album_type	2.00
followers	86835.00
instrumentalness	0.00
liveness	0.57
mode	0.00
energy	0.36
acousticness	0.07
danceability	0.47
tempo_confidence	0.12
valence	0.16

Figure 14: LIME explainer for high popularity class

In this case as well, '*album type*' is the most impactful feature on the classification, while track characteristics like '*tempo*,' '*loudness*,' and '*instrumentalness*' seem to have less impact. It's interesting to see how 'followers' negatively impacts the classification, possibly because the number of followers for this observation makes it closer to one of the other two classes.

Also for high popularity, the main determinant for the classification is '*album type*,' followed by '*followers*,' both of which contribute positively to classifying the instance as '*high popularity*'. Additionally, consistent with what we observed for the 'medium popularity' class, the track characteristics have less impact. However, we can notice that all the track features such as '*instrumentalness*' positively influence the instance to be classified as 'medium popularity' while they have the opposite effect for the class "high popularity".

5.0.2 SHAP

SHAP uses Shapley values to ensure fair and consistent attribution of the model's prediction to individual features, offering both global and local explanations So, we thought it would be meaningful to explain the classifier using this tool.

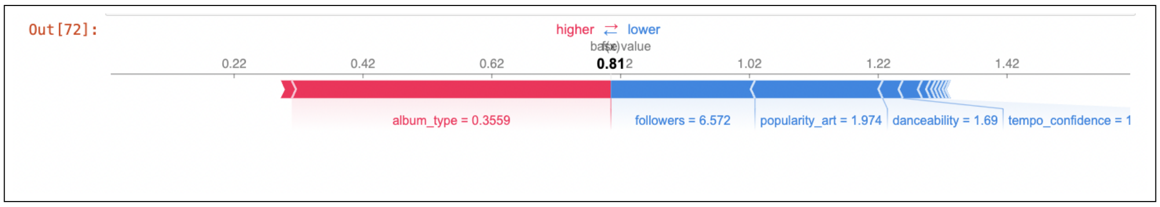


Figure 15: SHAP explainer for low popularity class

As shown in Figure 15, the SHAP explainer indicates that the '*album type*' is the most relevant feature for classifying an observation as having low popularity. This feature significantly contributes to bringing the prediction value close to the base value, which represents the initial average prediction. In contrast, features such as '*followers*', '*popularity_art*', and '*danceability*' decrease the prediction value.

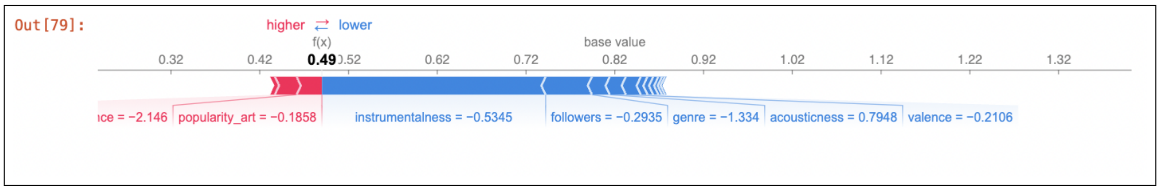


Figure 16: SHAP explainer for medium popularity class

Figure 16 shows an instance classified as 'medium popularity' we can se that '*popularity_art*' is the main feature raising the prediction value while '*instrumentalness*' '*followers*' and '*acousticness*' lower the prediction value differently from what we saw with LIME where all this features were positively contributing to classify the instance as 'medium popularity'.

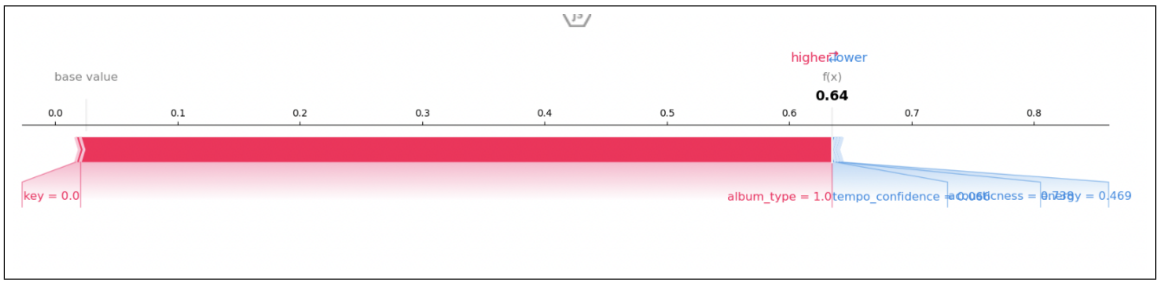


Figure 17: SHAP explainer for high popularity class

Figure 17 shows an instance classified as 'high popularity' , again we can notice how album type almost singlehandedly determines the classification as it raises the prediction value from a value close to 0 of the base value to 0.64 having a contribution of 1 pondered by the other features

6 Time Series Analysis

6.1 Data understanding and Preparation

The dataset, stored as .npy files, has been transformed into numpy arrays for analysis. It represents the first 30 seconds of songs across 20 different genres, with each genre containing 500 songs, resulting in a total of 10,000 time series, each with a length of 1280. To facilitate our analysis, we approximated the time series using various techniques learned during the course. For all the tasks on the time series dataset, we employed Piecewise Aggregate Approximation (PAA). After approximation, the time series length was reduced to 128. In the following section, we did the clustering task and analyze the results. Subsequently, we implemented motifs and anomalies within the different clusters.

