RAVDESS Analysis

Data Mining Project

Sandro Cantasano Martino, 660874
Alma Stira Martinez, 658699
Benedetta Amalia Iannolo, 658691

Contents

1 Introduction

The objective of this paper is to analyze "Ravdess", a dataset containing information about an American study in the field of Neuroscience. The goal of the analysis is the classification, through appropriate techniques, of different emotions leaked from the sounds of audio recordings performed by the actors examined. The target taken into consideration for the classification instance will therefore be the variable "Emotion" which contains eight different types of emotions. Analyses are also carried out on the variable "Vocal _channel" both in the experiment of reduction of the dimensionality and in that conducted with an unbalanced dataset(section 6). In addition, we will try to predict using "mfcc _mean _w3" with advanced regressors.

2 Data Understanding e Preparation

The inspection about the presence of emphMissing Value from positive outcomes, the dataset turns out to be devoid of it.
Always holding "Emotion" as the target variable, we use textbfSelectKBest algorithm to select the 10 features with higher ANOVA value. As you can see in Figure 1, all features have outliers, especially in emph"lag1 skew w3". We will then use Anomaly Detection algorithms to define the best strategy for dealing with these records.

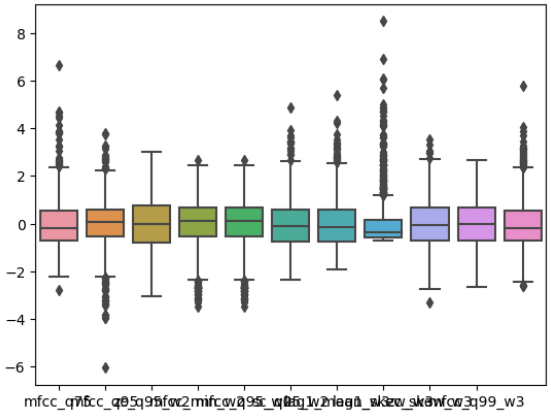

Figure 1: Box Plot

2.1 Data Preparation

RAVDESS consists of 434 features, 9 of which are categorical, the remaining are numerical. In the first instance, we go to delete the features "filename" and "actor" because they are considered not very useful for the analysis. The remaining ones are codified through the methodology textbf"LabelEncoder" according to which, given the variable to modify, the afferent classes to it will be replaced with integer values between 0 and the total cardinality of the variable taken in consideration, less 1.

The numerical variables are instead taken using the methodology textbf"StandardScaler" that subtracts to the specific instance the average of the values of the instance and divides everything by its standard deviation, so as to make all the data contained in our dataset of a textit textbfuniform scale and optimize the computing capacity of the algorithms that we will use later.

In addition, features with constant values are deleted to simplify analysis.

2.2 Matrice di Correlazione

In order to prevent the **Multicollinearità**phenomenon, the correlation study between the remaining variables (381 features) is implemented using the Pearson index. It is decided to eliminate variables with an absolute correlation of more than 85 This reduces the dataset to 163 features.

3 Basic Classifier

This section explains the classification algorithms and methodologies used to predict the variable "Vocal Channel". Where possible, a validation phase is implemented to select the best hyperparameters that maximize accuracy. In this phase, we use the algorithm "GrisSearch" and "StratifiedKFold" to determine the best parameters of the single classifier. The idea is to test a Base Classifier, gain the relatives performances, take another test on it using a dimensionality reduction technique in order to analyze the variations of the performances, where there are.

3.1 K- Nearest Neighbour

To find the best k, a Grid Search with values between 30 and 80 is implemented. The best result is k=40 which is the parameter that minimizes the average error as shown in figure 3. The graph on the left shows how the learning curve obtained by the classifier does not present overfitting problems/ underfitting since the performance obtained between training and test sets tend to coincide.

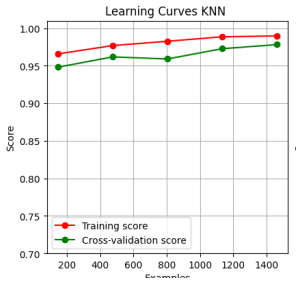

Figure 2: k-NN - learning curve

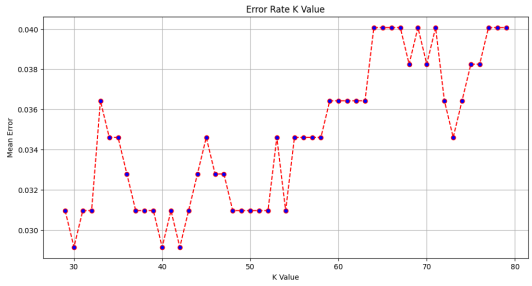

Figure 3: Select best K

The results obtained by the classifier are in Figure 4. The ROC curve demonstrates how the classifier makes an excellent distinction between classes, presenting an AUC of 0.997, whereas the Lift curve shows that the classifier is almost twice as good as the random classifier for 40 % of forecasts.

Accuracy	0.96
Precision	0.96
Recall	0.96
Specificity	0.97
F1	0.96

Figure 4: Performances

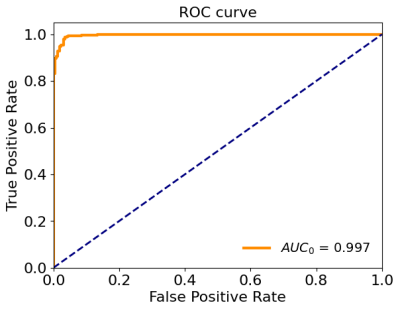

Figure 5: k-NN - ROC curve

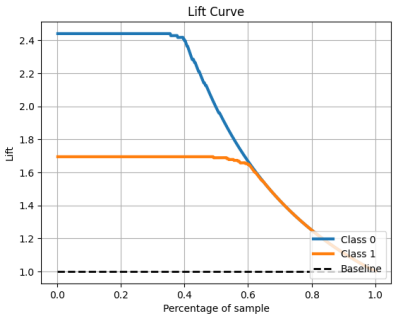

Figure 6: k-NN - Lift curve

3.2 Decision Tree

In the preliminary phase of the classifier the GridSearch allows us to optimize some parameters as follows: "criterion"="entropy", max depth = 15, min samples= 1 and min sample split = 5. These parameters allow the classifier to reach a performance of 94.35The lift curve shows how the Decision Tree based classifier improves the random classifier.

Accuracy	0.95
Precision	0.94
Recall	0.95
Specificity	0.94
F1	0.95

Figure 7: Performances

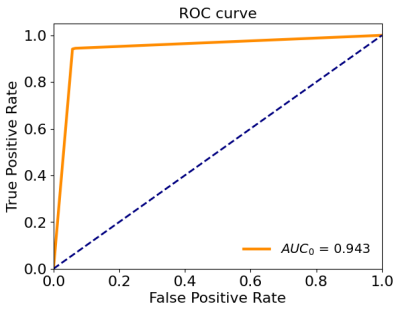

Figure 8: Decision Tree - ROC curve

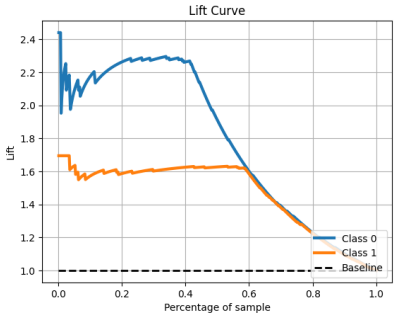

Figure 9: Decision Tree - Lift curve

3.3 Logistic Regression

Logistic regression is a probabilistic classification algorithm used to predict a binary discrete output variable. Specifically, we intend to identify a function that can map " textbfVocal $_c$*hannel*"

The algorithm is implemented with the following hyperparameters: beginitemize item textbfC= 1.0 – ¿ parameter used to penalise incorrectly classified examples;

item textbfSolve = 'newton-cholesky'– ¿ is the optimization algorithm used, the best in this case since we have more samples than features in the dataset; item textbftol= 0.0001 – ¿ parameter used to find the stop criterion for the optimization algorithm; enditemize We obtain a coefficient equal to 0.55 and an intercept of 0.39 and in the figures that follow the performance values are reported:

Accuracy	0.70
Precision	0.71
Recall	0.66
Specificity	0.69
F1-score	0.69

Figure 10: Performances

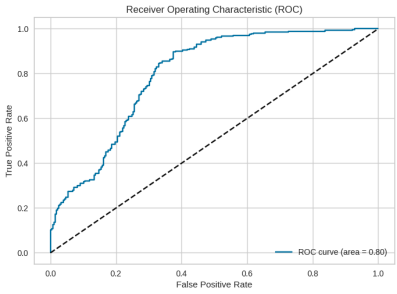

Figure 11: Logistic Regressor - ROC curve

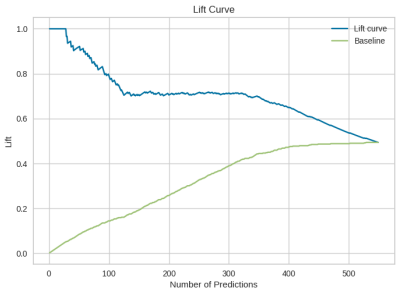

Figure 12: Logistic Regressor - Lift curve

3.4 Comparison

To conclude, all methods are performing well, exceeding the random classifier. Outstanding among them is knn performance with 96% accuracy.

