
UNIVERSITY OF PISA

DEPARTMENT OF INFORMATICS

Master Degree in Data Science and Business Informatics

# Data Mining 2

Matteo Rofrano - Gabriele Gori - Francesco Lanci Lanci

Academic Year 2022/2023

# Contents

# Chapter 1: Data understanding and preparation

The dataset we will use in this project contains 129 variables which are those selected after the data preparation task that will be described in this chapter.
Our analysis and every plots and tables that will be presented in this chapter refer to the training dataset only, while the test dataset has been modified at the end to be coherent with the training dataset.
First of all, since we have many variables (434 in the beginning) we decided to remove all that variables with only one value. In this way, we have eliminated variables such as "filename" and "modality" which do not bring any kind of information useful for the analysis. Then, we decided to drop even those variables which variance less than or equal to 15%. After this initial step, since we want to perform in the further chapters classification and regression tasks, we decided to drop all the variables that are highly correlated[1] in order to avoid multicollinearity. We have done this step for the Spearman and the Pearson coefficient of correlation.
In the end, to allow our supervised learning models to deal with variables that present a more normal distribution shape we have built a function that applies for all those variables with only positive values a skew test that checks if the variable has a skewed distribution with respect to a normal one shape. If the p-value of the test on the variable was too low[2] we are able to reject the null hypothesis that the variable has a normal distribution (this means that the variable has a skewed distribution) so we apply a logarithmic transformation to that variable. The effect of this approach can be seen from the QQ plot of the frame_count variable before and after the transformation.

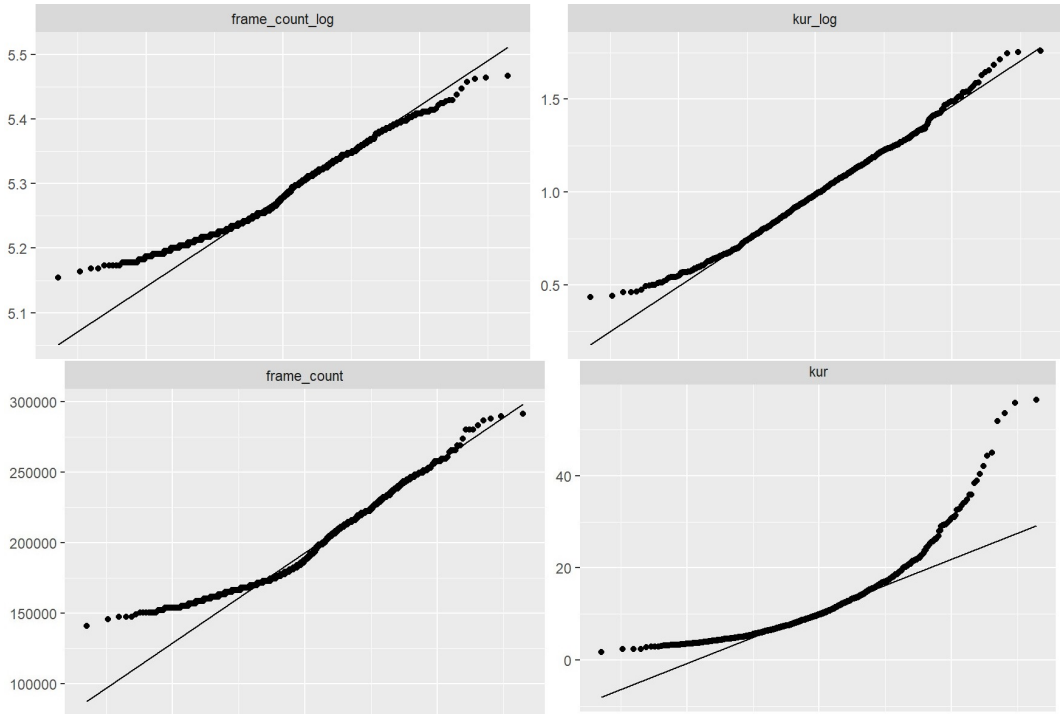



Figure 1.1: QQ plot log vs no log

The QQ plot shows the cumulative quantile distribution of a variable with respect to the cumulative

[1] In this report we assume that a correlation higher or equal to 90% brings to multicollinearity
[2] p-value$<$0.05

quantile normal distribution which is represented as the diagonal of the plot.
We can see that for "frame_count" we have a little change in the distribution but for "kur" it is more evident that after the transformation the distribution assumes a normal shape.

After the data preparation step, we focused on finding interesting patterns and information about the "emotion" and "frame_count_log" variables since they will be our target variables for the classification and regression tasks.

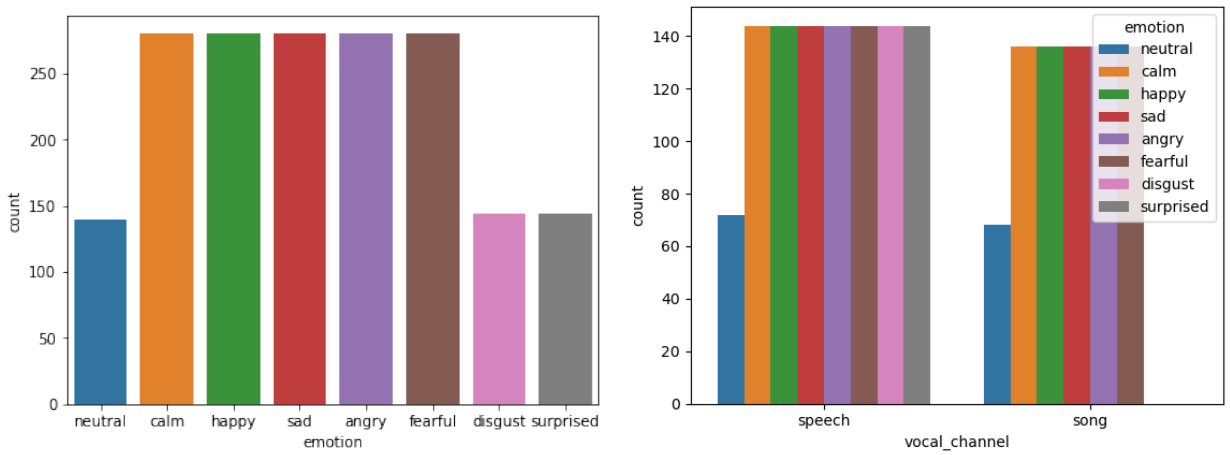



Figure 1.2: Emotion countplots

From the first countplot can be seen that the majority of the observation are relative to "calm", "happy", "sad", "angry" and "fearful" while there are some minority class such as "neutral", "disgust" and "surprised" so our supervised models could have worse performance on the minority classes. The second countplot shows that the song observations does not have surprised and disgust emotions and moreover we can see that we have more speech observations than song observations.

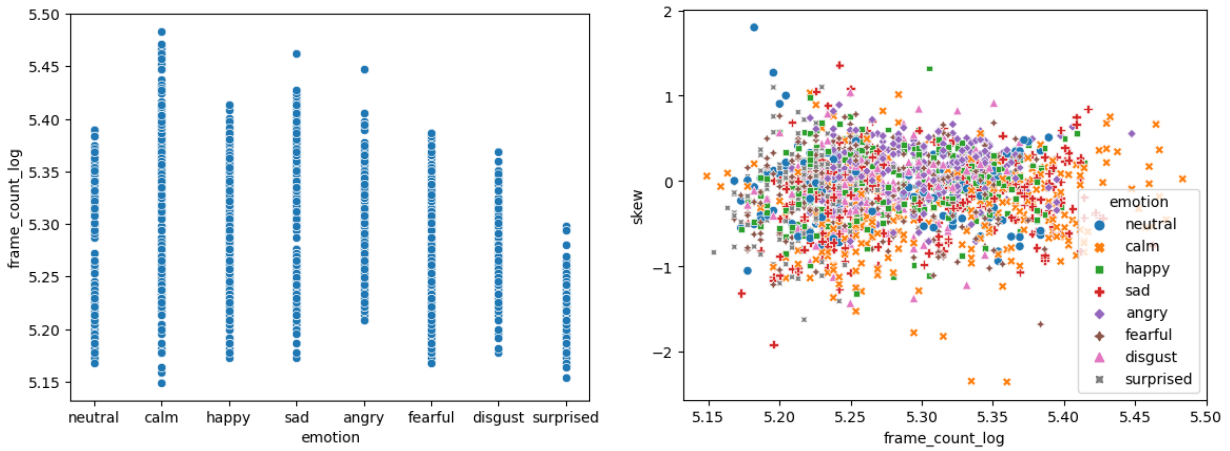



Figure 1.3: Emotion scatterplots

The first scatterplot shows that for observations with calm emotion, we could have audio samples with a longer duration while for surprised emotion our audio sample is likely to have a shorter duration. The second scatterplot shows that for audio samples with a positive skewed wave form of the audio we can have neutral emotion while for negative skewed waves, we can have calm emotion.

# Chapter 2: Imbalanced learning

To perform the imbalanced learning task we decided to use both the training dataset and the test dataset. Our target variable is the vocal channel and our classification algorithm will be the decision tree.
First of all, we have transformed all the categorical variables in dummies by using one-hot encoding. So our target variable is 0 if the vocal channel is a song for that observation and 1 if it is a speech observation.
Since our target variable is not imbalanced we decided to remove observation of the minority class, in this case our minority class is song so we have removed random song observation since the representation of song in the dataset is only the 3.57% (we have only 40 observation that are songs). We have repeated this process even for the test dataset and in this case the representation of songs is the 3.74%, in this way we have a sort of stratification. In this way the training set is representative of our imbalanced population that we want to learn that is represented from the test set.
In this chapter will be presented the main results we got after testing different algorithm used for the imbalanced learning task, in particular we will present an undersampling algorithm, an oversampling algorithm, a combined method that use both oversampling and undersampling and a tuning for the class weight parameter of the decision tree. Below is presented a visual representation of our dataset obtained from a PCA with 2 components.

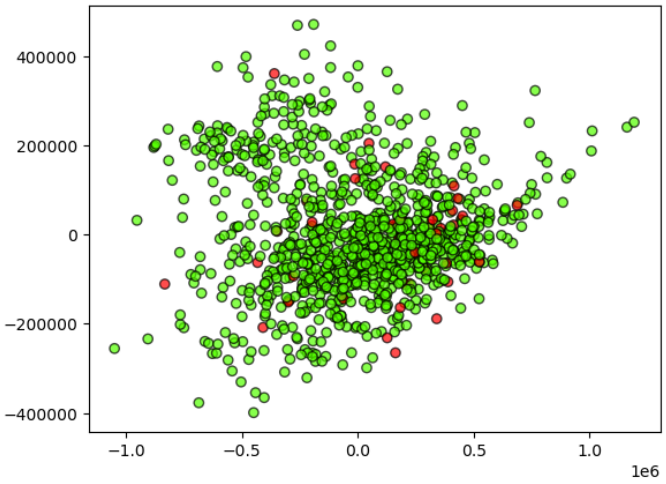



Figure 2.1: Imbalanced dataset PCA

## 2.1 Base model

In order to compare the different imbalanced learning techniques we first defined a base model. We obtained it by making hyperparameter tuning using a randomized search over the following hyperparameters: "max_depth", "min_samples_split", "min_samples_leaf" "min_impurity_decrease" and "criterion". To avoid some further model to get better performance just thanks to a different set of hyperparameters or a different range of values for the hyperparameters we decided to use the same randomized search structure for all the models evaluated in this chapter. The results are presented at the end of this chapter (table 2.1) while the ROC curve of the base model is the following:

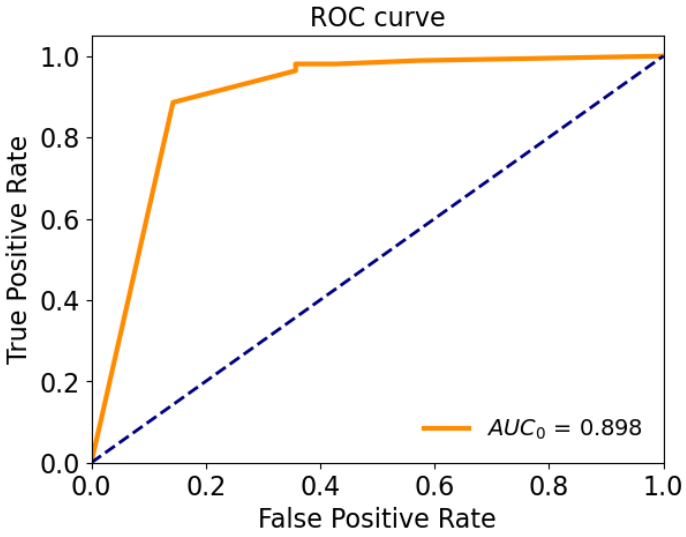
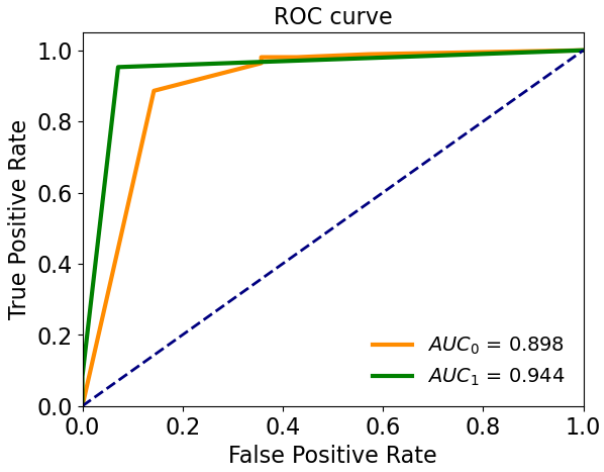



Figure 2.2: Base model ROC curve

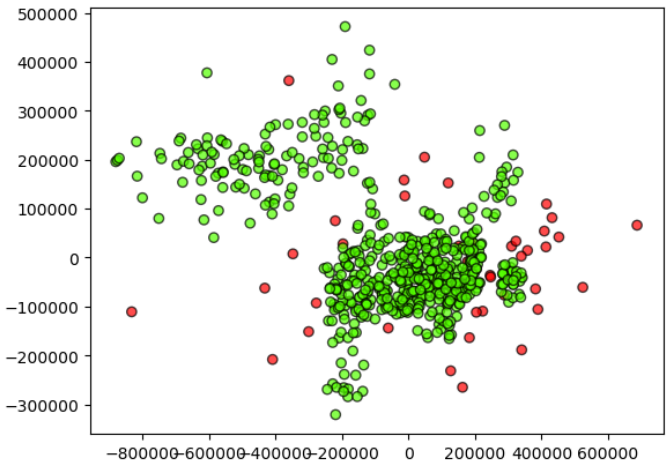
## 2.2 Undersampling

In this section are presented the result obtained from the Undersampling technique that gives us the best results. First of all we tried the Tomek-links method but for our dataset, it brings very little improvement with respect to the base case so we decided to apply the Edited Nearest Neighbours method. We tried this approach with different numbers of neighbours and our results suggested that for our dataset this method allows our model to get the better performance with number of nearest neighbours equal to 10. Thanks to this method we get better improvement of the recall metrics so to get even more tangible improvements we have chosen to use the Repeated Edited Nearest Neighbours. In this case our model has got the best performance with a number of nearest neighbours equal to 12. In the table at the end of the chapter (table 2.1) are presented the results while below there is the ROC curve comparison with the base model and the training set PCA after applying the RENN method.