6.2 Clustering

In this section, we conducted the cluster analysis on the approximated time series dataset to do so we used two different techniques of clustering: K-Means and Hierarchical. To improve the clustering results, we first reduced the dimensionality of the data using two methods: PCA and t-SNE. For each combination of clustering algorithm (K-Means or hierarchical) and dimensionality reduction method (PCA or t-SNE), we analyzed the characteristics of the resulting clusters, such as mean and standard deviation , and highlight the most frequent genres in each cluster.

6.2.1 K-Means Clustering

To apply the K-means algorithm, we found the optimal number of k by using the Elbow method using Euclidean distance and the Silhouette score

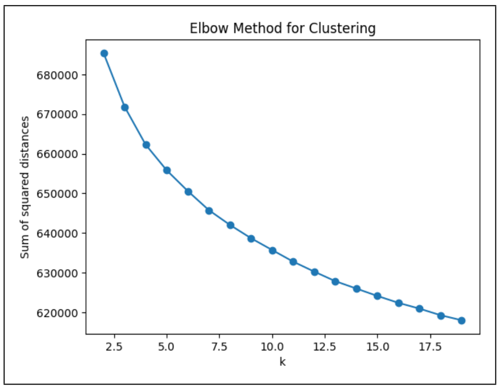


Figure 18: The Elbow method for clustering

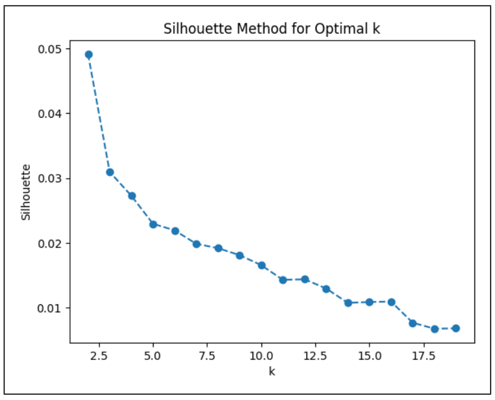


Figure 19: The distribution of the discrete and binary variables

As shown by the figure 18, the elbow method does not provide a conclusive indication of the optimal number of clusters, which seems to range from 4 to 6. However, with the support of the Silhouette score, which decreases as the number of clusters increases, we have decided to opt a value of k = 4.

Principal Component Analysis (PCA) To visualize the time series in a comprehensible manner, dimensionality reduction was performed, allowing us to represent the dataset

in a two-dimensional space. The two dimensions represent the principal components that express the greatest amount of variance in the data.
As seen from the Figure 20, the algorithm did not capture any particular shapes or natural clusters but rather divided the data into four roughly equal clusters with slight differences in the number of time series composing each cluster.

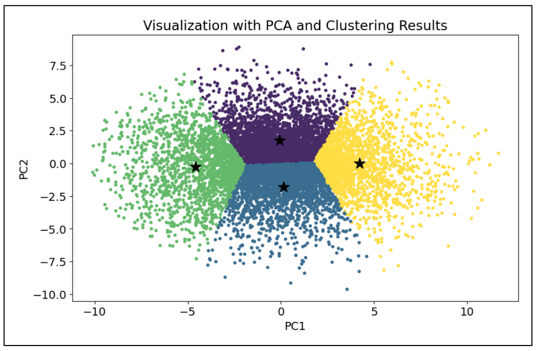


Figure 20: The K-means clustering plot after doing PCA

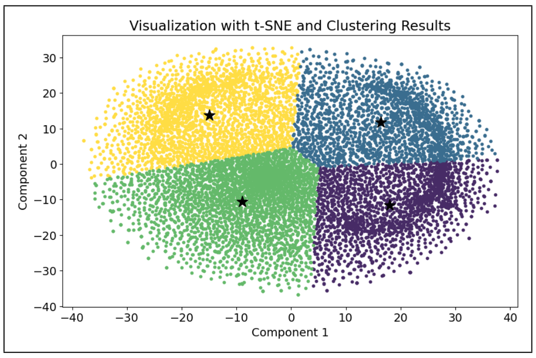


Figure 21: The K-means clustering plot after applying t-SNE

t-SNE t-SNE (t-distributed Stochastic Neighbor Embedding) is a powerful machine learning algorithm used primarily for the visualization of high-dimensional data. It reduces dimensions by embedding high-dimensional points into a two or three-dimensional space, making it easier to visualize complex datasets. Unlike traditional methods like PCA (Principal Component Analysis), t-SNE focuses on preserving the local structure of data points, which means it maintains the distances between nearby points and captures the intrinsic similarities. This characteristic makes t-SNE particularly effective in uncovering patterns and clusters that may not be apparent with other dimensionality reduction techniques
We utilized t-SNE with the following parameters: n_components=2, random_state=0, perplexity=30, learning_rate=200, and n_iter=1000. Each parameter was carefully chosen to optimize the performance and interpretability of the visualization. The perplexity=30 parameter was selected to balance the local and global aspects of the data. Perplexity can be thought of as a smooth measure of the effective number of neighbors.
The figure 21 showns the clustering plot after doing t-SNE. It is not too different from the analysis conducted on PCA, but higher density regions can be observed within some clusters. This is due to the nature of the dimensionality reduction algorithm, which better preserves local distances.

6.2.2 Hierachical Clustering

For the number of clusters, we chose to use the same number of cluster like K-Means algorithm to see the difference between the two techniques
It is evident how the shape of the clusters changes significantly. While with K-Means the boundaries between clusters were straight lines, in this case, they are much more irregular due to the nature of the algorithm. This method does not assume a predefined shape for the clusters and can produce clusters of irregular shapes, as seen in the image. The flexibility of hierarchical clustering makes it suitable for identifying complex structures in

the data, but it can result in less compact clusters.

