
UNIVERSITÀ DI PISA

**Master Degree in Data Science and Business Informatics**

Data Mining: Advanced Topics and Applications

# Project Report: the RAVDESS dataset

**Vali Craciun**
**Francesca Scotti**
**Valeria Picchianti**

Academic year 2022-2023

# Table of contents


**1 Module 1** **2**
- 1.1 Data Understanding and Preparation . . . 2
- 1.2 Outlier Detection . . . 2
  - 1.2.1 LOF and COF . . . 2
  - 1.2.2 ABOD . . . 2
  - 1.2.3 Isolation Forest . . . 3
  - 1.2.4 Treating outliers . . . 3
- 1.3 Imbalanced Learning . . . 3
  - 1.3.1 Oversampling . . . 4
  - 1.3.2 Undersampling . . . 4
  - 1.3.3 Oversampling and Undersampling: a combination . . . 5

**2 Module 2** **7**
- 2.1 Advanced Classification . . . 7
  - 2.1.1 Logistic Regression . . . 7
  - 2.1.2 Support Vector Machine . . . 9
  - 2.1.3 Neural Networks . . . 10
  - 2.1.4 Ensemble Methods . . . 12
  - 2.1.5 Final comparison . . . 15
- 2.2 Advanced Regression . . . 15
  - 2.2.1 Gradient Boosting Regressor . . . 15
  - 2.2.2 Support Vector Regressor . . . 16

**3 Module 3** **18**
- 3.1 Data Understanding and Preparation . . . 18
- 3.2 Clustering . . . 18
  - 3.2.1 Content analysis and visualization . . . 19
- 3.3 Motifs and discords . . . 20
- 3.4 Classification with KNN . . . 22
  - 3.4.1 State of the art Time Series Classification . . . 23
  - 3.4.2 Shapelets . . . 25

**4 Module 4** **27**
- 4.1 Local explanation . . . 27
- 4.2 Global explanation . . . 28

# 1 Module 1

## 1.1 Data Understanding and Preparation

In the first part of the project, the correlation analysis was carried out: it resulted in dropping variables that measured a correlation greater than 0.95 with at least 3 other variables. Such procedure allowed to remove 92 variables. Subsequently, the analysis of variance was carried out: a value of 0.15 was chosen as threshold and in this way an additional 159 variables were selected and removed. Finally, the decision tree-based feature selection technique was applied: the classifier was firstly hyper-tuned (see table 1 for values, best ones: `criterion = ''entropy'', max_depth = 39, min_samples_leaf = 0.130, min_samples_split = 0.148`) and then trained. This strategy allowed to drop 119 more variables, keeping only the variables features chosen by the model. The dataset resulting from this selection consists of 55 numerical variables, 10 categorical variables, and 1828 observations.

| **Parameter** | **Tested values** |
|---|---|
| min_samples_split | loguniform(1e-2, 1e0) |
| min_samples_leaf | uniform(0.001, 0.2) |
| max_depth | randint(2, 200) |
| criterion | gini, entropy |

Table 1: Hyper parameters tested for selecting features from the decision tree classifier.

## 1.2 Outlier Detection

The second phase of the project concerned the detection of outliers: three families - density-based, angle based and model-based - were explored, dealing with COF and LOF at first instance, then ABOD and lastly Isolation Forest.

### 1.2.1 LOF and COF

LOF and COF are some of the most used density-based approaches for outlier detection. Since we are dealing with densities and hence, distances, the first step was to normalize the data. Even though density-based approaches are not well suited for high dimensional datasets due to the loss of meaning of the concepts of distance and density, we applied them anyway for the sake of comparison. LOF is very useful since it accounts for the relative density of a point to its neighbours, but this exposes the method to a relevant problem which is the sensibility to the size of the neighbourhood ($k$). Indeed the number of outliers changes depending of the chosen value of k, affecting the reliability of the method. The number of outliers changes from 21 to 37 if we consider k between 1 and 200.

This is the main reason that led us to use also COF along LOF. In fact, even if COF itself uses a notion of neighbourhood, is way less sensitive to changes of k by virtue of the chaining distance, used instead of the euclidean or manhattan distance. In our case, tickling k didn't lead to any changes at all in the number of outliers found, which is 183 so roughly 10% of the instances.

All scenarios were handled without the usage of the novelty parameter since the goal was not to classify a new instance as an outlier or not, but to find outliers in the dataset.

### 1.2.2 ABOD

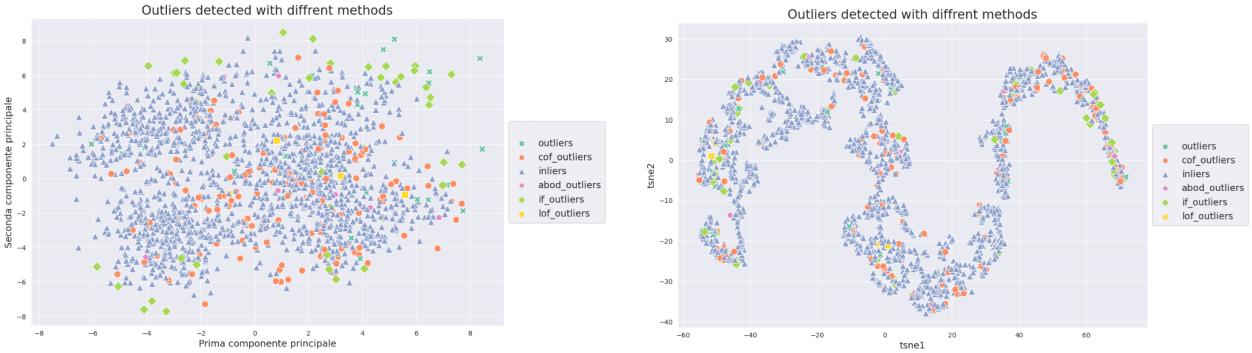
The decision to include ABOD among the methods used is due to the fact that it aims to solve the issue of curse of dimensionality, from which all distance- or density-based methods are affected, and it is therefore particularly suitable for high-dimensional data. To identify outliers, we applied ABOD to the preprocessed and scaled dataset. Although two different versions are supported - the original and the fast version - Fast ABOD was performed: it uses k-nearest neighbours to approximate the variance instead of calculating it from all possible pairs for a given point. This choice allowed to avoid the high time complexity of $O(n^3)$ of the original method, but made it necessary to define the number of $k$ nearest neighbours to be considered. Also, a parameter *contamination*, which is the proportion of outliers in the dataset, needed to be set: it was considered appropriate to set it equal to the proportion of outliers identified by the other methods, that is 0.02. As regards the number of nearest neighbours, we iterated over $k$, since by varying the value of k, we could explore different levels of local versus global outlier detection. However, the number of outliers identified by ABOD as a function of k fluctuates between 34 and 40 for a range of k values between 5 and 60, meaning that such number doesn't seem to be much sensitive to $k$. By testing different values of k, we could be more confident that the results are robust and not sensitive to a particular choice of parameter.

### 1.2.3 Isolation Forest

Isolation Forest is a tree-based model that performs outliers detection efficiently in high-dimensional datasets. It was decided to keep the default parameters *(n_estimators = 100, max_samples = 256, max_features = 1.0 , contamination="auto")* and 38 outliers were obtained, accounting for 2.078% of the observations.

### 1.2.4 Treating outliers

Subsequently, the analysis of outliers was deepened: the strategy adopted involved checking which observations were outliers for *at least three* of the four methods performed. The result was that 1.47% of the total number of observations fell into this set and we decided to remove them completely from the dataset.




(a) Outliers on the first two principal components

(b) Outliers on two t-SNE dimensions.

Figure 1: Outliers scatterplot.

From figure 1 plot (a) and (b) it is possible to notice that the majority of outliers come from the COF method, while the less frequent ones come from the ABOD method.

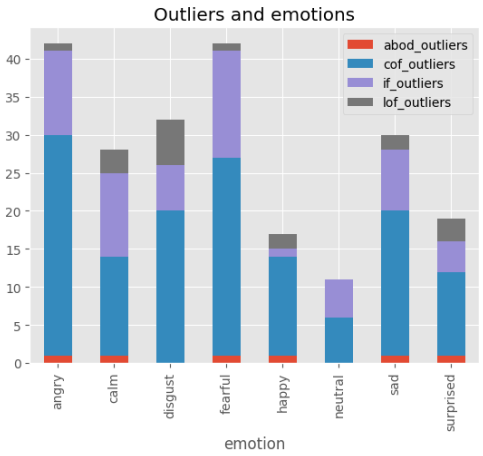



Figure 2: Outlier stacked bar chart for emotion.

In figure 2 it is possible to observe the distribution of the outliers detected with different methods among the different emotions. The *fearful* emotion has the most outliers while *neutral* has the least. Most of the outliers are detected with the COF method.

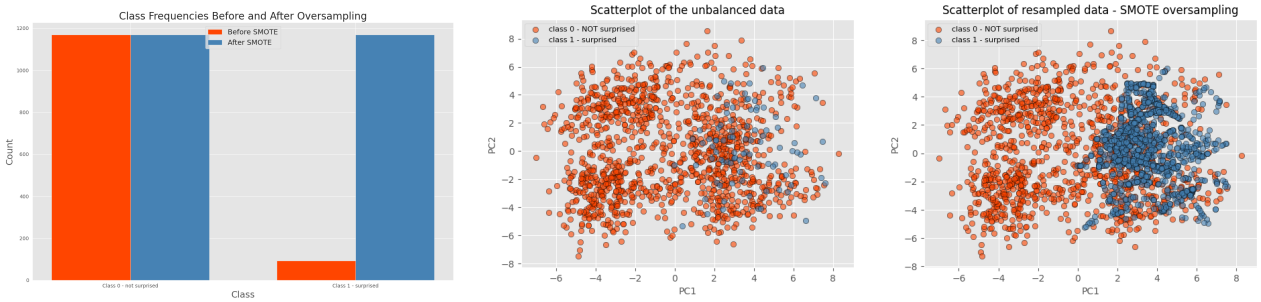
## 1.3 Imbalanced Learning

The study of imbalanced learning was carried out by considering an unbalanced binary classification task, using Emotion *surprised* and *not surprised* as the target variable, then solved with a Decision Tree. We firstly considered a base case, i.e. a decision tree hyper- tuned[1] on the unbalanced training set, and each time we rebalanced the training set, a new decision tree was tuned and trained on it. The imbalance is such that 93% of *not surprised* class records vs 7% of *surprised*.

[1] `criterion = ''entropy'', max_depth = 39, min_samples_leaf = 0.130, min_samples_split = 0.148.`

### 1.3.1 Oversampling

The two techniques that were considered to perform oversampling are SMOTE and ADASYN, which aimed to balance the training set only, by oversampling the minority class (*surprised*). Figure 3 shows a visualization of how SMOTE worked on our dataset; a similar visualization could also be done for ADAYSIN (not shown since it is completely analogous). In both cases, once the dataset was re-balanced, to test the effectiveness of the two approaches, the performance of the decision tree trained on the resampled data space was evaluated and compared to the performance on the unbalanced data. Table 2 shows such comparison in terms of F1 score.




(a) Class frequencies (b) Unbalanced data (c) SMOTE rebalancing

Figure 3: Showing the effect of SMOTE oversampling: plot (a) shows class frequencies before and after SMOTE and it is evident that after applying such technique we have a perfect ratio in the dataset between classes; plot (b) and (c) aim to show where synthetic points of the minority class lie in the reduced PCA space, comparing with the unbalanced data.

| | F1-score(class 0) | F1-score(class 1) | MACRO |
|---|---|---|---|
| **Base** | 0.96 | 0.00 | 0.48 |
| **SMOTE** | 0.93 | 0.46 | 0.69 |
| **ADASYN** | 0.92 | 0.42 | 0.67 |

Table 2: Main performances of a classifier trained on the oversampled space.

Overall, we can see the effectiveness of both oversampling techniques inspected, which do not show particularly different results.

### 1.3.2 Undersampling

The same task of classifying *surprised* was also tackled using undersampling techniques. More specifically, we exploited Random undersampling, Edited NN, Condensed NN and Tomek links. The most relevant results are summarized in Table 3.

| | Ratio[2] | F1-score (class 0) | F1-score (class 1) | MACRO |
|---|---|---|---|---|
| **Base** | 0.07 | 0.96 | 0.00 | 0.48 |
| **Random Undersampling** | 0.5 | 0.80 | 0.34 | 0.57 |
| **Edited NN** | 0.5 | 0.67 | 0.30 | 0.48 |
| **Condensed NN** | 0.37 | 0.92 | 0.41 | 0.67 |
| **Tomek Links** | 0.07 | 0.95 | 0.00 | 0.47 |

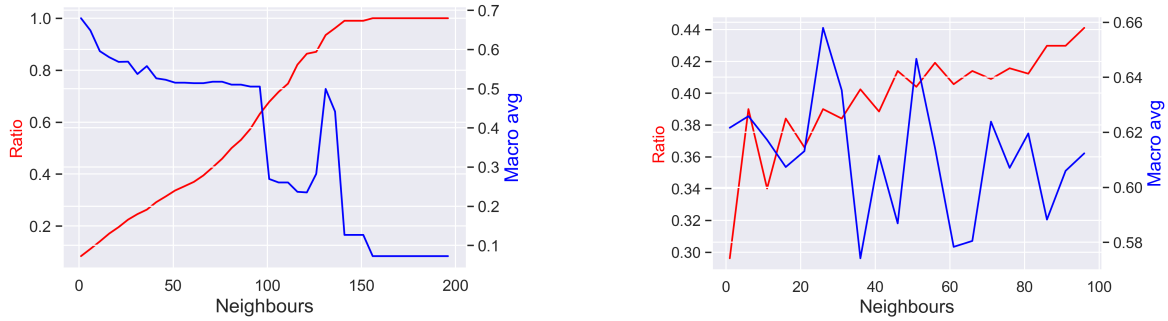
Table 3: Results from various undersampling techniques

We can clearly see how the base classifier is completely unable to classify the minority class due to the high imbalance. If we only look at the *macro* value, CNN obtained the best result and it's also noticeable how it succeeded in not losing too much performance on not surprised points. Of course this is due to the fact that CNN returned a slightly imbalanced training set if we consider the ratio value for random undersampling or ENN, which succeeded in perfectly rebalance the training set, but still, even with less surprised points the F1(1) was the highest, meaning that CNN made wiser choices on which points to remove from class 0. Tomek links

[2]Ratio represents the presence of class 1 relatively to the whole rebalanced training set (0.5 means both classes are equally represented in the rebalanced training set).

didn't even been able to rebalance the the training set, leaving ratio and performances untouched from the base case.

