Data Mining II Project

Dataset For Music Analysis

Federico Ceciliani - 620342
Michele Andreucci - 628505

UNIVERSITÀ DI PISA

MSc in Data Science and Business Informatics
Department of Informatics
Università di Pisa

June 2021

Table of Contents

1 Data Understanding

The **Dataset For Music Analysis** reports the data of 106.574 tracks (objects) with their respective 53 attributes where we can find useful information about the the license typology, interest of the track, information of the album, creation of the album.

1.1 Assessing Data Quality

1.1.1 Missing Values

First of all, we analyzed the data set from a point of view of **Missing Values**.

Data columns with too many missing values are unlikely to carry much useful information. For this reason, we opted to remove all the columns with a ratio of missing values greater than a specific threshold. In our case, the value of our threshold was equal to 50%. In addition to that, we dropped the 7% of the rows with missing values.

Afterwards, it remained five attributes with missing values. They were all categorical features, so we filled them with the **Mode** value. To be more precise, we chose that method due to the fact that if the attribute is categorical, then the most commonly occurring attribute value can be taken.

1.1.2 Pairwise correlations and elimination of redundant variables

At this stage of our analysis, we looked for the **Pairwise Correlation**. Data columns with very similar trends are also likely to carry very similar information, and only one of them will suffice for classification.

We decided to remove all the features with a correlation higher than 0,5 because redundant variables.

1.1.3 Variables Transformation

Four attribute which needed some further investigation were AlbumDateCreated, AlbumDateReleased, ArtistDateCreated and TrackDateCreated.

After converting them to the right format it was possible to separate their information into new attributes: day, month, year and daytime: the last one, in particular, was divided between morning, afternoon, evening and night. For what concerns the former, the division between daytime could be really interesting, even more than the date itself.

In addition to that, we transformed all the categorical attributes in our data set in continuous one. We used the **Label Encoding** approach. This method is very simple and it involves converting each value in a column to a number.

1.2 Target Variables

Among our variables, we decided to consider as our target one the variable named **TrackLicense**. We analyzed the distribution of this particular feature as shown in Figure 1. We took in consideration the most popular license existing.

The target variable to predict is whether a track had a 'Attribution-Noncommercial-Share Alike 3.0 United States' license or not. So we need to transform the TrackLicense attribute into a binary feature.

Consequently there are two possible outcomes: "1" (17.680 values) and "0" (81.724 values). This demonstrates us that 17,786% of the tracks has that so popular license label. [Figure 2]

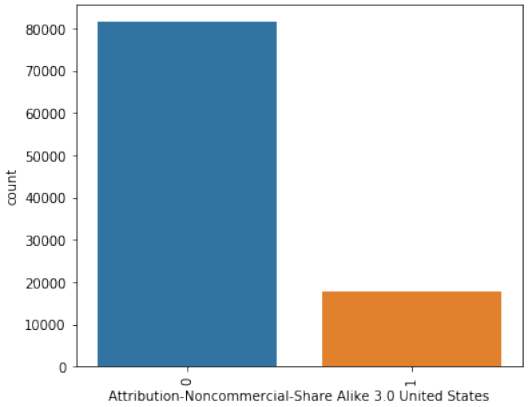

Figure 2: Target Variable.

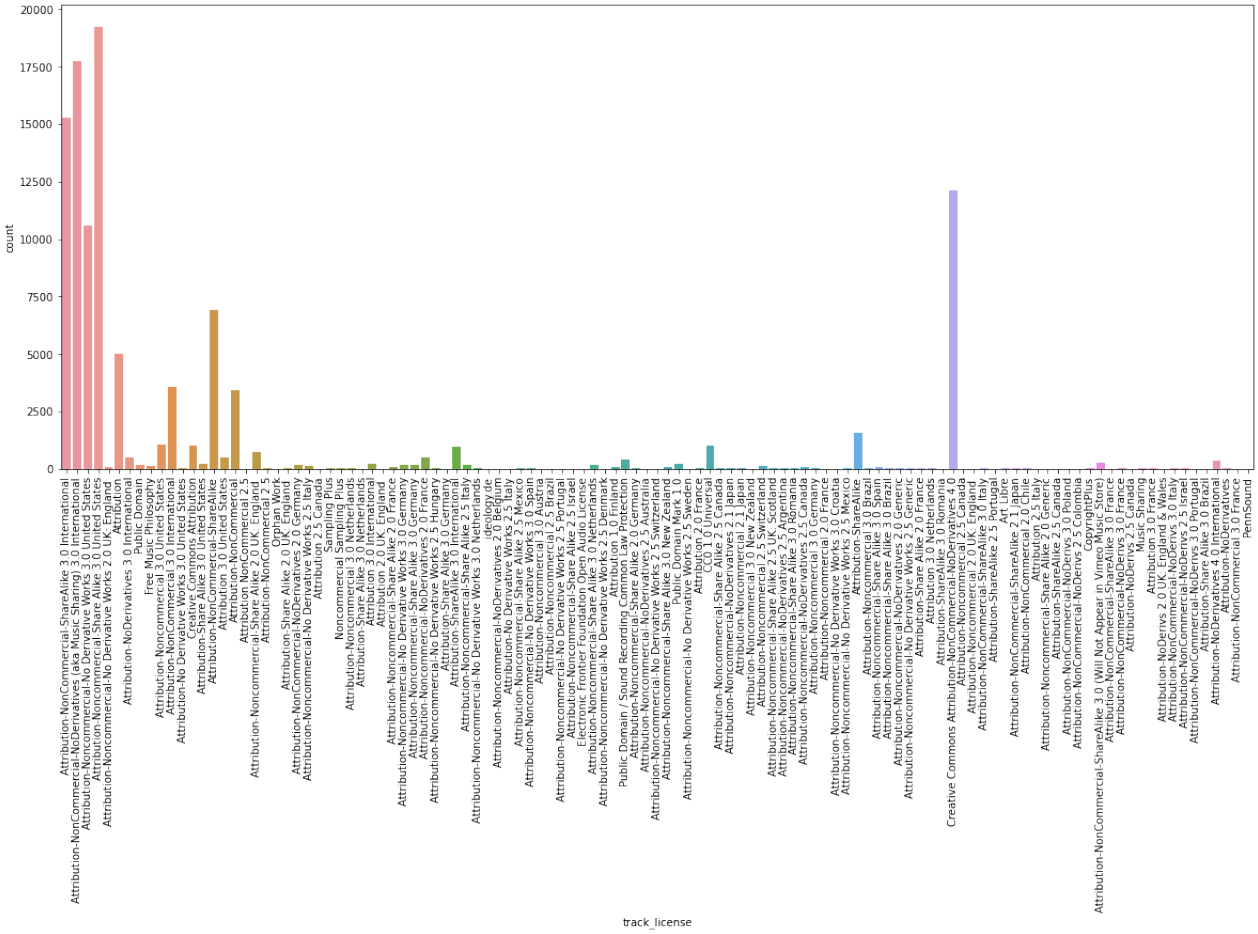

Figure 1: Variable Distribution.

1.3 Most Important Features

Despite our considerations, we still had a too large data set with a huge amount of data.

We tried to select the most important features using the **Random Forest**. It comes under the category of *Embedded methods*, that combine the qualities of filter and wrapper methods.

What we found out is shown in Figure 3. There are represented the 15 features in order of importance in our data set.

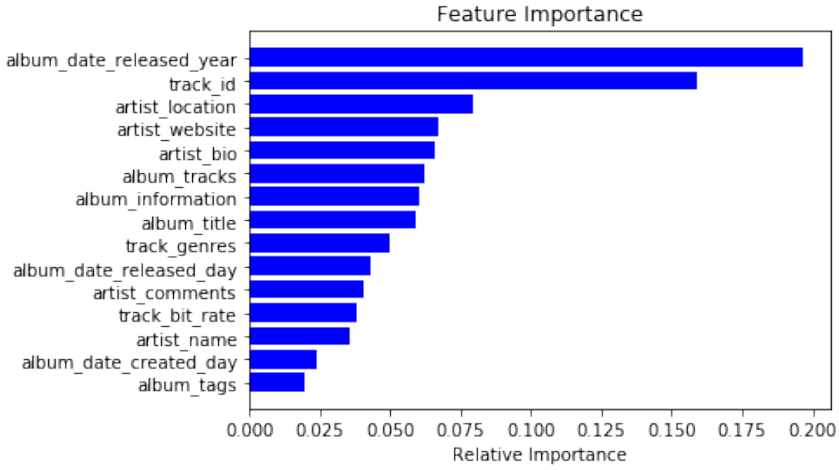

Figure 3: Feature Importance.

2 Classification

The classification analysis of the **Dataset For Music Analysis** was performed using the *Scikit-learn* machine learning library.

Before implementing the machine learning models, we used the **Holdout Method**, the simplest sort of method to evaluate a classifier. In this method, the data set is separated into two sets, called the Training set and Test set; in our case, the data set was split into 70% Training and 30% Testing in a stratified fashion so that the training and test sets had the same proportions of our class name labels, 'Attribution-Noncommercial-Share Alike 3.0 United States', as the input data set.

From a total of 99.404 instances, the training and test data sets contain 69.582 and 29.822 instances, respectively.

The goal is to construct an efficient model that were able to predict, based on the values of the other features, whether the target variable took value 1 or 0.

In this section, two different classifiers were used to perform the classification analysis: the **Decision Tree Classifier** and the **K-NN**.

The data set obtained after the Data Understanding section was used to be fed into the classification algorithms.

2.1 Decision Tree

The **Decision Tree Classifier** was used as a first classification algorithm. The default hyper-parameters were used: we adopted the *Gini* criterion, we did not imposed a maximum depth, we assigned a value of minimum samples split equal to 2 and a value of minimum samples leaf equal to 1.

Figure 4 shows the feature importance generated according to the model. Based on this chart, we can conclude that TrackId plays a very important role in deciding our specific license, the *target* variable. Apart from that, ArtistLocation, AlbumTitle and AlbumInformation are also among the top contributors. On the other hand, features such as ArtistComments and AlbumTags tend not to contribute as significantly.

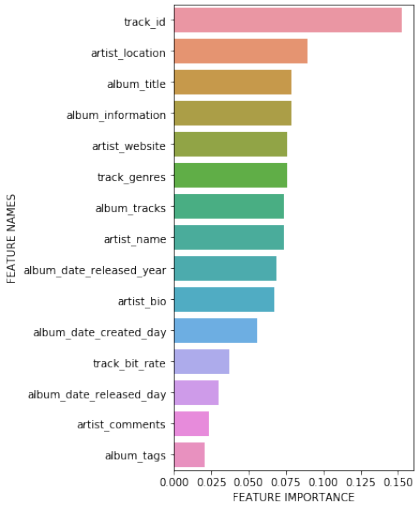

Figure 4: Feature importance according to the Decision Tree Classifier with default hyper-parameters.

For a matter of space, we only show the firsts nodes of Decision Tree which was built using the model with default parameters [Figure 5].

The root node is represented by the AlbumDateReleasedYear feature, with a total of 69.582 samples, the number of instances in the training set. The model partitioned the instances among the two possible Attrition classes, 57.206 for License No and 12.376 for License Yes. The class predicted by the Decision Tree is License No as it is the most numerous class at the root node.

The first decision boundary of the tree is AlbumDateReleasedYear with a value lower or equal than 2009,5. The criterion adopted was the **Gini** one. Applying the same principle again and again, the Decision Tree then used the TrackId and the TrackBitRate features to split the second and third node in two, respectively.

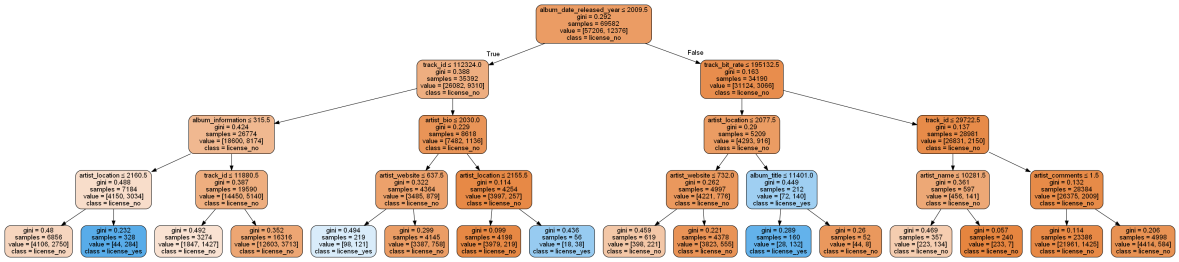

Figure 5: Decision Tree built with default hyper-parameters.