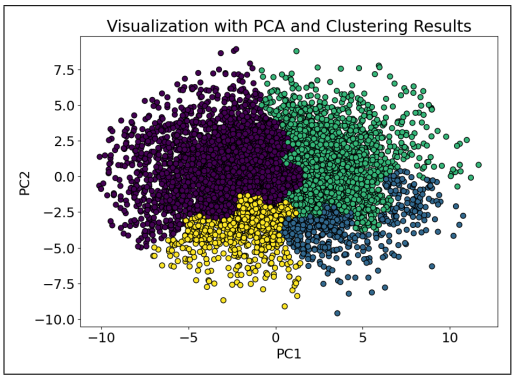


Figure 22: The Hierarchical clustering plot after applying PCA

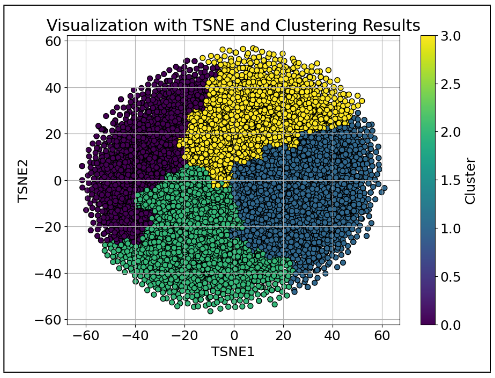


Figure 23: The Hierarchical clustering plot after applying t-SNE

For the t-SNE reduction, we used the same parameters as for the k-means clustering. This decision was made because, after several attempts to modify the parameters, we found that this set was the best for our data. Additionally, using the same parameters makes it easier to compare the two methodologies.
As with hierarchical clustering with dimensionality reduced by PCA, we observe that the boundaries between clusters are irregular. However, the overall shape of the data is more regular compared to the reduction with PCA.

6.2.3 Analysis of Cluster

In this section, we performed analysis with each cluster. Firstly, we conducted the clustering and then grouped the time series according to their respective clusters. In the following section, we compared the clustering results obtained with k-means by relating the two different dimensionality reduction methodologies. Then, we proceeded to do the same for hierarchical clustering, ultimately comparing the results in their entirety.

Mean and Standard Deviation To perform this analysis, we first conducted the clustering and then grouped the time series according to their respective clusters. In the following section, we compared the clustering results obtained with K-Means by relating the two different dimensionality reduction methodologies. Then, we proceeded to do the same for hierarchical clustering, ultimately comparing the results in their entirety.

There are evident similarities in the trends of mean and standard deviation between the two methods of dimensionality reduction. For instance, in figure 24, cluster 3 (left) and cluster 2 (right) have substantially the same trends. However, some differences are also interesting. For example, in the clustering performed with t-SNE, there is a cluster that maintains a constant mean from start to finish, while no cluster behaves similarly in the clustering performed with PCA.

Similarly, it is evident that, in figure 25 cluster 3 (left) and cluster 1 (right) exhibit the same behavior, which can also be observed in the k-means clustering. This seems to be the most consistent cluster.

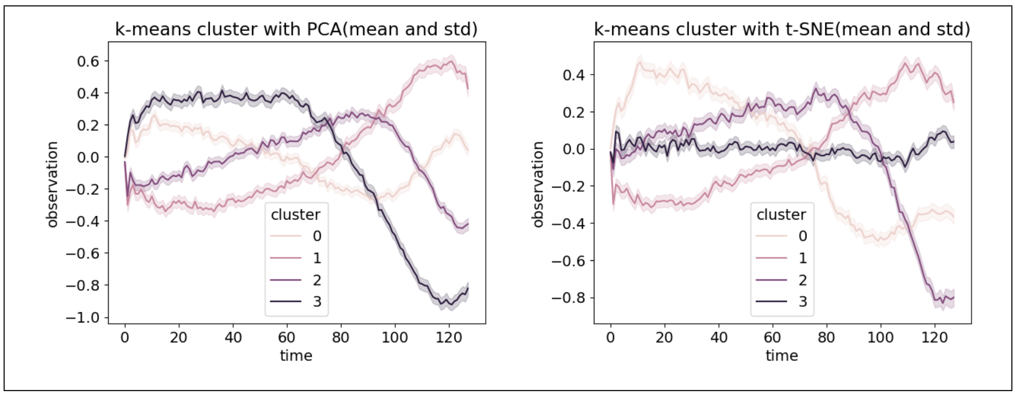


Figure 24: Mean and Standard Deviation of K-Means with PCA (left) and with t-SNE (right)

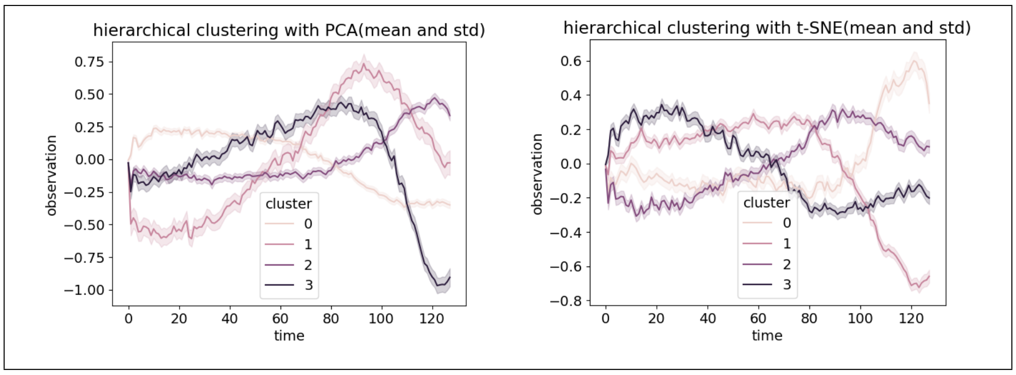


Figure 25: Mean and Standard Deviation of Hierarchical Clustering with PCA (left) and with t-SNE (right)

Genre Analysis To further understand the trends and uncover patterns, we examined the most frequent genres for each cluster. As done with the mean and standard deviation, we will first compare the clusters obtained with K-Means, followed by those obtained with hierarchical clustering.