For ENN and CNN we have to take into account their dependence by the number of neighbours as a parameter. The results in the table refers to the best value of n for both, where for best we mean a good trade off between ratio and macro.




(a) Ratio and Macro for ENN

(b) Ratio and Macro for CNN

Figure 4: ENN and CNN performances by changing the number of neighbours

From the figure we can observe that for ENN the macro tend to decline as n grows, while this not hold for CNN. Another interesting observation is that CNN is not able to perfectly rebalance the data, actually having a pretty stable ratio since it only ranges between 0.30 and 0.44 in our case, while ENN is able to range from 0 to 1. The values in table 3 are derived from setting n = 300 in ENN and n = 41 for CNN.

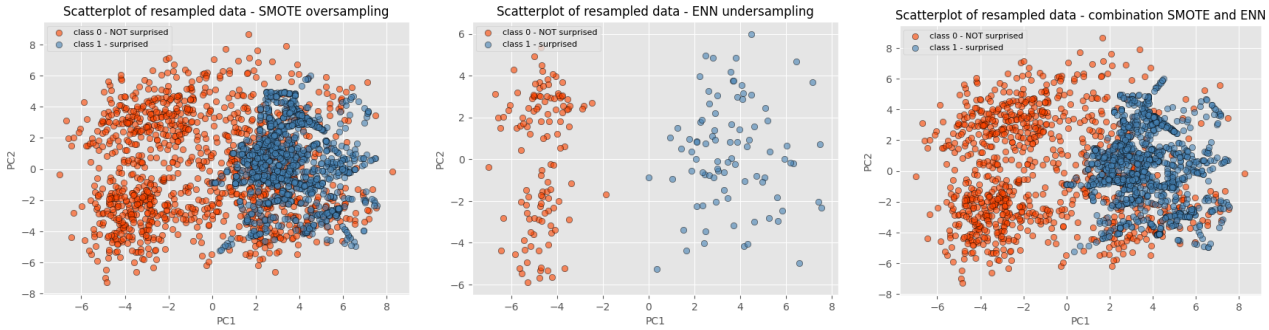
### 1.3.3 Oversampling and Undersampling: a combination

We also tried to explore the combined use of oversampling and undersampling techniques to rebalance the dataset and in this section we'll briefly report the results. Specifically, SMOTETomek and SMOTEENN were performed. Both are initially using SMOTE to oversample the minority class and then Tomek links or ENN to clean the decision boundaries, by removing noisy majority class samples . As before, a decision tree classifier was trained on the two different resampled datasets and tested it on the original imbalanced test set. The results of the classification reports show that both SMOTETomek and SMOTEENN were actually able to improve the performance of the classifier: the recall of the minory class went from zero (unbalanced training) to 84% for SMOTEENN and 82% for SMOTETomek (see table 4). Such increase in recall means that the classifier is now able to correctly identify a much higher percentage of the positive instances in the dataset: the resampling techniques have made the classifier more sensitive to the minority class.

| | Recall(class 0) | Recall(class 1) | MACRO |
|---|---|---|---|
| **Base** | 1.00 | 0.00 | 0.50 |
| **SMOTE ENN** | 0.80 | 0.84 | 0.82 |
| **SMOTE Tomek** | 0.79 | 0.87 | 0.83 |

Table 4: Main results of combining undersampling and oversampling techniques

The effect of the techniques is also shown visually in figure 5: it can be seen that, although not particularly marked, the effect of combining SMOTE and ENN is precisely to clean up the decision boundaries, where we actually see fewer points (c).




(a) SMOTE (b) ENN (c) SMOTEENN

Figure 5: Scatter plots obtained performing different rebalancing techniques.

# 2 Module 2

## 2.1 Advanced Classification

In this chapter, the results of the classification task will be presented: such task is performed on different datasets, inlcuded the pre-processed one as described above. The target variable used is *emotion*. Models explored, which will be discussed in separate sections, are: Logistic Regression, Support Vector Machine, Neural Networks, Ensemble Methods, Gradient Boosting.

### 2.1.1 Logistic Regression

For logistic regression we decided to try different combination of predictive features. Initially, all 429 features were used to predict the target variable. This choice was made in order to create a base case to improve and so to measure these improvements while setting different parameters. Moreover, logistic regression works well also with high dimensional data since it doesn't rely on densities or distances computations. We then build a model using only the 55 features selected from the feature selection section. From now on we'll refer to the model with all the features as "whole" and "sel for the selected features model.

The target variable is categorical with 8 classes, so this is a case of multinomial regression handled in a one vs all fashion, which means that all the odds are intended as $\frac{p(c_i)}{1-p(c_i)}$, where $c_i$ is the $i^{th}$ class. This approach seemed to us more clear than picking a base class to be compared with all other classes.

**Regularization** The untuned whole base classifier was extremely overfitted, with a train accuracy of 0.95 and test accuracy of 0.45, and first step towards reducing overfitting was to compare the effect of l1 and l2 regularization. The accuracies reported earlier came from a model with default parameters and scikit learn uses l2 as default regularization. We observed that just by switching form l1 to l2 overfitting is reduced, indeed train accuracy was then 0.86 and test accuracy is 0.49, all else being equal. This is, of course, still too much difference but is noticeable how much the coefficients of the model are impacted. With l2, for each class we have 51 coefficients with 0 value, while using l1 each class has more or less half of the coefficients set to 0. This come from the fact that l1 is able to set coefficients to 0 while l2 is only able to set them very close to 0.

We then made a more deep research on the impact of the regularization strength with both l1 and l2 approaches. This factor is represented by the C hyper-parameter, where a high value of C means more emphasis on the training data at the expense of model complexity, and the inverse for a low value of C. The first step was to analyse a broad range of C, as shown in figure 7, to then find an interesting narrower range to be used for the grid search. The same methodological path was used also for the sel model. Other hyper-parameters didn't affect the model significantly.

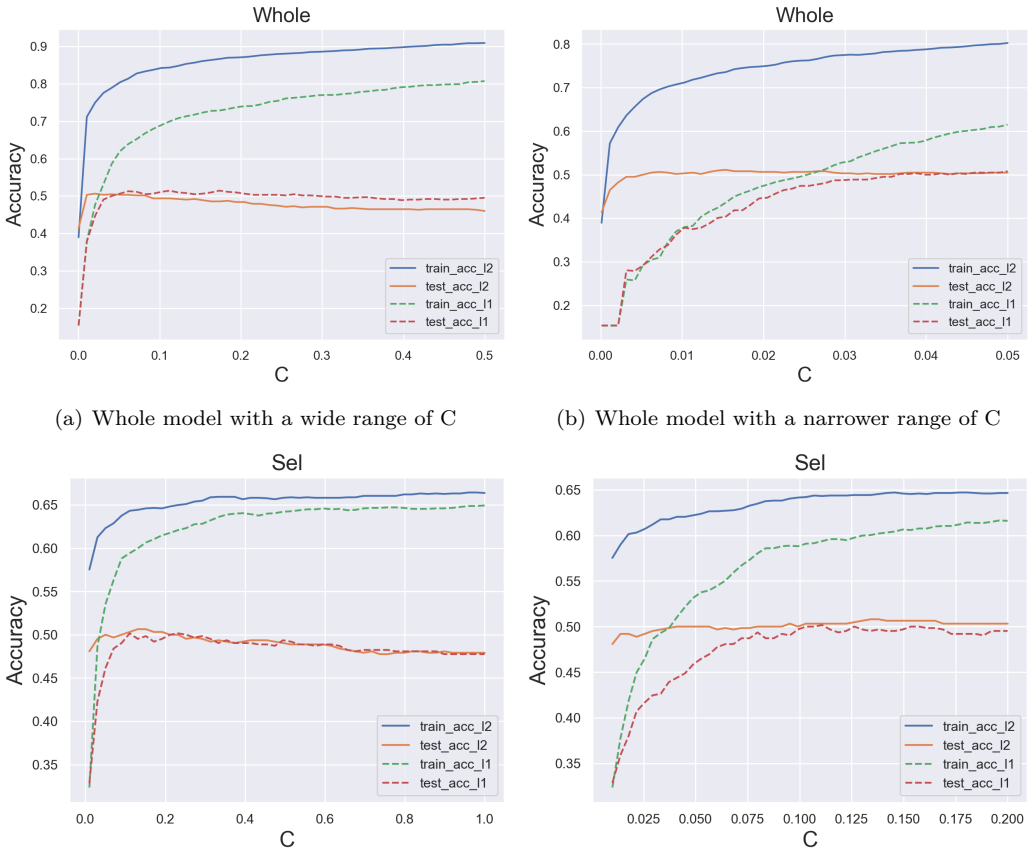
| | Train Accuracy | Validation | Test Accuracy |
|---|---|---|---|
| **Whole Base** | 0.95 | / | 0.48 |
| **Sel Base** | 0.66 | / | 0.48 |
| **Whole tuned** | 0.62 | 0.59 | 0.49 |
| **Sel tuned** | 0.63 | 0.56 | 0.50 |

Table 5: Base models vs Tuned models

| | Regularization → best | C → best | Solver |
|---|---|---|---|
| **Whole model** | {L1, L2} → L2 | [0.001, 0.005] → 0.003 | Liblinear |
| **Sel model** | {L1, L2} → L2 | [0.05, 0.2]→ 0.2 | Liblinear |

Table 6: Hyper-parameters choices for both models

L1 regularization is a bit less susceptible to overfitting, but this also lead to a less test accuracy especially for small values of C. Notice that this reasoning is valid for both models but with an important difference: the range of interest is bigger by an order of magnitude in the sel model than the whole model. This is more clear by looking at table 5, where we can see that the range for whole is scaled down w.r.t to the sel one. This translates to the need of a powerful regularization for the model considering all attributes.




(a) Whole model with a wide range of C

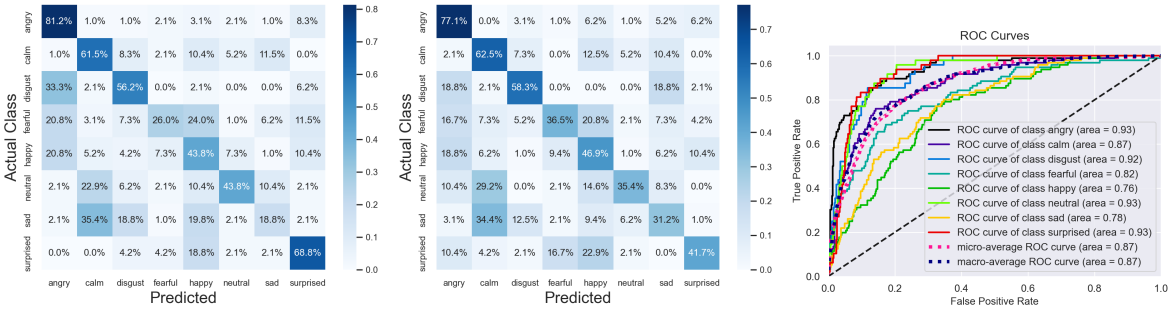
(b) Whole model with a narrower range of C

(c) Sel model with a broad range of C

(d) Sel model with a narrower range of C

Figure 6: Impact of different C ranges on Whole and Sel models

**Performances** From table 5 we can see that both models report similar performances. This is true not only for accuracy but also if we look at roc cruves and confusion matrix. Actually, we can find few differences especially for the surprised class, where the sel model seemed to handle it way better than whole, and in general sel achieved better recall on more "extreme" classes, while whole had better results in classifying less distinguishable classes.




