University of Pisa
Data Science and Business Informatics

DM2 Project

Data Mining: Advanced Topics and Applications

Authors
Matteo Garrò 620025
Petra Giada 626939
Elisa Pashku 628389

Academic Year 2020/2021

Summary

1 Module 1

1.1 Introduction

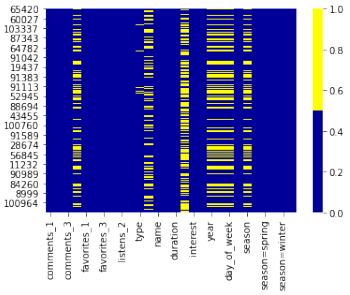
The dataset discussed in this project comes from the Free Music Archive (FMA), a free and easily accessible dataset for the music analysis. The FMA dataset derives from an original study and it is composed of 4 tables (tracks.csv, genres.csv, features.csv, echonest.csv), but only tracks dataset will be investigated, because it is the largest and more useful to our purposes. Tracks.csv encompasses metadata tracks such as ID, title, artist, genres, tags and so on. It counts 106,574 records and 53 attributes. First of all the dataset is split into Training Set and Test Set, respectively composed of 79930 rows (75%) and 26644 (25%).
Python 3.3 is used for analysis and model fitting. The IDE used is Jupyter.

1.2 Data understanding and Data preparation

This phase is carried out on the Training Set in order to make the prediction on the Test Set, after having applied to it the same data preparation process.
At the beginning, in order to reduce the number of columns, Training Set is filtered keeping only the significant attributes. Therefore, the remaining features are 3 *comments* (song, artist, album), *date_ released*, *engineer*, 3 *favorites* (album, artist, song), 2 *listens* (album, song), *producer*, *tracks*, *type*, *active_ year_ begin*, *active_ year_ end*, *location*, *name*, *bit_ rate*, *composer*, *duration*, *genre_ top*, *interest*, *language_ code*, *lyricist*, *number*, *publisher*.
To facilitate the analysis multiple features are renamed (*comments_ 1*, *comments_ 2*, *comments_ 3*, *favorites_ 1*, *favorites_ 2*, *favorites_ 3*, *listens_ 1*, *listens_ 2*), keeping the same order, and some columns are converted to a better type (float or datetime).
In preparation for deleting attributes with too many missing values, the graph demonstrates them and the table shows their presence as a percentage.

Table of Missing Value:

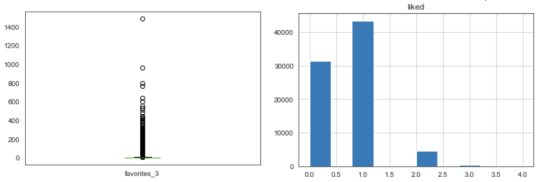
date_ released	34%	*location*	34%
engineer	86%	*composer*	97%
producer	83%	*genre_ top*	53%
type	6%	*language_ code*	86%
active_ year_ begin	79%	*lyricist*	100%
active_ year_ end	95%	*publisher*	99%



Attributes with more than 60% are eliminated. This threshold is due to the importance of *genre_ top*, which deserves a replacement attempt with the others. Before proceeding with the analysis, dataset needs some **variable transformations and creations**:

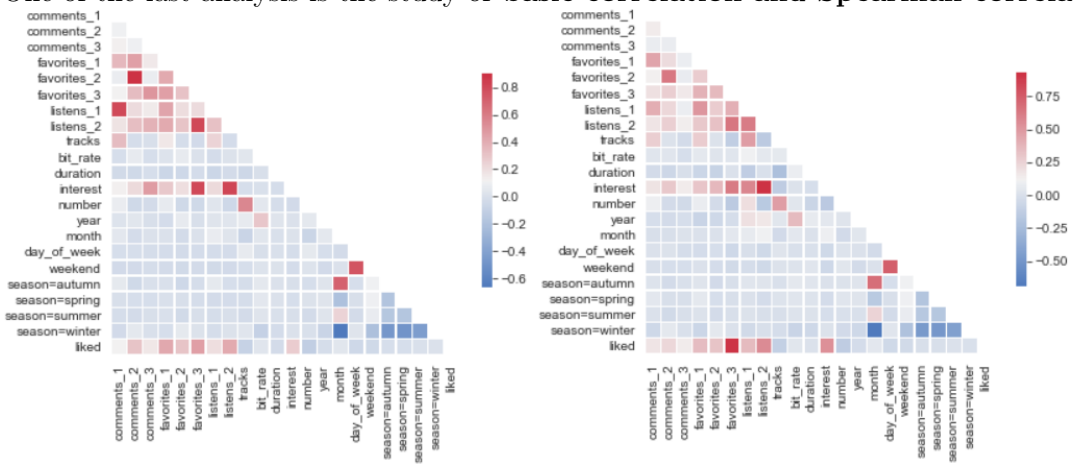
- *date_ released*: helpful to create *year*, *month* (in number), *season* (spring, summer, autumn or winter) and *day_ of_ week* (where 0 is monday).
- *day_ of_ week*: used to create the binary variable *weekend*.
- *season*: splitted into 4 one-hot encoding variables.
- *favorites_ 3*: prepared to be a target variable.
  Because of its distribution it is impossible to divide it with quartiles, that would have created a balanced target variable. So it is divided into intervals labelled with 0 (*favorites_ 3* < 1), 1

(*favorites_3* < 10), 2 (*favorites_3* < 50), 3 (*favorites_3* < 100) and 4 (*favorites_3* ≥ 100). These labels can be seen as 'not liked', 'little liked', 'liked', 'high liked' and 'absolutely liked'.



The new variable, *liked*, assumes the distribution of a Gaussian with a long tail, that is plausible for this type of phenomenon.

One of the last analysis is the study of **basic correlation and Spearman correlation**:



Obviously there is a correlation between *weekend* and *day_of_week*, but also between *month* and one season, *season=autumn*. The first graph shows other high correlations: *listen_1* - *comments_1*, *favorites_2* - *comments_2*, *listens_2* - *favorites_3*, *interest* - *favorites_3* and *interest* - *listens_2*.
In the second graph, with spearman correlation, it is possible to find the relation between *liked* and *favorites_3*.
Features highly correlated with another are eliminated (*comments_1*, *comments_2*, *listens_2*, *interest*, *favorites_3*, *month*, *day_of_week*) and also redundant variables (*season* and *date_released*).

The last part of data preparation consists in **replacing missing values**:

- *type*: with just 6% can be easily replaced by the mode.
- *location*: is filled with 'Unknown', because an unknown recording studio can tell a lot of things about the song and the artist, for example that was recorded in an homemade studio by an emergent artist.
- *genre_top*: needs an accurate replacing. Filling operation has to be done using the mode filtered by another variable, different from the possible target variable otherwise will be difficult to replace the Test Set.
  So the choice is filtering by *type* and replacing with 'Experimental' (if *type*='Album'), 'Rock' ('Live Performance', 'Radio Program', 'Contest') and 'Electronic' ('Single Tracks').
- *year*: replaced by a rounded mean.

Before passing on to the next step, the classification, it is necessary to remove features with a wide range of possible values.

```
len(train.location.unique())
```
2236

```
len(train.name.unique())
```
14240

→ *location* and *name* is removed from the dataset.

1.3 Classification tasks

In the previous step *favorites_3* was modified to create the target variable *liked*, that expresses how much the song is appreciated through five labels, from 0 to 4. This could be a good outcome variable, because it represents an approval rating of the song's listeners, but also because the correlation analysis shows its independence.
The main purpose of this phase is to use basic classification methods, decision tree and KNN, to predict the likeness of a track. Before starting, it is essential normalizing features, to give them the same relevance, and labelling not numeric variables. In this project will be used MinMaxScaler that allows to have a fitter to apply to Test Set.
Training Set is further divided into Training Set and Validation Set. The two datasets are composed of 71937 rows (90%) and 7993 (10%).

1.3.1 Decision Tree

Tuning of the tree is implemented on Validation Set by creating multiple trees that recombine different values.

```
'max_depth': [None] + list(np.arange(2, 20)),
'min_samples_split': [2, 3, 4, 5, 10, 20, 30, 50, 100],
'min_samples_leaf': [1, 2, 3, 4, 5, 10, 20, 30, 50, 100]
```

To decide which is the best criterion to use this procedure is applied for both Gini and Entropy, after this the top 3 trees are visualized.

Gini:

```
Model with rank: 1
Mean validation score: 0.700 (std: 0.001)
Parameters: {'min_samples_split': 30, 'min_samples_leaf': 4, 'max_depth': 14}

Model with rank: 2
Mean validation score: 0.699 (std: 0.001)
Parameters: {'min_samples_split': 30, 'min_samples_leaf': 2, 'max_depth': 15}

Model with rank: 3
Mean validation score: 0.699 (std: 0.002)
Parameters: {'min_samples_split': 50, 'min_samples_leaf': 3, 'max_depth': 16}
```

Entropy:

```
Model with rank: 1
Mean validation score: 0.699 (std: 0.003)
Parameters: {'min_samples_split': 50, 'min_samples_leaf': 5, 'max_depth': None}

Model with rank: 2
Mean validation score: 0.698 (std: 0.004)
Parameters: {'min_samples_split': 50, 'min_samples_leaf': 5, 'max_depth': 18}

Model with rank: 3
Mean validation score: 0.697 (std: 0.002)
Parameters: {'min_samples_split': 30, 'min_samples_leaf': 10, 'max_depth': 17}
```

It is easy to notice that all Gini's Mean Validation Scores are better or equal to the Entropy's first. Accordingly to results, the project proceeds with a depth analysis of Gini's top 3.
In the table below it's possible visualizing more detailed information about the Gini's top 3 trees. **Precision** is chosen as a significant value, instead of Recall, because *liked* has more than two classes and they have the same relevance.