Figure 26 shows the case in K-Means clustering, it is interesting to see that clusters 3 (up) and 2 (down), which showed similar trends in mean and standard deviation, share three out of five genres. This suggests a possible pattern in the behavior of those genres.

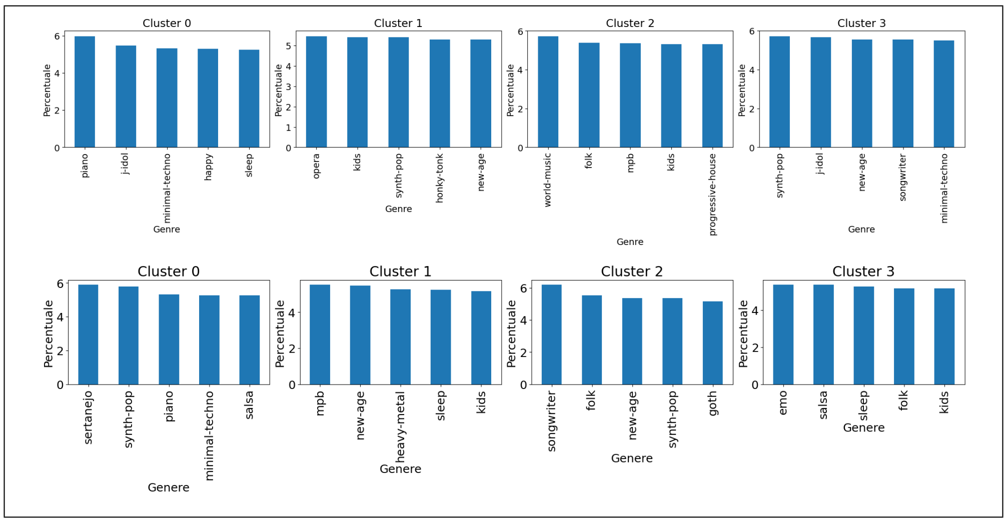


Figure 26: Top five genres per cluster by K-Means with PCA (up) and t-SNE (down)

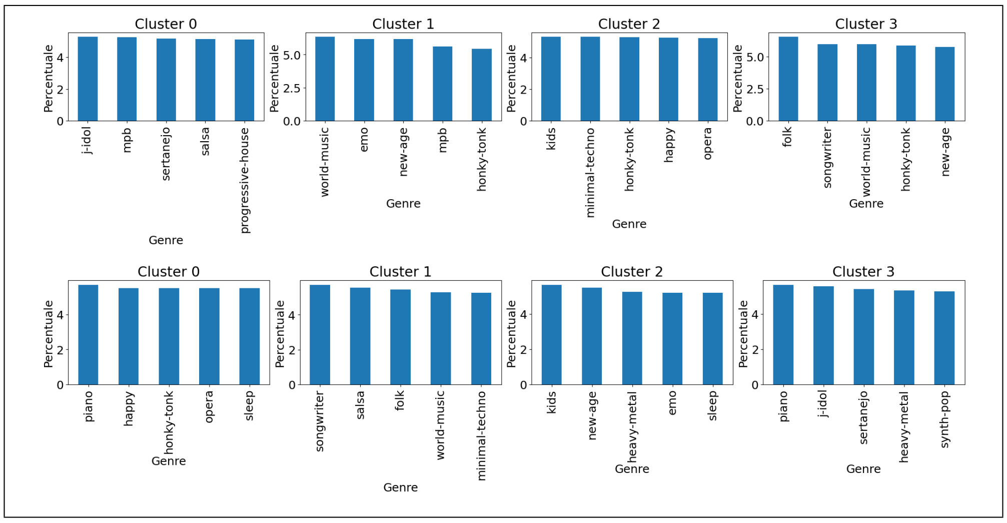


Figure 27: Top five genres per cluster by Hierarchical Clustering with PCA (up) and t-SNE (down)

Similarly, clusters 3 (up) and 1 (down), figure 27 which correspond to the most common trends, share three out of five genres in the top five. This further supports the presence of a pattern in the behavior of these genres.

6.3 Motif and Anomalies

In this section, we analyzed the motifs and anomalies. Motifs are frequent patterns within individual time series, while anomalies are the least frequent patterns. To identify motifs and anomalies, a matrix profile is used. This consists of an array that stores the minimum distances between all subsequences of a time series.
For our analysis, we extracted several time series from each cluster obtained from the previous analysis and identified the motifs and discords for each of these time series. The analysis was performed on the clusters derived from each of the methodologies used in the previous section. However, for brevity, we show only one of them below.

6.3.1 Motifs Hierarchical Clustering

In figure 28, we show the motifs and discords for four time series, one for each cluster. We used a window size of w = 5 for the subsequences, as after several attempts with different window sizes, this parameter provided the clearest and most significant results. We searched for the top three motifs.