(a) Confusion matrix of Sel

(b) Confusion matrix of Whole

(c) Roc curves of Sel

Figure 7: Performance comparison between Whole and Sel

**Coefficient analysis** At this point of the analysis might be also worth it a brief discussion on the coefficients values. Since in both cases l2 was chosen we have that formally all attributes have a coefficient different than 0, even if for most of them this value is negligible. Now, with all these parameters we are extremely exposed to multicollinearity. The phenomena doesn't affect the predictive power of the model itself, but it may cause several problems if we want to better understand the impact of specific attributes on the final prediction. This happen because multicollinearity increases the variability of coefficents estimators, leading to a less reliable

estimate. To overcome this issue we used the variance inflation factor to recursively eliminate features with a high vif (literature set around 5 as a rule of thumb[3]) i.e. all those features easily linearly regressed using all other features, and thus redundant.

| | First Attribute | $\beta$ pre VIF | Second Attribute | $\beta$ pre VIF | First Attribute | $\beta$ post VIF | Second Attribute | $\beta$ post VIF |
|---|---|---|---|---|---|---|---|---|
| **Angry** | emotional intensity | -1.33 | mfcc_mean | 0.85 | emotional intensity | -0.28 | / | / |
| **Calm** | vocal channel | -0.86 | stft_kur_w3 | 0.83 | vocal channel | -0.09 | stft_kur_w3 | -0.03 |
| **Disgust** | vocal channel | -1.14 | mfcc_std_w2 | 0.73 | vocal channel | -0.44 | / | / |
| **Fearful** | vocal channel | -1.46 | sc_min_w4 | -0.90 | vocal channel | -0.27 | sc_min_w4 | -0.0008 |
| **Happy** | mfccc_sum_w2 | 1.04 | mfcc_q23_w3 | -0.65 | / | / | / | / |
| **Neutral** | emotional intensity | -0.93 | mfcc_q25_w3 | 0.85 | emotional intensity | -0.66 | / | / |
| **Sad** | mfcc_kur | -0.88 | stft_sum_w3 | -0.76 | / | / | / | / |
| **Surprised** | mfcc_q25_w3 | -1.20 | mfcc_q99_w2 | 0.64 | / | / | / | / |

Table 7: $\beta$ values of the best 2 attributes of the for each class before and after VIF analysis for the sel model.

To better highlight the impact of multicollinearity, in table 7 we report the highest 2 coefficients in absolute value for each class, then we report their value after the vif analysis. We can observe how in some cases attributes previously chosen disappear because they have been removed during the vif analysis, due to their high collinearity. This is an important result since relying exclusively on the coefficient values obtained by the model may mislead us to overestimate the impact of a certain attribute in predicting specific class. In the end, the sel model remained with only 20 features after the vif analysis.

### 2.1.2 Support Vector Machine

To perform classification using Support Vector Classifier (SVC) as a model, the dataset was initially scaled. In the first instance, we observed the performance of the SVC model using the default parameters: *kernel: "rbf", gamma: "scale", c: 1.0*. The result is that the model has an accuracy on the test set of 47% and a precision weighted by taking the number of samples in each class as a weight of 47%. To improve performances, the hyper-parameters of the model were optimised using as a technique a Grid search performing 5-fold cross validation. The hyperparameters being tuned are "C", "kernel", and "gamma", exploring a set of predefined values. Different configurations of grid values were inspected. Specifically, at the beginning values shown in table A were tested, giving an average accuracy on the validation test of 59% and an accuracy on the test set of 42%. Subsequently, completely different values of C and gamma, found by investigating research best practices[4], were inspected (table B). This solution involved setting values for gamma and C to be exponentially growing sequences. As can be seen, although in this case the average accuracy on the validation test is lower than in the previous case, 46%, the accuracy on the test set is improved by 6 percentage points (48.2%). Finally, the last attempt at improvement involved further fine-tuning around the best hyperparameters found so far, trying a more focused search, where values are varied in a smaller range around the best ones - C = 32.0, class weight = "balanced", gamma = 0.001953125 (see table C). The result obtained was that the best hyper-parameters identified in this latest search (Randomized Search using 5-fold cross validation) are C = 31, class weight = "balanced", gamma = 0.00191, kernel = "rbf".

Once the model was trained using the above-mentioned hyper parameters, its performance was evaluated using both a confusion matrix and ROC curves; both are shown in figure 9. The confusion matrix provides a useful summary of the performance of the model across all classes: classes *angry, neutral, surprised* have the highest proportion of correctly classified instances, while we might notice that the worst proportion is for *sad*, for which only 20.8 percent of records classified as *sad* class were actually from that class. For the *happy* class, the model misclassified it as *fearful, surprised, and angry* the most. For *sad*, the model misclassified it as *angry, calm, and fearful* the most.This suggests that the model has difficulty distinguishing between such classes. When it comes to the ROC curves, it should be noted that these highlight that, although the performance of the model may appear particularly low, it should be remembered that the model is still far better in predictions than the random classifier. In our case, the ROC curve shows that our model achieves discrete performance for most classes. As expected, it is possible to observe some variability in the performance across different classes, with AUC scores for class *disgust* being higher (0.94), while AUC scores for class *sad* being lower (0.75).

[3]Menard S. Applied Logistic Regression Analysis. 2nd edition. SAGE Publications, Inc; 2001.
[4]https://www.csie.ntu.edu.tw/ cjlin/papers/guide/guide.pdf

| Parameter | Tested values |
|---|---|
| C | 0.01, 0.1, 1, 10, 50 |
| gamma | .01, .1, 1, 5, 10, 100 |
| kernel | linear, rbf, poly |

(a) Table A - first attempt

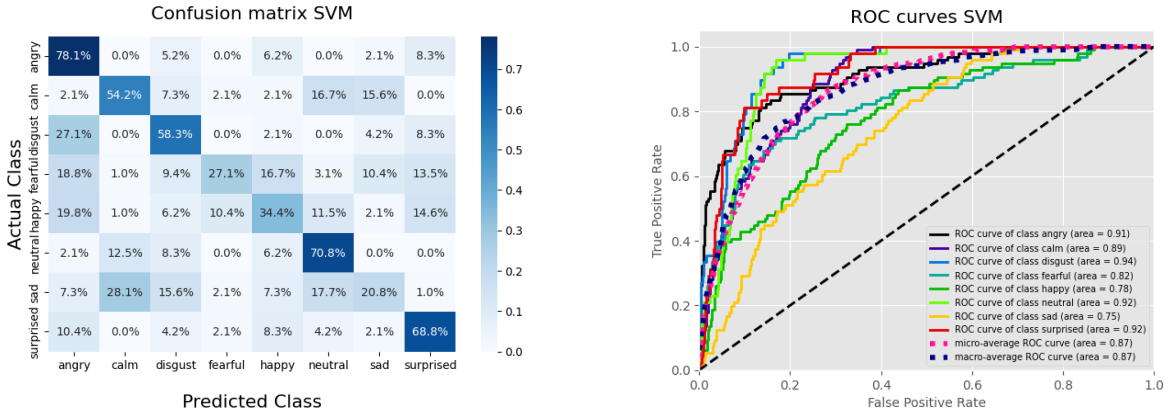
| Parameter | Tested values |
|---|---|
| C | `np.logspace(-5, 15, num=11, base=2)` |
| gamma | `np.logspace(-15, 3, num=19, base=2)` |

(b) Table B - final values

| Parameter | Tested values |
|---|---|
| C | `loguniform(20, 33)` |
| gamma | `loguniform(0.0005, 0.00410)` |

(c) Table C - fine-tuning around the best values

Figure 8: Grid search parameter values for SVC model




(a) Confusion matrix SVM

(b) ROC curves SVM

Figure 9: Evaluation of the classification results: (a) shows a confusion matrix while (b) shows ROC curves.

Finally, plot in figure 10 shows the mean permutation importance of the ten most significant features for the model. The most important feature is *mfcc_q25_w3*. Values are generally quite low, which might indicate that the model is not very reliant on any single feature for making predictions.

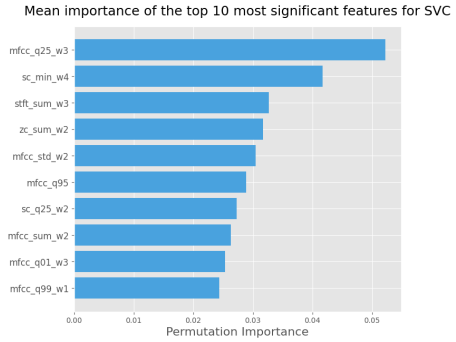


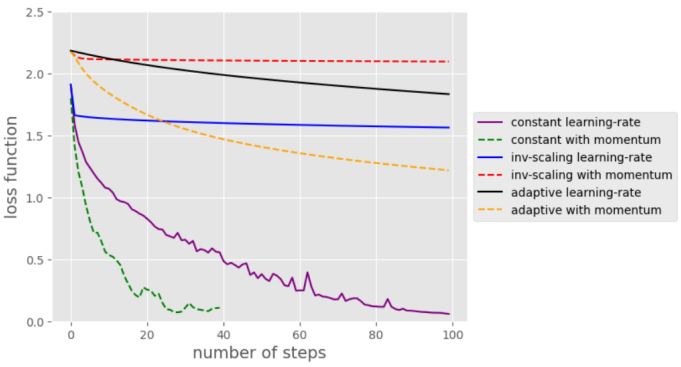

Figure 10: Feature importance of the SVC model

### 2.1.3 Neural Networks

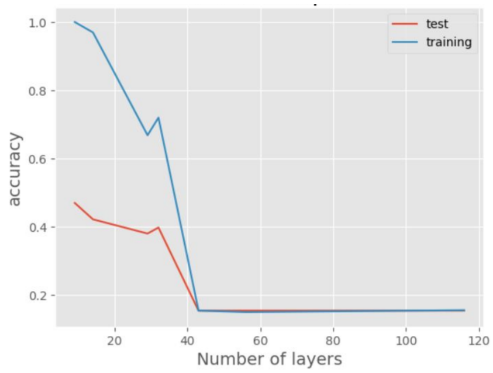
In order to perform classification on the variable emotion with neural network, both the training and the test dataset were scaled. The numerical variables taken into account were 55, while the categorical variables were 6, handled using the one hot encoding technique. The first task was to decide which values of the neural network's hyper-parameters were suitable for this classification problem. In order to choose which hyper-parameters to test we decided to draw few plots to better understand how the performances of the neural network changed varying the values of different hyper-parameters.

Plot (a) of figure 11 displays the loss function of six neural networks trained with different type of learning rate and momentum. The loss curves of the "inv-scaling" learning rate, that is to say the one that gradually decreases the learning rate at each time, and of the "adaptive " learning rate perform much worse than the others. Therefore it was decided not to test them further. The same approach was used for plot (b) of figure 11,

several neural networks were trained, each with different numbers of layers. It is possible to notice that after about 40 layers the performance quality decreases.




(a) Loss curves of six different neural networks




(b) Accuracy score of neural networks with different layers' number

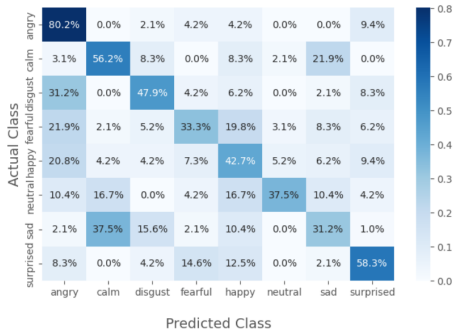
Figure 11: Hyper-parameters plots for Neural Networks.

In figure 11, plot (a) displays the loss function of six different neural networks with two hidden layers of 100 nodes each, "sgd" solver and "relu" activation function. The only thing that differs between each of the six classifier is the learning rate and the momentum as it is shown in the legend of the plot. Figure 11 plot (b) displays the accuracy of the classification task on the variable "emotion". The accuracy score is calculated on several neural networks with different number of hidden layers, while not changing the other parameters. Regarding the other hyper-parameters it was decided to test them using randomized search performing 3-fold cross validation according to table 8:

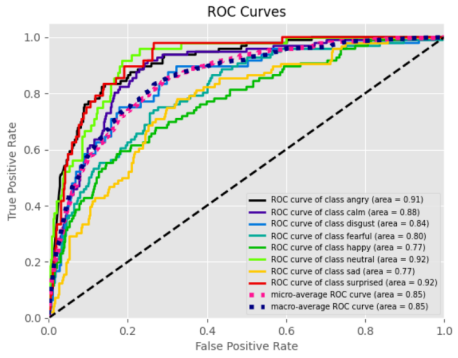
| **Hyper-parameters** | **Tested values** |
|---|---|
| optimizer learning rate | 0.001, 0.01, 0.1, 1 |
| modelactivation | relu, tanh |
| optimizer | adam, sgd |
| epochs | 10, 20,50,70, 100, 200 |
| hidden layer sizes | 2 layers of 10 nodes, 3 layers of 100 nodes, 3 layers of 50 nodes,<br>3 layers of 10 nodes, 7 layers of 100 nodes, 16 layers of 100 nodes<br>16 layers of 10 nodes, 27 layers of 100 nodes, 35 layers of 100 nodes. |

Table 8: Parameters tested for Neural Networks

The best result was obtained with *optimizer learning rate*: 0.01, *optimizer*: sgd, *model hidden layer sizes*: (50, 50, 50), *model activation*: relu and *epochs*: 100. Subsequently the model with such hyper-parameters was trained and evaluated using mainly confusion matrix, classification report and ROC curves.




(a) Confusion matrix of the Neural Network classifier




(b) ROC curve of the Neural Network classifier

Figure 12: Evaluation plot of the Neural Network classifier.

After performing the randomized search with cross validation , the Neural Network was tested on the test

set. The results of the classification task are shown in figure 12. The accuracy reached is 48.55% and as it can be seen in plot (b) of figure 12 the model is far from being a random classifier. The emotion with the highest proportion of correctly classified observations is *angry*, as it can be seen in the confusion matrix (a).