Hereinafter in Figure 6 and 7 are shown our results.

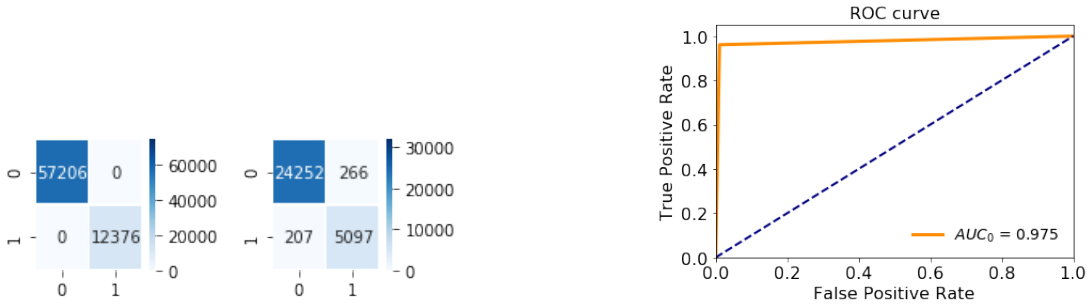

Figure 6: Confusion matrix according to the Decision Tree Classifier for the Training and Test set with default hyper-parameters *(left)* and ROC curve *(right)*.

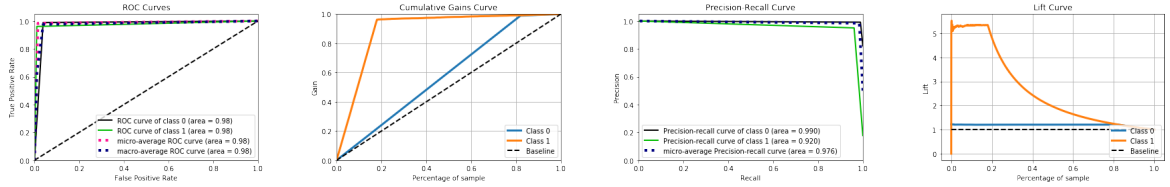

Figure 7: ROC curves, Cumulative Gains curve, Precision-Recall curve and Lift curve for the Decision Tree Classifier.

2.2 KNN

The **KNN classifier** works by computing the distance between the record to be classified and every other record in the training set, basing its prediction on the most common value in the neighborhood of the point, precisely between its *K* nearest neighbours. The metric used to compute the distances between objects was the *Euclidean* distance.

The data was normalized by using the *MinMaxScaler* which scaled and translated each feature individually in the range between 0 and 1.

In this case, three different models were implemented, shown in the following points.

2.2.1 KNN - Default Hyper-Parameters

In the first model, we used as default hyper-parameters a number of neighbors equal to 3, 5, 7, 9 and 11.

We found the best results with K equal to 3. They are shown in Table 1 and Figure 8 and 9.

KNN Model with 3 neighbors	
Train Results	**Test Results**
Accuracy Score: 98.65%	Accuracy Score: 96.92%
Precision Score: [99.07%, 96.69%]	Precision Score: [97.83%, 92.54%]
Recall Score: [99.29%, 95.70%]	Recall Score: [98.43%, 89.93%]
F1 Score: [99.18%, 96.19%]	F1 Score: [98.13%, 91.22%]

Table 1: Classification reports for the KNN model with K=3 neighbors.

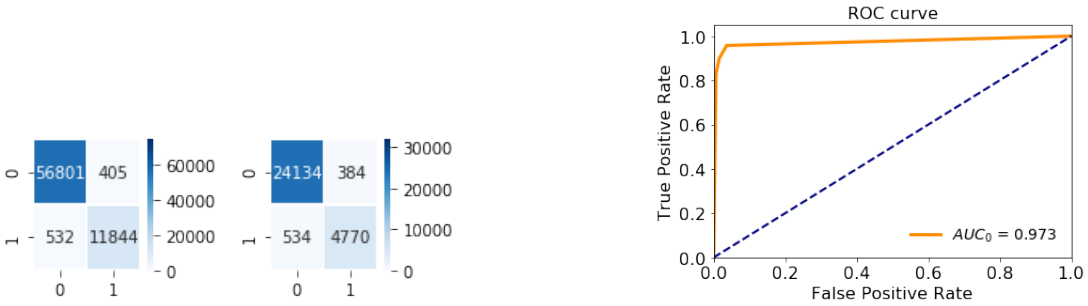

Figure 8: Confusion matrix with K=3 neighbors for the training and test set *(left)* and ROC curve *(right)*.

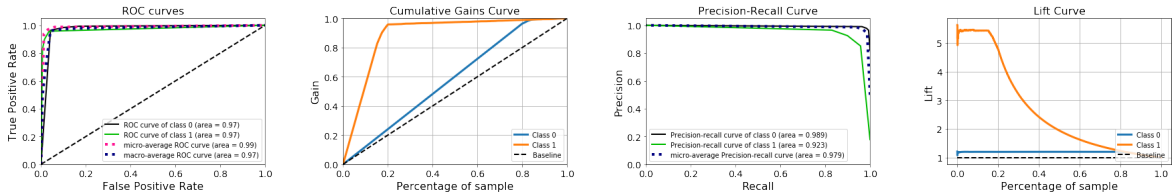

Figure 9: ROC curves, Cumulative Gains curve, Precision-Recall curve and Lift curve with K=3 neighbors.

2.2.2 KNN - Random Search CV

In the second model, the hyper-parameters were optimized using the *Random Search CV* with 5-fold cross validation. The optimal number of neighbors was found to be 46. The results are expressed in Figure 10 and Figure 11.

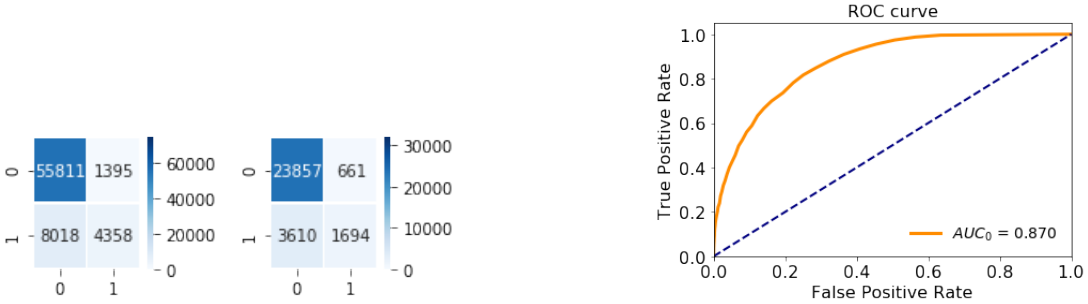

Figure 10: Confusion matrix with optimized parameters for the training and test set *(left)* and ROC curve *(right)*.

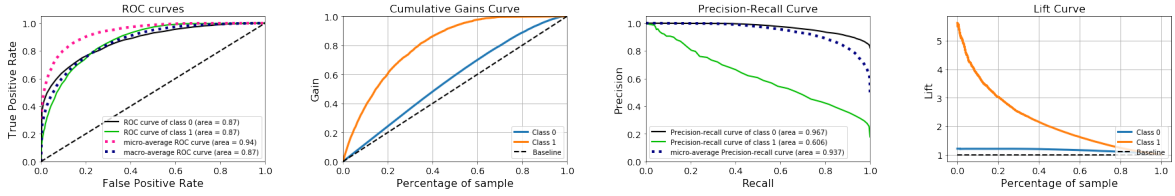

Figure 11: ROC curves, Cumulative Gains curve, Precision-Recall curve and Lift curve with K=46 neighbors.

2.2.3 KNN - Grid Search CV

In addition to that, we used the *Grid Search CV* to find out the optimal number of neighbors based on the research of the best one. The optimal number was 1. [Figure 12 and Figure 13]

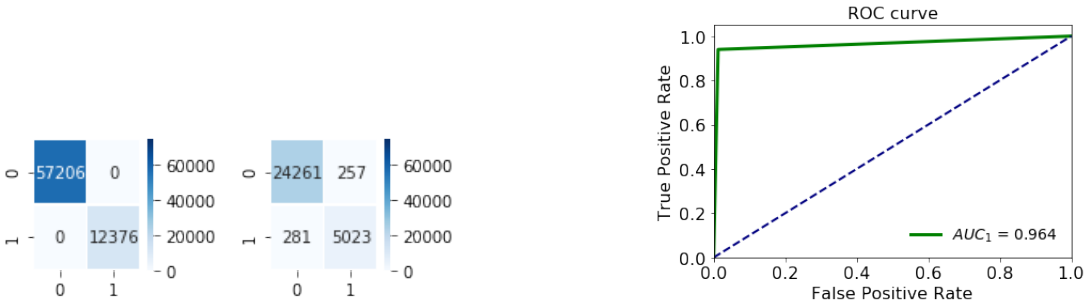

Figure 12: Confusion matrix with K=1 neighbors for the training and test set *(left)* and ROC curve *(right)*.

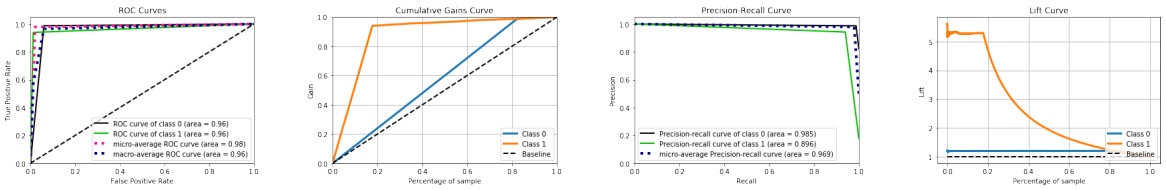

Figure 13: ROC curves, Cumulative Gains curve, Precision-Recall curve and Lift curve with K=1 neighbors.

3 Outlier Detection

Outliers are observations in a data set that do not fit in some way. Perhaps the most common or familiar type of outlier is the observations that are far from the rest of the observations or the center of mass of observations.

Different approaches were analyzed and the results are shown in the following points.

3.1 Boxplot - Visual Approach

The first and most basic outlier detection method to be applied was the visual approach.

Boxplots were generated, as shown in Figure 14. This type of plot is used to easily detect outliers. It can also tell us if the data is symmetrical, how tightly it is grouped, and if and how the data is skewed.

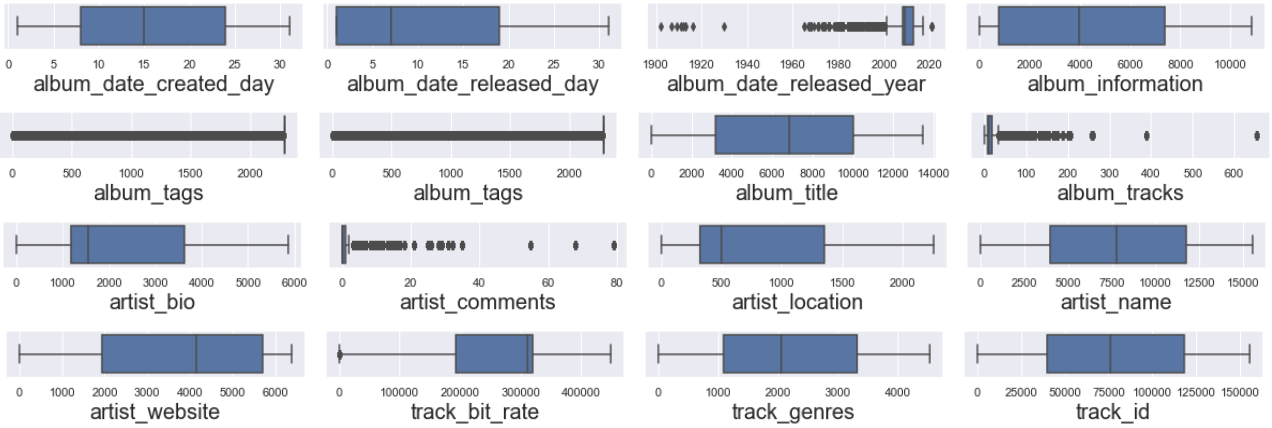

Figure 14: Boxplots.

Hereinafter, more advanced outlier detection techniques will be discussed. Such methods are not just simple visual approaches, but they exploit other characteristics of the records, such as distances between the records or others measures and statistics.

3.2 KNN - Distance Based

The fundamental assumption in the **Nearest-Neighbor** family is that similar observations are in proximity to each other and outliers are usually lonely observations, staying farther from the cluster of similar observations.

This distance-based approach was the one that assign to each point its distance from its KNN as outlier score, using a value of k equal to 15.