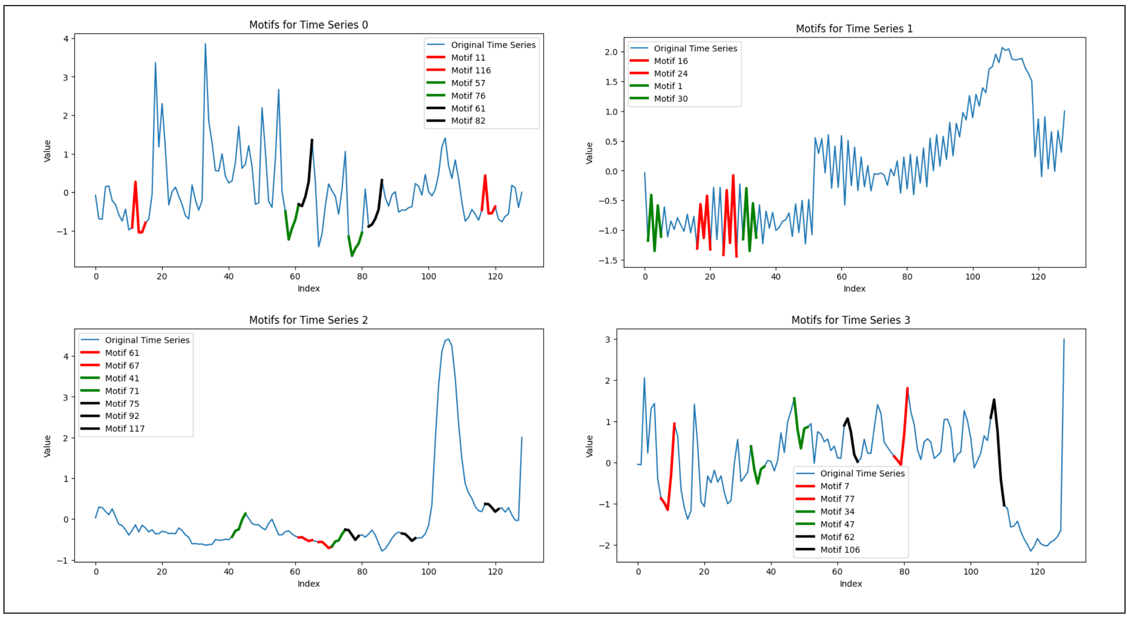


Figure 28: Motifs for the time series belonging to different clusters

It is interesting to note that in the time series of cluster 1, the algorithm found only the top 2 motifs instead of 3. This is likely because, beyond the first two subsequences, no other subsequence is repeated in the time series.
It is interesting to note how the time series have very different shapes depending on the cluster, and consequently, the shapes of the motifs also change.

6.3.2 Anomoly Discovery

After having seen the motifs in the timeseries, the anomalies are observed. Using the same parameter w = 5 , we searched for anomalies in the same time series. The figure 29 in the next page shows the anomalies for the time series belonging defferent clusters.

These anomalies highlight the unique and infrequent patterns within each cluster's time series, providing insights into the less common behaviors that may be of interest for further analysis.

6.4 Shapelet Based Classification

Two approaches were employed to perform a task: one considering all classes in the dataset and another performing binary classification using time series from 2 out of 20 genres.

First Approach The dataset was processed (normalized and approximated with PAA) and split into 70% training and 30% test sets. Shapelets were extracted using the Shapelet-TransformClassifier model, which autonomously extracts shapelets, transforms the dataset,

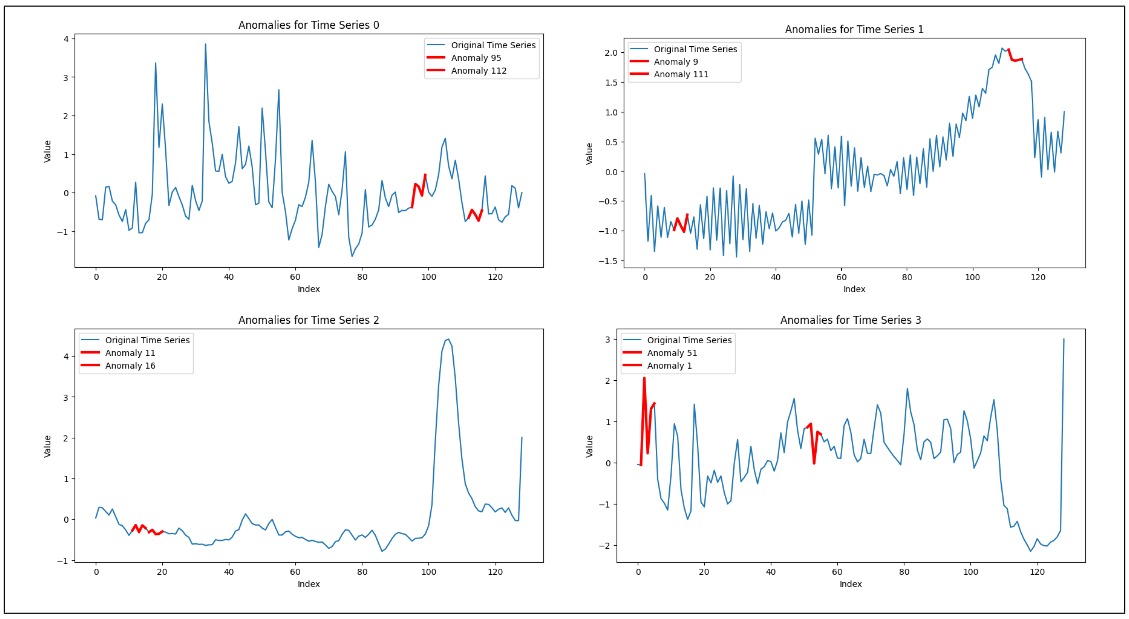


Figure 29: Anomalies for the time series belonging to different clusters

and classifies the shapelets using rotation forest. The table 17 below summarizes the parameters and test set performance:

Model	**No. of Shapelet Samples**	**Max Shapelet Length**	**Max. Shapelets per Class**	**Test Set Accuracy (%)**
Shapelet Transform Classifier	1000	10	100	17

Table 17: Performance of the Shapelet Transform Classifier Model

Despite the difficulty of discriminating between many classes, the model performs well, possibly due to the use of rotation forest over simpler classifiers.