Placement	Set	Accuracy	Labels' Precisions
1°	Training	0.7609	**0:** 0.74 **1:** 0.78 **2:** 0.75 **3:** 0.58 **4:** 0.67
1°	Validation	0.7026	**0:** 0.69 **1:** 0.72 **2:** 0.62 **3:** 0.19 **4:** 0.33
2°	Training	0.7718	**0:** 0.75 **1:** 0.79 **2:** 0.76 **3:** 0.59 **4:** 0.70
2°	Validation	0.7066	**0:** 0.69 **1:** 0.73 **2:** 0.62 **3:** 0.19 **4:** 0.33
3°	Training	0.7636	**0:** 0.75 **1:** 0.78 **2:** 0.71 **3:** 0.56 **4:** 0.67
3°	Validation	0.7047	**0:** 0.69 **1:** 0.73 **2:** 0.61 **3:** 0.21 **4:** 0.33

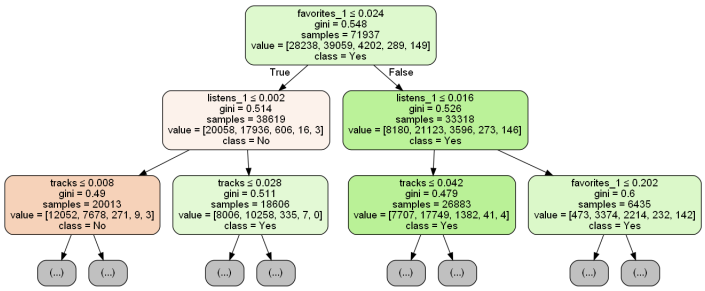
The highest Accuracy is, obviously, in the first tree, but only for the Training Set, because the highest Validation Set's Accuracy is in the second. At the end, the third decision tree is chosen, because of their fairer distribution of Precision in the Validation Set.

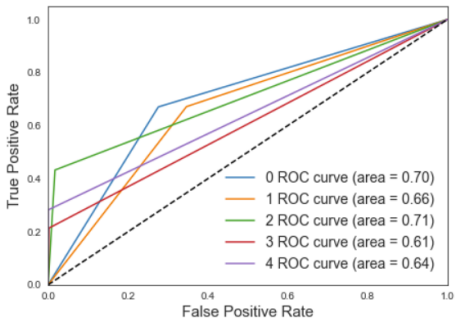
Now, the third tree is applied to the Test Set:

Placement	Set	Accuracy	Labels' Precisions
3°	Test	0.6540	**0:** 0.62 **1:** 0.69 **2:** 0.62 **3:** 0.31 **4:** 0.45

The Accuracy is not so far from third Validation Set, this is a good thing, and Precision is fairer than all Validation Set.

Below it can be seen how the most important features (*favorite_1*, *listens_1* and *tracks*) contribute to the construction of tree and the ROC curve of Test Set prediction:





Labels with the best area under the curves are, in order, 2, 0, and 1. These three labels have a great representation and this aspect affects positively the classification. ROC curves analysis shows a quite fair classification.

1.3.2 KNN

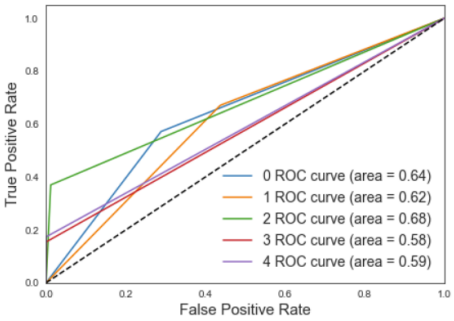
In this section is used the Cross Validation to determine the number of most suitable neighbors for the KNN approach, therefore the most suitable *n_neighbor* for the classifier. The Accuracy and F1-score are averaged at the end of every *n_neighbor*'s Cross Validation analysis.

```
n_neighbors= 1              n_neighbors= 3              n_neighbors= 5              n_neighbors= 7              n_neighbors= 9
Accuracy: 0.6635 (+/- 0.01) Accuracy: 0.6796 (+/- 0.01) Accuracy: 0.6804 (+/- 0.01) Accuracy: 0.6775 (+/- 0.01) Accuracy: 0.6738 (+/- 0.01)
F1-score: 0.4586 (+/- 0.06) F1-score: 0.4674 (+/- 0.06) F1-score: 0.4749 (+/- 0.07) F1-score: 0.4627 (+/- 0.06) F1-score: 0.4473 (+/- 0.06)

n_neighbors= 2              n_neighbors= 4              n_neighbors= 6              n_neighbors= 8              n_neighbors= 10
Accuracy: 0.6631 (+/- 0.01) Accuracy: 0.6751 (+/- 0.01) Accuracy: 0.6764 (+/- 0.01) Accuracy: 0.6739 (+/- 0.01) Accuracy: 0.6707 (+/- 0.01)
F1-score: 0.4342 (+/- 0.06) F1-score: 0.4623 (+/- 0.07) F1-score: 0.4656 (+/- 0.07) F1-score: 0.4440 (+/- 0.04) F1-score: 0.4356 (+/- 0.06)
```

The best prediction, so with the highest value of Accuracy and F1-score, is found by using 5 neighbors. The right *n_neighbor* is applied to the Test Set's KNN analysis.

n_neighbor	Set	Accuracy	Labels' Precisions
5	Test	0.6103	**0:** 0.57 **1:** 0.64 **2:** 0.67 **3:** 0.39 **4:** 0.50



Accuracy is lower than the decision tree's classification, but the distribution of Precision is the fairest until now. However the areas under the curves tell that this is a very poor classification, it is worse than the Decision Tree one.

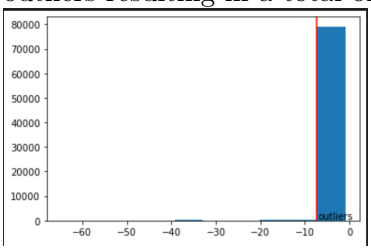
1.4 Anomaly Detection

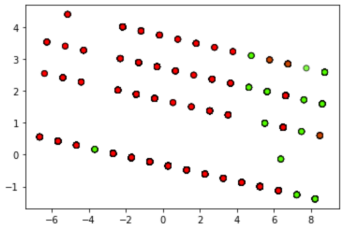
The goal of this task is to identify anomalies within the dataset, report the top 1% anomalies and after that deal with the outliers. In order to complete this task it has been decided to implement different methods belonging to different classes in particular: density based, neural network and ensemble approach.

1.4.1 Density Based approach

The first algorithm implemented belonging to this class is the DBSCAN. This clustering algorithm does not need to precompute the number of clusters and automatically detects outliers points in the dataset marking them as '-1' after the analysis. After setting the hyperparameters with *min_points*=8 and *eps*= 0.5 (found through the knee method) the algorithm detected a total

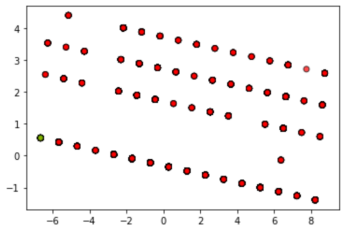
number of 1213 outliers; which is expected considering the kind of data explored.
The second algorithm based on density applied has been the Local Outlier Factor, since the LOF presented in general high values the points with a score greater than 7 have been chosen as outliers resulting in a total of 800 points.





1.4.2 NN based approach

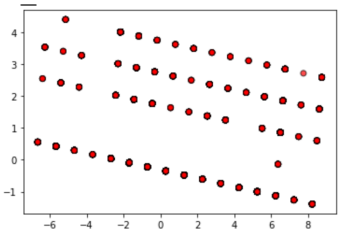
For this method it has been decided to implement the AutoEncoder algorithm, it represents the points through compression and decompression of a Deep Neural Network. The algorithms calculates the **reconstruction error** for each point labeling as outliers the points that score the highest values. The parameters of this model are the following:
Epochs: 50 *hidden_neurons*=[4,2,2,4] *contamination*= 0.01



The last parameter, *contamination*, reports the first 1% points classified as outliers, which were 800. The image on the left, shows a representation of outliers in the PCA with the number of components equal to 2.

1.4.3 Ensemble Method

For this class of algorithms it has been chosen to implement the Isolation Forest. This algorithm has been run with the following parameters after trying different configurations:
n_estimators=100 *contamination*=0.01 *random_state*=0 *bootstrap*=true



The number of outliers found with this method has been lower, as a matter of fact the number of points that have been classified as outliers is equal to 222. The image on the left, shows a representation of outliers in the PCA with the number of components equal to 2.

Given the dimensions of the dataset the number of outliers found with the methods listed above has been substantial. For this reason instead of just listing all the points, which wouldn't be much informative by its own it has been decided to go a bit more in depth in the analysis: an inner join on the datasets found by implementing the above mentioned methods has been implemented giving as a result the common points identified as outliers.
The results are interesting because of all points found as anomalies, the common found are very few: 2 to be precise. This raised confusion because given the dimension of the dataset it would have been expected to find many common points. The explanation given to this phenomenon is the following: although the dataset is big and the points identified as anomalies are a relatively large number the methods implemented belong to different classes of algorithms (density, ensembled and a NN-like method).

The high number of hyperparameters to estimate might also be considered a factor.

1.4.4 Conclusion

After the elimination of anomaly points, the distribution has drastically changed, as it is shown in table below.

Class	Before	After	%
0	31375	31102	0.9%
1	43399	43068	0.8%
2	4669	4609	1.3%
3	321	269	16.1%
4	166	82	50.6%

Anomaly Detection identifies a lot of anomaly records in the class 4. Decreasing this class transforms all its records into outliers, because the class became too small to be considered a class again.

Having a lot of anomaly points in the class 4 is reasonable, because it contains songs and albums very appreciated by the public, so they are very far from "normal" values and fewer than the others.

Applying the best predictor, the Decision Tree, with the parameters of the third one, the changes become clear.