From the result we can see that the precision on the class 0 has decresed with respect to the base model but now we have a very high recall on it so this means that our model is able to correctly classify almost the total of the observation that are truly song even if it tends to produce high false positive for this minority class while in the base model almost the 50% of the times that the correct prediction was song it predicted speech.



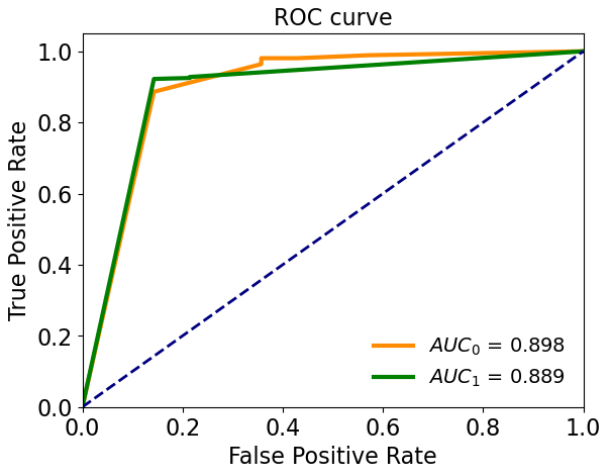

(a) RENN ROC curve



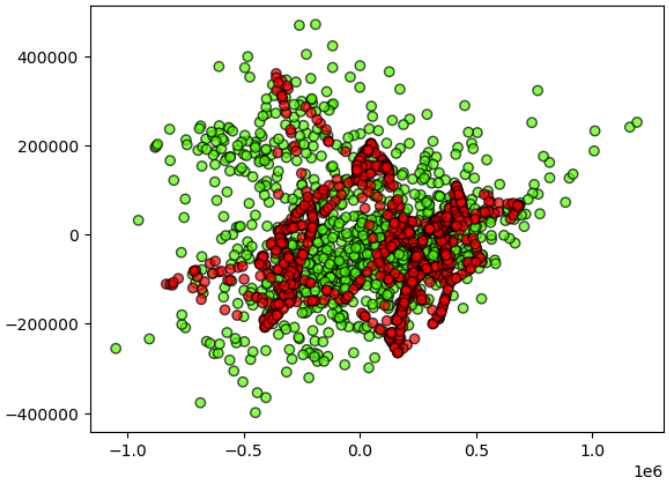
(b) RENN PCA

## 2.3 Oversampling

Regarding the oversampling methods we have tried to use both the SMOTE algorithm and the ADASYN. The SMOTE just like the Tomek does not bring any interesting improvement to the learning of the minority class while the ADASYN has worked better with our dataset. In particular after having tried different value for the n_neighbour hyperparameter has turned out that it works good with its default value of 5 with our training dataset. After the hyperparameter tuning of our decision tree model on the rebalanced training set. The comparison with the base model's roc curve is presented below.




(a) ADASYN ROC curve



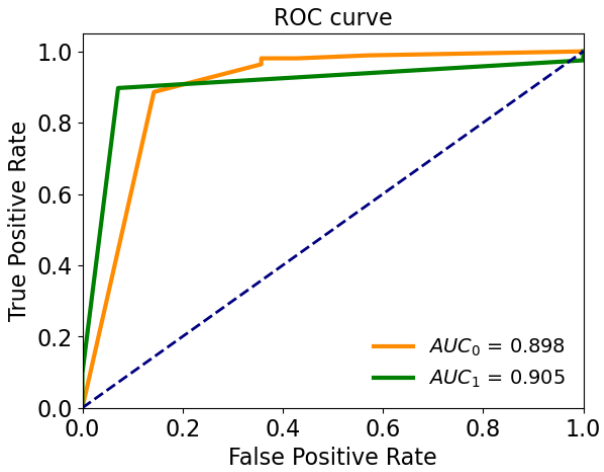

(b) ADASYN PCA

From the results showed in table 2.1 it can be seen that we got an improvement on the recall of the song class but the precision is terribly lower with respect to the base model and the Repeated edited nearest neighbours as the F-score. So this approach allows us to be more confident that when the class predicted is song in nearly the 80% of the case we have made a right prediction but it is way worse than the Repeated edited nearest neighbours. Moreover this method since generate synthetic observation could make the training set not representative of the test set and this makes avoiding overfitting more difficult in fact the difference between the accuracy of the validation set and the test set is around 6%. The effect of the ADASYN on the training set can be addressed from the plot above.

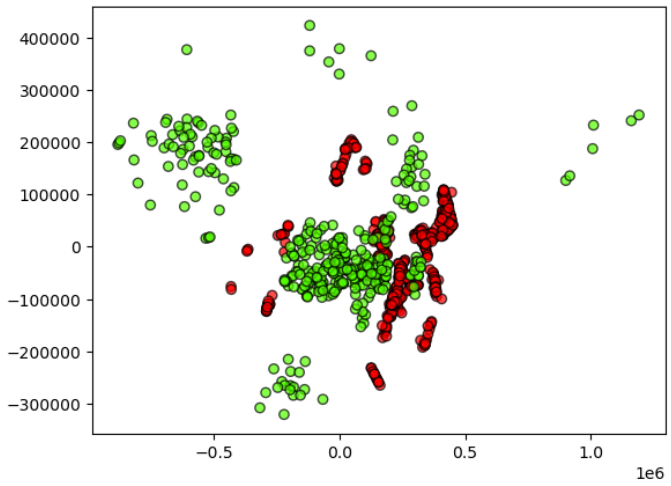
## 2.4 Combined oversampling and undersampling

In this section we analyze the 2 different algorithms used to deal with unbalanced classes that combines first an oversampling technique and then an undersampling one. In particular, we have used the Tomek+Smote introduced by Batista et al (2003) where it is applied first the SMOTE on the minority class and then it is used the Tomek links to choose from the majority class random data and if the nearest neighbor is an observation of the minority class we remove the Tomek link. Since both Tomek and Smote does not perform well on our dataset as expected this method does not provide great result with respect to the other method we tested that is the SMOTE-ENN method developed by Batista et al (2004). This method use first the SMOTE to generate observation of the minority class and then apply the ENN algorithm so after having calculated the nearest neighbours of an observation if the class of the observation and the majority class from the observation's K-nearest neighbor is different, then the observation and its K-nearest neighbor are deleted from the dataset. In our code we find that the best nearest neighbour is 16 and the sampling strategy for the ENN algorithm has been set

to "all" as suggested by the author of the paper.
The classification report in table 2.1 shows that the model trained on the new training set obtained after applying the SMOTE-ENN algorithm allows us to reach the same performance as the Repeated Edited Nearest Neighbours algorithm but at the same time we get worse performance in terms of precision and f-score. Moreover we can notice that this method tend to change a lot the training set so we can see the same problem of overfitting of the ADASYN method but this time the effect is absorbed partially by the undersampling method, in fact on the validation set our accuracy is 98% while on the training the accuracy is around 93%.




(a) RENN ROC curve




(b) SMOTE-ENN

## 2.5 Class_weight hyperparameter tuning

In this section we have tried to make hyperparameter tuning over the class_weight of the decision tree algorithm using the original unbalanced dataset.
From the results presented at the end of the chapter (table 2.1) we can assert that tuning the class_weight for our dataset is better than using any of the oversampling method proposed in this report since by tuning the class weight we reach the same level of recall of the ADASYN algorithm but we reach even a great precision on the minority class.

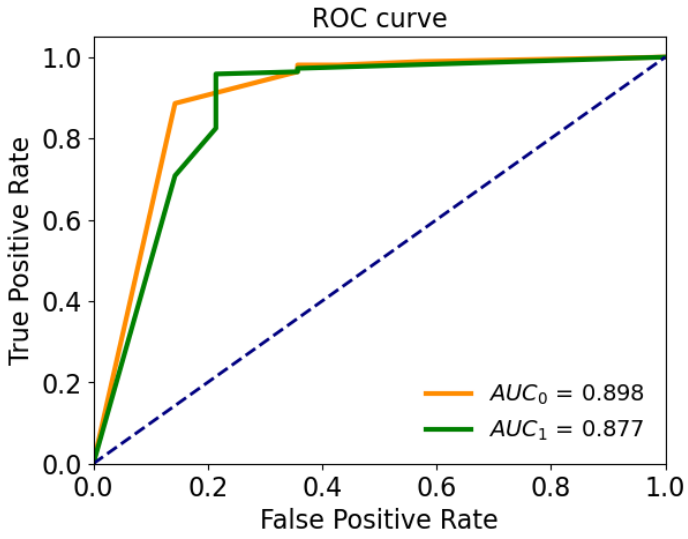



Figure 2.6: Class_weight tuned model

## 2.6 Conclusion

Table 2.1: Imbalanced classification reports

| | Base model | | | RENN | | | ADASYN | | | SMOTE-ENN | | | class_weight | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Class | Precision | Recall | F-score | Precision | Recall | F-score | Precision | Recall | F-score | Precision | Recall | F-score | Precision | Recall | F-score |
| 0 | 0.56 | 0.64 | 0.60 | 0.43 | 0.93 | 0.59 | 0.29 | 0.79 | 0.42 | 0.26 | 0.93 | 0.41 | 0.42 | 0.79 | 0.55 |
| 1 | 0.99 | 0.98 | 0.98 | 0.99 | 0.95 | 0.97 | 0.99 | 0.93 | 0.96 | 0.99 | 0.90 | 0.94 | 0.99 | 0.96 | 0.97 |
| accuracy | | | 0.97 | | | 0.95 | | | 0.92 | | | 0.90 | | | 0.95 |
| macro avg | 0.77 | 0.81 | 0.79 | 0.72 | 0.94 | 0.78 | 0.64 | 0.86 | 0.69 | 0.63 | 0.91 | 0.68 | 0.71 | 0.87 | 0.76 |
| weighted avg | 0.97 | 0.97 | 0.97 | 0.98 | 0.95 | 0.96 | 0.96 | 0.92 | 0.94 | 0.97 | 0.90 | 0.92 | 0.97 | 0.95 | 0.96 |

After having examined all the methods using to solve the unbalanced learning task we can state that for our dataset and our model the best algorithm to use is the Repeated Edited Nearest Neighbour since it brings to the higher performance of the model in terms of both precision, recall and F-score, moreover the difference between the validation accuracy and the test accuracy is contained in the range of 1%-2% and the AUC is the greatest among those of the other methods. The second best method in terms of recall and AUC is the SMOTE-ENN but it tends to suffer a bit of overfitting since the difference between the validation accuracy and the test accuracy is the 5% and the precision on the minority class tend to lower with respect the base model. Below is presented the tree of the model trained over the training set modified according to the Repeated Edited Nearest Neighbour.

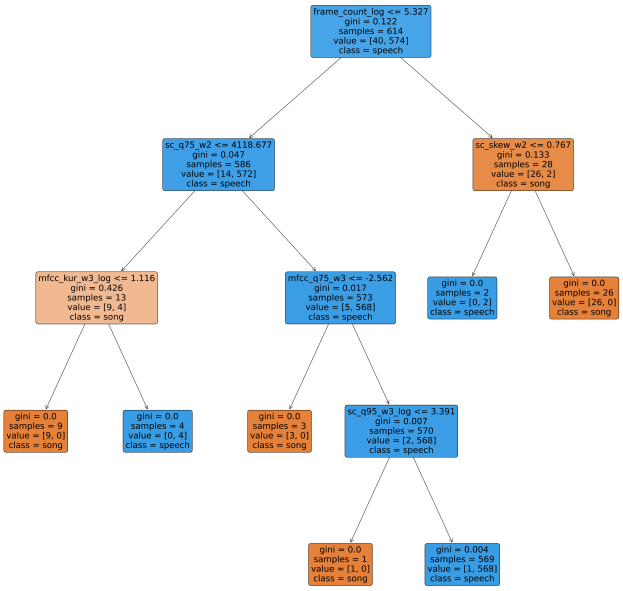



Figure 2.7: Repeated Edited Nearest Neighbour

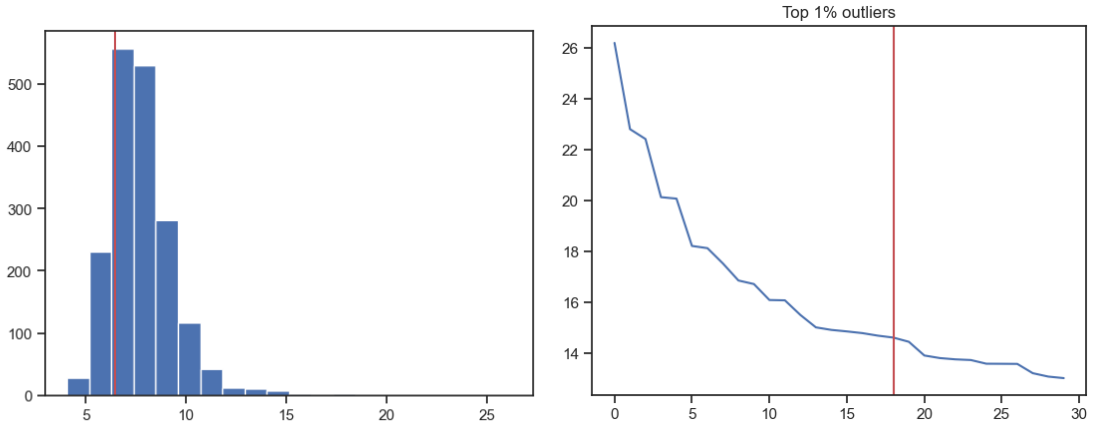
# Chapter 3: Outlier detection

For outlier/anomaly detection we exploited the numerical variables only. Data are normalized exploiting the Scikit-learn standardScaler. The objective was to use different methods to obtain more complete information. The involved algorithms are KNN, COF, CBLOF and ABOD. We preferred models with an outlier score to make later comparisons. At the end, we tried to merge these different pieces of information provided by the four algorithms.

## 3.1 Algorithms

### 3.1.1 KNN

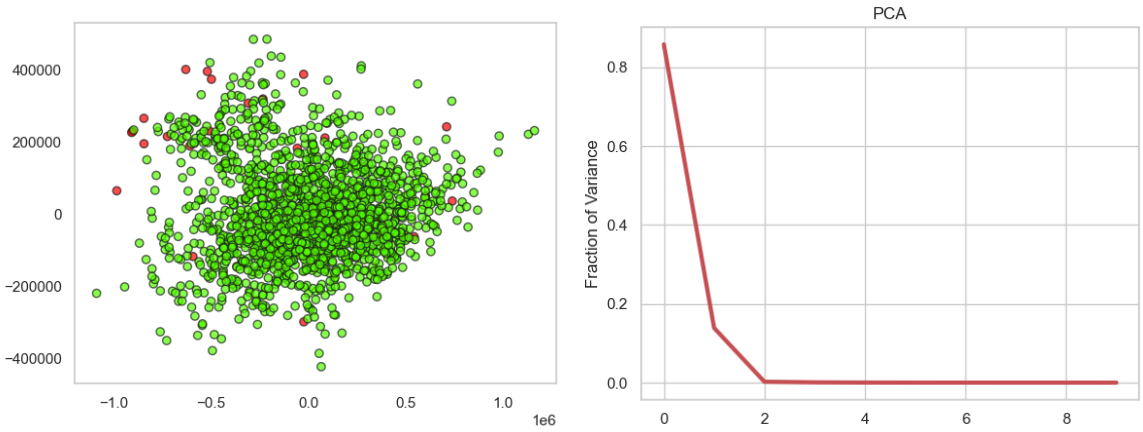
KNN was performed with 5 neighbors. With this method we found 30 outliers. In order to plot them,




(a) KNN decision score (b) Top 1% outliers (based on KNN score

Figure 3.1: KNN outliers

we performed a PCA with 3 components, since they capture almost all the variance (3.2b shows an example with 10 components).