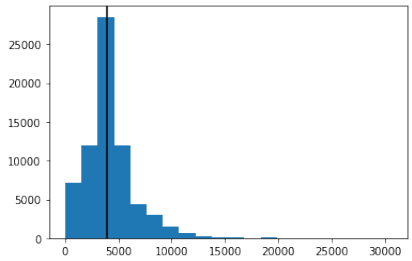

Figure 15: KNN outliers with K=15.

3.3 DBSCAN - Density Based

Unfortunately, *distance-based* approaches are very sensitive to density variations, which is why also a *density-based* techniques were tested.

The foundation for this type of approaches is trying to compare the different densities of the points with the ones of their local neighbours, assuming that normal data would have similar values, while outliers are expected to have a considerably different density from their neighbours.

This method is simple, easy to understand, it needs only two parameters to set and scales well but, on the other hand, it is sensitive to sparse global sampling. In fact, the clustering algorithm not only works without needing the number of clusters in advance, but it also identifies outliers as points not belonging to any of the individuated clusters.

DBSCAN is a simple and popular density-based unsupervised algorithm for outlier detection. It uses distance, *epsilon*, and a minimum number of points per cluster to classify a point as an outlier: in our case we used respectively 0,2, after the adoption of the knee method, as shown in Figure 16 (*left*), and 32.

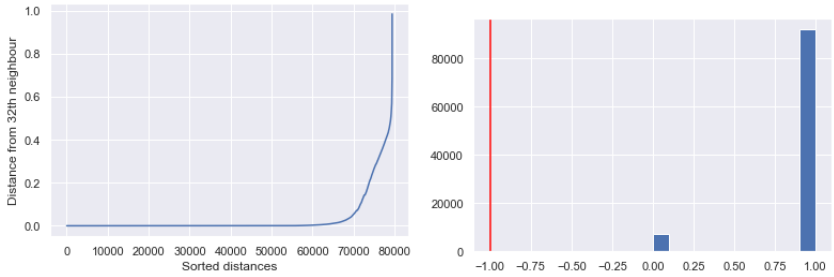

Figure 16: Knee Method *(left)* and Density-Based outliers *(right)*.

The main problem with this type of approaches is how to compare the neighborhood of points from areas of different densities.

3.4 LOF - Density Based

The **Local Outlier Factor** (*LOF*) algorithm is an unsupervised anomaly detection method which computes the local density deviation of a given data point with respect to its neighbors. It considers as outliers the samples that have a substantially lower density than their neighbors.

The LOF of a point tells the density of this point compared to the density of its neighbors. If the density of a point is much smaller than the densities of its neighbors, then the point is far from dense areas and, hence, considered as an outlier.

We found out a number of outliers equal to 9.941.

The results are shown in Figure 17.

3.5 ABOD - Angle Based

Another class of outliers detection techniques is the *angle-based*, whose idea is to examine the spectrum of pairwise angles between points, detecting outliers as the ones with a highly fluctuating spectrum.

The **ABOD** (*Angle-baset Outlier Degree*) algorithm was therefore implemented.

The selected number of neighbors was 32, the contamination value, the proportion of outliers to be detected in the data set, was estimated equal to 0,1 thanks to the previously observed results. The implemented method was the fast one, which only considers k-neighbors of the training points to reduce the computational cost.

The results are shown in Figure 18.

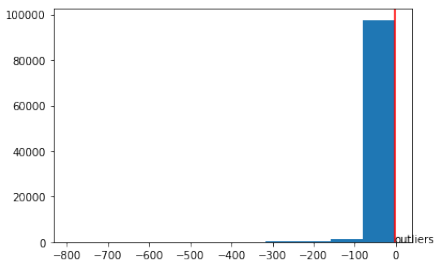

Figure 17: LOF outliers.

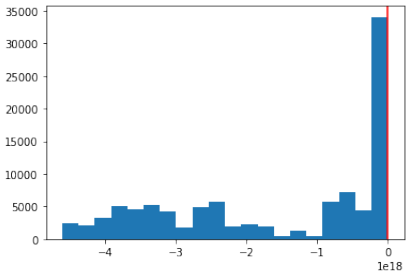

Figure 18: ABOD outliers.

3.6 AUTOENCODER

The last implemented method was the **Autoencoder**, which represents each record through the compression and decompression of a *Deep Neural Network*.

Such technique is based on detecting outliers by observing the reconstruction error of the *DNN* and labelling as outliers the points that generate a large value for such error. The DNN used for the analysis was made of 4 hidden layers composed as follows [4, 2, 2, 4] and after 50 epochs, 9.941 outliers were detected, whose distribution is shown in Figure 19.

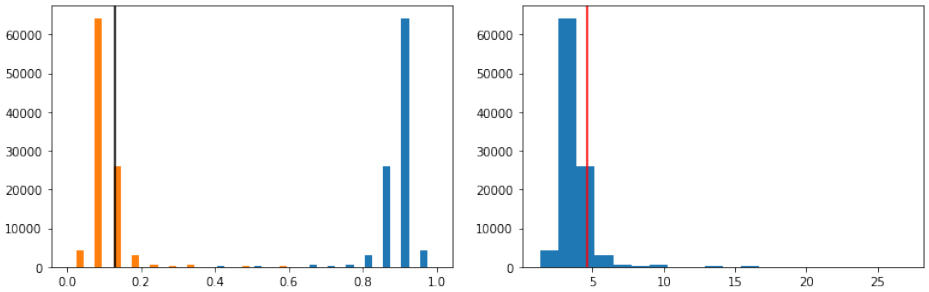

Figure 19: Autoencoder Probability *(left)* and Anomaly Score *(right)*.

3.7 PCA Reduction - Outliers Detection

These techniques helped to identify some possible anomalies in the records.

In Figure 20 and 21 are shown the plots using the PCA main component to represent the outliers identified by the various explained techniques.

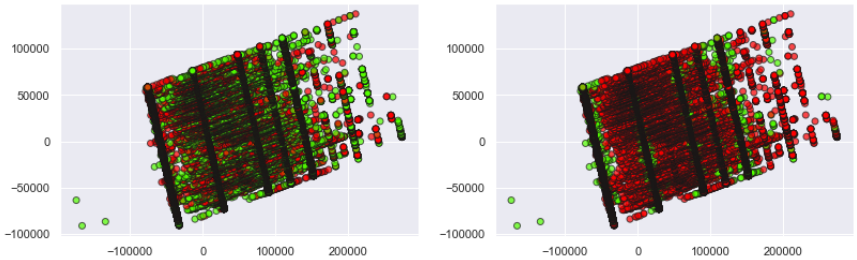

Figure 20: DBSCAN *(left)* and LOF *(right)*.

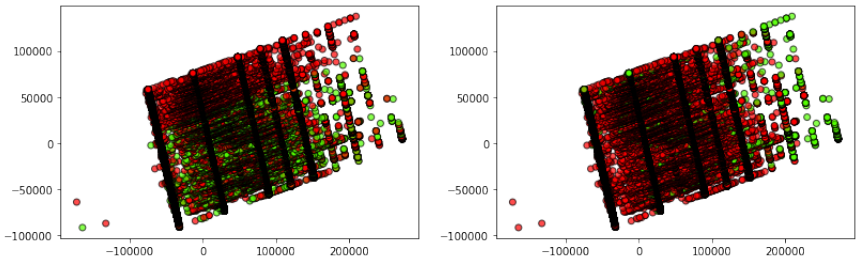

Figure 21: KNN *(left)* and AUTOENCODER *(right)*.

It was decided to treat the detected outliers by using the quantile-based flooring (the 10th percentile) for the lowest values and capping the 90th percentile for the higher values. After removing the outliers, we get a much better skewness value.

4 Imbalanced Learning

4.1 Imbalance Transformation

Another very interesting topic is to learn how to deal with an imbalanced data set, but since the one analyzed in this report was not unbalanced enough to consider it a problem, to be precise we had a 82,21% of the data set characterized by ”0” value and 17,79% by ”1” value, some transformations need to be applied first.

One feasible solution in order to transform the available data set into an unbalanced set was removing all the records that could be considered easily classifiable.

We decided to create imbalance removing the rows characterized by the release of the albums previously than the first semester 2009. With the same logic, we removed all the rows with TrackBitRate values lower or equal than 195.132,5 and with TrackId values lower than 112.324,0.

The result was a data set of 15.190 records with the following distribution for our target variable, whose with a license ’Attribution-Noncommercial-Share Alike 3.0 United States’, shown in Table 2:

Value	%	Dataset	Training	Test
0	93.93	14.267	9.987	4.280
1	6.07	923	646	277

Table 2: Imbalanced Learning - Data Set composition.

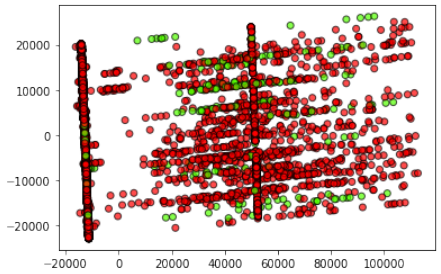

Figure 22: Imbalanced learning.

4.2 Oversampling

The first technique was the **Oversampling** one.

The methods used were Random Oversampling, *SMOTE*, *ADASYN* and *SVMSMOTE*. Plotting the first and second eigenvectors of the PCA can help appreciating how different techniques affected the distribution of the data set’s principal component; such plot are shown in Figure 23.

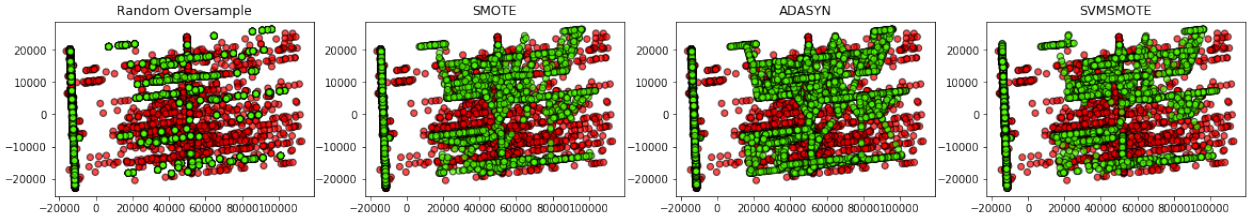

Figure 23: Oversampling Scatter Plots.

These methods were evaluated by repeating the classification with *Decision Tree* and *Random Forest classifiers*. Surprisingly, Random Oversampling was the technique that performed better, leading to a 99.3% accuracy for the Decision Tree and 99.9% for Random Forest; the obtained ROC curve is presented in Figure 24.

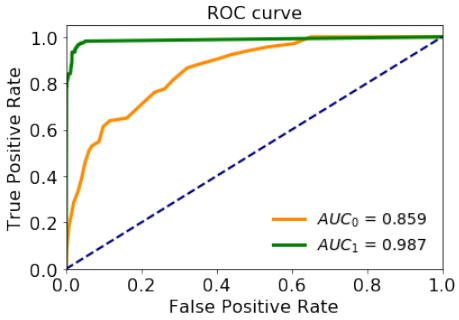

Figure 24: Random Oversampling - ROC Curve.

SVMSMOTE, SMOTE and ADASYN also gave good results with Random Forest, reaching respectively an accuracy of 99.1%, 99.1% and 99.8%, while none of them performed that good with the Decision Tree algorithm.

4.3 Undersampling

The following implemented method were Random **Undersampling**, in particular *TomekLinks*, *Condensed Nearest Neighbor* and *Near Miss*. As for the Oversampling approaches, they were evaluated by repeating classification with *Decision Tree* and *Random Forest classifiers*.

TomekLinks was the one that performed better, reaching a value of Accuracy equal to 99.8% with the Random Forest.

CNN (Condensed Nearest Neighbor) also gave good results: 99.6% of Accuracy with the Random Forest. From the results obtained it could be asserted that it was still possible to classify records using an advanced classifier like the Random Forest, but the Decision Tree proved to be inefficient in predicting the target variable. In fact, with the *Decision Tree classifier* none of the implemented methods allowed to reach an accuracy value higher than the 96%.

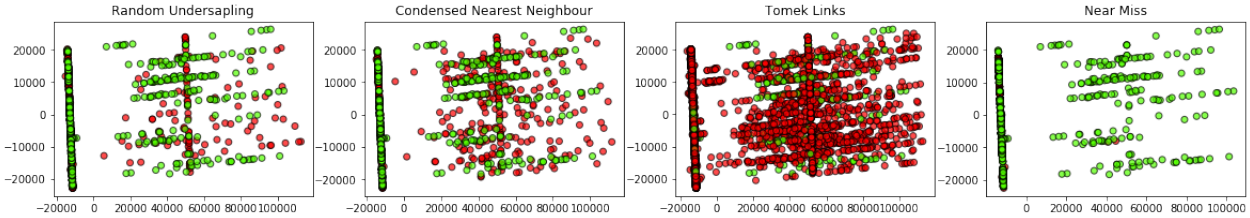

Figure 25: Undersamplings Scatter Plots.