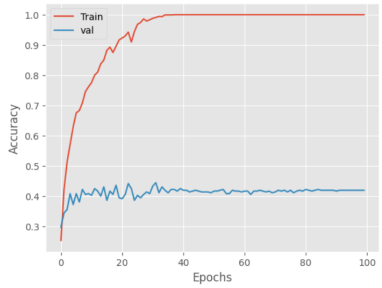
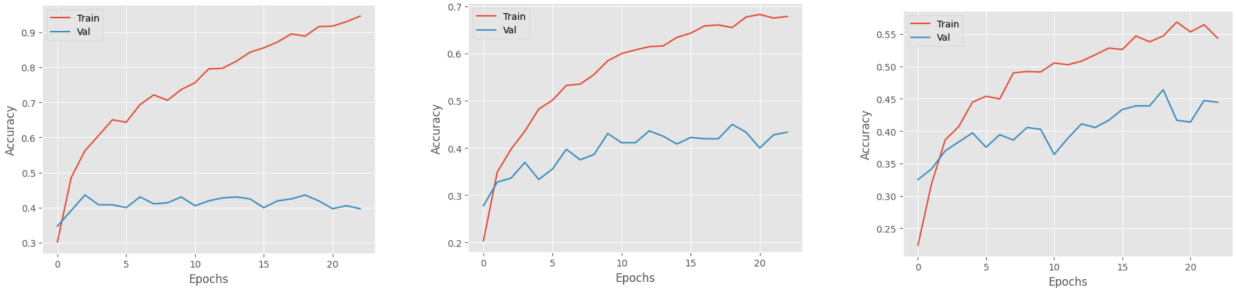



Figure 13: Accuracy score over the epochs

Figure 13 shows the accuracy of the classifier for every epoch and both from the training data and from the test data. As can be observed, there is a big gap between the two lines. This means that the classifier performs very well on the training data and poorly on the test data. To overcome this overfitting problem and improve the overall performances of the model we decided to implement three methods: dropout, early stopping and L2- regularization.

The early stopping method was set to have a "patience" of 20 epochs and to monitor the accuracy of the validation set. It stopped the classifier after 23 epochs. Later, l2 regularization was also added, which was set to have kernel regularizer=l2(0.001)) for every hidden layer. Finally the dropout method was added to the model with a dropout of 0.5 for every hidden layer.




(a) Early stopping (b) L2-regularization (c) Dropout

Figure 14: Training and Test accuracy over epochs for early stopping, l2 regularization and Dropout

Comparing this plots with figure 13 and figure 14 it is possible to see that the overfitting problem was partially solved in the plots (a) and (b) since there is still a gap bigger than 0.1, but the accuracy score of the training is lower. In plot (c) the two lines almost overlap, so the overfitting problem is solved. Finally, to decide which model performed the best classification of the emotion variable, this last three models (early stopping, L2-regularization and dropout) were trained and tested. The classification reports were compared and as can be seen in table 9, the best model overall is the one with l2-regularization.

| **Model** | **Accuracy** | **ROC** |
|---|---|---|
| simple | 0.4855 | 0.8505 |
| early stopping | 0.4759 | 0.8535 |
| l2-regularization | 0.4807 | 0.8610 |
| dropout | 0.4727 | 0.8684 |

Table 9: Model comparison

### 2.1.4 Ensemble Methods

**Adaboost** The first ensemble method we analysed was ADABOOST. All models were trained and tested with both original and normalized data, resulting in no relevant performance differences, focusing on the reduced dataset with 65 features, since using the whole dataset lead to very similar results. Since in ADABOOST the estimator can be chosen, we decided to make a comparison between:

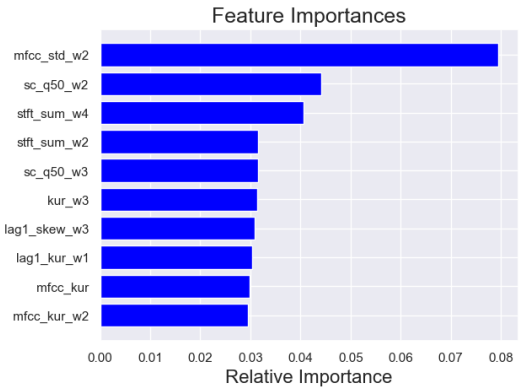



Figure 15: Feature importance on best 10 features for opt_log

- ADABOOST with decision stump (base)
- ADABOOST with a tuned decision tree (tuned_tree)
- ADABOOST with the optimized logistic model (opt_log)

Again, the first step was to train and test a base classifier with default parameters[5]. This model achieved a train accuracy of 0.40 and a test accuracy of 0.33, denoting a way less starting overfitting than non ensemble models. Such result is coherent with one of the rationales of ensemble methods, which is reducing variance, even if in this case it came along with an increment of bias.

After this, we tuned a tree[6] to be then used as weak learner in ADABOOST, using the same procedure of checking wide ranges of parameters and the focus on more interesting and narrower ones, also trying to find the best parameters for ADABOOST itself. We noticed that the number of estimators affected the performances of the model since, after roughly 100, the test accuracy stopped increasing while train accuracy not, worsening overfitting. With regard to learning rate, since this parameter depends on the number of estimators chosen, we decided to try multiple values between 0 and 1 directly in the grid search while also changing the number of estimators. The algorithm parameter seemed not to impact model performances, so was set to default as "SAMME.R".

| | **# of estimators → best** | **Learning rate → best** | **Train accuracy** | **Test Accuracy** |
|---|---|---|---|---|
| **Tuned tree** | [50,120] → 60 | [0.001, 1] → 0.09 | 0.48 | 0.39 |
| **Opt_log** | [10,25] → 10 | [0.001, 1] → 0.06 | 0.48 | 0.32 |

Table 10: Summary of best ADABOOST params and performances

In table 10 we can find a recap of the params for different versions of ADABOOST along with their performances. The best result was obtained by the tuned tree model, while opt_log not only recorded a worse accuracy than the base model, but in this case the ensemble method worsened the performances of its estimator. In figure 15 we also report the 10 most important features for the model. From the initial 55, only 23 were assigned a greater than zero importance.

**Random Forest** The approach followed to implement Random Forest initially involved hyper parameter tuning of the model (a randomised search performing repeated stratified k-fold cross-validation) considering different combinations of parameters, the most relevant of which are shown in table 16, with their respective results. In the first instance, although the mean accuracy on the validation set was in line with the results obtained with the other models, it was possible to note, evaluating the performance on the training set, that the model was clearly overfitting (training accuracy 99% vs validation accuracy 58%). In order to reduce this phenomenon and prevent the model from lacking generalisation, the strategy used involved acting on the two parameters that we considered to be the most influential: *max_depth* and *max_leaf_nodes*. With regard to the latter, a value ranging between 20 and 40 allowed overfitting to be limited, decreasing the accuracy on training without negatively affecting the one on validation. The same was noted for the *max_depth* parameter, for which clearly overfitting was more limited for smaller values of *max_depth* (ranging between 2 and 9). These considerations made it possible to choose the values for the grid more appropriately.

[5]estimator = Decision stump, n_estimators = 50 , learning_rate = 1, algorithm = SAMME.R
[6]max_depth=4, min_samples_leaf=0.1, min_samples_split=0.2, criterion = "gini"

| | Tested values | Best values |
|---|---|---|
| **max_depth** | 2, 3, 5, 6, 7, 8, 9 | 8 |
| **max_leaf_nodes** | 20, 22, 25, 30, 35, 40 | 30 |
| **min_samples_split** | 2, 3, 4 | 2 |
| **min_samples_leaf** | 2, 3, 4, 5, 6, 10, 15 | 6 |
| **max_features** | 4, 5, 6, 7, 8, 9 | 6 |

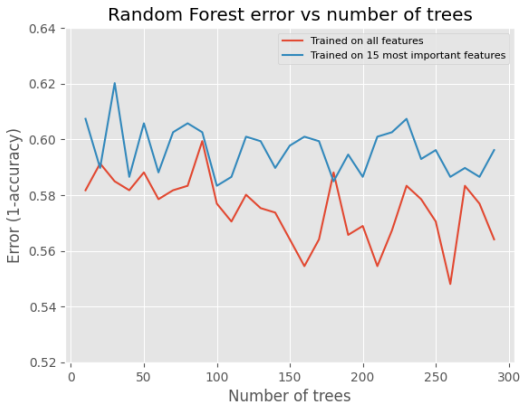
(a) Table A - parameter grid

| | |
|---|---|
| **Training accuracy** | 0.482 |
| **Validation accuracy** | 0.693 |
| **Test accuracy** | 0.433 |

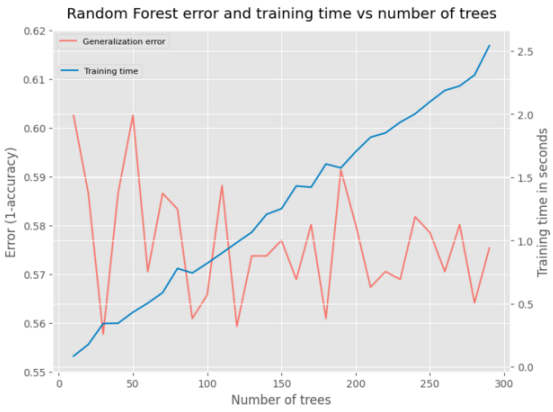
(b) Table B - accuracy scores

Figure 16: Grid search parameter values for Random Forest and accuracy scores

Once the model had been trained with the best parameters, it was interesting to see which variables were the most important and how these affected the performance of the classifier: taking the fifteen most important variables (those with *importance* $> 0.02$), a new model was built and its performance compared to the original model (figure 17). This showed that essentially the reduced model achieved roughly the same performance. Finally, just for the sake of seeing first hand the trade-off between ensemble performance and computational cost, we investigated how the training time increases as the number of trees increases. The result of the analysis is shown in 17, in which we not only see that the fitting time increases linearly with the number of estimators, but also that the accuracy on the test does not increase proportionally, fluctuating between two values. While increasing the number of estimators may improve the accuracy of the model to a certain extent, the marginal gain in accuracy does not appear to justify the increase in fitting time. Instead, a trade-off between accuracy and computational cost should be considered when selecting the optimal number of estimators for the random forest model. Although this is not a problem in our dataset which has a small size, this might be particularly important in real-life scenarios.




(a) Errors as a function of the number of estimators




(b) Fitting time as a function of the number of estimators

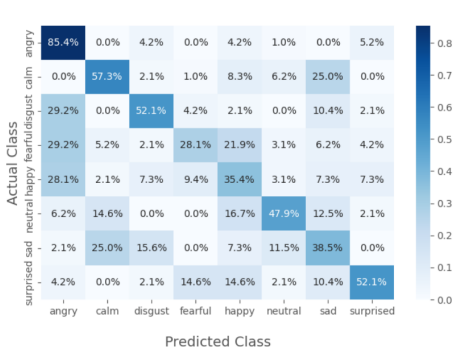
Figure 17: The figure displays two different plots aiming to compare a) the accuracy on the test set of the model trained on 61 features with the accuracy of the model trained on the 15 most important features as the number of estimators varies; b) shows the fitting time and the generalization error of the original model with respect to the number of trees.

**Gradient Boosting Machines** In order to perform a classification task on the emotion variable we decided to use Gradient boosting method as well, in particular LightGBM (Light Gradient Boosting Machines) because of its faster training speed and higher efficiency. It was decided to test some of the hyper-parameters of the model using a Randomized Search with 5 cross validation fold. The other hyper-parameters were set to *max depth*: -1 in order not to have a limit on the depth, *subsample for bin*: 200000 which is the default value, *objective*: multiclass, since there are 8 emotions, *reg alpha*=0.001, and finally *reg lambda*=0.001 which are the regularization terms on weights.

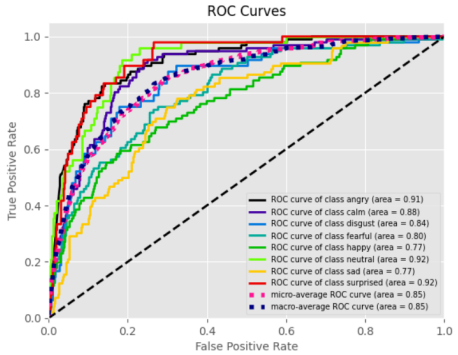
The best model found is defined by *num_leaves*: 4, *n_estimators*: 200, *min_data_in_leaf*: 300, *learning rate*: 0.1, *boosting type*: "gbdt". It's performances are shown in figure 18.

| Hyper-parameters | Tested values |
|---|---|
| boosting type | gbdt, goss |
| num leaves | 31,5,10,4,50,60,25 |
| min data in leaf | 100,200,300,1000,2000 |
| learning rate | 0.1,1,0.01,0.001 |
| n estimators | 10,20,30,50,100,200 |

Table 11: Parameters tested for LightGBM




(a) Confusion matrix of the GBM model on the test set




(b) ROC curve of the GBM model on the test set

Figure 18: Evaluation plots for lightGBM model

The confusion matrix of figure 18 (a) represents the results of the LightGBM model tested on the test set. As we can observe the best classification is performed on the emotion *angry* with 85.4% of correct classified instances. The worst classification involves the emotion *fearful*. The ROC curve of figure 18 (b) shows that the classification is far from being random.

### 2.1.5 Final comparison

Figure 19 gives us an overview of the more relevant models tested until now for each family of classifiers, along with their performances. We can clearly see that none of them was able to reach an accuracy on the test set higher than 0.5, but still all of them performed way better than the dummy classifier. We also have to take into account that an 8 class classification is not easy at all, especially if we also want to keep complexity as low as possible to avoid overfitting and computational resources consumption. An interesting point to highlight is that even more complex models such as Neural Networks didn't thrive, not only failing to achieve the best performer role, but also reporting decent levels of overfitting despite all the techniques applied to avoid it. Then, ensemble models as well didn't perform successfully, doing worse then almost every non ensemble model.

## 2.2 Advanced Regression

### 2.2.1 Gradient Boosting Regressor

In order to perform a regression task on the variable "frame_count" with the Boosting Gradient Regressor, the training set and the test set were scaled. 55 numerical variables were considered along with 7 categorical variable transformed with one hot encoding.