4 Dimensionality Reduction

Dimensionality reduction techniques is a technique used to reduce the number of variables in the dataset while maintaining the most significant information. They are divided into two categories: textbffeature selection and textbffeature projection. The following sub-sections show the tests performed.

4.1 Feature Selection

In this approach the **b**asic classifiersk-NN, **Decision Tree** and *textbfLogistic Regression* are tested using textit textbfVariance Threshold and it **Univariate Feature Selection**. These algorithms were trained using all the attributes present in the dataset and, later, their performance was evaluated on the test set.

4.1.1 Variance Threshold

The goal is to eliminate features that have low variance, so limited variation within the dataset, so removing these features can simplify data analysis and reduce the noise or complexity of later models. The dimensionality reduction technique is implemented and performance is then evaluated over a range of variance values ranging from [0.02, 0.5]. In the implementation with Decision tree, with the increase of Threshold are eliminating from 1 to 4 variables recording an increase in performance with Threshold equal to 0.5, for an increase, albeit minimal, to 95.08% of accuracy.

The implementation of the same on k-NN results in an increase of a percentage value of the accuracy metric, going to eliminate again in this case from 1 to 4 variables.

The implementation of the same on Logistic Regression is the most exciting result, with this method we pass from an accuracy of 40 % as shown in the previous section, to an accuracy of 96 %-97 % with an alteration in the cardinality of the features used identical to the previous cases.

4.1.2 Univariate Feature Selection

In the reduction of the dimensionality, in this case, the selection of univariate features considers each feature independently, evaluating its relationship with the desired output or target using the ANOVA test or the chi-framework. Values from 20 to 70 are evaluated.

For k-NN, almost all the selected indices increase the performance even if slightly, showing it in a range between 0.96 and 0.97, therefore a slight improvement.

For Decision tree the method does not perform well at least for the defined range, decreasing by two percentage points in accuracy.

For Logistic Regression the method allows to obtain an optimal improvement of performance as in the case of Variance Threshold, allowing to reach performance from 0.96 to 0.98 percentage points.

4.1.3 Recursive Features Elimination

The ***Recursive features Elimination*** algorithm for Decision Tree and Logistic Regression is also tested, confirming the improved performance of Logistic Regression (0.96), common to previous experiments, and 0.95 for Decision Tree. A variation of features is reported up to 13 for Decision Tree and 66 for Logistic Regression

4.2 Feature Projection: Principal Component Analysis

The PCA method was used to reduce the dataset size to 26 dimensions, so as to capture the greatest number of variances.
This allows us to bring the performance to:

- k-NN – 96 %
- Decision Tree – 84 %

- Logistic Regressor – 97 %

4.3 Comparison

Below are plotted decision boundaries using the PCA 2-dimensional.

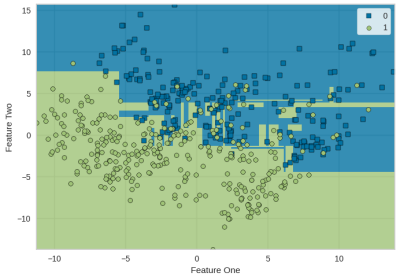

Figure 13: Decision Tree - Decision Boundaries

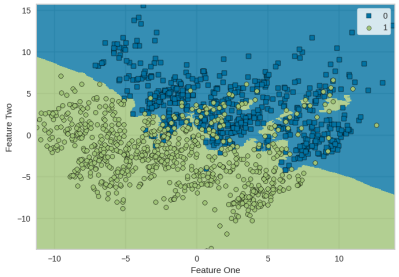

Figure 14: k-NN - Decision Boundaries

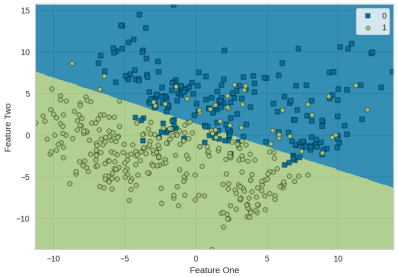

Figure 15: LogisticRegression - Decision Boundaries

5 Imbalance Learning

In this section we explore the possibility that the target variable that we consider for our analysis is unbalanced, meaning that the number of records pertaining to a class is different from the number of records related to the different class. This discrepancy can make classification operations more difficult, so we use the following techniques to solve the problem. In our case, we choose to conduct the analysis on the feature *"Vocal _channel"* which presents a class distribution equal to:

- 748 record appartenenti alla classe "Normale";
- 1080 appartenenti alla classe "Strong".

An experiment is conducted: we unbalance the class in order to obtain a percentage of 97.3 % for class "Normal" and 3.7 % for class "Strong" (40 vs 1080). We perform the analysis in two different directions, implementing a decision tree both with the technique of textit emphUndersampling of the majority class, that emphOversampling of the minority class. Trying to improve the performance of the classifier we implement a search for the best features on which to start the classification using an emphRecursive Feature Elimination in a range of values ranging from 10 to 100 selectable features.
We select the result that allows us to maximize the accuracy as shown in the figure.

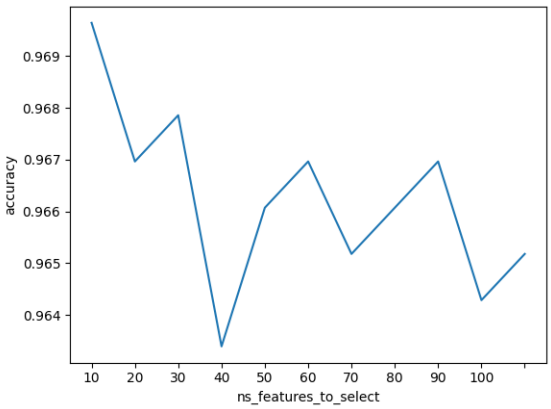

Figure 16: Recursive Feature Elimination

We select 10 features as it is the value for which you get greater accuracy. In fact this experiment does not allow us to improve the state of performance much, but we obtain important results on the metric of "Area Under the Curve" outlining a situation as shown in the figure:

	Accuracy	**F1-score**	**Recall**	**Precision**	**AUC**
DT	0.97	0.83	0.85	0.81	0.848
DT + RFE	0.97	0.80	0.85	0.77	0.921

Table 1: Performances

Since accuracy measures how well the decision tree correctly classifies instances as a whole, while the AUC defines the classifier's ability to correctly discriminate different classes, we take the choice to continue the analysis using the result derived from the operations of reduction of dimensionality.

5.1 UnderSampling

Undersampling is a technique that reduces the number of majority class samples to match the number of minority class samples, in order to improve the performance of the classification algorithms applied later. The algorithms tested are **Random UnderSampler**, **CondesedNearestNeighbour** and **Tomek Links**. we get the following results:

	Accuracy	**F1-score**	**Recall**	**Precision**	**AUC**
RUS	0.88	0.66	0.94	0.62	0.921
CNN	0.94	0.73	0.87	0.68	0.891
T-L	0.98	0.84	0.85	0.83	0.884

Table 2: Undersampling Performances

As you can see, the three algorithms perform in substantially different ways, but in relation to what was explained above, we believe it is appropriate to favor the metric of AUC as primary in the definition of which among these algorithms is better. In fact, we get the best result in the case of textitRandomUnderSampler therefore better class discrimination. The other two are similar according to the AUC metric, although they are quite different, especially in precision.