The last explored approach was to evaluate the performances of a Random Forest on the original data set after adjusting the Class Weight, giving 97% of weight to the class 0 and 3% to the class 1. The results of Cross-Validation are reported in Table 3.

	F1	AUC	AVG PRECISION	ACCURACY	PRECISION	RECALL
Random-CV	0.68	0.69	0.56	0.97	0.90	0.55
Stratified-CV	0.68	0.72	0.60	0.97	0.93	0.55

Table 3: Random Forest with Class Weights Cross-Validation.

Figure 26 also shows the decision maps of an **AdaBoost** algorithm applied with a decision stump as a base estimator. The two plotted scatters are from Random Oversampling and Undersampling.

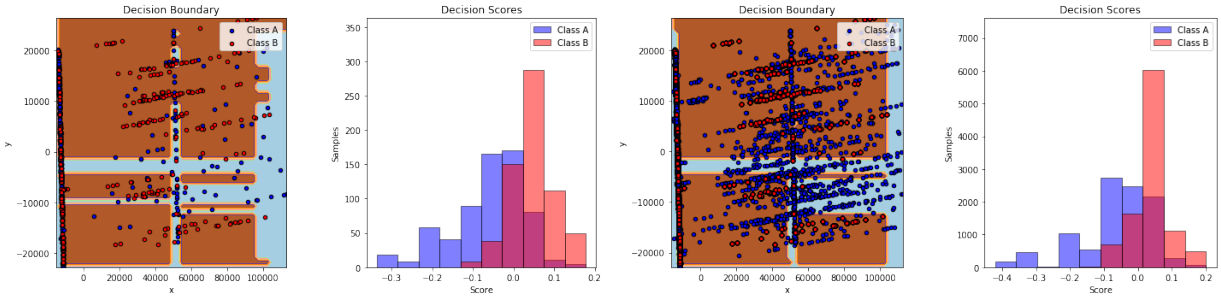

Figure 26: AdaBoost decision maps - Random Undersampling *(left)* and Random Oversampling *(right)*.

5 Regression

In this section a **Regression** problem will be addressed, based on two different attributes of the data set: TrackBitRate and TrackId.

5.1 Linear Regression and Huber Regressor

The **Linear Regression** approach was applied and evaluated through the typical measures: *R2 Score*, *Mean Squared Error* (MSE) and *Mean Absolute Error* (MAE), whose values are shown in Table 4. In particular, the MSE measures the variance of the residuals and the MAE coefficient measures the average of the residuals in the dataset.

The objective of Linear Regression is to find a line that minimizes the prediction error of all the data points. [Figure 27]

Huber regression was also implemented, which should be an approach similar to the Linear one but more robust to outliers.

	Linear	**Huber**
R2	0,070	0,038
MSE	1.857.396.272,385	1.921.844.962,504
MAE	37.453,057	37.470,562

Table 4: Regression Evaluation.

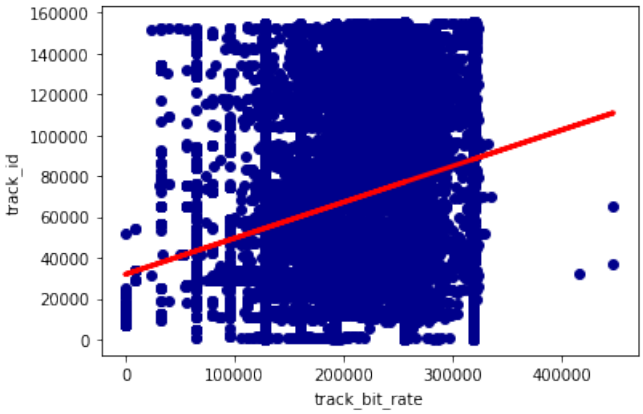

Figure 27: Linear Regression.

5.2 Multiple Linear Regression

Since the *runner-up* between the tested 2D problems was the one composed by TrackBitRate and TrackId, it seemed interesting to include in the aforementioned regression problem the **TrackLicense** variable which represents the license of the tracks.

The so defined problem was used to perform various multiple regression methods such as **Linear**, **Ridge**, **Lasso**, **Huber**, **Theil** and **Bayesian Ridge**.

Apart from the Theil and Huber approaches, which proved to be inadapt for the data, the others performed very similarly, which is why it was retained sufficient to present the results only for the Multiple Linear Regression.

- R2 = 0,096
- MSE = 6.343.930.161,58
- MAE = 78.438,88

Figure 28 shows a 3D representation of the variables.

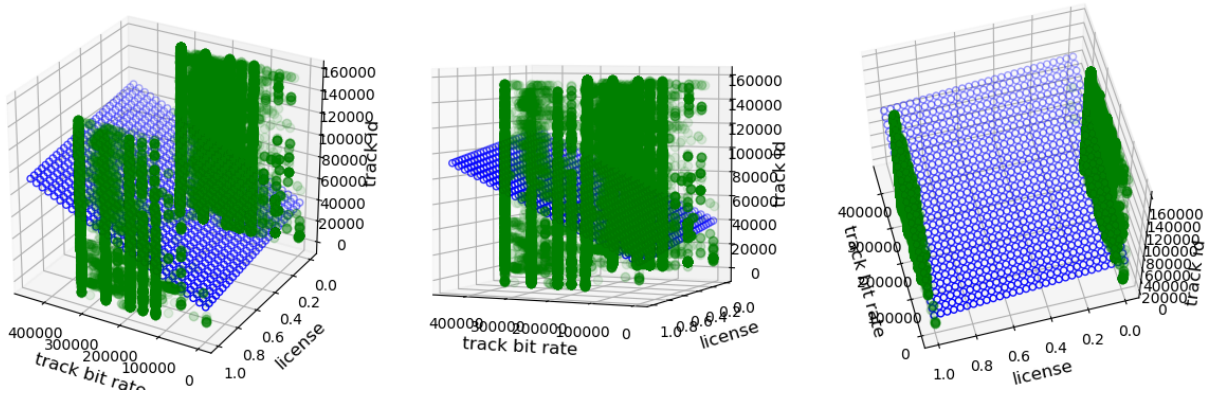

Figure 28: Multiple Linear Regression.

6 Advanced Classification Methods

6.1 Naïve Bayes

A **Naive Bayes classifier** is a probabilistic machine learning model. The crux of the classifier is based on the Bayes theorem.

The best results were obtained with a *Categorical* approach.

6.1.1 Gaussian Naïve Bayes

The **Gaussian Naive Bayes** algorithm is a special type of *NB* algorithm. It is specifically used when the features have continuous values. It is also assumed that all the features are following a gaussian distribution, for instance a normal distribution.

We tried to test the quality of our data set based on this algorithm and we obtained an Accuracy value of 71.67%.

The results are shown in Figure 29.

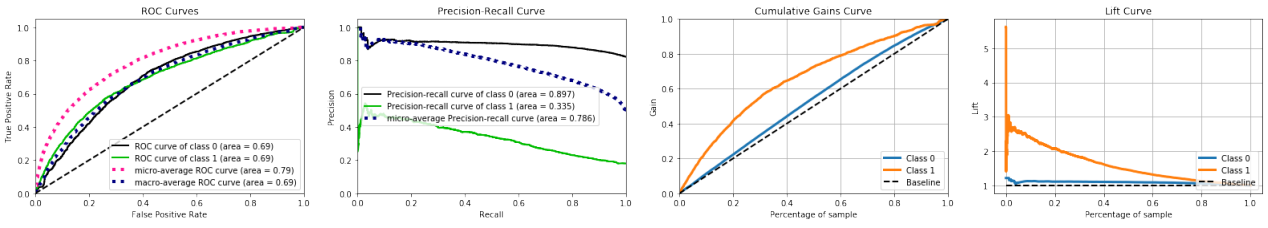

Figure 29: Gaussian Naïve Bayes Curves.

6.1.2 Categorical Naïve Bayes

On the other hand, the **Categorical Naive Bayes** classifier is suitable for classification with discrete features that are categorically distributed. The categories of each feature are drawn from a categorical distribution.

The same of previously, we tested the quality of our data set based on this algorithm and we obtained a better value of Accuracy than before, to be precise 95.16%.

The results are shown in Figure 30.

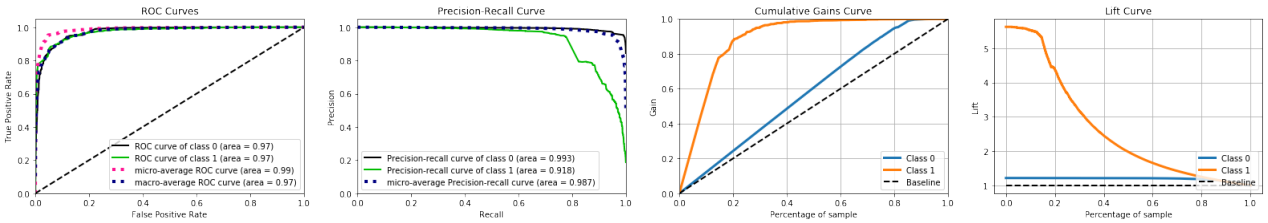

Figure 30: Categorical Naïve Bayes Curves.

6.1.3 Bernoulli Naïve Bayes

Bernoulli Naïve Bayes classifier is suitable for discrete data. It is similar to the previous type but the predictors are boolean variables.

6.2 Support Vector Machines

Support Vector Machines (SVM) are a supervised learning model that represents the decision boundary using a subset of the training examples, known as the *support vectors*. After scaling opportunely the data set, linear **SVM** was implemented, which is a classifier that searches for the hyperplane with the largest margin.

The results were almost immediately satisfying, which made not very meaningful the following parameter tuning. In fact, the worst result was obtained by using the Squared Hinge Loss function and C, the parameter representing the penalty of misclassifying training instances, equal to 0.01; such configuration gave a value of Accuracy equal to 82.47%. The best outcome was obtained with the Hinge Loss Function and C equal to 100, which performed with a 82.48% of Accuracy.

Support Vector Machines are fantastic because they are very resilient to overfitting. SVM are naturally resistant to overfitting because any interior points are not going to affect the boundary.

6.3 Neural Networks

In this paragraph we will present the task of building an **Artificial Neural Network**, for which a 30% of the Training set was used as Validation set. In fact, it was fundamental to see after every epoch how well the Neural Network constructed so far fits the model, comparing the loss curves obtained with the training and validation sets, and then select the best configuration based on the optimal validation error.

All the continuous attributes were normalized in order to avoid scalability problems that could have affected the efficient working of the Neural Networks.

The approach was to initially compare the *Loss Curves* provided by different configuration of **MLPC** (Multiple Linear Perceptron Classifier) before moving to the verification of the built models.

The obtained MLPC Loss Curves are shown in Figure 31, from which it is possible to observe that the *Adam* adaptive learning rate optimization algorithm was the most compliant with our research for the lowest loss.

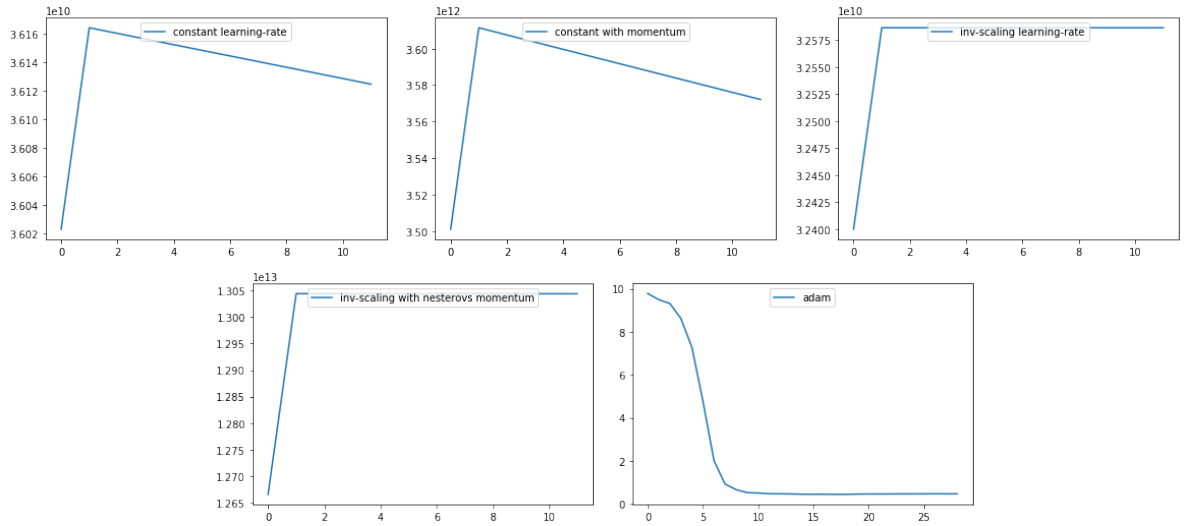

Figure 31: Loss curves for MLPC.