Second Approach To evaluate the effectiveness of shapelets in genre discrimination, the dataset was filtered to include only instances belonging to the genres "heavy-metal" and "salsa." The data was then normalized and split into 70% training and 30% test sets for model evaluation.
The following pipeline was executed to train and evaluate the shapelet-based classification model:

1. **Shapelet Extraction:** Shapelets were extracted from the training set using the ShapeletTransform model provided by the ‘py-ts‘ library.

2. **Feature Transformation:** Both training and test set features were transformed into a Shapelet dataset suitable for classification.

3. **LightGBM Classification:** A LightGBM classifier was used to classify the time series instances in the test set.

4. **Hyperparameter Tuning:** The best hyperparameters for the LightGBM classifier were determined using RandomizedSearchCV for efficient exploration of the parameter space.

The final model parameters obtained after hyperparameter tuning will be listed and discussed in the following section:

Shapelet Extraction	**Parameters**	**Classification**	**Parameters**	**Accuracy on Test Set (%)**
ShapeletTransform()	window_sizes: [5, 12, 24, 36]	LGBMClassifier()	boosting_type: 'gbdt' max_depth=1 num_leaves=31 objective='binary'	65

Table 18: The optimal parameter proposed by RandomSearchCV

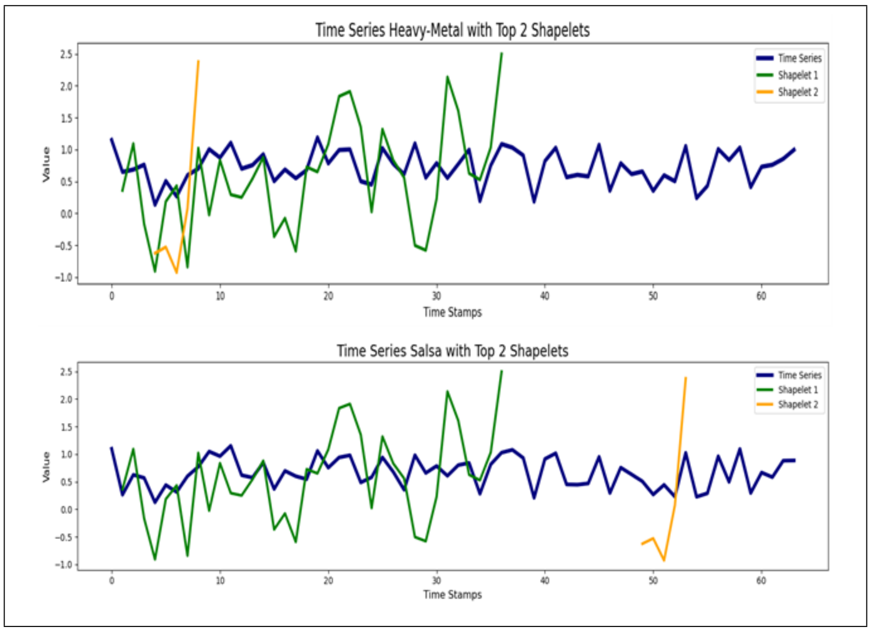


Figure 30: Top 2 Shapelets aligned time series of "Heavy-Metal" (top) and "Salsa" (down)

The figure 30 above represents two time series from the genres 'heavy-metal' and 'salsa' correctly classified by the model. The first observation concerns the differences in the length of the top-2 most discriminative Shapelets, suggesting that satisfactorily discriminating between the two is not as straightforward as one might think based on intuition, but rather it seems necessary to have a particularly long subsequence. Observing the length of the second most discriminative Shapelet, it is immediately noticeable that it is much shorter than the first, indicating the presence of a nuance in the songs that marks a clear difference between the two genres. Intuitively, this could be due to the frequencies generated by the strong presence of Caribbean-origin instruments that characterize the 'salsa' genre, a feature completely absents in the tracks related to the 'Heavy-Metal' genre.

