IN SUPREMÆ DIGNITATIS
1343
UNIVERSITÀ DI PISA

Department of Computer Science
Master in Data Science and Business Informatics

DM2 – Data Mining
Advanced Topics and Applications

Submitted to:

Prof. Riccardo Guidotti
Francesco Spinnato, TA

Submitted by:

Nimra Nawaz
Hafiz Muhammad Umer

Academic Year 2022/23

CONTENT

Chapter 1

1. Data Understanding & Preparation

1.1. Data Understanding

The dataset was created from the RAVDESS dataset, extracting basic statistics from the original audio data and after transforming it using: differencing, zero-crossing rate, Mel-Frequency Cepstral Coefficients, spectral centroid, and the stft chromagram. Features were extracted from the 2452 wav files. Features are extracted also by dividing each time series into 4 non overlapping windows.

Dataset has already been divided into training and test set. The training set contains 1828 records while the test set contains 624 records. The dataset is composed of 434 features from which 405 belongs to floating point continues variables, 21 of them are integers, while the other 8 of them are categorical type variables and there is no missing value in any feature.

To determine the distribution of the data, a subset of variables selected to be examined because analyzing all 434 variables would be excessively time-consuming. Thus, a random sample of 20 variables was selected and analyzed by plotting their values using a histogram. The best fitting distribution function was identified by calculating the sum of squared error against the histogram and choosing the function with the lowest value. The analysis revealed three main distribution types: Log-Normal, Gamma, and Normal. Figure 1 displays the most representative outcomes of the analysis.

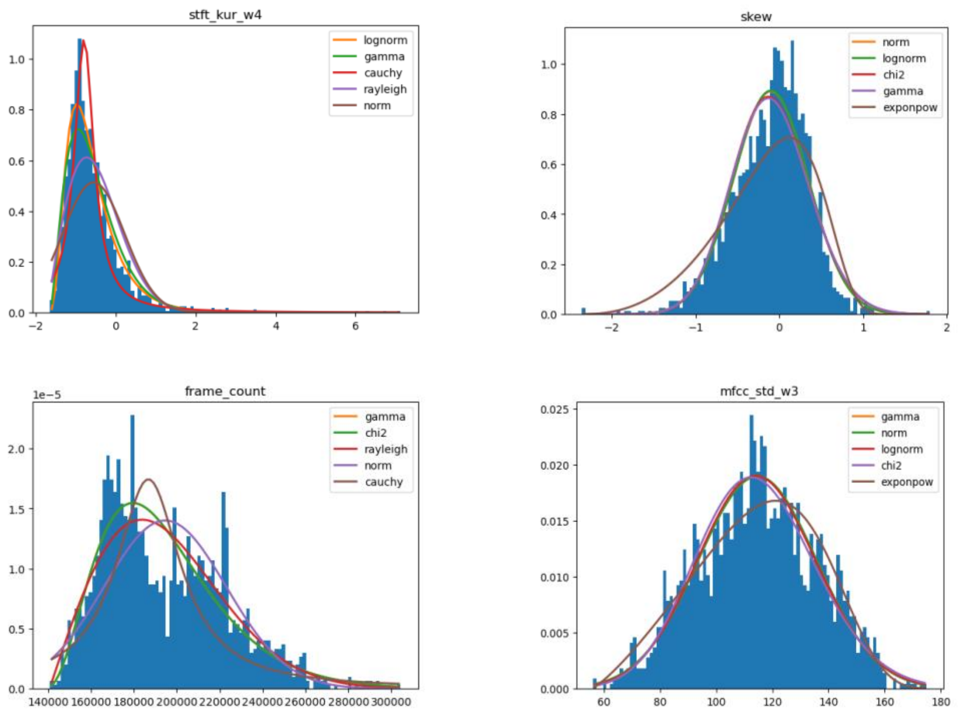


Figure 1.1 Most common distributions among the X variables

1.2. Data Preparation

To understand our data better, we made a table using the feature names and combined the data we use to train and test. This helped us see all the information together. We found that there is no null value in the whole dataset. Next, we considered vocal channel as our target class for the basic classification task so we separated it from our dataset.

1.3. Basic Classification

Initially, we decided to evaluate the classification report's findings using two distinct classification techniques, namely Decision Tree and KNN. To establish a benchmark for comparison, we executed the algorithms without enhancing the dataset in any manner.

1.3.1. Decision Tree

In this project, the Decision Tree classifier is used to predict the target variable by acquiring simple decision rules from the data attributes. The classifier's default hyperparameters are used since the aim is to establish a starting score for evaluating other algorithms.

From the classification report, class 2, speech, appeared to be more perfectly classified with a value of 0.94, 0.91, and 0.93 for precision, recall, and f1-score respectively. The other class, song, is also appeared to be very close to the class 2 but a little less accurate than the other class. Figure 1.2 represents the AUC-ROC and Precision-Recall Curve for the decision tree classifier and we can see both classes have the similar results.

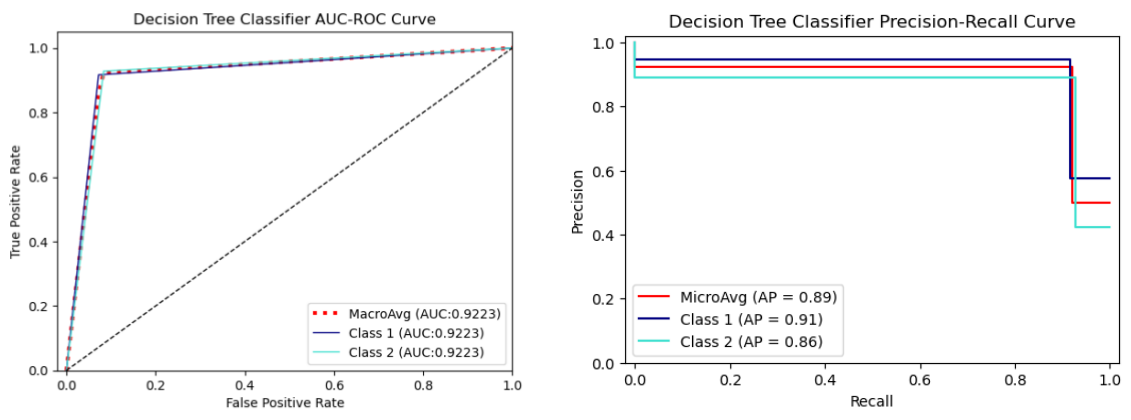


Figure 1.2 AUC-ROC and Precision-Recall Curve for Decision Tree Classifier

1.3.2. K-Nearest Neighbors

KNN has been applied on the same dataset without any hyperparameters tunning and distance used for the training was default such as Minkowski. The classification report for KNN represents a drop in each metrics such as precision, recall, and f1-score for both classes and the overall accuracy has been dropped to 84%.

Figure 1.3 represent AUC-ROC and Precision-Recall Curve for K-Nearest Neighbors Classifier, where we can clearly see a drop in both of the classes. The drop is neglectable as it is very less as

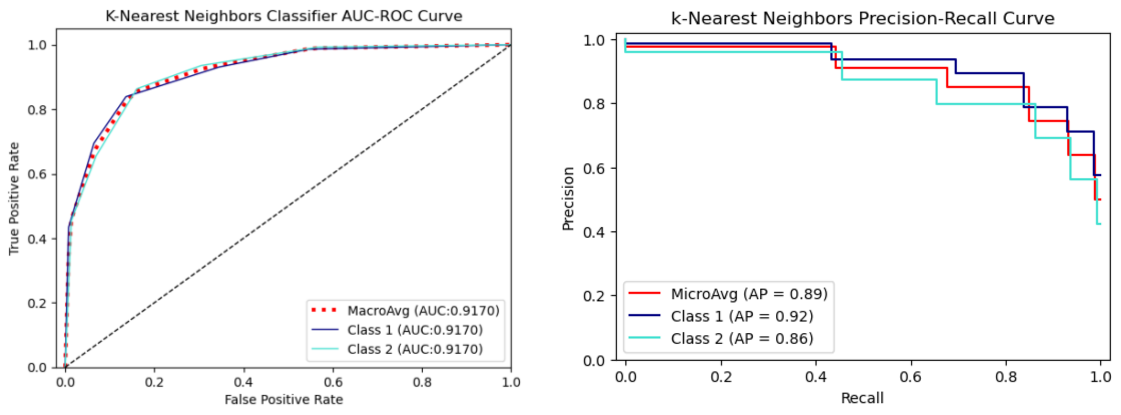


Figure 1.3 AUC-ROC and Precision-Recall Curve for K-Nearest Neighbors Classifier

compare to decision tree classifier but still, we can see the difference between the two classifiers.

Whereas, Table 1.1 represents the different metrices such as accuracy, precision, recall, f1-score, and AUC-ROC for each classification task e.g. decision tree classifier, and k-nearest neighbors classification.

Table 1.1 Basic Classification Tasks Report

	Accuracy	Precision	Recall	F1-score	AUC-ROC
Decision Tree	0.9215	0.9182	0.9223	0.9200	0.9223
K-Nearest Neighbors	0.8494	0.8453	0.8513	0.8472	0.9170

In the next part of the project, we have identified top 1% outliers exists in the dataset using different techniques such as density-based, angle-based anomaly detection.

2. Outliers Detection

Outliers' detection means finding things that are unusual because they are very different from what usually happens. It is important to do this because it can help us analyze data better. Our goal was to find the top 1% of outliers in the data. We used different methods to do this, and in the next part, we will talk about what we found.

2.1. Histogram-based Outlier Score

Next, we used the Histogram-based Outlier Score (HBOS) outlier detection algorithm with default parameters to identify outliers in our dataset. According to the output of the algorithm, there were 183 outliers in our dataset. We next plotted (Figure 2.1) histograms with 30 bins to better understand the distribution of the data. To better understand where the outliers are located in the data, we drew a vertical line at the minimal outlier score value. This information can help us understand any problems with the data and in understanding which characteristics of the data are causing the outliers.

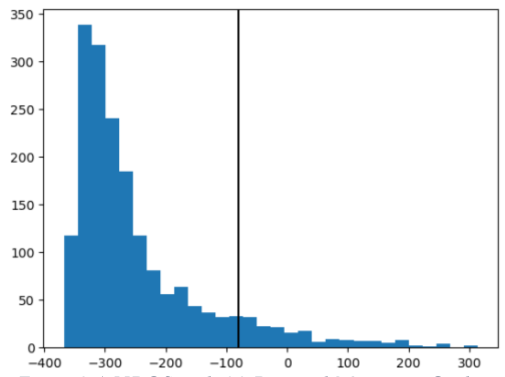


Figure 2.1 HBOS with 30 Bins and Minimum Outlier Score Threshold

2.2. Isolation Forest

Next, we used the Isolation Forest algorithm to detect outliers in the dataset. We set the contamination value to 0.05, which means we expect to have 5% of the data points as outliers. We also set the max_samples parameter to 600, which is the maximum number of samples used to build each tree in the forest. After applying the algorithm, we found 92 outliers in the dataset. This means that these data points are significantly different from the majority of the data points.

2.3. Local Outlier Factor

We used the LOF algorithm to find outliers in our dataset. It is similar to DBSCAN and uses "core distance" and "reachability distance" to calculate the density of data points. It calculates local density by looking at the distance of k nearest neighbors. The local outlier factor is then based on this density. We found 19 outliers using the contamination value of 0.01 and n_neighbors equal to 7. To visualize the output, we plotted a histogram (Figure 2.2) with 100 bins, where the x-axis represents the negative outlier factor (NOF) and the y-axis represents the frequency of data points. We added a vertical line at the maximum value of NOF for the detected outliers.

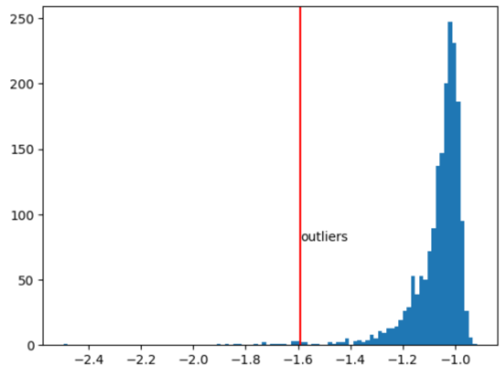


Figure 2.2 Negative Outlier Factor Distribution Plot for LOF Algorithm

2.4. Angle-based Outlier Detection

In this experiment, we applied ABOD algorithm to the dataset to detect the outliers. ABOD stands for Angle-Based Outlier Detection, which uses the concept of angles between data points to identify the outliers. The algorithm detected 223 outliers in the dataset with the default parameters of ABOD algorithm, which are contamination value and n_neighbors 0.01 and 5, respectively.