	Training set score	Training set loss
Constant learning rate	0.788	0.7495
Constant with momentum	0.788	24.2312
Inv-scaling learning-rate	0.788	0.7547
Inv-scaling with momentum	0.788	24.2442
Adam	0.989	0.493

Table 5: Models comparison.

Such algorithm is a stochastic gradient based optimizer that uses the squared gradients to scale the learning rate and than takes advantage of momentum by using moving average of the gradient instead of the gradient itself. It was therefore used to build the first NN, by implementing the MLPC with a single hidden layer composed of 3 nodes, a logistic activation function and an initial learning rate of 0,2.

The NN just described performed with a 82.21% of Accuracy, giving a Loss of 0.420 on the Training and of 0.422 on the Validation set.

It was then explored the possibility of building a more complex model: a **Deep Neural Network**.

A sort of randomized search between the parameters was made and each combination was evaluated on the validation set, basing such evaluation on Loss and Accuracy. The goal was to select the number of hidden layers and nodes in such a way that the loss curves of the validation and training sets resulted collinear. After numerous attempts, very good results were obtained with a DNN with three hidden layers of sizes 50, 100 and 50, whose composition is represented graphically in Figure 32 to better understand how the information was processed and classified. The nodes, in green in the Figure 32, represent the attributes, while there are fifty hidden layers with different sizes which bring the process to the output node for *Track License*.

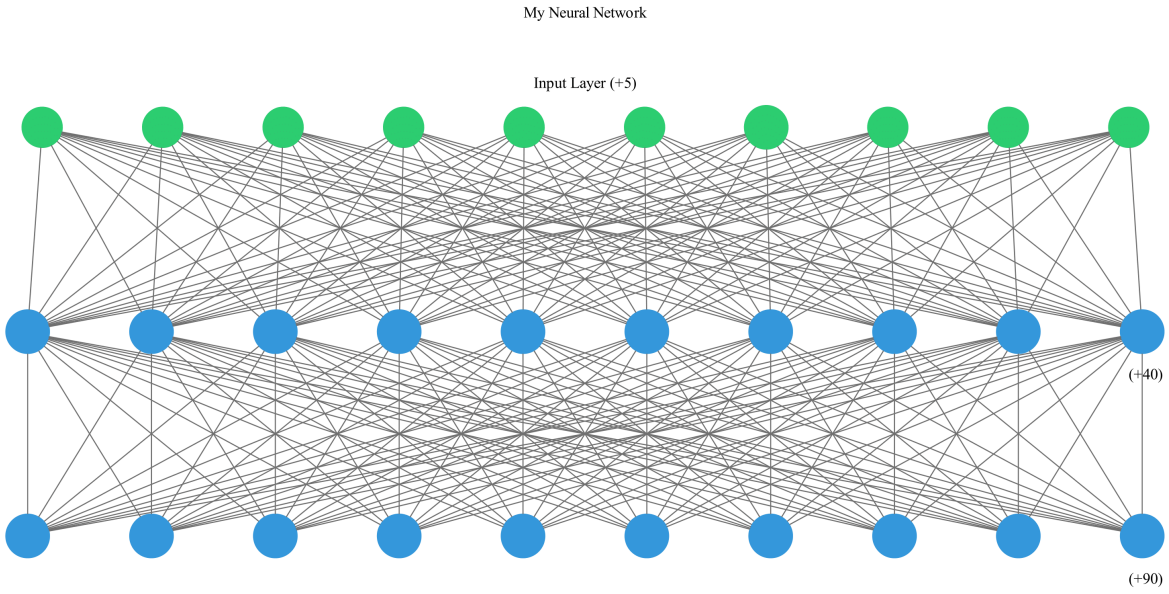

Figure 32: DNN Structure.

The selected activation function for the hidden layers was the rectified linear unit, while the one for the output node was the sigmoid. The applied learning mode was the *Minibatch*, updating the weights every 10 records. The performances of such model, evaluated through Cross-Validation, were the following:

- Accuracy = 98.0%
- F1 Score = 94.26%

It was also possible to read the history of the training process, whose last three rows are reported in Table 6.

Early Stopping, *L2 Regularization* and *Dropout* were also applied, but since the described model did not seem to present an overfitting problem, they were tested on the same DNN but with a larger number of epochs, to be precise 1000 epochs. None of the 3 methods lead to higher accuracy and they were therefore not needed. In fact, in Figure 33 is possible to appreciate the loss both on the Validation and on the Training set in a more accurate way.

	Loss	Accuracy	Val Loss	Val Accuracy
Epoch 98/100	0.0228	0.9931	0.1900	0.9761
Epoch 99/100	0.202	0.9939	0.1980	0.9759
Epoch 100/100	0.210	0.9937	0.1845	0.9774

Table 6: Last 3 epochs of the training process.

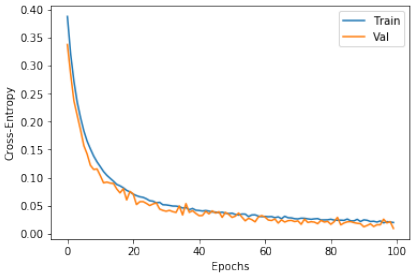

Figure 33: Model fitted on 100 epochs.

6.4 Ensemble Methods

The methods described in this paragraph are based on the so-called *Wisdom of the Crowds*, which says that collective knowledge of a diverse and independent group of people typically exceeds the knowledge of any single individual.

Translating this idea to the classification task, it can be supposed that aggregating a multitude of weak classifiers forms a strong classifier.

Random Forest gave a similar result to the one of the *Decision Trees* previously analyzed: TrackId was still on of the main actor in the classification while the other features were less influencing, as shown in Figure 34.

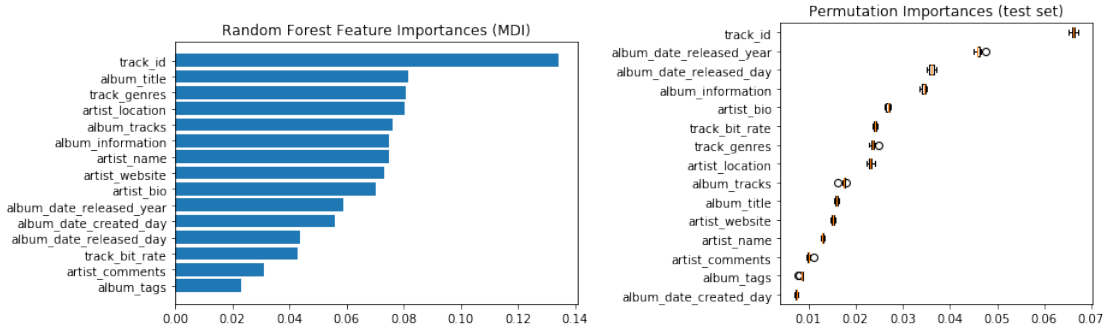

Figure 34: Feature Importance - Ensemble Methods.

The Random Forest was able to reach a 99.1% value of Accuracy and the Cross-Validation score was equal to 98.81%.

Boosting and *Bagging* were also applied using the Random Forest as base classifier and the results about the Accuracy value were respectively 99.09% and 98.86%.

A stacking technique was also tried, using the Logistic Regression and then the Multilinear Perceptron as final estimators. All this experiments were evaluated using Cross-Validation, where they all provided a value equal to 99% of Accuracy. F1 Score was near to 99% in all these cases for the training set, the best result was 99.44%, and around the 97% for the test one, the best value was 97.41%.

In Figure 35 is shown the relation between different measures used to evaluate **AdaBoost** and the number of estimators used. It can be observed that the algorithm SAMME.R, discrete AdaBoost that take into account the probability estimation, converged faster.

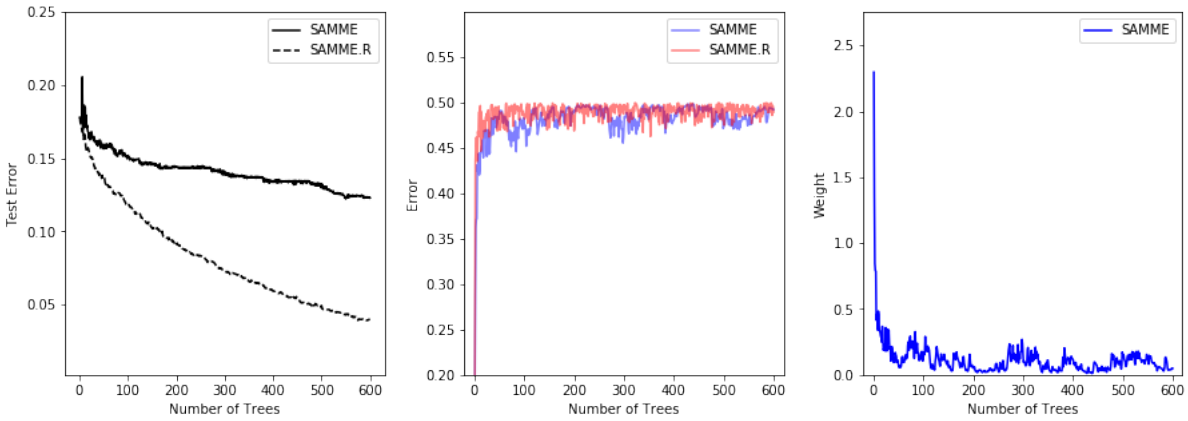

Figure 35: Evaluation of the AdaBoosted Decision Trees.

6.5 Logistic Regression

Since the target variable TrackLicense was binary, the **Logistic Regression** appeared to be an efficient method to fit a curve to the data.

For this task three different sets of attributes were defined: one including the attributes, one using the main component extracted after the application of the *PCA* algorithm and one composed of only the variable TrackId, which seemed to be very good at classifying in the previous paragraphs.

Other classifiers such as *SGD* classifier, *Perceptron*, *Passive Aggressive* classifier and *Ridge* were also tested and compared both in terms of Accuracy, reported in Table 7, and graphically, shown in Figure 36.

	Attributes	**Main Component**	**Track Id**
Logistic	82.56	61.38	69.95
Perceptron	25.41	58.90	82.47
Ridge	82.44	61.64	58.81
Passive Aggressive	82.47	59.01	21.20
SGDC	82.19	59.10	82.31

Table 7: Accuracy comparison.

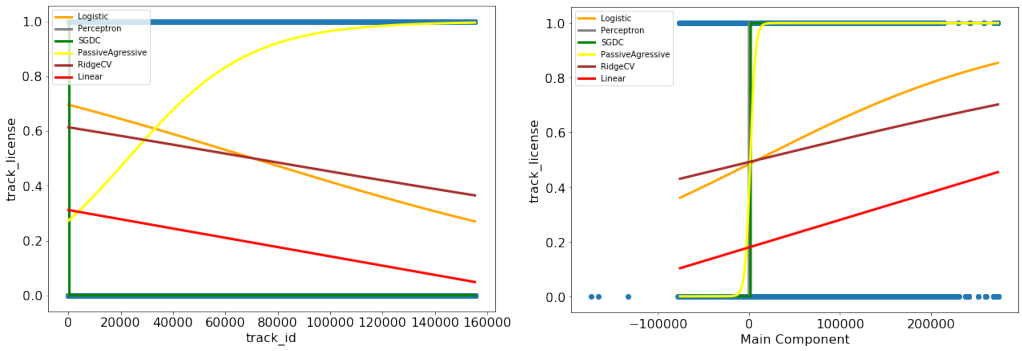

Figure 36: Graphical comparison.

SGDC minimizes the gradient and find a clear division in the records, separating dichtomously its output, just as the **Perceptron** model, and was therefore considered appropriate for analyzing the data in question.

In general, all the classifiers were able to predict very accurately when using all the attributes, accuracy higher than 82%, except the Perceptron.

Overall, the Perceptron model was the one that performed worst, but of all of the explored methods the only one that was able to perform as good as the Logistic Regression was SGDC.

Thus the Logistic Regression proved its goodness in this type of problems and its performances' evaluations are shown in Figure 37.