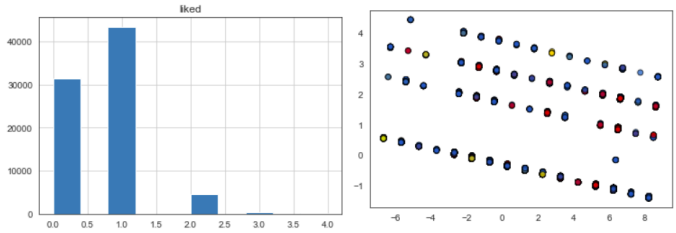
Approach	Set	Accuracy	Labels' Precisions
Tree	Training	0.7663	**0:** 0.75 **1:** 0.78 **2:** 0.71 **3:** 0.50 **4:** 0.00
Tree	Validation	0.7010	**0:** 0.67 **1:** 0.73 **2:** 0.63 **3:** 0.38 **4:** 0.00
Tree	Test	0.6612	**0:** 0.62 **1:** 0.70 **2:** 0.60 **3:** 0.30 **4:** 0.00

Accuracy could seem better, but this effect is due to a smaller representation of the class 4. The distribution reduction decreases the Precision's value.

1.5 Imbalanced Learning

Here, balancing is used with outliers, in order to show a comparison in the next section.
Before starting, a quick view to the distribution of *liked*'s classes could be very useful.

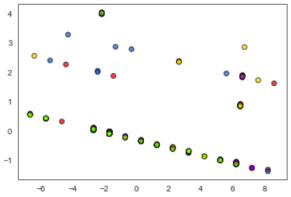
Class	Records	Distribution
0	31375	39.3%
1	43399	54.3%
2	4669	5.8%
3	321	0.4%
4	166	0.2%



In PCA representation is not clear the classes' distribution, because of too many different colors due to the 5 classes and a big dimension reduction. But, in the table and graphic is evident that classes are imbalanced.
For a better analysis Training Set is again divided into Training Set (90%) and Validation Set (10%). Balancing process, applied only to the first one, goes through different methods: **Undersampling**, **Condensed Nearest Neighbors**, **Oversampling**, **SMOTE** and **Weighted Decision Tree**.
Every prediction is compared with:

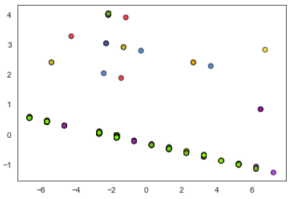
- Predictions and values of Third Decision Tree, chosen before as the best
- Predictions and values of KNN with $n\_neighbors$=5

With the aim of obtaining a fair comparison, balanced Decision Tree and KNN have the same parameters.



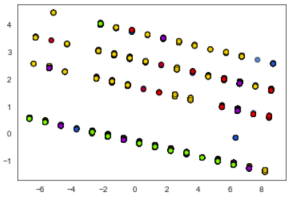
Undersampling: this technique could negatively affect the result, because it is more difficult to train a tree or a KNN by decreasing randomly the number of examples.
The Accuracy appears very low, but also Precision's value are lower than the Decision Tree and KNN.

Approach	Set	Accuracy	Labels' Precisions
Tree	Training	0.6497	**0:** 0.72 **1:** 0.64 **2:** 0.55 **3:** 0.62 **4:** 0.72
Tree	Validation	0.5180	**0:** 0.57 **1:** 0.68 **2:** 0.18 **3:** 0.08 **4:** 0.11
Tree	Test	0.5164	**0:** 0.58 **1:** 0.68 **2:** 0.18 **3:** 0.05 **4:** 0.07
KNN	Training	0.5946	**0:** 0.54 **1:** 0.55 **2:** 0.58 **3:** 0.67 **4:** 0.72
KNN	Validation	0.4469	**0:** 0.45 **1:** 0.56 **2:** 0.19 **3:** 0.04 **4:** 0.02
KNN	Test	0.4573	**0:** 0.46 **1:** 0.58 **2:** 0.23 **3:** 0.06 **4:** 0.04



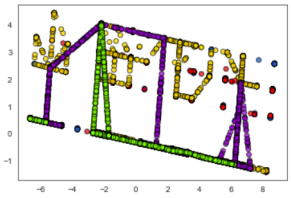
Condensed Nearest Neighbor: is based on KNN and it is sensitive to noise.
Accuracy and Precision's values are the worst until now in all datsets.

Approach	Set	Accuracy	Labels' Precisions
Tree	Training	0.5535	**0:** 0.68 **1:** 0.59 **2:** 0.52 **3:** 0.45 **4:** 0.56
Tree	Validation	0.5147	**0:** 0.54 **1:** 0.62 **2:** 0.23 **3:** 0.02 **4:** 0.30
Tree	Test	0.4826	**0:** 0.51 **1:** 0.62 **2:** 0.21 **3:** 0.02 **4:** 0.24
KNN	Training	0.4311	**0:** 0.37 **1:** 0.42 **2:** 0.42 **3:** 0.48 **4:** 0.66
KNN	Validation	0.3918	**0:** 0.43 **1:** 0.55 **2:** 0.11 **3:** 0.01 **4:** 0.11
KNN	Test	0.3972	**0:** 0.45 **1:** 0.56 **2:** 0.11 **3:** 0.01 **4:** 0.32



Oversampling: is not so smart and the different weight of classes could change the behavior of algorithm.
The Accuracy of Training and Validation seems better, but values in Test Set are lower than the Decision Tree and KNN.

Approach	Set	Accuracy	Labels' Precisions
Tree	Training	0.8634	**0:** 0.76 **1:** 0.74 **2:** 0.86 **3:** 0.95 **4:** 0.98
Tree	Validation	0.6516	**0:** 0.67 **1:** 0.75 **2:** 0.31 **3:** 0.12 **4:** 0.14
Tree	Test	0.6193	**0:** 0.62 **1:** 0.71 **2:** 0.32 **3:** 0.13 **4:** 0.20
KNN	Training	0.9143	**0:** 0.80 **1:** 0.85 **2:** 0.93 **3:** 1.00 **4:** 1.00
KNN	Validation	0.6549	**0:** 0.63 **1:** 0.74 **2:** 0.39 **3:** 0.15 **4:** 0.33
KNN	Test	0.5785	**0:** 0.55 **1:** 0.66 **2:** 0.39 **3:** 0.15 **4:** 0.17



SMOTE: creates structures in the data generating points in the line between two points, but there is no guarantee that new points have the same label of the nearest two.
Values are quite good in Training Set and Validation, but Test Set's values are still low.

Approach	Set	Accuracy	Labels' Precisions
Tree	Training	0.8508	**0:** 0.76 **1:** 0.74 **2:** 0.86 **3:** 0.92 **4:** 0.96
Tree	Validation	0.6710	**0:** 0.66 **1:** 0.75 **2:** 0.38 **3:** 0.10 **4:** 0.19
Tree	Test	0.6350	**0:** 0.63 **1:** 0.72 **2:** 0.33 **3:** 0.09 **4:** 0.14
KNN	Training	0.9102	**0:** 0.79 **1:** 0.85 **2:** 0.93 **3:** 0.99 **4:** 0.99
KNN	Validation	0.6573	**0:** 0.64 **1:** 0.75 **2:** 0.39 **3:** 0.14 **4:** 0.17
KNN	Test	0.5729	**0:** 0.56 **1:** 0.65 **2:** 0.34 **3:** 0.13 **4:** 0.13

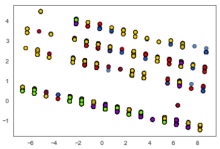
Weighted Decision Tree: is the same tree as before, but classes have different weights {0:1, 1:1, 2:1, 3:3, 4:3}. Values are not so high, especially Precision's values of class 3 and 4.

Approach	Set	Accuracy	Labels' Precisions
Tree	Training	0.7101	**0:** 0.69 **1:** 0.81 **2:** 0.43 **3:** 0.26 **4:** 0.44
Tree	Validation	0.6611	**0:** 0.66 **1:** 0.75 **2:** 0.36 **3:** 0.12 **4:** 0.14
Tree	Test	0.6272	**0:** 0.63 **1:** 0.73 **2:** 0.31 **3:** 0.07 **4:** 0.16

Oversampling and **SMOTE** have obtained the best results, although they are worse than the previous, obtained without balancing. Methods have been influenced by negative aspects of the approaches and, also, by the structure of the dataset, a multiclass and very unbalanced case and probably an unsuitable example. A clear proof of this issue is represented by the resizing of **Undersampling**, where every class weighs 149 records, this is due to the weight of the class 4.

1.6 Conclusions

After the previous examination of the dataset, it is possible to make same inferences:

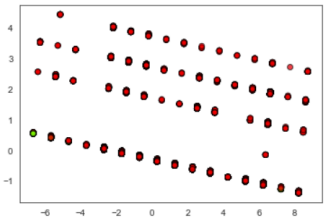
- The Decision Tree approach is better than the KNN one, so the dataset isn't suitable for the neighborhood investigation.
-  Joining Anomaly Detection and balancing could be a good solution for the low level of class 4. The same Decision Tree criteria are applied, after the balancing with the Oversampling.

Approach	Set	Accuracy	Labels' Precisions
Tree	Training	0.8744	**0:** 0.76 **1:** 0.75 **2:** 0.87 **3:** 0.97 **4:** 0.99
Tree	Validation	0.6426	**0:** 0.64 **1:** 0.75 **2:** 0.34 **3:** 0.08 **4:** 0.09
Tree	Test	0.6196	**0:** 0.62 **1:** 0.71 **2:** 0.31 **3:** 0.13 **4:** 0.00

The Tree works better with the Training Set and the Validation Set, but it fails in the Test Set. The distribution problem could not be solved by using the balance method.

- The dataset structure does not allow a correct balancing. To discover if with a binary class a deeper analysis will be possible, the target label, so far multiclass, will be modified into:
  Class 1: includes the old classes 3 and 4. It represents songs and albums having success.
  Class 0: composed by the old classes 0, 1 and 2. It can be seen as the class 'No success'.