5.2 Oversampling

Oversampling is a technique that increases the number of samples in the minority class to match the number of samples in the majority class. The goal is to improve subsequent classification algorithms and prevent under-representation. The algorithms **ADASYN**, **RandomOverSampler** and **SMOTE**are tested. otteniamo i seguenti risultati:

	Accuracy	**F1-score**	**Recall**	**Precision**	**AUC**
ADASYN	0.99	0.93	0.96	0.90	0.941
ROS	0.98	0.86	0.92	0.81	0.902
SMOTE	0.97	0.84	0.92	0.78	0.901

Table 3: Oversampling Performances

As previously defined, the AUC is taken "primary metric", as you can see ADASYN performs better also obtaining excellent performance in subsequent indicators. The results obtained with the other two algorithms tested are optimal and in general, we note that the results of this section are significantly different from those detected with the undersampling techniques,

5.3 Combinations

In an attempt to optimize the performance and to lose the least amount of useful information from the sampling practices mentioned above we have conducted an experiment of combination between models of Undersampling and oversampling. Specifically **SMOTE + Edited Nearest Neighbor** and **SMOTE + TomekLinks**

	Accuracy	**F1-score**	**Recall**	**Precision**	**AUC**
SMOTE + ENN	0.98	0.90	0.92	0.87	0.922
SMOTE + T-L	0.98	0.88	0.92	0.85	0.921

Table 4: Combination Performances

In this case, the techniques perform well, to an almost comparable extent, do not perform better than the Oversampling techniques but since these techniques are a combination of the two types of algorithms previously exposed, they allow to obtain results *on average* better avoiding the danger of losing important information in the case of undersampling, and incurs overfitting in the case of oversampling.

5.4 Comparison

The following are visualizations conducted with the technique of Reduction of the dimensionality **PCA** of the three algorithms considered best:

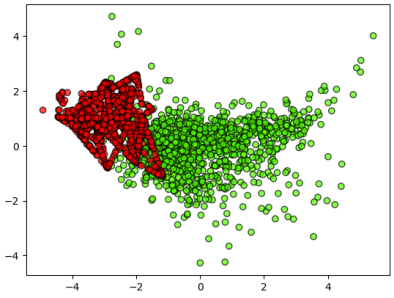

Figure 17: Adasyn

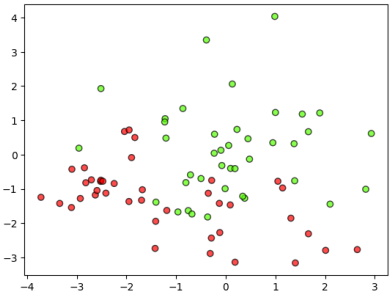

Figure 18: RUS

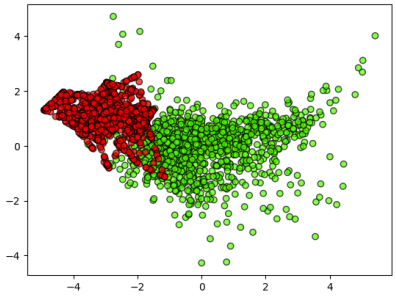

Figure 19: sMOTE + T-L

6 Anomaly detection

For the purpose of analyzing advanced classification techniques in the following sections, we perform anomaly analysis on the target variable **Emotion**. As seen in Figure 1, the dataset contains many outliers.

6.1 DEPTH-BASED APPROACHES

The first algorithm analyzed was **Elliptic Envelope**, belonging to the family " *Depth-based*". This algorithm is based on the assumption that data from a normal population follows an elliptical distribution in space. Then an ellipse is constructed that encloses most of the data and considers the points outside as anomalies.
For the correct functioning of this algorithm, in order to preserve the distances in the original data structure but ensure a visualization, we first perform a reduction of the 2-component dimensionality using ***T-SNE***. This is a dimensionality reduction algorithm that tries to maintain the proximity relationships between the points in the original dataset in low-dimensional visualization. By running the algorithm, 183 anomalies are detected.

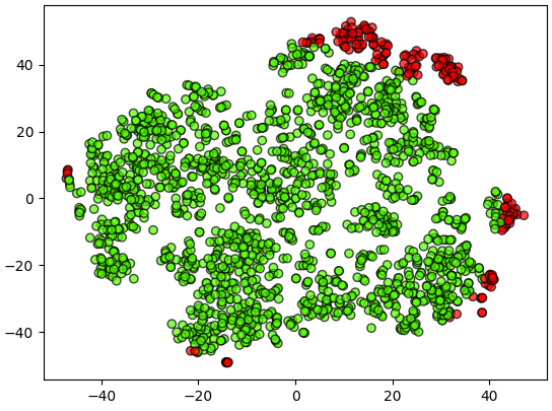

Figure 20: T-SNE

6.2 DISTANCE-BASED APPROACHES

6.3 PROXIMITY-BASED APPROACHES

The last algorithm presented is **CBLOF (Clustering-Based Local Outlier Functor)**, an algorithm belonging to the **Proximity-Based** family that combines the clustering methodology with the detection of anomalies. Then clusters are formed (with K-Means or DBSCAN) and then the local outlier factor is calculated on a data point relative to its neighbors in the cluster. Thus outlier points with higher LOF.
The algorithm allowed us to identify 183 outliers with an average value of 20,203 for a Threshold of 14,371.

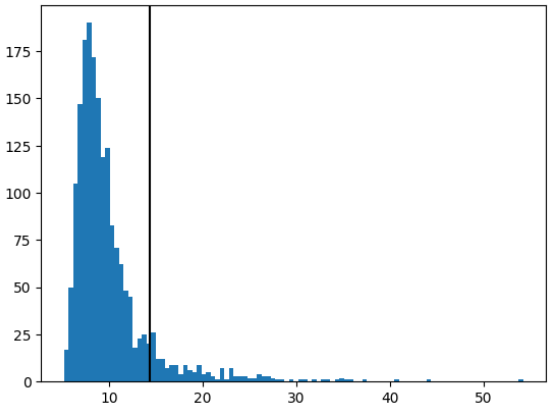

Figure 21: Hinstogram

6.4 Dealing with outliers.

You can see from the images below the records defined as outliers by the different algorithms previously illustrated.

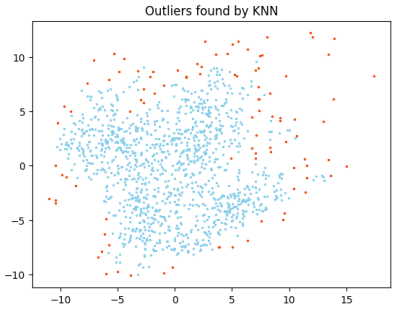

Figure 22: Outliers from KNN

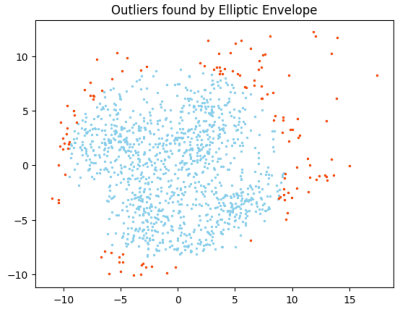

Figure 23: Outliers from Elliptic Envelope

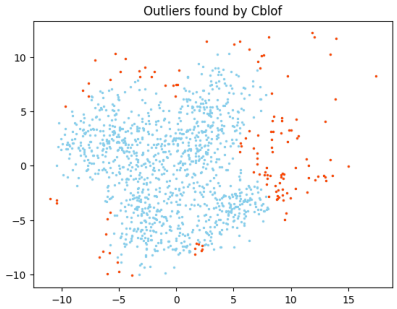

Figure 24: Outliers from CBLOF

Of the three algorithms used, we decided to find the common records identified as outliers and remove them. In the figures below we notice first the records in common between the algorithms and then the visualization of the points in the dataset without outliers.

Figure 25: Outliers in common

Figure 26: Outliers removed

7 Advanced Classifiers

This section describes the use of advanced classifiers for predicting the variable "Emotion". In the first instance, we experience a cross-search of hyperparameters via GridSearch or RandomizedSearch. The hyperparameters tested are shown in the figure.