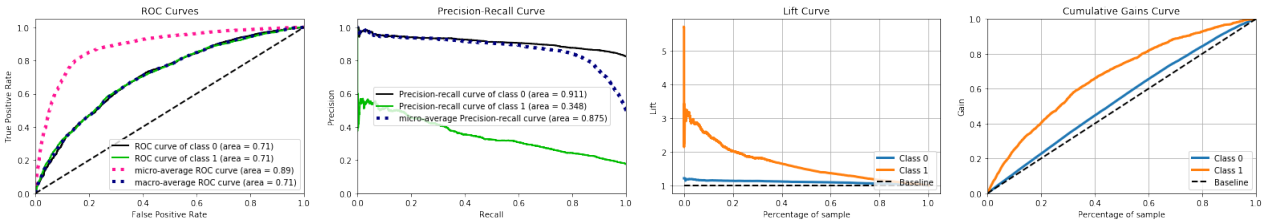

Figure 37: Evaluation graphs for Logistic Regression.

7 Time Series

Given the presence of many temporal feature in our data set, it was possible to analyze the data as a **Time Series** and such analysis will be presented in this section. We chose one of the most characterized temporal attribute, *AlbumDateReleased*.

We managed this attributes in order to transform it in a *datatime* data type; in particular, we were interested in its division in year, month and day.

In order to inspect the data examined so far as a Time Series, they were manipulated in order to highlight their temporal dimension, using the attribute AlbumDateReleased as the new index of the data set.

After some exploration, the selected attribute to be extracted was *AlbumListens* because of its good temporal trend with our temporal attribute.

The data set were structured so that they would be composed by records from contiguous days. At this point various Time Series could be extracted using different criteria, using three distinct time intervals, determining three new time series data sets:

- Day: the extracted time series were composed of a daily span.
- Month: the extracted time series were composed of a monthly span.
- Year: the extracted time series were composed of a yearly span.

The Time Series shown in Figure 38 are characterized by a temporal trend of 10 years, which we considered the most significant, from the 01-01-2006 up to 31-12-2016, divided respectively from left to right in a TS frequency of Day, Month and Year.

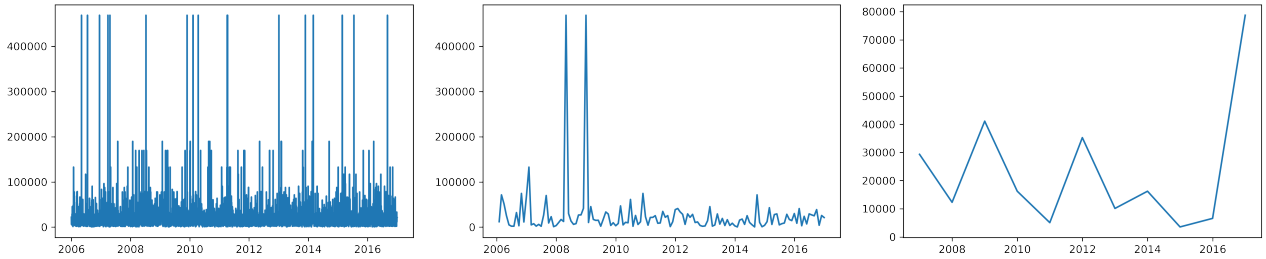

Figure 38: Time Series data sets focusing on a 10 years trend.

7.1 Forecasting

To provide a **Forecasting** analysis, the data from the data set used for the daily Time Series was used since it was the largest set available. Such records had been collected between the 01-01-2006 and 31-12-2016.

7.1.1 Time Series components

Before proceeding with forecasting, data needed to be normalized and therefore *Logarithm* and *Rolling Mean* transformations were applied. This process was necessary to exclude some components that could prevent us from making an accurate prediction, like trend and seasonality. Such components were exposed thanks to a deseasonality process, that permitted to identify the elements that composed the Time Series, which are plotted separately in Figure 39.

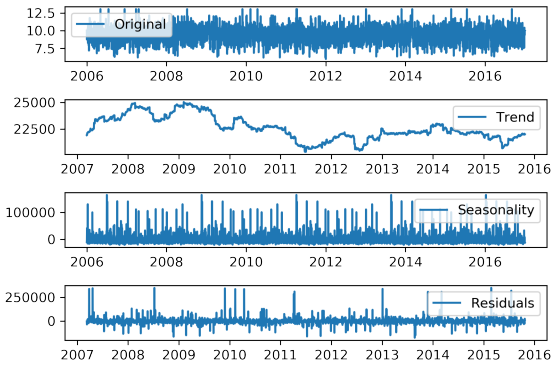

Figure 39: Time Series components.

7.1.2 Exponential smoothing

We tried to deal with forecasting and in particular with **Exponential Smoothing**. It is a Time Series forecasting method for univariate data that can be extended to support data with a systematic trend or seasonal component.

The results are shown in Figure 40.

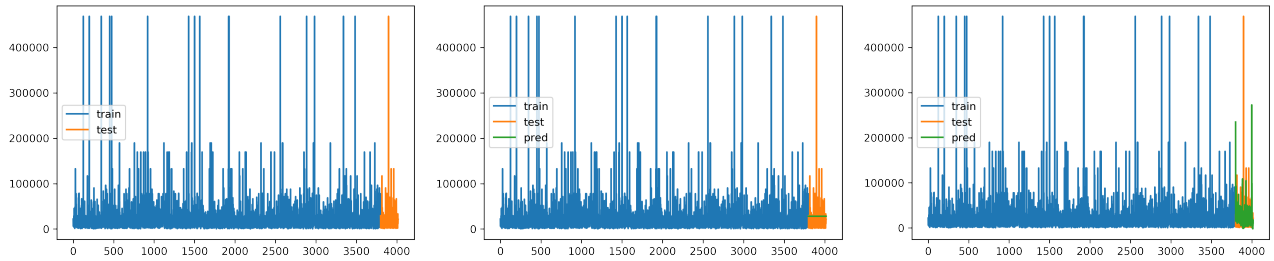

Figure 40: Time Series Forecast.

7.2 Motifs

Once obtained the complete Time Series, an interesting task was to discover **Motifs**, which are repeated sub sequences in the analysed TS.

For the purpose of finding such sub sequences it was required to use a different values of window on the Matrix Profile. We made some tests with the Time Series with a daily split and with a monthly split as well.

About the monthly one, we tried to see the difference changing the window value; we used 40, 50 and 60, from left to right respectively, shown in Figure 41.

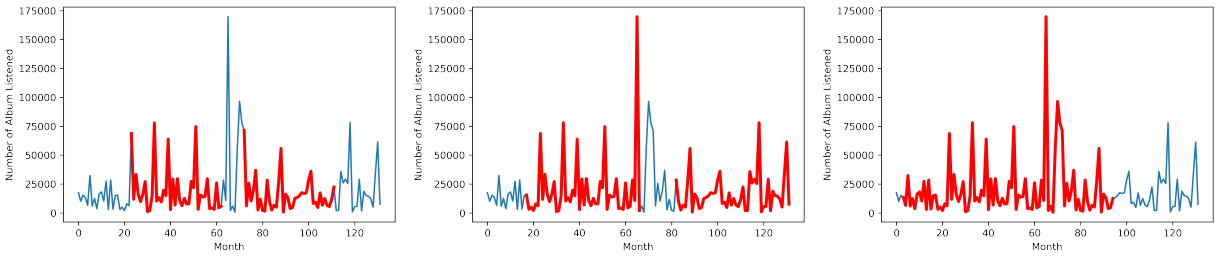

Figure 41: Time Series Motifs - Monthly TS.

On the other hand, in the case of the daily TS, we adopted window values of 100, 160 and 220, as shown in Figure 42 respectively from left to right.

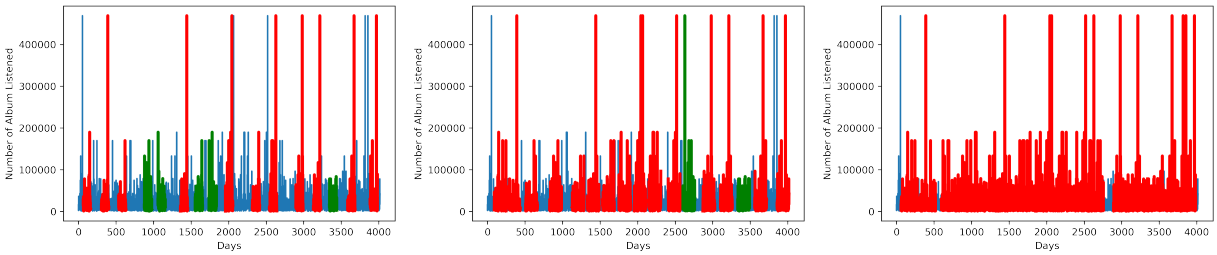

Figure 42: Time Series Motifs - Daily TS.

The window value generally refers to a number of samples taken from total Time Series in order and represents a particular period of time.

It was interesting to see these differences in the plots by changing and increasing the window value number for both the time series.

7.3 Anomalies

An **Anomaly** is an outlier data point, which does not follow the collective common pattern of the majority of the data points and hence can be easily separated or distinguished from the rest of the data.

We tried to deal with the daily Time Series using different values of the window, in particular 250, 460 and 600; the results are shown in Figure 43 from left to right.

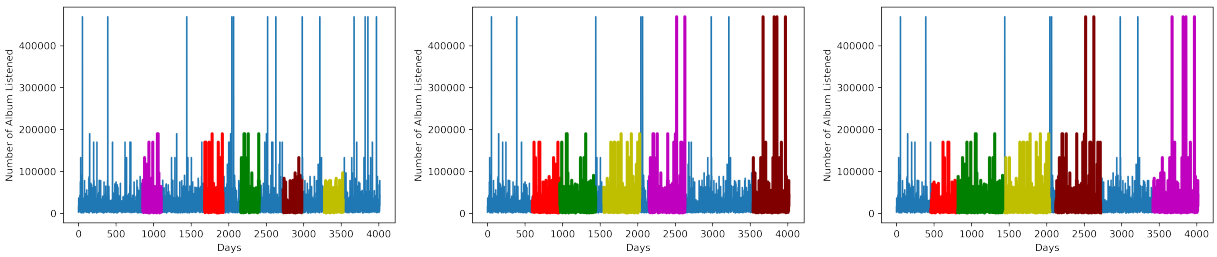

Figure 43: Time Series Anomalies - Daily TS.

7.4 Clustering

The basic principle of **K-Means** involves determining the distances between each data point and grouping them into meaningful clusters.

The Time Series used for the clustering method was different than before. We made a study based of a weekly Time Series. For all the analyses shown in the following points, we used a number of cluster k equal to 3.

7.4.1 K-Means - Euclidean distance

First of all, we tried to analyze the behaviour of the **K-Means** clustering with the **Euclidean** distance.

Figure 44 shows the result using this typology of metric. One issue with this metric is that it is not invariant to time shifts, while the data set at stake clearly holds such invariants.

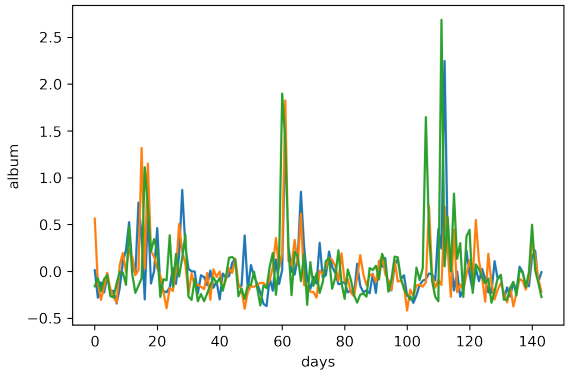

Figure 44: Time Series Cluster - Euclidean distance.

7.4.2 K-Means - DTW distance

To overcome the previously illustrated issue, we applied a distance metric dedicated to Time Series, the **Dynamic Time Warping** (DTW) and the **Soft DTW** as well.

Dynamic Time Warping is a method of calculating distance that is more accurate than Euclidean metric. It has an advantage over Euclidean if datapoints are shifted between each other. It calculates the smallest distance between all points.