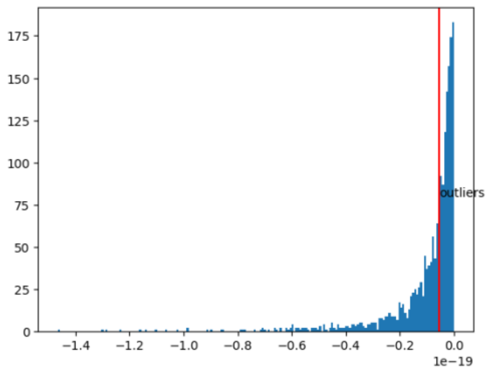


Figure 2.3 Histogram of ABOD Decision Scores for Anomaly Detection

Figure 2.3 visualize the output of ABOD, we plotted a histogram of the decision scores of each data point with 200 bins. The axvline in the histogram represents the minimum value of the outliers.

2.5. Dealing With Outliers

In this analysis, ABOD seems to have a worst performance on our dataset finding too many outliers with respect to other methods, which could be due to the default hyperparameters used in the implementation. Additionally, the decision scores plot shows that the scores are not well-distributed, with a majority of the points having low scores and a few having extremely high scores. On the other hand, Isolation Forest performed better than the other algorithms in this specific dataset for outlier detection. Next, we tried to deal with the outliers in two different ways removing and transforming and then checking the existing of outliers after the result of the two methods. Furthermore, we also applied the same algorithms on the cleaned and transformed dataset as we applied in the first chapter to establish a benchmark for comparison.

2.5.1. Removing Outliers

We tried several approaches to manage outliers. The first was to eliminate the top 1% of outliers that were common among at least 3 of the 4 previously proposed methods, this made their identification more reliable. After this, the number of identified outliers is 31, representing 1.7% of the training set. At this point we created a copy of our original dataset and dropped the outliers; we then evaluated the performance of the baseline classification model with the new data to see if there is any improvement. These results can be seen in table 2.1.

2.5.2. Transforming Outliers

In this case a new dataset has been created adding new rows to the dataset, as much as the outliers previously removed and then filled with the mean of the respective columns. The improvements of the result after transforming the outliers could be seen in table 2.1 with respective to the baseline results in the previous chapter.

Table 2.1 Performance comparison of methods to deal with outliers

	Accuracy	Precision	Recall	F1-score
Decision Tree Baseline	0.9215	0.9182	0.9223	0.9200
Removing	0.9119	0.9085	0.9165	0.9108
Improvement %	-1.04%	-1.05%	-0.63%	-1%
Transforming	0.9119	0.9167	0.9205	0.9183
Improvement %	-1.04%	-0.16%	-0.19%	-0.18%
KNN Baseline	0.8494	0.8453	0.8513	0.8472
Removing	0.8510	0.8469	0.8527	0.8488
Improvement %	0.18%	0.18%	0.16%	0.18%
Transforming	0.8510	0.8469	0.8527	0.8488
Improvement %	0.18%	0.18%	0.16%	0.18%

Although the baseline classification performance didn't improve after removing or transforming the outliers. In any case, we expect to improve upon this process in a further section.

3. Imbalanced Learning

Figure 3.1, the original training set consists of 1080 records with Vocal Channel class = speech and 748 records with Vocal Channel class = song, respectively about 59.1% and 40.9%. In order to get a heavily biased dataset with 96% of records belonging to class speech and 4% of records belonging to class song, by means of proportion it turned out that the number of records belonging to class Vocal Channel = 1 should be equal to 43 (705 less), figure 3.2. The dataset thus composed was obtained by randomly selecting the records to delete. To make the random selection stay always the same np.random.seed(18) was placed. Subsequently, KNN classifier was used to train the highly unbalanced classes. So far, after applying standard scalar on the dataset we got quite impressive result with test set. The overall accuracy we got is 82%.

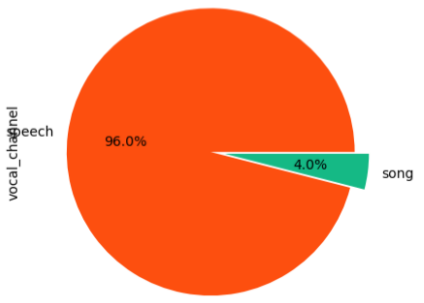


Figure 3.2 Vocal Channel Pie Chart After

Figure 3.2.1 Vocal Channel Pie Chart Before

3.1. UnderSampling

3.1.1. Random UnderSampling

The class with the highest frequency i.e. speech has been reduced to a number of 748 (the same number as the lowest class) randomly extracting records without repeats by setting random state at 42. Using this technique for the K-NN algorithm, the result accuracy of the model increases significantly from 0.82 to 0.93.

3.1.2. Condensed Nearest Neighbor

The CondensedNearestNeighbour algorithm to balance an imbalanced dataset consisting of two classes 'song' and 'speech' and then trained a K-NN classifier on it. The random state was set to 42, and the number of jobs was set to 10. We got 748 records belonging to the class song and 335 records belonging to the class speech which makes the dataset partially unbalance.

The results show that the accuracy of the classifier is 0.9006, which is a decrease of 3% compared to the previous results. Figure 3.3, represents a confusion matrix for this experiment.

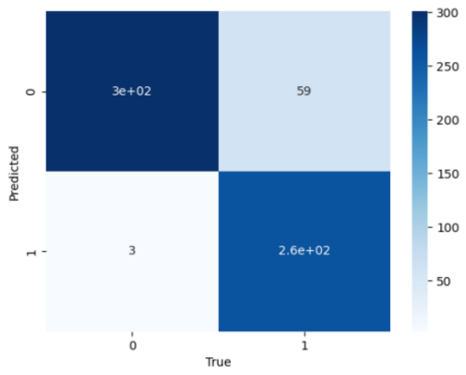


Figure 3.3 Confusion Matrix for K-NN algorithm on imbalanced dataset with CondensedNearestNeighbour

3.2. OverSampling

3.2.1. Random OverSampling

The RandomOverSampler has been applied to the imbalanced dataset, which generated synthetic data points for the minority class 'song' to make the dataset balanced. After that, we applied K-NN algorithm on the balanced dataset and evaluated the model using accuracy and

F1-score. We found that the accuracy has increased from 90.06% to 94.23%, which means the model is now able to predict more correct labels. Similarly, the F1-score has also improved from 0.8938 to 0.9345 for the minority class, which indicates that the model's ability to correctly identify the minority class has increased.

3.2.2. SMOTE

We applied SMOTE on the imbalanced dataset to balance the number of records in both classes. The results of the K-NN algorithm on the balanced dataset showed that the accuracy of the model was 0.9407 which is slightly lower than the results of the RandomOverSampler. However, the F1-score for both classes is almost the same, indicating that the model performs well for both classes.

Overall, both RandomOverSampler and SMOTE have improved the results of the K-NN algorithm compared to the original imbalanced dataset. However, RandomOverSampler performed slightly better than SMOTE in terms of accuracy.

4. Dimensionality Reduction

In this phase we will try to reduce the dimensionality of the dataset by considering all the classifiers analyzed so far, paying particular attention to those that show better performance, such as KNN, following the reduction. The methods used to reduce dimensionality will be:

- **Feature Selection:** Variance Threshold, Univariate Feature Selection using SelectKBest algorithm and Select from Model;
- **Feature Projection:** PCA, Gaussian Random Projection, and ISOmap;

4.1. Feature Selection

Feature selection tries to select a subset of the original features present in our dataset. This allows us to remove redundant and irrelevant features without having much loss of information since the most important features are kept. As before, the results of every method will be evaluated and compared at the end of the subsection in table 4.1.

4.1.1. Variance Threshold

Variance Threshold is a feature selector that removes all the features having a low variance in the dataset since they are not very informative or at least not as the ones with high variance. In this case the threshold of the model has been set to 0.20, meaning we removed the features that are at least 80% similar, therefore reducing the risk of multicollinearity, with this we obtained a selection of 187 features.

4.1.2. Univariate Feature Selection

Univariate feature selection consists of a selection of the best features in the dataset based on statistical analysis tests. In this case the method SelectkBest is used, which returns the k highest scoring features, the optimal value of k is found by iterating through a range of 200 to find the k which gave the best results in KNN figure 4.1. With this we get k to 81.

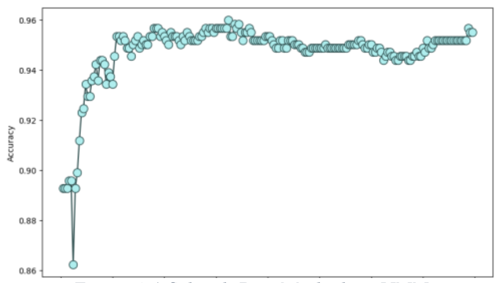


Figure 4.1 Select k Best Method on KNN

4.1.3. Select From Model

Select From Model performs a feature selection based on the importance attribute threshold given by the chosen model which, in this case, is the Logistic Regression model. Features whose importance is greater or equal to the given threshold are kept while the others are removed. The logistic regression assigns each row a probability of being True and then makes a prediction for each row with a default threshold of 0.5.

Table 4.1 Performance Comparison between methods of feature selection

	Accuracy	Precision	Recall	F1-score
KNN Baseline	0.8494	0.8453	0.8513	0.8472
Variance Threshold	0.81	0.80	0.80	0.80
Improvement %	-4.6%	-4.17%	-4.85%	-4.39%
Univariate Feature Selection	0.95	0.96	0.96	0.96
Improvement %	11.8%	13.5%	12.7%	13.3%
Select From Model	0.90	0.92	0.91	0.91
Improvement %	5.9%	8.8%	6.9%	7.4%

4.2. Feature Projection

Feature Projection, unlike feature selection, is the transformation of data from a high-dimensional space into a low-dimensional space that still maintains the meaningful properties of the original data. As before, the results of every method will be evaluated and compared at the end of the subsection in table 4.2.

4.2.1. Principal Component Analysis (PCA)

PCA is a technique that projects the features by creating new uncorrelated variables that maximize variance. To know how much variance is explained with n components we can plot the variance ratio with a Scree Plot (fig. 4.2) to visualize the Proportion of Explained Variance PEV in each n component. We optimized n_components through iterative process and we got 7 components, giving us an important reduction of features with only 2.3% loss of accuracy, more information in table 4.2.

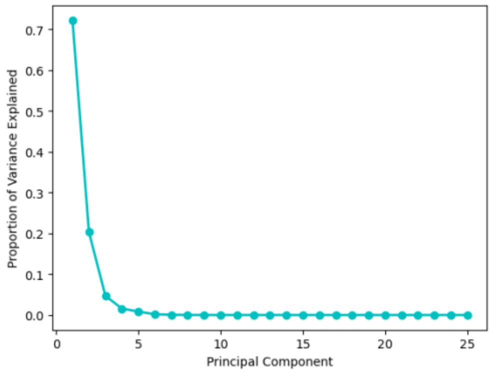


Figure 4.2 PCA Scree Plot

4.2.2. Gaussian Random Projection

The Gaussian Random Projection projects the original space on a randomly generated matrix in order to reduce the dimensionality. Optimizing the number of components, we got to 19, with slightly improvement in the results. More information could be found in table 4.2.

4.2.3. ISOmap

ISOmap is a non linear dimensionality reduction method that maintains the geodesic distance between two points (the shortest path on the surface itself). Its main advantage is that it is able to reveal and maintain the structure of the data, also revealing the ones that lie in multiple manifolds and clusters, making it useful for visualization, while not so much for other tasks, it selected 32 optimized components through iterative process.

Table 4.2 Performance comparison between methods of Feature Projectio

	Accuracy	**Precision**	**Recall**	**F1-score**
KNN Baseline	0.8494	0.8453	0.8513	0.8472
PCA	0.83	0.85	0.84	0.84
Improvement %	-2.3%	0.55%	-1.32%	-0.85%
Gaussian Random Projection	0.88	0.90	0.88	0.89
Improvement %	3.6%	6.5%	3.4%	5%
ISOmap	0.78	0.80	0.79	0.80
Improvement %	-8.1%	-5.3%	-7.2%	-5.5%

4.3. Exploiting Dimensionality Reduction

Dimensionality reduction can be very helpful to capture the "essence" of the data. As previously mentioned, PCA provided us with good results in terms of accuracy and F1-score, even though it resulted in a loss of information and explainability. The choice of using the PCA as the main dimensionality reduction method in this section, in order to search for the outliers in a low dimensional space, comes from the fact that it is easier to work and visualize the outliers using a scatter plot when using a 2-dimensional space.