Linear SVM	Non-Linear SVM
'C': [0.001, 0.01, 0.1, 1.0, 10.0, 50.0, 100.0] *'tol'*: [1e-4, 1e-5, 1e-6] *'penalty'*: ['l1', 'l2']	*'C'*: [0.1, 1, 10] *'tol'*: [1e-4, 1e-5, 1e-6] *'kernel'*: ['sigmoid' ,'rbf', 'poly'] *'degree'*: [2, 3, 4]
Multilayer Perceptron	**Random Forest**
'hidden_layer_sizes': [(12, 23, 11), (23, 43, 32), (128, 64, 32)] *'activation'*: ['logistic', 'relu'] *'learning_rate'*: ['constant', 'invscaling', 'adaptive'] *'tol'*: [1e-2, 1e-3, 1e-4]	*'criterion'* : ['gini', 'entropy'] *'max_depth'*: [None, 10, 11, 15, 20] *'min_samples_split'*: [2, 5, 10, 20] *'min_samples_leaf'*: [1, 5, 10, 20] *'n_estimators'*: [25, 50, 75]
Bagging	**Adaboost**
'max_features' : [1, 2, 3, 4] *'max_samples'* : [0.05, 0.02, 0.01, 0.1, 0.2, 0.5] *'n_estimators':* [1, 5, 10, 25, 50, 100]	*'n_estimators'*: [5, 10, 25, 50, 100] *'learning_rate' :* [0.1, 0.25, 0.5, 0.75, 1] *'n_estimators'*: [1,10,15,50,100]

Figure 27: Hyper parameter tuning

However, unlike the cases analyzed in the previous sections, we tried to optimize the F1 score, because the target in this case is multi-class and no longer binary.

7.1 Support Vector Machine

SVM is a model that allows you to classify data both linearly separable and not. In the latter case, which turns out to be closer to reality, it is possible to transform the input data into a space separable through a set of functions Kernel. Both the kernel and the adjustment parameters can cause overfitting, which is why the selection of these parameters has been carefully defined.

7.1.1 Linear SVM

Linear SVM is a type of binary classification model that uses a linear hyperplane to separate observations from two different classes. The goal of a linear SVM is to find the best hyperplane that maximizes the margin between the two classes in the dataset. The best parameters are selected from:

- C = adjustment parameter;
- tol = tolerance to define a stop criterion;
- penalty = Specifies the standard used in the penalty

We get the following result from GridSearch: textbfC = 0.1, tol = 1e-06, penalty= "l2". We get the performance equal to 0.46 for the Linear SVM and the parameters we have tested are those listed above because with "C" we go to define a parameter of regularization very important to prevent overfitting. In fact, the performance on the training set and the test set are to avoid this scenario. We get an F1 score of 0.47 with particular flaws on class 4 and class 6, although the first class gets important results.

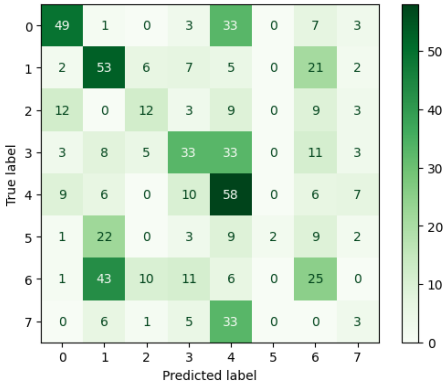

Figure 28: Confusion Matrix

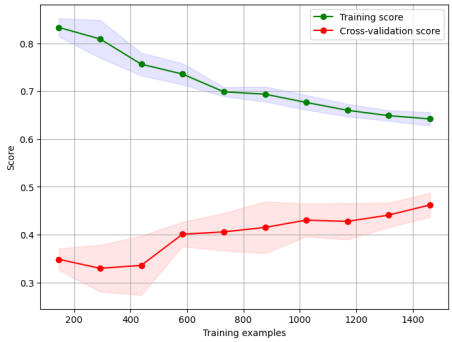

Figure 29: Linear SVC - Learning curve

7.1.2 Non-Linear SVM

In the non-linear Support Vector Machine, we test different kernel functions (as well as other reference parameters). These functions allow to transform the distribution of data from two dimensions to large so as to be able to generate a distribution in space such that the algorithm can better define the decision boundaries. The "sigmoid", "rbf" and "poly" functions are tested and what "sigmoid" does better on our data. As for the linear model, we test diffent parameters of "C" and "tol" obtaining for these a performance in terms of F1 score equal to 0.39. We go to evaluate the presence of Overfitting and Underfitting and in fact it shows that the model is affected by the first. Then two routes are tried:

- I try to simplify the model, making a selection of features through the **Select KBest** dimensionality reduction technique;
- tries to complicate the model by oversampling the classes of the target variable then adding dummy data, in a sense;

In both cases, the error on the training and test set converges to about 30 % the problem is solved.

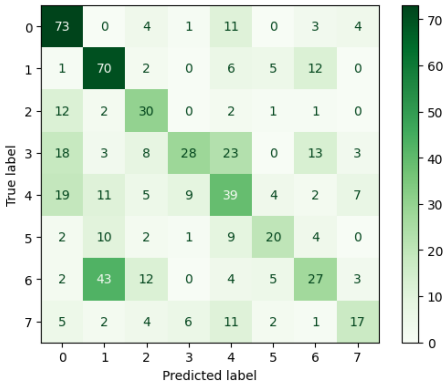

Figure 30: Confusion Matrix

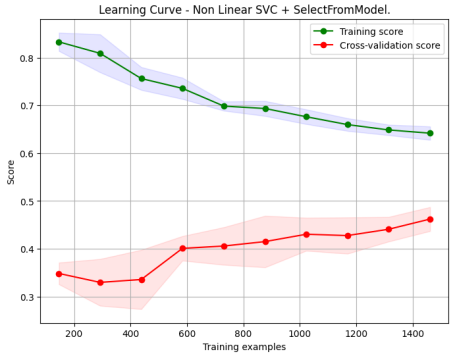

Figure 31: Non Linear SVC - Learning curve

7.1.3 Results

Overall, therefore, the two models are almost identical at least in compliance with the F1-score metric.

	Accuracy	Precision	Recall	F1
Linear SVM	0.49	0.48	0.49	0.47
Non Linear SVM + Hyperparameter Tuning	0.49	0.50	0.48	0.47
Non Linear SVM + H.T. + Select K Best	0.48	0.47	0.49	0.47
Non Linear SVM + H.T. + OverSampling	0.48	0.50	0.48	0.47

Figure 32: Combination Performances

7.2 Neural Network

7.2.1 MultiLayer Perceptron

Since Neural Network is particularly subject to overfitting, we go to optimize appropriate adjustment parameters through GridSearch. The algorithm is also tested with undersampling and oversampling techniques. In figure we report the hyper parameters tested and subsequent performance:

	Parameter	Accuracy	Precision	Recall	F1
Multilayer Perceptron	{'activation': 'relu', 'hidden_layer_sizes': (128, 64, 32), 'learning_rate': 'adaptive', 'tol': 0.0001}	0.42	0.40	0.40	0.40
Multilayer Perceptron+Undersampling	{'activation': 'logistic', 'hidden_layer_sizes': (128, 64, 32), 'learning_rate': 'constant', 'momentum': 0.2, 'tol': 0.001}	0.45	0.43	0.46	0.44
Multilayer Perceptron+Oversampling	{'activation': 'relu', 'hidden_layer_sizes': (128, 64, 32), 'learning_rate': 'adaptive', 'momentum': 0.4, 'tol': 0.001}	0.47	0.45	0.46	0.45

Figure 33: Hyper parameter tuning

which are understood as:

- **Hidden layer** = represents the architecture of the network, as the various layers are formed;
- **activation** = the activation function for individual layers;
- **learning rate** = widely used parameter for overfitting prevention;
- **early stopping** = mode to block the network operation mechanism;

The Perceptron multilayer appears to be overfitting in all case studies despite the adjustments performed.

7.2.2 Deep Neural Network

Several sequential neural models have been defined that differ according to the number of intermediate levels (all of Dense type) and units dedicated to each of them. In particular different combinations were tried before defining the following structures that share the usual output layer at 8(number of emotion classes) units, all trained with 300 epochs:

- **Model 1** = Input layer(512-tanh) + 4*Hidden Layer(64-relu);
- **Model 2** = Input layer(512-tanh) + 1*Hidden layer(512-tanh)
- **Model 3** = Input layer(512-relu) + 1*Hidden layer(num.features-relu)

Several sequential neural models have been defined that differ according to the number of intermediate levels (all of Dense type) and units dedicated to each of them. In particular different combinations were tried before defining the following structures that share the usual output layer at 8(number of emotion classes) units:

	Accuracy	F1-score
Model 1	50%	49%
Model 2	52%	50%
Model 3	52%	51%

Figure 34: Performances

As Loss function was used "sparse _categorical _crossentropy" for our case multiclass and "adam" as net weight optimizer. The following results are visible obtained on the test set where the models 2 and 3 performed better:

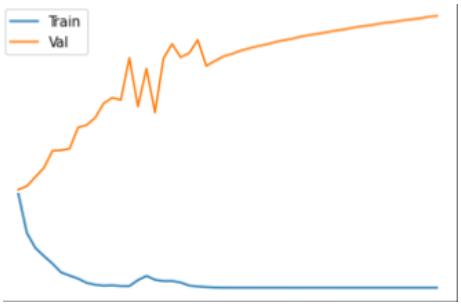

Figure 35: Loss Curve

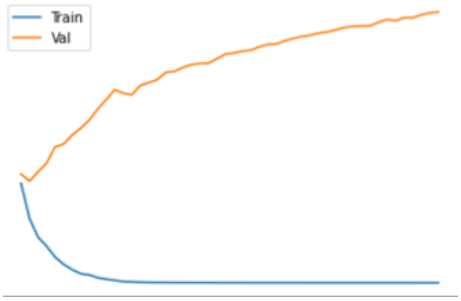

Figure 36: Loss Curve

It has been noted that in general, as in the case of Perceptron, the complexity of the network did not bring benefits but rather worsened performance on data not observed and especially the discrepancy already clear between the curve loss of the train and the validation for the various models. In fact, As can be seen from the graphs below of the best defined models (3-4), neural networks suffer from overfitting.

To resolve this problem, a level of Dropout=0.05 before the output layer and the regularization coefficients=0.04 have been inserted L2 for the input layer and the hidden one. The correct number of training examples, identified by the batch parameter _size, on which to train the network for each era, by selecting a value of 400. In the following the results after the changes made:

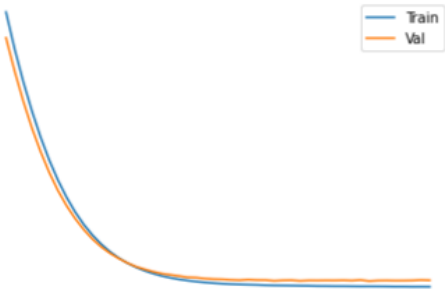

Figure 37: Loss Curve

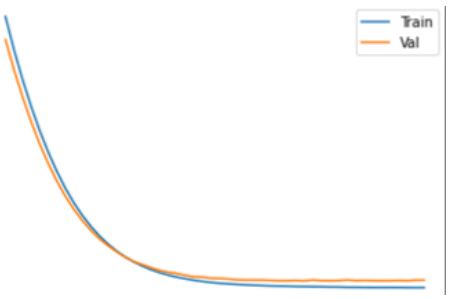

Figure 38: Loss Curve

7.3 Ensemble Methods

In this section we analyze the ensemble models Random Forest, Adaboost and Bagging. The Adaboost model has been tested using Decision Tree as a base estimator as this obtained the worst performance among the base estimari analyzed in the previous sections. Then **two experiments are conducted, both with "Vocal channel" target to appreciate the improvement of performance in the base estimator, and with "Emotion" to make a subsequent comparison between all advanced estimators** about a multi class classification task.

In the first case we focus our attention on the accuracy metric while on the second, as previously specified, we observe more carefully F1 score.

In the analysis of the variable "Vocal Channel", Random forest improves of some percentage point the prediction of the variable if associated with Undersampling attesting the score of accuracy to 99 % outstanding in the performances obtained from the various combination experiments, although Bagging and Adaboost also improve performance in the range of one or two percentage points [0.96 - 0.98 of accuracy].

In the analysis of the variable "Emotion", the ensambles get excellent performances (in the order of 0.75 of F1 score). Below are the results obtained by each:

	Accuracy	Precision	Recall	F1
Random Forest	0.66	0.64	0.66	0.64
Bagging + Decision Tree	0.57	0.55	0.57	0.55
Adaboost + Decision Tree	0.54	0.52	0.54	0.52

Figure 39: Performances

The results obtained were achieved by implementing a search for hyper parameters as shown in section 6, which allowed us to achieve interesting results, certainly prominent up to this point. The hyperparameters analyzed in Random Forest are as follows:

- **n _estimators** = the number of trees to be trained in the forest;
- **criterion** = represents the function that measures the quality of splitting;
- **min _samples _split, min _samples _leaf and max _depth** = characteristics of the Decision Tree;

They follow the confusion matrix of the Random Forest and the relative Leaning curves from which, as you can see, is overfitting. Despite all the attempts made, we have not been able to remedy the problem.

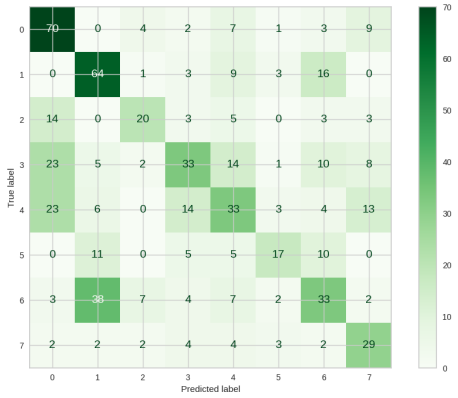

Figure 40: Confusion Matrix

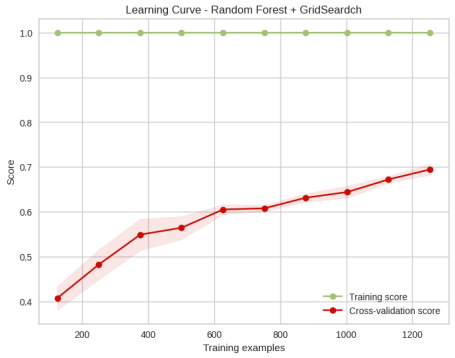

Figure 41: Random Forest - Learning curve

The figure above depicts the performance of Random Forest as it has allowed us to achieve very important results. For Bagging and Adaboost the performances were not as good so we decided to omit it from the discussion. It is also tested the combination between these algorithms previously exposed with the algorithms of Imbalanced learning, in table the results:

	Accuracy	Precision	Recall	F1
Random Forest+Oversampling	0.73	0.73	0.73	0.72
Random Forest+Undersampling	0.72	0.72	0.72	0.70

Figure 42: Combination Performances

7.4 Gradient Boosting Machines

This section analyzes the Gradient Boosting Machines algorithm, a machine learning algorithm that allows, with a set of decision trees, to perform a classification in which the trees affect the results of the subsequent constructed by the model. Specifically, the parameters that we have given more attention to are:

- the " **loss function**", is used to measure the discrepancy between the model values and the actual values of the training set. The goal of the GBM algorithm is to minimize this discrepancy, iteratively improving the model through the boosting process.
- the " **learning rate**", the learning rate reduces the contribution of each tree through learning. There is a compromise between the learning rate and the estimators.
- the " **number of estimators**": the number of boosting steps to be performed;
- the " **subsample**", understood as the fraction of samples to be used for the assembly of individual basic students.
- the " **tol**", understood as the tolerance for early stop, value linked to the parameter "n _iter _no _change" which is used to determine whether "early stopping" will be used.

So by virtue of the absence of Overfitting and Underfitting we leave the values of tolerance and "n _iter _no _change" as by default and go to evaluate the general performance.

The best parameters are searched using the technique of RandomizedSearch, tested the algorithm with these and analyzed the same to verify the presence of overfitting of the model.
The following parameters are then analysed:

	Parameter
Gradient Boosting Classifier	*'loss'* : ["log_loss"] *'learning_rate'* : [0.1,0.6,1.1] *'n_estimators'*: [50, 100, 150,200] *'subsample'* : [0.5 ,1.0]}

Figure 43: Hyperparameter tuning

With the optimized parameters the classifier is able to obtain a F1 score of 0.49 on Emotion variable targets. The performances are almost unchanged if not slightly regressed in the combination experiment with the techniques of Undersampling and Oversampling.

7.4.1 Ulteriori esperimenti

The **XG Boost** and **Light GB** algorithms are also tested:

- XGBoost is a very powerful machine learning algorithm that can be used to address a wide range of classification and regressionXGBoost has several features that make it particularly effective, including:
  1. A customizable loss function that can be used for classification and regression problems;
  2. A regularization process that helps to prevent overfitting of the model;
  3. A search algorithm for the best hyperparameters (model parameters that are not learned during training) that helps optimize model performance.
- LightGBM is a gradient-boosting machine learning algorithm that focuses on speed and efficiency. It is designed to train large models on large datasets. LightGBM uses a leaf-growing (leaf-wise) approach instead of a level-growing (level-wise) approach like XGBoost, which allows it to achieve greater efficiency.

7.4.2 Results

The results of the three algorithms previously examined are reported, which allow us to observe how the examined algorithm that was more performing was Light Gradient both in terms of Accuracy and F1 score, although the remainder are not very different from the above.