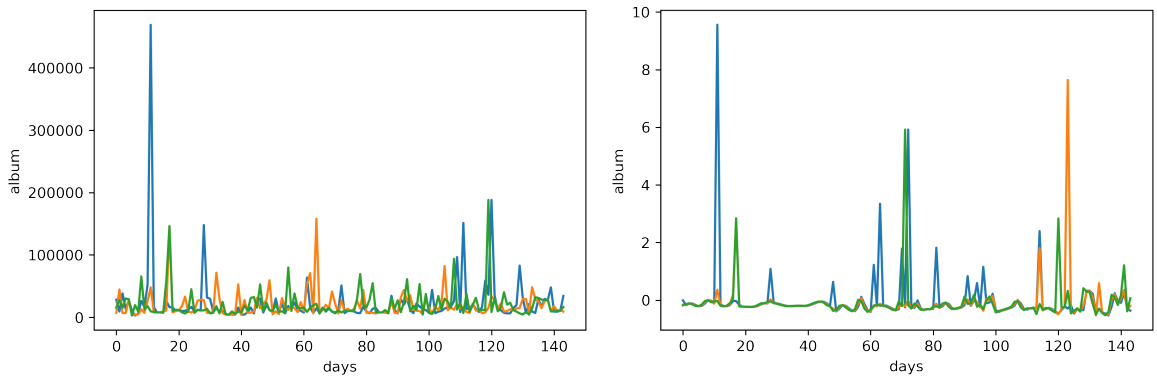

Figure 45: Time Series Cluster - DTW *(left)* and Soft DTW distance *(right)*.

7.5 Classification

For this task the daily TS shown in the previous paragraphs were used. We tried to focus on exploiting the information based on a Daily set: in particular we tried to predict whether a Time Series had been extracted from morning, afternoon or evening. The utilized Time Series was extracted from the feature AlbumListens. We changed the temporal feature due to the fact that the AlbumDateReleased has always the same timing in the night [Figure 46].

For our considerations, we adopted the AlbumDateCreated variable. In Figure 47 is shown the plot the class identified with the approaches mentioned.

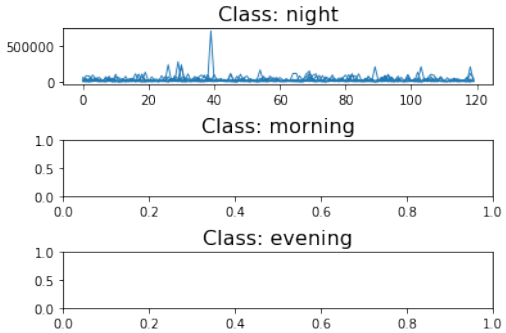

Figure 46: Class identification using AlbumDateReleased.

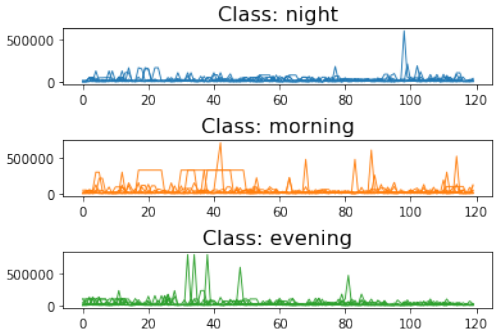

Figure 47: Class identification using AlbumDateCreated.

We made some considerations about the Shapelets extraction, which are subsequences that are extremely representative of class membership for each identified class. The most representative one is shown in Figure 48.

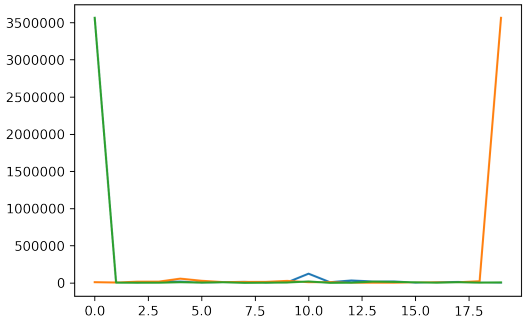

Figure 48: Shapelet Classification.

7.6 Sequential Pattern Mining

For this task, the data considered was the one of the Time Series obtained in the previous points, in particular the Weekly split TS. The idea was to exploit the Time Series and transform it adequately.

SAX was applied to reduce the size of the series, dividing it into 75 segments of equal length whose values were then replaced with the mean of the segment they belonged to; finally they were divided into 5 equi probable regions and renamed according to a specified alphabet. Once completed this process, the search for recurrent patterns started: the idea was to consider each day as a sequence of transactions and apply the Prefix Span algorithm to find them. Dividing the time series into 5 equal parts, the database of sequences of transactions was obtained, which are shown in Figure 49.

By applying Prefix Span, the search was made between closed frequent pattern with various values for support, which are presented in Table 8 together with the number of patterns with a specified minimum length for each of those supports.

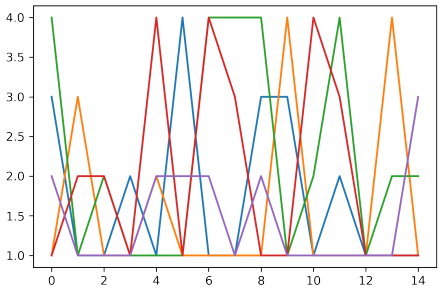

Figure 49: Daily Sequences.

Considering a minimum support equal to 3, the longest patterns found were three of length 8, five with length 7 and 8 with length 6, as better shown in Figure 50.

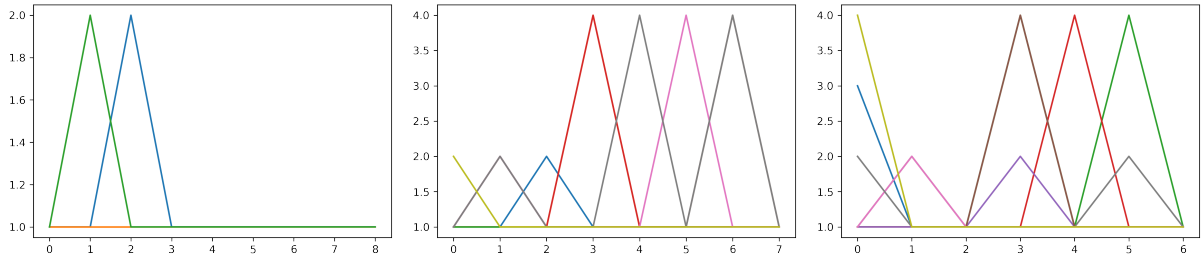

Figure 50: Longest pattern with length 8 *(left)*, 7 *(center)* and 6 *(right)*.

In all cases the represented patterns seemed to follow an increase in the values reaching a local max and than a decreasing zone.

MinSup MinLen	$\{>=1\}$	$\{>=2\}$	$\{>=3\}$	$\{>=4\}$
3	64	64	62	58
4	23	23	21	19
5	9	9	7	6

Table 8: Number of frequent patterns.

The three most interesting patterns were the ones with at least 5 elements and a minimum support of 3, which were the following: [3, 1, 1, 4, 1], [1, 1, 2, 4, 1] and [1, 2, 1, 2, 1].

8 Clustering

8.1 Advanced Clustering

8.1.1 Mean Shift

Mean Shift clustering aims to discover blobs in a smooth density of samples. It is a centroid-based algorithm, which works by updating candidates for centroids to be the mean of the points within a given region. These candidates are then filtered in a post-processing stage to eliminate near-duplicates to form the final set of centroids. Unlike the popular

K-Means cluster algorithm, Mean Shift does not require specifying the number of clusters in advance. The number of clusters is determined by the algorithm with respect to the data.

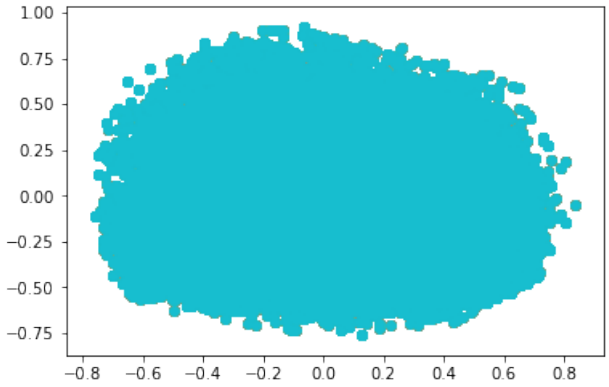

Figure 51: Mean Shift Plot.

8.1.2 OPTICS

OPTICS (Ordering Points To Identify the Clustering Structure), closely related to DBSCAN, finds core sample of high density and expands clusters from them. It adds two more terms to the concepts of DBSCAN clustering, *Core Distance* and *Reachability Distance*.

This clustering technique does not explicitly segment the data into clusters, but,instead, it produces a visualization of Reachability distances and uses this visualization to cluster the data, better shown in Figure 52.

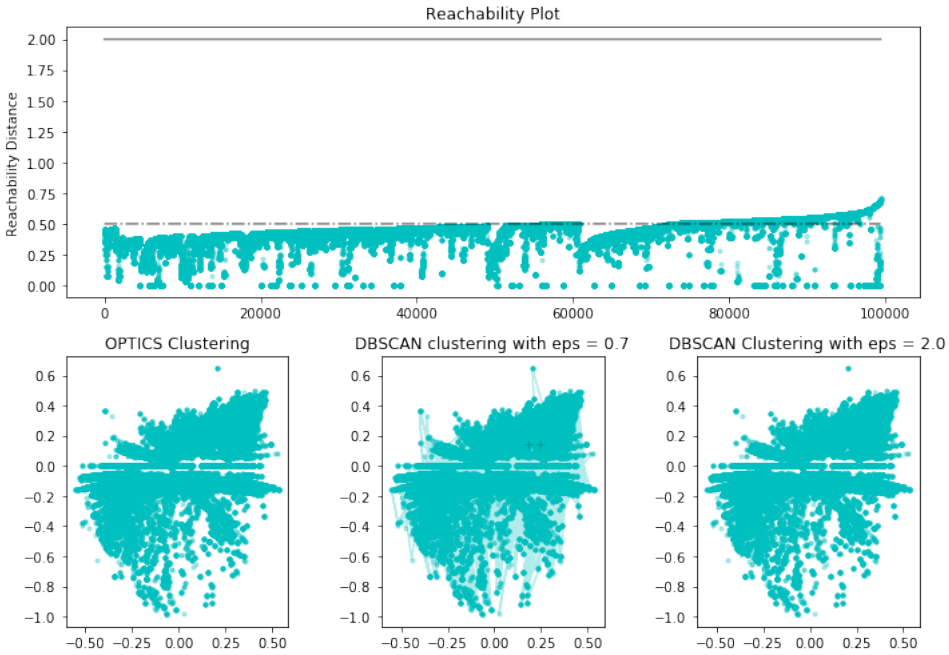

Figure 52: Reachability and Clustering Plots.

8.1.3 X-Means Clustering

X-Means clustering algorithm, an extended K-Means which tries to automatically determine the number of clusters based on **BIC** (Bayesian Information Criterion) scores.

Starting with only one cluster, this algorithm goes into action after each run of K-Means, making local decisions about which subset of the current centroids should split themselves in order to better fit the data. [Figure 53 and 54]

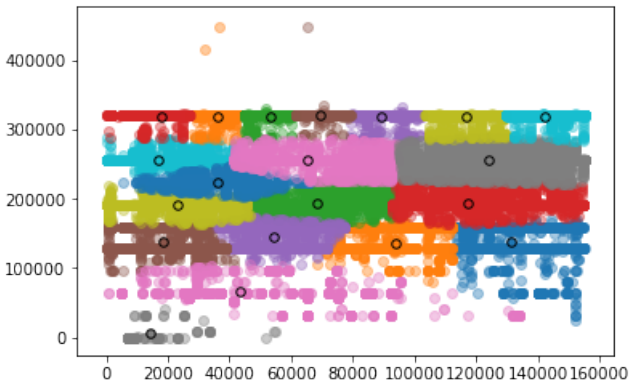

Figure 53: X-Means Plot.

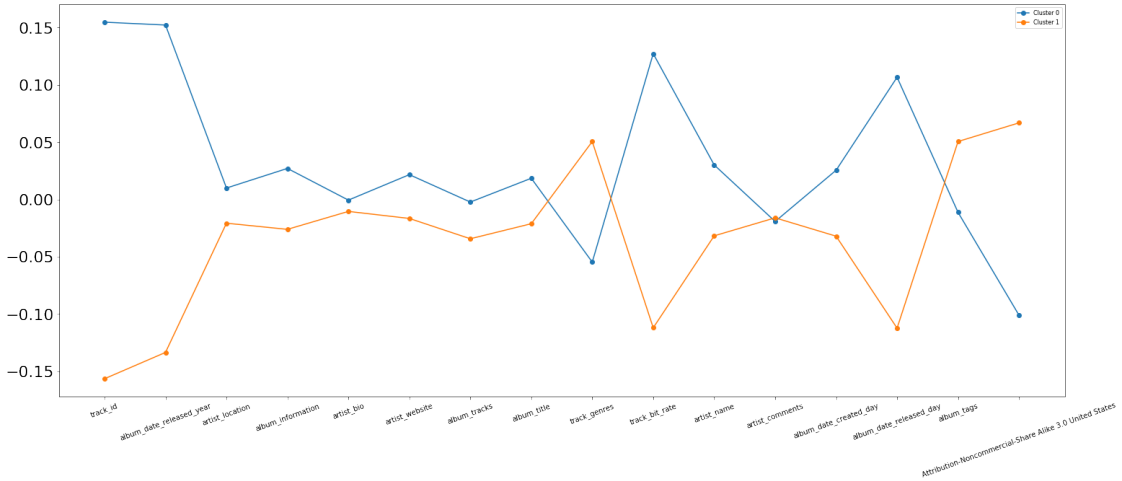

Figure 54: X-Means Plot - Focus on centroids.