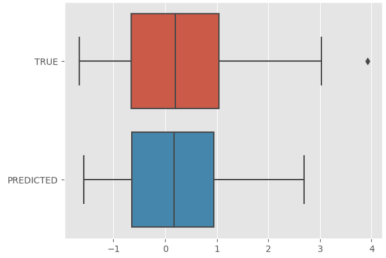



Figure 20: Boxplot of the target variable "frame_count" before and after the regression is applied to the test set.

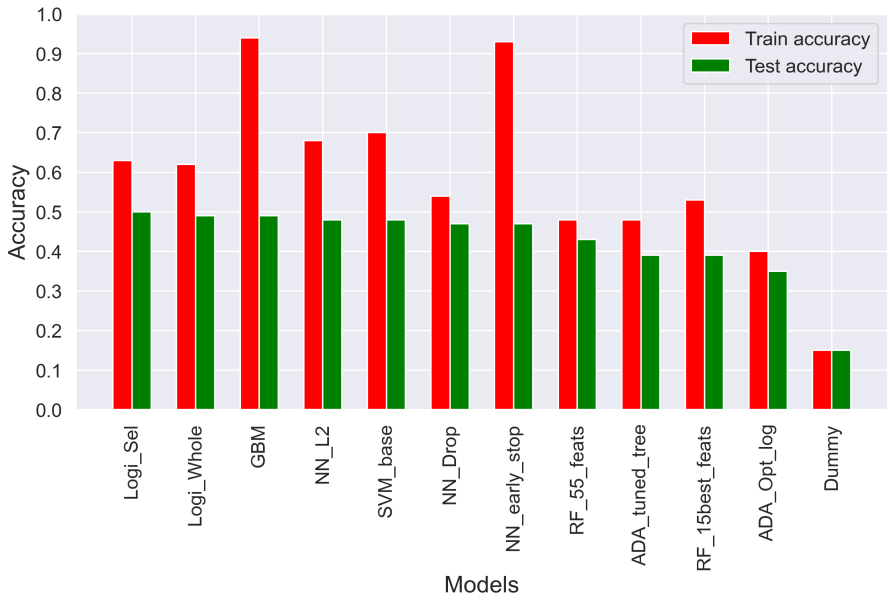
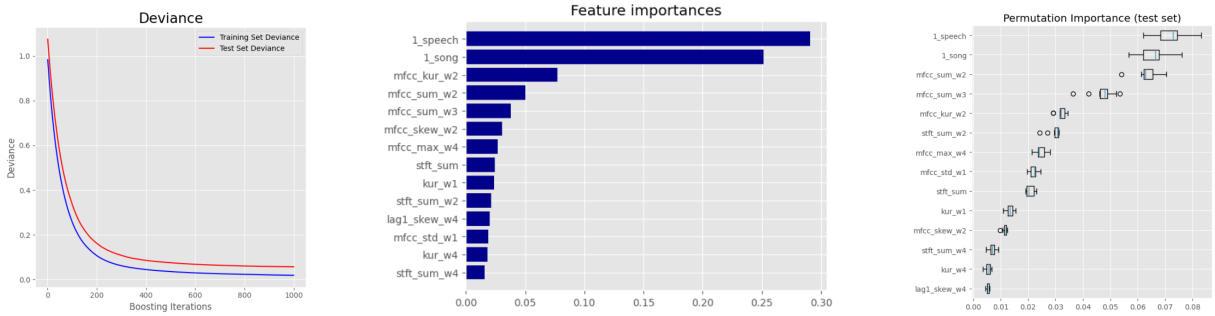



Figure 19: Performance comparison of main classifiers

Regarding the parameters, n_estimators was set to 1000 since the gradient boosting regressor is fairly robust to over-fitting. Max_depth was set to 3 which is the default value, min_samples_split was set to 5 , the learning_rate was set to 0.01 and lastly we decided that the loss function to be optimized was the squared error.

The model was trained on the training set and tested on the test set. To evaluate its performance it it observed that $R^2$ reached on the test set is 0.945 which is quite high, the Mean Absolute Error (MAE) is 0.1729, and the Mean Squared Error (MSE) on test set is 0.0564. As can be seen from figure 20 the results are satisfying as the two boxplots are almost identical.




(a) Deviance (b) Feature importance (c) Permutation importance

Figure 21: Evaluation and feature importance of the Gradient Boosting Regressor.

Figure 21 shows in plot (a) how the deviance (Mean Squared Error) decreases with the boosting iteration both on the training and in the test. Plot (b) shows the 14 most important features with their respective importance score. Lastly plot (c) shows the permutation importance of the features on the test set: permutation feature importance measures the increase in the prediction error of the model after the feature's values are permuted.

### 2.2.2 Support Vector Regressor

In order to compare two completely different nonlinear regression models, the same task of predicting "frame_count" target variable was performed using Support Vector Regression. In this case, both the training and test sets, with the same features, were scaled as well. To choose the best hyperparameters for the SVR model, a grid search was performed. The parameter grid consisted of three different kernels (rbf, linear, and poly) and three different values for both the regularization parameter C (0.01, 0.1, 1, 10) and the epsilon parameter (0.01, 0.1, 1, 10, 100). The grid search was carried out using 5-fold cross-validation and the negative mean squared error was used as the scoring metric. The best set of hyperparameters [7], with a corresponding negative mean

[7] `C = 1, epsilon = 0.1, kernel = ''linear''`

squared error of -0.025 on the validation set, were used to train the final SVR model. The results of the model (12)indicate that it is able to produce much more accurate predictions compared to the baseline model, that is the model always predicting the average value of frame_count. In particular, the model has a very high $R^2$ on both the training set (0.98) and the test set (0.97), which indicates that it can explain a large percentage of the variance in the dependent variable (frame_count). Additionally, the MSE of the model is very low on both the training set (0.02) and the test set (0.03), which indicates that the model can produce very precise predictions on the target variable.

| | $\mathbf{R^2}$ | **MAE** | **MSE** |
|---|---|---|---|
| **Baseline** | -0.061 | 0.875 | 1.092 |
| **GB regressor** | 0.945 | 0.173 | 0.056 |
| **SV regressor** | 0.970 | 0.127 | 0.031 |

Table 12: Summary table showing the differences between values for the three main metrics for regression; baseline model is the model always predicting the average value of *frame_count*.

Looking at the performance of both regression models, we notice that there are no substantial differences (table 12). Compared to the baseline model, the results of both models are significantly better. The baseline model has a negative $R^2$, which indicates that it is unable to explain the variance in the target variable, while the MSE and MAE of the baseline model are much higher than those of the two models, which indicates that it is far less accurate in making predictions. Overall, these results suggest that the trained SVR regression model and GB regressor are able to produce very accurate predictions on the target variable, and that the model significantly improves predictions compared to a baseline model.

# 3 Module 3

## 3.1 Data Understanding and Preparation

To ensure the suitability of our data for exploration and modeling we implemented a range of data preparation techniques. Initially, we had to set the sample rate to 16,000 to fit within our computational constraints, resulting in a reduced dataset of approximately 90,000 timestamps from the original 300,000. This approximation allowed us to efficiently work with the data while still preserving its essential characteristics.

Next, we applied a moving average smoothing technique with a window size of 12 to minimize noise and fluctuations in the time series, enhancing its clarity. Subsequently, we performed mean-variance scaling to normalize the data, enabling meaningful comparisons across different time series and improving the quality of subsequent approximations.

For the approximation step, we utilized Symbolic Aggregate approXimation (SAX) to simplify the representation of the time series. We performed two runs of SAX, one with 100 ($SAX\_s$) segments and another with 1000 segments ($SAX\_b$), to capture potential differences resulting from the granularity of the approximation. This approach facilitated efficient analysis and allowed to perform comparative assessments.

Additionally, we employed the Discrete Fourier Transform (DFT) with 64 coefficients to capture the frequency components of the time series. While SAX provided a simplified representation, the DFT analysis complemented our understanding by revealing frequency-related information.

By implementing these comprehensive data preparation steps, we successfully transformed and enhanced the time series data, setting the stage for insightful analysis and modeling.

## 3.2 Clustering

The clustering process involved the application of K-means algorithm using both Dynamic Time Warping (DTW) and Euclidean distance metrics on the three approximated datasets discussed in the previous section. However, for the Discrete Fourier Transform technique, DTW was not applied.

Table 13 provides a summary of the obtained results. It is important to note that this table does not intend to compare the results across different datasets, as they originate from different approximation techniques and pertain to distinct domains. Consequently, the comparisons made were solely within the same approximation technique.

| | **Best k** | **SSE** | **Silhouette** |
|---|---|---|---|
| **SAX_s (euclidean)** | 3 | 32 | 0.015 |
| **SAX_s (DTW)** | 2 | 7.5 | 0..25 |
| **SAX_b (euclidean)** | 3 | 1138 | 0.2 |
| **SAX_b (DTW)** | 3 | 686 | 0.22 |
| **DFT (euclidean)** | 5 | 5400 | 0.89 |

Table 13: Summary table for clustering evaluation. The best k was chosen in such a way to strike a balance between SSE and silhouette.

The observation that utilizing Dynamic Time Warping instead of the Euclidean distance metric for calculating similarities resulted in improved SSE for both SAX datasets is noteworthy. The smaller SAX dataset achieved a better SSE but a lower silhouette compared to the larger dataset. This suggests that using less approximated datasets leads to better cluster allocation in this particular case.

Figure 22 illustrates how the SSE and silhouette values change for clusters in the DFT and SAX_b cases when varying the value of k. As mentioned earlier, when adopting the Euclidean distance metric, the SSE is approximately halved compared to using DTW. Additionally, the silhouette values tend to be higher for the SAX datasets (waiting for SAX_b DTW results).

Two key observations can be made from the figure. Firstly, in both cases, the trend of the SSE does not exhibit a distinct elbow shape. Secondly, the silhouette trend for SAX_b appears to be less influenced by changes in k compared to the DFT approximation, which shows more fluctuations.

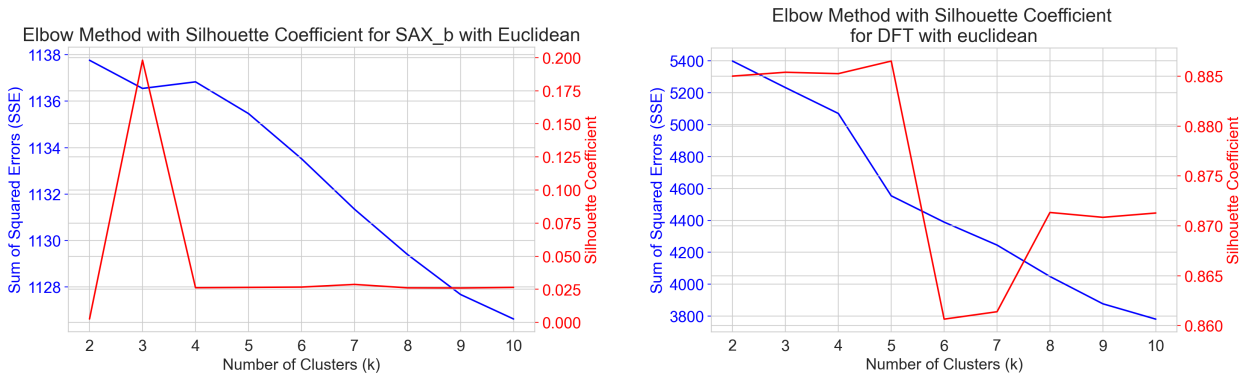


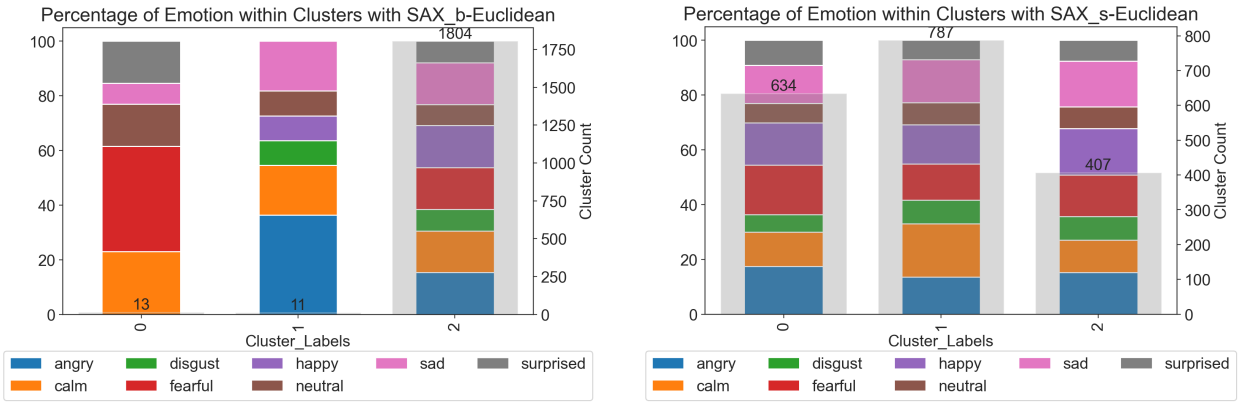

Figure 22: Impact of varying k on different clustering.

### 3.2.1 Content analysis and visualization

Regrettably, none of the approaches employed yielded significant results. Despite achieving high silhouette scores with the DFT method, the clusters remained highly imbalanced, with one overwhelmingly large cluster containing the majority of the data points. This imbalance persisted even when varying the value of k, indicating that it was not a result of a specific parameter setting. Similar results were observed when using the SAX_b technique.

A relatively more balanced distribution of clusters was only achieved with the SAX_s technique. However, these clusters were unable to effectively differentiate between emotions, resulting in a very low silhouette score. Figure 23 illustrates the different compositions of clusters obtained with SAX_b and SAX_s, utilizing the Euclidean distance metric, along with the corresponding cluster counts for each cluster.

These outcomes highlight the challenges encountered in accurately clustering the given data, indicating that the selected techniques and distance metrics were not effective in achieving meaningful cluster separation.




(a) (b)

Figure 23: Comparisons between 2 different cluster compositions

The three clusters depicted in Figure 23 (b) have been visually represented using three distinct dimensionality reduction techniques, enabling us to project the high-dimensional SAX_s dataset (with 100 dimensions) onto a 2-dimensional space. The choice to visualize these particular clusters stems from the fact that they do not form a single, large cluster encompassing a majority of the data points, as such a visualization would be uninformative.

Among the three visualization techniques employed, t-SNE yields the most favorable results. The t-SNE visualization exhibits a greater dispersion of data points, allowing for better highlighting of clusters 0 and 1, while cluster 2 shows less distinct separation. This outcome is consistent across PCA and IsoMap as well, although these techniques generate a somewhat noisier representation.

In conclusion, the t-SNE visualization presents the most advantageous portrayal, demonstrating improved separability between clusters 0 and 1 compared to cluster 2.

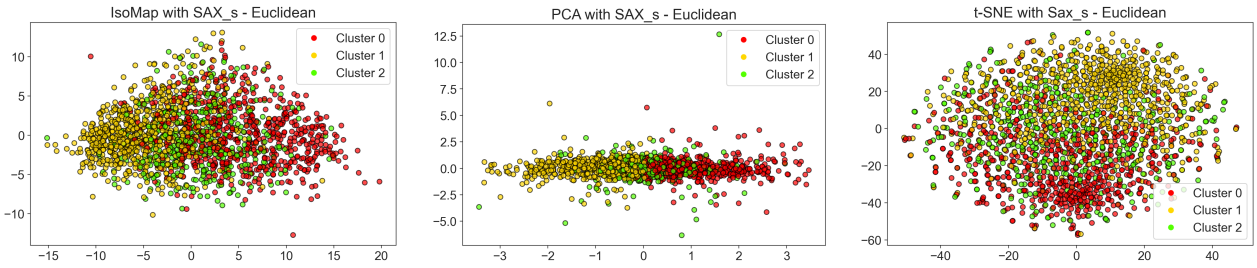



Figure 24: Cluster visualization in low dimensionality

## 3.3 Motifs and discords

This paragraph aims to briefly present motifs and discords discovered in the time series. Specifically, to get individual time series to solve this task, we made use of the results from the previous clustering task. Starting from the three centroids, we selected the three time series from the non-approximated dataset (without noise) that were most similar (i.e., least distant) to such centroids. This decision was made to ensure that motifs and discords we discovered were not only representative of the clusters previously identified but also actual instances from the dataset. By using real time series, we aimed to provide a more accurate and meaningful representation of motifs and discords in the data. Choosing an appropriate window size is crucial for accurate motif extraction, so an explanation of the rationale behind the decisions made is presented below.

We conducted many different experiments using various window sizes, encompassing a range of small, medium, and large sizes. The matrix profile was computed for each window size, and motifs detected were evaluated visually, trying to pay particular attention to whether such motifs represented complete patterns or if they were fragmented. This visual evaluation was complicated by the fact that we were working with the entire non-approximated time series. However, we recognized that selecting a proper window size is crucial to strike a balance between capturing meaningful patterns and avoiding over or under representation of motifs.

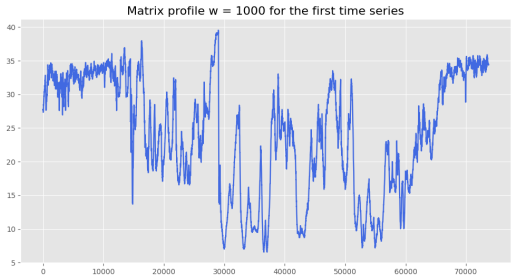
Since choosing an appropriate value for such long time series was not trivial, initially the strategy we opted for was domain-dependent: we tried selecting window sizes corresponding to time instants of 0.25 seconds, 0.5 seconds, and 1 second (audio files were 3-4 seconds long). Considering the sample rate used to extract the time series from the audio files (16000), this meant considering window sizes of w1 = 4000, w2 = 8000, and w3 = 16000, respectively.

The matrix profiles calculated with these windows are shown in Figure 25 (only the matrix profiles for the first TS are shown). It is noticeable that the local minima - the minimum distances between subsequences - are values significantly far from zero. This means that the motifs identified at these minima were not truly motifs, since they were actually quite different one another.

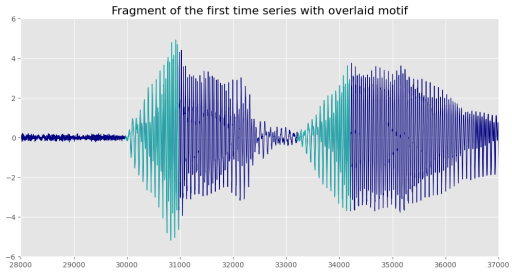

Figure 25: The figure displays three different matrix profiles obtained by computing them with window sizes corresponding to 1 second, 0.5 seconds, and 0.25 seconds.

Later, abandoning the previous approach, we tried various ranges of values for the window sizes: for small window sizes ($30 \leq w \leq 70$), we observed that the risk of capturing fragmented or partial motifs was quite high.

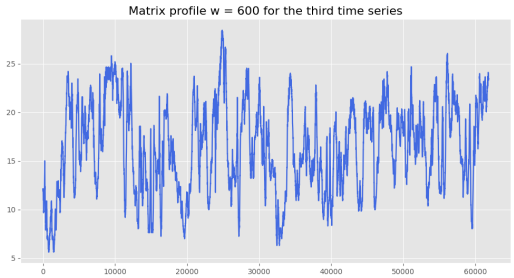
Although in the preprocessing step noise was removed, small window sizes increased the sensitivity to noise and local fluctuations. Therefore, larger window sizes were considered: at the other extreme, meaning for overly large windows (from 1500 on), another problem has been identified. In fact, using large window sizes made it challenging to distinguish motifs from the surrounding data, resulting in less discriminative patterns. In Figure 26 matrix profile and the corrisponding motifs of the two different time series are shown.



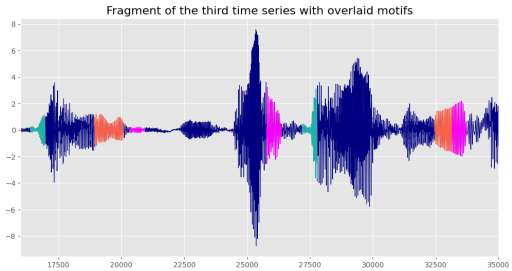

(a) Matrix profile for the first TS setting w = 1000




(b) First time series with motifs



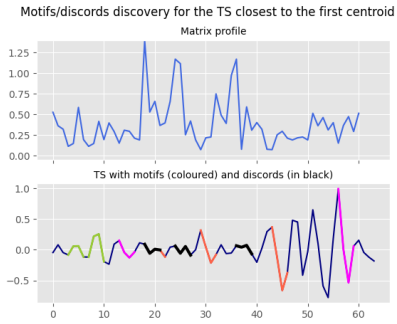

(c) Matrix profile for the third TS setting w = 600



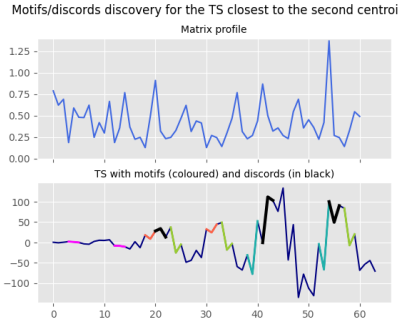

(d) Third time series with motifs

Figure 26: Matrix profile and time series with different overlaid colored motifs, for two different TS and two different widnow sizes. Note that the entire time series was not shown for readability purposes, but only a fragment corresponding to the motifs' timestamps.

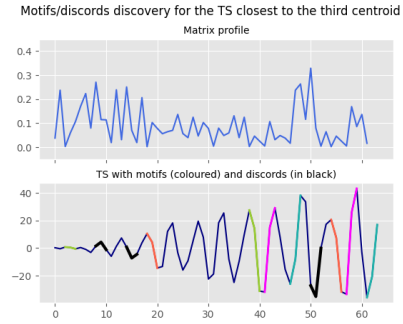
Finally, an attempt was made to extract motifs from approximated DFT time series as well. While our primary focus was on working on the original time series, we nevertheless tried to explore the potential use of the approximated time series, also to compare such motifs with shapelets (which have been retrieved from the approximated dataset). It is trivial to say that working with the approximated time series led to improvements in the matrix profile computation time, but it also helped in getting a clearer and denoised representation of the outcomes. Results are shown in Figure 27.




(a) TS closest to the first centroid




(b) TS closest to the second centroid




(c) TS closest to the third centroid

Figure 27: Each plot shows the matrix profile (above) and the approximated time series with its motifs and discords overlapped. Motifs are represented with thinner colored segments, while discords are represented by the thickest black lines. The window size for the matrix profile computation is always set to 4; parameters *ex_zone* and $k$ for discord discovery are set to 3, since three different peaks can always be detected from each matrix profile. One might notice that in plots c) the distances in the matrix profiles reaches 0, while in a) and b) they are slightly higher.

### 3.4 Classification with KNN

In order to perform classification with KNN on the time series, we decided to first use the dataset approximated with SAX with 100 segments. The KNN algorithm is applied on the training set and in figure 28 it is possible to see the accuracy scores of KNN with different metrics and different number of clusters. The three metrics (Euclidean, Manhattan and DTW) seem to follow the same trend on the training set: after about 15 clusters, the Euclidean and Manhattan lines are slightly under the DTW line. We then tested the classifiers with the values for k that performed better on the training set. We used a Randomized Search with 5 Cross Validation fold. Table 14 shows the parameter that we decided to test.

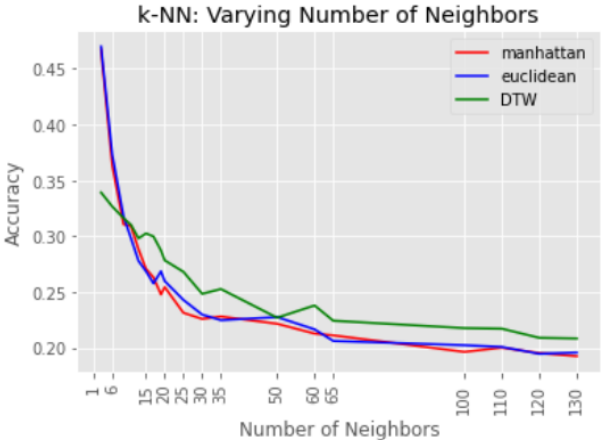
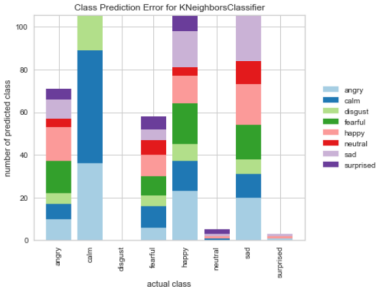



Figure 28: Accuracy score for different metrics and different values of k on the training set

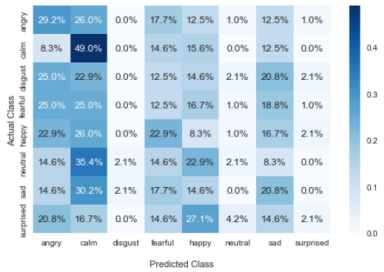
| **Hyper-parameters** | **Tested values** |
|---|---|
| Number of neighbors | 3, 6, 15, 17, 30, 35 |
| Metric | Euclidean, dtw_sakoechiba, Manhattan |

Table 14: Table shows the parameters tested with Randomized Search.

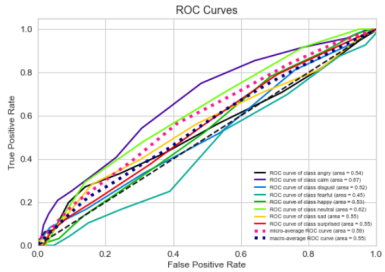
The optimal outcome was achieved using the following configuration: *number of neighbors*: 35, *metric*: 'dtw_sakoechiba'. The corresponding accuracy score obtained was 0.1920. Figure 29 displays the evaluation of this classification. The results are very poor, as can be seen in plot (a) and (b) the classifiers struggles to classify most of the emotion and there is an abundance of records classified as 'calm', while almost none in the class 'disgust' and 'surprised'. Another confirm of the poor performance of the classifier can be seen in 29 (c), since the ROC curves are below the bisector, meaning that even a random classifier would have been better.




(a) Class prediction error for KNN best classifier.




(b) Confusion matrix of KNN best classifier.

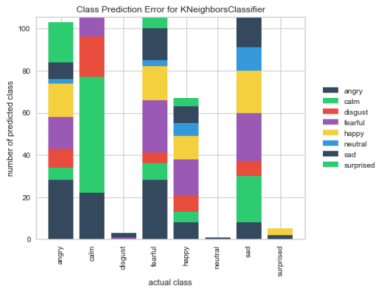



(c) ROC curves of KNN best classifier.

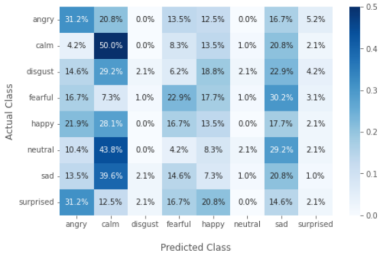
Figure 29: Evaluation of KNN classification with SAX approssimation and target variable emotion.

It was decide to try two other classification tasks with KNN, the first was to classify the variable emotion on the dataset obtained with the Discrete Fourier Transformation, with 64 coefficients and the second one was to classify on this last dataset the variable sex, in order to have a binary classification task. For the classification on emotion on this new dataset we repeated all the steps done with the SAX approximated dataset, except for the use of the DTW metric (observing the accuracy score on the training set, randomized search and result

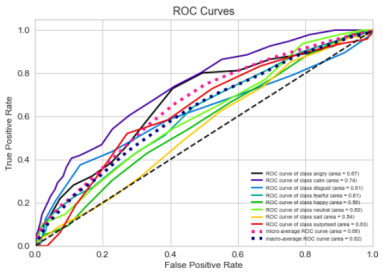
evaluation). The result of the classification are achieved using *number of neighbors:* 60 and metric: 'manhattan' with an accuracy of 0.230. The evaluation of this classifier is shown in figure 30.




(a) Class prediction error for KNN best classifier.



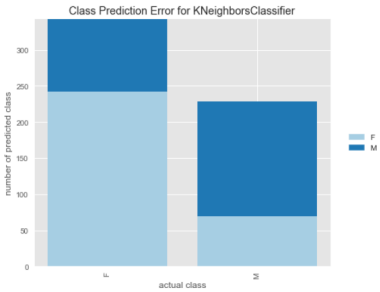

(b) Confusion matrix of KNN best classifier.



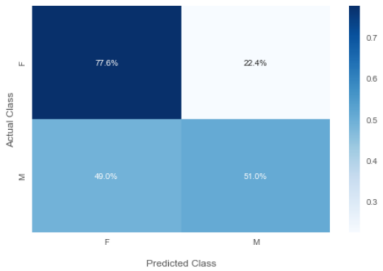

(c) ROC curves of KNN best classifier.

Figure 30: Evaluation of KNN classification with Discrete Fourier transformed datasetand target variable emotion.

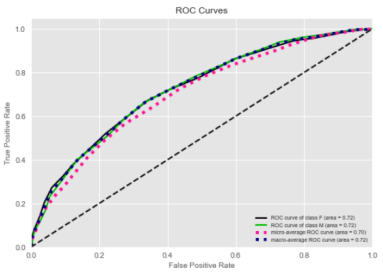
Lastly the binary classification was carried out, always following the same approach and the result were better: the best classifier was achieved using *number of neighbors:* 35, metric: 'manhattan'. The accuracy obtained was 0.6253. Figure 31 displays the evaluation of this classifier.




(a) Class prediction error for KNN best classifier for binary classification.




(b) Confusion matrix of KNN best classifier for binary classification.




(c) ROC curves of KNN best classifier for binary classification.

Figure 31: Evaluation of KNN classification with Discrete Fourier transformed dataset and target variable sex.

Overall, the classification tasks of the variable emotion are not satisfactory as there are problems in allocating all the classes and as can be seen in the ROC curves, the performances of the classifiers are very close to the bisector, meaning that the classification is close to being a random one, and sometimes even worst, when the curves are below the bisector. The classification on the Discrete Fourier transformed dataset are slightly better than the one with the SAX approximation, but still not satisfactory.

### 3.4.1 State of the art Time Series Classification

In our study, we employed state-of-the-art classification techniques, namely Rocket and MiniRocket transformations, in combination with a Ridge Classifier, to analyze their performance. While the authors recommend using simple linear models as classifiers, we also investigated the efficacy of non-linear classifiers such as Decision Trees and ensemble methods such as Random Forest. The results obtained from these alternative classifiers were comparable to those achieved by the Ridge Classifier. Consequently, we retained the Ridge Classifier due to its advantageous characteristics, such as fast training time and simplicity.

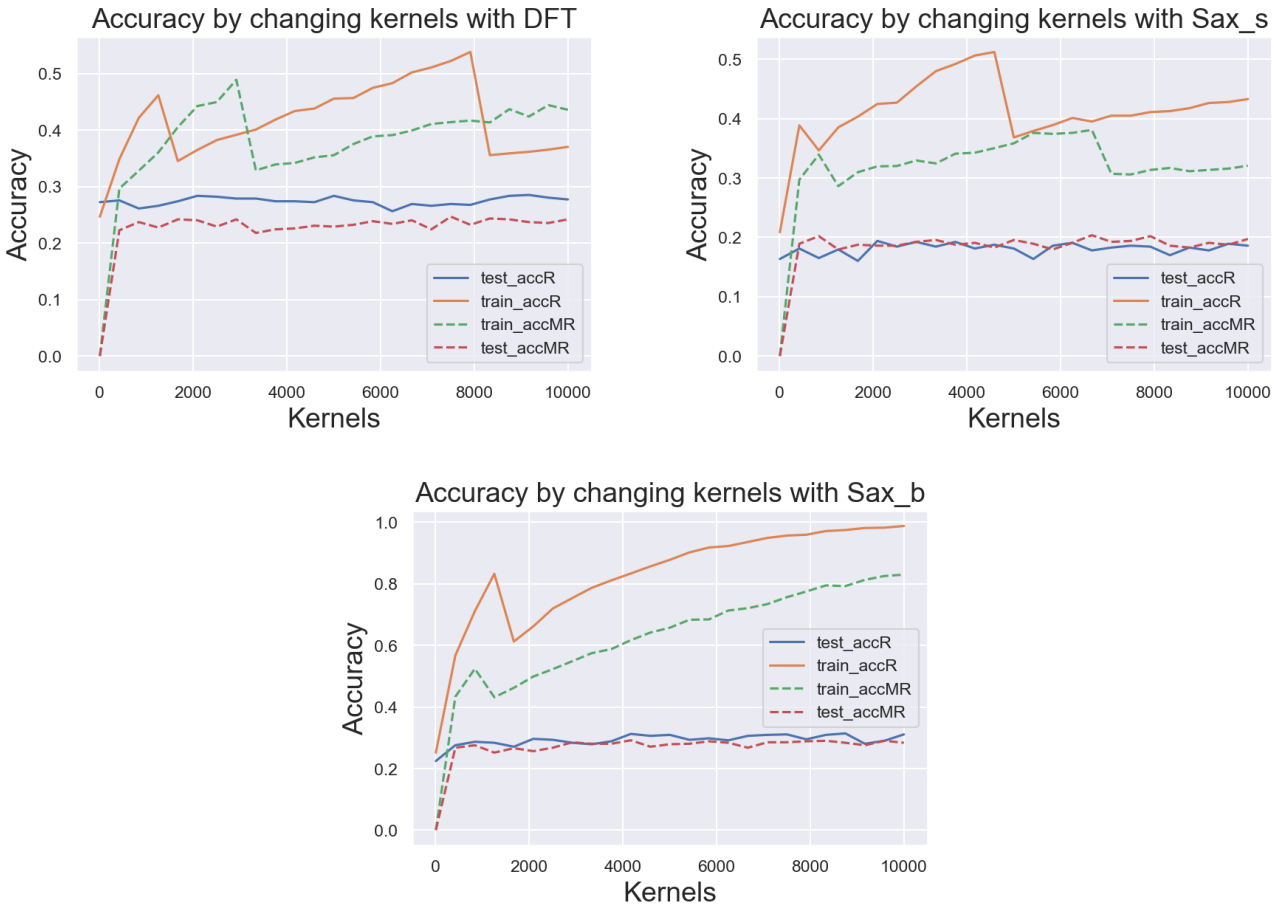
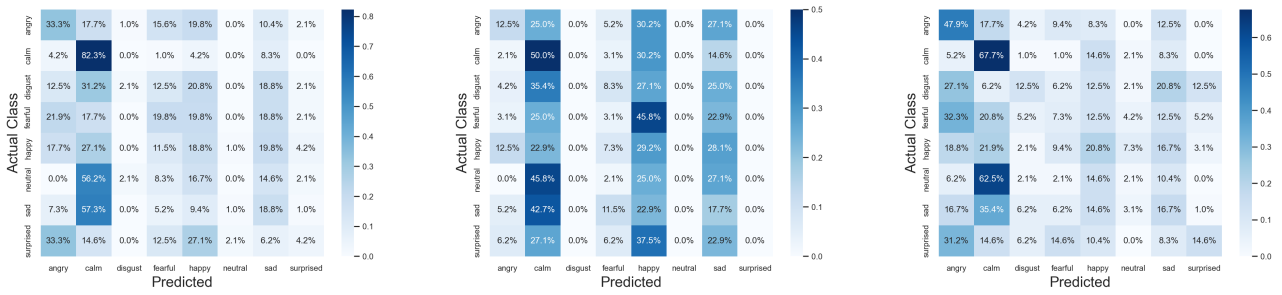



Figure 32: Train and test accuracy for different values of kernels with Rocket(R) and MiniRocket(MR)

Additionally, we explored the impact of varying the number of kernels used in both transformers. The effect of this variation is illustrated in Figure 32. Notably, the test accuracies exhibited slight fluctuations, remaining marginally below 0.30 for Rocket and approximately 0.25 for MiniRocket in the DFT dataset. Conversely, the train accuracies displayed an increasing and decreasing trend that persisted even after surpassing 10,000 kernels. However, the achieved accuracy values never exceeded 0.55-0.60. This phenomena also happens with SAX_s and SAX_b, with the first having a test accuracy fluctuating around 0.20 for both Rocket and MiniRocket, while the second around 0.30 for both Rocket and MiniRocket. Moreover, Sax_b suffers from serious overfitting right after few thousands kernels.

Based on these findings, it can be inferred that augmenting the number of kernels does not necessarily lead to improved classification performance. This outcome is significant since it allows for the utilization of a considerably lower number of kernels, resulting in smaller transformed datasets and less complex models.




(a) Confusion matrix for DFT with Rocket. Overall accuracy = 0.27

(b) Confusion matrix for SAX_s with Rocket. Overall accuracy = 0.18

(c) Confusion Matrix for SAX_b with Rocket. Overall Accuracy = 0.30

Figure 33: Confusion Matrices comparison for Rocket transformation with Ridge Classifier.

In Figure 33 the classification results for Rocket with 1,000 kernels are depicted, denoting few interesting

details. First of all, with SAX_s some classes as neutral, surprised and disgust are not even classified once, concentrating most of the prediction between happy, calm and sad. The situation slightly improves with DFT, where only neutral has very few prediction and gets a even a bit better with SAX_b.

Another observation is that, obviously, less approximated datasets with SAX are able to achieve better classification performance, but the DFT is able to achieve similar performances of SAX_b, the latter one having 1000 dimensions, while the DFT only 64.

### 3.4.2 Shapelets

The Shapelet Transform algorithm was used to extract the most discriminative shapelets from the dataset obtained with the Discrete Fourier Transformation, having 64 coefficients. The algorithm automatically selects a window size of 5.

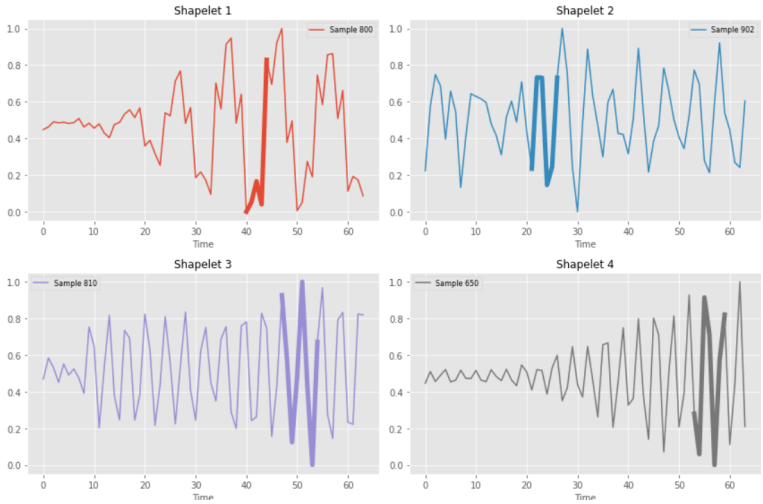



Figure 34: Plot of the four most discriminant shapelets

| index | start | end | mutual info | emotion |
|---|---|---|---|---|
| 800 | 40 | 45 | 0.1163 | happy |
| 902 | 21 | 27 | 0.1056 | calm |
| 810 | 47 | 55 | 0.10512 | sad |
| 650 | 53 | 60 | 0.1019 | sad |

Table 15: Information about the four most discriminant shapelets

In order to perform classification using shapelets, it was decided to choose as the target variable the *sex* of the actor. The number of shapelets per size was obtained using the `grabocka_params_to_shapelet_size_dict` function of *Tslearn* and resulted in 4 shaplets of size 19.

We then proceed to build the model, using the function *ShapeletModel* and 'sgd' as the optimizer. After fitting the model on the training data, the prediction on the test set were observed. Although the accuracy of this classifier reaches 0.5, it is not meaningful since all observations are classified as sex Male.