Class	Records	Distribution
0	79443	99.4%
1	487	0.6%



Also in this case the dataset is unbalanced.

The dataset now has a class label more important than the other, class 1, so it's possible to make the Recall analysis of this class. Below there are the prediction's values with third Decision Tree parameters:

Balancing	Set	Accuracy	Labels' Precisions	Class 1's Recall
No	Training	0.9957	**0:** 1.00 **1:** 0.76	0.42
No	Validation	0.9952	**0:** 1.00 **1:** 0.69	0.41
No	Test	0.9944	**0:** 1.00 **1:** 0.50	0.33
Oversampling	Training	0.9916	**0:** 1.00 **1:** 0.98	1.00
Oversampling	Validation	0.9804	**0:** 1.00 **1:** 0.19	0.67
Oversampling	Test	0.9826	**0:** 1.00 **1:** 0.18	0.62

Models doesn't improve, but after balancing the Recall has an higher value. This depth analysis is possible only with a binary class.

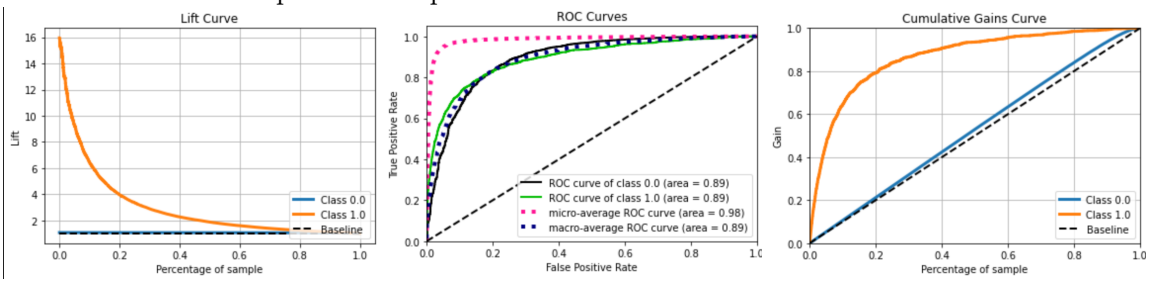
2 Module 2

2.1 Advanced Classification

The dataset as prepared in section 1 has been used for further analysis, in order to complete the classification task. It has been decided to create a new attribute from the original one *liked*: this was a multi-valued one (with values ranging from 0 to 4), for this task it has been transformed in a binary one of value 0 and 1 named *liked_bin*.

2.1.1 Logistic Regression

Given the fact that the class to predict for this specific task is a binary one the logistic regression algorithm has been applied in order to perform the classification. The results obtained were satisfactory since the model scored an accuracy of 94.89% in predicting the target variable while running with the following parameters:
C=100 *penalty*= l2 *class_weight*= balanced *solver*="lbfgs"
Here can be seen the plots of the performance evaluation of the model obtained.



As can be seen above the model performs well for the class 1 but not as well for class 0 since in the cumulative gain plot the class 0. The line is just a little bit higher than the baseline.

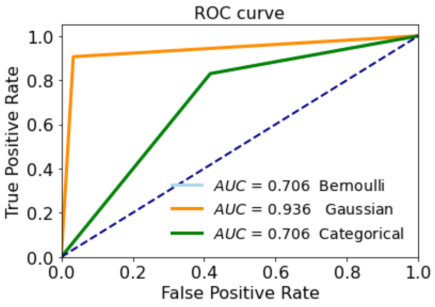
2.1.2 Naive Bayes Classifier

For the Naive Bayes approach the algorithms implemented were taken from the sklearn library, in particular the following algorithms were applied to the data: BernoulliNB, GaussianNB and CategoricalNB. Different sets of parameters have been tested while performing the hyperparameters tuning phase many of which resulted in an overfit of the models.

The final sets of hyperparameters can be found in the following table.

Model	Alpha	Var Smoothing
GaussianNB	None	$1e^{-9}$
BernoulliNB	0.5	None
CategoricalNB	1	None

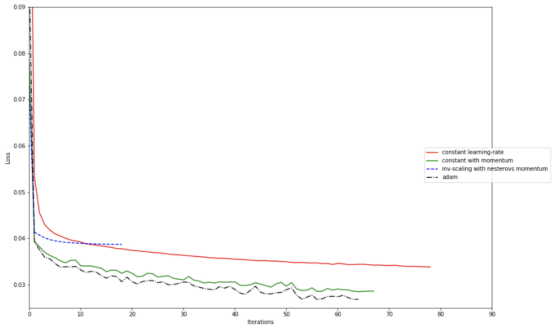
As a summary of the results obtained through Naive Bayes classifiers, the following image presents the plot of the AUC curves of the models.



As can be seen from the picture above the best performer is the Gaussian with an AUC value of 0.936 while the Bernoulli and Categorical perform far worse than that with an equal AUC value of. 0.706.

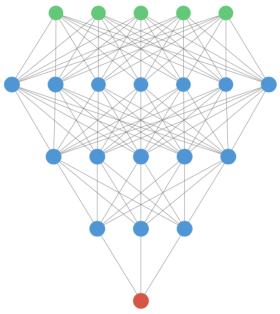
2.1.3 Neural Network

In this section will be illustrated the results of the Neural Network classification. In order to perform this analysis the Training Set was further split in a validation set consisting of 30% of its data,in order to see how well the model is performing after every epoch. The library used for this analysis is Keras which provides simple tools for this kind of task. As a start, multiple configurations of a Multi Layer Network have been implemented on a 100 epochs range. The results can be seen in the following figure.



As can be seen, the better performer is the algorithm which implements the Adam learning rate optimization algorithm. Adam can be used instead of the classic stochastic gradient descent procedure to update network weights iterative based in training data.

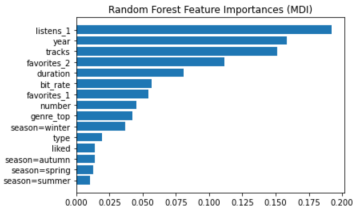
As a consequence of this finding, Adam was implemented in the first model which consisted of a single hidden layer made by four nodes with a logistic function as activation function and an initial learning rate of 0.3. This model scored an accuracy of 96.32% with a loss of 0.04 on Training and a loss of 0.0497 on the Validation Set.
Even though the results of this simple model were quite satisfactory, it has been decided to test another more complex model, a Deep Neural Network model. As a first step different hyperparameters combinations have been explored, the evaluation was made on the Validation set taking in consideration Accuracy and Loss as metrics.The final model selected is composed of three hidden layers of size 7, 5 and 3 as can be seen in the picture that follows.



The selected activation function in each layer is the *"relu"* function and the one for the output node is the logistic. The applied learning rate was the minibatch with the value of 10. The Accuracy achieved by the model is equal to 0.9738 at epoch 100.

2.1.4 Ensemble Methods

In this section will be implemented Ensemble Methods for the task on classification. The ratio on which this methods base is the so called wisdom of the crowd, the concept by which a general knowledge from a group of individuals can outperform the knowledge of a single expert subject. Translated to the machine learning domain this means that the following algorithms generate not a single model but many models at the same time and then aggregate them together. Random forest classifier performed well in this analysis scoring an accuracy of 96.9% and a cross validation-score of 96.72%.



The figure shows the feature importances of the random forest classifier.
As can be seen the feature *listens_1* is the most relevant followed by *year* and *tracks* in the first three positions. Bagging and Boosting were also applied to the Random Forest as the base classifier scoring accuracy values of 95.3% and 97.03% respectively.

2.1.5 Support Vector Machines

Support Vector Machines are a form of supervised classification. For this section two kinds of SVMs have been implemented: the linear and the non-linear kernels. Starting with Linear SVMs multiple combination of parameters have been tried as can be seen in the table below:

Loss	**C**	**Weight**
Hinge	100	Balanced
Hinge	1	Balanced
Hinge	0.01	Balanced
Hinge	100	None
Hinge	1	None
Hinge	0.01	None

Keeping parameters unchanged, instead of Hinge the "squared Hinge" loss function has been tested. The results showed that when the LinearSVM was implemented with the "Squared Hinge" loss function the model performed the worst. The best performer is the model with the following parameters combination:

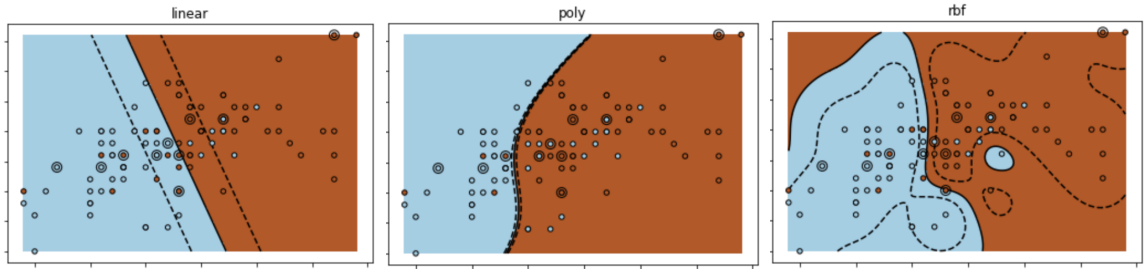
Loss	**C**	**Weight**
Hinge	100	Balanced

In this case the model scored the following results: **Accuracy** = 0.82 and **Recall** = 0.86.
For the non-linear SVMs in order to determine the best hyperparameters a random search has been performed. The elements compared were the following.
Kernels: 'linear', 'polynomial', 'radial basis function', 'sigmoid'

C: 1, 0.01, 10, 100 and 1000 *degree (only for polynomial)*: 2, 3, 5, 7
class weight: None, balanced *gamma*: 'scale', 'auto'
Because of the dimensionality of the dataset this step took a great amount of time but at the end the final model returned was the following:

Kernel	gamma	C	Weight
rbf	auto	100	balanced

The results scored by this model are the following: **Accuracy** = 0.84 and **Recall** = 0.82. Below a picture of the boundaries of the various SVM considered in this section.