We made some analysis about the **Silhouette Score**, better represented in Figure 55, and the **Sum of Squared Error** (SSE), which is 17.285.334.339.523,535.

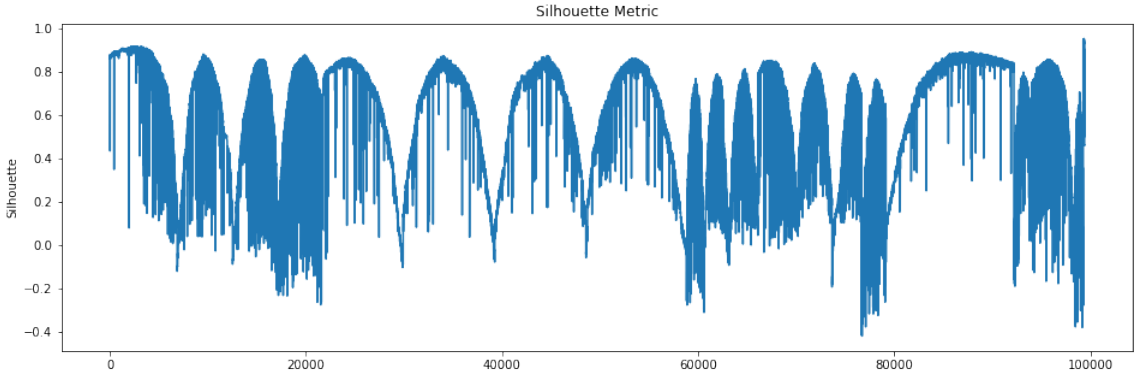

Figure 55: Silhouette Metric.

8.1.4 Bisecting K-Means Clustering

Bisecting K-Means is a kind of Hierarchical Clustering using a divisive, or top-down approach: all observations start in one cluster, and splits are performed recursively as one moves down the hierarchy.

Bisecting K-Means can often be much faster than regular K-Means, but it will generally produce a different clustering, as shown in figure 56.

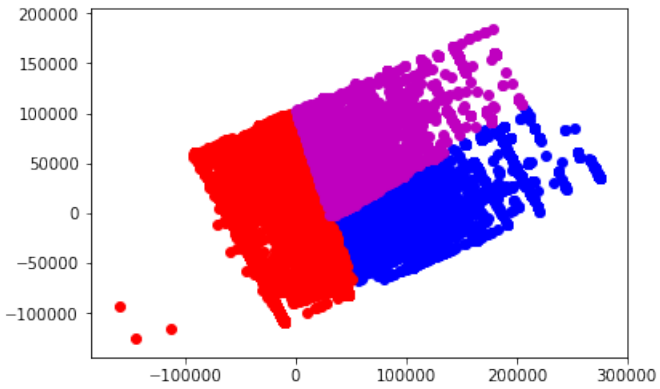

Figure 56: Bisecting K-Means Plot.

In this case, the value of the Sum of Squared Error (SSE) was equal to 230.528.515.121.318,72, and the Silhouette Score equal to 0.4207.

8.2 Transactional Clustering

8.2.1 K-Modes Clustering

The **K-Modes** approach modifies the standard K-Means process for clustering categorical data by replacing the Euclidean distance function with the simple matching dissimilarity measure, using modes to represent cluster centers, and updating modes with the most frequent categorical values in each of iterations of the clustering process. The dissimilarity metric used for K-Modes is the *Hamming distance* from information theory.

To summarize, K-Modes is used for clustering categorical variables. It defines clusters based on the number of matching categories between data points.

To apply this algorithm we first turned our dataset with continuous variables into a dataset with categorical variables by using *Binning* and we choose the number of clusters through the Elbow Method (Figure 57 on the right).

We iterated the K-Modes algorithm over our selected variables 50 times by varying the number of clusters every time. Our results are shown in Figure 57.

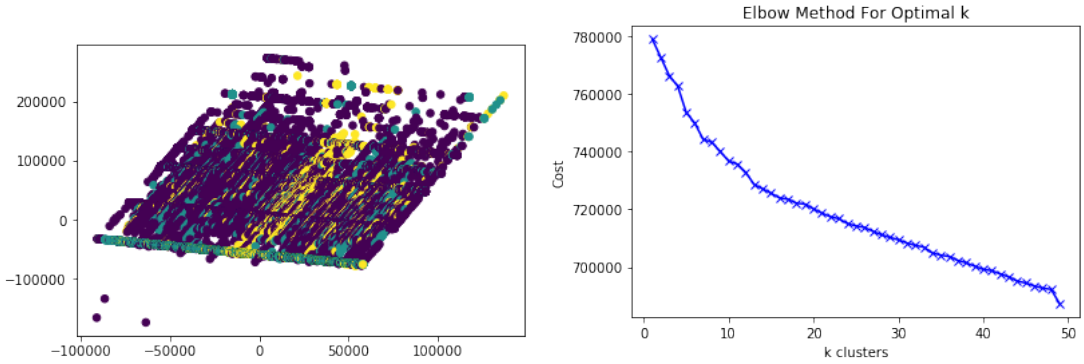

Figure 57: K-Modes Plot *(left)* and Elbow Method *(right)*.

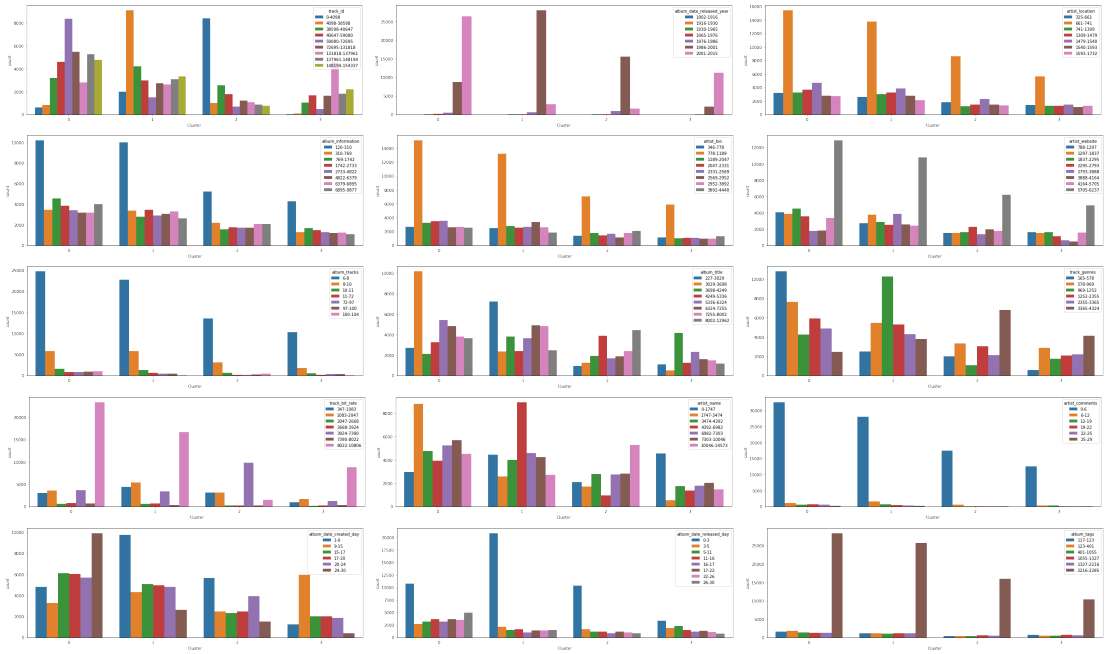

Figure 58: K-Modes Cluster Distribution.

8.2.2 ROCK

The **Rock** algorithm is a robust hierarchical algorithm for clustering transactional data. It uses links to cluster instead of the classical distance notion and the notion of neighborhood between pair of objects, that prohibited us from using binning algorithm, due to its dependence from continuous variables.

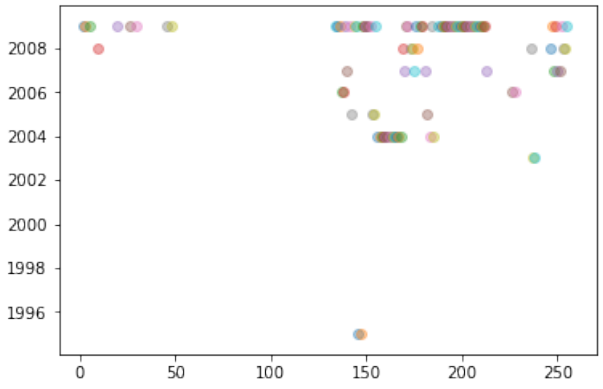

Figure 59: Rock Plot.

8.2.3 TX-Means

TX-Means is a parameter-free clustering algorithm able to efficiently partitioning transactional data in a completely automatic way. TX-Means is designed for the case where clustering must be applied on a massive number of different datasets, for instance when a large set of users need to be analyzed individually and each of them has generated a long history of transactions.

Normalized Mutual Info Score	Purity	Number of Custers
0.0855	0.4245	15

Table 9: TX-Means Evaluation.

9 Explainability

In order to conclude the classification task, **Explainability** was examined, trying to make the classification process more understandable and interpretable by humans. First of all, a black box model was trained, based on the Random Forest. For what concerns global explanation, partial dependance plots for the dataset's features are presented in Figure 60.

A random record was then selected in order to obtain some local explanation: **LIME** method was applied.

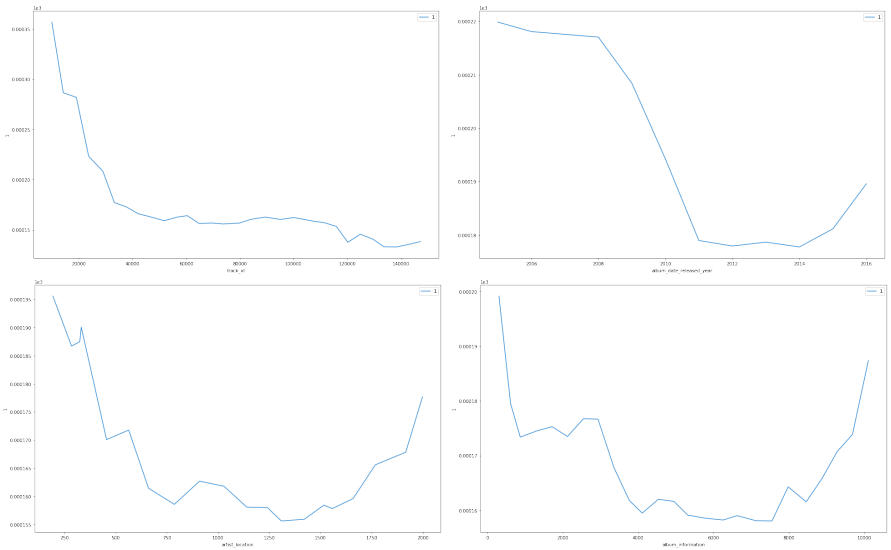

Figure 60: Partial Dependence Plots.

It confirmed the importance of the variable TrackId for classification: in Figure 61 are shown the feature importance obtained with **Lime Tabular Explainer**.

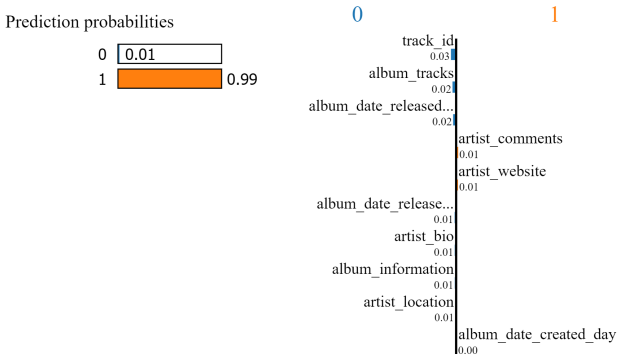

Figure 61: Feature Importance.