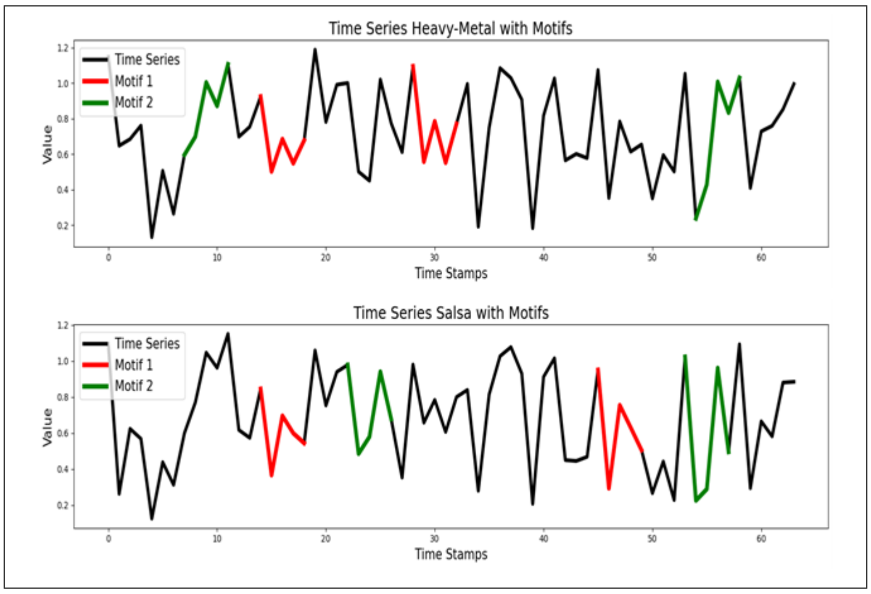


Figure 31: Top 2 Motif and time series of "Heavy-Metal" (top) and "Salsa" (down)

The figure 31 displays the top 2 motifs (sliding window = 5) from 'Heavy-Metal' and 'Salsa' time series, showing no relationship with the class variable as they depict repeated patterns within the same series. These motifs likely reflect repeated frequencies, such as guitar riffs in heavy metal or horn sections in salsa choruses. Thus, motifs seem more related to rhythmic and compositional structures rather than genres. Conversely, Shapelets capture nuances indicating clear genre differences.

6.5 Time Series Classification

Given the difficulty in achieving satisfactory accuracy for classifying each of the 20 genres, with the highest accuracy obtained through DTW classification being only 0.15, we have decided to focus on a targeted classification task for a subset of genres. The selection of genres is based on data obtained from cluster analysis in figure 32, this is the results of Hierarchical Clustering that we completed in clustering section. Therefore, we have decided to pursue the following classification tasks:

1. Classify the most frequent genre of every cluster
2. Classify four of the most frequent genres within one of the clusters.

Both of these tasks will be addressed using K-NN with both Euclidean distance and Dynamic Time Warping (DTW).

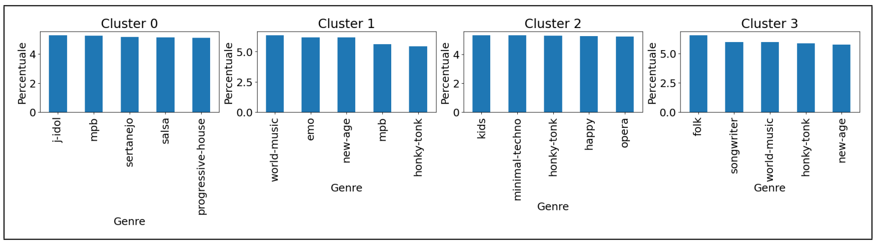


Figure 32: Top five genres per cluster by Hierarchical Clustering with PCA

6.5.1 Classify the most frequent genre of every cluster

To perform the classification, we first approximated the time series using PAA and removed the noise by filtering the dataset for the genres of our interest: '*folk*', '*j-idol*', '*kids*', and '*world-music*'. Subsequently, we searched for the value of k that maximizes the accuracy of our model.

Genre	**Euclidean Distance**			**DTW**		
	Precision	**Recall**	**F1-score**	**Precision**	**Recall**	**F1-score**
folk	0.23	0.26	0.24	0.23	0.30	0.26
j-idol	0.24	0.22	0.23	0.34	0.42	0.38
kids	0.19	0.10	0.13	0.24	0.21	0.23
world-music	0.24	0.35	0.28	0.30	0.20	0.23
Accuracy	0.23			0.28		

Table 19: Performances of the most frequent genres of every cluster by Euclidean and DTW distance

K-NN with Euclidean distance We found the highest precision value for k = 7 through grid search. The K-NN model using Euclidean distance demonstrates limited effectiveness, achieving an overall accuracy of 0.23. The model performs relatively better in identifying '*world-music*' instances, with a recall of 0.35 and an F1-score of 0.28, indicating a higher ability to find instances of this category despite some false positives. However, the model struggles significantly with the '*kids*' category, showing a very low recall of 0.1 and an F1-score of 0.12, highlighting its difficulty in correctly identifying and recalling instances of this category.

K-NN with Dynamic Time Warping (DTW) We found the highest precision value at k = 5. The K-NN model using Dynamic Time Warping (DTW) shows an improved overall accuracy of 0.28 compared to Euclidean distance. The most notable performance is in the '*j-idol*' category, with a precision of 0.34, recall of 0.42, and an F1-score of 0.38, indicating a higher effectiveness in identifying and correctly classifying '*j-idol*' instances. In contrast, the model has reduced performance in the '*world-music*' category, with a recall of only 0.2 and an F1-score of 0.24, reflecting a lower ability to recall these instances accurately.

6.5.2 Classify four of the most frequent genres within one of the clusters

We selected four of the most frequent genres from cluster 2 for this classification task: '*minimal-techno*', '*opera*', '*happy*', and '*honky tonk*'. The table 20 shows the performance of this section by 2 types of distance Euclidean and DTW.