	Accuracy	Precision	Recall	F1
Gradient Boosting Classifier	0.49	0.50	0.49	0.49
Extreme Gradient Boosting Classifier	0.47	0.46	0.49	0.46
Light Gradient Boosting Classifier	0.52	0.52	0.52	0.51

Figure 44: Performances

From the plot of the Roc curve it is possible to define which of these experiments have been more performing:

From the comparison of the Roc Curves but specifically the observance of the value of AUC, the algorithm **Light GBM** with a coefficient AUC=0.88 is the best among the examined.

7.5 Comparison

The experiments conducted allow us first of all to say that all algorithms improve the random classifier although some of them do not stand for clarity. The best result is**Random Forest** that in terms of F1 score returns the best results (in the experiment with target=Emotion).

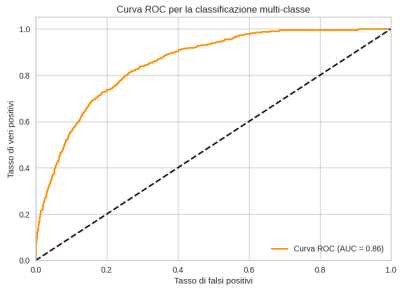

Figure 45: Roc Curve Gradient boosting

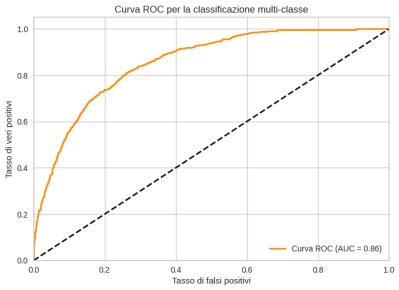

Figure 46: Roc Curve Extreme Gradient Boosting

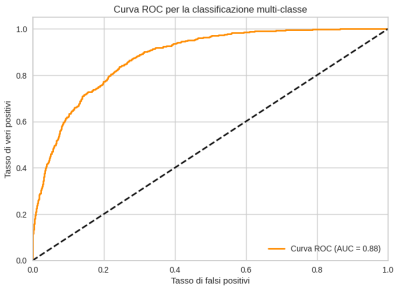

Figure 47: Roc Curve Light Gradient boosting

8 Multivariate Regression

Experiments are conducted using Random Forest Regressor, GradientBoostingRegressor, AdaboostRegressor using ***"mfcc _mean _w3"***

1. ***RandomForest Regressor*** – ¿ n _stimatori = 1000, Max _dephth = 2
2. ***Gradient Boosting Regressor***– ¿ n _estimators = 500, Max _dephth = 4, learning rate = 0.01, loss = square loss
3. ***Adaboost Regressor***-¿ n _estimators = 100, learning rate = 0.01, loss = loss squared

8.1 Results

The results of the three regressors are very good, unlike the previous analyses. We get good results in terms of R2 score, also known as coefficient of determination, is a common evaluation measure used to evaluate the performance of a regression model. Indicates how well the model adapts to the data against the variability of the data.

Model	R2	MSE	MAE
Random Forest	0.922	0.079	0.217
Gradient Boosting	0.842	0.163	0.300
AdaBoost	0.938	0.081	0.192

Figure 48: Performances

8.2 Linear Model

From the model we get **R2** information which is equal to 0.312 and **Adjusted R2** which gives us information about the model and the function used for prediction, if it deviates much from R2 it means that the model has too many variables, is too complex, so prediction can be of poor accuracy. But in our case we find a parameter of 0.311 therefore compliant and acceptable.

A linear regression model is implemented using as independent variables "**stft_sum_w3**" and "**zc_skew_w3**", and "**mfcc_q99_w3**" as dependent variable. The *statmodels* library is used, which provides a summary summary of all important information. In other words, it would be like predicting a certain frequency spectrum, taking into account stft _sum _w3 multiplied by zc _q95 _w3. Let's then calculate how many times, a given frequency spectrum mfcc crosses the zero and contributes to define in a completely **heuristic** the frequency at which an actor speaks, and on the basis of this, with appropriate and subsequent analysis(discretization of the variable mfcc _q95 _w3, detection of the most important characteristics, classification) define which are, for each actor, the main frequencies.

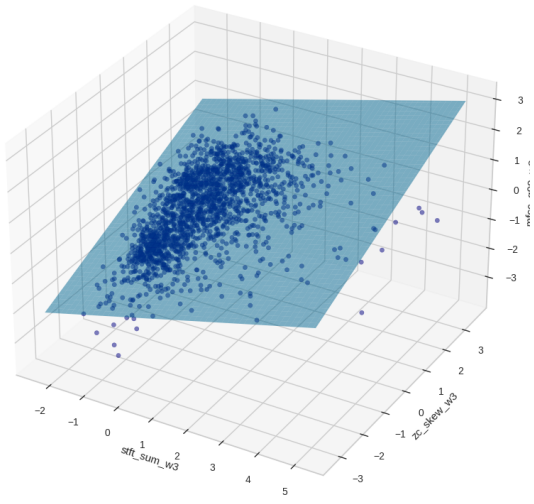

Figure 49: Iperpiano

9 Explainability

Two different experiments are conducted in such a way as to use a Global algorithm and a local one. For the first, a Basic Decision Tree with Vector Machine support is implemented. The second, local, will be the LIME algorithm.

9.1 LIME

Explainabiliry **LIME** algorithm tested on NonLinear SVM. Being LIME a Local algorithm, we have identified 4 records to analyze:

- **20**: Vocal channel(song), Emotion(disgust), Emotional intensity(normal)
- **24**: Vocal channel(song), Emotion(disgust), Emotional intensity(strong)
- **36**: Vocal channel(song), Emotion(sad), Emotional intensity(normal)
- **40**: Vocal channel(song), Emotion(sad), Emotional intensity(strong)

in all four cases the algorithm identifies as important variable for **mfcc** and **lag01** and **stft**(all three with different quantiles and associated statistical variables variants) although they also conform to the feature group that creates misclassification. The prediction obtained in the first 3 of the four cases is negative (according to the results obtained from the matrix that, for simplicity and smoothness of reading, we decided not to include in the treatment), while in the latter case the prediction is correct although if it is 50 %

9.2 Basic Decision Tree

With this method we tried to understand the logic and the process of classification through a simple decision tree with a maximum depth of 4. To do this they were followed two steps:

1. Train the SVC model, with the hyper parameters defined in the appropriate section
2. Instantiate the Decision Tree by training it on the usual train and target variable emotion data.

The results in figure:

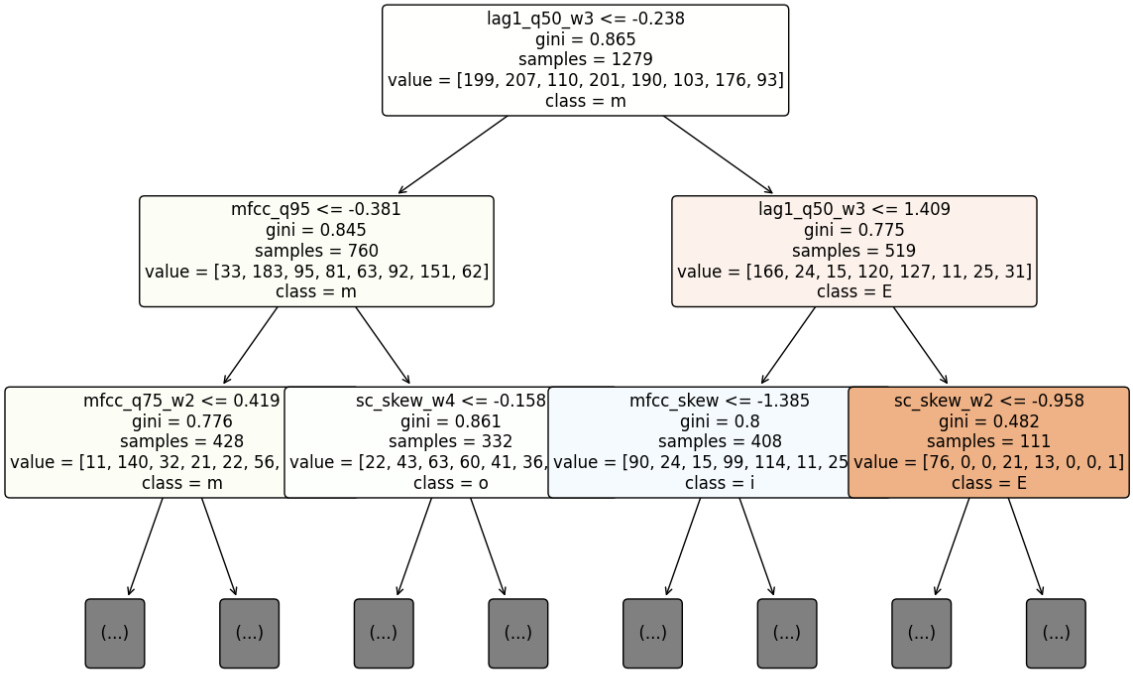

Figure 50: Decision Tree

We can see from the figure how the tree tries to approximate the logic of the MLP; this methodology has not been of great help as an analysis with a tree with greater depth. At each you can see that the variable placed in "root" is *lag1 _q50 _w3* that is also found in the next leaf. The presence of *mfcc* proves the analysis and affirms the importance of this variable in an analysis dataset like this one.