2.1.6 Rule Based Classifier

In this section classification task will be performed implementing models that base their prediction on rules extracted from patterns of data in the dataset. The class to be predicted is the class *"liked_ bin"*. For this specific task the algorithms implemented are the RIPPER and the IREP algorithms. The RIPPER algorithm performs with an **accuracy** of 94.78%. The rules found are a very large number even when optimizing the hyperparameters but in order to make the work more comprehensible only the top four rules will be listed:

```
[listens_1=0.02-0.33^favorites_1=0.06-1.0^favorites_2=0.39-1.0]
[listens_1=0.02-0.33^favorites_1=0.06-1.0^favorites_2=0.04-0.39]
[listens_1=0.02-0.33^favorites_1=0.06-1.0^tracks=0.02-0.02^season=winter=1.0]
[listens_1=0.02-0.33^favorites_1=0.06-1.0^favorites_2=0.02-0.04^tracks=0.02-0.02]
```

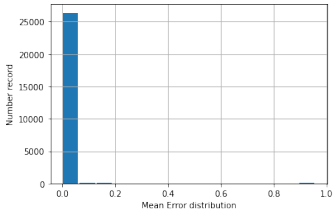
The IREP algorithm has a slightly higher **accuracy** than RIPPER scoring a value of 94.88%. Below are listed the first four rules extracted with this method.

```
[listens_1=0.02-0.33^favorites_1=0.06-1.0^favorites_2=0.39-1.0]
[listens_1=0.02-0.33^favorites_1=0.06-1.0^favorites_2=0.04-0.39]
[listens_1=0.02-0.33^favorites_1=0.06-1.0^tracks=0.02-0.02]
[listens_1=0.02-0.33^tracks=0.01-0.01^favorites_1=0.06-1.0^bit_rate=0.57-0.71]
```

2.2 Regression

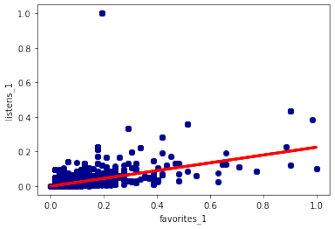
After exploring all possible pairs of features, the regression problem is addressed based on *favorites_ 1*, used as independent variable and *listens_ 1* as dependent variable.
The linear regression approach was applied and evaluated through the typical measures: **R2 score, Mean Squared Error** and **Mean Absolute Error**.

The values obtained, also visible from the histogram, indicate that the prediction of the values of the dependent variable is good, although not optimal, since there is a high presence of errors, mainly less than 0.2.



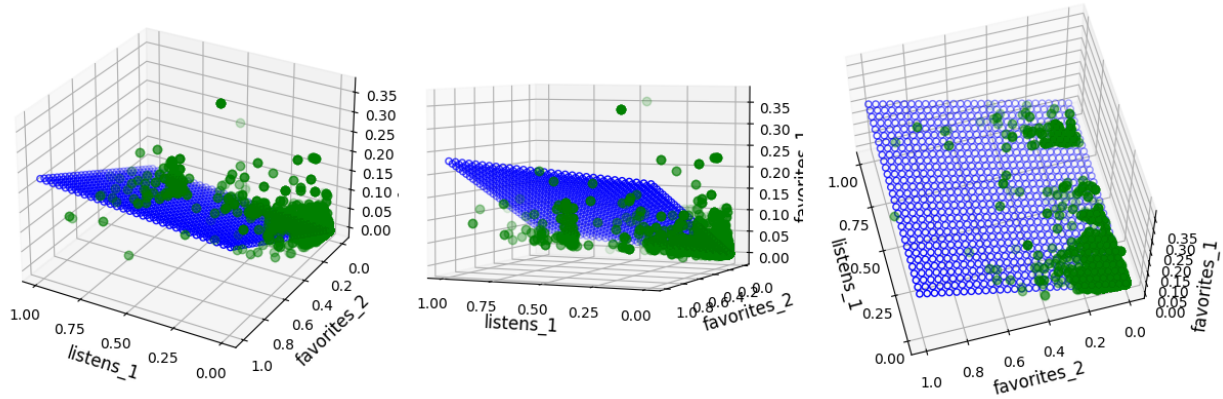
The same problem has been solved with other types of regression algorithms (LassoLarsCV, Ridge

and Bayesian Ridge) but obtaining identical results, which is why it was retained sufficient to present the results only for the linear regression: R2=0.155, MSE=0.002 and MAE=0.008 . The relationship discovered from the linear regression between the analysed features can be expressed through the equation $\mathbf{listens\_1} = 0.2251 * \mathbf{favorites\_1} - 0.0008$.



The graphics below show a 3D representation of the variables and the plane described by the equation resulting from the multiple linear regression, which was $\mathbf{listens\_1} = 0.03622 * \mathbf{favorites\_2} + 0.1935 * \mathbf{favorites\_1} - 0.0008$.

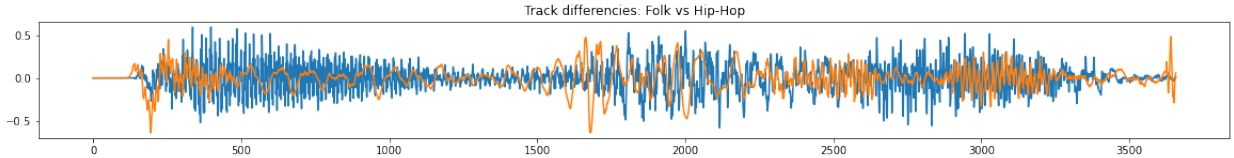
The problem has been generalized to several dependent variables through multiple regression. In fact the addition of *favorites_2* as dependent variable, made it possible to obtain a slight improvement in the metrics, with R2=0.375, MSE=0.002 and MAE=0.007.



3 Module 3

3.1 Time Series Creation

For this module a new dataset has been created in order to extract the time series. The time series have been extracted from the *fma_small* in the form of spectral centroids and filtered by the genres *Folk* and *Hip-Hop*, in order to have a balanced dataset and because it is expected that these two genres will generate different spectrogram waves. The most important decision to take while extracting the data is to correctly choose the sample rate of the music files. This parameter influences the quality of the audio which in return influences the capability of correctly examine the data in the future steps. After different tests the decision taken is to consider a sampling rate of 7000. While 7000 might appear generally low for this task proved to be acceptable. For each song we have considered a segment of 10 seconds.



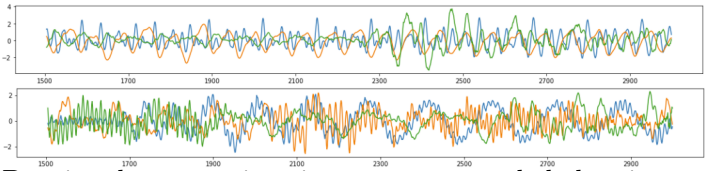
3.2 Clustering

3.2.1 Approximation

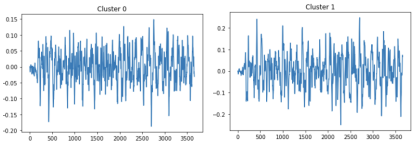
Spectrogram waves have a very different shape, so they need to be approximated in order to become more similar to the others belonging to same genre. Approximations applied are:

- Offset translation
- Amplitude Scaling
- Smoothing (with a window of size 5)

Trend removal was not used, because songs haven't a particular trend, it is easy to notice that all times series have a flat trend.



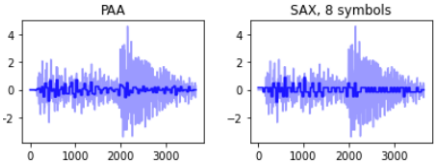
Here it is possible to visualize the result with three examples of *Folk*'s trends, in the first graph, and also three examples of *Hip-Hop*'s trend, in the second graph.
Despite the approximation process, trends belonging to different classes appear different and distinct, but also more fluid and clearer.
Before applying the compression, a Shape-based Clustering with the Euclidean distance is computed. The other distances are more computationally expensive and they will be applied after the compression approaches.
The Kmeans for Time Series doesn't return a good result, because the cluster distribution is not able to separate *Folk* songs from *Hip-Hop* songs.



Centers of clusters 0 and 1 are not so different, this could explain the bad genre's distribution.
Cluster 0: 629 *Folk* and 610 *Hip-Hop*.
Cluster 1: 390 *Folk* and 371 *Hip-Hop*.

3.2.2 Compression

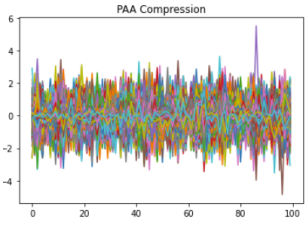
For the compression phase two types of approaches are used:



- **PAA**: with 100 segments;
- **SAX**: with 100 segments and 8 symbols.

For both compressed dataset are used and compared three different distances: Euclidean, DTW, soft DTW (a light version) and Manhattan. Manhattan distance needs a precomputed distance matrix and a cluster algorithm able to accept the precomputed process, so it is used in DBSCAN.

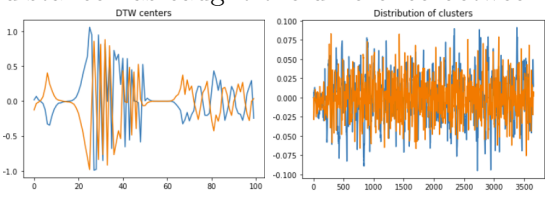
PAA:



Distance	Cluster 0 distribution	Cluster 1 distribution
Euclidean	50 *Hip-Hop* and 4 *Folk*	996 *Folk* and 950 *Hip-Hop*
DTW	650 *Hip-Hop* and 139 *Folk*	861 *Folk* and 350 *Hip-Hop*
soft DTW	79 *Hip-Hop* and 11 *Folk*	989 *Folk* and 921 *Hip-Hop*

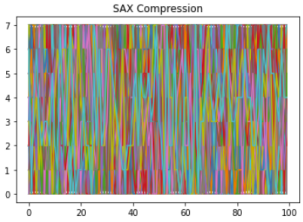
DBSCAN: with its best tunning, it finds 6 clusters, each one composed by 2 records, and 1988 outliers.