(a) KNN outliers (PCA) (b) Fraction of variance

Figure 3.2: KNN outliers visualization with PCA

### 3.1.2 COF

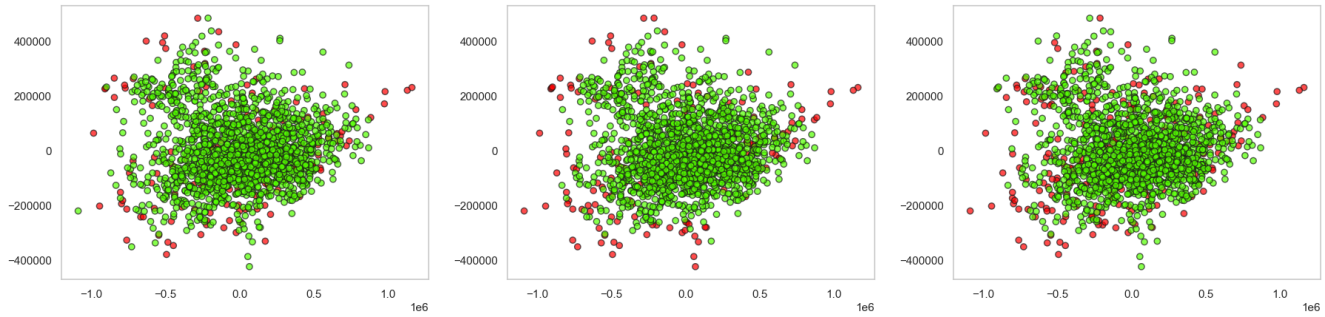
The last three algorithms captured an higher number of outliers than KNN. With a density-based approach such as COF we obtained 183 outliers (visualizations in figure 3.3).

### 3.1.3 CBLOF

CBLOF was performed with alpha=0.9 and beta=5, identifying 183 outliers.

### 3.1.4 ABOD

Exploiting ABOD (contamination=0.1, method='fast', n_neighbors=5) we found 258 outliers.

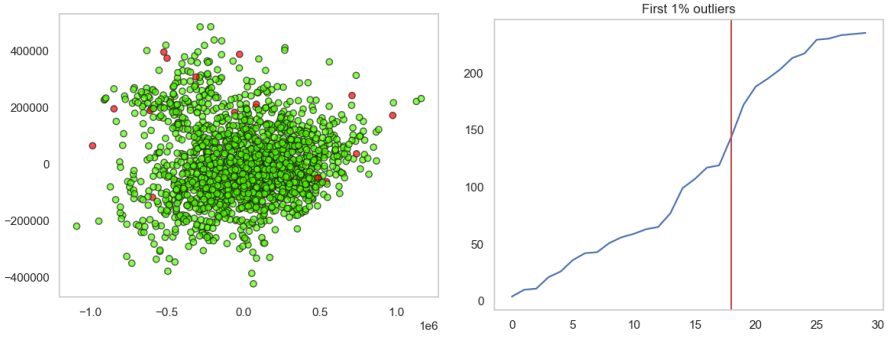



(a) COF outliers (b) CBLOF outliers (c) ABOD outliers

Figure 3.3: Outliers visualization with PCA

## 3.2 Comparisons

If we take the first 70 outliers for each method, we observe that there are 18 common outliers (we are interested in the top 1% outliers, given a train set of 1828 records). However, we tried a different approach to determine the top 1%, considering records which have an higher overall outlier_score. The idea is simple: in our model each record gets four scores, based on how high it is placed in the scale of each method. For instance, it will get 1 if it is the top 1 outlier of that particular method, 2 if it's the second one and so on. At the end, all scores will be added up to get the final score for each item. We highlighted the top 1% records according to the final overall score.



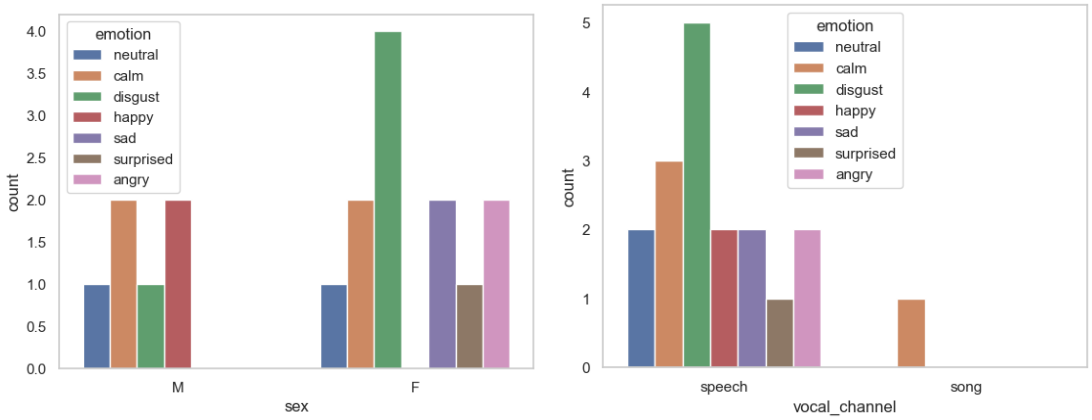

| Record index | Overall score |
|---|---|
| 361 | 4 |
| 0 | 10 |
| 223 | 11 |
| 1604 | 21 |
| 1398 | 26 |

(a) PCA visualization (b) Overall score (c) Top 5 outliers

Figure 3.4: Common outliers

We have found some interesting insights: for example, record number 361 has an overall score of 4, it means that it is the top outlier for every method. Indeed, all the 18 interesting outliers have an

overall score that is not greater than 127, it means that all the algorithms agrees on these particular records.




(a) Sex (b) Vocal_channel

Figure 3.5: Common outliers composition

Surprisingly (4.14b), just 1 out of 18 records belongs to the 'song' class, while they are homogeneous with respect to other variables, for example 'sex'. The most frequent emotion is 'disgust'. Exploiting PCA dimensionality reduction we can plot a 3D scatterplot highlighting outliers.

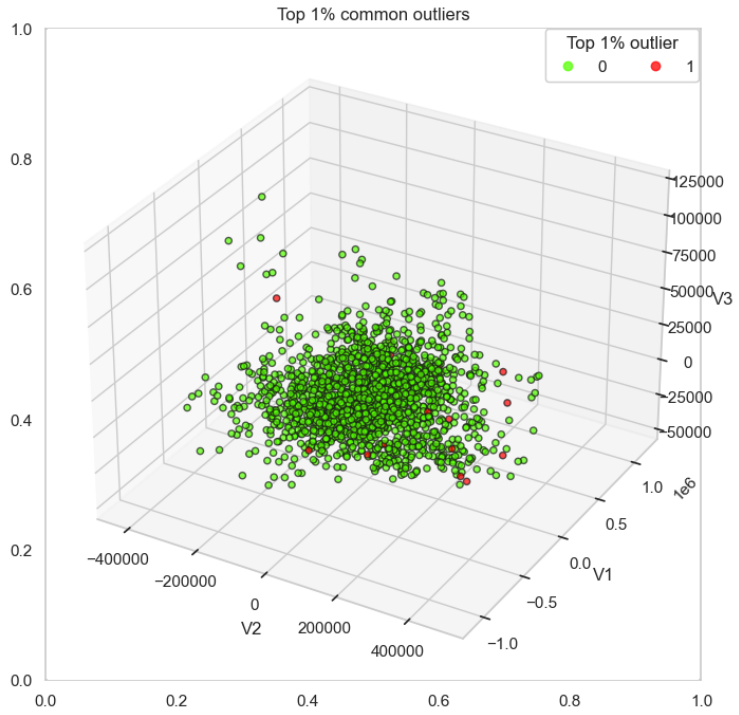



Figure 3.6: 3D visualization (axis X and Y were rotated to improve readability)

## 3.3 Conclusions

We have successfully found the top 1% outliers. These records were removed from the dataset, since we wanted to keep information as clean as possible, especially for Chapters 4 and 5. After some attempts with Neural Networks, for example, we observed that outliers are responsible for some bad model convergences.

# Chapter 4: Advanced classification

In this section, we try to classify the different emotions with several advanced methods, so our target variable will be the label encoded "emotion". Our dependent variable assumes value 0 if the emotion is "angry", value 1 for "calm", 2 for "disgust", 3 for "fearful", 4 for "happy", 5 for "neutral", 6 for "sad", 7 for "surprised". Due to save space, at the end of this chapter will be presented the classification report that contains the result of the best models only.
Below we will show a 2D representation of our dataset by using first the PCA to reduce the dimensionality to 50 and then T-SNE. We have used this approach because PCA only captures linear relationships between data points while T-SNE is a non-linear dimensionality reduction technique but it can suffer from the high-dimensionality, so the combination of this two methods can lead to a good visualization of complex datasets.

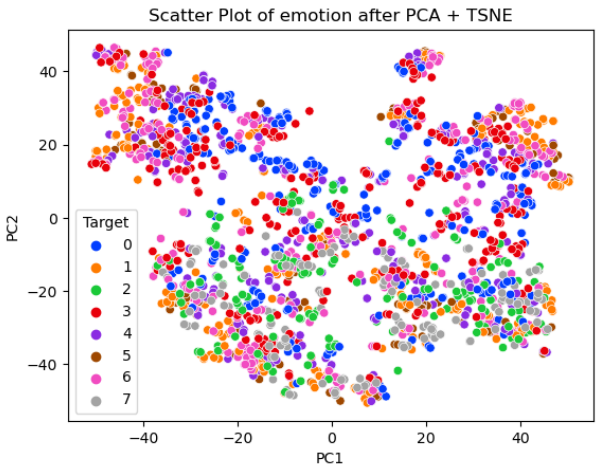


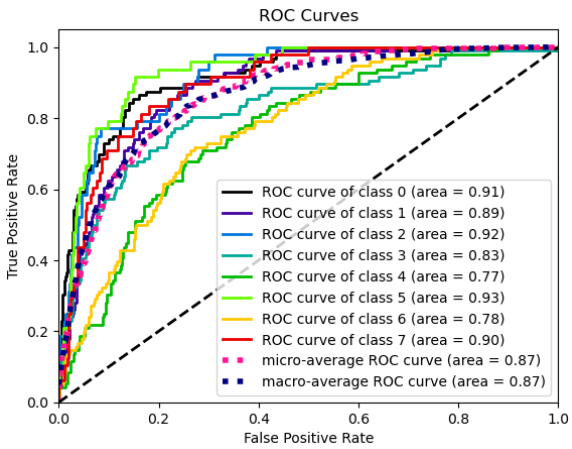

Figure 4.1: Emotion dimensionality reduction scatter plot

## 4.1 Logistic Regression

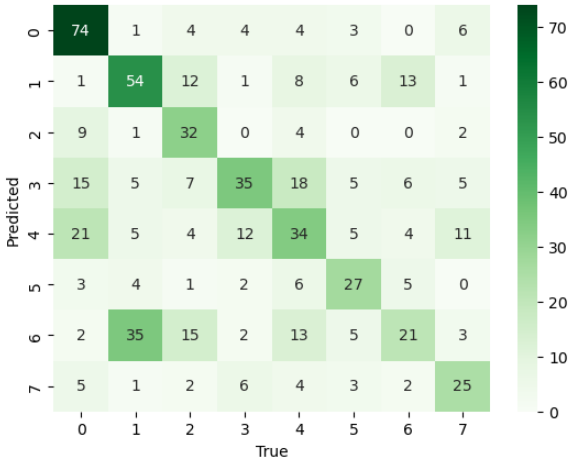
The first approach we have used is the logistic regression. First of all, we normalized our data by fitting the standardscaler on the training set and by transforming the training set and the test set. Since if we will use all the variables for training our model can overfit some authors have suggested to first run a univariate logistic regression using each time a different predictor variable and then use those variable with high value of significance to run the multivariate logistic regression. Since sklearn does not support a p-value to test the significance of an estimator we have made feature selection by using a pipeline. The pipeline consists on first applying the selectKbest and then a univariate logistic regression. We have used then this pipeline to make a gridsearch to find the best K for our selectKbest. It turns out that the best K is equal to 95. After this step, we used the new dataset with only the Kbest features to make hyperparameter tuning of the multivariate logistic regression using the randomized search with repeated stratified k fold. It turns out that the best value for C is 0.75 with the saga solver and the best score on the validation set obtained by this model is 59.4%. However, this model has more overfitting than the one that we will describe soon, in fact, the accuracy on the test set is just the 47.6%.
Since we noticed that on the original dataset[1] this best penalty for our model is the l1 we decided to let the feature selection task to the lasso regularization. So we have made hyperparameter tuning by

[1]The one before making selectKbest

using the randomized search and repeated stratified k fold as before by using the L1 penalty. Now the best values for our hyper parameters are C=0.605 and solver=saga. The accuracy obtained on the validation set is 59.24% while on the test set is 48.5%. The most important coefficient, the one with the highest value, is "mfcc_mean_w3" but we cannot know if it is significant since sklearn does not provide this kind of test. Below are presented the result of this model, the roc curves and the confusion matrix.




(a) logistic regression ROC curve

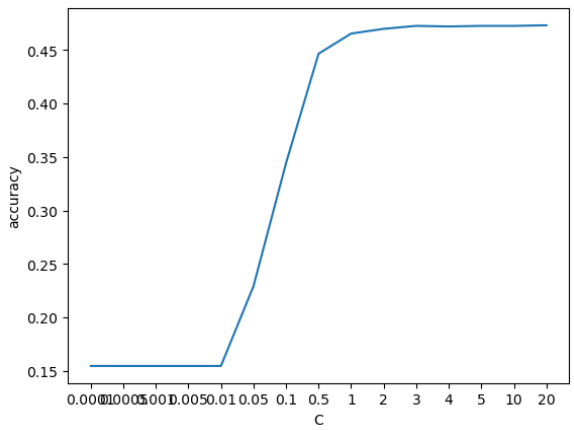



(b) logistic regression CF

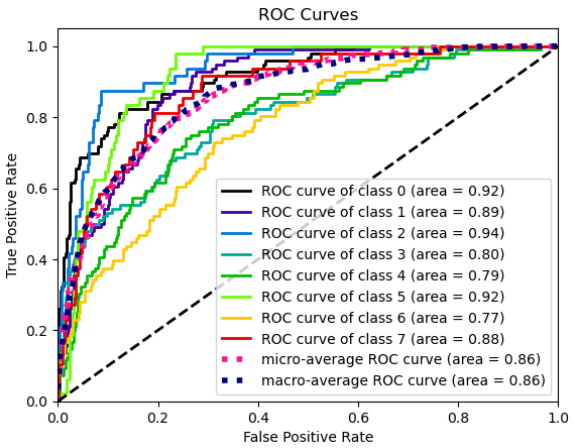
From the results we can see that the class predicted correctly most of the time is the class 0 (angry) while the class predicted wrongly most of the time is class 1 (calm) which is often misclassified as class 6 (sad). From the roc curve can be seen the classes with the worst performance for every level of false positive rate until 0.5 are class 6 (sad) and class 4 (happy) then for higher false positive rate the emotion with the worst performance is class 3 (fearful).