4.3.1. Improving Anomaly Detection

Now having a low dimensional dataset with only 2 variables obtained with PCA, it is possible to add a visual component to the Anomaly Detection phase. As such, we repeated the same steps of chapter 2 with the added benefit of a scatter plot (seen in figure 4.3) to mark the identified outliers. Removing the outliers this time actually resulted in a slight increase in classification scores, proving that dimensionality reduction can improve anomaly detection.

As in chapter 2, we also replaced the identified outliers with the mean of each variable. The

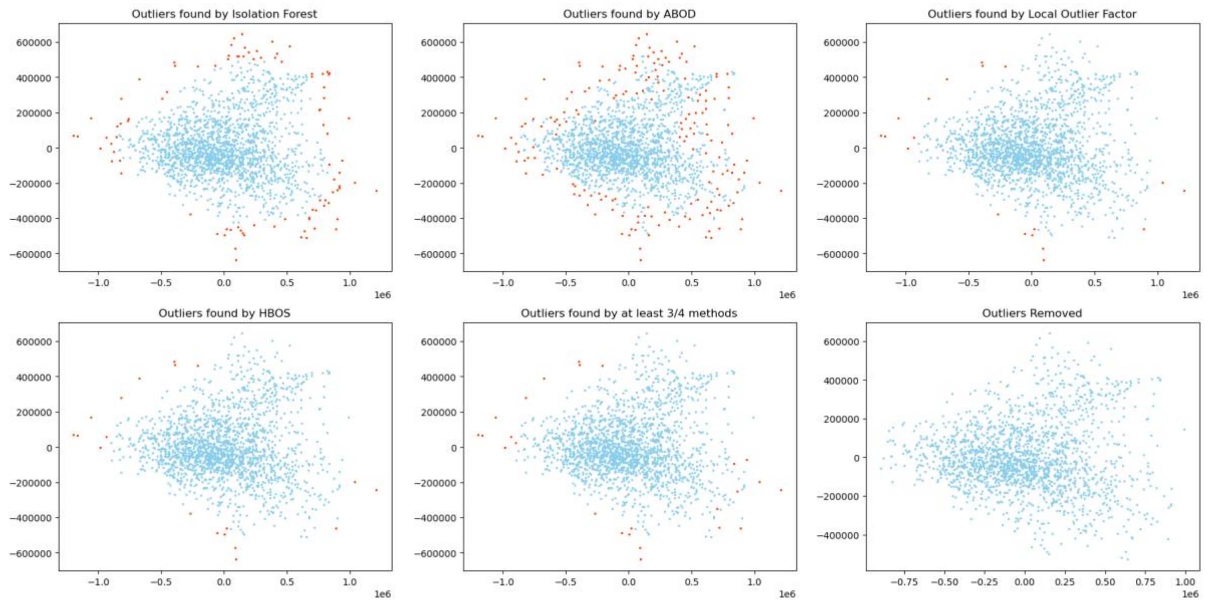


Figure 4.3 Outliers Visualization Using Dimensionality Reduction Technique PCA

comparison in performance can be seen in table 4.3.

Table 4.3 Dealing with Outliers Performance Improvement by using Dimensionality Reduction

	Accuracy	Precision	Recall	F1-score
KNN Baseline	0.8494	0.8453	0.8513	0.8472
Removing	0.9551	0.9551	0.9551	0.9551
Improvement %	12.4%	12.9%	12.1%	12.7%
Transforming	0.9551	0.9551	0.9551	0.9551
Improvement %	12.4%	12.9%	12.1%	12.7%

5. Advance Classification

In this section, first of all we defined a new classification task to classify multiple classes and we considered emotion as out target class variable. Then we implemented different classification algorithms, first as a base model and then optimized by using the hyperparameters found during the hyper tuning phase. We did this by trying many different settings quickly with a RandomSearch. This helped us know which settings to use when we did a more careful search with a GridSearch. This took longer, but it gave us the best settings. A 5-fold Cross Validation technique was also used during the searches to avoid overfitting the models.

Finally having both the baseline and optimized models, they were used to classify the testing set. Their performance was evaluated with the help of a classification report and other matrices. At the end of this section, in figure 5 we can find a summary of ROC and PR curves for each model. In addition, table 5 provides a comparison of the evaluation metrics for each model.

5.1. Naïve Bayes

We implemented two different Naïve Bayes algorithms, Gaussian and Bernoulli. First, we trained both algorithms without any hyperparameter tunning and we found 35% accuracy for Gaussian and 30% accuracy for Bernoulli. GridSearch have not provided any best parameters for this algorithm as it was expected because one of its principal characteristics is that it assumes independence between the predictors X, hence the name Naïve, this poses a problem since we already know from the feature selection phase that our features are indeed dependent on each other, therefore one would expect this classifier to perform badly on our classification task.

5.2. Logistic Regression

The logistic model or logistic regression, is a nonlinear regression model used when the dependent variable is of a dichotomous type. The goal of the model is to determine the probability with which an observation can generate one or the other value of the dependent variable; thanks to the Scikit-Learn library it is possible to use linear regression also in the presence of a multiclass, through the approach "one vs the rest". By using Logistics Regression without any hyperparameter tunning, we discovered 29% of accuracy. But after applying the hyperparameters found during the GridSearch phase i.e. penalty='l1', random_state=0, solver='liblinear' the accuracy have been improved to 44%.

5.3. Support Vector Machines

The objective of the Support Vector Machine algorithms is to find a hyperplane in an n-dimensional space (where n indicates the number of features) that distinctly classifies the data points. Hyperplanes are decision boundaries used in order to classify the data points. Data points falling on either side of the hyperplane can be attributed to different classes. The dimension of the hyperplane is related to the number of features: it can be a line or a two-dimensional plane. Support Vector Machines algorithms are very effective in high dimensional spaces.

5.3.1. Linear Support Vector Classification

Linear SVC is usually used when we can easily separate data with hyperplanes by drawing a straight line. Firstly, linear SVC has been applied to the dataset without hyperparameter and we got the accuracy of 32% which is very low and it was expected if we consider the dimensionality of the dataset. After, we considered using the parameters produced by the GridSearch i.e. C=1, dual=False, fit_intercept=False, random_state=0 we were able to achieve an accuracy of 47% which represents the benefits of using GridSearch.

5.3.2. Non-Linear Support Vector Classification

Non-Linear SVC is usually used when data cannot be separated with a straight line. The model uses Kernel functions and transforms non-linear spaces into another dimension (linear spaces) so that the data can be classified. Even if the performance with the tuned Linear SVC was a little better in terms of error measures, with the base model of the Non-Linear SVM the accuracy we got was 23%. After applying hyperparameters suggested by GridSearch i.e. C=1, gamma=0.1, random_state=0, the performance of Non-Linear SVC has improved to 47%.

5.4. Artificial Neural Networks

In this section, we have applied multi-layer perceptron classifier to classify emotions. Later, we did a learning strategies comparison.

5.4.1. Multi-Layer Perceptron Classifier

The most important hyperparameters to take into account are: hidden_layer_sizes. Which allows us to set the number of layers and the number of nodes we wish to have in the perceptron. Max_iter. Used to set the number of epochs. activation. Defines the activation function. solver. Specifies the algorithm for weight optimization across the nodes. The best results were found with the sdg solver, relu activation, 1000 max iterations and 2 hidden layers with 368 number of neurons. We have achieved 47% and 50% accuracy before tunning the model and after tunning the model respectively.

As seen from figure 5.1, the loss curves of both the base model and the tuned one, we can see that the NN trains the data in a very efficient way, with minimal loss and with a good fit considering the validation score, nor overfitting nor underfitting is present in the training stage. Additionally, after tuning, the results get slightly better.

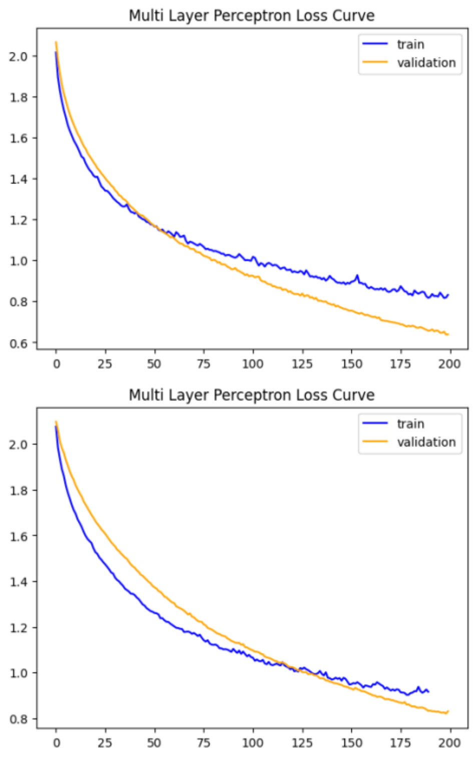


Figure 5.1 Multi-Layer Perceptron Loss Curve Before and After Tunning

5.4.1.1. Learning Strategies Comparison

Using the loss curve as a visualization tool to evaluate the quality of the learning phase, we can extend it to compare different learning strategies to help with hypertuning the model. By plotting them together we can see how a constant learning rate produces the least amount of loss while also generating a smooth learning process. This way we know to stay clear from learning strategies using momentum, which doesn't benefit our model at all. The multiple plots can be appreciated in figure 5.2.

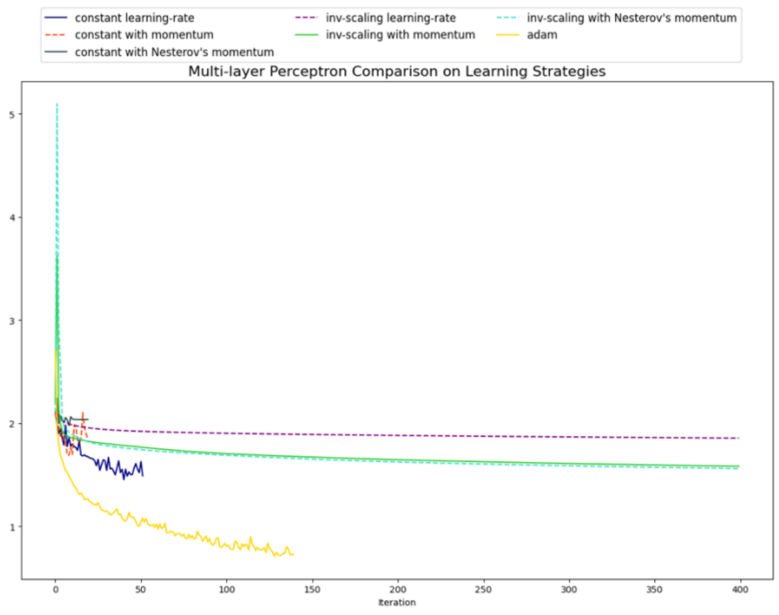


Figure 5.2 Stochastic Learning Strategies Comparison for MLP

5.5. Ensemble Classifiers

Ensemble learning improves the accuracy by aggregating the predictions of multiple classifiers in order to predict class labels of test records by combining the predictions made by multiple weak classifiers. The target is not to obtain highly accurate base models, but rather to obtain base models which make different kinds of errors. Ensemble classifiers usually can handle large datasets efficiently.

5.5.1. Random Forest Classifier

Random Forest Classifier runs efficiently on large databases so it should give us good results since it can also handle thousands of input variables.

Plotting the confusion matrix, we obtained the highest error when predicting class 7, predicted 42 times as class 2. However the model seems to work quite well.

5.5.2. Extra Trees Classifier

Extremely Randomized Trees Classifier aggregates the results of multiple decision trees, not correlated between them and collected in a "forest" to output its classification result. Random Forest chooses the optimum split while Extra Trees chooses it randomly. However, once the split points are selected, it chooses the best one between all the subset of features.

The confusion matrix of the Extra Trees model gave us slightly better results as the Random Forest: the highest error is again obtained while predicting class 7, predicted, this time, only 28 times as class 2.