K-NN with Euclidean distance We set k was equal to 3, for this classification as it was the best parameter we found with grid search, The classification results using Euclidean distance reveal that while the method achieves moderate overall accuracy (0.43), its performance varies significantly across different categories. The "*happy*" category shows a high recall (0.72) but low precision (0.345), indicating that while many true "*happy*" instances are correctly identified, there is a high rate of false positives. "*minimal-techno*" demonstrates a balanced performance with the highest precision (0.56) and recall (0.65), resulting in the best f1-score (0.60) among the categories. However, the method struggles

Genre	**Euclidean Distance**			**DTW**		
	Precision	**Recall**	**F1-score**	**Precision**	**Recall**	**F1-score**
minimal-techno	0.34	0.72	0.47	0.47	0.67	0.55
opera	0.48	0.31	0.38	0.58	0.46	0.52
happy	0.56	0.65	0.60	0.60	0.67	0.63
honkey-tonk	0.33	0.02	0.04	0.56	0.37	0.44
Accuracy	0.43			0.54		

Table 20: Performances of the most frequent genres for cluster 2 by Euclidean and DTW distance

with the "*opera*" category, exhibiting a very low recall (0.02) and f1-score (0.04), suggesting difficulty in correctly identifying "*opera*" instances. These results highlight that while Euclidean distance can be effective for certain categories, it may not be equally reliable across all, particularly for those with less distinctive features like "*opera*".

K-NN with Dynamic time warping (DTW) K was set equals to 7, The classification results using Dynamic Time Warping (DTW) demonstrate an overall improvement in performance compared to Euclidean distance, with a higher overall accuracy (0.54). The "*happy*" category maintains a high recall (0.67) and improves in precision (0.47), resulting in a better f1-score (0.55), indicating a more balanced identification of true instances with fewer false positives. "*minimal-techno*" continues to perform well, achieving the highest precision (0.60), recall (0.67), and f1-score (0.63) among all categories, suggesting DTW's effectiveness in capturing the temporal dynamics of this category. The "*opera*" category also shows significant improvement, with increased precision (0.56), recall (0.37), and f1-score (0.44), indicating better identification and fewer misclassifications. Overall, DTW enhances the classifier's ability to accurately recognize and differentiate between categories, particularly for those with distinct temporal patterns. It is important to note that classification using DTW is extremely time-consuming, as it required approximately 3.5 hours of running time for both classifications.

6.6 Advanced Classification

In this section, other time series classification techniques were explored. A binary classification was performed on the filtered dataset, considering the time series of the genres 'Heavy metal' and 'Salsa'. Three different classifiers were tested: Convolutional Neural Networks (CNN), ROCKET, and MrSEQL.

Convolutional Neural Networks (CNN) For this classifier, GridSearchCV was used to search for parameters among the following:

1. 'n_epochs': [1000, 2000],
2. 'batch_size': [16, 32],
3. 'n_conv_layers': [2,4],
4. 'activation': '*softmax*', '*relu*',

The parameters that resulted in the best performance were: (1000), (32), (2), '*softmax*'. The performances are listed table 21 .

Model	Accuracy	Precision	Recall	F1-score
CNNClassifier	0.57	0.57	0.54	0.55
RocketClassifier	0.70	0.69	0.70	0.70
MrSEQL	0.59	0.59	0.58	0.58

Table 21: Performances of the Advanced Time Series Classification models

ROCKET Classification was conducted using the ROCKETClassifier, with parameter search via GridSearchCV among kernel numbers [1000, 5000, 10000], revealing that 10000 is the optimal parameter. The results we added in table 21

MrSEQL Finally, the MrSEQL model was tested, which is considered state-of-the-art in time series classification. The model does not have many parameters, so GridSearchCV was not necessary. Instead, various tests were performed by varying the only available parameter: the type of approximation applied to the dataset. The most convincing result was obtained with the SAX approximation.
In summary, ROCKET is the model that shows the best ability to capture the relationship between features and the class variable.

7 Conclusion

In this report, we had the opportunity of working with a large dataset that put at our disposal, many alternatives for solving various advanced classification and clustering techniques. In Chapter 1, we did data understanding tasks, with several techniques to clean anomalies, merge 2 datasets as well as visualize some numerical features distribution. In Chapter 2, we performed the Anomalies Detection with 8 techniques to detect top 1% of anomalies. After that, majority voting is used to determine the most outliers in all of these algorithms. At the end of this step, we removed totally 215 records and it helped to improve the quality of dataset. Furthermore, the Imbalanced Learning was applied in this chapter. The data is classified based on "*explicit*" feature which already has imbalance binary class. Several algorithms of Oversampling and Undersampling were employed to improve the performance. We did the classification of KNN and Decision Tree to compare the results between before and after doing Imbalanced Learning. Finally, there was no techniques helped to improve the performance. In chapter 3, we applied the advanced classification models such as Multi-Layer Perceptron, Linear and Non-Linear SVM, Logistic Regression, Ensembled Methods and Gradient Boosting. The "*popularity*" class was chosen and encoded into "*high*", "*medium*" and "*low*" of popularity. We performed the hyper-parameter tuning phases to find the optimal parameters. The Bagging Method with None of Estimator gave the best results in this chapter. In chapter 4, we did the advanced regression models. We applied 2 techniques of Multilayer Perceptron Regress and Random Forest Regress. The hyper-parameters tuning were used to find the best parameters. The value of "*acoustic*" were used to predict by other features in dataset. The results of Random Forest Regression provided slightly better than the Multilayer Perceptron Regression. In chapter 5, the Explainability were performed with LIME algorithm. The task was deployed based on previous advanced classification to understand how the model work in a simple way. In chapter 6, we implemented the Time Series analysis with several tasks such as data understanding, approximation, clustering, motif and classificaion.
In conclusion, the project helps us to enhance our ability in reasoning and analytic skills as well as consolidate our knowledge in the theoretical concepts covered in the course.

We would like to say thank you to Professor R.Guidotti and Teaching Assistant A.Fedele. Your insightful lectures and recommendations throughout our discussions were useful to complete this project.