UNIVERSITÀ DI PISA

Data Mining II Project

Michele Velardita
578770

A.Y. 2022/2023

Contents

Chapter 1

Data Understanding and Preparation

1.1 RAVDESS Dataset

The RAVDESS dataset contains audio data recorded from 24 professional actors, (12 male and 12 females), vocalizing, in a spoken or sung way, two different statements. Spoken audio includes 8 emotions: *neutral*, *calm*, *happy*, *sad*, *angry*, *fearful*, *surprise* and *disgust*. Sung audio contains 5 emotions: *calm*, *happy*, *sad*, *angry* and *fearful*. Each expression is acted at two levels of emotional intensity (normal, strong). In total the dataset contains 2452 records.
Statistical values such as sum, mean, standard deviation, min, max, kurtosis, skewness and quantiles were extracted for each audio track. Subsequently, every audio file underwent a transformation process involving the following techniques:

- Lag1: calculates the difference between each observation and the previous one, expressed as the difference between observation(t) and observation(t-1).
- ZC (Zero Crossing Rate): measures the frequency of sign changes in a signal over time.
- MFCC (Mel-Frequency Cepstral Coefficients): generates a representation of the short-term power spectrum of a sound, mimicking human auditory perception.
- SC (Spectral Centroid): identifies the center of gravity of the spectrum of a sound.
- STFT (Short-Time Fourier Transform) Chromagram: a visualization technique representing the distribution of energy in different frequency bands.

In addition, the signal is divided into four windows of equal size. The same feature extraction procedure is then performed on each window.

The dataset is divided in two files: Train and Test with respectively 1828 and 624 records.

1.2 Data Cleaning

The first step was to look for (and remove) features that do not provide any information, i.e. those features with variance equal to 0. It is found 50 continuous features and one categorical feature (*modality*). Also, no attributes containing null values were found. The feature *filename* has been eliminated as it does not contain useful information that can be used for subsequent analyses. The next step was to remove all the highly correlated features. It was chosen to remove

all the features that have a correlation score higher or equal than 0.98 using the spearman formula. In total 99 features were removed. After the cleaning operation 152 features were removed, leaving the dataset with 282 attributes.

1.3 Data Normalization

Exploring the dataset, the presence of outliers (via boxplot) and a heterogeneous distribution of the different features were found. The Standard Scaler suffers from outliers, as it uses the mean to normalize the data. The PowerTransformer transforms a non-normal distribution into a more Gaussian one. It is particularly useful for our data as many features exhibit skewness or heavy-tailed distributions; but given the presence of extreme values that can have a strong impact on the scaler, the choice fell on the **Robust Scaler**, less sensitive to the presence of outliers.
Furthermore, the categorical variables were all mapped to numeric values.

Chapter 2

Outliers

This chapter will review the various methods used during the outlier detection processes and discuss their results. For visualize the data in a 2D plot it was used UMAP, a dimension reduction technique similat to t-SNE but preserve better the local structure of data. Being a stochastic method, different executions may lead to different outputs, so the random_state parameter was used with the same seed for all executions.

2.1 Convex Hull

The first method used was the convex hull depth base approach. It was necessary to implement a function that calculate the convex hull and eliminates recursively the outermost points. This approach showed criticality from the beginning, as convex hull being extremely complex to calculate in high dimensions, it was not possible to use the whole dataset as is, so a dimensional reduction technique was used to reduce the data into a manageable dimension. For this case it was used a UMAP reduction into 2 dimension. Another problem with this approach is, being a labeling method, it is not possible to identify the top 1% percent of outliers; the outermost points in the dataset are identified as outliers.

In figure 2.1 the outliers found by the method are marked in red. The depth parameter used was 3. In total, 72 points of the dataset were marked as outliers. As we can see in figure 2.1 only the outermost points are marked as outliers, this can lead to errors because some outliers aren't necesssary at the boundaries of the dataset or viceversa, some inliers may be on the boundarie

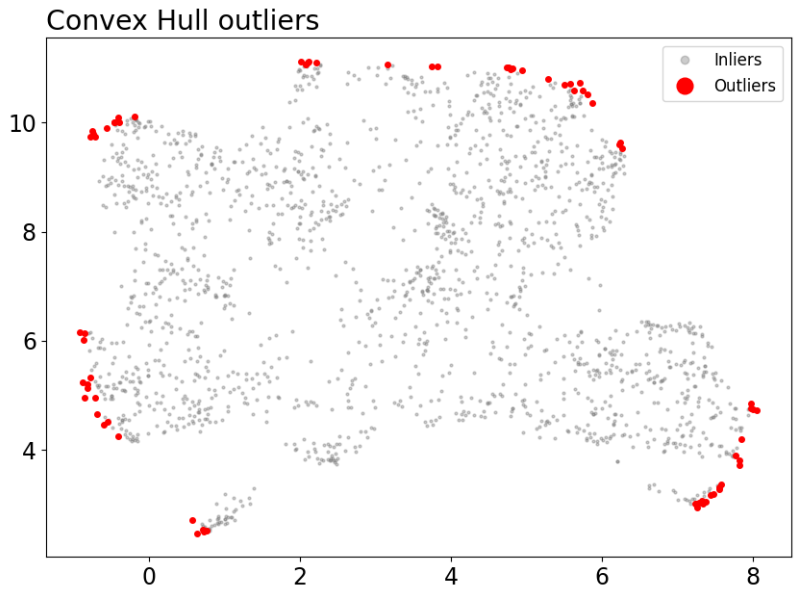

Figure 2.1: Outliers found by Convex Hull approach with depth 3

of the dataset. The application of this method was only a didactic exercise since the down sides are far more than the upside.

2.2 KNN

The next method tried for identifying outliers was the scoring based on KNN distances. The parameter used for the algorithm were n_neighbors=35 and method=mean. The method identified in total 157 points as outliers, of this, 19 were the top 1%. In figure 2.2 the outliers are represented in red.

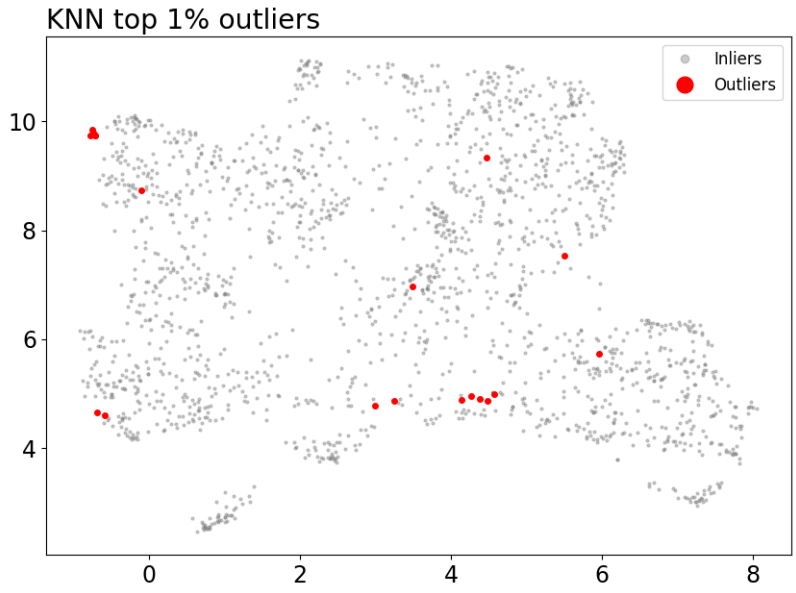

Figure 2.2: Outliers found by KNN approach with n_neighbors=35 and method=mean

The critical aspect of this method are the same of a KNN classifier i.e. curse of dimensionality, for this reason is not the best choice for our case.

2.3 LOF

The outliers found by the lof method were similar to those found earlier by the KNN. The parameter used for the LOF were n_neighbors=35. The method found 72 outliers in total, of this 19 were the top 1%.

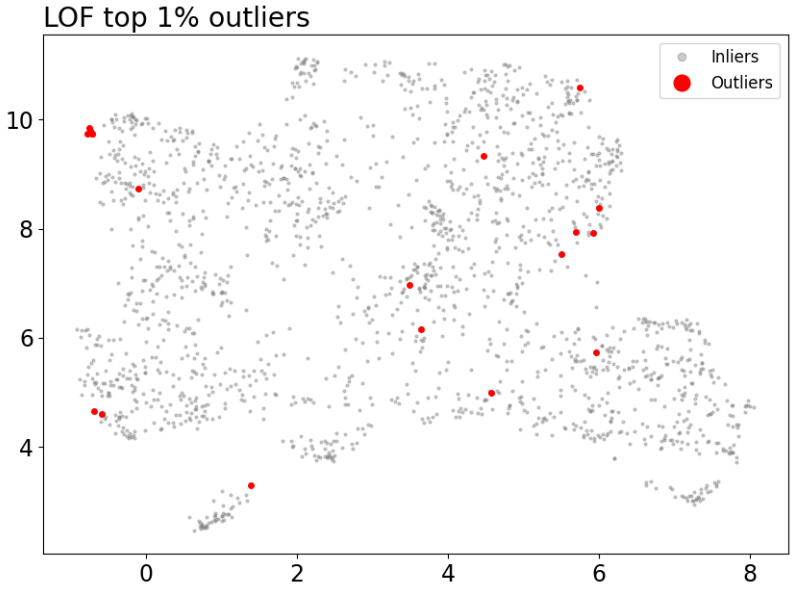

Figure 2.3: Outliers found by LOF

2.4 Isolation forest

In comparison to the other method used, isolation forest found a different set of outliers. Given that is more robust to high dimensional data, it makes it more suitable for our case. The method found 60 outliers in total, of this 19 were the top 1%.

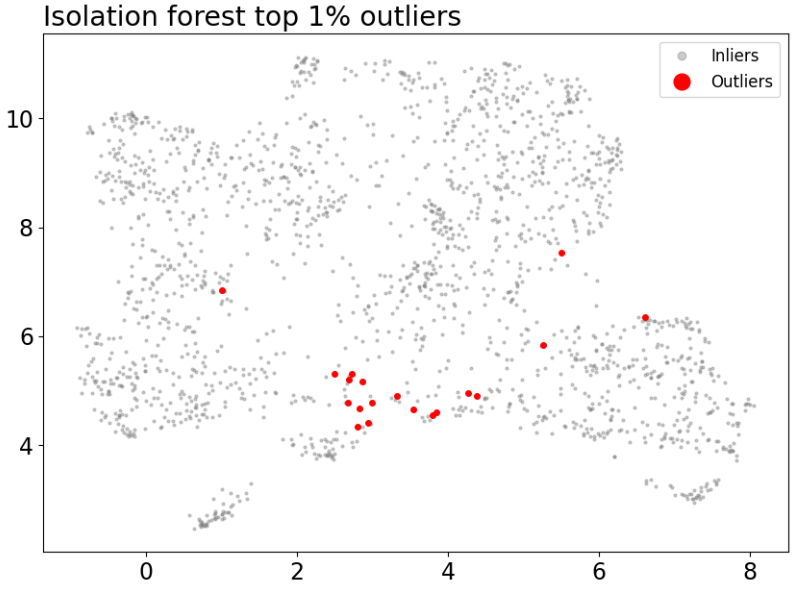

Figure 2.4: Outliers found by Isolation forest

2.5 Conclusion

From the figures shown above, it can be seen that the various methods return different results, in particular it can be seen that the isolation forest differs greatly from the results of the KNN and LOF.