5.5.3. Bagging Classifier

A Bagging classifier is an ensemble method that fits base classifiers each on a random redistribution of the training set, its final prediction is an aggregation obtained either by voting or by averaging. Unlike Random Forest, all features are considered for making the samples and splitting a node and not just a random selection. Bagging reduces overfitting by averaging, however, this leads to an increase in bias. The confusion matrix of the Bagging Classifier shows a misclassification of class 2 and 7, with 2 being predicted 23 times as class 7 and 7 predicted 22 times as class 2.

5.5.4. GradientBoost Classifier

The Gradient Boost Classifier is a group of machine learning algorithms: many weak learning models are combined together to create a stronger predictive model. The confusion matrix of this classifier produced the more or less same results as other ensemble classifiers e.g. class 7 has been predicted as class 2 for 31 times.

5.5.5. XGBoost Classifier

XGBoost is a refined and customized version of the GradientBoosting decision tree system, XGBoost stands for "eXtreme Gradient Boosting", meaning that the algorithms have been customized to improve the limit of GradientBoosting algorithms.

5.6. Performance Comparison

In order to compare the performances of the different algorithms previously introduced, we have plotted the Precision-Recall Curve and the ROC Curve for the training data with a 5-fold validation with all the models before and after tuning the hyperparameters as shown in figure 5.4. The accuracy score can be seen in figure 5.3.

Looking at the Precision-Recall Curve we can see that the best models, in terms of error measures, are XGBoost and Gradient Boosting. As for improvement, tuning the hyperparameters using random search and grid search did not increase the scores in any substantial amount, except for some cases like Bagging, Logistic Regression, and LinearSVC.

Since LinearSVC works really well when the classification task involves well separated classes, we can assume that this is probably the case even though we have 8 different classes. As previously noticed, also the Precision-Recall Curve of the logistic regression has its values close to 0.5, given to the fact that the "one vs the rest" approach has been used to make a multi-class prediction. Regarding the ROC Curve, the situation appears to be more or less the same: the Extra Trees and Random Forest have optimal curves. It’s important to note that all the models showed a relatively high misclassification rate when predicting class 7, predicted many times as class 2.

Table 5.1 shows the performance comparison between baseline and optimized models, in the case of the Ensemble Classifiers it is possible to notice that a better performance is usually achieved improving the number of n_estimators, obtaining more stabilized results. Interestingly, some metrics like Precision and Recall actually scored lower in the tuned model than in the baseline, this is because during the hypertuning phase, Accuracy was used as the goal metric to maximize.

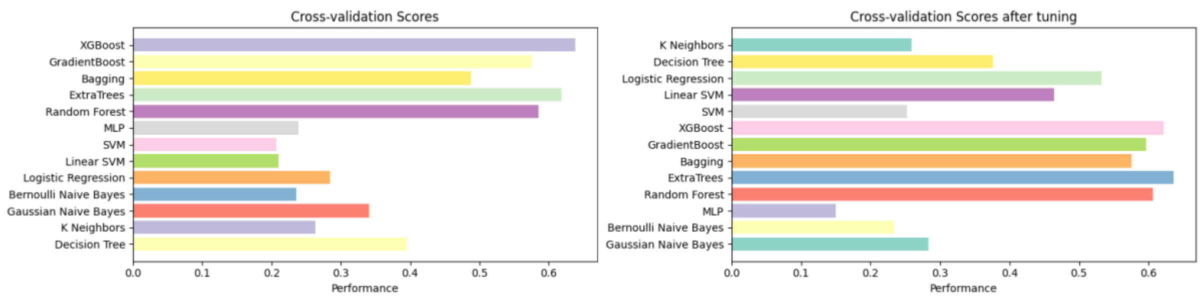


Figure 5.3 Cross Validation Accuracy performance before and after tuning

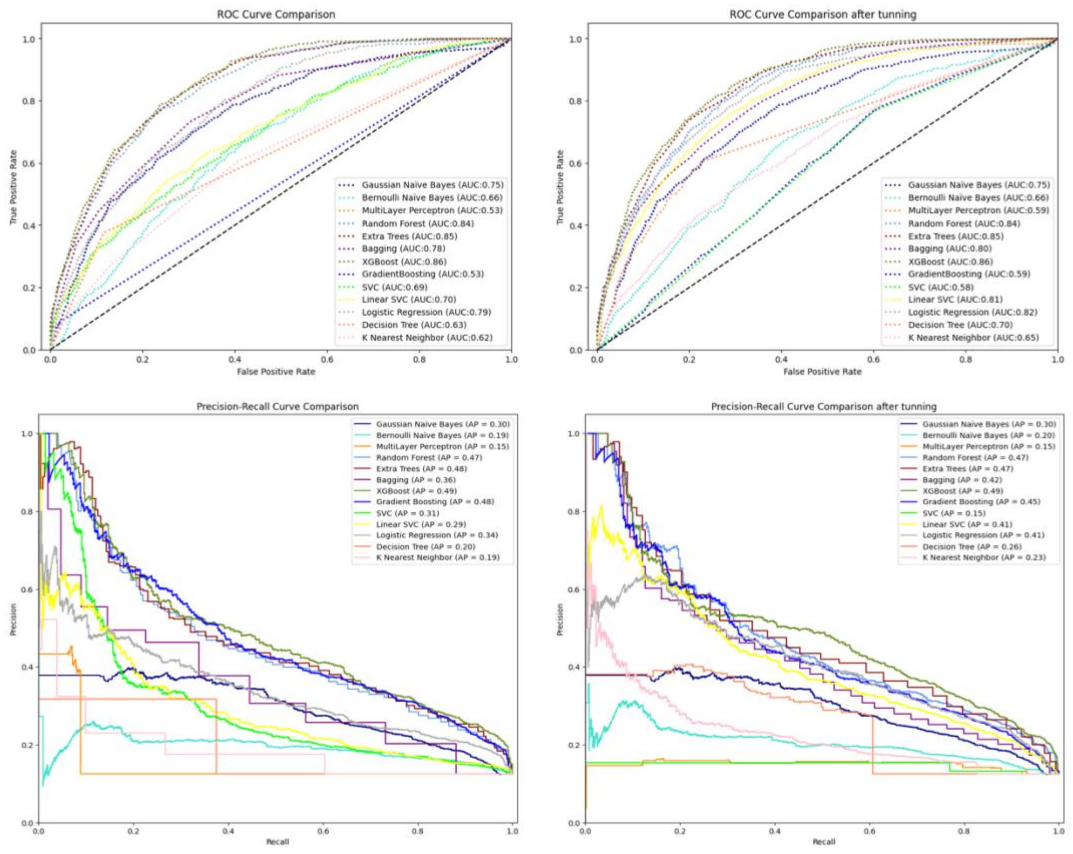


Figure 5.4 AUC-ROC and Precision-Recall Curves before and after tuning

Table 5.1 Performance comparison between baseline and optimized models

	Accuracy	**Precision**	**Recall**	**F1-score**
Gaussian Naïve Bayes	0.35	0.37	0.38	0.31
Bernoulli Naïve Bayes	0.30	0.29	0.30	0.29
Logistic Regression	0.29	0.28	0.26	0.25
Tunned Logistic Regression	0.44	0.43	0.44	0.40
Improvement %	51.7%	53.5%	69.2%	60%
Linear SVC	0.32	0.42	0.28	0.23
Tunned Linear SVC	0.47	0.47	0.51	0.43
Improvement %	46.8%	11.9%	82.1%	86.9%
Nonlinear SVC	0.23	0.16	0.19	0.15
Tunned Nonlinear SVC	0.47	0.47	0.45	0.44
Improvement %	104.3%	193.7%	136.8%	193.3%
MLP	0.47	0.47	0.47	0.45

Tunned MLP	0.50	0.49	0.49	0.48
Improvement %	6.3%	4.2%	4.2%	6.7%
Random Forest	0.41	0.40	0.39	0.39
Tunned Random Forest	0.43	0.45	0.42	0.41
Improvement %	2.4%	12.5%	7.6%	5.1%
Extra Trees	0.47	0.48	0.46	0.45
Tunned Extra Trees	0.45	0.44	0.44	0.43
Improvement %	-4.2%	-8.3%	-4.3%	-4.4%
Bagging	0.42	0.39	0.39	0.39
Tuned Bagging	0.45	0.46	0.44	0.43
Improvement %	7.1%	17.9%	12.8%	10.2%
Gradient Boosting	0.42	0.42	0.40	0.40
Tunned Gradient Boosting	0.40	0.41	0.38	0.38
Improvement %	-4.7%	-2.3%	-5%	-5%
XGBoost	0.28	0.36	0.25	0.23
Tunned XGBoost	0.28	0.40	0.25	0.23
Improvement %	0%	11.1%	0%	0%

6. Linear Regression

6.1. Simple Linear Regression

We decided to use Linear Regression, despite not being the perfect method for our dataset (continuous target variable is preferred). In the first approach (figure 6.1), we used two features with the highest correlation with the target variable. We found the best values of R^2, MSE, and MAE using ANOVA f-tests to deal with continuous attributes, but these results were not satisfactory. So, we performed another linear regression between the most correlated variable and a randomly selected one to get better results. In fact, in this case we obtained very bad results, all the results can be seen in table 6.1 below.

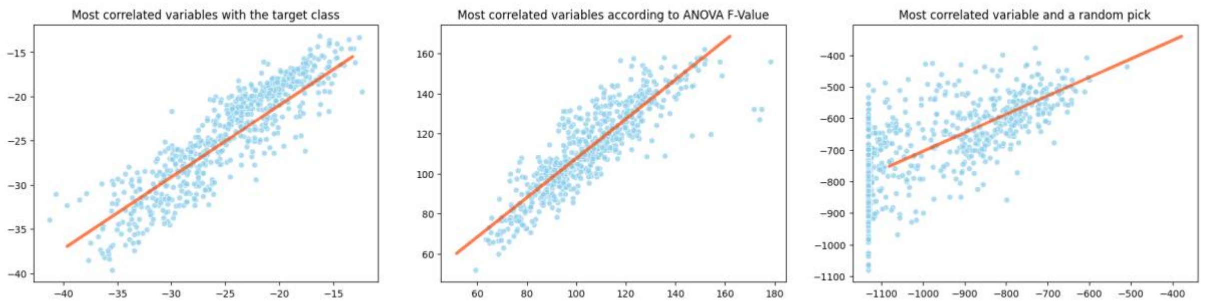


Figure 6.1 Simple Linear Regression on several continuous values

6.2. Multiple Linear Regression

We used multiple linear regression to address the issue of multicollinearity, VIF (Variance Inflation Factor) was used as a measure of the amount of multicollinearity present in the set of multiple regression variables. VIF is calculated by taking the ratio of the overall model variance to the variance of a model that includes only a single independent variable.

Unfortunately, the degree of multicollinearity is particularly high in our dataset, having 425 variables that are very similar to each other. It was possible to discover this through the scores obtained from the VIF defined above: usually, with a value greater than or equal to 5, the degree of multicollinearity is very high; in our case it is possible to reach even more than 1000. For this reason the values of R cannot be taken into 2 account, as they are highly distorted.

Nevertheless, for educational purposes, we decided to perform a multiple regression with the features with VIF value less than 10, giving us back 55 features.

6.2.1. Regular Linear Regression Method

Afterwards, we tried out a type of regression called regularized regression. This kind of regression limits the size of the coefficient estimates, so the model is less likely to overfit the data. There are two common types of regularized regression: Ridge regression and Lasso regression.

6.2.2. Ridge Linear Regression Method

Ridge regression is used to create a simple model when there are more predictor variables than observations, or when there is a correlation between predictors. It assigns small but evenly distributed weights to the predictors, which prevents overemphasizing one predictor. Ridge regression is good when there are many predictors, so we used the original dataset instead of reducing the predictors using VIF.

6.2.3. Least Absolute Shrinkage and Selection Operator Method

Lasso regression is another type of linear regression that uses shrinkage. It tends to give sparse weights (most zeros), because the L1 regularization cares equally about driving down big weights to small weights, or driving small weights to zeros.

As we expected, Ridge performs significantly better as it performs better when the number of predictors is large, unlike Lasso.

6.2.4. Solving Multiple Linear Regression with Gradient Boosting Machine

Another interesting approach is undoubtedly the use of Gradient Boosting for regression construction. As we mentioned before, Gradient Boosting is based on many weak learning models combined together to create a stronger predictive mode. For this purpose, we used the GradientBoostingRegressor of the Scikit-Learn library on the new dataset that came out from VIF analysis.

This method builds an additive model in a forward stage-wise fashion; it allows for the optimization of arbitrary differentiable loss functions. In each stage a regression tree is fit on the negative gradient of the given loss function. All the regression models performances are shown in the following overview table:

Table 6.1 Linear Regression Methods Results Comparison

	R^2	MSE	MAE
SLR Most correlated features	0.963	1.234	0.954
SLR Most correlated feature and random	-0.688	28969.390	161.298
MLR Regularized	0.003	5.088	1.910

MLR Ridge	0.120	4.490	1.749
MLR Lasso	0.098	4.600	1.810
MLR Gradient Boosting	0.027	4.962	1.886

7. Time Series Analysis

Time series analysis is the analysis of a sequence of data points collected over a specific interval of time. For time series analysis purpose, the audios from RAVDESS dataset have been transformed into segments using librosa library.

7.1. Data Preparation

The dataset has 2452 audio files including speeches and songs, with 1440 speeches and 1012 songs. After transforming each audio file, the dataset was split into a training set and a testing set. Time series samples for actors 1 to 18 were used for training, while the remaining samples were used for testing. This resulted in 1828 training samples and 625 testing samples.

7.2. Transformations

We can compare the difference between time series data by looking at their shape. By selecting two random time series i.e. 0th and 2nd from the dataset and plotting the shape between 50000 to 580000 timestamps, we calculated their distance using Euclidean and Manhattan distance metrics.

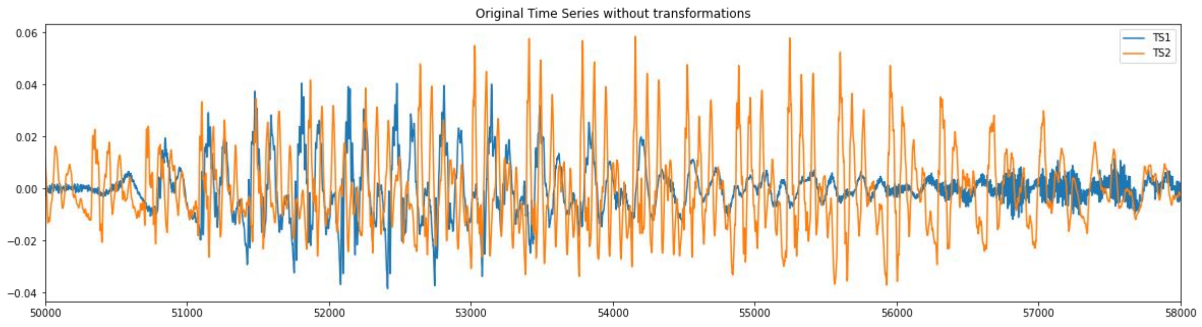


Figure 7.1 Raw time series without transformations

As shown in figure 7.1 the time series appear to be similar from one another with the Euclidean distance equal to 1.6112 and the Manhattan distance equal to 108.974.

7.2.1. Offset Translation

The first transformation applied on the time series is the Offset Translation: it consists in subtracting the mean from the original value of the time series. After the transformation the 2 time series appear to be slightly closer figure 7.2, with the Euclidean distance same as 1.6112 and the Manhattan distance slightly reduced to 108.967.

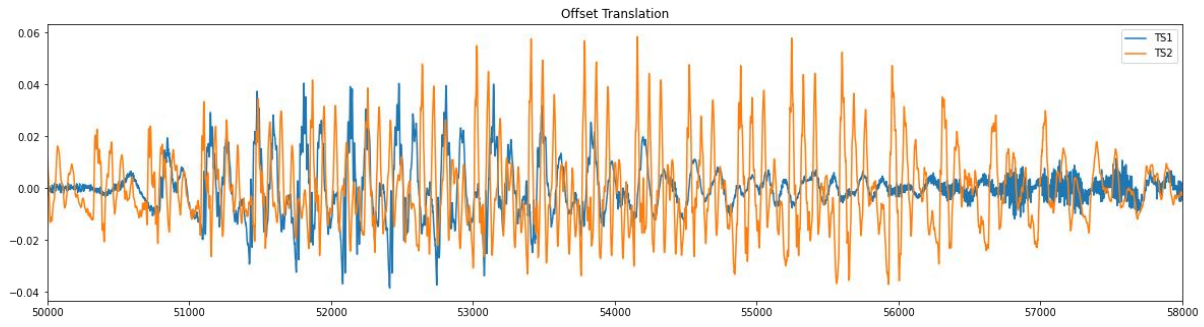


Figure 7.2 Offset Translation

7.2.2. Amplitude Scaling

Another useful transformation consists in scaling the amplitude of the times series. To scale the amplitude, the mean is subtracted from the original value of the time series and then the result is divided by the standard deviation figure 7.3. Doing so the Euclidean distance value raised to 125.69 and the Manhattan distance to 8460.36.

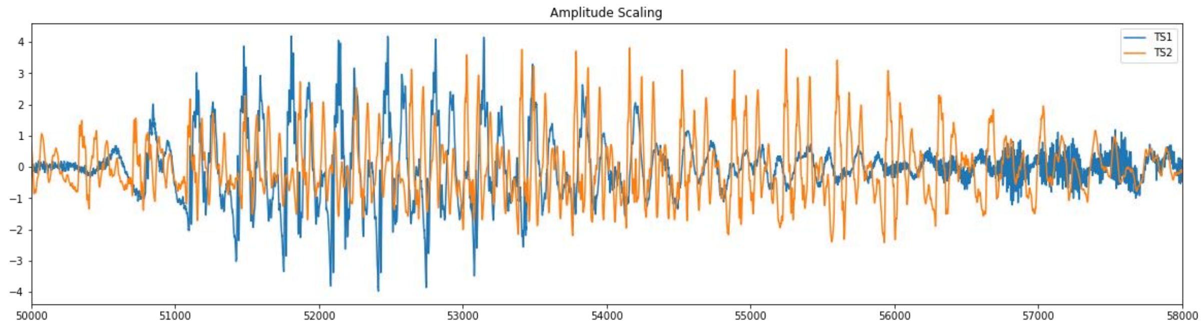


Figure 7.3 Amplitude Scaling

7.2.3. Trend Removal

Trend Removal aims to remove the differences given from the regression line; trend removal is important since trends can result in a varying mean over time making the time series non-stationary. We proceeded setting the size of a moving window equal to 8 subtracting each time its mean from the original value of the time series. As shown in figure 7.4, by removing the linear trend, the time series move in a straight direction, closer to one another resulted in the Euclidean distance equal to 0.5640 and the Manhattan distance equal to 37.103.

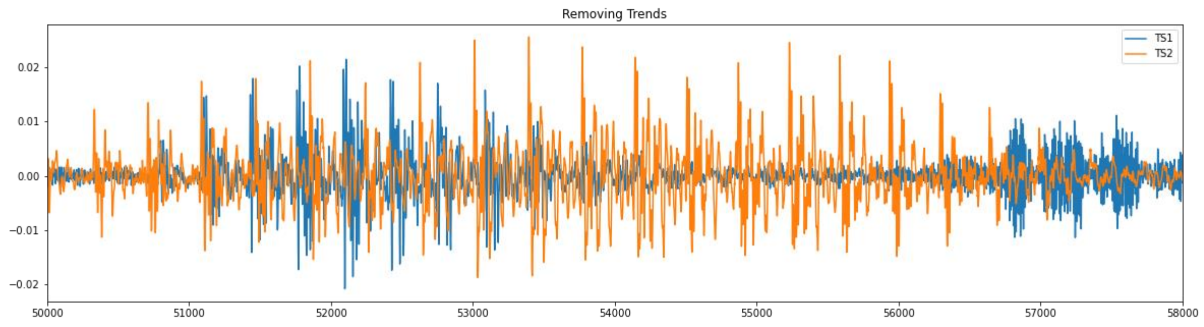


Figure 7.4 Trend Removal

7.2.4. Noise Removal

Noise Removal is another important transformation for the time series analysis, since the presence of messy data can hurt the final predictions. Removing the noise smoothed the time series giving us less messy lines figure 7.5, and the Euclidean distance equal to 121.73 and the Manhattan distance equal to 8170.9.

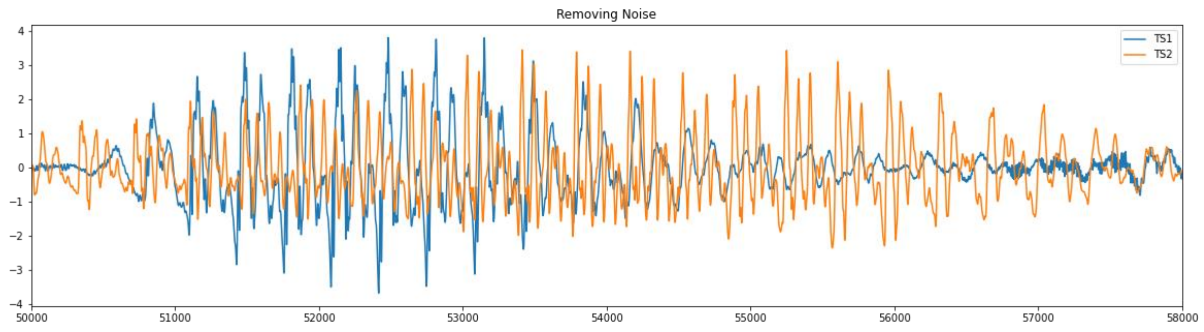


Figure 7.5 Noise Removal

7.2.5. Combined Transformations

Observing previous transformations helped understand how each of them affects the time series. Then we combine them in order to obtain a time series with the least amount of distortion. This resulted in the time series being more aligned and smooth figure 7.6, with the Euclidean distance equal to 0.2197 and the Manhattan distance of 11.647, way less than the ones found at the beginning.

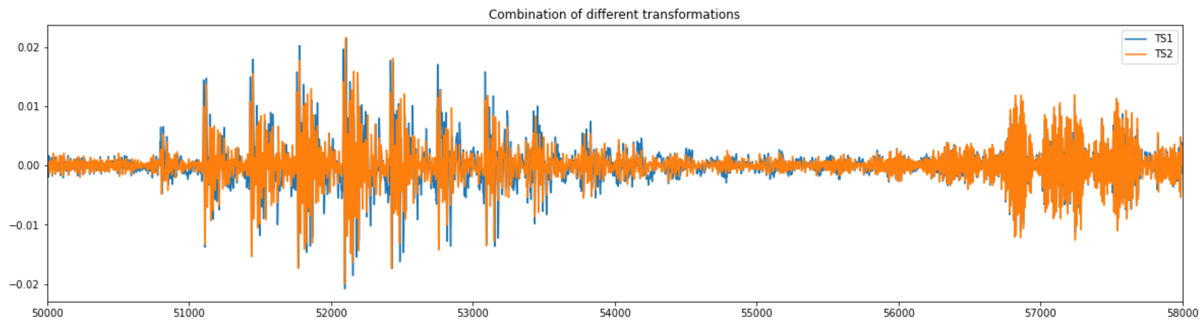


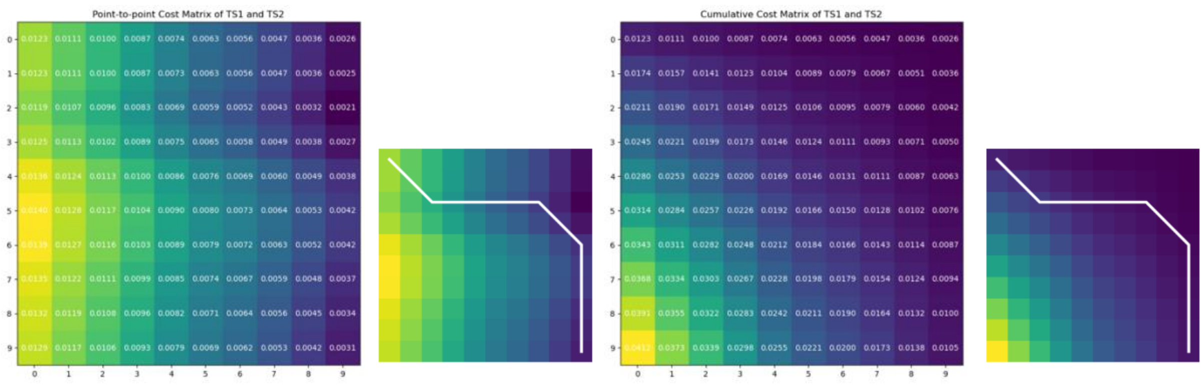
Figure 7.6 Combined Transformation

7.3. Dynamic Time Warping

Dynamic Time Warping is useful for aligning time series that are conceptually similar but evolve at different speeds. When using Euclidean distance as a similarity measure, the data can become misaligned since the sequences are aligned one to one in terms of points. With Dynamic Time Warping, it is possible to minimize the Euclidean distance between aligned time series under all admissible temporal alignments, which can provide better alignment results.