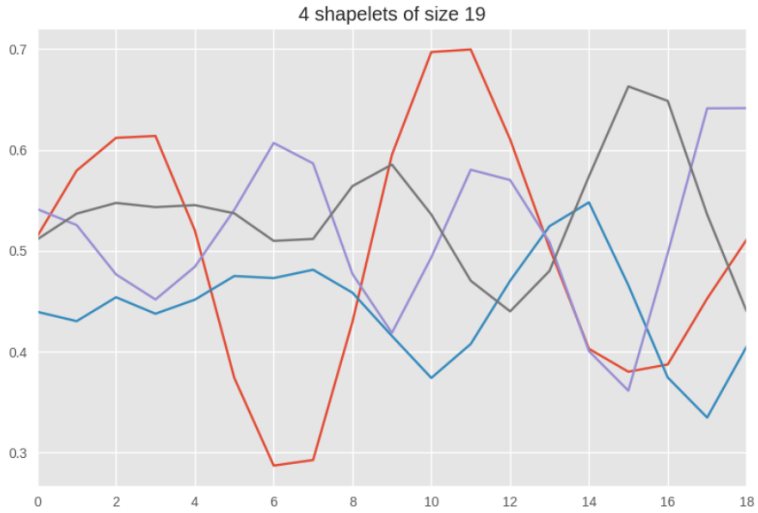
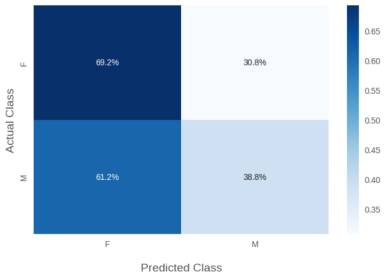



Figure 35: Plot of the different discovered shapelets

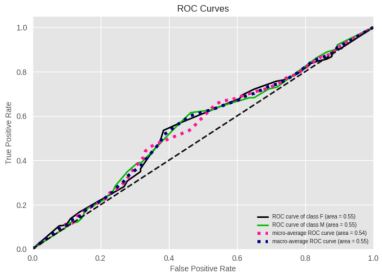
Finally we decided to perform a binary classification task with *sex* as target variable on the dataset transformed with shapelets, with a Decision Tree classifier. Even though the accuracy of this classifier is similar to the previous one, being 0.54, it does not classify all the time series in the same class.

In Figure 36 it is possible to see the classification of the records in the two classes and the evaluation of this classifier. As can be seen in the ROC curve (Figure 36 - (c)) the classifier is close to the performances of a random one.


(a) Class prediction error for Decision Tree classifier.




(b) confusion matrix for Decision Tree classifier.




(c) ROC curves for Decision Tree classifier.

Figure 36: Evaluation of the classification of variable 'sex' with dataset transformed and Decision Tree classifier.

Overall, none of the classification tasks performed are satisfactory, this can be due to the extreme approximation of the dataset (the dataset has just 64 coefficients) that we performed in order to significantly decrease the computational cost.

**Motifs and shapelets comparison** Finally, we compared the extracted shapelets (Table 15) with the motifs from the previous section. The distance between each shapelet and each motif from the three representative time series used in the motif discovery task was calculated. The resulting table 16 shows the best matching motif, i.e. the least distant, for each shapelet.

| Shapelet Index | Best Matching Motif | Distance |
|---|---|---|
| 800 | motif starting at index 3 of TS idx 298 | 24.9 |
| 902 | motif starting at index 18 of TS idx 1161 | 8.5 |
| 810 | motif starting at index 13 of TS idx 298 | 20.2 |
| 650 | motif starting at index 20 of TS idx 249 | 144.8 |

Table 16: Comparison of Motifs and Shapelets: optimal matches

# 4 Module 4

The purpose of this final section is to present the results obtained for the explainability task. To achieve this, we have chosen SVM classifier (`gamma = 0.00191, kernel = rbf, C = 31, class_weight = balanced, probability = True`) as the model for the explanation, as discussed in Section 2. We'll first present a local explanation and then a global one.

## 4.1 Local explanation

To explain our SVM classifier we assumed that we do not have access to the original training set, meaning that our focus was on generating explanations using the test set alone. This scenario aims to mimic a real-world situation where the SVM model is actually treated as a "black box" created by someone else, and we aim to explain specific instances without knowledge of the original training data. To obtain meaningful insights, we focused on instances where the SVM classifier exhibited significant misclassification. We examined the confusion matrix (9) and identified the most frequently misclassified emotions:

- In 28.1% of cases, when the true class was *sad*, SVM classified it as *calm*.
- In 27.1% of cases, the classifier misclassified instances as *angry* instead of the true class *disgust*.
- For instances belonging to the *fearful* class, the classifier misclassified them as *angry* 18.8% of the time.

Initially, we decided to generate counterfactual explanations without any constraints on the indexes of the features to be modified. In other words, we allowed the algorithm to freely choose at most one variable to modify. The first five instances we chose to explain are the first five instances from the test set where the classifier predicted class *calm* despite the ground truth being *sad*. The table below presents the counterfactuals we obtained allowing for the variation of one feature.

| Idx | Actual | Predicted | Modified feature | Original value | New value | Distance | New pred |
|---|---|---|---|---|---|---|---|
| 21 | sad | calm | sc_q95_w3 | -0.5604 | -2.1652 | 1.6048 | sad |
| 80 | sad | calm | mfcc_kur | -0.7102 | -1.7019 | 0.9917 | sad |
| 81 | sad | calm | mfcc_kur | 0.0377 | -1.7019 | 1.7396 | sad |
| 82 | sad | calm | sc_std_w3 | -1.3616 | 1.5675 | 2.9292 | sad |
| 86 | sad | calm | stft_sum_w3 | 1.0920 | -1.8718 | 2.9639 | sad |

Table 17: Each row of the table represents a counterfactual explanation for a single instance, identified by *Idx*, providing information about the modified feature, original value, new value, distance between them, and the new predicted emotion. The actual emotion is *sad*, the model initially predicted *calm* and after the modification, the new prediction is correct, being *sad* and showing that the local explanations were correct.

Afterwards, we generated local explanations for the other misclassified instances in the test set (Table 18). This time, our attention was directed towards the classes that were consistently misclassified as *angry*, despite having different actual classes, which was a common mistake as can be seen from SVM confusion matrix.

| Idx | Actual | Predicted | Modified feature | Original value | New value | Distance | New pred |
|---|---|---|---|---|---|---|---|
| 50 | disgust | angry | mfcc_mean_w3 | 0.4912 | -1.0886 | 1.5799 | disgust |
| 101 | fearful | angry | mfcc_max_w4 | 1.3361 | -1.3729 | 2.7091 | fearful |
| 72 | happy | angry | lag1_skew_w4 | 0.6994 | 10.8311 | 10.1317 | happy |

Table 18: Three different counterfactual explanations for three test instances that were misclassified as *angry*.

However, it is not so easy to understand the continuous features reported in the table above, as they often lack direct interpretability. To address this issue, we made an attempt to force the generation of counterfactual explanations using three categorical variables only, which we believe they provide more comprehensible insights. Such features are: *sex, emotional intensity* and *repetition*.

| Idx | Actual | Predicted | Modified feature | Original value | New value |
|---|---|---|---|---|---|
| 444 | angry | happy | sex | M | F |
| 11 | calm | sad | sex | F | M |
| 413 | fearful | angry | sex | F | M |
| 180 | happy | angry | sex | F | M |

Table 19: Four different counterfactual explanations showing how variable *sex* is affecting their predictions. For instance, considering instance 444, we notice that the actual emotion is *angry*, but the model predicted *happy*. The feature modified is *sex* which is changed from Male (M) to Female (F): it suggests that changing the gender from Male to Female would result in the model predicting *happy* instead of *angry*. It indicates that the gender feature has some influence on the model's classification of *these* instances.

| Idx | Actual | Predicted | Modified feature | Original value | New value |
|---|---|---|---|---|---|
| 575 | surprised | fearful | emotional intensity | normal | strong |
| 45 | disgust | angry | emotional intensity | normal | strong |
| 399 | sad | calm | emotional intensity | strong | normal |
| 282 | happy | neutral | emotional intensity | normal | strong |

Table 20: Four different counterfactual explanations highlight that there are instances in which simply changing the value of the *emotional intensity* from strong to normal or vice versa completely alters the prediction of the black box.

## 4.2 Global explanation

Our global explanation was conducted using a decision tree with a maximum depth (*max_depth*) of 2. The objective of this phase was to obtain an interpretable representation of the predictions generated by the black box. The decision tree was trained using the predictions of the SVC classifier as the target variable. The maximum depth of 2 was chosen to keep the tree simple and easily interpretable, enabling a clear understanding of the underlying decision rules. The decisions made at each node of the tree are based on simple logical rules that can be easily understood.

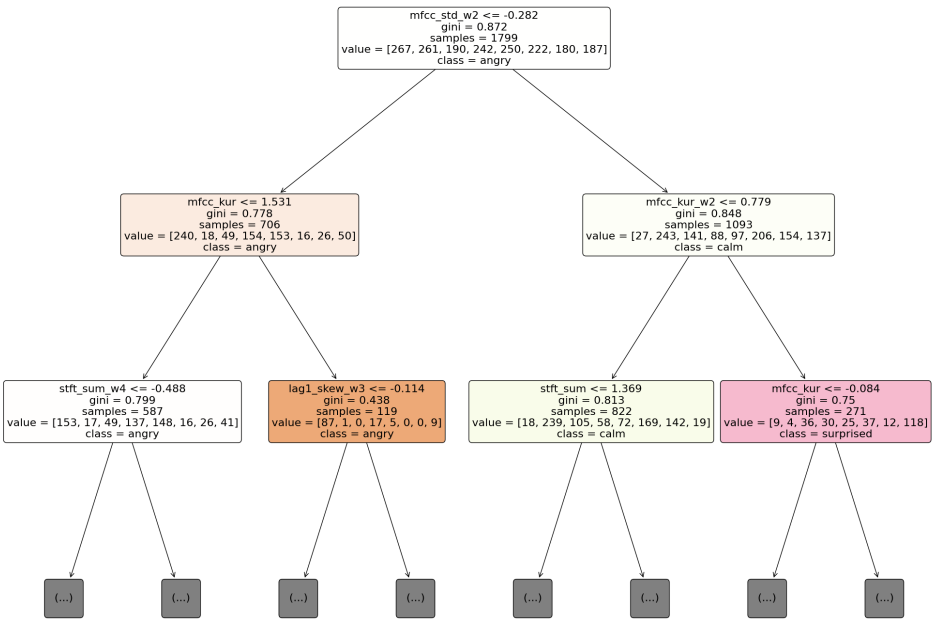



Figure 37: Decision tree explaining the black box.

To further enhance the readability of the decision tree, we decided to include a representation created with the `dtreeviz` library. Figure 38 incorporates interesting information: it shows pie charts of leaf nodes, where the color of the pie chart majority slice gives the leaf prediction, and stacked histograms of internal nodes as well. Histograms illustrate feature-target space. For instance, we notice that in the root, samples with target emotion *calm* are clustered at the low end of *mfcc_std_w2* feature-space while samples with target variable *sad* and *surprised* are clustered at the high-end.

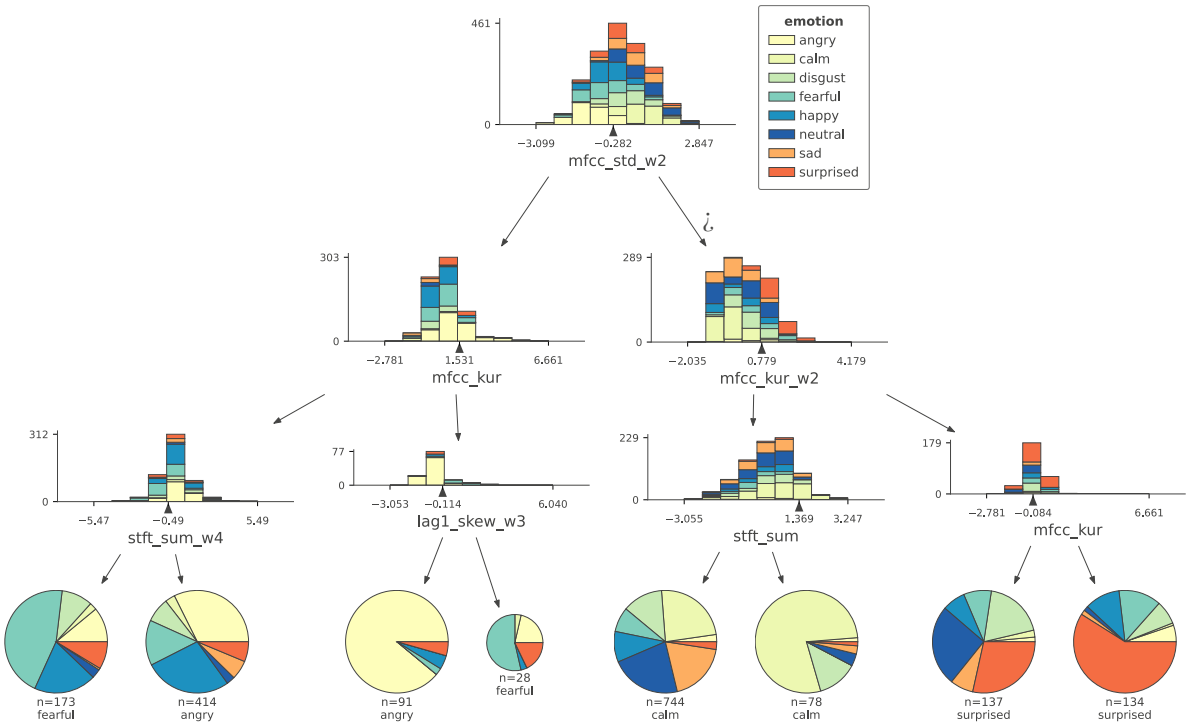



Figure 38: DTREEVIZ visualisation of the previous decision tree