## 4.2 Support Vector Machine

We first tried the linear SVM and then the nonlinear SVM, the last one had slightly better results. In particular, to get the best possible results from the nonlinear SVM we used the cross-validation technique to plot the changes in accuracy for different values of our hyperparameters, for example, in the figure below we can see accuracy increment as C increases until a certain value of C, from here on the accuracy remain almost the same, so to avoid overfitting we bounded above C.




(a) Accuracy and hyperparameter C




(b) ROC curve of the nonlinear SVM

Then we used this information to build the Randomized Search and our best hyperparameter

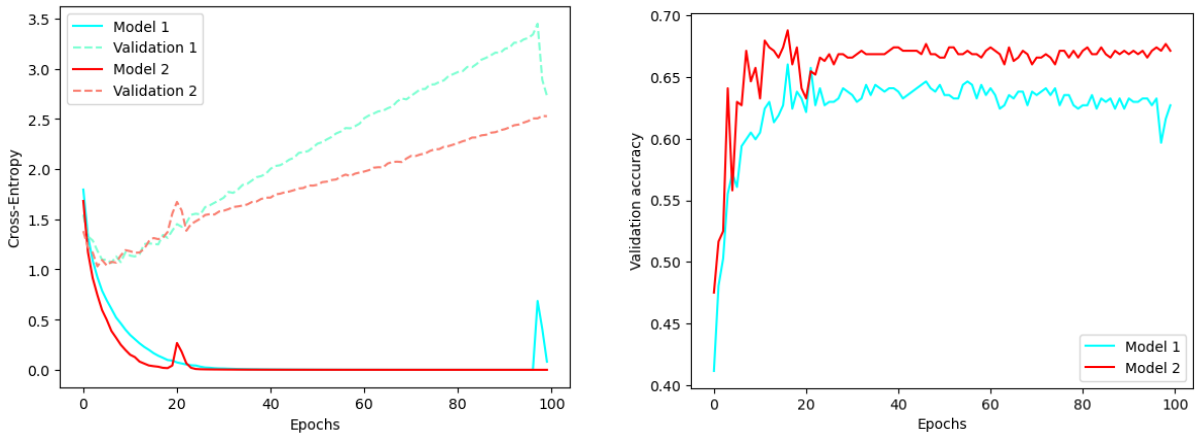
configuration is the following: 'C': 2.957, 'class_weight': 'balanced', 'gamma': 0.008, 'kernel': 'rbf'. With these settings, we have achieved 66% accuracy on the validation test and 51% on the test set. From the ROC curve above, we can observe that class 2 (disgust) is better classified, while 0 (angry) and 5 (neutral) have the same area, but from values of false positive rate less than 0.2 the best one is angry, while for values greater than 0.2 the best one is neutral. The worst classified class is the 6 (sad).

## 4.3 Neural Networks

For this task we exploited both Scikit-learn neural networks and Keras models. We started with Keras because it's a more complete framework, then we compared results with Scikit-learn.

### 4.3.1 Keras

The first goal was to find a correct size of the neural network. We started with a simple model composed of two hidden layers of 64 neurons, then we compared the results with the same model but with 128 neurons in each layer, without any regularization. The activation function is ReLU for the hidden layer and softmax for the output layer. The loss function is the Categorical Cross Entropy, and we used a batch size of 10. Results were not good, since we bumped into overfitting. As (4.8) shows, even though validation accuracy is pretty good in both models (0.543 and 0.575 respectively), validation loss is too high. We run 100 epochs because we noticed that from this point on the results are pretty the same, we reached a good convergence, Validation set was chosen with a 0.2 split.




(a) Validation losses

(b) Validation accuracy

Figure 4.4: Model 1 VS Model 2.

To avoid overfitting, we considered using Dropout layers provided by Keras. Moreover, we added a L2 regularization term. In order to find good hyper-parameters for the model a grid-search was performed. Since results are not so different between the two models, the model we tested for grid-search had 64 neurons in each hidden layer. This choice let us run the code in less time, and it was the right choice in our opinion to avoid as much as possible overfitting by reducing the model complexity. Results shows that the best parameters are 0.005 for the L2 kernel regularization and 0.3 for the dropout probability, providing a 0.5634 validation accuracy.

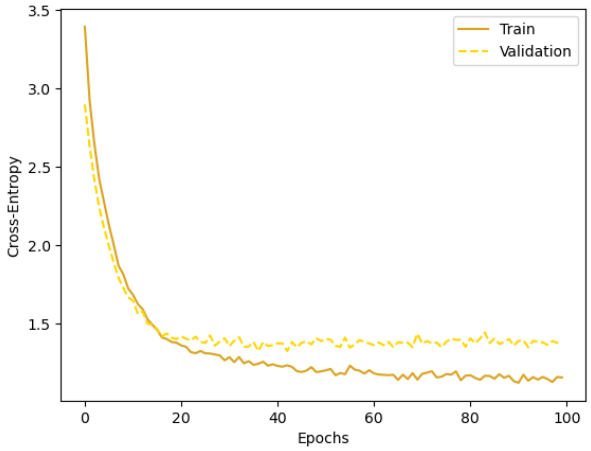



Figure 4.5: Model 3 performances with L2-norm and Dropout

With such a model results are better in terms of overfitting. At this point we considered testing a more complex version of model 3, with 128 neurons, since we reached more stability. After a second grid-search, optimal hyperparamters are 0.01 for L2-norm and 0.5 for Dropout.

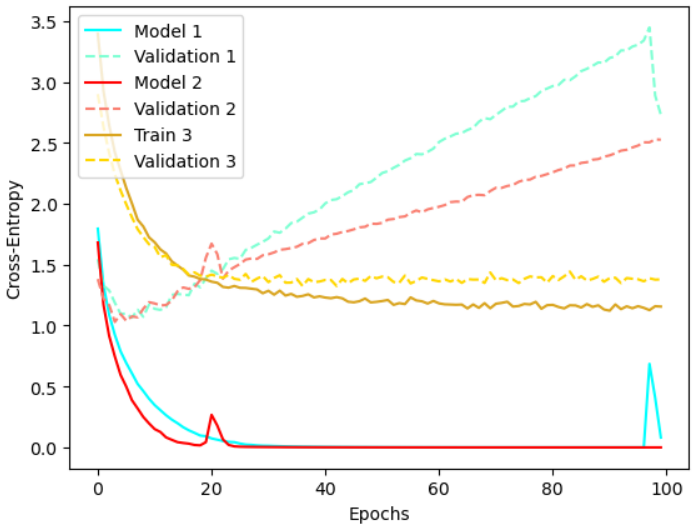



Figure 4.6: Models comparisons

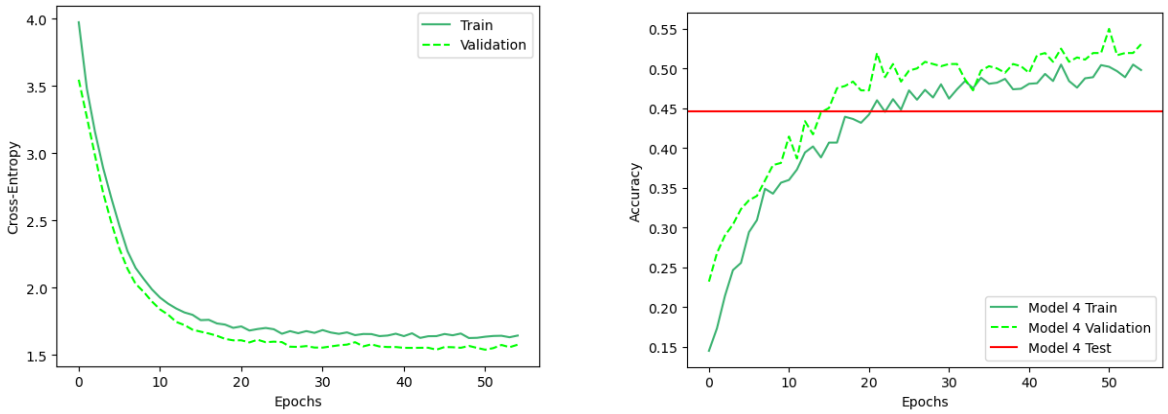
100 epochs were run to compare results. We gained in terms of overfitting prevention, but we lost in terms of performances. However, it seemed the right choice because Validation loss has decreased and it was essentially stable, unlike the first 2 models. Moving to test set, results are pretty good in terms of accuracy but Model 3 has a significantly better result in terms of loss.

Table 4.1: Models testing comparisons

| Models | Validation set | | Test set | |
|---|---|---|---|---|
| | Accuracy | Loss | Accuracy | Loss |
| Model 1 | 0.627072 | 2.729788 | 0.511218 | 5.735857 |
| Model 2 | 0.671271 | 2.525730 | 0.493590 | 6.260267 |
| Model 3 | 0.654696 | 1.379016 | 0.503205 | 2.109065 |

However, there is still a large gap between validation accuracy and test accuracy. So, even though results are quite good so far, we wanted to try another approach. Model 4 is composed by three

hidden layers (64, 32, 64). Moreover, we added an early stopping criterion because we wanted to monitor the validation loss (patience = 10).



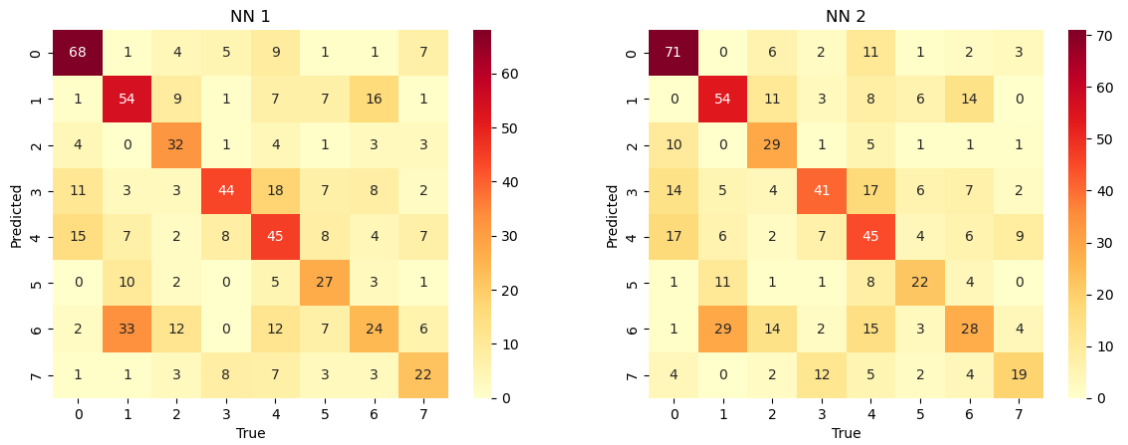

(a) Loss (b) Accuracy

Figure 4.7: Model 4 performances

The algorithm stopped after just 55 iterations. With such a model (4.7), the gap between validation and test set is significantly better. With a test accuracy of 44.5513% and a validation accuracy of 49.7267%, performances have decreased but we expect it to be a more general model (loss function for the test set is 1.699722).

### 4.3.2 Scikit-learn

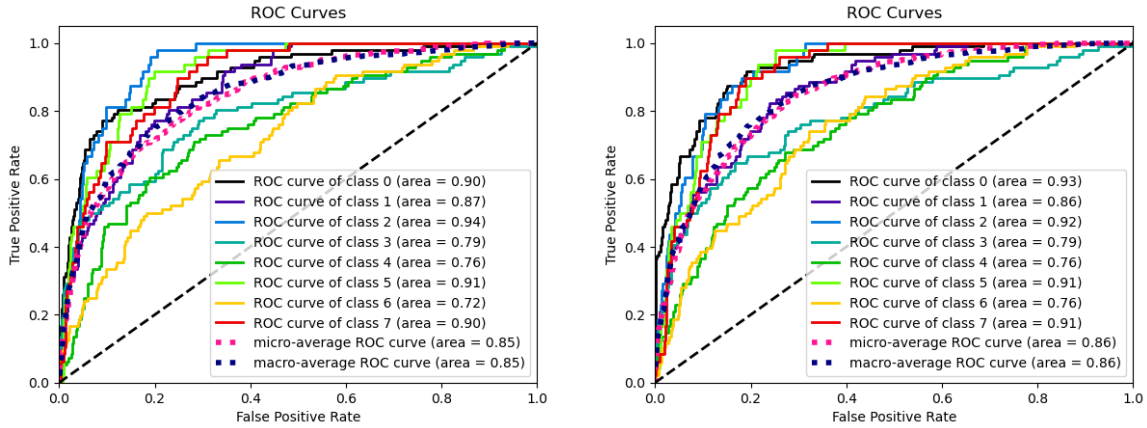
Scikit-learn implements the MLPClassifier. We repeated what we have already done with Keras. These are the results of the first two NNs with 2 hidden layers, the first model with 64 neurons each and the second one with 128. We used the same parameters and activation functions as section 4.1.1. With the first model we obtained a 50,6410% accuracy and 49,5192% with the second one. There




(a) 64 neurons (b) 128 neurons

Figure 4.8: Scikit-learn models performances

are important differences between classes, as (4.9) shows. As in DM1 project, there are emotions which are easier to detect, while others generates poor performances. Moving on to hyper-parameters tuning, we performed a gridserach with cross-validation and we found that the best optimizer for our problem is still 'adam', while the optimal L2 norm regularization term 'alpha' is 0.001. We tried different hidden layers combination as we did for Keras, and gridsearch results show that two layers




(a) 64 neurons (b) 128 neurons

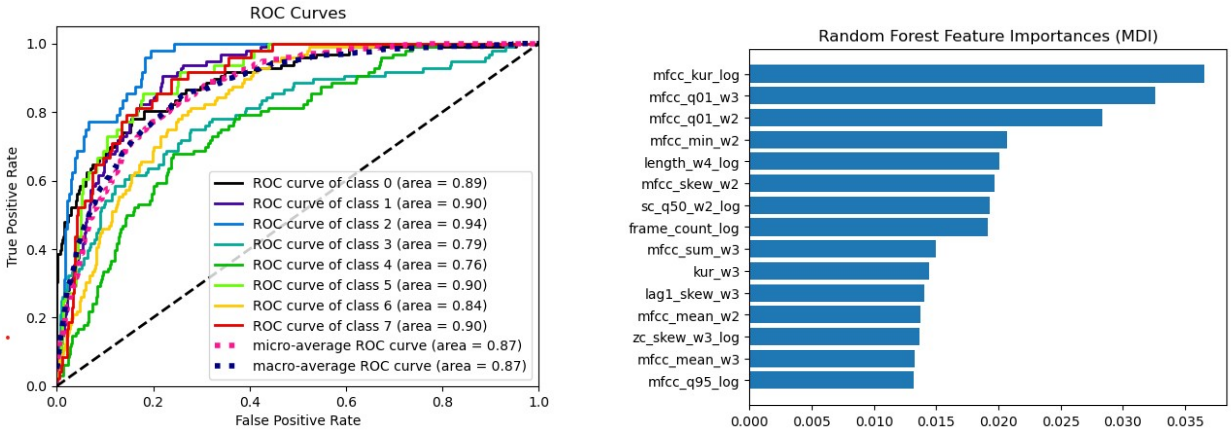
Figure 4.9: ROC curves

of 128 neurons is the best choice. With such a model, the results are practically identical to the previous (4.8b). Here we face the same problem as before, because the validation score is 65,1313%, higher than the 51.7628% accuracy. There are no big differences between Keras and Scikit-learn performances. However, we believe that Keras' flexibility is a big advantage in terms of overfitting prevention, and the possibility of fine tuning the parameters makes it the optimal choice.

## 4.4 Ensemble Methods

### 4.4.1 Random Forest

For the Random Forest algorithm, first of all, we plotted the changes of accuracy for different values of our hyperparameters, then we construct the Randomized Search, and our best hyperparameter configuration is the following: 'criterion': 'entropy', 'max_depth': 50, 'min_impurity_decrease': 0.004, 'min_samples_leaf': 1, 'min_samples_split': 5, 'n_estimators': 290. With these settings, we have achieved 61% accuracy on the validation test and 46% on the test set.




(a) Random Forest ROC curve (b) Random Forest feature importance

Figure 4.10

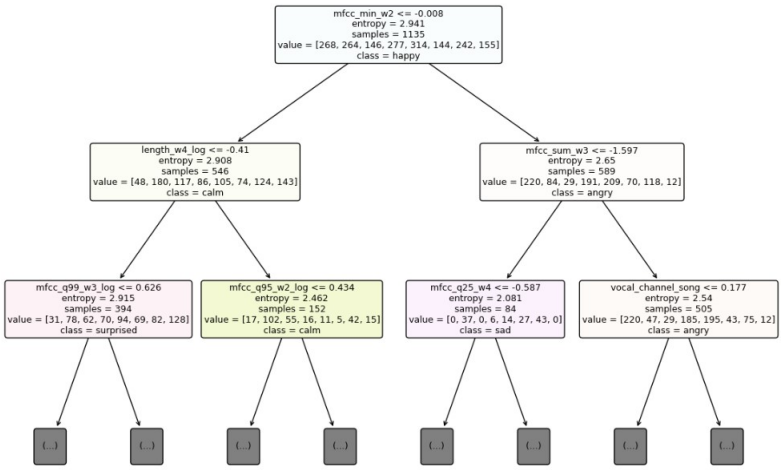
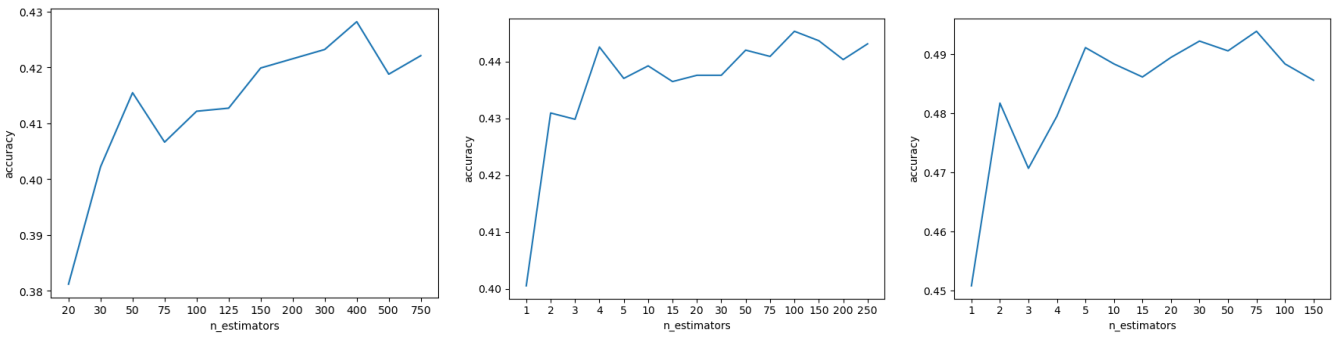



Figure 4.11: First Random Forest tree

From the ROC curve, we can notice that class 2 (disgust) is better classified, while the worst is class 4 (happy). From the feature importance, we can see the main features responsible for the splits, and in the last figure, from the first part of the first tree, the first split is by 'mfcc_min_w2', this was fourth in the ranking.

### 4.4.2 Bagging

For the Bagging Classifier, we decided to try 5 different base classifiers: Decision Tree, Random Forest, nonlinear SVM, Logistic Regression, and Neural Network with 1 hidden layer. First, we plotted the changes of accuracy for different numbers of base estimators, so we had a first estimate of which the best base classifiers were and the different ranges of n_estimators for the tunings. For example, in figure 4.12, we can see the accuracy for the Decision Tree, the Random Forest, and the Neural Network (1 hidden layer) as base classifiers: the first one requires more estimators, at least 50, to achieve good accuracy, while the others (Random Forest, NN, but also SVM and Logistic Regression that are not in the figure) reach a good accuracy with few estimators, more or less 5.




(a) Decision Tree (b) Random Forest (c) NN 1 hidden layer

Figure 4.12: Accuracy on number of estimators in Bagging

After finding the best hyperparameters for each Bagging Classifier with a different base model, we concluded that the best base classifier is the Neural Network, and with the following hyperparameters: 'n_estimators': 55, 'base_estimator__hidden_layer_sizes': (200,), 'base_estimator__activation': 'relu' we achieved 66% accuracy on the validation test and 51% on the test set.

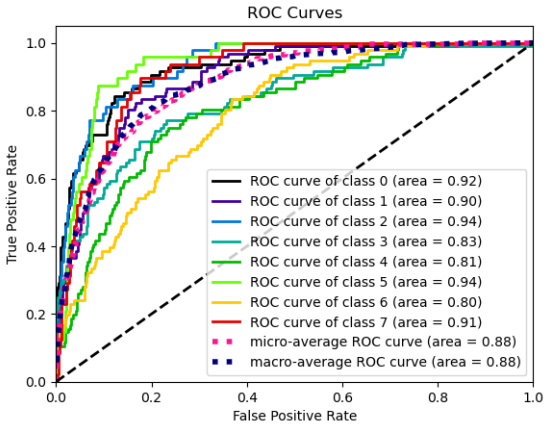
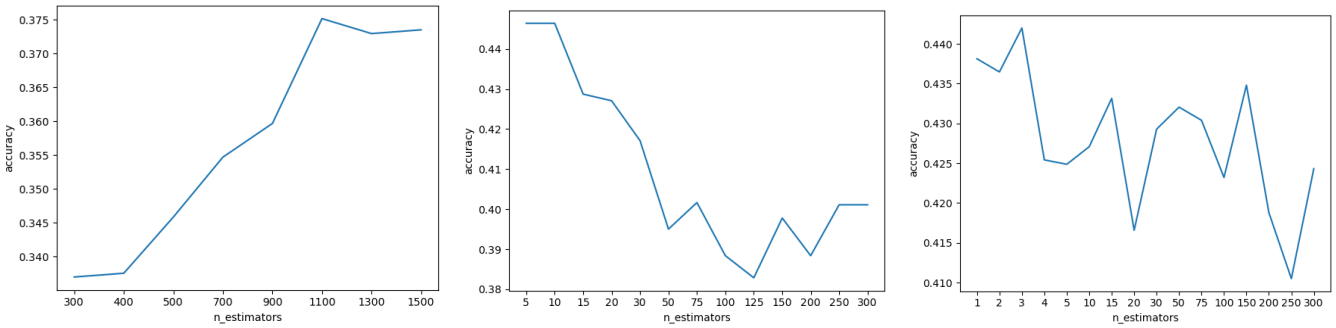



Figure 4.13: Bagging with NN ROC curve

From the ROC curves, we can observe that class 2 (disgust), like almost every model, and the class 5 (neutral) are the better classified, in addition, for the first time none of the classes have an area of less than 0.8.

### 4.4.3 Boosting

For the Boosting method, we used the algorithm AdaBoostClassifier, and we did the same procedure as Bagging, so we plotted the changes in accuracy for different numbers of base estimators. Also here (4.14) the Decision Tree as base classifier requires more estimators, while the others (SVM, Random Forest, but also Logistic Regression that is not represented in the figure below) after a few estimators have decreasing accuracy.




(a) Decision Tree (b) Nonlinear SVM (c) Random Forest

Figure 4.14: Accuracy on number of estimators in Boosting

Then we found the best hyperparameters for each AdaBoostClassifier with a different base model, and we achieved the best results with the Random Forest as the base classifier, with the following hyperparameters: 'base_estimator__criterion': 'gini', 'base_estimator__max_depth': 56, 'base_estimator__min_impurity_decrease': 0.004, 'base_estimator__min_samples_leaf': 3, 'base_estimator__min_samples_split': 67, 'base_estimator__n_estimators': 459, 'learning_rate': 0.131, 'n_estimators': 22 we achieved 57% accuracy on the validation test and 48% on the test set. In this case, we decided to omit the representation of the ROC curve, as it is almost identical to the one of the Random Forest in figure 4.10.

### 4.4.4 Gradient Boosting Methods

For the Gradient Boosting Methods, we tried both XGBoost and LightGBM, the results were similar, but finding the best hyperparameters with the LightGBM was faster due to the lower complexity of the algorithm, so we used that one. Like always we plotted the changes in accuracy for different values of our hyperparameters, for example in this figure we can see that the learning rate shouldn't be too high, in particular after 0.6 the accuracy drops quickly by 10%.

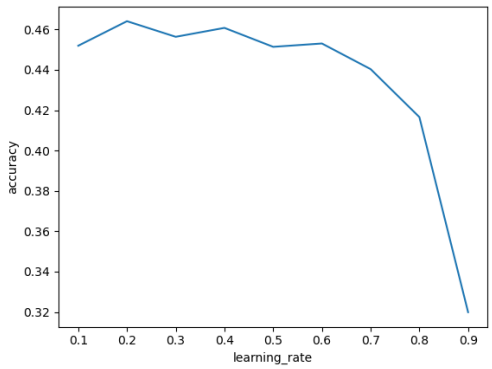



Figure 4.15: LightGBM learning rate

In the end, with the following hyperparameters: 'learning_rate': 0.257, 'max_depth': 4, 'min_split_gain': 1.861e-05, 'n_estimators': 155, 'reg_alpha': 0.134, 'reg_lambda': 4.006 we achieved 64% accuracy on the validation test and 47% on the test set.

## 4.5 Comparisons

As a matter of readability, we decided to show in the table below only the reports of the classifiers that have an accuracy greater than 50%, so Nonlinear SVM, Neural Network, and Bagging with Neural Network (1 hidden layer) as the base classifier.

Table 4.2: Classification reports

| | Nonlinear SVM | | | Neural Network | | | Bagging (NN) | | |
|---|---|---|---|---|---|---|---|---|---|
| Class | Precision | Recall | F-score | Precision | Recall | F-score | Precision | Recall | F-score |
| 0 | 0.60 | 0.76 | 0.67 | 0.67 | 0.71 | 0.69 | 0.57 | 0.75 | 0.65 |
| 1 | 0.54 | 0.53 | 0.53 | 0.50 | 0.56 | 0.53 | 0.56 | 0.56 | 0.56 |
| 2 | 0.45 | 0.60 | 0.51 | 0.48 | 0.67 | 0.56 | 0.48 | 0.65 | 0.55 |
| 3 | 0.61 | 0.45 | 0.52 | 0.66 | 0.46 | 0.54 | 0.61 | 0.42 | 0.49 |
| 4 | 0.43 | 0.50 | 0.46 | 0.42 | 0.47 | 0.44 | 0.43 | 0.49 | 0.47 |
| 5 | 0.41 | 0.56 | 0.47 | 0.44 | 0.56 | 0.50 | 0.50 | 0.48 | 0.49 |
| 6 | 0.52 | 0.25 | 0.34 | 0.39 | 0.25 | 0.30 | 0.42 | 0.28 | 0.34 |
| 7 | 0.43 | 0.42 | 0.42 | 0.45 | 0.46 | 0.45 | 0.49 | 0.50 | 0.50 |
| accuracy | | | 0.51 | | | 0.51 | | | 0.51 |
| macro avg | 0.50 | 0.51 | 0.49 | 0.50 | 0.52 | 0.50 | 0.51 | 0.52 | 0.50 |
| weighted avg | 0.51 | 0.51 | 0.50 | 0.51 | 0.51 | 0.50 | 0.51 | 0.51 | 0.50 |

First, we see that all 3 classifiers have similar mean values for all evaluation metrics. Going into detail, we can observe that for class 6 (sad) we have the worst results, the only data that differs from the others is the precision of the SVM, which is about 10 % higher than the values of the other two, thus produces less false positive. Another interesting thing is the precision of class 2 (disgust): we have relatively low values (less than 50%), while as seen earlier in the ROC curves of the 3 algorithms, the AUCs for this class were high (above 0.9), this shows the importance of looking at multiple evaluation metrics to judge the goodness of a classifier.

# Chapter 5: Advanced regression

In this chapter we use neural networks and support vector machine to solve the regression task using as target variable the "sc_max_log". First of all, we have transformed all the categorical variables in dummies, then we have normalized the data by using the standard scaler. To figure out the complexity of this regression task we have applied a linear regression using all the variables and we get an $R^2$ of 60% so we can expect that non linear models can get better performance with respect to this simple linear regression.

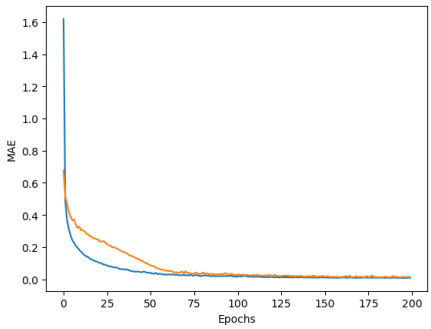
## 5.1 Neural Network

We have built several neural networks using keras. To begin we have defined a simple model composed by 2 hidden layers, the first with 10 nodes and the second with 32 nodes, both the 2 layers use the "relu" activation function and the "normal" kernel initializer. As loss function has been used the mean squared error and the "adam" optimizer. This model has been trained using 150 epochs and batch_size=50 and evaluated on a validation split of 20% of the training data. We have used this model to find the best value for the batch_size and the number of epochs. We will use the value of this 2 parameters that minimize the loss function on the validation set, and in particular this value are 200 for the epochs and 20 for the batch_size. By testing this model on the test set we see that the performance are around the 62% of $R^2$. To increase the performance of our neural network we have made a randomized search for testing different kind of structures[1], different optimzers, in particular "adam" and "sgd", different values for the learning rate, momentum rate and different values for l2 regularization that has been added to all hidden layers. Even the dropout regularization has been added after each hidden layer. The activation function remain "relu" for every layer and even the batch_size and epochs are still 20 and 200. The results show that the best structure among those proposed is composed by 3 hidden layers where the first and the last layer has 32 nodes while the middle one has 50 nodes. The best optimizer is "adam" with a learning rate of 0.01 and a momentum of 0.4 while the best value for the regularization l2 is 0.001. With this model we reach an $R^2$ on the validation set of 66%.
Then we decided to make little adjustment to this model by removing the dropout from the last hidden layer and by removing the l2 regularization from the last hidden layers and replacing the regularization on the first hidden layer with a l1 regularization with power of 0.0005. By making these changes the model now reach and $R^2$ of 69% on the test set. Below are reported some statistics about the performance of this model.

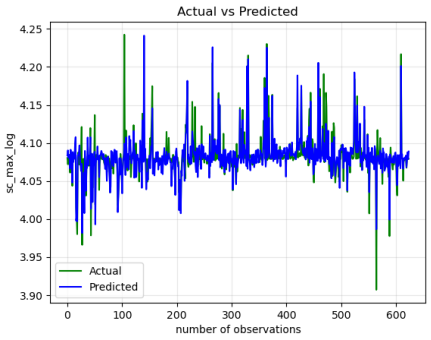
Table 5.1: NN final model performance measures

| Data | $R^2$ | mean_error | std_error | min_error | max_error |
|---|---|---|---|---|---|
| Validation_set | 0.82 | 0.0089 | 0.0092 | $1.31e^{-6}$ | 0.0798 |
| Test_set | 0.69 | 0.0117 | 0.0157 | $8.85e^{-6}$ | 0.1409 |

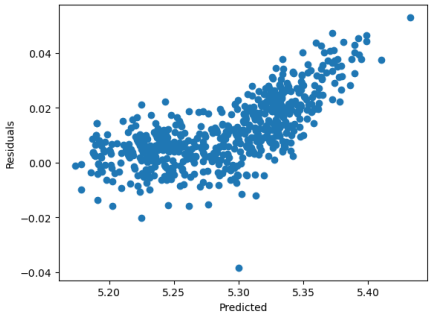
[1]We have tested these possible structures (50, 50, 50), (32, 50, 32), (50, 32, 10), (10, 32, 50), (32, 50, 50, 32), (80, 80, 80, 80)




(a) NN loss function



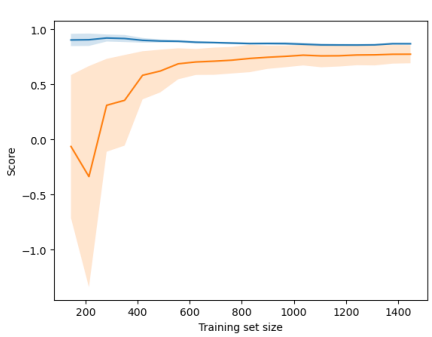

(b) NN actual vs predicted plot



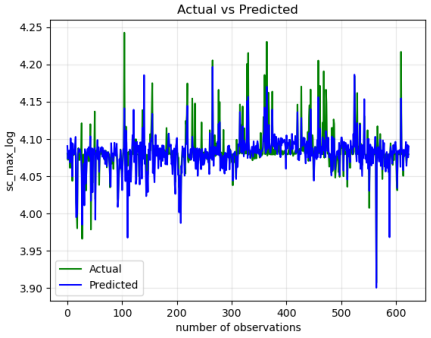

(c) NN residuals

## 5.2 Support Vector Regression

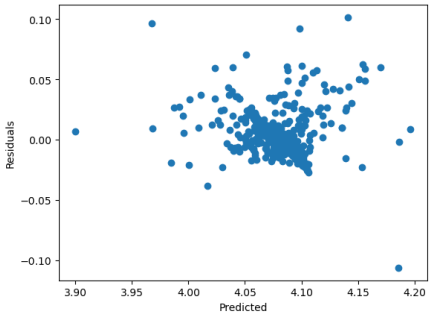
In order to find the best set of hyper parameters that allows the SVR to reach the best possible performance we first tried on a base model different type of kernels. The results shows that for our dataset the best ones are "rbf" and the "poly" kernels. In order to find the best hyper parameters that allows us to avoid overfitting we used a randomized search over important parameters as "C", "epsilon", "gamma", "coeff0" and the "degree". The best result obtained over 1000 iteration with a tollerance for the stopping criterion of $1e^{-4}$ we have found that the best hyperparameters for our SVR are: "C=86.61", "coef0"=7.80, "degree"=1.33[2], "epsilon"=0.0099, "gamma"=scale and "kernel"=poly. The $R^2$ of the test set is equal to 0.66 while on the training set is 0.84.




(a) SVR learning curve




(b) SVR Actual vs predicted plot




(c) SVR residuals

## 5.3 Conclusion

From the result we can see that the Neural Network is the best model between the 2 that we have tried since it brings to the higher $R^2$ and has less overfitting with respect the training data. From the plot of the residual of the NN it is clear that the model tends to predict less accurately those observation with a prediction higher than 5.30, in fact we can see that for those observation the real value is always higher. This behaviour seems to exists even for the SVR although the residuals seems to be more centered to 0. From both the loss function of the NN and the learning curve of the SVR we can see the comparison of the training vs the validation results and from these the model does not overfit because the performances of the training set and the validation set tend to converge so one reason to the poor performance on the test set could be that the training set is not really a very good representation of the test set so one way to improve these result can be to improve the quality of the data.

[2]This means that our problem is almost linear

# Chapter 6: Time series data preparation and understanding

Now, our dataset consists of 1828 records in the training set and 624 records in the test set, and each record is a time series that is 304304 timestamps long. We will prepare the time series so that it will be possible to carry out future tasks in the next chapters. The following procedure was performed for both the training and test set. First, we replaced all Nan from the time series with zeros, next, since the first and the last parts of all time series have values very close to 0, we decided to cut those timestamps, that were not useful for our upcoming tasks, in Figure 6.1 we can see the actual cuts represented by vertical red lines.

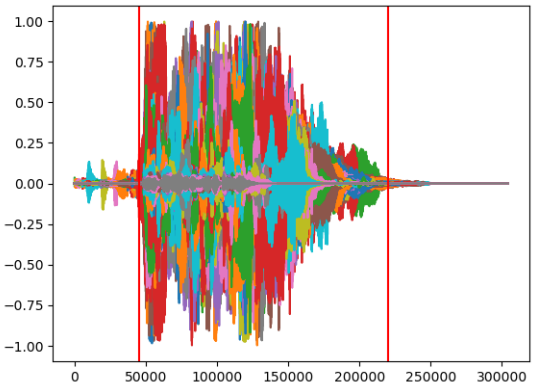
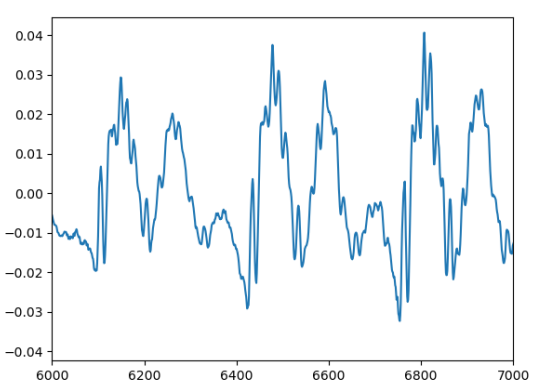
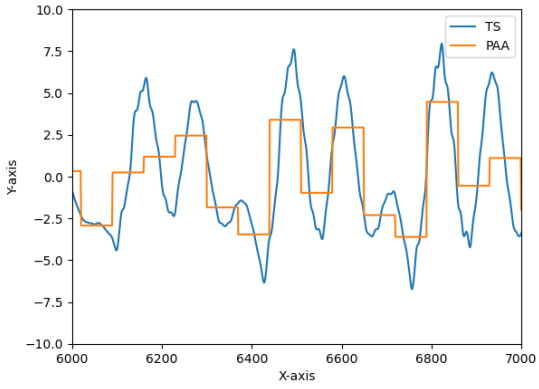



Figure 6.1: Time stamps cutting

Afterward, we decided to perform two transformations: amplitude scaling, to amplify or decrease the oscillations of each time series; Noise removal, using a window of 20 time stamps, since after several attempts it seemed the best fit. Noise removal turned the first 19 values of each time series into Nan, so we replaced them with zeros. We did not remove trends because there was no need in our dataset. In the end, given the size of the dataset, and thus to reduce the execution time of the algorithms we will use for the next tasks, we decided to approximate the dataset using the PAA, dividing each time series into 2500 segments. In Figure 6.2a we can see a stretch of a time series before transformations and approximation, in Figure 6.2b, however, we have the same stretch but the time series is transformed and approximated, this one approximates some areas well (between 6500 and 6600 captures well the two high peaks and the low peak) and fewer others (between 6100 and 6300 the first high peak and the low peak are not well identified).




(a) Stretch before processing




(b) Stretch after processing

# Chapter 7: Motifs/Discords and Clustering

## 7.1 Clustering

To perform clustering, we have been using the approximated dataset using PAA (2500 timestamps). We have tested 2 methods: MiniSom and Kmeans. In particular, we have obtained the best result by using Kmeans.

### 7.1.1 Kmeans

We have run Kmeans both using Euclidean distance and DTW, with 5 clusters.

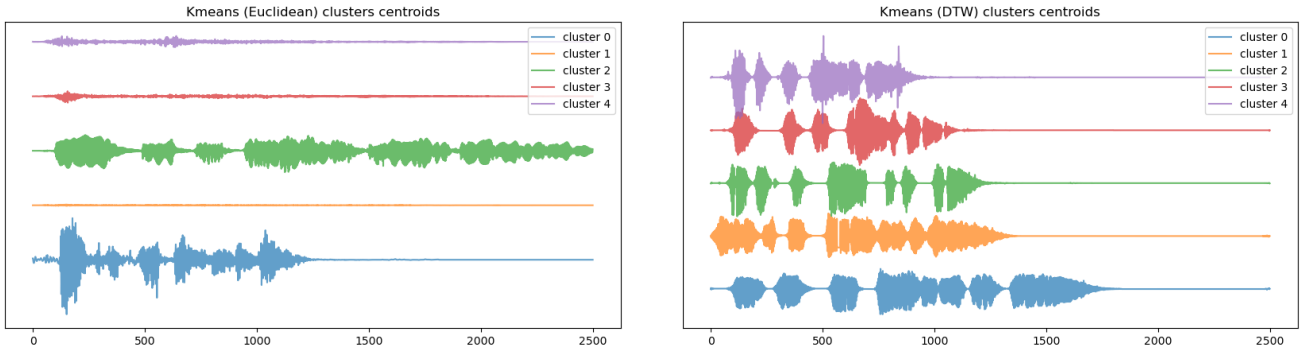



Figure 7.1: Kmeans clusters centroids

There are enormous differences between using Euclidean distance and dynamic time warping, as the plot shows. DTW is way more expensive to compute with such a big dataset, but it provides a more accurate result. Euclideans clusters are heterogeneous from every point of view, even for "easier" tasks like the distinction between songs and speeches. Moreover, it provides a few clusters with 10 or less observations (their centroid plot is similar to a single waveform, since they are basically the mean of just a few records). We only report results about the DTW version (7.2), because Euclidean distance cannot provide an useful distinction between different classes. However, emotions are still difficult to be detected. The inertia score using DTW is pretty high, 825.023.

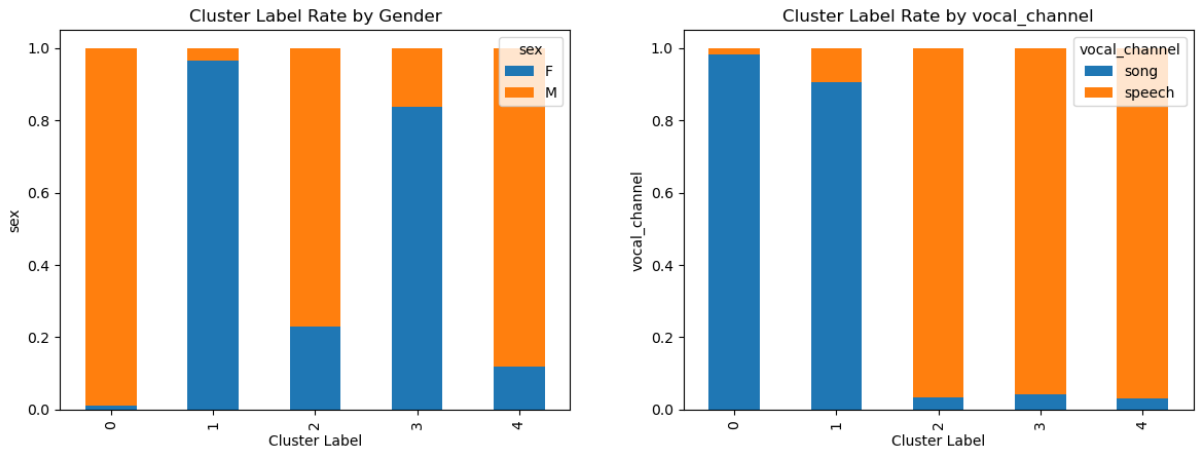



Figure 7.2: Kmeans (DTW) clusters label rates

Then, we also tried to visualize the clusters with different dimensionality reduction techniques, and below we reported the plot using t-SNE, as it shows a good division of the clusters.

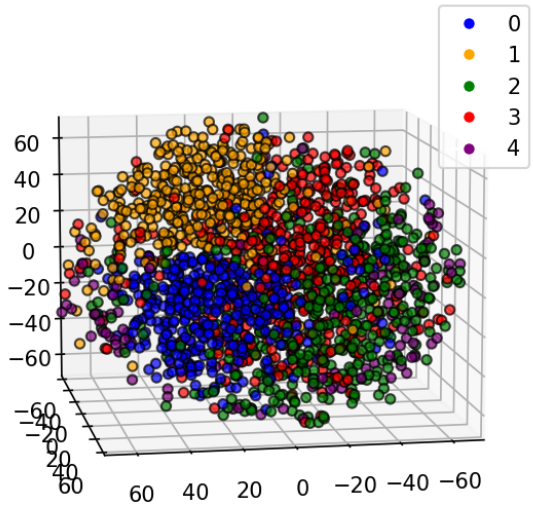



Figure 7.3: Kmeans clustering (t-SNE)

### 7.1.2 miniSOM

A different approach is the use of a self-organizing map (SOM). It is a special type of neural network which produces a low-dimensional representation of the input space (a map). For this task the library we exploited is miniSOM. Since this algorithm works with a grid of (som_x * som_y) clusters, we have chosen a 2x3 grid providing 6 clusters.

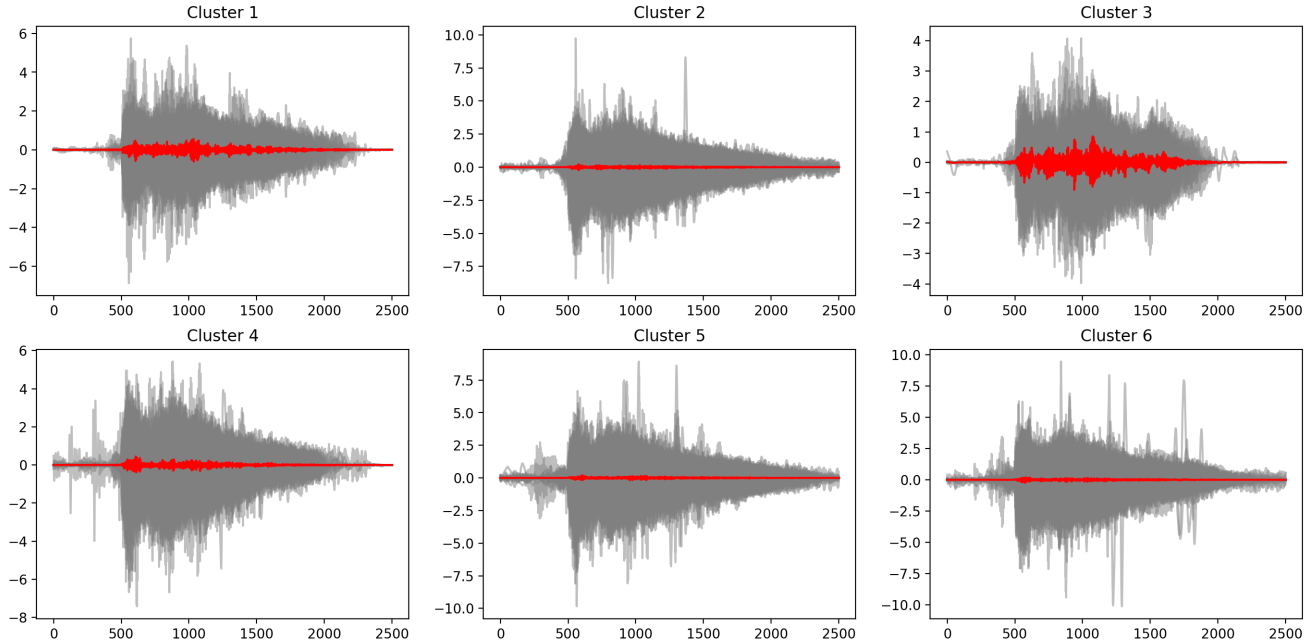



Figure 7.4: SOM clustering

Parameters were sigma=0.9 and learning_rate=0.4 with 100 iterations. Clusters average are coloured in red, while the gray plots show every time series in the cluster.

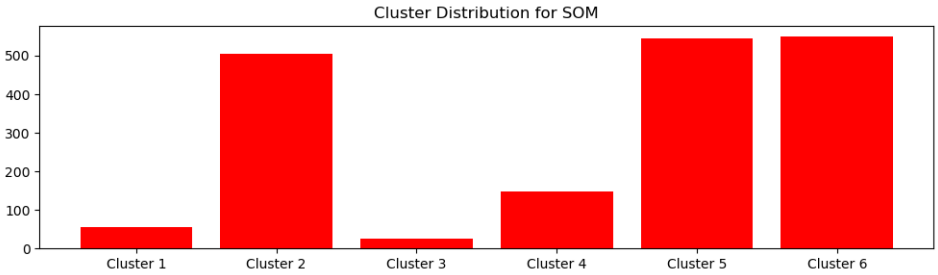
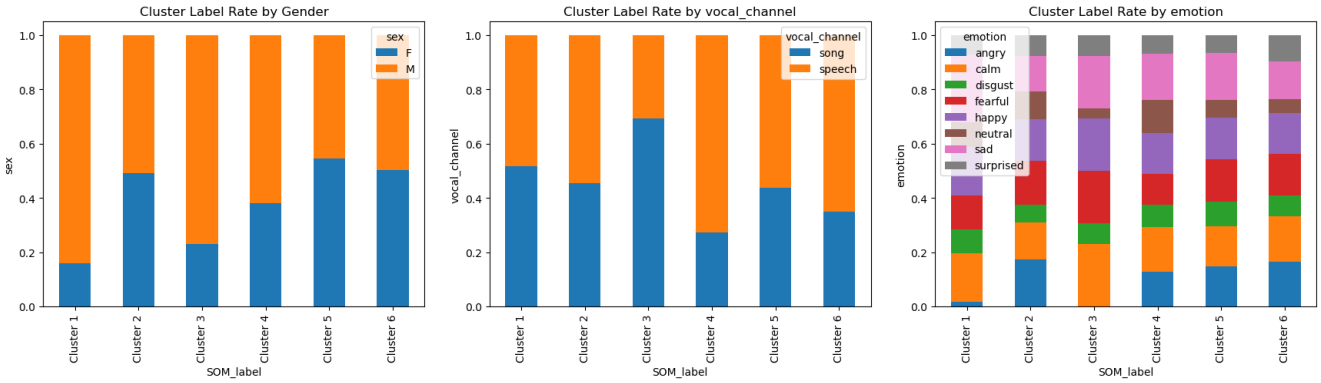



Figure 7.5: SOM clusters composition

As we can see, there are three main clusters involving the vast majority of observations. As for Kmeans, clusters cannot distinguish emotions, but minor clusters (like 1 and 3) are composed by males speakers, mainly (7.6a). It can also be noticed that cluster 1 and 3 almost never catch an 'angry' emotion.




(a) Sex (b) Vocal_channel (c) Emotions

## 7.2 Motifs & discords

We started from the DTW kmeans clustering for this task. After having computed the matrix profile for each cluster centroid, we analysed the motifs. Here is reported only the results for cluster 4 (mainly speeches from male actors) (7.7) for lack of space, since it is the most readable one. We used a window of size 15 to match the one we used for shapelets (8.2).

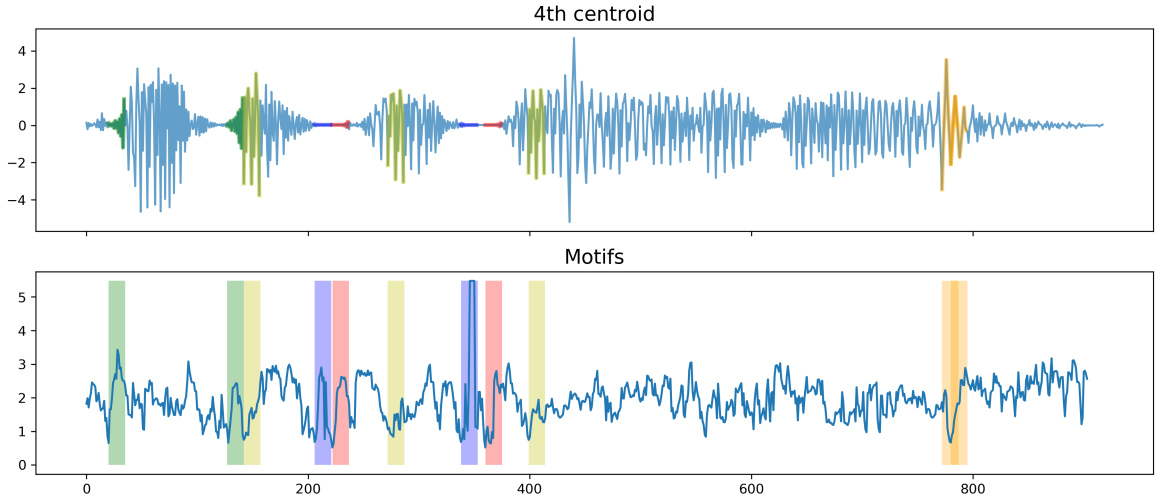



Figure 7.7: Cluster 4: matrix profile and motifs

The plot shows the interval 0:1000 where all the information is contained, ignoring the long tail at the end. It can immediately noticed that the yellow motif is repeated three times, while the orange ones are basically consecutive.

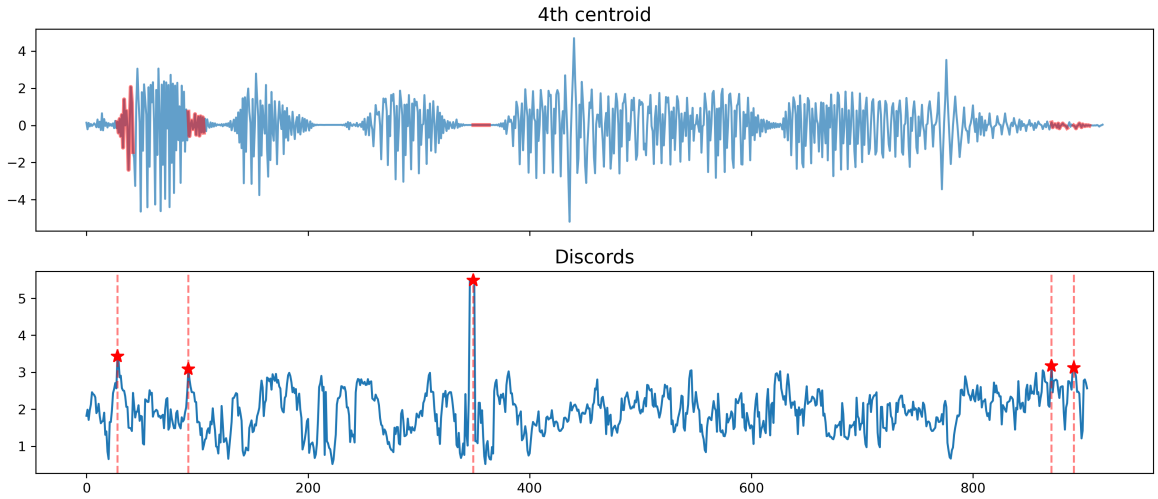



Figure 7.8: Cluster 4: matrix profile and discords

In 7.8 it can be noticed that there is one major peak in the matrix profile, meaning that it is the most appealing discord. Interestingly enough, it corresponds to a pretty flat sequence. Another interesting point is that two discords start in the middle of a motif sequence, including the one corresponding to the highest peak.

### 7.2.1 Relationship with shapelets

In this section we analyze the shapelets retrieved from Chapter 8.2.

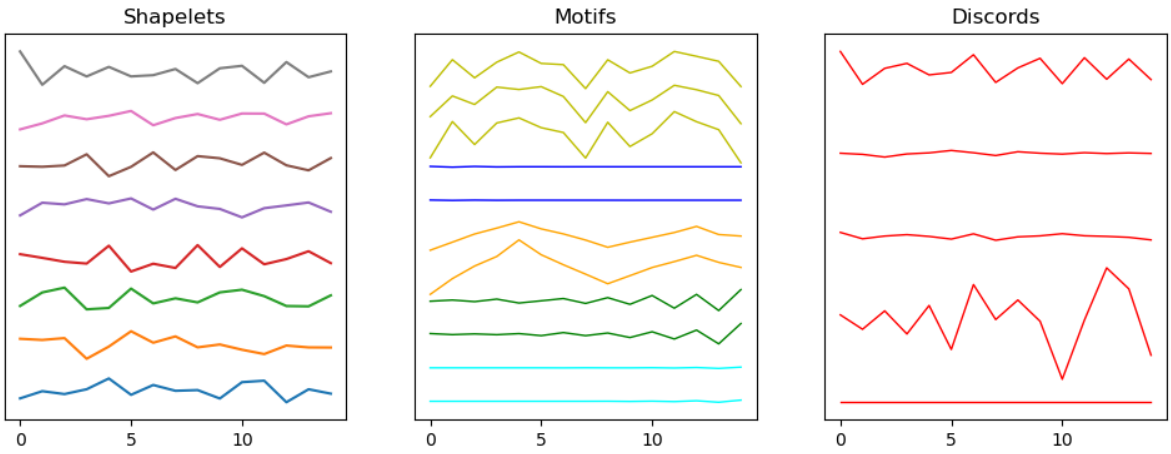



Figure 7.9: Shapelets compared with motifs/discords

At this point, we wanted to know if any of the shapelets eventually has a relationship with what we have found in this chapter. In Figure 7.9 we reported all the shapelets, motifs, and discords, and we will indicate them with numbers (from 0) starting from the bottom. At a first glance, it may seem that shapelet 7 is similar to discord #4. Comparing all the shapelets with all the motifs/discords (via DTW), we observed that shapelet 4's best match is motif #4, while the best match for shapelet 6 is discord #1.

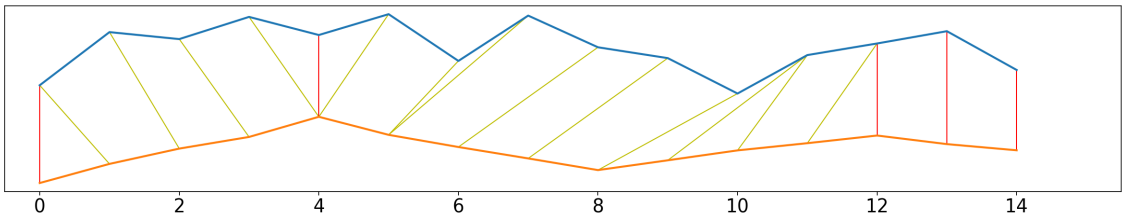



Figure 7.10: Best motif match (DTW distance = 5.8498)

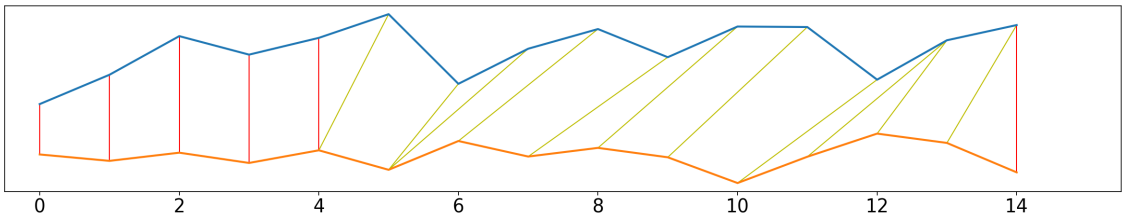
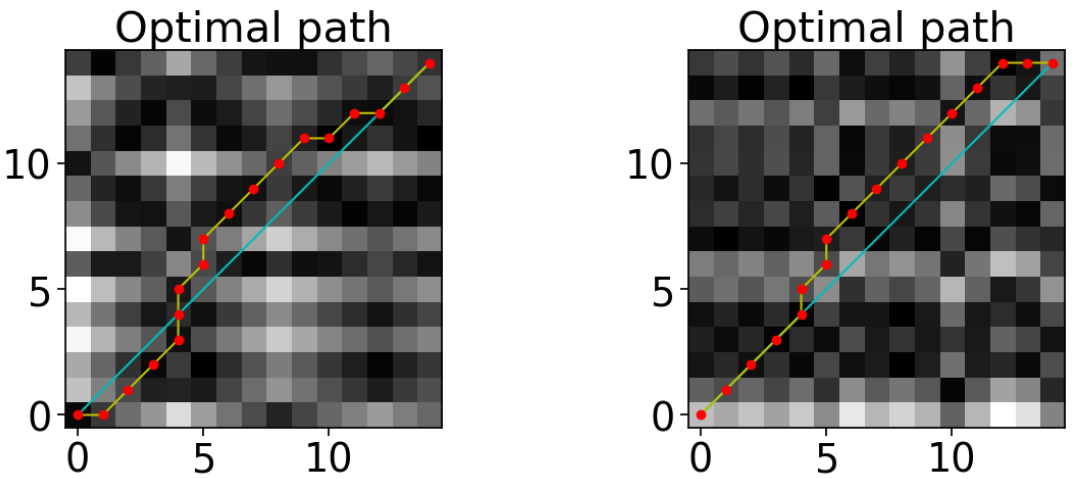



Figure 7.11: Best discord match (DTW distance = 7.9533)

The lines show similarities, which means that there is a certain degree of relationship between them. The path connecting the two shapes, in both cases, lies pretty much on the diagonal.




(a) shapelet 4/motif 4

(b) shapelet 6/discord 1

Figure 7.12: DTW optimal paths

# Chapter 8: Time series Classification

In this section, we classified the different emotions with several methods, so our target variable is the label encoded "emotion". As in previous classifications, at the end of the chapter, there will be a summary table containing the performance of three models. Our dependent variable assumes value 0 if the emotion is "angry", value 1 for "calm", 2 for "disgust", 3 for "fearful", 4 for "happy", 5 for "neutral", 6 for "sad", 7 for "surprised". First of all, we scaled the training and the test set using the TimeSeriesScalerMeanVariance with mean=0 and std=1.

## 8.1 KNN

The first algorithm that we used is the KNN with the Euclidean distance, with a grid search we obtained the best results with 'n_neighbors': 11, and the accuracy on both validation and test set is at 25%. Using DTW as distance, the results are significantly better, and the accuracy on the test set is 36%. Figure 8.1 shows the confusion matrix of the KNN with DTW, here we can see that the class predicted correctly most of the time is class 1 (calm), while the class predicted wrongly most of the time is class 2 (disgust) which is predicted a few times. Another interesting result is that when the model predicts class 6 (sad), it guesses often (42 times), but also misses a lot (25 times when in reality is 3, etc.).

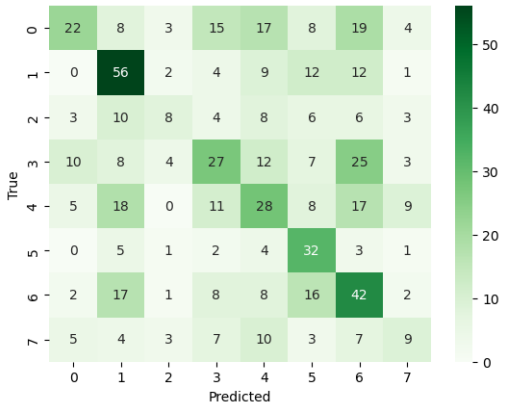



Figure 8.1: Confusion matrix KNN (DTW)

## 8.2 Shapelets

For the classification by the shapelets, we used the LearningShapelets algorithm (a new version of the ShapeletModel) from tslearn. Through the randomized search, we found our best hyperparameters: 'batch_size': 256, 'n_shapelets_per_size': 15: 8, 'optimizer': 'Adam' (with a learning rate of 0.01), 'weight_regularizer': 0.003. With these settings, using the model implemented in the library, we achieved 44% on the validation test and 37% on the test set, we decided to not report the confusion matrix, as this appears similar to that obtained from the KNN (Figure 8.1), but we will comment on the differences in the summary table at the end of the chapter.

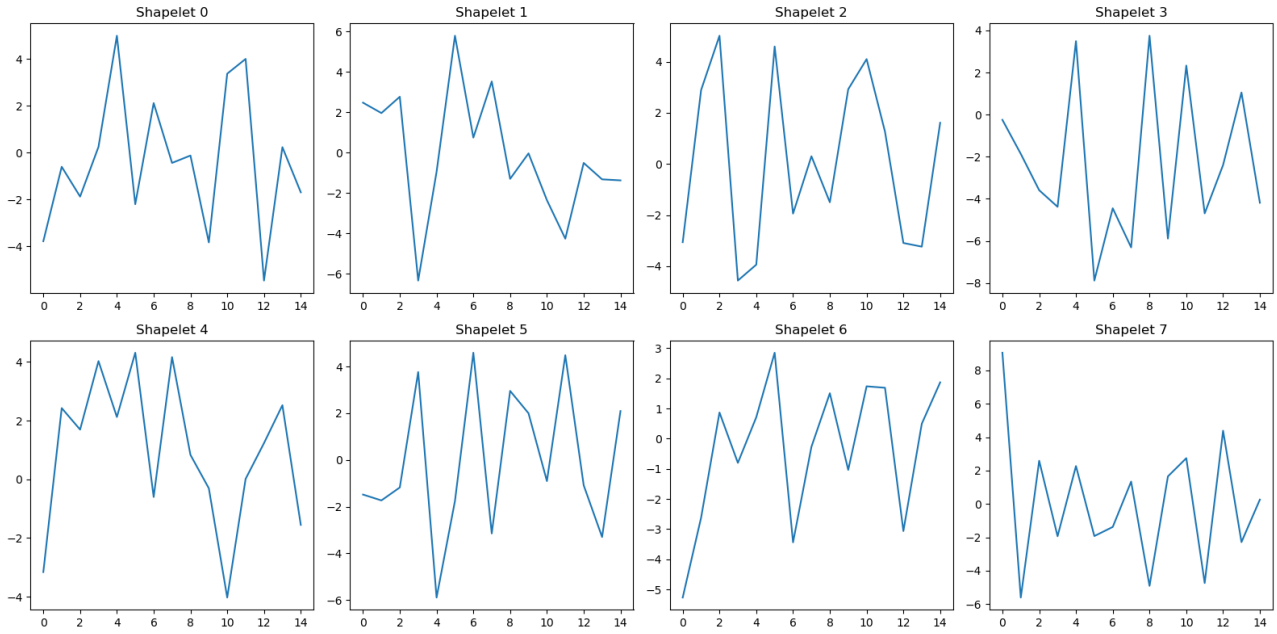
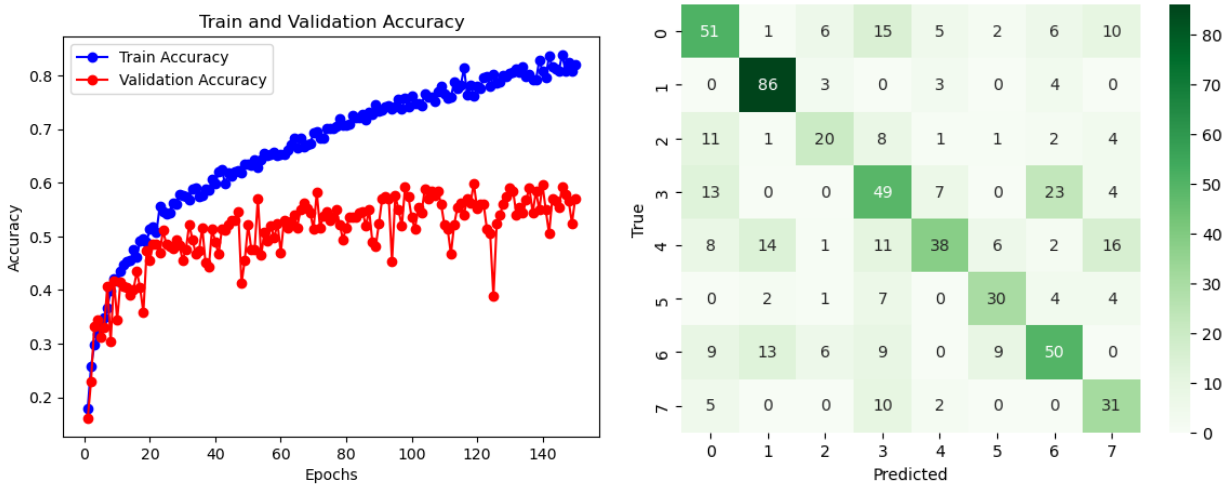



Figure 8.2: Shapelets

In Figure 8.2 there are 8 shapelets, and they differ in amplitude and position of the peaks, for example, shapelets 0, 2, and 4 have more or less the same range of motion on the y, but they differ for the peaks, while the others have all different ranges, the most unique is the shapelet 7, which has a signal that starts at 8 and then stabilizes on a lower range.

## 8.3 CNN

After KNN and shapelets, we tried different models, but the one that gave us the best results was the CNN. We tested several CNN builds, and in the end, we obtained the best performance with 4 convolutional layers with increasing filter sizes, each of them followed by batch normalization, ReLu activation, and dropout. After the convolutional layers, there is a global average pooling, followed by two dense layers with ReLU activation and dropout. The output layer is a dense layer with softmax activation for the classification, and the optimizer is Adam.




(a) Train and validation accuracy CNN

(b) Confusion matrix CNN

Figure 8.3a reported the train and the validation accuracy as epochs increase, from the plot we can see that after about 30 epochs the validation accuracy slows down, to reach the maximum (about 55%

accuracy) around the 50th epoch, the training accuracy, on the other hand, continues to rise. With this model we achieved the best results in the whole project, the accuracy is 57%, and the confusion matrix (Figure 8.3b) shows us again that the most guessed class is 1 (calm), and the class with less is 2 (disgust).

## 8.4 Comparisons

In this section, we decided to show the table containing the results of the KNN with DTW, shapelets, and CNN models.

Table 8.1: Classification reports

| | KNN (DTW) | | | Shapelets | | | CNN | | |
|---|---|---|---|---|---|---|---|---|---|
| Class | Precision | Recall | F-score | Precision | Recall | F-score | Precision | Recall | F-score |
| 0 | 0.47 | 0.23 | 0.31 | 0.31 | 0.42 | 0.35 | 0.53 | 0.53 | 0.53 |
| 1 | 0.44 | 0.58 | 0.50 | 0.48 | 0.61 | 0.54 | 0.74 | 0.90 | 0.81 |
| 2 | 0.36 | 0.17 | 0.23 | 0.05 | 0.02 | 0.03 | 0.54 | 0.42 | 0.47 |
| 3 | 0.35 | 0.28 | 0.31 | 0.35 | 0.29 | 0.32 | 0.45 | 0.51 | 0.48 |
| 4 | 0.29 | 0.29 | 0.29 | 0.40 | 0.47 | 0.43 | 0.68 | 0.40 | 0.50 |
| 5 | 0.35 | 0.67 | 0.46 | 0.46 | 0.48 | 0.47 | 0.62 | 0.62 | 0.62 |
| 6 | 0.32 | 0.44 | 0.37 | 0.35 | 0.23 | 0.28 | 0.55 | 0.52 | 0.53 |
| 7 | 0.28 | 0.19 | 0.23 | 0.26 | 0.25 | 0.26 | 0.45 | 0.65 | 0.53 |
| accuracy | | | 0.36 | | | 0.37 | | | 0.57 |
| macro avg | 0.36 | 0.36 | 0.34 | 0.33 | 0.35 | 0.33 | 0.57 | 0.57 | 0.56 |
| weighted avg | 0.36 | 0.36 | 0.34 | 0.35 | 0.37 | 0.35 | 0.58 | 0.57 | 0.56 |

First, we see that the first two classifiers have similar accuracy and that both have the lowest recall for class 2 (disgust), even if the classifier via shapelets is basically 0. The second classifier also has low precision for class 2, while the KNN has its precision averaged with the other classes. Then, we note that the CNN has much higher accuracy than the other 2 models, and that the class 1 (calm) is predicted correctly almost all the time (recall 0.9) and has also low false positives (precision 0.74).

# Chapter 9: Explainable AI

In this chapter we use two of the most famous explainable AI methods to asses the behavior of our best classifier (see 4.2) (Neural Network) presented in chapter 4. In particular, we will try to give a global representation of the behavior of our model and then we will focus on some misclassified records to discover possible biases.
On the first hand, we have used SHAP values to get a global picture of the impact of each attribute to the final output. More specifically, we have adopted the DeepExplainer method provided by the shap library that allows a fast computation of shapely values of deep learning models to get a global explanation of the model. The summary plot is presented below:

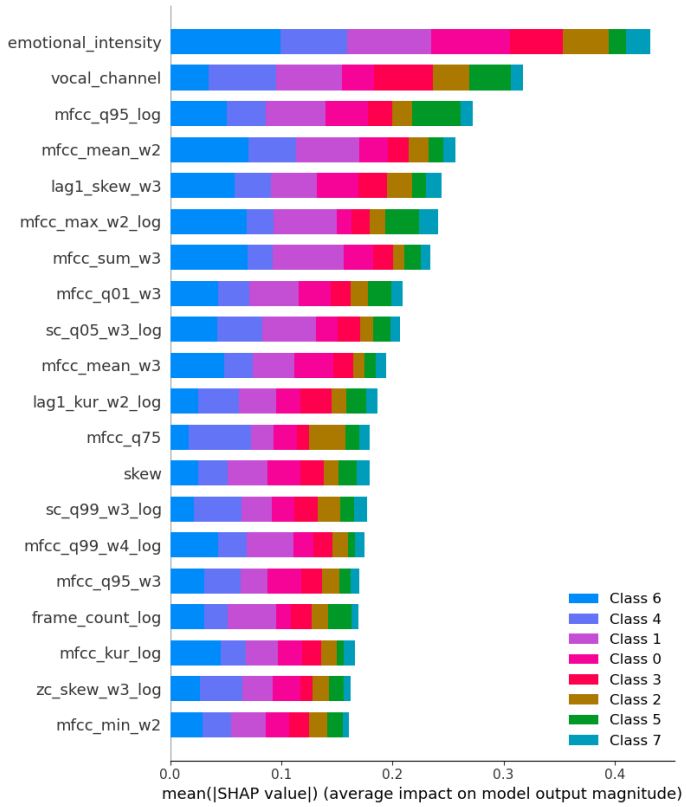



Figure 9.1: Summary of global explainability

From the plot we can see that "emotional intensity" has the highest average impact on the classification. The most impact is, in particular, on class 6 (sad) then the second most influent feature is "vocal_channel" as can be expected since some emotions are not present for song audio files.
Next, we have checked which is the most misclassified class and it turns out that it is the "sad" class (class 6), and in particular, the most of the time it is misclassified with class "calm" (class 1). For this reason we have decided to use LIME, which is a local explainer, to asses the reason behind this behaviour of our model.
We have picked one misclassified instance and by using LIME we have analyzed the 2-top label for that instance. It turns out that for both classes the attribute with the highest impact is "emotional_intensity" which agrees with the global explanation but there are few more variables that have contributed to the misclassification such as "vocal_channel" and other continuous attributes. However, by computing the $R^2$ of the linear model used by LIME to return the feature importance we observe poor results since it is around the 20%.

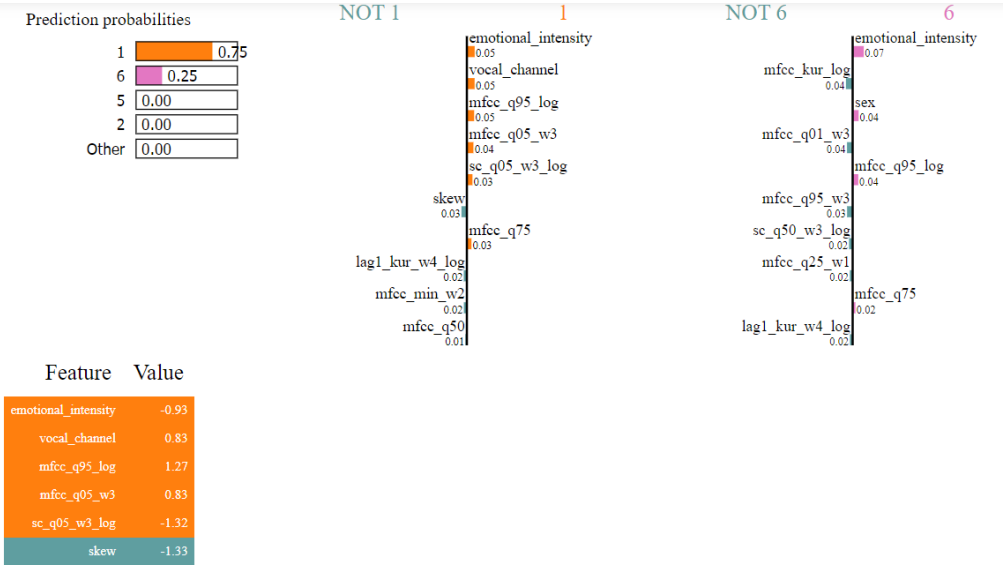



Figure 9.2: Local explanation