10 Time Series Analysis

10.1 Data Understanding and Preparation

The dataset of this analysis is composed of 4 different files, of which two Numpy arrays, composed by arrays containing the values of the single time series. The preprocessing intended for these two sets of data was a normalization using the *TimeSeriesScalerMinMax* from the *tslearn* library as well as a *Amplitude scaling* because it has been found that the time series had different peaks in case the one was put in comparison to another.

The other two files(.csv) include categorical features specific to the dataset used in the first part of this report. These were treated using the *LabelEncoding* technique commonly used in these cases.

Nan and **null** values are searched, so they are zero: the first ones are absent unlike the second ones. We preferred not to delete them to preserve the correctness of the analyses, in concomitance of the nature of the single time series that we are analyzing.

10.2 Motif Discords Discovery

Four time series are considered for the Motif & Discord Discovery task. We intend to analyze an actor who exercises the statement " *Kid are talking by the door*", with an emotion *calm*, with an intensity *normal*, *speaking*. What we intend to analyze is how the time series changes on the same actor (first we analyze a woman, then a man) to the variation of **repetition**(1st or 2nd).

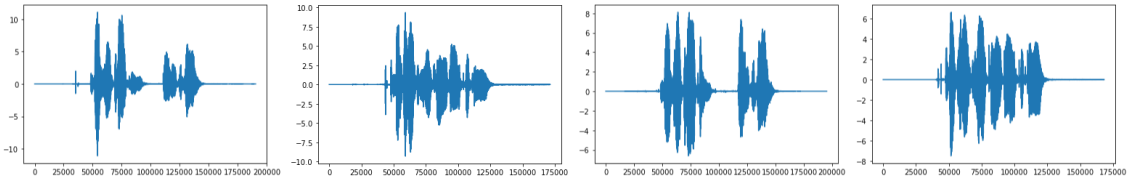

Figure 51: Vocal audio waves

We consider it interesting to propose this visualization (the first two waves are afferent to the male sex, respectively first and second repetition while the remaining two are relative to women following the same precedence). It is noted first of all that in the two sexes is shared an increase in the frequency of the waves over time as the number of repetitions increases but above all as the woman has an amplitude and a phase on average larger than the man. We could, from this, prove that women have a higher vocal frequency than men.

To analyze the two different audio is the matrix profile has been defined by trying different time window values and then selecting a window_size=25 for each series.

From the analysis we obtain the following motifs, related to the respective discords:

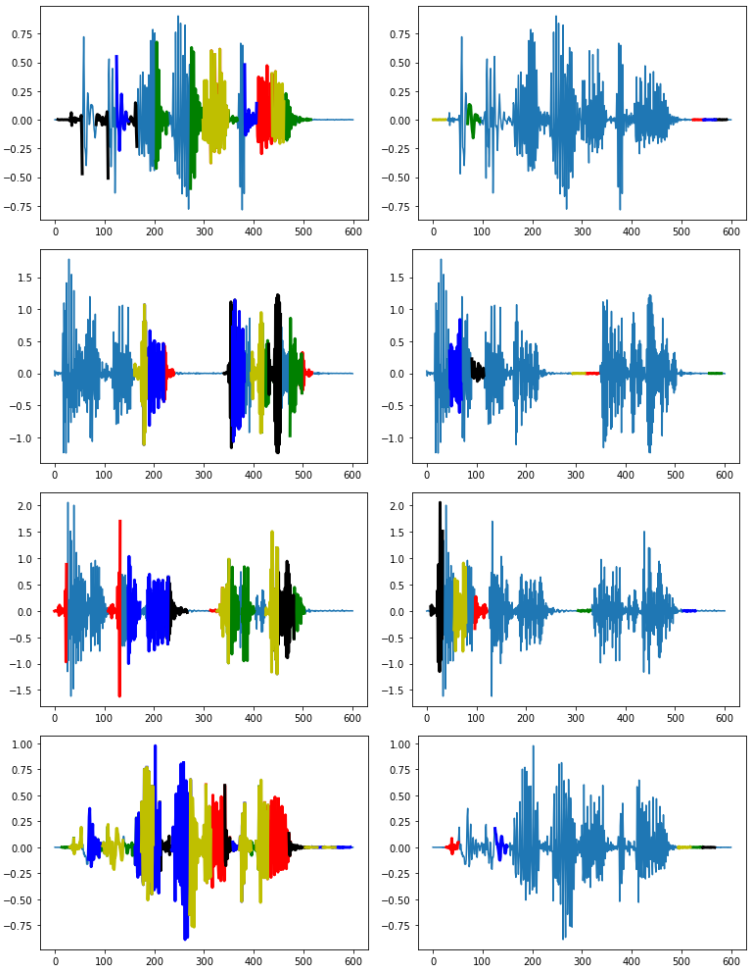

Figure 52: Motifs vs Discords

The figure above shows from left to right the Motifs and the Discords relative to an actor respectively to make a comparison; the first two lines of charts are men who execute the statement for the first and then the second repetition; The next two lines of charts follow the same logic, only the wave is relative to a woman. Five Motifs (relative to the first column of visualization) and five Discords are identified in this analysis (relative to the second column of visualization).

10.3 Clustering

Two different tests are performed using the approaches:

- ***Partitionale***: the K-Means algorithm, the most commonly used for these tasks, which uses Dynamic Time Warping as its metric;
- ***Hierarchical***: sklearn's AgglomerativeClustering algorithm, using different linkage criteria (Single, Complete, Ward and Average).

Both tests were carried out using two different types of approximations on time series in order to compare clusters. The approximations are set according to the following criteria:

- *Discrete Fourier Transformation, DFT* : 64 segments;
- *Symbolic Aggregate Approximation, SAX* : 100 segments, 64 symbols;

The choice of the above criteria is to be linked to a trade-off between computationality and clarity of results produced after affixing on time series. In fact, Sax's first attempt involved 1000 segments and 128 symbols, but it was then too computationally challenging to deal with the subsequent algorithms that we will cover in the next sections.

10.3.1 K-Means

As specified in the previous section, the K-Means algorithm is implemented using an approximate dataset with both ***Discrete Fourier Transformation*** and ***Symbolic Aggregate Approximation***.
In the first case (DFT), to identify the best K a search was made in a range of values of k between 2 and 10. From this we have selected a k=6 (with a consequent value of SSE=514929.64), parameter that we have validated in how much an increment even of only 1 of the number of k would have involved a minimal improvement of SSE, so we preferred to work in this way to optimize computationality. This allows us to get an average textitSilhouette score of 0.868 allowing us to assert that clusters have a good separation.

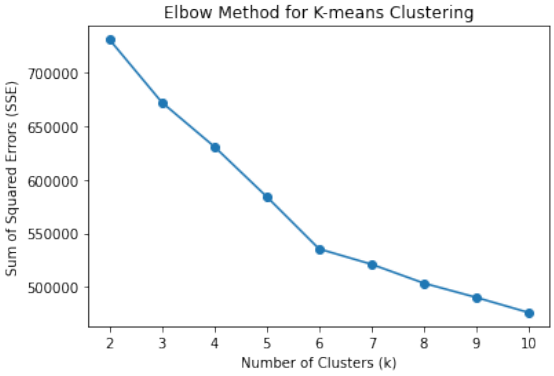

Figure 53: Trade off SSE - n. of K

In the second case (SAX), also in this case the choice of the number of cluster to identify has involved the strategy previously explained. Also in this case we identify k=6 that allows us to have less interesting results in terms of textbfSilhouette score equal to 0.1598 although the textbfSSE gets a significant decrease up to 15346. Both measures, the silhouette score and the ESS, provide information on the quality of clustering, but focus on different aspects. The silhouette score assesses both the internal cohesion of clusters and the separation between clusters, providing an overall measure of clustering quality. So we can say that the first approximation contains a higher variance than the second but also a worse performance in terms of Silhouette score.

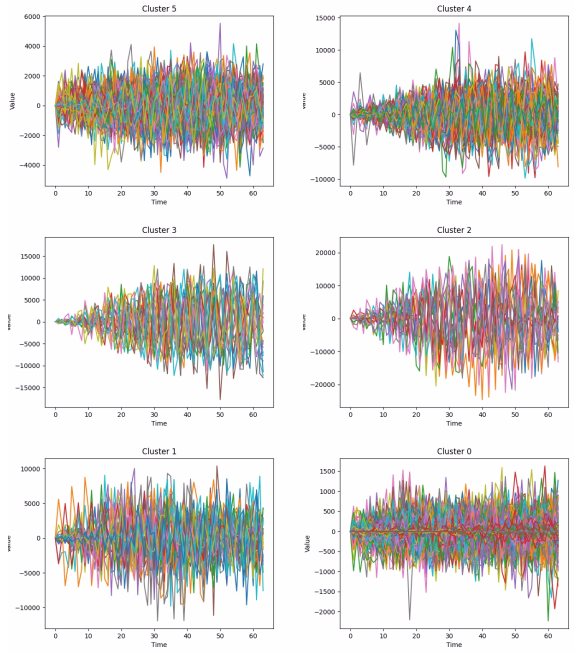

Figure 54: Cluster from Kmean - DFT

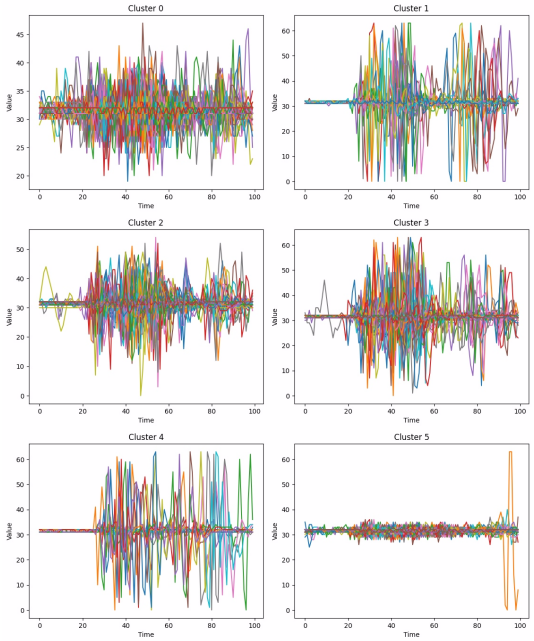

Figure 55: Cluster from Kmean - SAX

	Clusters with Sax					
Emotions	**0**	**1**	**2**	**3**	**4**	**5**
angry	50	5	33	8	1	183
calm	65	2	7	1	0	205
disgust	14	2	4	9	2	113
fearful	48	6	27	18	6	175
happy	35	7	10	14	3	211
neutral	18	0	0	0	0	122
sad	45	6	8	11	1	209
surprised	29	3	7	5	2	98

	Clusters with Dft					
Emotion	**0**	**1**	**2**	**3**	**4**	**5**
angry	201	1	0	3	24	51
calm	230	3	0	1	6	40
disgust	122	4	4	3	4	7
fearful	187	14	3	3	23	50
happy	221	1	2	7	16	33
neutral	139	0	0	0	0	1
sad	230	7	6	1	12	24
surprised	116	1	3	2	8	14

Table 5: Cross Tab of clusters

It can be seen that the SAX approximation provides better distributed clusters although it is diametrically opposed to how the first approximation clusters many instances in cluster 0 in the case of SAX, Analago situation however in cluster 5 in the case of approxiazione with DFT.

10.3.2 Agglomerative Clustering

By using Agglomerative Clustering the same approximation experiments as above have been implemented. In this case, different types of linkage have also been implemented to appreciate, varying the methodology with which distances are calculated to form a single cluster. The number of clusters searched is 6 to make a comparison between the two algorithms. Both in the case of implementation with DFT and SAX we report results in our opinion not significant and conflicting in meaning. We get not well defined clusters (1800 records are identified in a single cluster, and the remaining 5 are made up of one or two records) with a very high silhouette score (in the order of 0.80 average) therefore very contrasting.

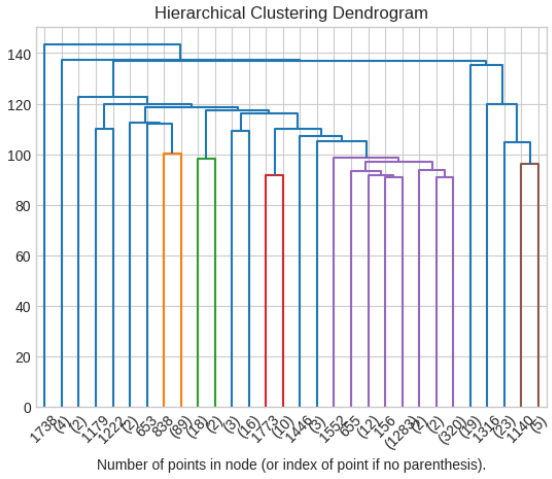

Figure 56: Complete Linkage

10.4 Classification

At the end of this analysis, the time series are analysed according to classification practices. For this task, the target variable ***"Emotional _intensity"*** has been taken into account and can take **normal(label 0)** and **strong(label 1)**.

For this task we decided to approximate our dataset with SAX. The following lists the different models defined and evaluated through a table that compares the accuracy and F1-score of the various experiments.

- Shapelet Classifier
- KNeighborsClassifier
- Random Forest
- Bagging and Decision Tree
- Decision Tree
- KNeighborsClassifier with Euclidean Distance
- KNeighborsClassifier with Manhattan Distance
- Convolutional Neural Network
- Canonical Interval Forest(CIF)

All the algorithms illustrated in the previously exposed bet list have been evaluated both using the distances calculated between the 5 shapelet (identified by the first model). Then both the normal Numpy array by default and the numpay array obtained from the shapelet extractor algorithm were used as train data.

No exhaustive searches of hyperparameters have been performed through GridSearch as in previous chapters, due to the high computational expenditure.

The neural network was structured on four sequential blocks identical (each separated by a dropout layer to control overfitting) consisting of three layer:Conv1D, BatchNormalization, Activation Relu; the architecture then culminates with a layer of pooling GlobalAverage1d and a Dense node that uses the sigmoid activation function to return the class forecast.

The table below shows the results in terms of performance of the tested models:

	Accuracy	F1
Shapelet Model	0.64	0.56
Shapelet-based KNN	0.64	0.61
Shapelet-based Random Forest	0.67	0.62
Shapelet-based Decision Tree	0.68	0.65
Shapelet-based Bagging+DT	0.69	0.69
KNN - Euclidean	0.52	0.47
KNN – Manhattan	0.63	0.60
Convolutional NeuraNetwork	0.82	0.82
CanonicalIntervalForest	0.73	0.69

Figure 57: Performances

As can be seen from the table, the best results have been obtained with more complex models such as Convolutional Neural Network that gets an excellent 82% although even the CanonicalIntervalForest can reach a decent 0.73%. On average, all classifiers achieve acceptable performance. Finally, it was decided to focus attention on the Shapelets obtained by the classifier and used to discriminate time series in the two classes Normal and Strong.

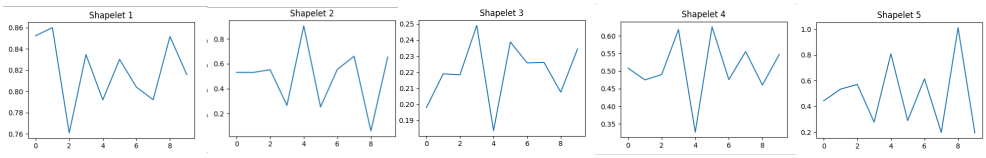

Figure 58: Shapelets

The top graph shows the 5 identified shapelets. The third and fourth are very similar. Overall all five move in a range of amplitude between 0.19 and 1. In their analysis, with Decision Tree and knn, the highest textbffeature importancevalues (corresponding to the shapelets) are relative to the second and third, since the third is similar to the second. We report the values of features importance:

$$[0.15874987, 0.28331232, 0.26579809, 0.13832475, 0.15381497]$$