The best cluster distribution is obtained with the DTW, because in the cluster 0 we have the majority of *Hip-Hop* class and in the cluster 1 we have the majority of *Folk* class. The DTW distance has caught the difference between the two genres quite good.



Centers are well separated, they are almost like mirrors of each other and the distribution of clusters shows that the two sets of trends are not so different.

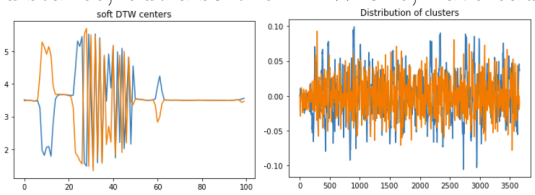
SAX:



Distance	Cluster 0 distribution	Cluster 1 distribution
Euclidean	128 *Hip-Hop* and 27 *Folk*	973 *Folk* and 872 *Hip-Hop*
DTW	850 *Folk* and 473 *Hip-Hop*	563 *Hip-Hop* and 150 *Folk*
soft DTW	639 *Hip-Hop* and 158 *Folk*	842 *Folk* and 361 *Hip-Hop*

DBSCAN: with its best tunning, in this case it finds 5 clusters, each one composed by 2 records, and 1990 outliers.

The best cluster distribution is obtained with the soft DTW, because in the cluster 0 we have the majority of *Hip-Hop* class and in the cluster 1 we have the majority of *Folk* class. The soft DTW distance, but also the DTW one, have caught the difference between the two genres quite good.



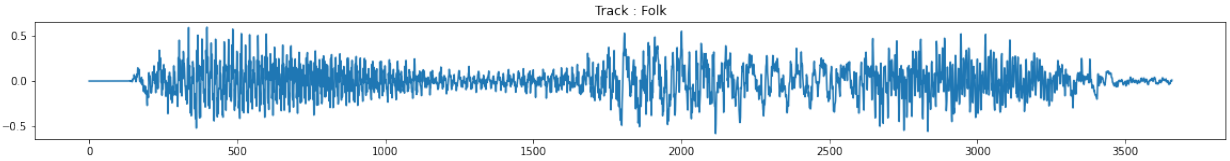
Also here centers are quite the opposites and the distribution of cluster shows that the two set of trends are not so different .

Conclusions:

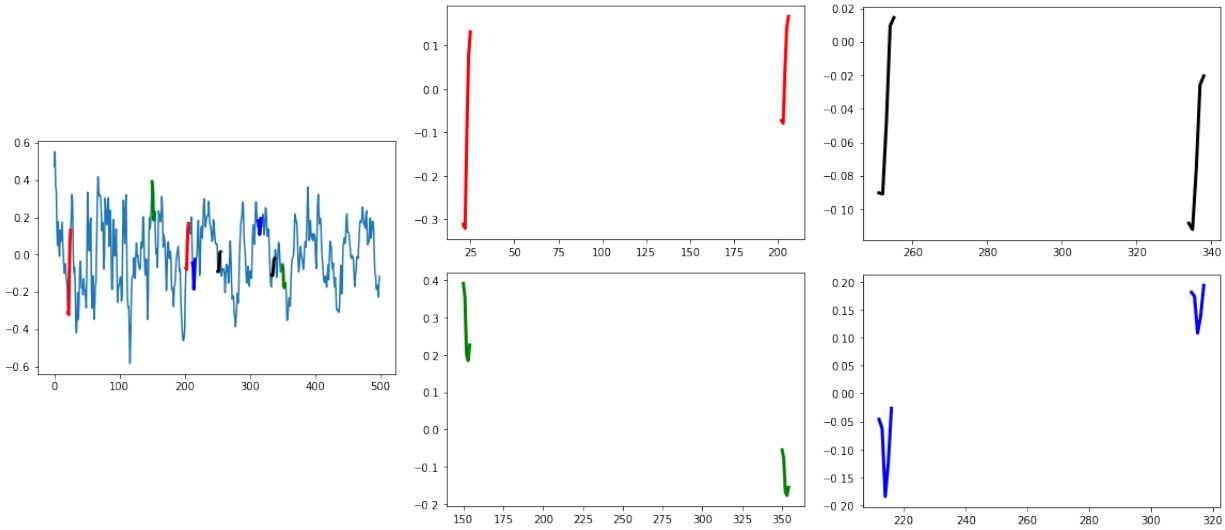
- The best distance approach is the DTW, both with the complete version and the light one. This because this approach is specific for time series.
- Using the Manhattan distance with DBSCAN is not convenient, because it is based on density and time series are not suitable for this approach.
- Euclidean distance has bad results.

3.3 Motif and Anomalies

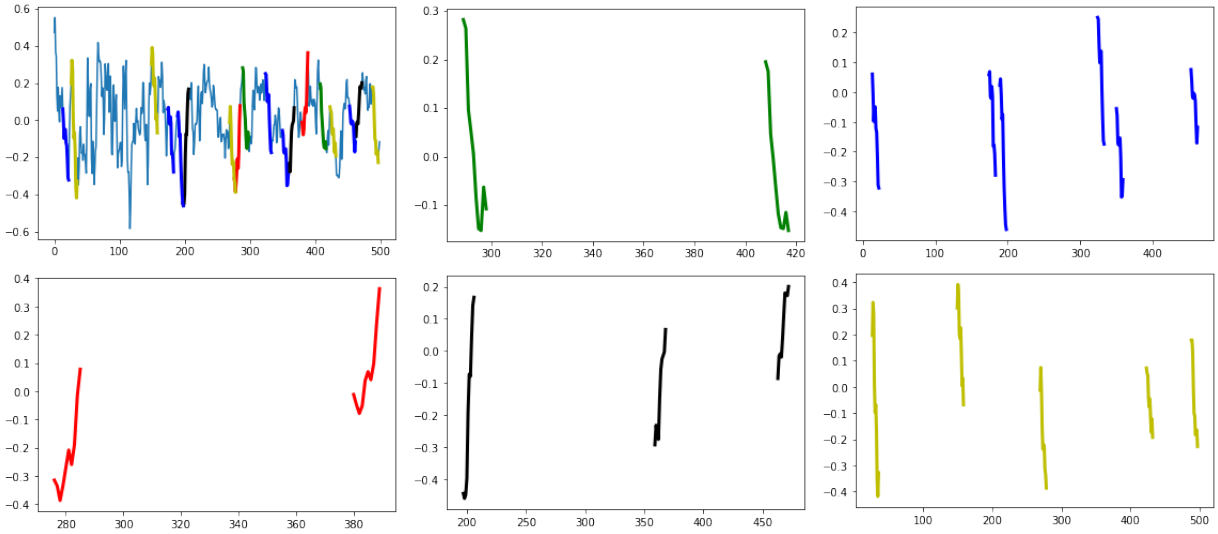
To search for motifs and anomalies the first track of *Folk* genre was chosen. More precisely, the middle part (1500:2000) of this track was chosen so that motifs and anomalies could be better visualized.



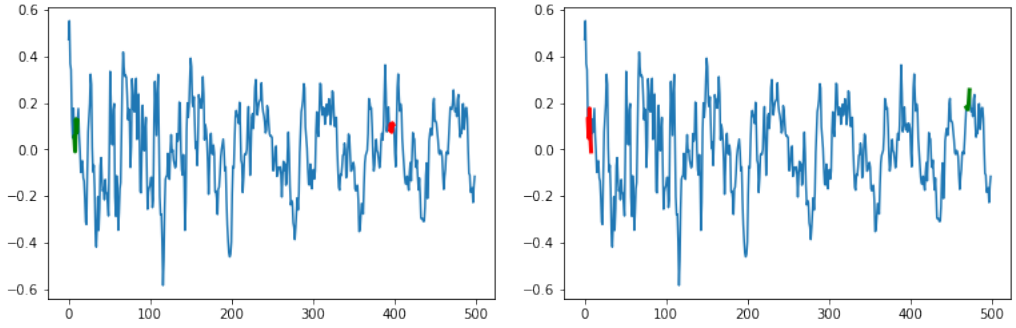
Different matrix profiles were built, giving the parameter of window size values of 5, 10, 12, 16, 20, 40. The most efficient value appeared to be window size equal to 5. This value allowed to capture only the most interesting motifs, excluding irrelevant details (white noise). Subsequently, the most frequent motifs were searched, which are shown in the graphs below.



For comparison, the results obtained with a window size of 10 are also reported, which are however interesting.



The **anomalies** were extracted using a window size of 5 (figure below left) and a window size of 10 (figure below right). What could be noted is that only a part of anomalies are in common.



3.4 Classification

For this task the dataset has been *normalized* in order to better perform the classification while implementing the various methods. The methods adopted to perform the classification task are

the following: shapelet approach, a distance based approach (KNN based algorithm), Decision tree algorithm, a Convolutional Neural Network and a special type of Recurrent Neural Network architecture called Long Short-term memory (lstm).