7.3.1. Computing the Cost Matrix

We computed point-to-point and cumulative cost matrices to find the best alignment between two time series using a time-span of 10 for better visualization. The point-to-point matrix helped us to see local distances between the points of both time series, and the cumulative matrix helped us to visualize the accumulated cost of any warping path. We aimed to find the optimal warping path that had the lowest total cost among all possible paths between the 2 time series. Figure 7.7 displays the results of the above experiments.



Point-to-point Cost Matrix of TS1 and TS2

	0	1	2	3	4	5	6	7	8	9
0	0.0123	0.0111	0.0100	0.0087	0.0074	0.0063	0.0056	0.0047	0.0036	0.0026
1	0.0123	0.0111	0.0100	0.0087	0.0073	0.0063	0.0056	0.0047	0.0036	0.0025
2	0.0119	0.0107	0.0096	0.0083	0.0069	0.0059	0.0052	0.0043	0.0032	0.0021
3	0.0125	0.0113	0.0102	0.0089	0.0075	0.0065	0.0058	0.0049	0.0038	0.0027
4	0.0136	0.0124	0.0113	0.0100	0.0086	0.0076	0.0069	0.0060	0.0049	0.0038
5	0.0140	0.0128	0.0117	0.0104	0.0090	0.0080	0.0073	0.0064	0.0053	0.0042
6	0.0139	0.0127	0.0116	0.0103	0.0089	0.0079	0.0072	0.0063	0.0052	0.0042
7	0.0135	0.0122	0.0111	0.0099	0.0085	0.0074	0.0067	0.0059	0.0048	0.0037
8	0.0132	0.0119	0.0108	0.0096	0.0082	0.0071	0.0064	0.0056	0.0045	0.0034
9	0.0129	0.0117	0.0106	0.0093	0.0079	0.0069	0.0062	0.0053	0.0042	0.0031

Cumulative Cost Matrix of TS1 and TS2

	0	1	2	3	4	5	6	7	8	9
0	0.0123	0.0111	0.0100	0.0087	0.0074	0.0063	0.0056	0.0047	0.0036	0.0026
1	0.0174	0.0157	0.0141	0.0123	0.0104	0.0089	0.0079	0.0067	0.0051	0.0036
2	0.0211	0.0190	0.0171	0.0149	0.0125	0.0106	0.0095	0.0079	0.0060	0.0042
3	0.0245	0.0221	0.0199	0.0173	0.0146	0.0124	0.0111	0.0093	0.0071	0.0050
4	0.0280	0.0253	0.0229	0.0200	0.0169	0.0146	0.0131	0.0111	0.0087	0.0063
5	0.0314	0.0284	0.0257	0.0226	0.0192	0.0166	0.0150	0.0128	0.0102	0.0076
6	0.0343	0.0311	0.0282	0.0248	0.0212	0.0184	0.0166	0.0143	0.0114	0.0087
7	0.0368	0.0334	0.0303	0.0267	0.0228	0.0198	0.0179	0.0154	0.0124	0.0094
8	0.0391	0.0355	0.0322	0.0283	0.0242	0.0211	0.0190	0.0164	0.0132	0.0100
9	0.0412	0.0373	0.0339	0.0298	0.0255	0.0221	0.0200	0.0173	0.0138	0.0105

Figure 7.7 Point-to-point Cost Matrix and Cumulative Cost Matrix with their Respective Paths

Knowing the different costs, it is possible to find the optimal warping path among all the possible ones, as shown in figure 7.8: the white line represents the optimal path drawn over the point-to-point cost Matrix using all the 50000 to 58000 time-span. The distance between the 2 time series using the Dynamic Time Warping is equal to 0.7855.

Since Dynamic Time Warping is very expensive to calculate, in order to speed up the similarity search it is possible to use global constraints. We applied the Sakoe-Chiba constraint with a radius equal to 80, which is the number of off-diagonal elements to consider which result in a distance equal to 0.9762. The last global constraint used is, in fact, the Itakura constraint with a maximum slope of 2. The Itakura parallelogram has a varying width, contrary to the Sakoe Chiba band which has always the same width in all of its parts, this allows larger time shifts in the middle. The optimal path generated with the Itakura constraint appears to be the best one, keeping a shape very similar to the original one, but faster and more straight, with a distance equal to 0.7891.

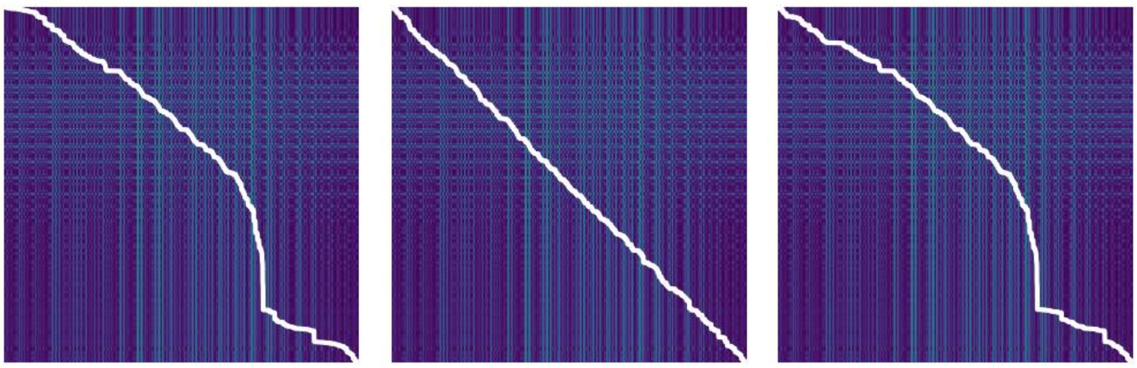


Figure 7.8 Complete Optimal DTW path without constraints (left), Sakoe-Chiba (center) and Itakura (right)

7.4. Approximation

Approximation is a type of dimensionality reduction used in time series analysis that simplifies the time series into a smaller space, while preserving important information and removing noise. Approximation can also be useful in clustering by reducing the space and then the Euclidean distance between the points in the time series. In this chapter different types of Approximation will be applied on the 0th time series of the dataset.

7.4.1. Discrete Fourier Transform

Discrete Fourier Transform (DFT) is a form of Approximation which describes a time series as a frequency rather than as a function of time, where the n_coefs parameter controls the number of Fourier coefficients to keep. Setting the number of coefficients equal to 480 we obtained the time series visible in figure 7.9. Between all the different Approximation methods we will be using this in this chapter, the DFT creates a wavy line, since it derives a frequency-domain representation of the signal.

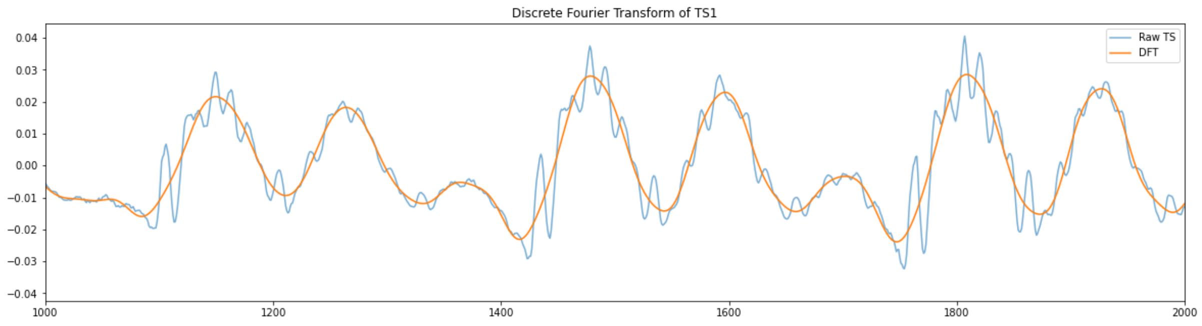


Figure 7.9 Discrete Fourier Transform

7.4.2. Piecewise Aggregate Approximation

Piecewise Aggregate Approximation (PAA) divides the time series in a previously specified number of equisized segments and then uses the mean of each segment to reduce the number of points in the time series by creating a vector of the values. By choosing a number of segments equal to 480 we plotted the approximated time series below the original one (figure 7.10). The new representation appears to be way more smooth than the original one.

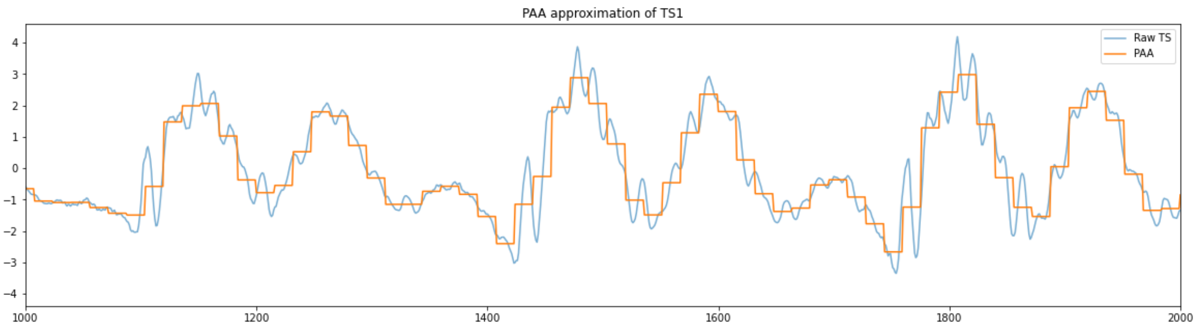


Figure 7.10 PAA Approximation

7.4.3. Symbolic Aggregate Approximation

The most important parameter of the Symbolic Aggregate Approximation algorithm (SAX) is the alphabet size used to discretize the time series in as many segments as the alphabet size that divides the distribution space into equiprobable regions. In this case the size of the alphabets has been set to 480 symbols. Plotting (Figure 7.11) the approximated time series it is possible to see that its shape is very similar to the one found using the PAA approximation, but this time the number of visible vectors is equal to the number of the letters in the alphabet.

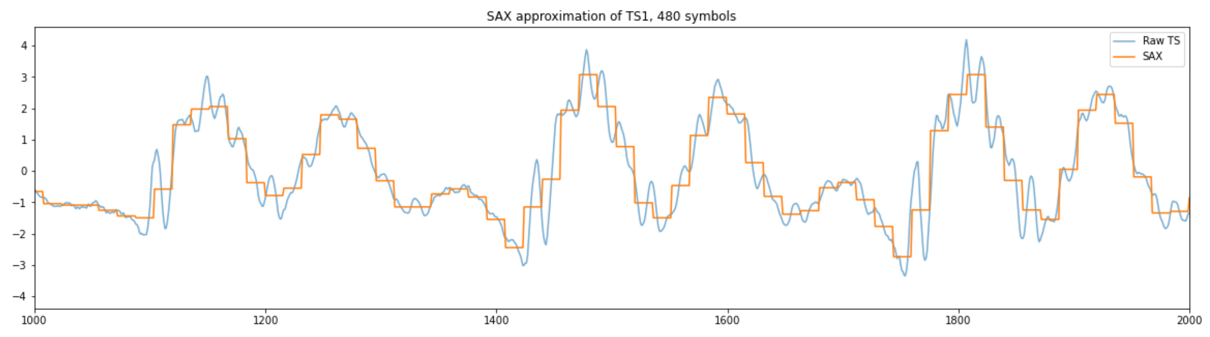


Figure 7.11 SAX Approximation

7.5. Time Series Clustering

Time series clustering works just like regular clustering, in which we measure the similarity of a set of observations and group them together in clusters that should best represent them. In this case we focused on partitional clustering, which is closely related to distances and hence, more fitting to our data. We applied two different types of clustering i.e. Features Based Clustering and Approximation Based Clustering on our dataset with different metrices such as Euclidean distance and dynamic time warping.