Regarding the treatment of outliers, it was decided not to perform any elimination and keep them in the dataset; this decision starts from the type of data we're working on. The dataset, containing sentences recited by actors and recorded in a controlled environment, do not contain unexpected values, the data were created "artificially" and all in the same and controlled way so we given for granted that all the features were part of possible obtainable data. Furthermore, their removal did not lead to significant improvements in the performance of the classifiers (Decision Tree and KNN).

Chapter 3

Imbalanced Learning

In this section a classification task on an unbalanced variable is defined, applying different rebalancing models and analyzing the results obtained.

3.1 Feature and resampling techniques selected

The variable taken into consideration is emotion. In particular, it was decided to try to discriminate the 'surprised' emotion from the others, thus passing from a categorical variable with 8 possible outcomes to a binary one. In this case the dataset is unbalanced, as the surprised emotion is present in 8% of the records. Various oversampling methods (random, smote, svm-smote, adasyn and smote-nc) were then applied in order to analyze the performance of the classification models in relation to the different algorithms.

3.2 Results

Applying two different models (decision tree and knn) to predict the rare class, significantly lower results than the most frequent class were obtained, obviously due to the discrepancy between the number of records for each class label. The figure 3.1 shows the improvements achieved with two different classifiers after applying different oversampling techniques.[1].

[1]The decision tree achieved an F1-score of 30% on the 'surprised' class and 94% on the most common class. On the other hand, the KNN model achieved an F1-score of 6% on the less frequent class and 95% on the most frequent one

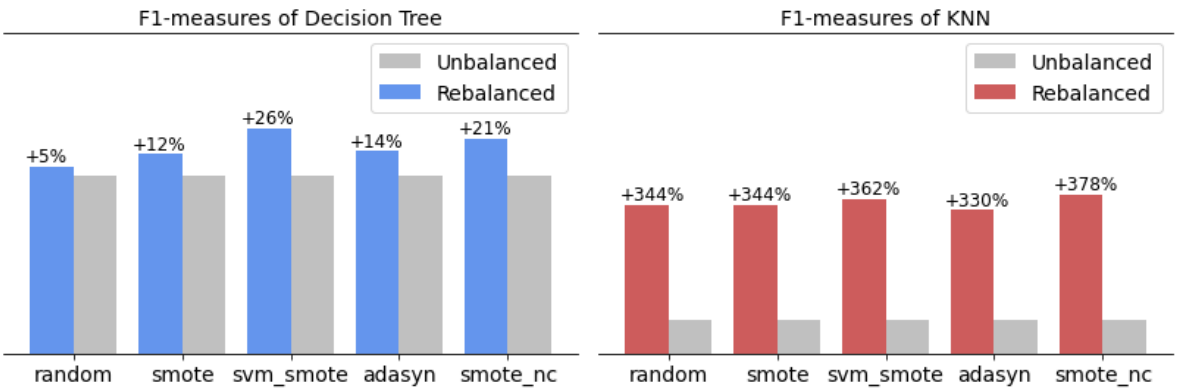

Figure 3.1: Comparison of f1-measures on *surprised* emotion before and after applying different oversampling technique.

In the case of the KNN classifier, oversampling can be particularly effective because the algorithm relies on finding the k-nearest neighbors of a sample to make predictions. If the minority class is underrepresented, the KNN algorithm may fail to find enough neighbors from that class and perform poorly. Oversampling can help to address this issue by increasing the number of minority class samples, which in turn increases the likelihood of finding enough neighbors from that class.

Instead, from the analysis of the values obtained with the decision tree it is possible to observe more relevant behaviors. The **random oversampling** method involves replicating data from the minority population, which boosts the data quantity but fails to offer any novel information to the model and, for this reason, it is the worst among the techniques applied. **Smote-nc**, unlike smote, also works with categorical features, thus being able to obtain slightly better results than the classic smote. In contrast to other over-sampling techniques, the **svm-smote** approach focus on minority class instances located near the decision boundary. This region is critical for accurately determining the decision boundary; additionally, the method generates new instances in a way that expands the minority class area towards the side of the majority class where few majority class instances exist[1]. This led to a better result than the other methods. **Adasyn** generates synthetic data based on data density, with more data created where minority class density is low. Outliers in low-density data may result in excessive attention on those regions, leading, as in this

case, to a less than optimal result if we look at the one achieved by svm-smote.

Chapter 4

Advanced Classification

In this chapter, different classification methods and their results will be studied. In particular, the generalization quality of the Logistic Regression will be analyzed using two different regularization techniques (Lasso and Ridge). Two different ensembling methods will then be compared, and we will try to improve their performance by reducing the number of features through logistic regression. Finally, the quality of generalization and the performance of a neural network on a classification task will be presented.

4.1 Logistic Regression and Ensemble Methods

In this paragraph, we analyze the application of Logistic Regression to a classification task with an unbalanced target variable, rebalanced using the svmsmote technique. Two different regularizers, Lasso and Ridge, were tested with varying regularization strengths ranging from 10^{-4} to 10^{4}. The accuracy and matthew's correlation coefficient (MCC)[1] were plotted for both the training and validation sets to study the model's overfitting and ability to generalize (figure 4.1). The results show that with low regularization strength, Lasso generalizes well, with little difference between the training and validation sets, while Ridge tends to overfit more.

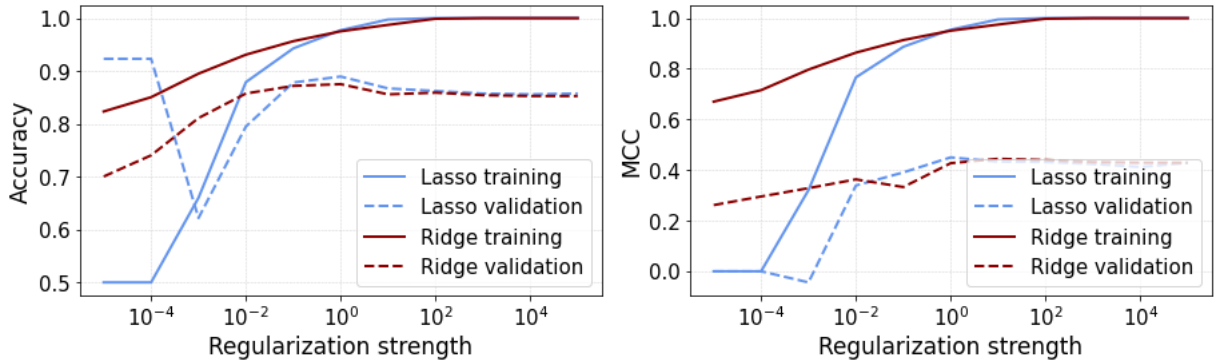

Figure 4.1: Comparison in terms of accuracy and MCC between the two regularization functions, Lasso and Ridge

[1]The Matthew's Correlation Coefficient (MCC) was used as a performance metric due to the imbalanced nature of the target variable. It is a reliable metric that considers both true positives and true negatives, even with imbalanced classes, and summarizes the performance of a binary classifier by taking into account all four possible outcomes.

However, as the regularization strength increases, both models achieve similar levels of performance and overfitting. One possible reason for this is that when the regularization strength is low, the Lasso model tends to completely remove less important features, leading to better generalization. On the other hand, the Ridge model may retain all the features, leading to overfitting. As the regularization strength increases, both models tend to shrink the coefficients towards zero, leading to similar performance and overfitting levels.

In order to improve the performance of the classification models, Logistic Regression[2] with Lasso regularization was used to remove features that had coefficients equal to zero. A total of 94 features were removed using this method. The resulting dataset was then used to train and test two ensemble methods, **RandomForest** and **AdaBoost**. The performance of these models was evaluated using the f1-measure of the rebalanced class. The experiments, summarized in the figure 4.2, showed that removing irrelevant features through Lasso regularization can lead to improvements in the performance of both ensemble methods.

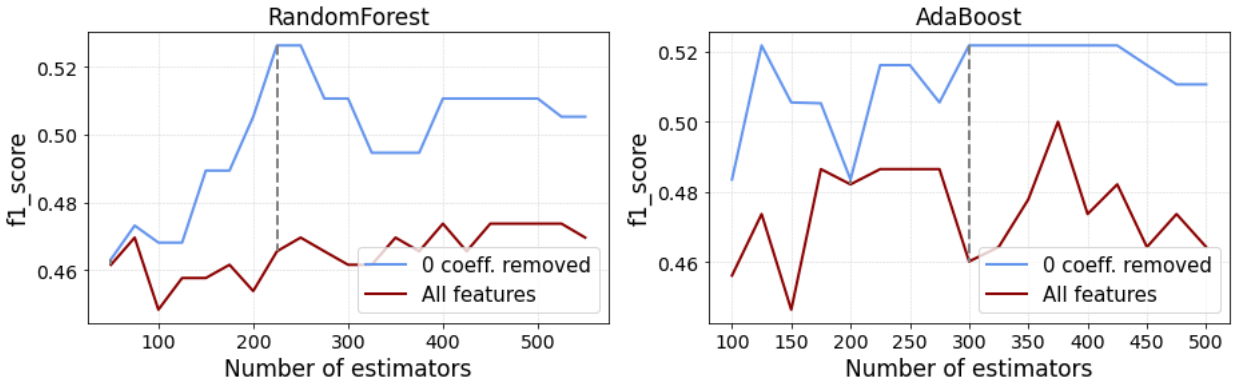

Figure 4.2: Performance analysis of ensembling models before and after the removal of zero coefficients

For RandomForest, increasing the number of estimators resulted in an improvement in the f1-measure, and the difference in performance between the model with all features and the model with features removed became more significant for larger values of estimators. This may be due to the fact that RandomForest combines the predictions of multiple decision trees, and removing irrelevant features can improve the performance of individual trees. In contrast, the difference in performance between the AdaBoost model with and without features removed was more irregular. AdaBoost assigns higher weights to misclassified samples in subsequent iterations, and removing irrelevant features may not have a consistent effect on the classification of all samples. Nevertheless, an overall improvement in performance was observed for both models.

[2]Libinear was used as the solver and a regularization strength of 0.5

4.2 Neural Network

In this study, a neural network[3] was applied to a balanced classification task, aimed at classifying the *emotional_intensity* feature. Initially, a simple neural network with an sgd solver was used, resulting in poor performance as seen in Table 4.1. To improve the results, a light GridSearch was performed to find a suitable starting point for tuning the hyperparameters of the neural network and studying its behavior with varying parameters. The GridSearch returned a better classifier with significant improvements in performance. Hyperparameter tuning began by exploring the structure of the neural network, including the number of hidden layers and the number of neurons in a single layer, as shown in the figure 4.3. The results indicates that as the number of layers increases, the neural network becomes more complex and produces worse results in terms of accuracy.

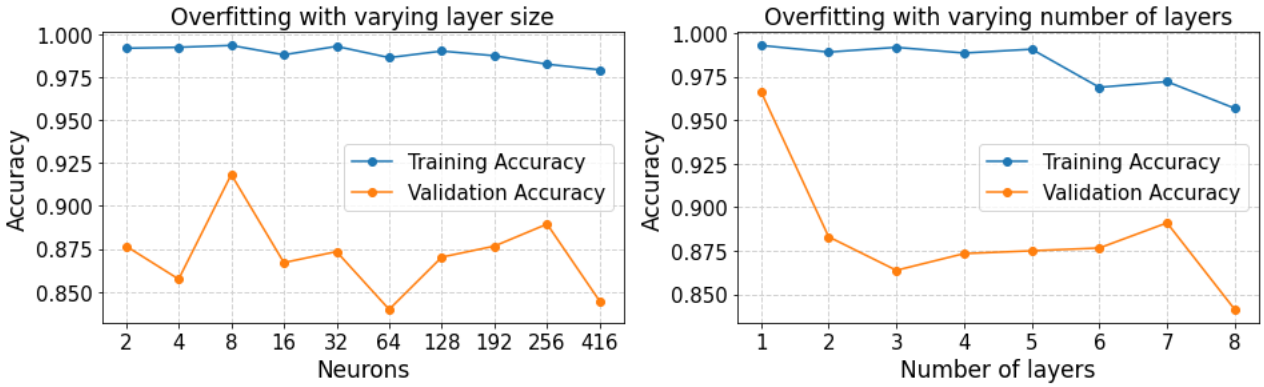

Figure 4.3: Performance analysis of the neural network by varying its structure

Increase the complexity of the neural network can lead to the model becoming too specialized to the training data and not generalizing well to new data. As the number of layers increases, the network may learn to memorize the training data instead of learning useful patterns. Additionally, more layers can make it more difficult for the network to optimize the parameters effectively, leading to slower convergence and potentially worse performance.
Regarding the number of neurons, the experiments shows that the optimal number of neurons is 8, as determined by the GridSearch, returning the best results in terms of accuracy and minimum overfitting among the limited set of possibilities explored.

As shown in Figure 4.4, increasing the number of epochs leads to improved accuracy on the training set. However, it also results in a reduced ability of the model to generalize to new data, as indicated by the widening gap between the training and validation curves. This can be attributed to overfitting, where the model becomes too complex and starts to fit to noise in the training data rather than the underlying patterns.

[3]It was decided to use the neural network model implemented in the scikitlearn library, because, despite missing some of the advanced features and performance optimizations present in tensorflow, it provides a simpler and more accessible way to use neural networks for our purpose

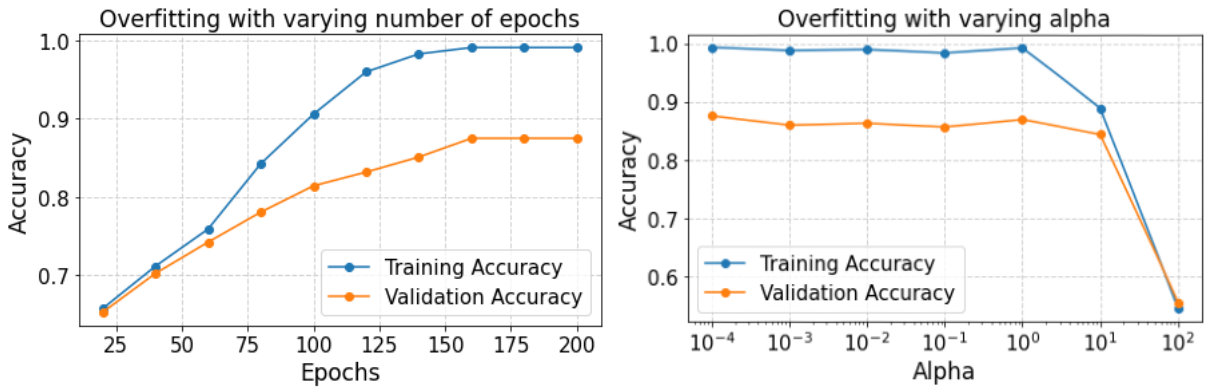

Figure 4.4: Performance analysis of the neural network by varying the number of epochs and the regularization strength (alpha)

Regarding the regularization strength, the plot shows that there are no significant changes in either accuracy or overfitting up to a certain value. Beyond this value, the model's ability to generalize reaches its maximum, but there is a drastic decrease in performance. This is due to the fact that the model becomes too simple and underfits the data. Therefore, it is crucial to find a balance between underfitting and overfitting by selecting an appropriate value for the regularization strength.

Neural Network	**Precision**	**Recall**	**F1**
Simple NN	73%	72%	72%
GridSearch NN	90%	90%	90%
Tuned NN	93%	93%	93%

Table 4.1: Different neural networks performance

Chapter 5

Advanced Regression

In this section, we apply two regression models, Gradient Boosting and Support Vector Machine (SVM), to a regression task. The chosen target variable is the *mfcc_q05* feature (0.05 quantile of the mel-frequency), which has a high variance and therefore it can be more informative to evaluate the regression models, as this can provide a more challenging and realistic test of the ability to make accurate predictions. We first discuss the individual results, and then compare the

two models based on their performance metrics and overfitting. This comparison will provide insights into the strengths and weaknesses of these two widely used regression techniques.

5.1 Extreme Gradient Boosting (XGBM)

XGBM was used for the gradient boosting approach. Tuning the hyper-parameters of the regressor was done through a grid search with a cross validation using only the Train dataset. The parametere tested were: *max_depth* ranging from 3 to 6, *learning_rate* testing the values 0.001, 0.005, 0.01, 0.05, 0.1, *reg_lambda* ranging from 1 to 5 and *n_estimators* testing 50 to 300 with increment 50. The grid search was performed using only the Train dataset with cross validation split of 5. Sugnu puppu.

The best values found were learning_rate: 0.05, max_depth: 4, n_estimators: 250, reg_lambda: 1. After the best parameters were found, the regressor was refitted using all the Train dataset and tested on the Test dataset. The results obtained in the test dataset were: **R2**: 0.941, **Mean squared error**: 0.034 and **Mean absolute error**: 0.118

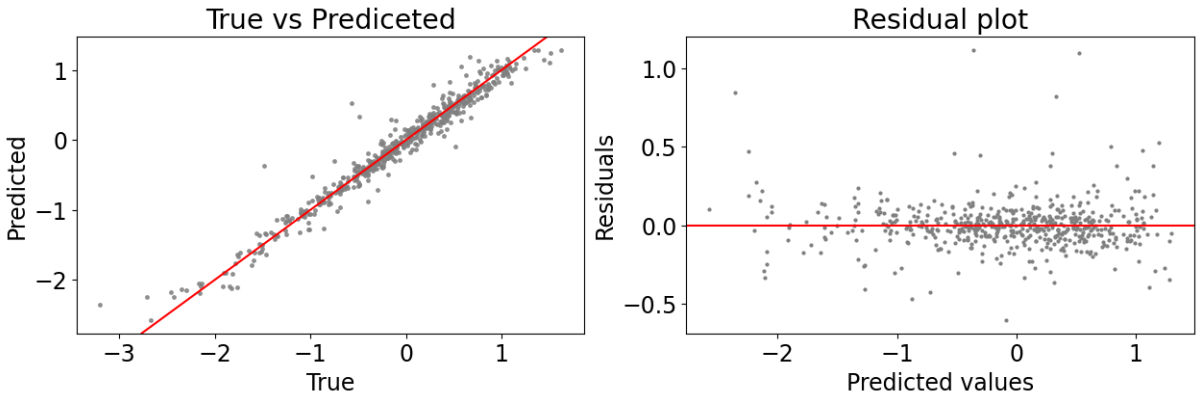

Figure 5.1: True vs predicted and Residual plots

5.2 Support Vector Machine

In this study, SVM for regression was applied and analyzed using R2 and MSE as metrics. The computational cost and general performance of the three kernel forms, i.e. linear, polynomial and rbf, were compared. It was found that the linear kernel took much longer time compared to the other two[1], and thus, it was decided to test only polynomial and rbf kernel. GridSearch was performed to obtain a good starting point for hyperparameter tuning, and the results showed that rbf kernel outperformed polynomial kernel in terms of performance.

The parameter **epsilon** determines the tolerance for errors. When epsilon is too small, the margin becomes too narrow and too many support vectors are used, which leads to overfitting. On the other hand, if epsilon is too large, the margin becomes too wide, which reduces the ability to fit the training data well[4]. In this case, as shown in figure 5.2, when epsilon was increased

[1]The reason behind the difference in computational cost between the linear kernel and the other two (polynomial and rbf) might lies in the mathematical formulation of the kernel functions. The linear kernel simply computes the dot product between two vectors, which can be a time-consuming operation in the case of high-dimensional datasets, as is the case here

gradually, the performance (but also the level of overfitting) of the model improved slightly. However, after a certain point, the performance of the model started to deteriorate drastically, suggesting that the model has become too sensitive to errors.

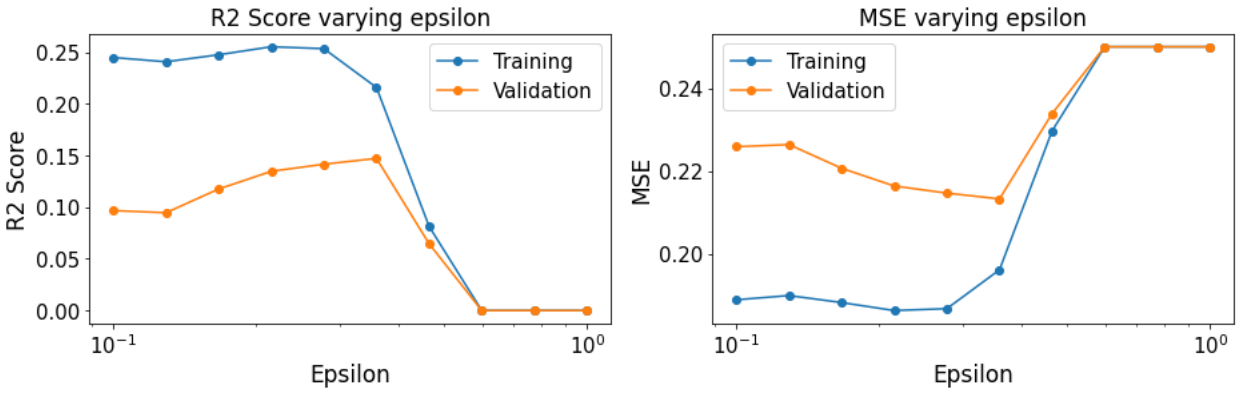

Figure 5.2: Performance analysis (polynomial kernel) as the epsilon amplitude varies