3.4.1 Shapelet Classifier

This algorithm is part of the "tslearn" library. It is based on the tensorflow package so there are many parameters to be tuned in order to obtain good results on the classification. In order to choose the best set of attributes various options have been considered.

optimizer: 'sgd', 'adam', 'adagrad' *weight regularizer*: 1, 0.01, 10 *max iter*: 50, 100, 200

The method applied to this search has been a RandomSearchCV with a CV parameter set to 5. After the execution of the algorithm the model found has the following characteristics:

MaxIter	Optimizer	weightReg
100	adagrad	10

The metrics of this model can be found in the table that follows. Even though the performance of the classifier is positive the values of accuracy and F1 scores were expected to be higher:

Accuracy	F1 Score Folk	F1 Score Hip Hop
0.67	0.47	0.66

3.4.2 KNN classifier

For the KNN algorithm the parameters have been chosen following the technique applied for the previous algorithm. In the end the best performing combination found is the following:

Neighbors	weights
10	uniform

This classifier performs way better than the one analyzed before, as shown in the table below:

Accuracy	F1 Score Folk	F1 Score Hip Hop
0.84	0.87	0.82

This algorithm will prove to be the one which returns the best results of all non-Neural Network algorithms.

3.4.3 Decision Tree

For this algorithms different combinations of parameters have been tested, in particular:

method: 'Gini', 'Entropy' *depth*: None, 8, 10, 20 *min sample split*: 2, 5, 10

The final model returned by the RandomSearchCV resulted to be the following:

Method	Depth	minSampleSplit
Gini	8	5

The final score of the model on the test set can be found in the table below:

Accuracy	F1 Score Folk	F1 Score HipHop
0.73	0.75	0.71

3.4.4 Convolutional Neural Network

Before running the algorithm it has been necessary to reshape the training, validation and test set. As a result of the reshape the *Number of timesteps* found is equal to 3658 and the *number of labels* is equal to 2. The final network implemented has been trained for 200 epochs and can be found in the picture that follows:

Model: "sequential"

Layer (type)	Output Shape	Param #
conv1d (Conv1D)	(None, 3651, 16)	144
batch_normalization (BatchNo	(None, 3651, 16)	64
activation (Activation)	(None, 3651, 16)	0
dropout (Dropout)	(None, 3651, 16)	0
conv1d_1 (Conv1D)	(None, 3647, 32)	2592
batch_normalization_1 (Batch	(None, 3647, 32)	128
activation_1 (Activation)	(None, 3647, 32)	0
dropout_1 (Dropout)	(None, 3647, 32)	0
conv1d_2 (Conv1D)	(None, 3645, 64)	6208
batch_normalization_2 (Batch	(None, 3645, 64)	256
activation_2 (Activation)	(None, 3645, 64)	0
dropout_2 (Dropout)	(None, 3645, 64)	0
global_average_pooling1d (Gl	(None, 64)	0
dense (Dense)	(None, 2)	130

Total params: 9,522
Trainable params: 9,298
Non-trainable params: 224

After the training phase and the validation phase, the model has been applied to the test set scoring the following results:

Accuracy	F1 Score Folk	F1 Score HipHop
0.86	0.81	0.90

Making it slightly better then the KNN classifier, a result expected due to the higher complexity of this algorithm. The *loss* of the model is equal to 0.21.

3.4.5 LSTM

The last model implemented for the time series classification task has been the *Long short term memory* architecture for a NN. A common LSTM unit is composed of a cell, an input gate, an output gate and a forget gate. The cell remembers values over arbitrary time intervals and the three gates regulate the flow of information into and out of the cell. The model implemented is the following:

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 256)	264192
dropout (Dropout)	(None, 256)	0
dense (Dense)	(None, 64)	16448
dropout_1 (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 2)	130

Total params: 280,770
Trainable params: 280,770
Non-trainable params: 0

The model didn't perform badly but didn't perform as well as the CNN presented in the section above. The final results can be found in this table:

Accuracy	F1 Score Folk	F1 Score HipHop
0.78	0.73	0.81

The *loss* found for this model is 0.29.

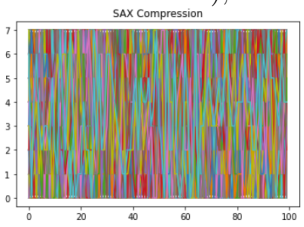
3.4.6 Final remarks

The classification task performed on the time series extracted from the original dataset proved to be a difficult task mainly because of the size of the data involved in the analysis. The approach taken in order to choose the best parameters has been successfully implemented but given the complexity of the algorithms it took a huge amount of time. We decided to take this approach because there isn't a clear way to proceed, many methods have been presented in the academic research but studies are still active on the subject.

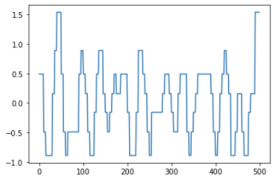
4 Module 4

4.1 Sequential Pattern Mining

This part of the analysis is computed in the central part of every time series, the purpose is discovering rules about the core of every song. Before the splitting, the times series dataset was approximated by applying the offset translation, the amplitude scaling and the smoothing (with a window of size 5), like in the previous step.



At the end the new dataset is compressed with the SAX method (with 100 segments and 8 symbols).
In order to have more significant results, entries are splitted into two dataset: one for the *Folk* songs and the other for the *Hip-Hop* ones.



Here there is an example of how the core of a random song appears. At the beginning the dataset started from column 0 and finished with the 3657 one. Now the analysis uses columns from 1500 to 2000, then 500 features.

The Pattern Mining research is applied on both *Folk* and *Hip-Hop* subsets. Below it's possible to see the top 10 frequent patterns and their supports.

```
Folk songs' results

[(1000, [3, 3, 3, 3, 3, 3, 3, 3, 3, 3]),
 (1000, [3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4]),
 (1000, [3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 3, 3, 3, 3, 3]),
 (1000, [3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4]),
 (1000, [3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 3, 3, 3, 3, 3]),
 (1000, [4, 4, 4, 4, 4, 4, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4]),
 (999, [3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3]),
 (999, [3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4]),
 (999, [3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4]),
 (999, [4, 4, 4, 4, 4, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3])]
```

```
Hip-Hop songs' results

[(999, [3, 3, 3, 3, 3]),
 (999, [4, 4, 4, 4, 4, 4, 4, 4, 4, 4]),
 (998, [4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 3, 3, 3, 3, 3]),
 (997, [3, 3, 3, 3, 3, 3, 3, 3, 3, 3]),
 (995, [3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4]),
 (995, [3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 3, 3, 3, 3, 3]),
 (994, [3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3]),
 (994, [3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4]),
 (994, [4, 4, 4, 4, 4, 4, 3, 3, 3, 3, 3, 3, 3, 3, 3]),
 (994, [4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4])]
```

After this new knowledge, some conclusions can be drawn:

- All *Folk* songs have in common 6 type of segment combinations.
- *Folk* songs are the most repetitive, but also *Hip-Hop* ones are not so unique.
- *Folk* and *Hip-Hop* songs are not so different, they have a great presence of segments 3 and 4 in the core, with different combinations.
- The segments 3 and 4 are the most popular segments, probably the length of the other segments is quite rare or not so used in these songs.

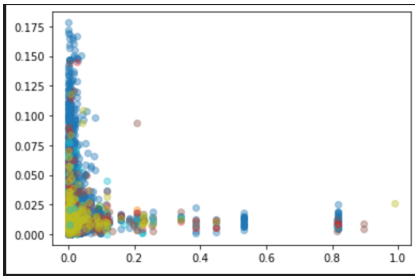
4.2 Advanced Clustering

Objective of this section is to perform clustering analysis on the original (non time series) dataset implementing advanced clustering algorithms. The libraries used to perform this task are: *SKLearn* and *pyclustering*.

4.2.1 OPTICS

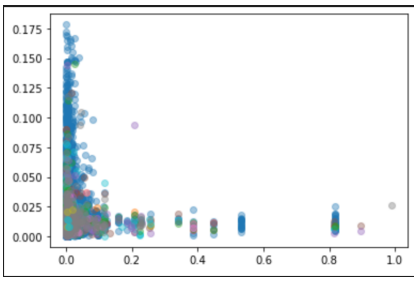
OPTICS is a density based clustering algorithm derived directly from the DBSCAN. OPTICS is more suited for large datasets since it allows to choose a variable range for the *epsilon* parameter.

SKLearn implementation deviates from the original OPTICS by first performing k-nearest-neighborhood searches on all points to identify core sizes, then computing only the distances to unprocessed points when constructing the cluster order. Time complexity will be O(n2).
The first clustering performed (parameter "cluster method" set to "dbscan") with this algorithm returned the results shown in the following figure:



As can be immediately noticed the clusters appear overlapping one another, giving a no clear answer on how many of them there are. The corresponding average silhouette score confirms that as it results to be equal to **0.0223**.

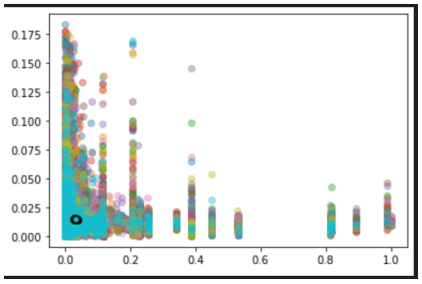
A second run of the algorithm has been performed but changing the previous mentioned parameter to "Xi". We expected a better performance as a result of this change but the results obtained were not significantly better scoring a silhouette score of **0.0225**, meaning that the results were basically the same even after implementing a different method for the clustering. The following image reports the graphical results of the clustering.



4.2.2 X-Means

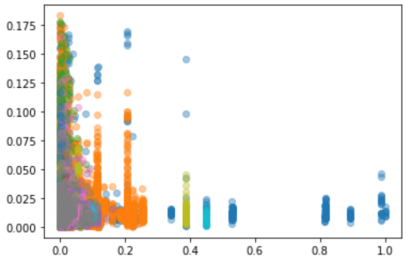
The second algorithm implemented to perform advanced clustering of the data is X-Means. The Python library used is PyClustering.
X-means clustering is a variation of k-means clustering that refines cluster assignments by repeatedly attempting subdivision, and keeping the best resulting splits, until a criterion such as the Akaike information criterion (AIC) or Bayesian information criterion (BIC) is reached.
Given the complexity and the high presence of noisy observations inside the dataset this algorithm is, perhaps, not the most fitting choice to perform clustering. Even so it has been decided to implement it in order to test if this assumption was correct.
The algorithm has been executed setting as minimal number of clusters the value 5 and as maximum number of k, 50. As splitting criterion the BIC (Bayes Information Criterion) has been

the final choice.
The final results were, as expected, not very significant since the clusters were overlapping and not well defined as can be seen in the following image.