7.5.1. Features Based Clustering

To represent our data in another meaningful way, we decided to compute a set of statistical features for every time series within our data. They are: mean, standard deviation, variance, median, 10, 25, 50, 75 and 90 percentile, interquartile range, covariance, skewness and kurtosis. It is worth noting that this type of clustering is no longer done on time series and therefore can no longer use DTW distances. Figure 7.12 shows the clusters found by analyzing these features.

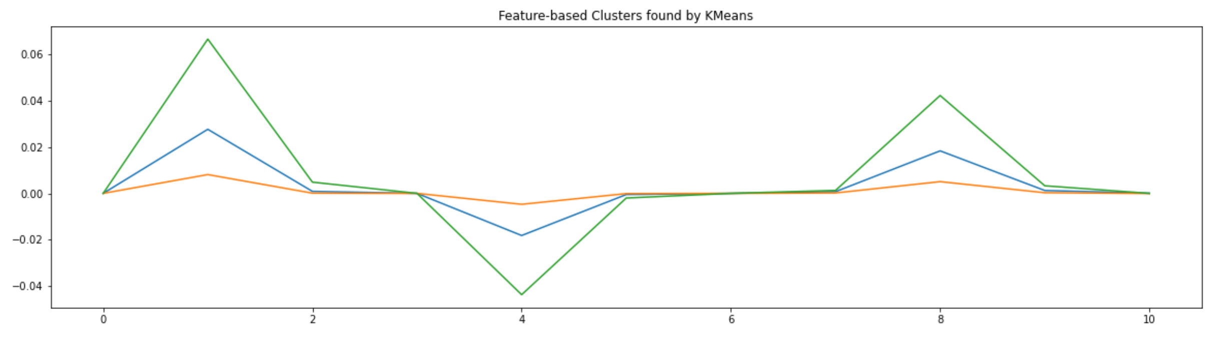


Figure 7.12 Feature-based Clusters

7.5.2. Approximation Based Clustering

7.5.2.1. KMeans Using Euclidean Distance

In this section, first we applied MinMaxScaler on the training set and narrowed down all the timeseries of the dataset to 5000 Piecewise Aggregate Approximation. Next, we used the elbow to specify the best value for k. Based on the SSE value we decided to use 4 as the k value where the SEE was 8.22 and Silhouette score was 0.14. We didn't get good clusters with these values. Figure 7.13 represents clusters found by KMeans algorithm using Euclidean distance.

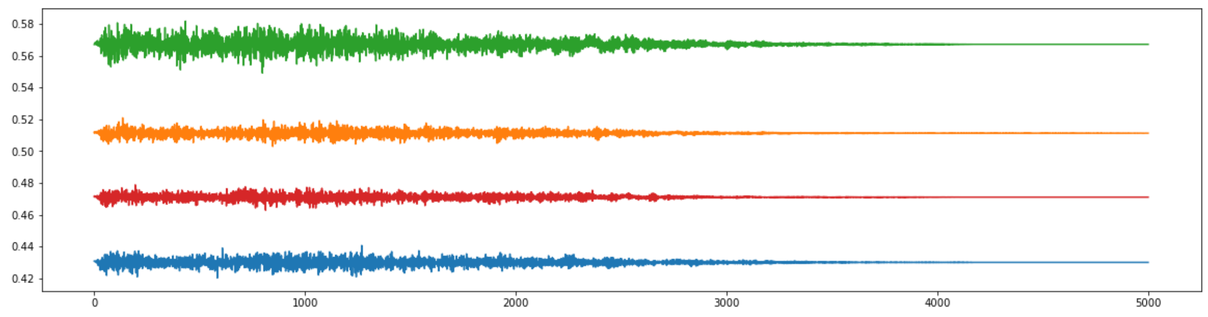


Figure 7.13 Clusters found by KMeans in approximated dataset using Euclidean distance

7.5.2.2. KMeans Using Dynamic Time Warping

In the next step, we used dynamic time warping as KMeans matric to compute clusters on the same dataset. Due to high complexity, computational cost, and lack of parallelization of DTW, it could take up to 5 days to compute clusters on this dataset, so we decided to reduce dimensionality of the dataset and to select only subset of the dataset i.e., records with only one type of statement expressing four different emotions from male and female, to run KMeans using DTW. In a nutshell, we come up with 560 records with 91648 timestamps. Then we used MinMaxScaler to scale the dataset and then applied 3000 Piecewise Aggregate Approximation. Even with that DTW were not able to run due highly complex task so we decided to reduce the dimensionality to 480 using PAA. Finally, we ran the KMeans using DTW we got the following results. We decided to compute 3 clusters by using the elbow method. The SSE score with 3 clusters was 0.112 and the Silhouette score was 0.521. Overall, we got messy clusters, there is still a lot of work needed to improve the results.

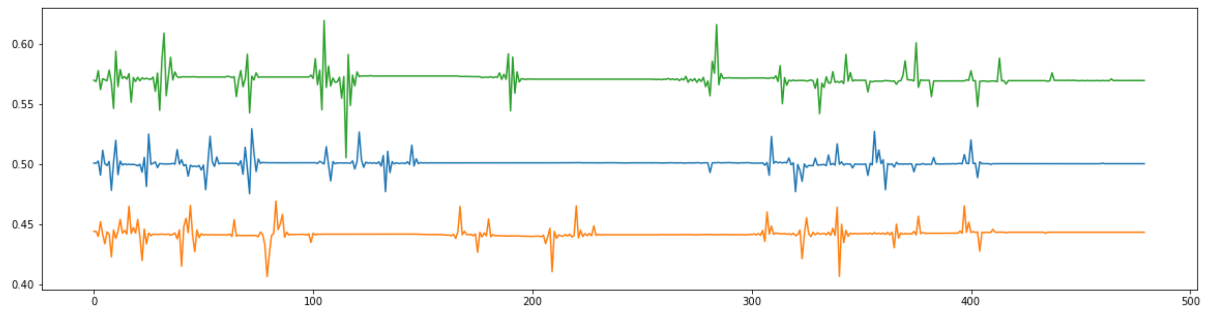


Figure 7.14 Clusters found by KMeans in approximated dataset using dtw

7.5.3. Clusters Analysis

In this section, we analyze the clusters using different techniques such as visualization of clusters using dimensionality reduction techniques, finding motifs, discords, and shapelets on the clusters produced by KMeans using Euclidean distance.

7.5.3.1. Cluster Visualization

Figure 7.15 displays clusters found in the first KMeans clustering using PCA techniques. As we said earlier the clusters result were not good and this is now clear that all the clusters are overlapping with each other and are not well separated. This may be due to the complex dataset and clusters may indicate that there are similarities or shared characteristics between the data points belonging to different clusters.

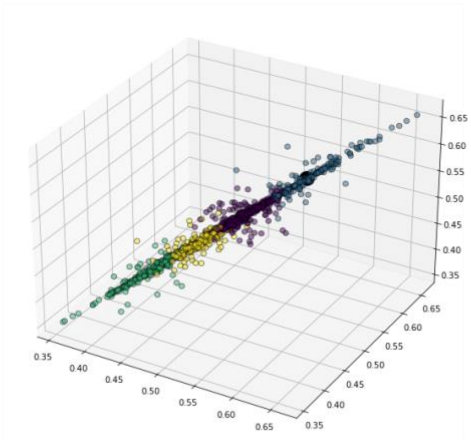


Figure 7.15 3D Visualization of Clusters

7.5.3.2. Shapelets Discovery

Shapelets Discovery aims to represent patterns that are highly predictive for the target variable. In this section we applied shapelet discovery on the clusters found using Euclidean distance. We found 5 Shapelets of length 500, due to very high dimensions of the dataset i.e. the clusters, the result of the shapelets also consist ambiguities and as like clusters if we improve our clustering then we can improve shapelets discovery as well. Figure 7.16 displays the shapelets found in the first cluster.

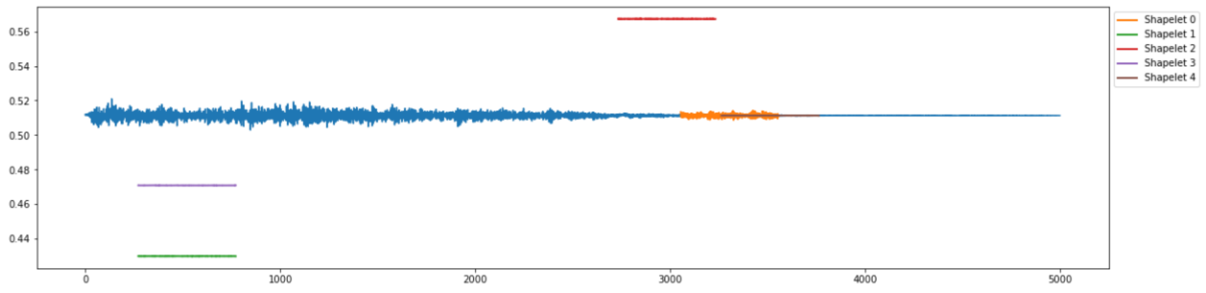


Figure 7.16 Example locations of shapelet matches in first cluster

7.5.3.3. Motifs and Anomaly Discovery

Time series Analysis can also be useful in order to find some patterns in our time series data. Those patterns can be representative of different classes or can repeat themselves in a sequence, creating a motif. Regarding motif analysis, we tried to find motifs on the clusters. Due to very high dimensions of the data, we didn’t get good results. We have tried several different techniques to improve the results but all in vain. The best results we get so far was using rolling window 12.

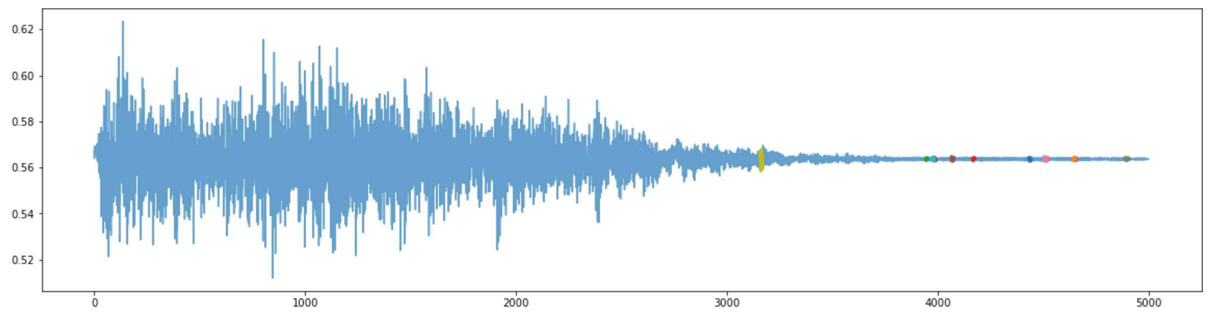


Figure 7.17 Motifs found in First Cluster

Anomaly discovery has given some good results. The following figure shows anomalies found in the first cluster where we can clearly see motions that are not found in any motif.

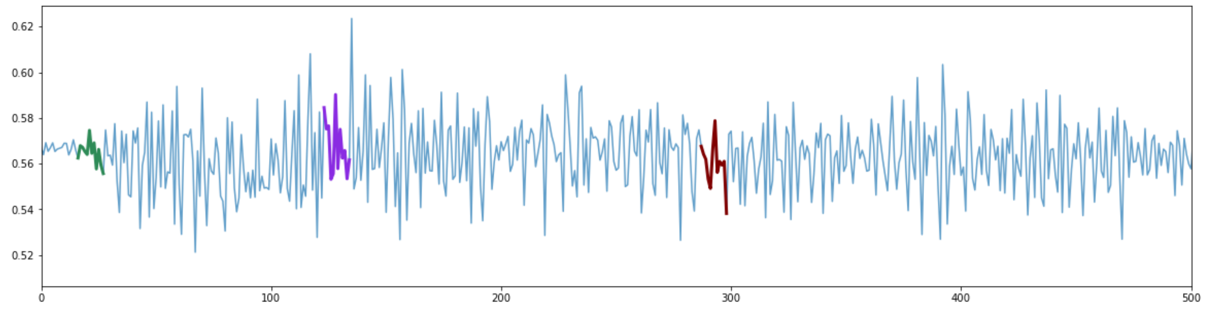


Figure 7.18 Anomalies found in First Cluster

7.6. Time Series Classification

The classification task we are going to solve in this section to classify vocal channel and emotions using the time series dataset. In the following section we have applied two different algorithms for the training purpose i.e. KNN, using two distance techniques such as Euclidean Distance and DTW, and two different neural networks algorithm.

7.6.1. K-Nearest Neighbor Classification

7.6.1.1. K-Nearest Neighbor using Euclidean Distance

First of all, we approximated our training and testing set to 2000 segments using PAA because keeping in mind the complexity of the dataset the algorithms could take days to compute the tasks. Then we scaled our dataset using TimeSeriesScalerMinMax(). After that, we applied the KNN classification on the processed data to predict vocal channel. The classification task produced good results around 60% of accuracy and 0.46 overall f-1 score.

After that the same dataset have been applied to predict multiclass target variable i.e. emotion. Keeping in mind the complexity the algorithm produced around 12% of accuracy and 0.09 overall f-1 score. Next, we tried to improve results using DTW as distance metric.

7.6.1.2. K-Nearest Neighbor using DTW

In this section, we have applied KNN on both types of classification tasks defined in the previous section. Keeping in mind the computational cost of DTW we decided to use sakoe chiba with radius 2 as our global constraint. After that, we applied the KNN algorithm on the same dataset and the results were improved exponentially with around 91% of accuracy and 0.91 overall f-1 score as well.

After that the same conditions have been applied to the multiclass classification task. The results for multi class classification have also been improved but the accuracy is still 23%. The results could be improved if more work is done on the dataset i.e. trying different hyperparameters but DTW is very costly and could take days for computation for a single hyperparameter. Table 7.1 shows a comparison of the KNN Binary class vs KNN Multi-class using Euclidean distance and DTW.

Table 7.1 Performance comparison of KNN Binary class vs KNN Multi-class using Euclidean Distance and DTW

	Accuracy	**Precision**	**Recall**	**F1-score**
KNN Binary Class - Euclidean	0.60	0.64	0.53	0.46
KNN Binary Class - DTW	0.91	0.91	0.91	0.91
Improvement %	51.6%	42.2%	71.6%	97.8%
KNN Multi Class - Euclidean	0.12	0.16	0.11	0.09
KNN Multi Class - DTW	0.23	0.21	0.23	0.22
Improvement %	91.6%	31.2%	109.1%	144.4%

7.6.2. Shapelet-Based Classification

In this section we applied Shapelet based classification on both of the classification tasks. First, we discovered shapelets considering binary class as a target and then we applied Shapelet Model on the shapelets discovered through the previous steps which produced 5 shapelets of size 300. The Shapelet model gave 0.50 of accuracy with 0.33 f-1 score. After that we transformed our training and test and then applied KNN on the transformed version which gave us 0.58 accuracy with f-1 score of 0.59 which is a good result in terms of high dimensionality of the dataset.

In the last, we discovered shapelets considering multi-class as a target and then we applied Shapelet Model which gave 0.13 of accuracy with 0.14 f-1 score. After that we transformed our training and test and then applied KNN on the transformed version which gave us 0.12 accuracy with 0.13 f-1 score. The overall result is not good as we can see the classifiers are founding it difficult to classify many classes.

7.6.3. Classification Using Neural Networks

In this section, we exploit the implementation of CNN and LSTM on time series for both classification task.

7.6.3.1. Classification using CNN

In this section, we have applied the same preprocessing on the dataset as we did in the previous section i.e. 2000 PAA approximation, then scaled the dataset using TimeSeriesScalerMinMax(). The CNN model we built was Sequential using relu as our activation function and adding a drop out of 0.3 at each hidden layer. At the dense layer we applied sigmoid activation function. After that we applied the model on multi class problem which gave us not so good results with only 0.18 of accuracy.

After that we applied the same model on the binary class problem and we got very good results of around 0.73 of accuracy on the validation set.

7.6.3.2. Classification using LSTM

In this section, keeping the same preprocessed dataset, we built a Sequential LSTM network using drop out of 0.5 at the first hidden layer and 0.2 at the second hidden dense layer. We applied relu as activation function at the first dense layer and the sigmoid activation function on the output layer. After that we applied the model on multi class problem which gave us not so good results with only 0.13 of accuracy.

After we applied the same model on the binary class problem which also didn't perform well and gave results around 0.51 of accuracy on the validation set.

8. Explainability

Explainability consists of a set of tools which make it possible to explain and understand, in a human comprehensible form, the black box models, whose internal functioning is usually inaccessible or not understandable (Neural Network models, Ensemble Classifiers, etc.). The goal is to provide an understandable explanation of how the decisions leading to a certain output were made. In this chapter we use the LIME algorithm to explain the predictions made using the Random Forest Classifier on our data and SHAP algorithm to explain the predictions made using the Random Forest Classifier as well.

8.1. LIME

LIME, Local Interpretable Model-agnostic Explanations, is a Black Box explanation method, meaning that it does not substitute the black box with a more understandable model, but rather tries to extract the logic behind it. As an agnostic model, it can be applied to any different kind of model, as long as it provides probability scores.

In figure 8.1, we can see an example of LIME applied to a particular instance of our data. In this case we want to see how the Random Forest classifying model came to the conclusion that a particular instance of our test data belongs to a certain class. Let's take for example record 35, which was predicted to belong to class 2 (Speech) by the model and is indeed Speech according to the data. For this record, the model assigned the following probabilities of belonging to class [1, 2], being: [0.22, 0.78]. Now LIME allowed us to see how these probabilities came into existence. by stating the average influence that a particular feature had on the predictions and the exact value of those features for the record in question. In the case of our example record, it tells us the features with the biggest weight on the classifying decision, namely stft_q25 and stft_q05_w4 among others, while at the same time stating the features that contributed the most to the other possibilities of classification, for example length_w2 and length_w4.

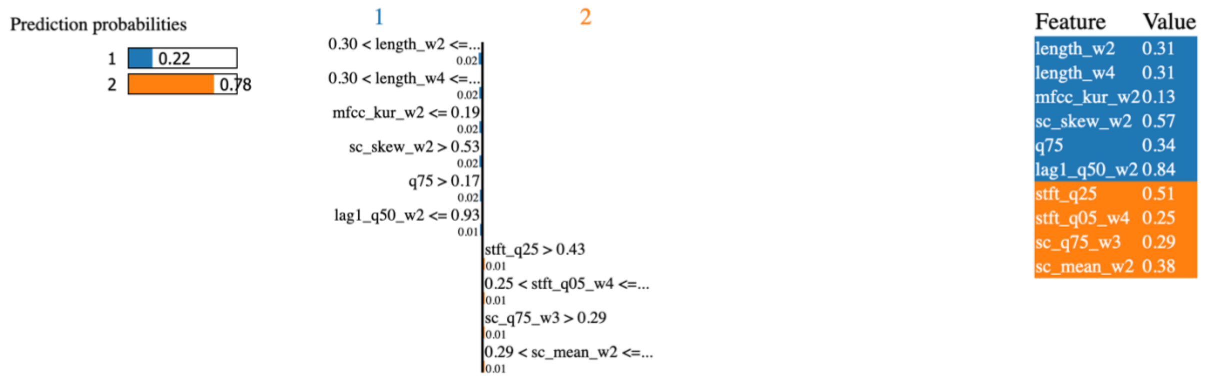


Figure 8.1 Results of LIME algorithm for the classification of record 35

Let’s take another for example, figure 8.2, record 95, which was predicted to belong to class 1 (Song) by the model and is indeed Song according to the data.

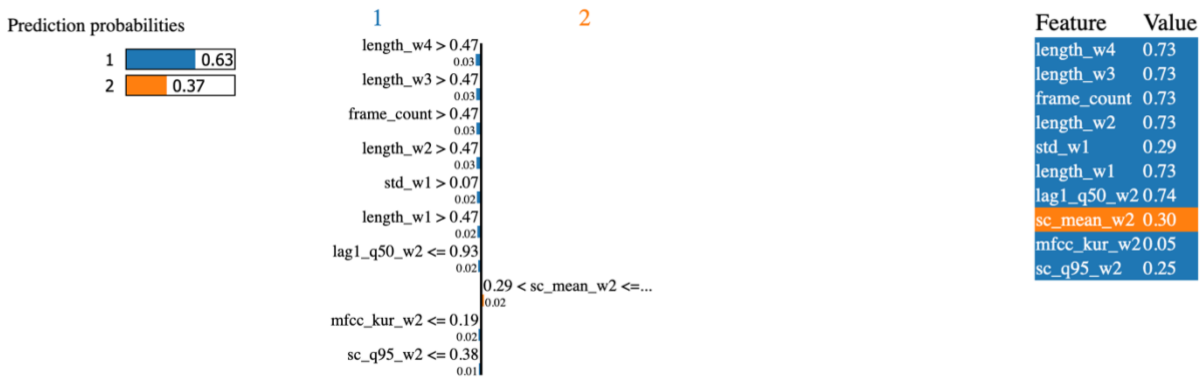


Figure 8.2 Results of LIME algorithm for the classification of record 95

8.2. SHAP

Shapley Additive Explanations is, as LIME, a Black Box explanation method. The method is based on shapley values which represent the average contribution of a feature value among all its possible combinations.

For this analysis, we decided to use SHAP on the same classification obtained through Random

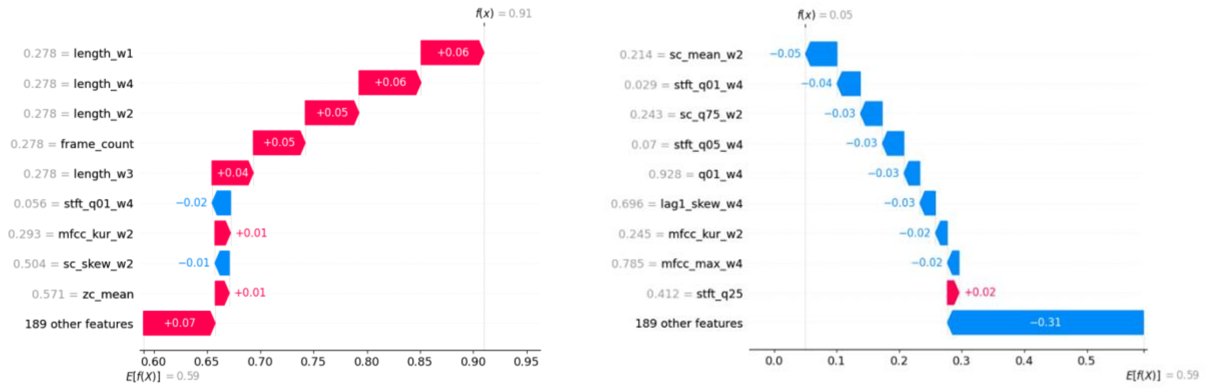


Figure 8.3 SHAP Waterfall plot for record 459 and 900

Forest Classifier. With regard to the study of features at the local level, we decided to compare two records of the training set representing the both classes: the 459th record for the Speech class and the 900th record for the Song class. Figure 8.3 shows which variables have the most influence on the instance of Speech and Song class.

After that we used the SHAP on the test record to get explanation of the test records from 0 to

50. The stacked force plot in figure 8.4 displays how each variable has contributed to the final result for each record in the test set.

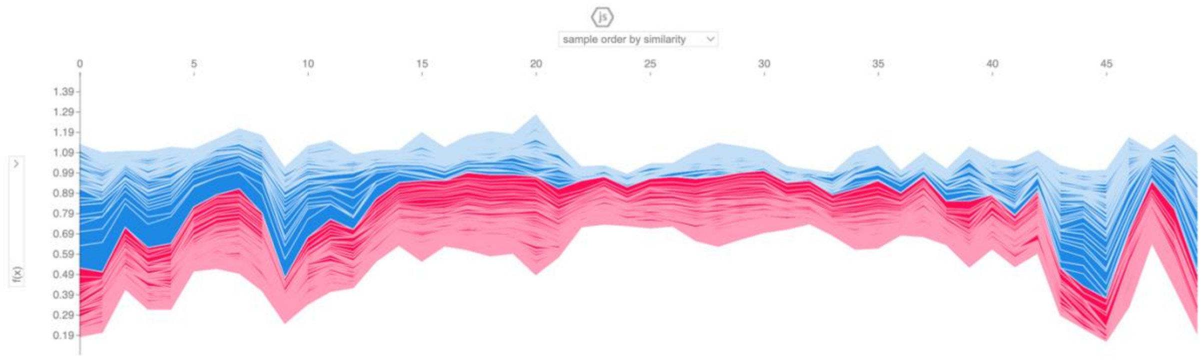


Figure 8.4 SHAP force plot for first 50 test records