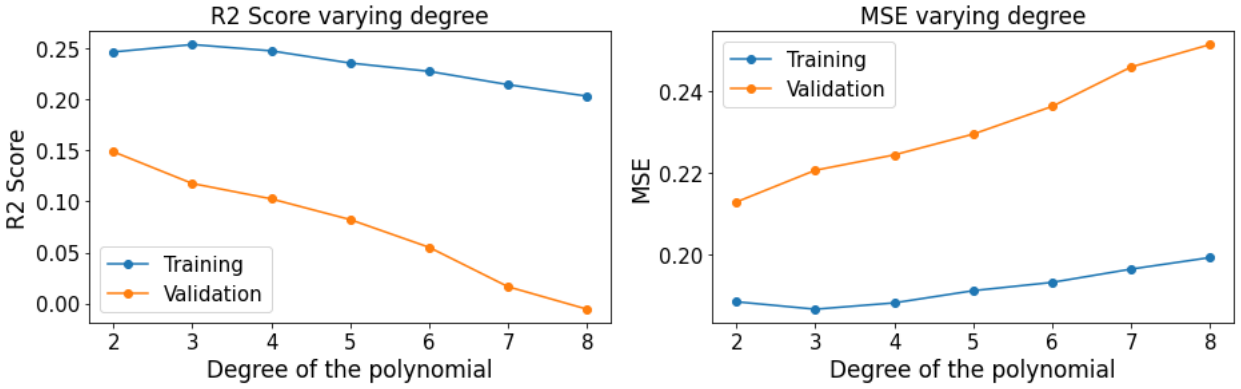

Figure 5.3: Performance analysis (polynomial kernel) as the polynomial degree varies

Increasing the degree of the polynomial leads to a more complex decision boundary, which can potentially fit the training data very well. However, a higher degree polynomial can also result in a weaker generalization ability (figure 5.3).

The C parameter in SVM is a penalty parameter that assigns a weight to the misclassified or margin-violating instances. Setting a high value of C leads to a reduction in misclassifications and a decrease in the number of support vectors. However, in case of non-linear kernels, setting C to a very high value may result in overfitting.

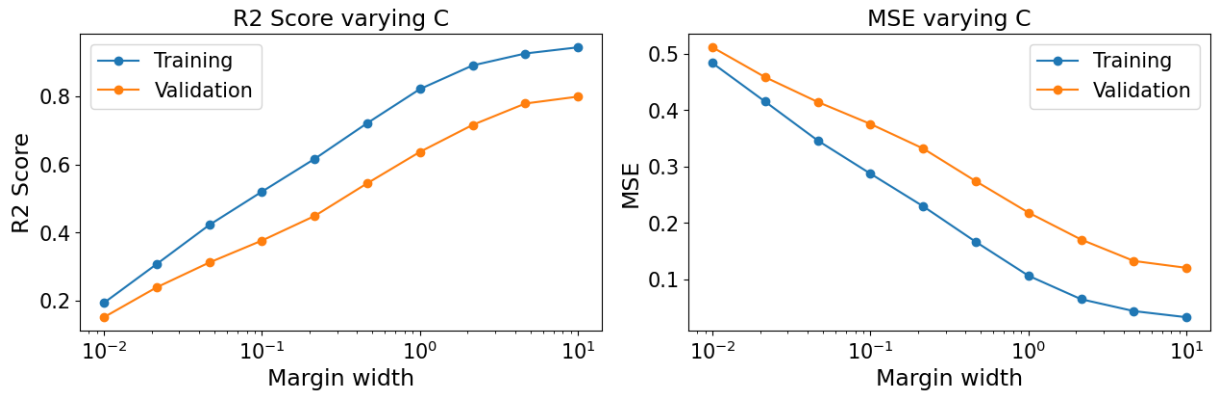

Figure 5.4: Performance analysis (rbf kernel) as the margin of the decision boundary increases

In fact figure 5.4 shows that, also in our case, as the parameter increases, there is an improvement in terms of R2 and MSE but a slight decrease in the ability to generalize.
After having performed all the experiments explained above, discrete levels of performance were reached, as shown in the table 5.1.

5.3 Comparison

XGB is a powerful ensemble learning algorithm that combines multiple weak learners to form a strong predictor. It can handle complex interactions between features and has been shown to perform well on our dataset. On the other hand, SVM is a simpler model that may not be able to capture complex patterns in the data as effectively. The difference in performance between the two models has been attributed, for this reason, to the nature of the regressors. Table 5.1 provides a comparison between the two algorithms used.

Regressor	R2	Mean squared error	Mean absolute error
XGBM	0.941	0.034	0.118
SVM	0.800	0.121	0.260

Table 5.1: R2, Mean squared error and Mean absolute error of Regressors

Chapter 6

Time Series: Data Understanding and Preparation

So far, all efforts have focused on a high-level structural similarity between the records, i.e. on all the statistical variables extracted from the audio files. In this chapter, we will focus on a different approach to analyze the similarity of data and conduct the experiments, based on the shape of the audios. The main difference between shape-based similarity and structure-based similarity is that the first methods focus on the temporal alignment and shape of the data, while the other one focus on the high-level content of the data. As a result, we expect to obtain different results from the studies conducted so far.

6.1 Trimming

Looking at the audio files, a discrepancy between the length of the graph and the actual duration of the vowel was noticed. This is due to the nature of *librosa* library of having extra time duration space to accommodate noise[5]. To deal with this problem it was decided to carry out trimming of all audio (as shown in figure 6.1), so any sound below 20 db was eliminated (for the choice of the threshold, the minimum value achievable by the human voice in an ordinary conversation was taken into consideration).

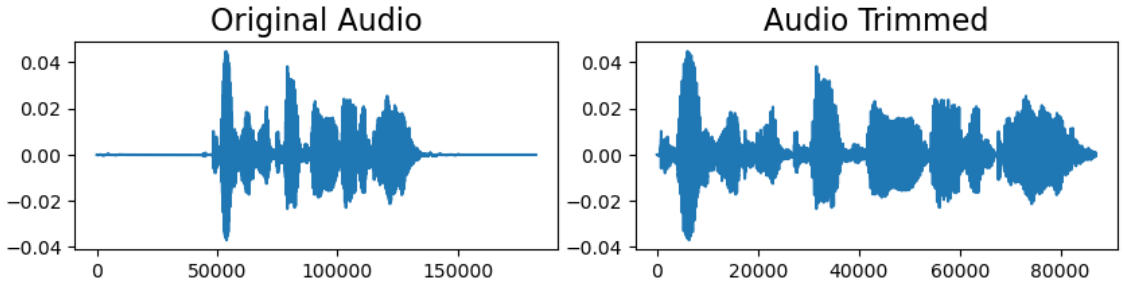

Figure 6.1: Audio before and after applying trimming operation.

It has been hypothesized that the fill silence in the original audios may negatively affect the distance between two time series. Experiments were conducted to verify how the distances and the variability of the latter change; The distances between audio with similar and non-similar

characteristics[1] were measured using dynamic time warping. Each time series was previously scaled with the minmax scaler and approximated with PAA.

Audio	μ (original)	μ (trimmed)	σ (original)	σ (trimmed)
Similar	48.1	**57.5**	31.7	**27.4**
Different	54.5	**68.8**	30.8	**24.5**

Table 6.1: mean and standard deviation of distances between similar audios (first row) and different ones (second row), before and after the trimming operation.

The results show a general increase in the distances but a decrease in the variability among the different distances measured. It was therefore concluded that this preprocessing operation affects the distance between two time series and, to visually understand how it is affected, different audios showing similar patterns were compared. Figure 6.2 shows how, during the approximation via PAA and at the same number of segments, the preprocessed audio is better able to maintain common patterns between the two time series.

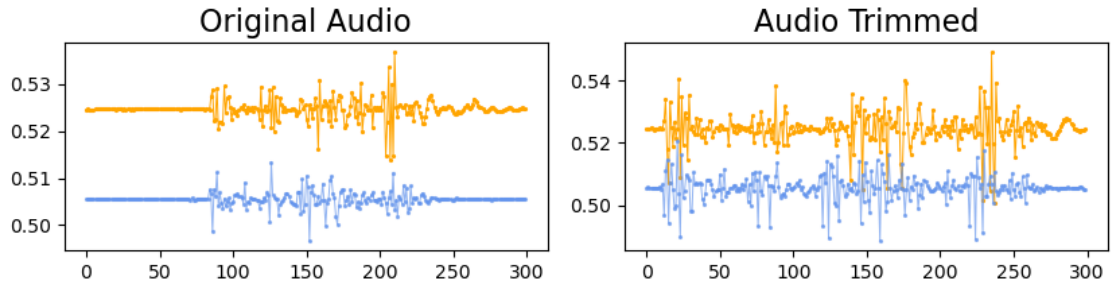

Figure 6.2: Comparison between two different audio with fearful emotion (approximated through PAA with 500 segments), before and after trimming.

This is obviously due to the fact that part of the segments of the approximation algorithm are ”wasted” in the first and last flat part of the time series. Consequently, to avoid significant loss of information, untrimmed audio needs a less strong approximation, i.e. a higher number of segments. In paragraph 6.3 different approximation methods will be analyzed and compared in addition to the PAA.

6.2 Regularization

To decide which transformations to apply, the nature of the data was taken into consideration: in the context of working with time series representing human vowels, applying linear trend removal and amplitude scaling as regularization techniques can potentially have negative effects. Linear trend removal was deemed inappropriate for our vowel data as it could remove essential pitch and intonation information. Similarly, amplitude scaling had the potential to distort the natural characteristics of the vocalizations by flattening important variations in emphasis or emotional expression.

[1]Two types of measurements were carried out: the first takes into consideration all audio with the same emotion (fearful), same gender, same vocal channel and same statement; the second one, instead, compares vowels with different characteristics (different sex, vocal channel and statement) on two type of emotion (happy and fearful).

6.3 Approximation

Due to the size of the dataset, it was necessary to approximate the time series for the subsequent operations. Based on the task performed different approximations were made, the main approximations used were **PAA** and **down sampling**. PAA was used with different number of segments depending on witch task needed to be performed (some algorithms have made it possible to use less approximate data). In the following chapters, if the approximation was performed before executing the task, it will be described what parameters it was performed with. The chosen downsampling technique involved aggregating a series of consecutive data points into a single representative point. In this case, a downsampling factor of 15 was applied, indicating that every 15 data points were consolidated into one. The factor of 15 was chosen because it is the maximum compression for which the time series remains a playable audio and lead to a good compression without too much data loss.

Chapter 7

Motifs/Discords and Shapelets

The matrix profile was used to analyze the motif and discord of the time series. Using the raw time series was unfeasible due to the large amount of memory and computational time required. Instead, an approximate version of the dataset was used, in this case it was used a downsampled version of the audios using a factor of 15.

7.1 Matrix profile

For the matrix profile, it is crucial to choose the right size of the window that will be used. In this case it was chosen the average length of a word, i.e, , 15000 timestamps or approximately 0.5 seconds. This number then was scaled in order to make up for the down sampling, so the final size of the window chosen was 1000. In figure 7.1 there is an example of matrix profile with the relative top motif found highlighted.

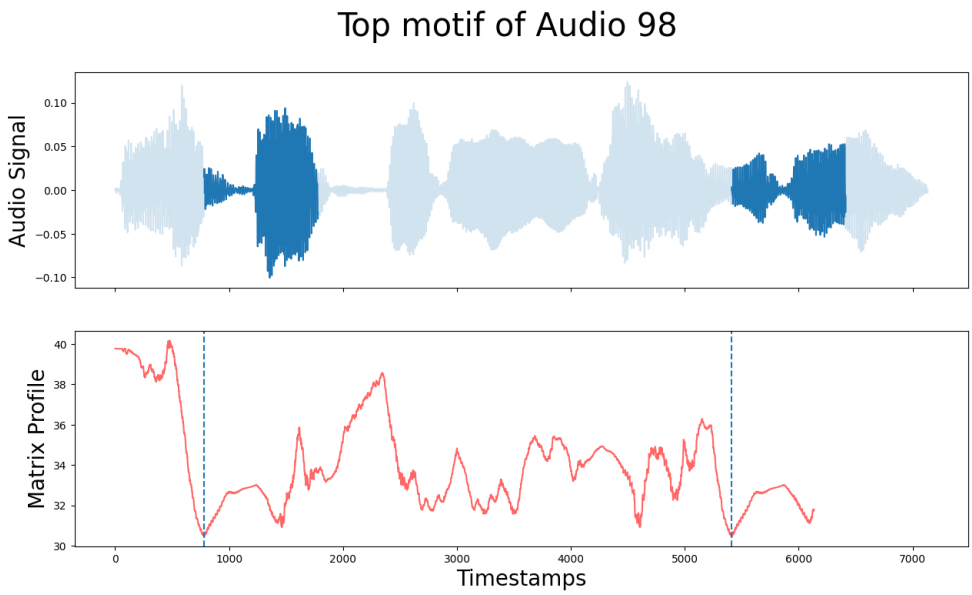

Figure 7.1: Matrix profile of audio 98 with the top motif found highlighted

We can see that the motif extracted are in between two word, this because in our audios we do not have repeated word that can be discriminate as motif, instead the matrix profile mark as motif similar sound inside the time series.

7.1.1 Top motif

After calculating the matrix profile on the whole time series dataset, the top motifs were extracted. The final step was to calculate the euclidean distance between all the motifs found and keep only the ones which distance are lower than 0.01; in this way we keep only the most common motif among all the dataset. In figure 7.2 it is plotted the most common motif found (it is possible to see that 5 audios in the dataset share the same shape). In the legend is plotted the statement said in the audio.

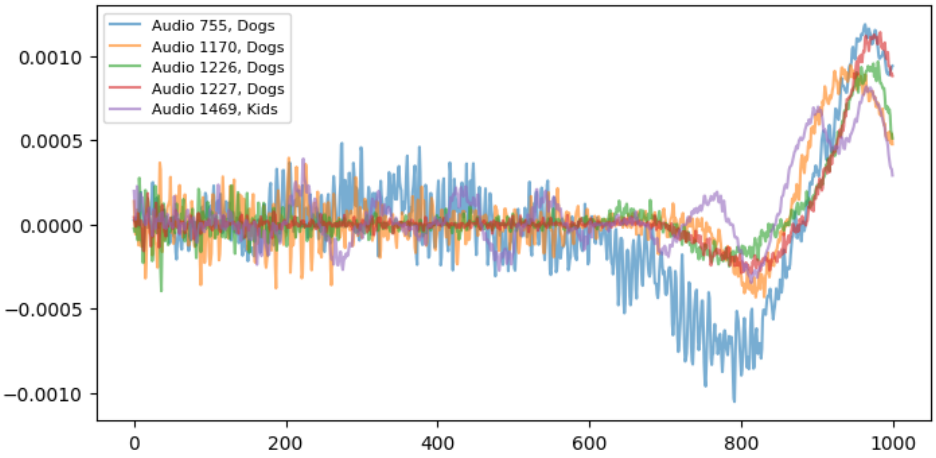

Figure 7.2: Most similar motif found among all timeseries

7.1.2 Top Anomalies

The same operation used for motif extraction were used in anomalies. In the image 7.3 we can see a plot of the most common discords among the timeseries

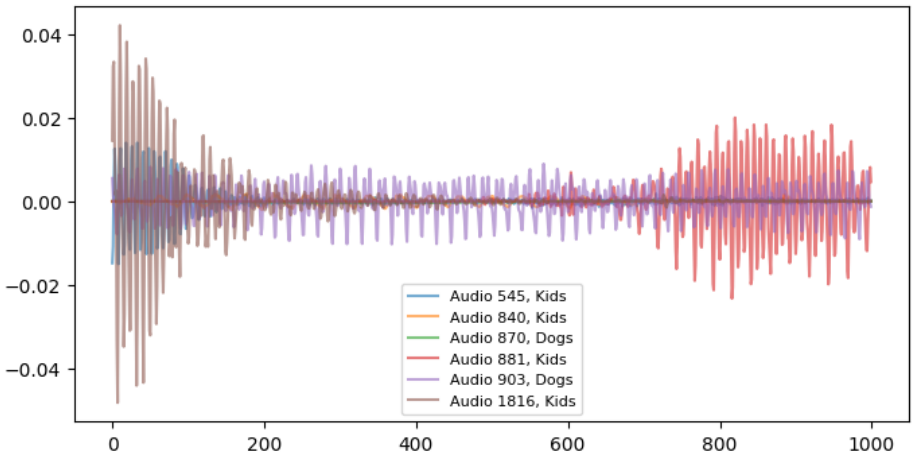

Figure 7.3: Most similar anomalies found among all timeseries

We can see form the image above that silence in the audio is often marked as anomalies, a possible explanation is given by the fact that, during preprocessing operations, the audios were trimmed, removing all the silent part at the beginning and at the end of the audios,leaving only the pronunciation of the sentences, hence silent part between word are not common and consequently marked as anomalies.

7.1.3 Shapelets comparison

For the shapelets analysis a different approximation was used, being a computationally expensive operation, the time series were approximated with a PAA to 1000 timestamps, in addition a random sampling was performed keeping 10 audios for each emotion (the chosen class used for the extraction of the most representative shapelets), thus obtaining 80 total records from which the shapelets will be extracted. The window size used for shaplets and motif was 50.

In the image 7.4 we can see the 8 most representative shapelets of class emotion and the relative time series from which we retrieved them.

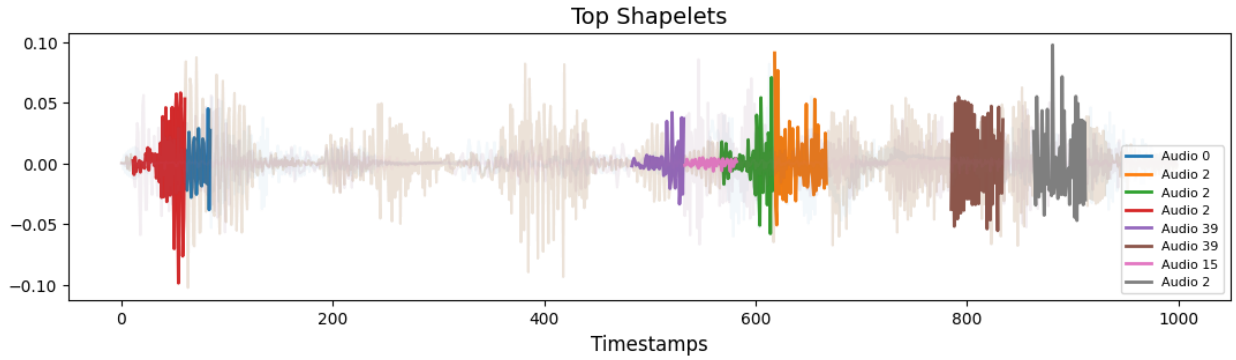

Figure 7.4: Top 8 discriminative shapelets of emotion class

The image 7.5 plot together the shapelets found and the top motif of the same time series from which the shapelets were obtained. The title of each plot represent the distance calculated with the DTW of the shapelet and the motif

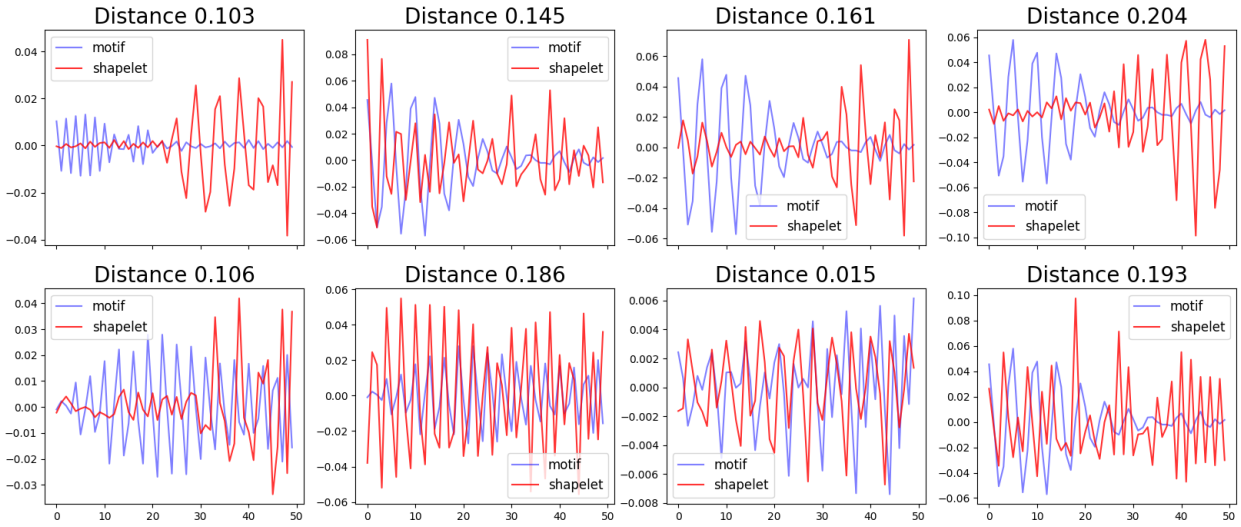

Figure 7.5: Shapelets and motif comparison

As can be seen from the images above, between shapelets and motifs there seems to be no strong correlation. One possible explanation is that motifs represent repeated patterns within the same time series, whereas shapelets represent patterns that are maximally representative of a class, testing multiple candidate shapelets at once.

7.2 Conclusion

Due to the nature of the data, motif and discords found were not informative, the analyzed audio files do not have repeated words within the sentence, so the motifs found represent sounds that are similar within the same audio, but do not have much meaning, moreover a fixed time windows fails to discriminate words of different length.

Chapter 8

Time Series: Clustering

In this chapter two different clustering approaches, hierarchical and partitional, will be discussed, focusing on the limits and justifying the results obtained..The time series were approximated by PAA using 2500 segments and all the previously described transformations were applied.

8.1 Hierarchical Clustering

To perform hierarchical clustering on time series data, it was used an **agglomerative approach** and the **MPDist algorithm**[3][2] to compute the distances between pairs of time series.

The latter considers two time series to be similar if they share many similar subsequences, regardless of the order of matching subsequences (extracted, in this experiment, through a sliding window with size 20). It can be computed more efficiently than DTW and, using the matrix profiles, this distance metric takes into account the shape and pattern of the time series rather than just comparing individual data points.
Different approaches have been tested by varying the parameters (different window size, linkage algorithm and criterion used to determine how clusters are formed). Hierarchical is unable to form full clusters that discriminate records based on emotion, but the ability to find small groups of clusters of the same emotion has still been observed.

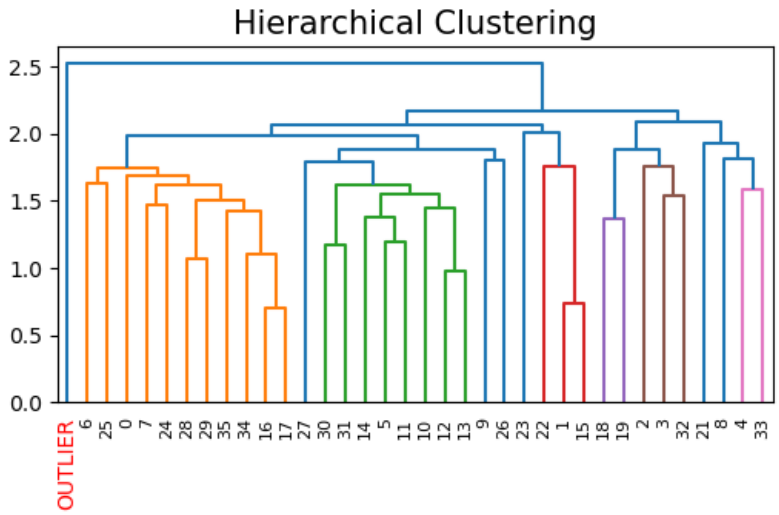

Figure 8.1: Hierarchical clustering on a small subset of data

For example, in figure 8.1 it is possible to see in yellow a small group of time series all belonging to the emotion disgust.
Despite the limited capabilities that were found, it was decided to exploit hierarchical clustering combined with MPDist to identify time series that significantly deviate from their ”similar” (i.e. presenting the same emotion, gender and statement). In figure 8.1 it is in fact possible to observe a time series (labeled as an outlier) particularly distant from the others. In the following paragraph the presence of these vowels which differ significantly in shape will be discussed using a ”shape-plot”.

8.1.1 Cluster ”outliers” and Shape-plot

We took advantage of hierarchical clustering to identify possible ”outliers”, i.e. those audio tracks which turned out to be particularly distant (in terms of shape as the MPDist algorithm was used) from audio tracks with similar characteristics (same sex, emotion, statement or vocal channels). First of all we looked for a way to reconstruct the most common shapes and similar patterns among a subset of data. It was plotted on a single graph all the time series with the same statement, emotion and vocal channel, setting the opacity at 0.1. This makes easy to observe directly the most common emotion shapes among the audio. In figure 8.2, for example, it is possible to appreciate the shape common to all vowels with a calm emotion. It has also been

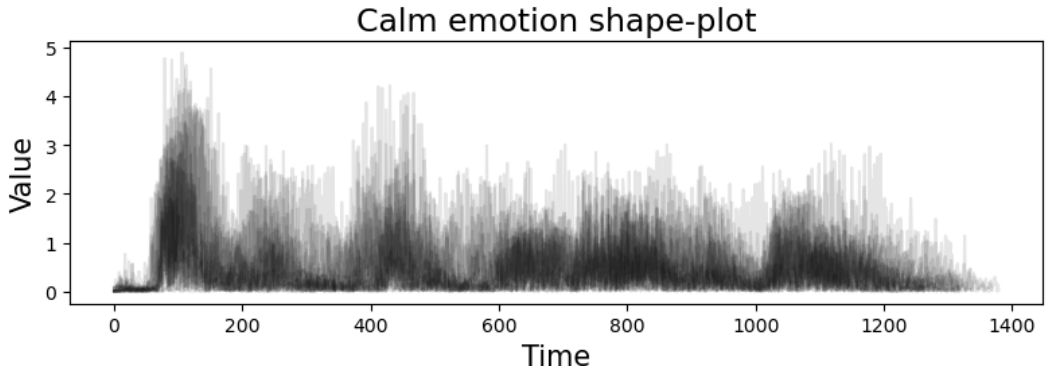

Figure 8.2: shape-plot of calm emotion

noted that some emotions vary significantly from each other and it is therefore more difficult to find a common shape.

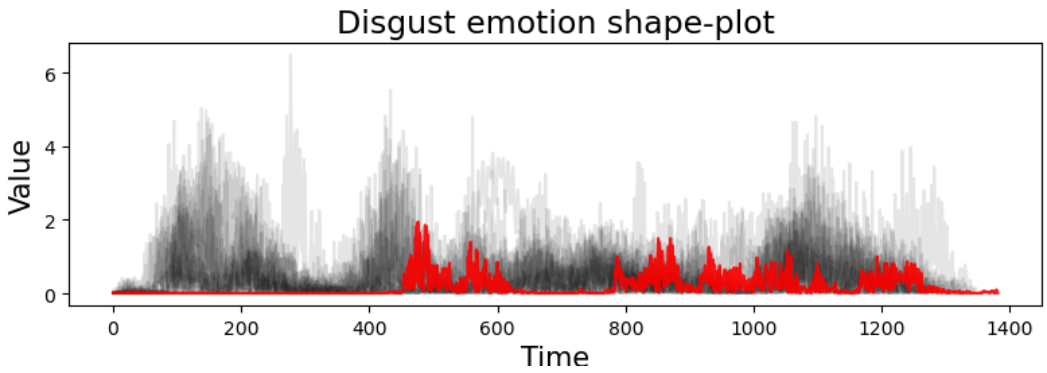

Figure 8.3: shape-plot of disgust emotion. The outlier identified by dendrogram is highlighted in red.

By combining the results of hierarchical clustering with MPDist and the shape-plot, it is possible to give a visual justification of how far a record is from what its cluster should be. Taking the audio with the emotion of disgust labeled as an outlier in figure 8.1, and inserting it into the shape plot of the corresponding emotion, it is possible to observe a very different shape compared to the common one (figure 8.3).

8.2 KMeans

In this section, the performance of k-means clustering it was investigated, using two different distance measures: Dynamic Time Warping (DTW) and Euclidean distance. An attempt was made to cluster the data into 8 different clusters, observing a possible grouping based on emotions. Due to the complexity of the dtw, it was decided to use a subset of the data, as a further approximation to reduce the length of the time series leads to significant loss of information. The subset taken into consideration, although very small (144 records), is representative of the entire dataset; contains 18 records for each emotion, all having the same characteristics (same statement, male gender, spoken vocal channel and normal emotional intensity).

8.2.1 KMeans with euclidean distance

The Euclidean distance, as expected, returns poor results, tending to cluster the data into a few sets. Figure 8.4 shows the centroids in red of the 8 clusters and in black the time series belonging to the respective cluster. Euclidean distance calculates the straight-line distance between corresponding points in the time series, thus failing to compare similar shapes that evolve at different times.

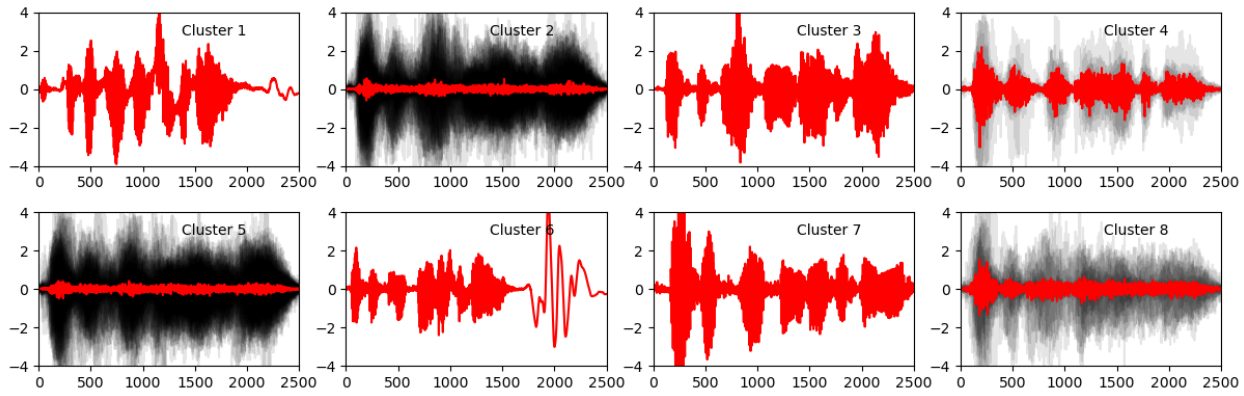

Figure 8.4: centroids of kmeans using euclidean distance

For this reason it is possible to observe two clusters with very flat barycenters as they contain most of the data, many of which have very different shapes. To overcome this limitation and improve clustering performance, the dtw was used.

8.2.2 KMeans with dynamic time warping

DTW considers the holistic shape and structure of the time series, specifically designed to handle temporal distortions and align time series sequences. As a result more heterogeneous clusters are obtained and ”flattened” barycentres are no longer present (figure 8.5). However, looking at the data labeling within the different clusters, an uneven distribution of the data was noticed, as there are clusters that collect the most data.

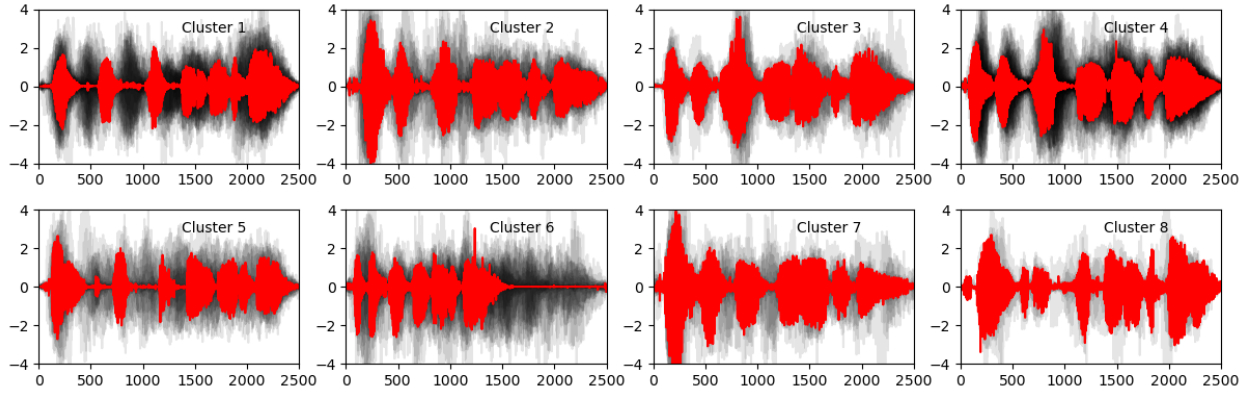

Figure 8.5: centroids of kmeans using dtw

As summarized in the table, dtw manages to obtain quantitatively better results but at the expense of a much higher computational cost (considering the same subset and number of clusters).

Distance	time magnitude	silouhette score
DTW	minutes	0.024
Euclidean	milliseconds	0.001

Table 8.1: kmeans summary using dtw and euclidean distance

8.2.3 Centroids and motif discovery

Analyzing the clusters resulting from the kmeans using the dtw, we focused on the fourth cluster (figure 8.5) as it contains almost exclusively audio with *calm* emotion and few *neutral* emotions. In particular, possible similarities were sought between the centroid of the cluster and the belonging time series.

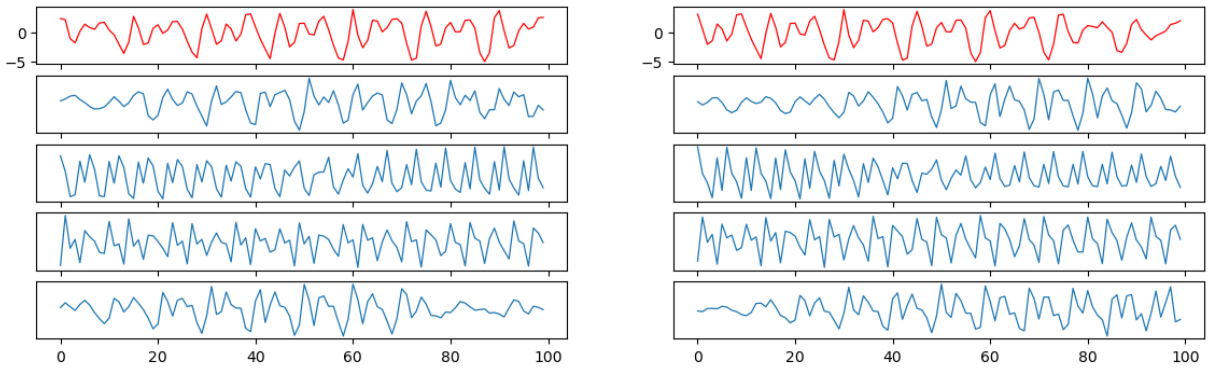

Figure 8.6: cluster 4 motifs; centroid in red, and time series belonging to the cluster in blue

8.3 Transactional Clustering: TXMeans and CLOPE

Experiments were performed using two clustering algorithms specifically designed for transactional datasets: TXMeans and CLOPE. **TXMeans** is a hierarchical divisive technique for clustering; it operates by repeatedly dividing clusters (using a *local* measure) and selecting a representative transaction for each one. It is able to achieve efficiency without compromising the quality of clustering, and primarily, without the need for parameter adjustment. On the other hand, **CLOPE**[1] is based on the idea of maximizing a *global* cost function and requires as a parameter the exact number of clusters. The experiments were conducted on a dataset created by selecting the basic statistics and the mfcc, and discretizing each feature in 16 equal-width intervals. A repulsion coefficient of 1.2 was used for the CLOPE.

algorithm	time magnitude	purity	NMI	n° cluster
TXMeans	milliseconds	0.24	0.082	6
CLOPE	seconds	0.16	0.012	8

Table 8.2: transactional clustering summary

[1]The algorithm has been implemented in python and tested on transactional datasets. The code is available in the following repository: https://github.com/LinoVelardita/CLOPE_Python_Implementation

To evaluate the quality of the clusters, two metrics were taken into consideration: **purity** and **normalized mutual information** (both external measures because they need the original class labels). Purity calculates the proportion of correctly classified instances within each cluster, i.e. the homogenity of clusters, while the NMI allows to compare results between different clusterings having different number of clusters (useful since it is not possible to control the number of clusters that will form the TXMeans).

The table 8.2 summarizes the results of the two approach. Although a purity equal to 24% was obtained with the TXMeans, an NMI equal to 0.082 implies a relatively weak overall agreement between the true labels and the assigned cluster labels. It has been noted that TXMeans manages to create slightly better clusters and with a much lower computational cost.

8.3.1 UMAP projection

Using UMAP projection, it is possible to see (visually) how well the clusters formed by TXMeans and CLOPE align with the underlying data structure in this small dimensional space.

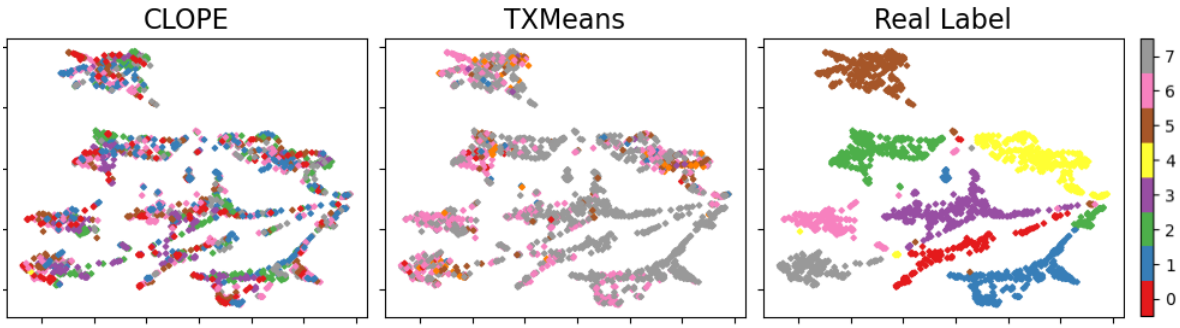

Figure 8.7: Cluster labels of CLOPE and TXMeans, plotted thorugh UMAP reduction

An attempt was made to visually separate the 8 emotion classes using UMAP, to then represent the labels obtained using the two clustering algorithms on the new reduced space (figure 8.7). Both algorithms are not able to clearly separate the different classes: TXMeans presents quantitatively very heterogeneous clusters as most of the points belong to two clusters; on the other hand, the CLOPE, despite having a more homogeneous distribution of the points, is also not able to provide a clustering more or less aligned with the true labeling of the points.

Chapter 9

Time Series: Classification

This chapter will discuss the results obtained from the classification operations. the data used were approximated according to the type of classification performed, the following paragraphs

will describe what type of approximation was used for each classifier used. The feature chosen for the classification was *emotion*, containing 8 class.

9.1 KNN

The KNN was tested using the time series without any approximation, the parameters used for the models were the same found during the previous application of KNN in the datasets: *n_neighbors*=12, *weights*='distance', *metric*='cityblock', this were the best parameter found for the emotion classification during the data mining 1 module. The result of the classifier is reported in table 9.1. The accuracy of the classifier is **14%**, a little better than a trivial one (accuracy: 12.5%), but it can discrimiate only 2 of 8 classes.

	Neutral	Calm	happy	sad	angry	fearful	disgust	surprised
Precision	0.00	0.08	0.00	0.16	0.00	0.00	0.00	0.00
Recall	0.00	0.11	0.00	0.77	0.00	0.00	0.00	0.00
F1	0.00	0.09	0.00	0.26	0.00	0.00	0.00	0.00

Table 9.1: Precision, Recall and F1 score of the KNN classifier with euclidean distance

Better results were obtained with a Dynamic time warping metric. The one used in the classifier was a sakoechiba band as constrain with window_size = 0.1. The result of the classifier is reported in table 9.2. The accuracy in this case was of **35%**, we can see that DTW greatly improve the classifier, in this case the classifier can discriminate all 8 classes.

	Neutral	Calm	happy	sad	angry	fearful	disgust	surprised
Precision	0.28	0.37	0.37	0.26	0.59	0.30	0.15	0.21
Recall	0.52	0.71	0.29	0.24	0.50	0.12	0.10	0.15
F1	0.36	0.49	0.33	0.25	0.54	0.18	0.15	0.17

Table 9.2: Precision, Recall and F1 score of the KNN classifier with DTW

9.2 Shapelets

For the shapelets base classification different method were used. At first, a brute force approach was tried, the procedure followed for the shapelet extraction, is the same as that described in the chapter 7.1.3. with the only difference of using 25 time series for each emotion in the training set and 10 for the test set, so, in total, 200 time series used for training and 80 for testing.

The classifier scored a **19%** of accuracy and **23%** of precision, **19%** of recall and **20%** of F1. For the purpose of simplification, only the macro average score of precision recall and F1 score are reported. In image 9.1 the top 3 most discriminative shapelets found with this metod are reported.

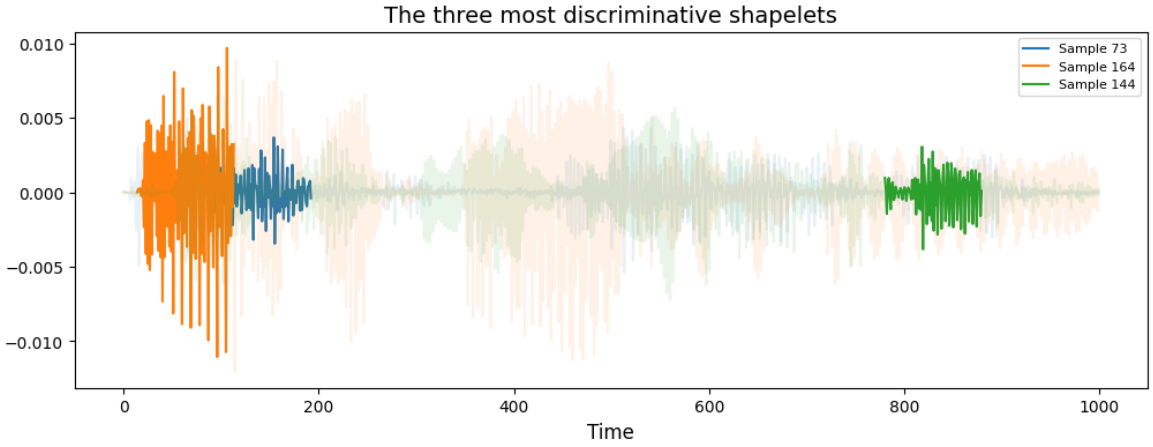

Figure 9.1: Top 3 most discriminative shapelets obtained with brute force approach

The best results were obtained with a distance based shapelet classifier. The first step was to fit a learinig shapelets model[1], than transform the train and test set in order to have, for all time series, the distances between the shapelets found and the time series itself. In our case, the method found 7 shapelets, so the transformed version of the dataset contain a row for each time series and 7 value representing the distance. Lastly, a decision tree was used for the classification of the transformed dataset. In figure 9.2 are represented the three most discriminative shapelts found by the learning base method.

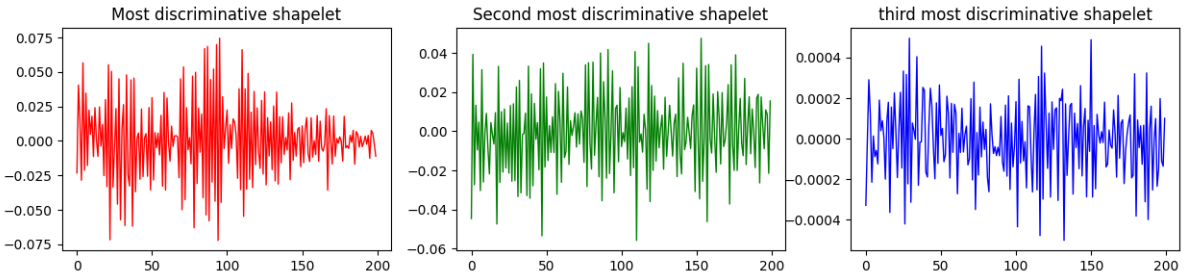

Figure 9.2: Top 3 most discriminative shapelets obtained with learning base approach

The classifier scored a **27%** of accuracy, **27%** of precision, **26%** of recall and **26%** of F1. For the purpose of simplification, only the macro average score of precision recall and F1 are reported.

9.3 Mini Rocket

Classification with MiniRocket transformer lead to best results so far in term of accuracy and computational time. The transform was executed with the standard parameter of kernels, 10000. After the transformation, the classification was done by a Ridge Classifier. The parameter used for the Ridge classifier are the default used after the rocket transform (alphas = np.logspace(-3, 3, 10)). This method also made it possible to use the data with less approximation, in fact, the down sampled dataset was used, which contains $\approx$ 16000 time stamps for time series.

[1]The method used for the shapelet learning task is the one described in the paper J.Grabocka et al. Learning Time-Series Shapelets. SIGKDD 2014, implemented in the library tslearn

The classifier scored a **57%** of accuracy, **59%** of average precision, **57%** of average recall and **57%** of average F1. In table9.3 there are the precision, recall and F1 score for each class.

	Neutral	Calm	happy	sad	angry	fearful	disgust	surprised
Precision	0.68	0.78	0.56	0.50	0.58	0.49	0.47	0.76
Recall	0.58	0.54	0.61	0.57	0.73	0.36	0.73	0.46
F1	0.59	0.64	0.59	0.53	0.65	0.42	0.57	0.57

Table 9.3: Precision, Recall and F1 score of the Ridge classifier with MiniRocket transform

Chapter 10

Explainability

In this section it will be discussed the methods used for the explainability of a classifier. The feature dataset of RAVDESS was used, and *sex* was chosen as the target variable. A neural network[1] will be analyzed through a local and global explainer and a random forest[2] through a global explainer.

10.1 Local Explanation

For the local explanation it was used the SHAP explainer. It works by assigns an importance value to each feature in the input data, indicating its impact on the model's prediction. It calculates these importance values by considering all possible combinations of features and evaluating their impact on the model's output. In the image 10.1 it's plotted the shap values of a single record with a waterfall plot. The number in each bar represent the shap values, the number beside the feature name in the y-axis is the value of that feature of the record. The absolute SHAP value indicates how much a feature has afflicted the prediction (in this case the prediction is female) and, in this case, *q50_w2* contributed the most.

[1]The parameters used are: solver=sgd, validation fraction=0.35, activation=relu, and the default layers structure
[2]The parameters used are: n°estimators=180, criterion=gini, max depth=12

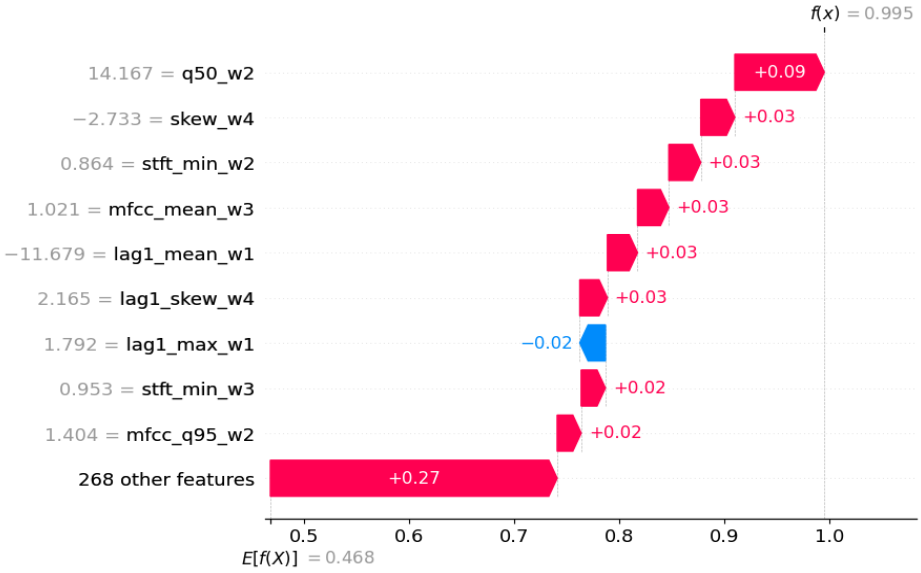

Figure 10.1: Shap values of record 9 of test set

In the image 10.2 it is plotted the the average of the shap values over the whole test set, the most important feature in the prediction of a class is *lag1_kur_w1*

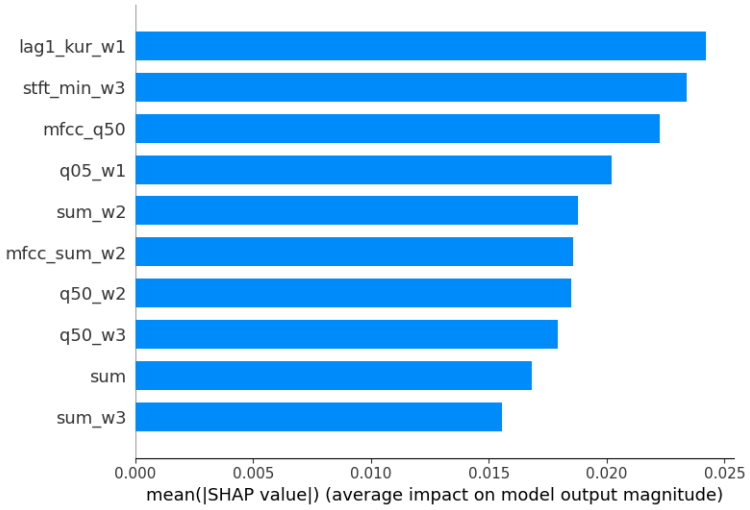

Figure 10.2: Average feature importance of SHAP on test set

10.2 Global Explanation

A permutation importance method was used to extract the most important features for the analyzed models (neural network and random forest)[3]. The algorithm in question exchanges values of the single features and measures the effects on the performance of model (cross-entropy was used as a metric for this experiment).

[3]The implementation provided by the skater library was used: https://github.com/oracle/skater

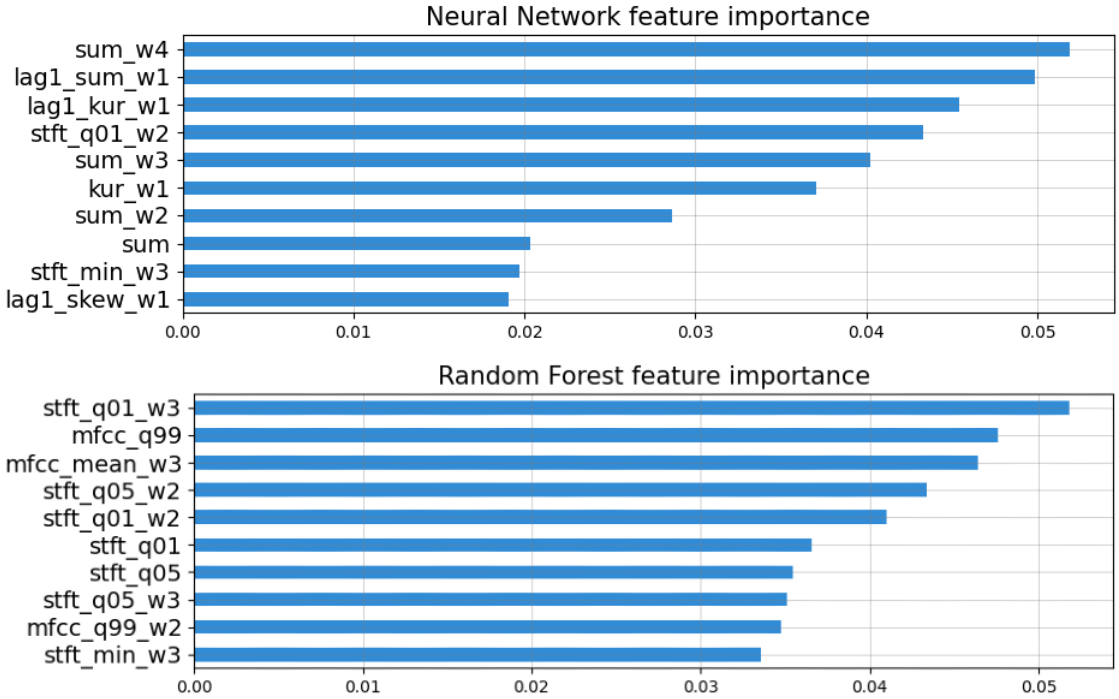

Figure 10.3: Feature importance for NN and RandomForest

The results are compatible with the results obtained through the SHAP; they both acknowledge some features like *sum* and lag1 stats as more important. Furthermore, random forest and the neural network recognize both as important features related to the chromagram (stft). Concluding, it can be stated that some statistics extracted from the audios (such as the spectral centroid and the zero crossing rate) do not contribute significantly to the classification of *sex* variable.

Bibliography

[1] Borderline Over-sampling for Imbalanced Data Classification. "Acoustic Characteristics of English Fricatives." In: (2009). URL: https://ousar.lib.okayama-u.ac.jp/en/19617.

[2] Shaghayegh Gharghabi et al. "MPdist library". In: (2018). URL: https://matrixprofile.docs.matrixprofile.org/api.html#matrixprofile-algorithms-mpdist.

[3] Shaghayegh Gharghabi et al. "MPdist: A Novel Time Series Distance Measure to Allow Data Mining in More Challenging Scenarios". In: *2018 IEEE International Conference on Data Mining (ICDM)* (2018). DOI: 10.1109/ICDM.2018.00119. URL: https://ieeexplore.ieee.org/abstract/document/8594928.

[4] Sergios Theodoridis. "SVM". In: (2022). URL: https://www.svms.org/parameters/.

[5] Simon Fong. "Audio-classification". In: *Kaggle website* (2023). URL: https://www.kaggle.com/code/raksh710/audio-classification.