4.2.3 Comparing Clustering Algorithms

In this short section we report the performance of the OPTIC algorithm in comparison to the "base" DBSCAN model. We expect a better performance of the OPTIC over the latter mainly because of the type of dataset we are dealing with and the ability of the OPTIC to choose its parameters.
In the following image it is possible to see the results of the clustering performed with DBSCAN with parameters set to *eps = 0.05* and *min samples = 8*.



It is possible to see already that the DBSCAN wasn't able to discern the clusters in a clear way, to support this finding an *average Silhouette score* has been computed resulting in the value of **0.0111537**.

Below a table to summarize the Silhouette scores of the two compared model.

Model	Silhouette Score
DBSCAN	0.0112
OPTIC	0.0223

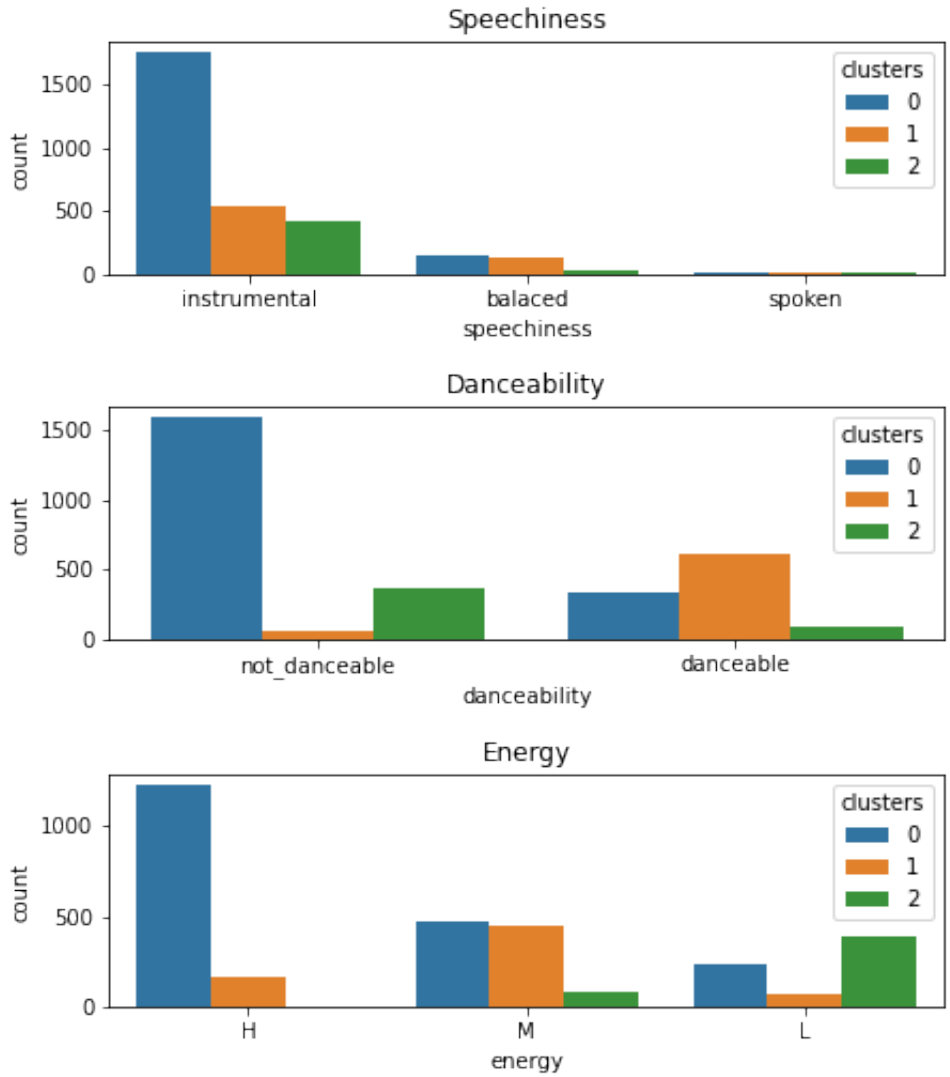
4.3 Transactional Clustering

For this task, *tracks.csv* and *echonest.csv* were used. Since transactional clustering employes categorical data, existing features were used and some additional were created by discretizing float features considering their distribution. Specifically, discrete features were created for danceability,

energy, listeners, liveness, speechiness. The final dataset consisted of **8** features and **3074** instances.
To determine the optimal number of clusters (considering that clustering is an unsupervised problem), several trials were performed among which the most interesting could be found below. The data were partitioned into **3** clusters using **Huang** initialization repeated **50** times. Interestingly, centroids captured different genres *(Electronic, Folk, Rock)*.

Cluster	**Centroid**
0	'en', 'Brooklyn, NY', 'high_listens', 'Rock', 'not_danceable','instrumental', 'studio_version', 'H'
1	'en', 'Baltimore, MD', 'high_listens', 'Electronic', 'danceable','instrumental', 'studio_version', 'M'
2	'en', 'New York, NY', 'high_listens', 'Folk', 'not_danceable','instrumental', 'studio_version', 'L'

Rock tracks are described as not danceable as they have high energy. In contrary, *Folk* tracks are not danceable because they appear to be slow and with low energy.



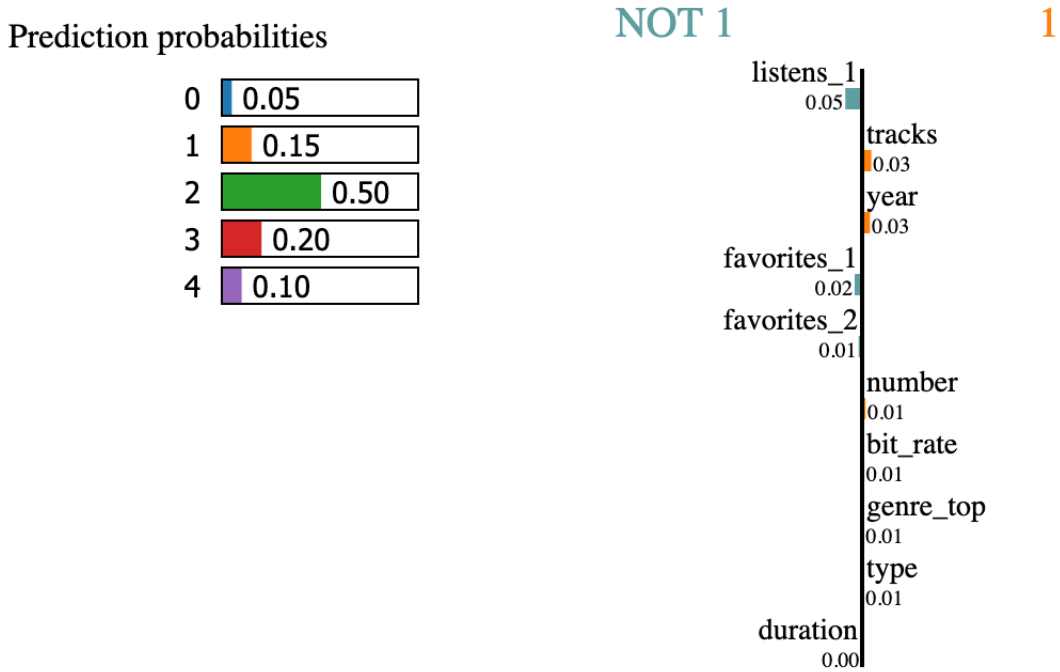
5 Module 5

5.1 Explainability

One of the main issues when implementing a machine learning model is to understand how the predictions are made, in other words how each feature affects the final result of the classification. In order to clarify the results obtained in the previous task involving classification different explainability techniques have been peformed on a black box model based on the Random Forest mentioned in Module 3.

5.1.1 LIME

Lime is short for Local Interpretable Model-Agnostic Explanations. Each part of the name reflects something that we desire in explanations. Local refers to local fidelity - i.e., we want the explanation to really reflect the behaviour of the classifier "around" the instance being predicted. This explanation is useless unless it is interpretable - that is, unless a human can make sense of it. Lime is able to explain any model without needing to 'peak' into it, so it is defined model-agnostic.

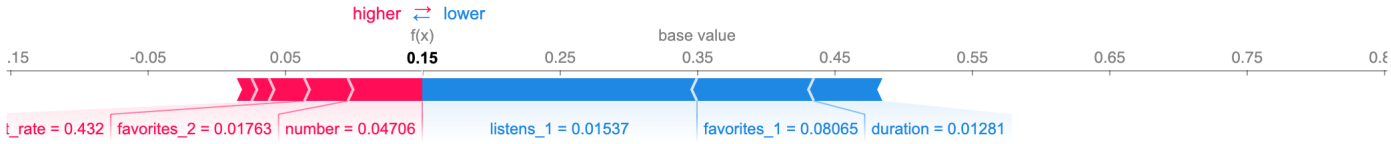


In the picture above it is possible to verify that the most important feature in the classification is the variable *listens 1*; it is also possible to see that the class that has the higher probability of being predicted is the class with value 2.

5.1.2 SHAP

The second method applied is SHAP.
As defined by the authors SHAP (SHapley Additive exPlanations) is a game theoretic approach to explain the output of any machine learning model. It connects optimal credit allocation with local explanations using the classic Shapley values from game theory and their related extensions. In the following image it is possible to see the feature importance visualized through a *force plot*, the

features represented in red push the prediction higher while blue ones push the prediction down.



It is also possible to put together the force plot explanations, perform a tranformation on the plane and see the explanations for a whole dataset. This second visualization can be seen in the following image: