Data Mining Project

Human Activity Recognition

2021/2022

AUTHORS

DAVIDE DI VIRGILIO
GIORGIA D'ANTONI
BRUNO LIMON

UNIVERSITÀ DI PISA

Table of contents

1 Data Understanding & Preparation

1.1 Data Understanding

The Human Activity Recognition Dataset monitors the activity of a group of 30 volunteers within an age bracket of 19-48 years. The activities performed by the volunteers while wearing a smartphone (Samsung Galaxy S II) on their waist are the following six: *Walking, Walking Upstairs, Walking Downstairs, Sitting, Standing and Lying.* These activities will be considered as the dependent variable y in our data and they are labeled from 1 to 6. Also, the dataset comes already divided into training and testing set. The dataset does not contain any missing value.

The dataset is composed by 561 features: using the sensors (gyroscope and accelerometer) in a smartphone, 3-axial linear acceleration (*tAcc-XYZ*) have been captured from the accelerometer and 3-axial angular velocity (*tGyro-XYZ*) from the gyroscope with several variations. The acceleration signal was separated into Body and Gravity acceleration signals*(tBodyAcc-XYZ* and *tGravityAcc-XYZ*). After that, the body linear acceleration and angular velocity were derived in time to obtain *jerk signals (tBodyAccJerk-XYZ* and *tBodyGyroJerk-XYZ*).

In order to know the distribution of the data, a sample of variables had to be taken to analyze since it would be extremely time consuming to analyze each and everyone of the 561 variables, as such, a sample of 20 random ones was chosen to be analyzed by plotting their respective values with an histogram and then finding the best fitting distribution function by calculating its sum of squared error against the histogram and then picking the one with the lowest value. This way, 3 main distributions were found; *Log-Normal, Cauchy* and *Normal*. The most representative results of the analysis can be visualized in figure 1.

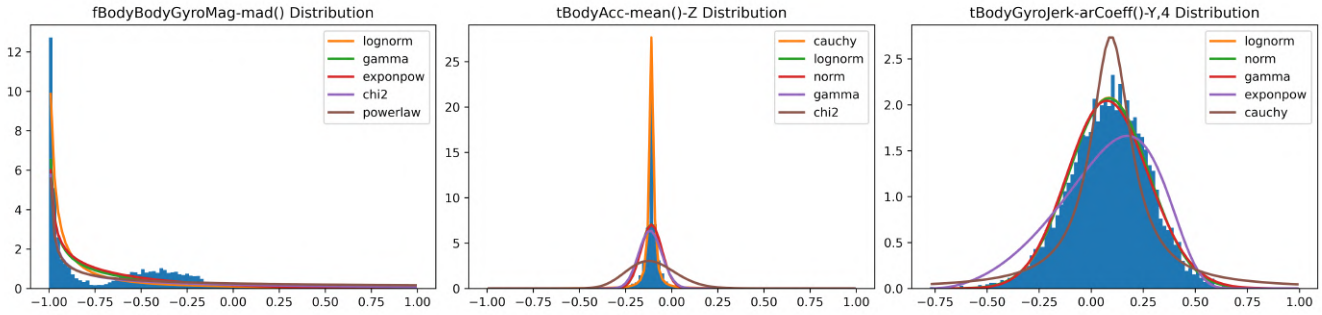


Figure 1 - Most common distributions among the X variables

1.2 Data Preparation

In order to make more sense of our data we used the list of features as column names and concatenated the training and testing data so that we could have a complete overview of the full data. To prepare the data for further use we then created a pair of vectors containing the values of features X and a pair of 1 dimensional arrays containing the values of the class labels y for the training as well as for the testing data.

1.3 A first approach to Classification

First of all we decided to check the results of the classification report using two different classification methods: Decision Tree and KNN. We ran the algorithms without improving the dataset in any way so that we could later have a baseline to compare to.

1.3.1 Decision Tree

The Decision Tree classifier is used to predict the target variable by learning simple decision rules inferred from the data features, in this case the classifier uses the default hyperparameters since we are only interested in having a baseline score to return to when evaluating further algorithms in this project.

From the classification report class 6, *Lying*, appeared to be perfectly classified, with a value of 1.0 in Precision, Recall and F1-score. The rest of the classes are more in line with the averages seen in table 1. From the AUC-ROC and Precision-Recall curves in figure 2 we can see that class 2 and 3 are the hardest to classify.

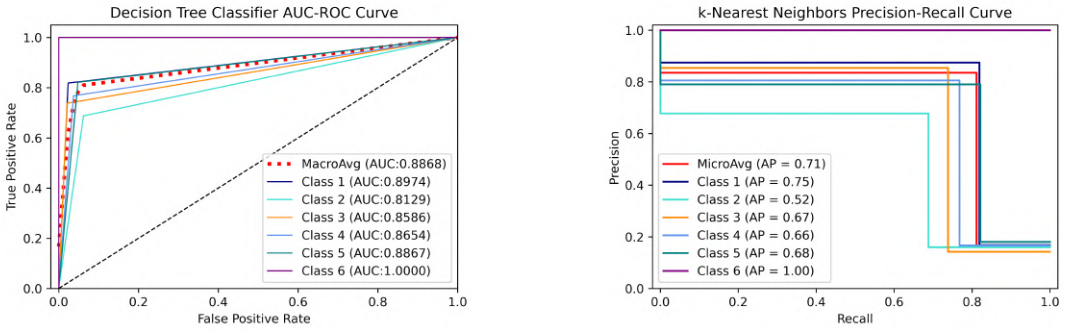


Figure 2 - AUC-ROC and Precision-Recall Curve for Decision Tree Classifier

1.3.2 K-Nearest Neighbors

KNN is a supervised learning algorithm which assumes that objects with similar properties are closer in terms of some distance metrics (Minkowski distance in our case). In other words, similar things are near to each other. As in the case of Decision Tree, no hyperparameter is needed.

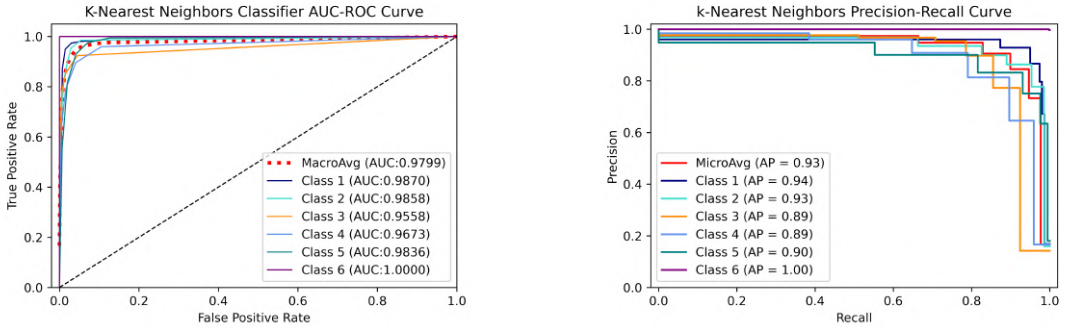


Figure 3 - AUC-ROC and Precision-Recall Curve for Decision Tree Classifier

Table 1 - First Approach at Classification

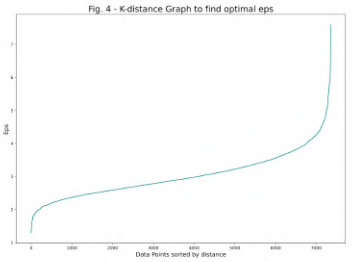
	Accuracy	Precision	Recall	F1-score	AUC-ROC
Decision Tree	0.8595	0.8593	0.8561	0.8567	0.8868
K-Nearest Neighbors	0.9016	0.9067	0.8968	0.8985	0.9799

2 Anomaly Detection

Anomaly detection is the identification of rare events, items, or observations which are suspicious because they differ significantly from standard behaviors or patterns. Finding anomalies in a dataset is a crucial step because it can drastically improve performances in every type of analysis. Our target was to identify the top 1% outliers in the dataset. The following section describes the algorithms used for this essential task and the corresponding results on our dataset.

2.1 DBSCAN

DBSCAN is often used as a clustering technique, but its robustness to outliers makes it an excellent tool for anomaly detection. It is a density-based clustering non-parametric algorithm: given a set of points in some space, it groups together points that are closely packed together (points with many nearby neighbors), marking as outliers points that lie alone in low-density regions. Using the Elbow Method in figure 4, we set Eps = 4.7. The output in this case gave us 155 records labeled as noise, in this case, our outliers.



2.2 Isolation Forest

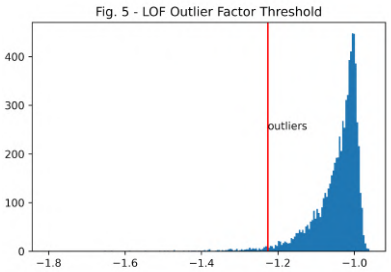
Isolation Forest explicitly identifies anomalies instead of analyzing normal data points. Isolation Forest is built on the basis of decision trees, the partitions of the trees are the result of a random selection of a feature and a random split between the minimum and maximum value of it. The core idea is that it should be very easy to “isolate” anomalies based on the characteristics that make them unique. Technically, fitting a decision tree on all the observations, outliers should be found closer to the root of the tree than “normal” instances. The number of outliers found assigning the contamination value at 0.02 and max_sample equal to 1000 is equal to 148.

2.3 Angle-based Outlier Detection

The angle-based outlier detection algorithm (ABOD) calculates the angle variance based on the whole data. Instead of examining neighborhoods as proximity-based concepts, ABOD assesses the broadness of the angle spectrum of a point as an outlier factor. In this case the algorithm was set with the following parameters: number of neighbors equal to 7 and a contamination of 0.2. The algorithm identified 161 outliers.

2.4 Local Outlier Factor

The Local Outlier Factor (LOF) algorithm is an unsupervised anomaly detection method which considers as outliers the samples that have a substantially lower density than their neighbors. LOF shares some concepts with DBSCAN such as the concepts of "core distance" and "reachability distance", which are used for local density estimation. The local outlier factor is based on a concept of a local density, where locality is given by k nearest neighbors, whose distance is used to estimate the density.



By comparing the local density of an object to the local densities of its neighbors, one can identify regions of similar density, and points that have a substantially lower density than their neighbors. These are considered to be outliers. Regarding the parameters, we chose a number of neighbors equal to 7 and a contamination of 0.02. It found 148 outliers.

2.5 Dealing with outliers

Among all the methods previously used, DBSCAN seems to have the worst performance on our dataset finding way too many outliers with respect to the other methods. This could probably be due to the fact that it is a density-based approach, meaning it usually works better in low dimensional space since density becomes less meaningful in high dimensional space. Despite this, the LOF method seems to work quite well and it differs from the DBSCAN method in the fact that it compares the local density of a point with respect to the local density of its neighbors. The Isolation Forest model works well on our dataset because it can handle high dimensional data and is also quick in isolating anomalous values.

2.5.1 Removing the outliers

After detecting the outliers, we tried several approaches to manage them. 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 68, representing 0,92% 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.

2.5.2 Transforming the Outliers

Transforming the outliers is another approach to deal with them. 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. After this we confirmed that the anomaly detection algorithms only detected the records previously left out of the 3 out of 4 methods filter.

Table 2 - Performance comparison of methods to deal with outliers

	Accuracy	Precision	Recall	F1-score
DT Baseline	0.8595	0.8593	0.8561	0.8567
Removing	0.8558	0.8549	0.8516	0.8521
Improvement %	-0.43%	-0.51%	-0.53%	-0.54%
Transforming	0.8585	0.8574	0.8546	0.8548
Improvement %	-0.12%	-0.22%	-0.18%	-0.22%

Although the baseline classification performance didn't improve after removing or transforming the outliers, it is still a viable technique to be considered when working with classifiers where the presence of outliers is highly detrimental to the classification performance. In any case, we expect to improve upon this process in a further section.

3 Imbalanced Learning

Looking at the class frequency distribution in figure 6, we can tell if the dataset is imbalanced.

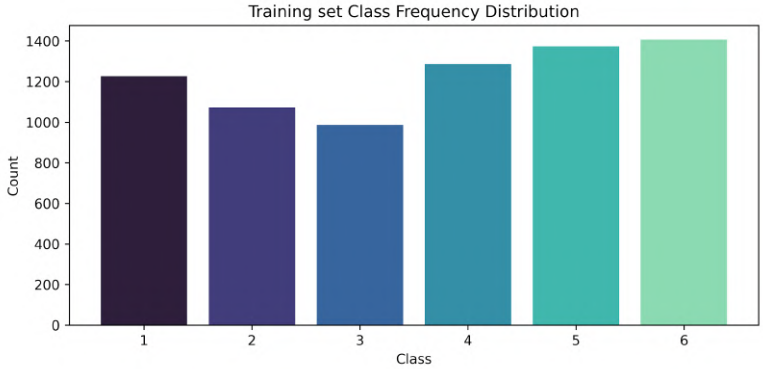


Figure 6 - Class Frequency Distribution

The plot shows that the 6 different classes are relatively balanced, counting 1226 values in the first class, 1073 in the second, 986 in the third, 1286 in the fourth, 1374 in the fifth and 1407 in the sixth and last class. For educational purposes, we decided to imbalance the data and then use balance techniques on it, aiming to find better results. We decided to keep classes 2 and 3.

3.1 Imbalancing pre-processing

As already mentioned, we kept class 2 and 3 (*Walking upstairs* and *Walking downstairs*) with 1073 and 986 values each, in order to deal with a binary class situation so as to be able to perform a meaningful oversampling, we first removed 942 random values from the second class, lowering it down to only 44 rows against the 1073 of the third one to obtain a 96% - 4% proportion.

Unlike oversampling, for a meaningful undersampling an oversizing process on the majority one was performed, necessary to avoid using a class containing less than 100 records, too few to be able to train a good model. That's the reason why we started from a dataset containing 8584 class 2 values and 986 class 3 values.

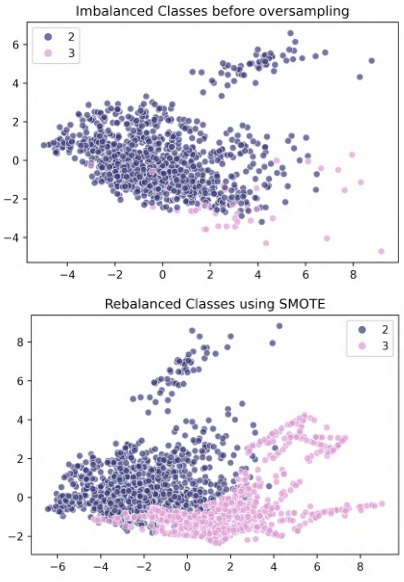


Figure 7 - Undersampling Results

3.3 Oversampling

3.3.1 SMOTE

The SMOTE method randomly selects an example from the minority class and then finds its k nearest neighbors, it then proceeds to create synthetic data between the random example previously taken and the randomly selected k-nearest neighbors.

3.4 Undersampling

3.4.1 Random UnderSampler

Using the Random Under Sampler which randomly selects examples from the majority class and then discards them from the training set, we balanced the classes obtaining 986 values for each class. After performing the rebalancing of the classes, we were left with equal size classes, ready to be used for a better fit into a predictive model. In this particular case the performance of the baseline model didn't improve since we had to heavily modify the dataset in order for it to fit our particular needs, however, the extent at which the oversampling and undersampling techniques can go is well demonstrated. Their 2D visualization can be seen in figure 8.

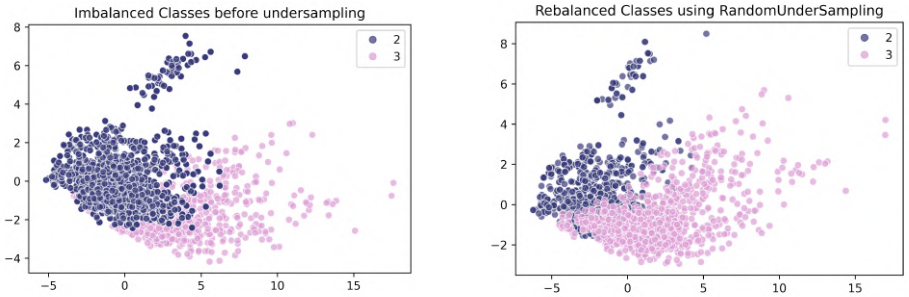


Figure 8 - Undersampling Results

4 Dimensionality Reduction

Dimensionality reduction is essentially the process of reducing the dimensionality of the feature space while trying to keep as much information as possible; it helps to reduce the execution time and resources needed, but more importantly, it is the best tool in the fight against the curse of dimensionality and data being too sparse, preventing overfitting of the model and leading to a better generalization of new data. Dimensionality reduction can be further divided into feature selection and feature projection.

4.1 Feature selection

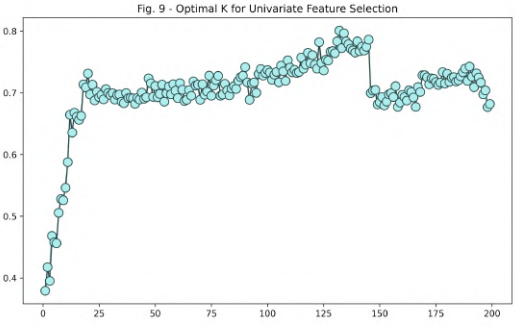
Feature selection tries to select a subset of the original features present in our dataset. This allows us to remove redundant and irrelevant features without having much loss of information since the most important features are kept. As before, the results of every method will be evaluated and compared at the end of the subsection in table 3.

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 0.10, meaning we removed the features that are at least 90% similar, therefore reducing the risk of multicollinearity, with this we obtained a selection of 189 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. Each feature gets compared to the target variable, to see whether there is any statistically significant relationship between them. 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 Decision Tree classification (seen in figure 9). With this we set k to 131.



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. This method gives the best results since Decision Tree directly partitions the sample space at each node, meaning that if the sample space decreases and so does the distance between the data points, it will be easier to find a good split.

Table 3 - Performance comparison between methods of Feature Selection

	Accuracy	Precision	Recall	F1-score
DT Baseline	0.8595	0.8593	0.8561	0.8567
Variance Threshold	0.828	0.829	0.8229	0.8246
Improvement %	-3.66%	-3.53%	-3.88%	-3.75%
Univariate Feature	0.7652	0.7815	0.7632	0.7592
Improvement %	-10.97%	-9.05%	-10.85%	-11.38%
Select From Model	0.8419	0.8391	0.8366	0.8372
Improvement %	-2.05%	-2.35%	-2.28%	-2.28%

Some insights on the interpretation of the previous results can be found in section 4.3.1 Improving Classification

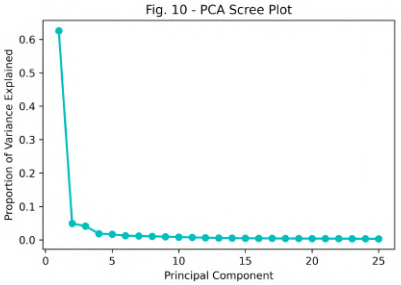
4.2 Feature projection

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.

4.2.1 Principal Component Analysis (PCA)

Principal Component Analysis 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. 10) to visualize the Proportion of Explained Variance PEV in each n component. We optimized n_components as in 4.1.2 and we stayed with 22 components, giving us an important reduction of features with only 5.72% loss of accuracy, a trade-off well worth it to reduce dimensionality.

Fig. 10 - 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 27, with worse results in accuracy and more dimensions. This is perhaps due to the non-Gaussian nature of our data. More information can be found in table 4.

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 needed 112 components to optimize for classification.

4.2.4 t-SNE

T-SNE is a non-linear dimensionality reduction algorithm where similar objects are assigned a higher probability and dissimilar objects are assigned a lower probability, it then reduces the dimensionality of the original data defining a similar probability distribution of the points in a low dimensional space. It is specially useful for visualization, while performing poorly in other tasks, therefore it is not included in the classification evaluation.

Table 4 - Performance comparison between methods of Feature Projection

	Accuracy	Precision	Recall	F1-score
DT Baseline	0.8595	0.8593	0.8561	0.8567
PCA	0.8103	0.8103	0.805	0.8057
Improvement %	-5.72%	-5.70%	-5.97%	-5.95%
Gaussian Random Projection	0.7251	0.7228	0.7188	0.7197
Improvement %	-15.64%	-15.89%	-16.04%	-15.99%
ISOmap	0.7652	0.7815	0.7632	0.7592
Improvement %	-10.97%	-9.05%	-10.85%	-11.38%

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 the best 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 chapter, 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 Classification

Seeing the results of section 4.1 and 4.2 we arrived at the conclusion that neither Feature Selection nor Projection improve the classification performance, at least with Decision Tree. This could be caused by underfitting the model, especially when projecting the data with too few components, or in the case of Feature Selection, where even removing the features with a variance threshold of .01, i.e. features with almost the same values in all samples, led to a decrease in accuracy in the model. This suggests that most features, even if highly correlated, contribute in some way or another to classifying the target value, resulting in information loss when removing them.

To overcome this, a combinational approach was devised, first selecting the best features from the importance threshold of a given estimator (logistic regression), which showed the least amount of information loss, and then using this 187 features to project 17 components using PCA, this results in the best possible classification with Decision Tree (outside of using the original data with all features). Additionally, we thought of using this projected dataset to train a model which would benefit from having a high variance set of features, like Gaussian Naive Bayes, and as expected. its performance greatly improved. These results can be seen in table 5.

Table 5 - Improving Classification with Dimensionality Reduction

	Accuracy	Precision	Recall	F1-score
PCA	0.8103	0.8103	0.805	0.8057
PCA using selected features	0.8568	0.8574	0.8527	0.8537
Improvement %	5.74%	5.81%	5.93%	5.96%
Gaussian Naive Bayes	0.7703	0.7925	0.7691	0.7672
Gaussian NB with PCA	0.8704	0.8786	0.8653	0.8661
Improvement %	12.99%	10.86%	12.51%	12.89%

Here we can finally start to see some improvement, proving that it is of the utmost importance to fully understand our algorithms and identify the transformations that would provide the biggest benefit depending on the nature of our data at hand, and realizing that some steps might not actually benefit the project depending on what we are looking to achieve.

4.3.2 Visualization

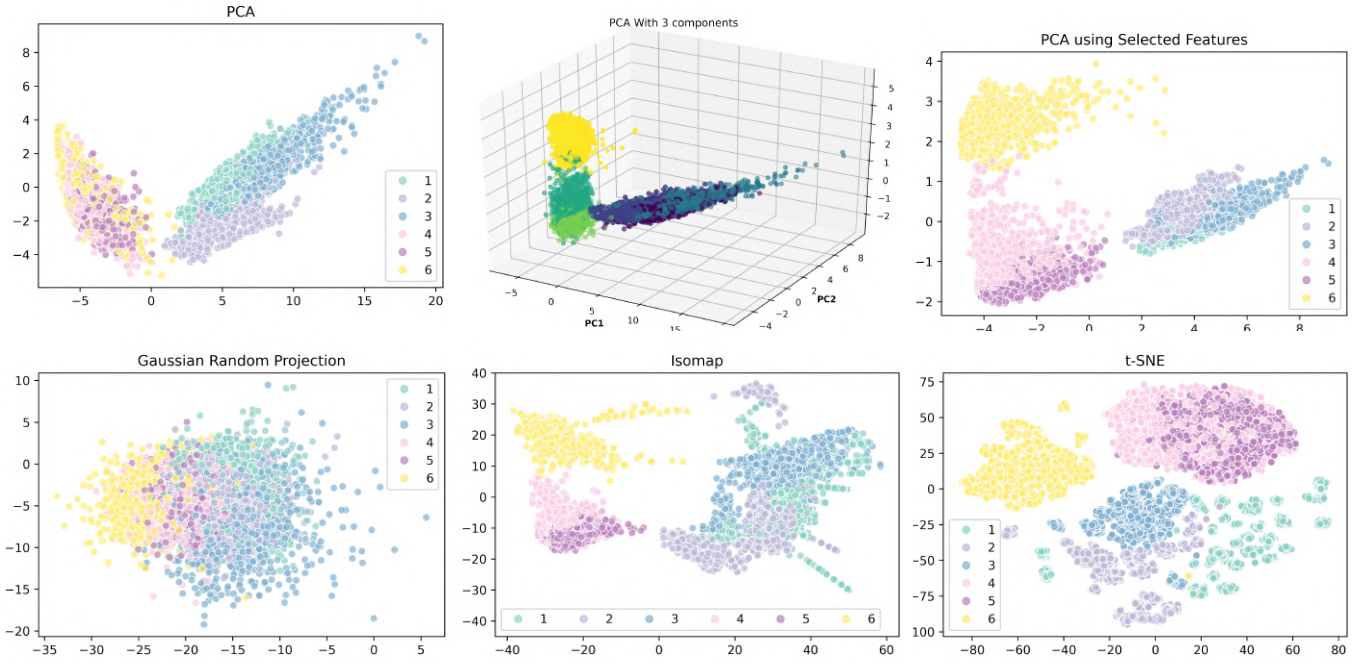


Figure 11 - Low-dimensional visualization of Feature Projection Methods

Setting the number of components to 2 or 3, we can get a low-dimensional scatterplot that allows us to visualize the projection of our data. Confirming what has been said in section 4.2, ISOmap and t-SNE provide very clear results and make class differentiation easy. Gaussian Random Projection performs badly. Another interesting thing to note is that when applying PCA to the selected features, we get a very similar plot to the original PCA but with greater class separation and less density.

4.3.3 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 section 2 with the added benefit of a scatter plot (seen in figure 12) 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.

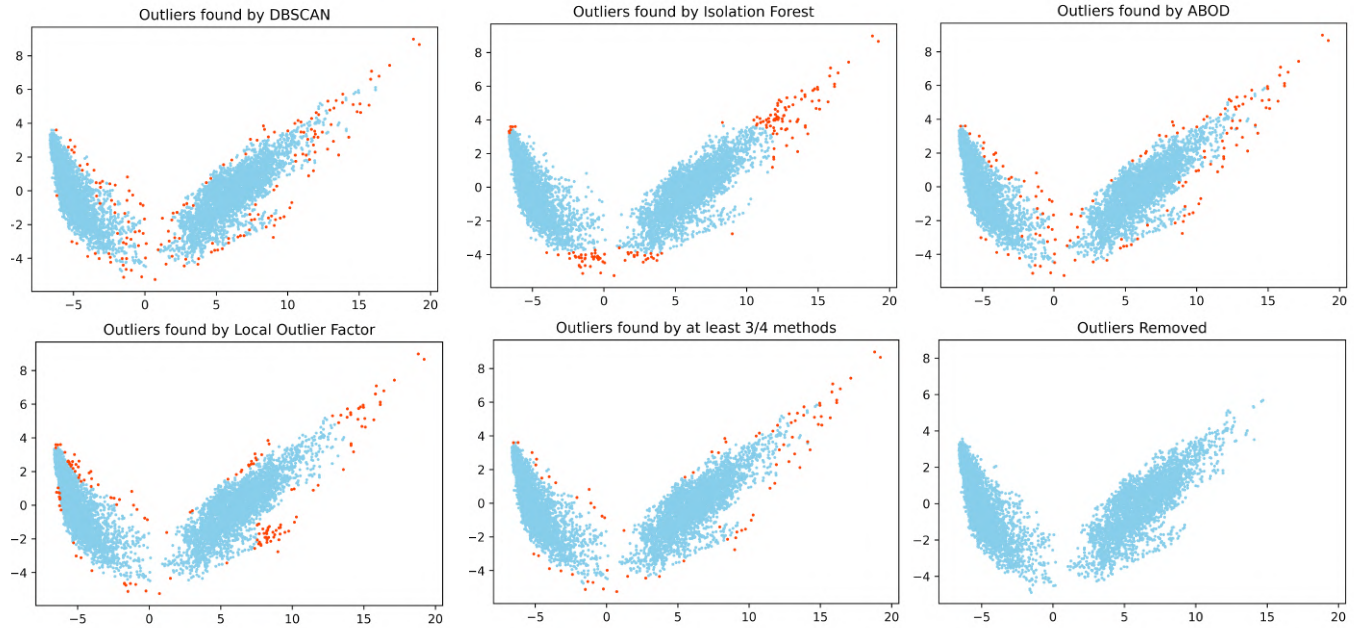


Figure 12 - Outlier identification with a low-dimensional approach (PCA)

As in section 2, we also replaced the identified outliers with the mean of each variable. The comparison in performance can be seen in table 6.

Table 6 - Dealing with Outliers Performance Improvement by using Dimensionality Reduction

	Accuracy	Precision	Recall	F1-score
Removing	0.8558	0.8549	0.8516	0.8521
Removing with DR	0.8595	0.8586	0.8559	0.8564
Improvement %	0.43%	0.43%	0.50%	0.50%
Transforming	0.8585	0.8574	0.8546	0.8548
Transforming with DR	0.8493	0.8462	0.8444	0.8447
Improvement %	-1.07%	-1.31%	-1.19%	-1.18%

5 Advanced Classification

For this section, several different classification algorithms were implemented, first as a base model and then optimized by using the hyperparameters found during the hypertuning phase. This phase consisted of combining the capabilities of both RandomSearch and GridSearch to get the best possible hyperparameters while maintaining relatively low computational costs. This is achieved by going with an inferential approach, that is, a coarse-to-fine tuning, in which we first ran a RandomSearch, which samples an n number of iterations through a given range to roughly test a large number of hyperparameters in a smaller time than Gridsearch, it of course doesn't find the optimal hyperparameters, but gives us a direction in which to dig deeper with Gridsearch, which can now exhaustively test every possible combination within a smaller range, providing us with a trade-off with performance and computational efficiency. 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, Confusion Matrix, AUC-ROC Curve and Precision-Recall Curve. At the end of this section, in figure 16 we can find a summary of ROC and PR curves for each model. In addition, table 7 provides a comparison of the evaluation metrics for each model.

5.1 Naïve Bayes

Naïve Bayes is a probabilistic machine learning algorithm based on the theorem formulated by Thomas Bayes which describes the probability of an event, based on prior knowledge of conditions that might be related to said event. It is stated as follows: $P(A|B) = \frac{P(B|A)P(A)}{P(A)}$ and when applied to machine learning, the $P(A)$ hypothesis can be interpreted as the target variable y and $P(B)$ are the conditions that provide prior knowledge, hereby identified as the dataset features, or variables, X.

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, instead, as it has already been proved before, [1],[2], the classifier still performs relatively well, even when the independence assumption is violated, this is due to the fact that even if the probability estimates are off by a wide margin, the probability ranking still holds, therefore resulting in a mostly correct classification.

It is used in a wide variety of classification tasks, although it excels in Natural Processing Language ones, with related scopes like sentiment analysis and recommendation systems, and depending on the goal and the data at hand, different approaches will be used:

5.1.1 Gaussian Naïve Bayes

The gaussian variant is used when the predictors take up a continuous value and are not discrete, since a typical assumption is that each class' continuous values are distributed according to a normal (Gaussian) distribution. Appearing as an attractive fit for our purposes, this is the first approach tested. The only hyperparameter to take into account is var_smoothing, which is the portion of the largest variance of all features that is added to variances for calculation stability. For our hypertuning it was set to 0.0001.

5.1.2 Bernoulli Naïve Bayes

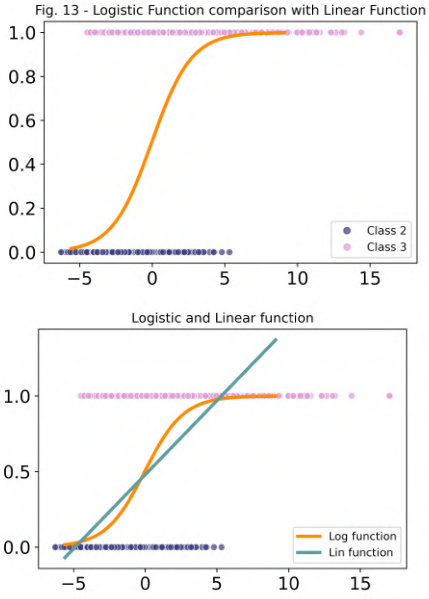
The Bernoulli approach implements Naïve Bayes for data that is distributed according to multivariate Bernoulli distributions, in which the features are assumed to be a boolean variable, thus requiring binary-valued feature vectors. Since our data is continuous, it was decided to binarize the input with the help of the classifier itself, with the help of the *binarize* parameter, which maps the input into boolean values.

This classifier as a base model performed better than the Gaussian one, confirming that the distributions of the variables *X* are not strictly normal, suffer from a high multicollinearity and adapt better to a binary approach. Still, as we will shortly see, due to the need for independence between variables, there are much better alternatives for this particular dataset.

5.2 Logistic Regression

In statistics, the logit model, also known as 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". Additionally, to be able to visualize the sigmoid logistic function which fits certain data much better than a linear regression function, we came back to the reduced version of our data which only contains target value 2 and 3, projected a principal component of it and plotted it along with the logistic and linear function.

Fig. 13 - Logistic Function comparison with Linear Function



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. The confusion matrix showed us a high misclassification especially in class 4, misclassified 57 times as class 5. Improving the model by tuning the hyperparameters to *C=1 (*which is usually 1 by default and is a regularization parameter), *dual=False (*which is preferable when the number of samples is greater than the number of features and is usually set to *True* by default) and *fit_intercept=False* (meaning that no intercept will be used in calculations) gave us a slightly worst result regarding the misclassification of class 4 rising the value from 57 to 58, but it also improved the classification of the other classes.

The AUC-ROC Curve of the Linear SVC appears to be very good with a sensitivity and specificity equal to 1, meaning an optimal true positive and true negative rate for all the classes except class 3 whose curve is a bit lower than the others.

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 classification of class 4 results to be better than the one of the LinearSVC. This is probably because LinearSVC works really well when the classification task involves just two classes that are usually well separated. In this case our dataset contains six different classes so they are more likely to not be separated that well.

5.4 Artificial Neural Networks

Artificial Neural Networks (ANN) are groups of interconnected nodes that draw inspiration from the human brain, loosely functioning as neurons. They receive an input signal X in the form of a vector and process it to produce an output y.

A perceptron is a basic NN consisting of an input layer, output layer and a single hidden layer that performs the transforming of the signal to arrive at a certain output in the form of a binary value $f(x)$. This can be implemented into a binary classifier that trains the neurons by forming probability weighted associations between the feature vector and previously known values of the target variable. This algorithm however can only make predictions based on a linear function, so in order to extend its capabilities, the Multi-Layer Perceptron comes into play.

5.4.1 Multi-Layer Perceptron Classifier

The Multi-Layer Perceptron (MLP), while having multiple hidden layers as the name implies, also employs a nonlinear activation function, making it possible to distinguish data that is not linearly separable. This results in a more robust learning algorithm.

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. Each element in the tuple represents the number of nodes, thus the length of the tuple denotes the total number of hidden layers in the network.
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 535 number of neurons.

As seen from figure 14, 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 even better, with a smoother curve and a tighter fit for generalization.

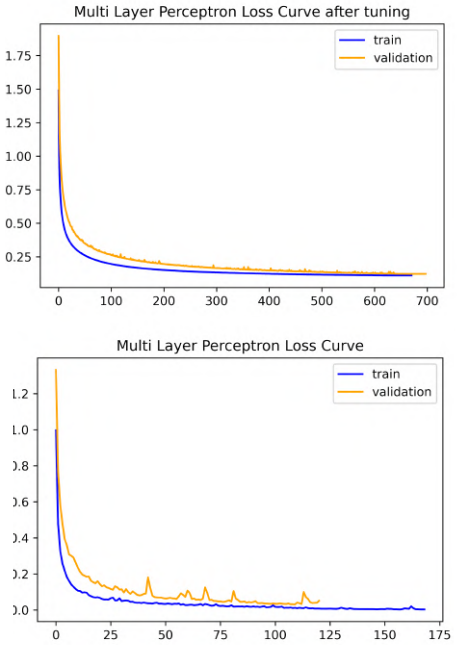


Figure 14 - MLP Loss Curves

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 15.

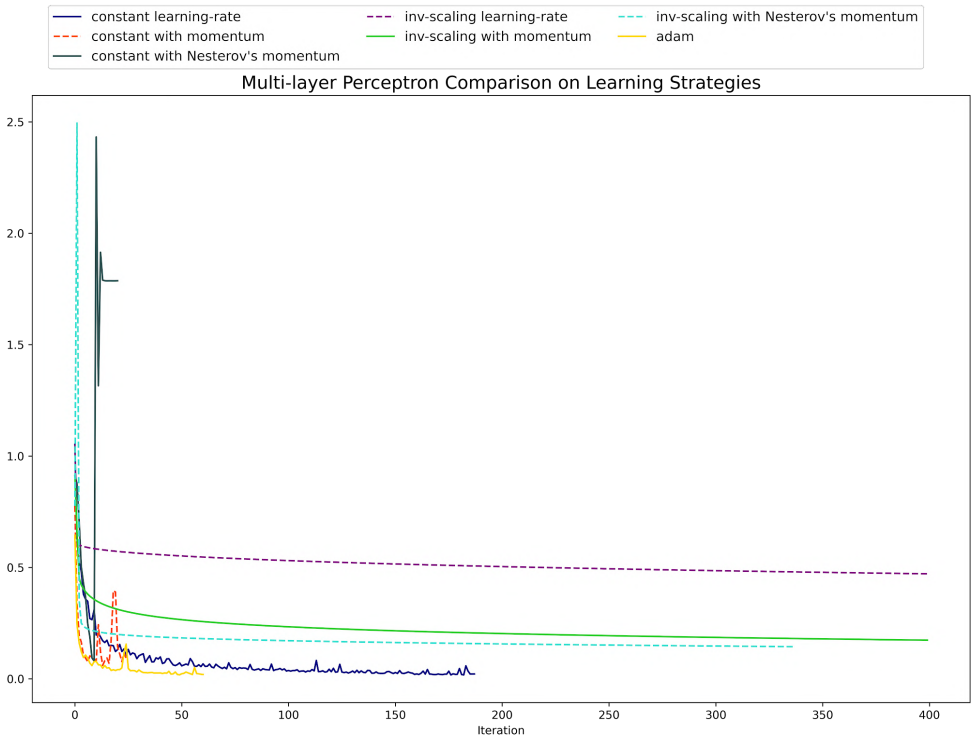


Figure 15 - Stochastic Learning Strategies comparison for MLP

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

In the Random Forest Classifier each classifier in the ensemble is a decision tree classifier and is generated using a random selection of attributes at each node to determine the split. During classification, each tree votes and the most popular class is returned. It 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 4, predicted 49 times as class 5. 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. In concept, it is very similar to a Random Forest Classifier and differs from it in the way the trees are constructed in the forest: Random forest uses bootstrap replicas (it subsamples the input data with replacement), whereas Extra Trees use the original sample. 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 more or less the same results as the Random Forest: the highest error is again obtained while predicting class 4, predicted, this time, 45 times as class 5.

The AUC-ROC Curve shows even better results than the ones obtained with Random Forest, showing an optimal curve (very close to 1) for almost each class except for 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 high misclassification of class 4 and 5, with 4 being predicted 90 times as class 5 and 5 predicted 69 times as class 4. Class 2 is predicted 74 times as class 1.

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 Gradient Boosting Classifier depends on a loss function (deviance by default) and each time a new weak learner is added to the model, the weights of the previous learners remain unchanged; contrary to AdaBoosting where the values are adjusted when new learners are added.

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.

The confusion matrix showed that class 4 remains the most misclassified, predicted 70 times as class 5. Tuning the hyperparameters, especially lowering down the number of max_depth from 6 (by default) to 2 actually helped with the XGBoost Classifier, the misclassification of class 4, the most significant one, improved with now 51 values predicted as class 5, against the previous 70.

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 16. The accuracy score can be seen in Fig 17.

Looking at the Precision-Recall Curve we can see that the best models, in terms of error measures, are LinearSVC and Logistic Regression. 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 Gaussian NB, as most models already performed pretty well since the beginning. Overall, LinearSVC was the model with the highest accuracy score, followed by Logistic Regression. It is still possible to see some improvements in the Non-Linear SVC and XGBoost curves, as well as a smoothing of the PR Curves of Bagging and KNN.

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 6 different classes. As previously noticed, also the Precision-Recall Curve of the logistic regression has its values very close to 1, given to the fact that the "one vs the rest" approach has been used to make a multi-class prediction. The results obtained indicate a cause-effect relation between the features and the target variables. Regarding the ROC Curve, the situation appears to be more or less the same: the LinearSVM and Logistic Regression have optimal curves with a sensitivity and specificity equal to 1, meaning an optimal true positive and true negative rate for all the classes. It's important to note that all the models showed a relatively high misclassification rate when predicting class 4, predicted many times as class 5, this is probably due to the fact that *Standing* and *Lying* are two activities that require the axes to be in the same position except for the Z axis.

Table 7 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.

For a better understanding, the tuned models used the following hyperparameters:

- DecisionTreeClassifier*(max_depth=10, max_leaf_nodes=50, min_samples_leaf=2)*
- KNeighborsClassifier*(algorithm='brute', n_neighbors=7, weights='distance')*
- GaussianNB*(var_smoothing=0.0001)*
- BernoulliNB*(alpha=0)*
- LogisticRegression*(penalty='l1', solver='liblinear')*
- LinearSVC*(C=1, dual=False, fit_intercept=False)*
- SVC*(C=5)*
- MLPClassifier*(alpha=0.05, hidden_layer_sizes=(535,), learning_rate='adaptive', max_iter=1000, solver='sgd')*
- RandomForestClassifier*(criterion='entropy', n_estimators=300, oob_score=True)*
- ExtraTreesClassifier*(n_estimators=300)*
- BaggingClassifier*(max_samples=0.5, n_estimators=300)*
- GradientBoostingClassifier*(n_estimators=300, random_state=0)*
- XGBClassifier*(base_score=0.5, learning_rate=0.25, max_depth=2, n_estimators=300, objective='multi:softprob', tree_method='exact')*

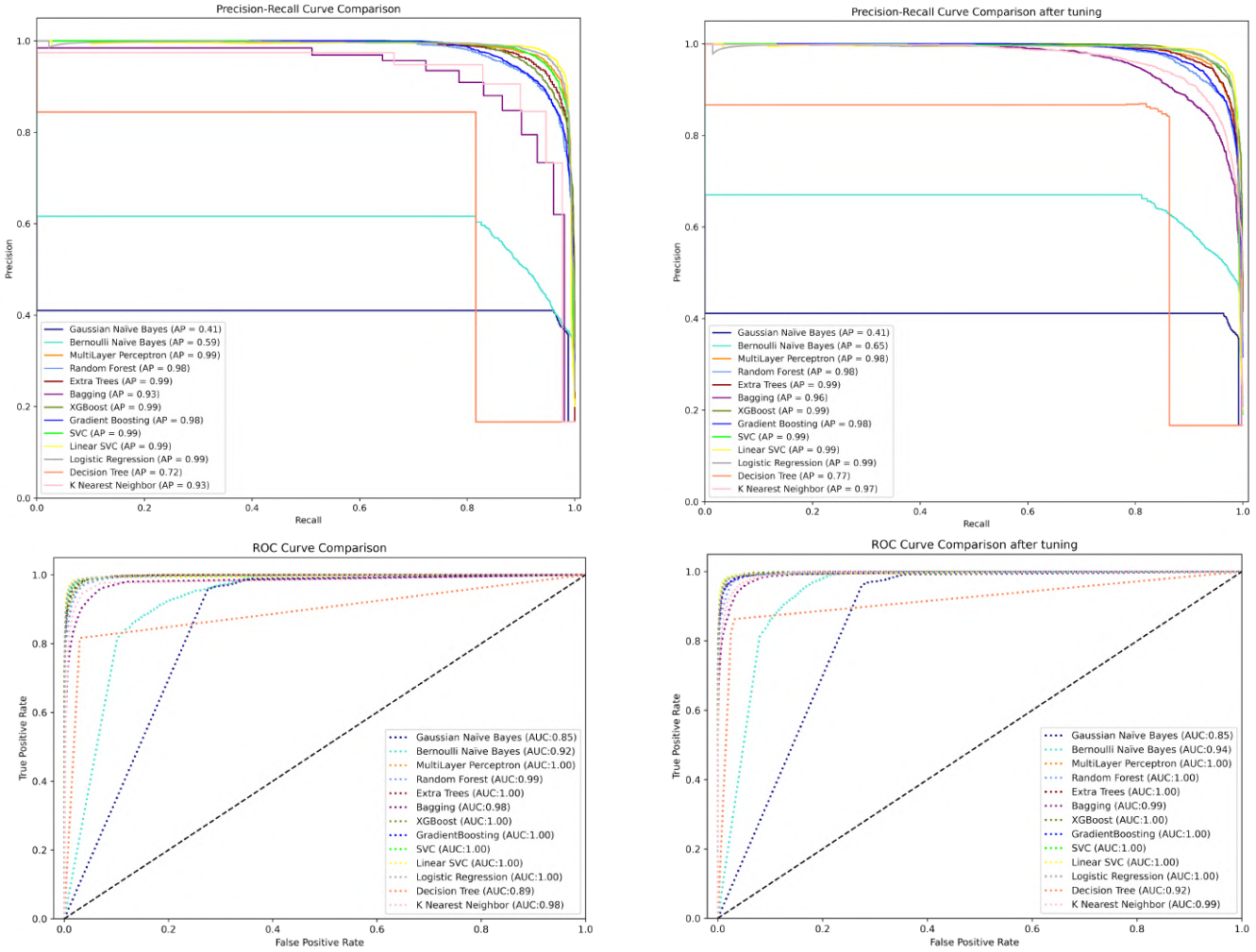


Figure 16 - AUC-ROC and Precision-Recall Curves before and after tuning

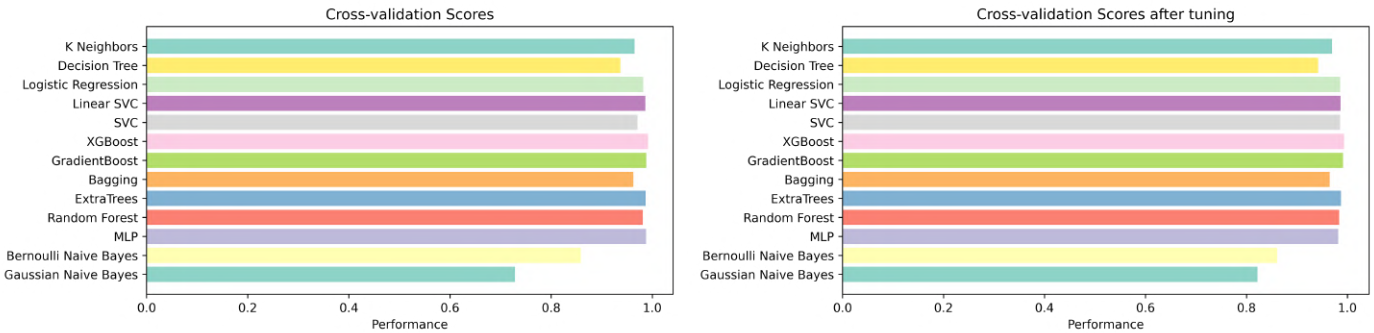


Figure 17 - Cross Validation Accuracy performance before and after tuning

Table 7 - Performance comparison between baseline and optimized models

	Accuracy	Precision	Recall	F1-score
Decision Tree	0.8595	0.8593	0.8561	0.8567
Tuned Decision Tree	0.8809	0.8829	0.8769	0.8776
Improvement %	2.49%	2.75%	2.43%	2.44%
K nearest neighbor	0.9032	0.9082	0.8986	0.9004
Tuned K nearest neighbor	0.9043	0.9082	0.8998	0.9014
Improvement %	0.12%	0.00%	0.13%	0.11%
Gaussian Naïve Bayes	0.7702	0.7925	0.7691	0.7672
Tuned Gaussian NB	0.8198	0.8423	0.8111	0.8064
Improvement %	6.44%	6.28%	5.46%	5.11%
Bernoulli Naïve Bayes	0.8492	0.8539	0.8409	0.8409
Tuned Bernoulli NB	0.8523	0.8558	0.8434	0.8434
Improvement %	0.37%	0.22%	0.30%	0.30%
Logistic Regression	0.9579	0.9593	0.9571	0.9578
Tuned Logistic Regression	0.9623	0.9651	0.9617	0.9625
Improvement %	0.46%	0.60%	0.48%	0.49%
Linear SVC	0.9649	0.9692	0.966	0.9667
Tuned Linear SVC	0.9661	0.9688	0.9657	0.9664
Improvement %	0.12%	-0.04%	-0.03%	-0.03%
Nonlinear SVC	0.9443	0.9458	0.9427	0.9437
Tuned Nonlinear SVC	0.9548	0.9566	0.9533	0.9544
Improvement %	1.11%	1.14%	1.12%	1.13%
MLP	0.9437	0.9473	0.9424	0.9438
Tuned MLP	0.9586	0.9602	0.9573	0.9581
Improvement %	1.58%	1.36%	1.58%	1.52%
Random Forest	0.9219	0.9284	0.9237	0.9251
Tuned Random Forest	0.9267	0.9283	0.9231	0.9246
Improvement %	0.52%	-0.01%	-0.06%	-0.05%
Extra Tress	0.9351	0.9381	0.9329	0.9341
Tuned Extra Tress	0.9429	0.9449	0.9391	0.9406
Improvement %	0.83%	0.72%	0.66%	0.70%
Bagging	0.8921	0.8874	0.8832	0.8843
Tuned Bagging	0.9019	0.9035	0.8991	0.9003
Improvement %	1.10%	1.81%	1.80%	1.81%
Gradient Boosting	0.9382	0.9399	0.9366	0.9376
Tuned Gradient Boosting	0.9412	0.9431	0.9398	0.9409
Improvement %	0.32%	0.34%	0.34%	0.35%
XGBoost	0.9389	0.9411	0.9371	0.9381
Tuned XGBoost	0.9558	0.9567	0.9559	0.9558
Improvement %	1.80%	1.66%	2.01%	1.89%

6 Linear Regression

6.1 Simple Linear Regression

Despite not being the perfect method for our dataset (continuous target variable is preferred), we decided to use Linear Regression as well. In the first approach (seen in figure 18) we used the two features with higher correlation to the target variable, according to several metrics such as ANOVA f-tests (in order to deal with continuous attributes), and as expected we found optimal values of R^2, MSE and MAE which are clearly not extremely satisfying in terms of target class explanation for our case. Therefore, we decided to go further by performing a linear regression between the most correlated variable and a randomly selected one, in order to obtain more potentially useful results. In fact, in this case we obtained very bad results, with a coefficient of determination equal to 0.178. All the results can be seen in table 8 below.

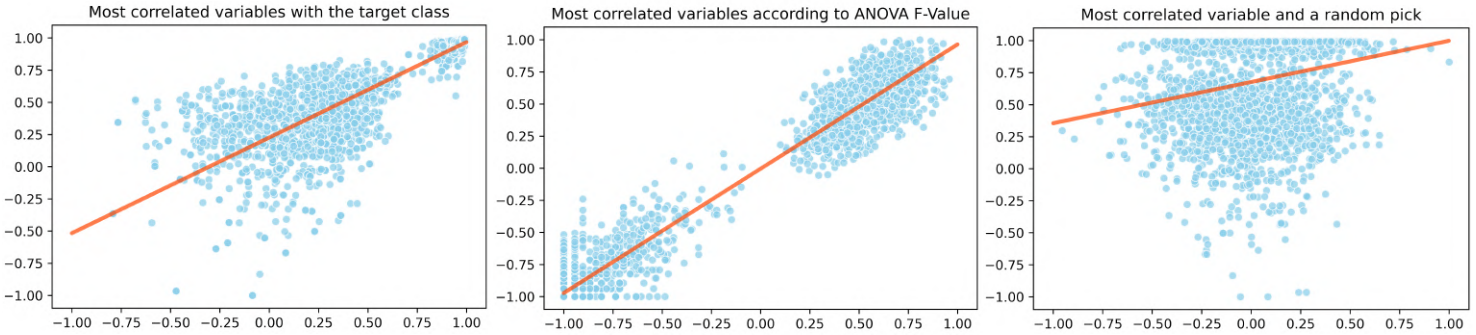


Figure 18 - Simple Linear Regression on several continuous values

6.2 Multiple Linear Regression

For this reason, we decided to use a multiple linear regression. To solve the problem of multicollinearity we used Variance Inflation Factor (VIF) which is a measure of the amount of multicollinearity in a set of multiple regression variables. Mathematically, the VIF for a regression model variable is equal to the ratio of the overall model variance to the variance of a model that includes only that single independent variable.

Unfortunately, the degree of multicollinearity is particularly high in our dataset, having 561 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^2 cannot be taken into 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 38 features.

6.2.1 Regular Linear Regression Method

Subsequently, we decided to experiment with regularized regressions. This is a form of regression that constrains (or shrinks) the coefficient estimates towards zero. In other words, this technique discourages learning a more complex or flexible model, aiming to avoid the risk of overfitting. Two commonly used types of regularized regression methods are Ridge regression and Lasso regression.

6.2.2 Ridge Linear Regression Method

Ridge regression is a way to create a parsimonious model when the number of predictor variables in a set exceeds the number of observations, or when a data set has multicollinearity. It tends to give small but well distributed weights, because the L2 regularization cares more about driving big weights to small weights, instead of driving small weights to zeros.In this case, considering that Ridge performs better when the number of predictors is large, we decided to use the dataset in its original form rather than the VIF reduction. Nevertheless, we would like to emphasize that R^2 remains a biased coefficient measurement.

6.2.3 Least Absolute Shrinkage and Selection Operator Method

Least Absolute Shrinkage and Selection Operator Method or simply 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 8 - Linear Regression Methods Results Comparison

	R2	MSE	MAE
SLR Most correlated features	0.999	0.001	0.022
SLR Most correlated feature and random	0.914	0.011	0.078
MLR Regularized	0.604	1.198	0.87
MLR Ridge	0.948	0.157	0.286
MLR Lasso	0.668	1.007	0.885
MLR Gradient Boosting	0.817	1.553	0.571

7 Time Series Analysis

Time series analysis is the analysis of a sequence of data points collected over a specific interval of time. Because our data originated from 9 different signals, we decided to use *total_acc_x_train* since it has a greater variance than the rest, meaning they could be more informative. The data contains 7352 time series with 128 time windows which have been transposed on the X axis for a better visualization.

7.1 Transformations

Looking at the similarity of the shape in different time series, it is possible to calculate how dissimilar (distant) they are from each other. To analyze this distance we proceeded to pick and plot two random values from *total_acc_x_train*, namely the 2153th and 4671th in order to have a first look of their shape, we then calculated their dissimilarity using both the Euclidean and Manhattan distance metrics.

As shown in figure 19 the time series appear to be very different from one another with an Euclidean distance equal to 2.1232 and a Manhattan distance equal to 20.0006.

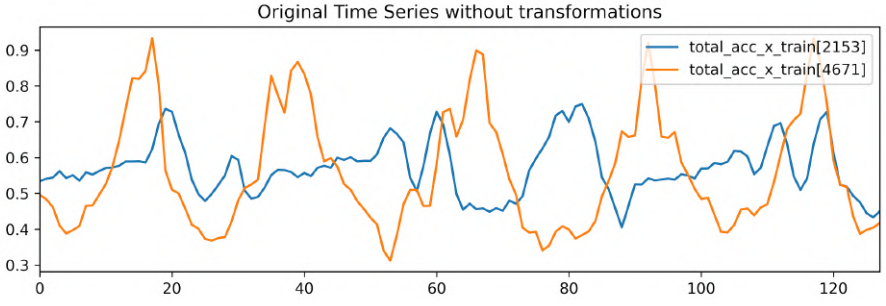


Figure 19 - Raw time series without transformations

7.1.2 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 20), with an Euclidean distance reduced to 2.1193 and a Manhattan distance reduced to 19.8389.

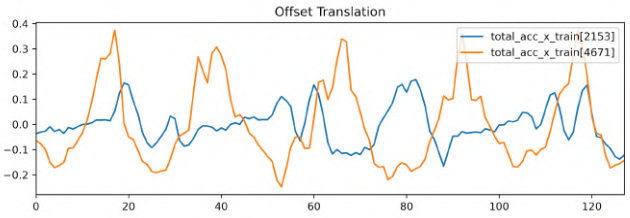


Figure 20 - Offset Translation

7.1.3 Amplitude Scaling

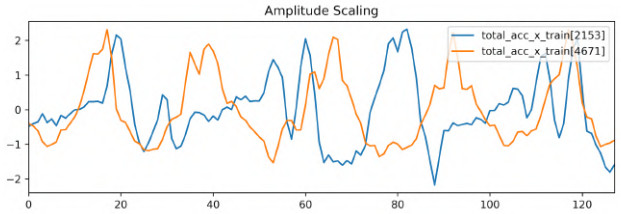


Figure 21 - Amplitude Scaling

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 21). Doing so the Euclidean distance value raised to 16.9895 and the Manhattan distance to 154.5826.

7.1.4 Linear Trend Removal

Linear 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 3 subtracting each time its mean from the original value of the time series. As shown in figure 22, by removing the linear trend, the time series move in a straight direction, closer to one another. Removing the linear trend from the time series resulted in an Euclidean distance equal to 0.7707 and a Manhattan distance equal to 6.8805.

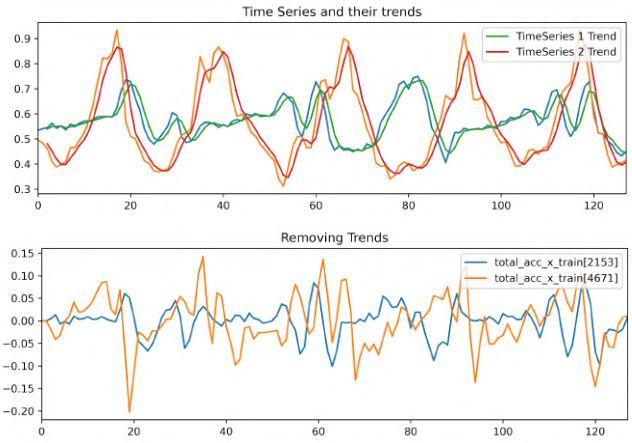


Figure 22 - Linear Trend Removal

7.1.5 Noise Removal

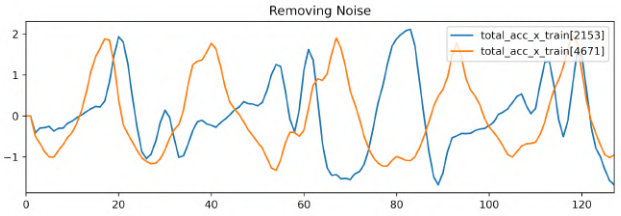


Figure 23 - Noise Removal

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 23), and an Euclidean distance equal to 16.0149 and a Manhattan distance equal to 145.5634.

7.1.6 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 24), with an Euclidean distance equal to 0.3192 and a Manhattan distance of 2.6059, way less than the ones found at the beginning.

Figure 24 - Combined Transformations

7.2 Dynamic Time Warping

Dynamic Time Warping is very useful when 2 time series are conceptually equivalent but evolve at different speeds. As previously seen, when combining all the different transformations, our time series appeared to be almost, but not perfectly, aligned. Using the Euclidean distance as a similarity measure can cause misalignment in data 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.

7.2.1 Computing the cost Matrix

In order to find the best alignment, meaning the path in the cost Matrix with the lowest value, we computed the point-to-point and cumulative cost Matrix. The goal is to find the optimal warping path that has minimal total cost among all possible warping paths between the 2 time series. For this purpose we selected another 2 random time series: 3574 and 7345, while using a time-span equal to 10, for a better visualization. First we computed the point-to-point cost Matrix to check the local distances between the points of the first and second time series, then we used the cumulative cost Matrix to visualize the accumulated cost of any warping path (figure 25).

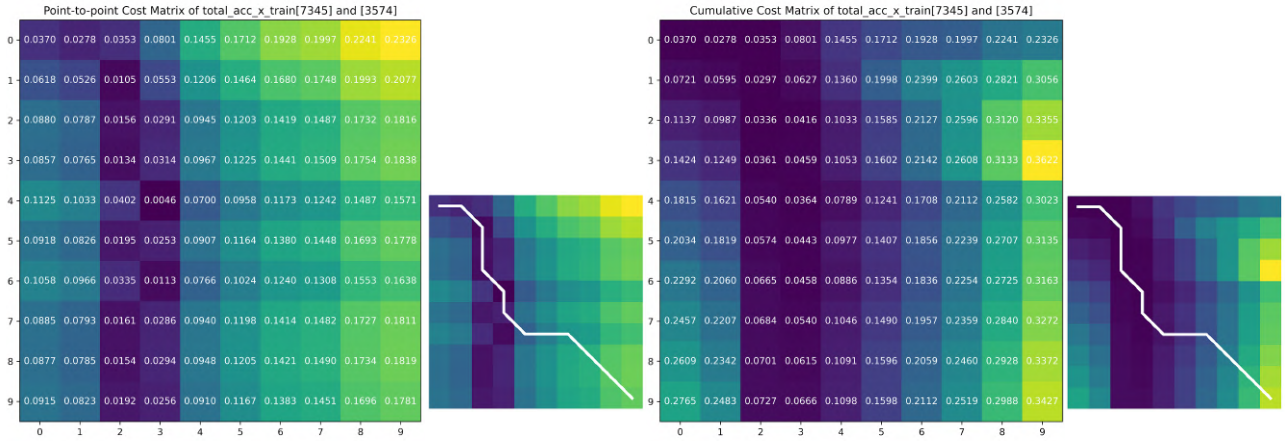


Figure 25 - Point-to-point Cost Matrix and Cumulative Cost Matrix with their respective paths

Knowing the different costs it is possible to find the optimal warping path among all the possible ones, as shown in figure 26: the white line represents the optimal path drawn over the point-to-point cost Matrix using all the 128 time-span. The distance between the 2 time series using the Dynamic Time Warping is equal to 0.6340.

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 2, which is the number of off-diagonal elements to consider, on both types of cost Matrix which result in a distance equal to 1.6433 with a straighter line than the previous one: this is because the constraint forces the path to stay very close to the diagonal, way more than the Itakura constraint. 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 0.7246. In figure 27 we can appreciate how the different bands for the Sakoe-Chiba and coefficients for Itakura constraint the path. We used band = 10,20,40 and coefficient = 0.5,1.0,5.0.

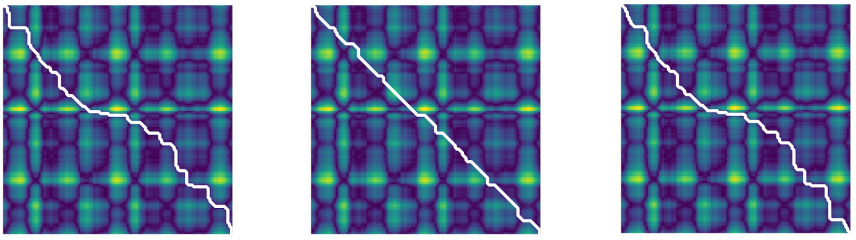


Figure 26 - Complete optimal DTW path without constraints (left), Sakoe-Chiba (center) and Itakura (right)

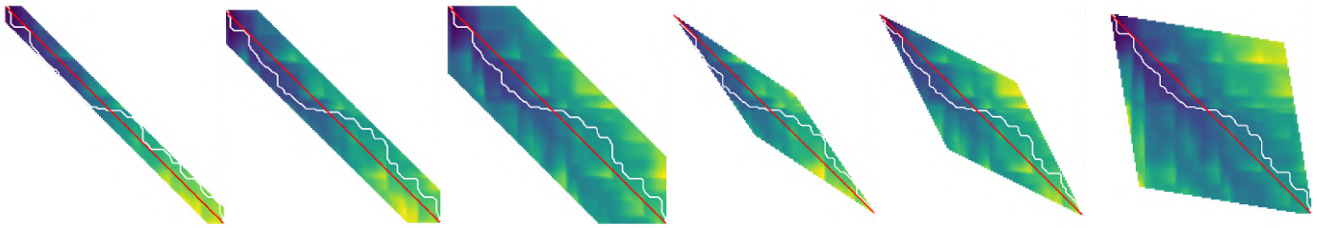


Figure 27 - Visualization of global constraints using band and parallelogram approaches

In addition to the path itself, it is also possible to visualize the warping of points (i,j) for the time series |Q| and |C|, by using the path previously generated inside the cost matrix, starting from points (0,0) and observing if either i or j increase by one or both of them do, in the first case, when only one time series move we have a warping which is the same as the path, in the second, when both time series move, we have a linear euclidean-like distance, which would be equivalent to having a straight line in the matrix path. In figure 28 we can appreciate the warping of points plotted in yellow and otherwise in red.

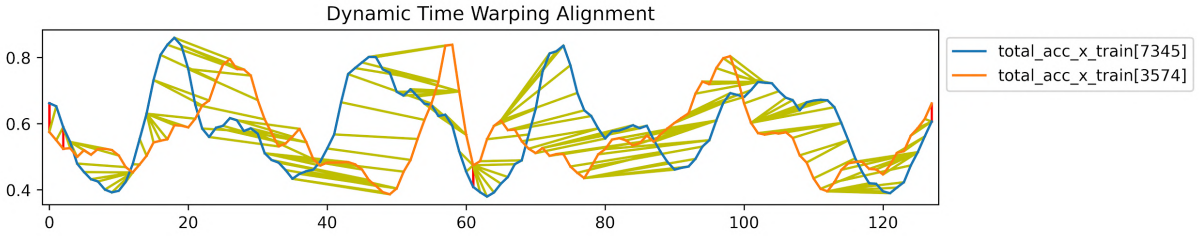


Figure 28 - Warping of points (i,j) between time series |Q| and |C|

Naturally, global constraints also affect the warping of points. Since we previously established that a straight diagonal line in the matrix path where both i and j increase at the same rate is equivalent to linear euclidean-like distances, then if we decrease the value of the band or coefficient, we should see a gradual approaching to vertical lines in the warping alignment plot, which is exactly the case as shown in figure 29, where setting the value of the coefficient to 0 actually eliminates any warping.

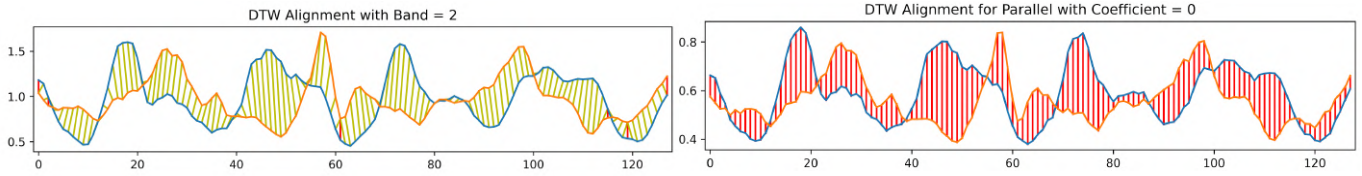


Figure 29 - Effect of global constraints on the DTW warping alignment

In order to find the optimal value of the parameters for the global constraints, we simply have to plot the DTW values depending on the value of the parameters along with the values for the case in which there are no constraints and find the intersection, i.e. the point in which further constraining gives no more positive results. (figure 30).

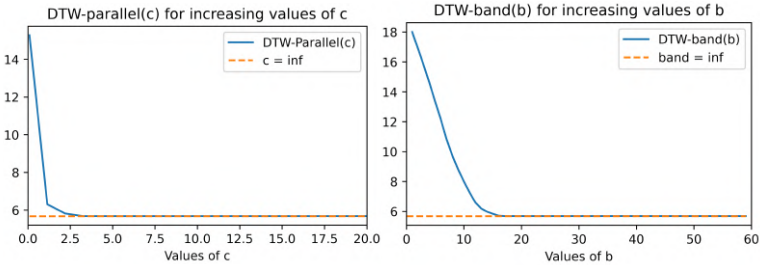


Figure 30 - Optimal value of parameter for global constraints Sakoe-Chiba and Itakura

7.3 Approximation

Approximation is a form of dimensionality reduction designed for time series analysis, it differs from Compression by creating a new representation of the time series into a smaller and simpler space which is always understandable. This new representation helps us by keeping the most importants information of the time series while removing the 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 6927th time series of *total_acc_x_train*.

7.3.1 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 16 we obtained the time series visible in figure 31. Between all the different Approximation methods we will be using in this chapter, the DFT creates a wavy line, since it derives a frequency-domain representation of the signal.

7.3.2 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 10 we plotted the approximated time series below the original one (figure 32). The new representation appears to be way more smooth than the original one.

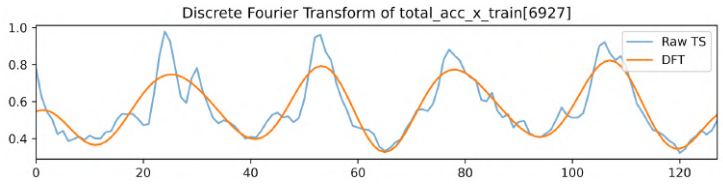


Figure 31 - Discrete Fourier Transform

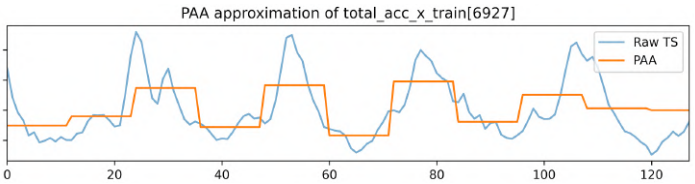


Figure 32 - PAA Approximation

7.3.3 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 8 symbols. Plotting 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. In addition we also explored 1d-SAX, which is based on the quantization of the linear regression of segments of the time series; they can both be seen in figure 33.

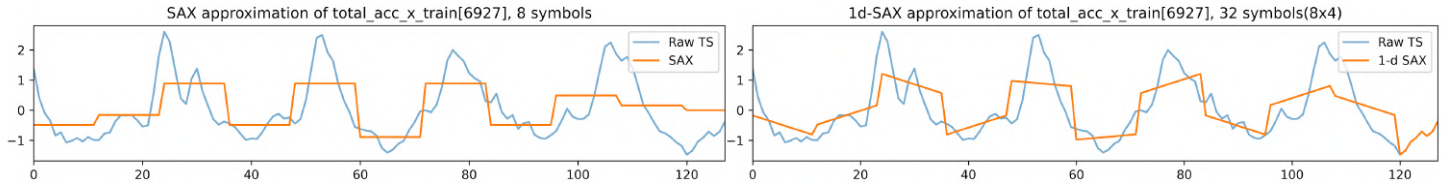


Figure 33 - SAX and 1d-SAX Approximations

7.4 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. As for data, we first used the raw shape of the time series contained in the signal *body_gyro_y_train*, then the set of features of said data and finally an approximated version of it. The raw shape of the time series can be seen in figure 34.

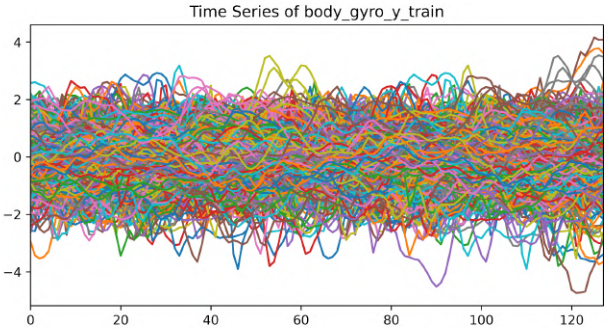


Figure 34 - Every time series within body_gyro_y_train

7.4.1 Shape-based Clustering

Shape-based clustering uses the raw time series as shown in fig. 34. We applied TimeSeriesKMeans and tried both euclidean and DTW distances. With the help of inertia values and silhouette scores we found the optimal K to be 4 in both cases. The found clusters can be seen in fig. 35.

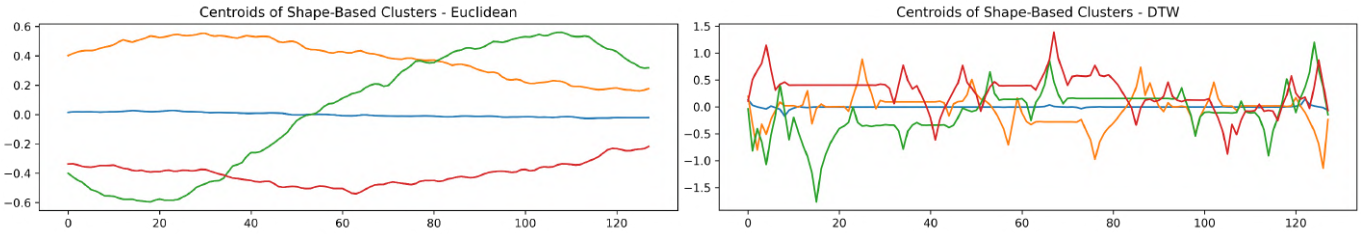


Figure 35 - Clusters found by KMeans in shape-based clustering using euclidean and DTW distances

7.4.2 Feature-based Clustering

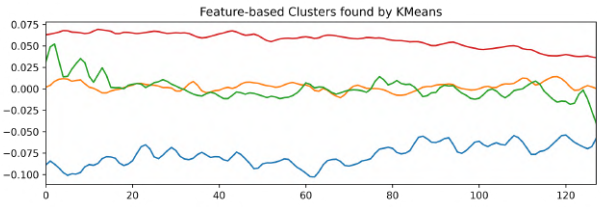


Figure 36 - Clusters found by KMeans in feature-based clustering

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 36 shows the clusters found by analyzing these features.

7.4.3 Approximate Clustering

As seen before, approximating our time series can be extremely beneficial for computational costs while maintaining a somewhat representative approximation of the original raw data. For this purpose we made use of PAA (since it best approximated our data) with 10 segments to represent our data. The approximated dataset can be visualized in figure 37.

Figure 37 - Approximated dataset with 10 segments using PAA

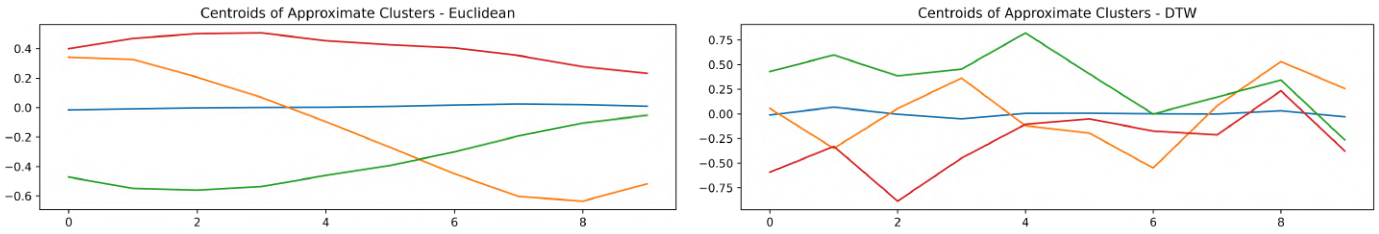


Figure 38 - Clusters found by KMeans in approximated dataset using euclidean and DTW distances

7.5 Shapelets and Motif 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.

For this particular task we decided to switch the signal to analyze to *body_gyro_y*, being the one with the 2nd highest variance. We performed a Shapelets Discovery using the 4637th time series of our dataset, Shapelets Discovery aims to represent patterns that are highly predictive for the target variable. We found 6 Shapelets of length 12 (figure 39).

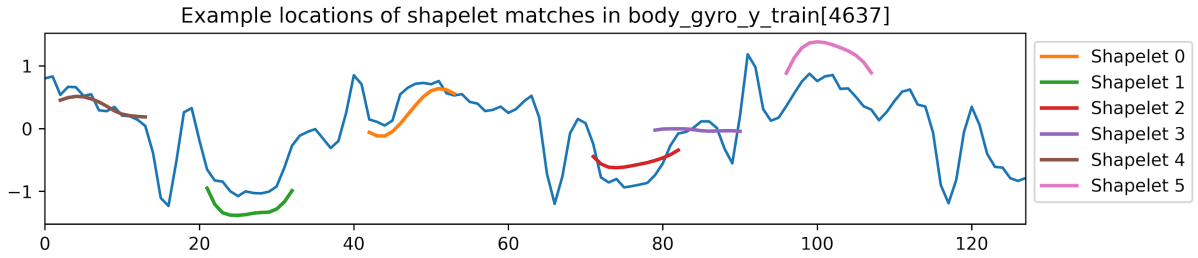


Figure 39 - Shapelet matches in sample time series of the dataset

Regarding motif analysis, we decided to check the target value of each time series to know which ones corresponded to the *Walking* and *Laying* classes respectively, since the two at least on a logical level, show greater differences in motion. After several attempts, we finally chose the 80th time series for the *Walking* class and the 55th time series for the *Laying* class of the *body_gyro_x_train* dataset, with *max_motif* equal to 3. In figure 40 shown below, we can see the motifs found by the Matrix Profile method which helped us during the identifying and visualizing phases. Through color matching, we tried to compare the motifs of the two time series that were most similar with the intention of finding micro-movements that were similar, despite the fact that the actions in terms of motion reproduced are totally different. As we can see, the purple and green motifs show similarities, unlike the orange ones. Regarding the anomalies in figure 41, we can clearly see motions that are not found in any motif.

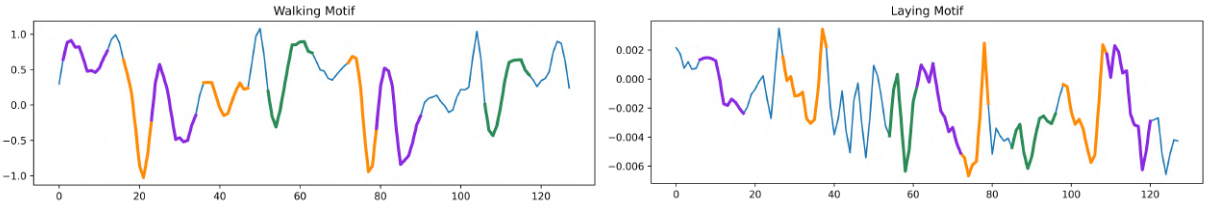


Figure 40 - Comparison between walking and laying motions and their motifs

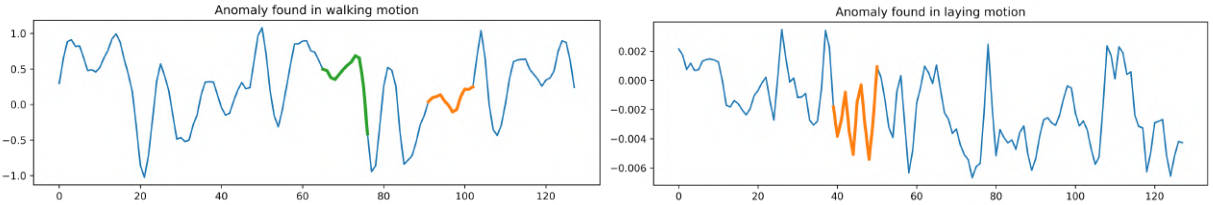


Figure 41 - Comparison between walking and laying motions and their anomalies

7.6 Time Series Classification

As before, we turn back to *total_acc_x_train* to be set as our train data and the one contained in *total_acc_x_test* as our test data in order to perform classification on our time series.

First of all we ran the Decision Tree and KNN algorithms on our raw time series dataset to have a first view of the classification results, then doing the same with shapelet-based and feature-based approaches. Overall KNN performed better than DT. In addition to these approaches we also tested LSTM and CNN neural networks, which performed way below the other algorithms, for which they were not further analyzed.

7.6.1 Shapelet-Based Classification

To perform a Shapelet-Based Classification we used the *ShapeletModel* algorithm to transform both our train and test data into 6 Shapelets, or vectors, representative of the 6 class labels. In order to evaluate the transformed data we used them as input both for KNN and Decision Tree algorithms: the models performed more or less the same in terms of accuracy, but it is still possible to establish that KNN algorithm performed better since the Decision Tree classifier has a very low F1-score value when predicting class 5.

7.6.2 Feature-Based Classification

Feature-Based Classification is a classification method that uses as train and test data their features values, such as the mean, the standard deviation, the percentiles values, etc. This way each time series is converted to a set of global features and similarity between time series and classification are based on high level structure and not on time series' shape. Also in this case, KNN performed slightly better than Decision Tree on the dataset, both having a low F1-score value when predicting class 5.

7.6.3 Hypertuning and Performance Comparison with Time Series

After finding that the raw time series data performed the best with the base models, we decided to take it further and apply more classification methods along with a hypertuning phase in order to achieve the best performance possible, this way we were able to achieve an accuracy score of .99 with XGBoost. The ROC curve shown in figure 42 shows the different results, which indicate that XGBoost and ExtraTrees algorithms have almost optimal curves with a sensitivity and specificity close to 1. Table 9 shows the performance comparison in terms of Accuracy between baseline and optimized models.
It's also important to note that the majority of the previously observed models showed the lowest F1-score value predicting class 5 when using time series data. This time, it is probably due to the fact that *class* 5 stands for the class label *Standing* which is an activity that does not involve movement over time, contrary to labels such as *Walking*.

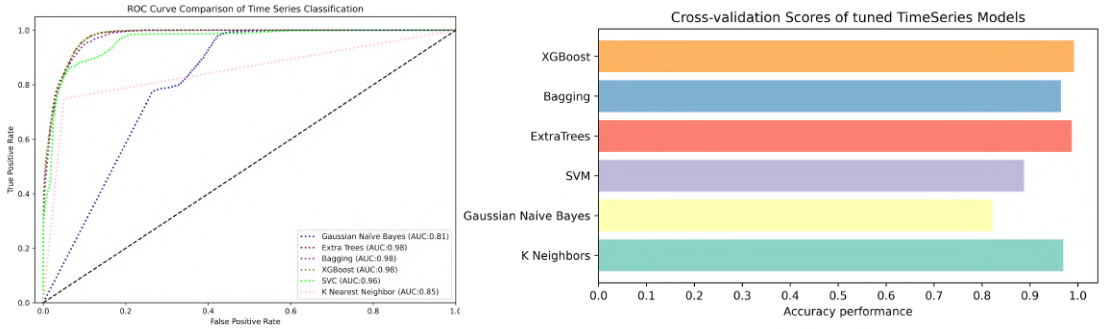


Figure 42 - ROC curve and cross-validation scores comparison between different classification methods

Table 9 - Performance comparison between classification methods for time series

	Base Accuracy	Tuned Accuracy	Improvement
Raw KNN	0.7448	0.9498	27.52%
Shapelet-based KNN	0.6694	-	-
Feature-based KNN	0.6396	-	-
Gaussian Naïve Bayes	0.7121	0.8219	15.42%
XGBoost	0.8347	0.9919	18.83%
Extra Tress	0.8431	0.9873	17.10%
Bagging	0.7842	0.9649	23.04%
SVC	0.8065	0.8876	10.06%

7.7 Conclusions

As confirmed by the previous tasks, through the analysis of time series we can obtain valuable information regarding the behavior of our data across a time dimension, this is particularly useful to extract patterns and get a clear picture of what has happened with our data within a certain timeframe. Another powerful tool in this area is dynamic time warping, which allows us to better understand the relationship between two time series that may not appear immediately similar, but can actually show some useful insights. Although computationally expensive, DTW can be complemented with the help of approximation, reducing the dimensionality of a time series while preserving its most important features. Regarding classification, even though we obtained optimal results, these were a lot harder to reach than with regular data, showing no apparent advantage over the latter.

8 Sequential Pattern Mining

In Sequential Pattern Mining the goal is to find all subsequences with support ≥ *minsup* within a given database of sequences. In our case the list of sequences used are those found in the *total_acc_x_train* signal, which consists of 7352 sequences with 128 elements. In order to analyze these sequences we transformed them into a discrete format using SAX transform with 12 segments. An example of the result would be the one found in sequence 42 of our signal; $S42$= {5, 4, 9, 7, 6, 5, 6, 4, 4, 8, 4, 6}.

8.1 Mining sequential patterns

For this task, we have focused only on those sequential patterns that have at least 3/4 of the length of the original sequence, that is, at least 9 elements. In table 10 we can look at the total amount of sequential patterns found when using varying degrees of lengths and minimum supports. From this we can gather that there are no subsequences to be found with a minimum support greater than 400.

Table 10 - Total sequential patterns found by varying values of MinSupport and Min Length

MinSupport/Min Length	≥ 9	≥ 10	≥ 11	≥ 12
10	270313	60629	7831	441
20	192373	38445	4378	196
40	114125	19516	1681	45
100	35355	2998	0	0
200	4793	0	0	0
300	337	0	0	0
400	0	0	0	0

Then in table 11 we extracted the 5 most frequent sequential patterns by descending order and by different lengths. An interesting thing to note here is how portions of the smaller subsequences tend to reappear within the more lengthy ones, these smaller sequences always show a greater support, essentially proving the *Apriori principle*, since $S1 \subseteq S2 \Rightarrow sup(S1) \geq sup(S2)$

Table 11 - Most frequent sequential patterns by different lengths

≥ 8		≥ 10		≥ 12	
Sup	*Pattern*	*Sup*	*Pattern*	*Sup*	*Pattern*
481	[4, 8, 7, 5, 6, 5, 4, 6]	182	[5, 4, 8, 7, 5, 6, 5, 8, 4, 6]	45	[5, 4, 8, 7, 7, 5, 6, 5, 5, 8, 4, 6]
477	[5, 4, 8, 7, 5, 6, 4, 6]	170	[5, 4, 8, 7, 7, 5, 6, 5, 4, 6]	33	[5, 4, 8, 7, 7, 5, 6, 4, 5, 9, 4, 6]
469	[5, 4, 8, 7, 5, 5, 4, 6]	146	[5, 4, 8, 7, 5, 6, 4, 8, 4, 6]	28	[5, 4, 8, 7, 7, 5, 6, 4, 5, 8, 4, 6]
419	[5, 4, 8, 7, 6, 5, 4, 6]	144	[5, 4, 8, 7, 7, 5, 6, 4, 4, 6]	24	[5, 4, 8, 7, 6, 5, 6, 5, 5, 9, 4, 6]
418	[5, 4, 8, 7, 5, 6, 5, 6]	139	[5, 4, 8, 7, 5, 6, 5, 5, 4, 6]	24	[5, 4, 8, 7, 7, 5, 6, 5, 5, 9, 4, 6]

9 Advanced Clustering

In order to explore the possibilities and capabilities of the clustering algorithms presented here, we experimented with different inputs. First of all, we used the raw training data previously available as X_train, getting poor results overall, with no algorithm being able to produce well-defined clusters. That is the reason why we decided to transform this input with a PCA with 2 components, benefiting especially X-means, which was able for the first time to obtain clear cluster shapes, also greatly increasing the silhouette score.

Then we finally tried a slightly different approach by reducing the dataset and only considering data from the *Walking* and *Lying* classes in an attempt to obtain more clear and defined clusters. However, we encountered the same problem, so we performed PCA again, this time on a dataset with a binary class. Aware of the fact that clustering is an unsupervised type of analysis, knowing that the dataset used is broken down into two completely different motor activities helped us to confirm that the clusters are well separated (and the best ones in terms of definition), decreasing the data-driven behavior of the PCA, making it more user-dependent, but above all, more interpretable. The clusters found with the first approaches can be seen in fig 45 and a comparison of the last approach in fig 46.

9.1 Mixture Gaussian Model

The Mixture Gaussian Model is an advanced clustering technique which assumes that all the data points in our dataset belong to a mixture of gaussian distributions with unknown parameters and computes the probability of data point belonging to a certain gaussian distribution in our dataset. As we remember from part 1.1, most of the features don't follow a normal distribution, still we wanted to see how this algorithm would work when presented with a mixture of distributions, as expected it had a mediocre performance, especially before reducing the dataset.

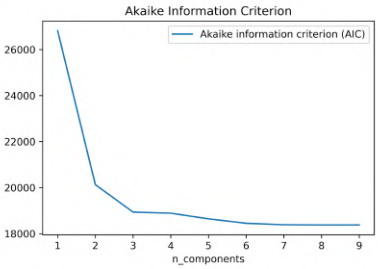


Figure 43 - AIC elbow method for reduced data after PCA

In order to get the optimal number of clusters to give as parameter, we used the Akaike Information Criterion AIC as a metric for goodness of fit for the model depending on the number of components and looked for the elbow. When applied to the raw data, the elbow could be found at 6 and when applied to the PCA at 2.

9.2 X-means

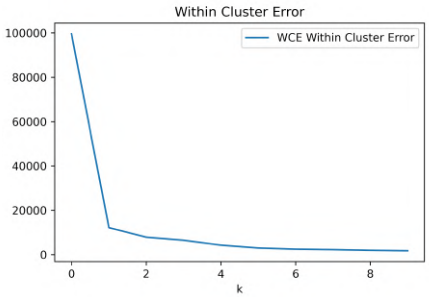


Figure 44 - WEC elbow method for reduced data after PCA

X-means is an extension of the K-means algorithm which uses the Bayesian Information Criterion to measure the improvement of a cluster when splitting it into children clusters in order to decide whether or not to keep the bisection. Once we plotted the Within Cluster Error to find the best number of clusters using the PCA input, we ran the X-means algorithm with a minimum and maximum number of clusters equal to 2. X-means appeared to work better than the K-means algorithm: the reason, as previously mentioned, is that the Bayesian Information Criterion represents a useful strategy to stop the algorithm when significant clusters are reached, avoiding overfitting and returning the best clustering configuration having the highest Bayesian Information Criterion.

9.3 OPTICS

The OPTICS algorithm is a DBSCAN evolution, also using the radius and MinPts as parameters, which can still be found using the same technique on our PCA input as that in figure 45. DBSCAN usually fails when working with different densities, while OPTICS solve the problem using as reachability distance between two points the maximum between the core distance of the first point and the real distance between the two. The points belonging to the same cluster will have a small reachability distance among them. Using a minimum number of points equal to 30 and a radius equal to 0.2 we were able to identify two different well separated clusters.

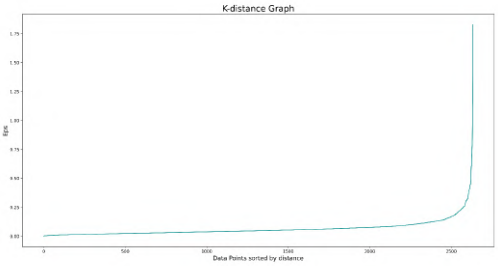


Figure 45 - Elbow method for reduced data after PCA

In table 12 we can see how the silhouette score changed for our different data and clustering techniques.

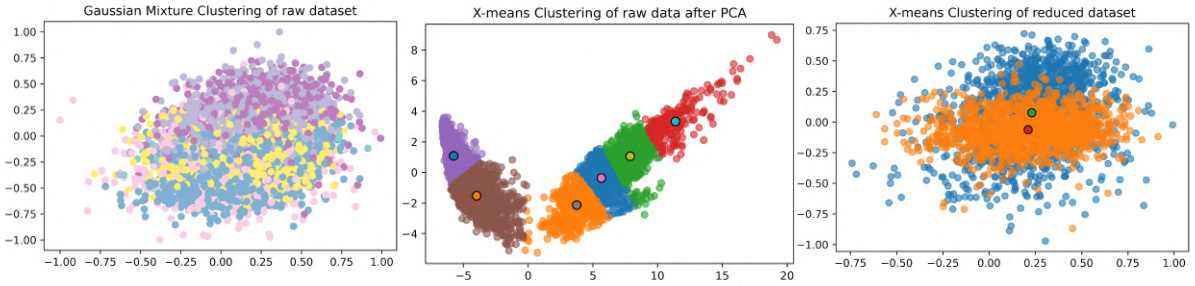


Figure 46 - First approaches at clustering

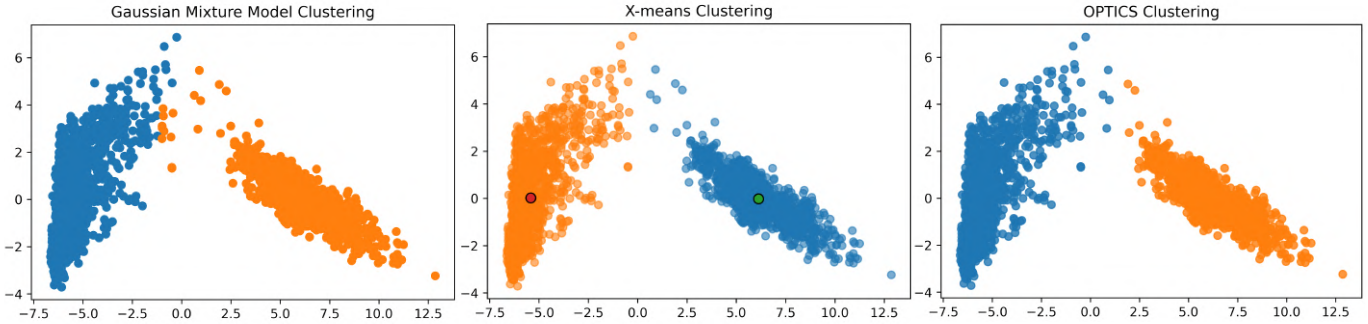


Figure 47 - Detailed comparison of clustering algorithms for PCA of reduced dataset

Table 12 - Silhouette Score for different clustering algorithms using several approaches

	Gaussian Mix	X-means	OPTICS
Raw Data	0.1432	0.2765	0.1255
PCA of Raw Data	0.3787	0.6423	0.2839
Reduced Dataset	0.5233	0.6587	0.5576
PCA of Reduced Dataset	0.7761	0.9274	0.7799

10 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 XGBRegressor as well.

10.1 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 47 we can see an example of LIME applied to a particular instance of our data. In this case we want to see how the ExtraTrees 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 2938, which was predicted to belong to class *Walking_upstairs* by the model and is indeed *Walking_upstairs* according to the data. For this record, the model assigned the following probabilities of belonging to class [1, 2, … , 6], being: [0.31, 0.43, 0.26, 0, 0, 0]. 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. For

multiclass classification, the explainer works in a fashion similar to that of a binary one, where it either belongs to a certain class or it doesn't.

In the case of our example record, it tells us the features with the biggest weight on the classifying decision, namely *54GravityAcc-min()-Y* and *103BodyAccJerk-entropy()X* among others, while at the same time stating the features that contributed the most to the other possibilities of classification, for example *75GravityAcc-arCoeff()Z,2*.

After this approach, we decided to make use of this algorithm to get a deeper understanding of the classes that tend to get misclassified the most, like W*alking_upstairs* and *Walking_downstairs*. To do this we focused only on these classes, leaving out the rest and repeating the same process as before. Now using record 774, we get a 53% probability that the record is classified as walking_upstairs, which is a correct classification. Now the interesting part comes when looking at those features which contributed the most to the decision, where we can see some familiar elements, like the previous feature number 54 found before, along with feature 42 for example, which appears in both cases. This provides us some valuable insights into our data, shedding light on which features are most closely related to each class, being consistent with the number of classes known beforehand. on which features are most closely related to each class.

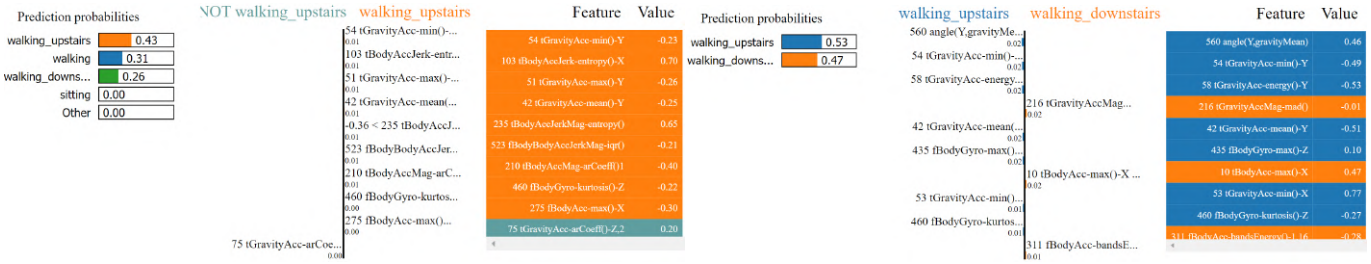


Figure 48 - Results of LIME algorithm for the classification of record 2938 and 774

10.2 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 classification obtained through an XGBRegressor. With regard to the study of features at the local level, we decided to compare two records of the training set representing the most distant motor classes: the 1087th record for the *Walking* class and the 1062th record for the *Laying* class. Figure 49 shows which variables have the most influence on the instance of *Walking* class. and the same has been done for the *Lying* class record in fig 50.

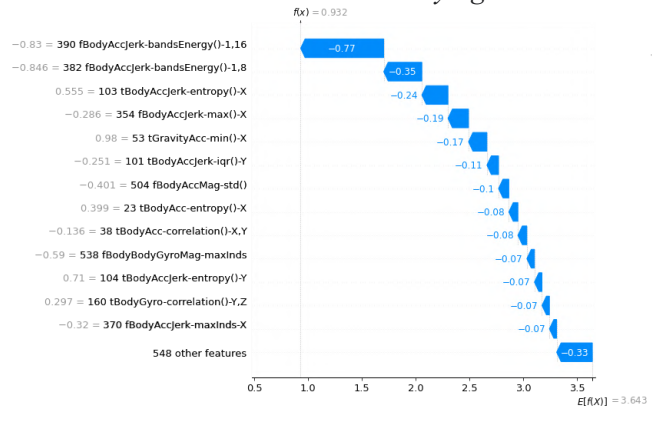


Figure 49 - Local SHAP Walking Class

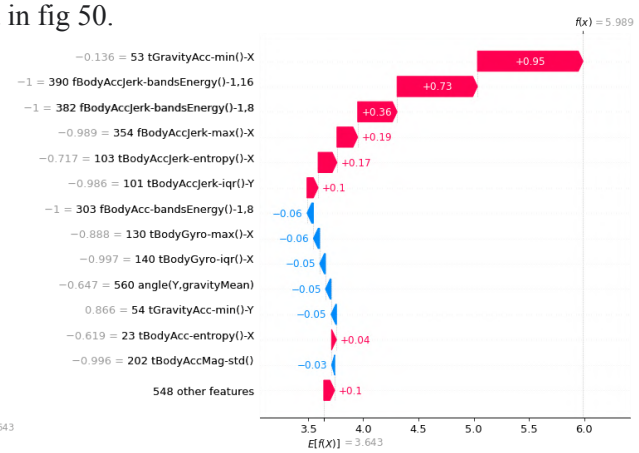


Figure 50 - Local SHAP Lying Class

While in class 6 (*Lying*) we can see how there are features that try to go "against the grain", it is interesting to highlight how for class 1 (*Walking)* the classification is much more direct and simple.

On the other hand, with regard to the behavior of the features on a global level, we can see from the plot in figure 51 which ones have the greatest influence on the decisions of the classifier under consideration. The plot shows in descending order the features that have the greatest impact on the dataset, as their value varies.

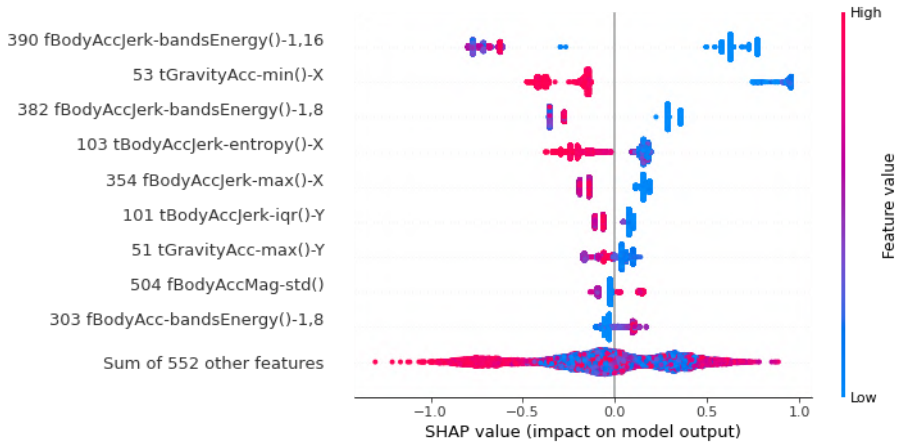


Figure 51 - Global SHAP

11 Conclusions

The Human Activity Recognition HAR dataset was already very well defined and structured, without missing values and with a target value for the most part balanced. Throughout most of the tasks performed on the HAR dataset, the results were overall satisfactory, especially when dealing with classification, which straight away was able to reach high scores close to .90 on metrics such as accuracy and F1-score with just the base model of simple algorithms such as decision trees or KNN. These results however, could be taken even further with the use of hyperparameter tuning and algorithms closer to the state of the art, such as XGBoost, which provided us with scores close to .99.

One of the tricky parts of the dataset however was the high number of features, with a high number of them having a high correlation value and therefore prone to be affected by the curse of dimensionality. This made the dimensionality reduction part especially important, since it greatly helped in areas such as visualization with PCA, or Univariate Feature Selection, which provided us with the most statistically significant features to use for a better linear regression model for example.

With regard to the time series section, we performed most of our analysis on the '*total_acc_x_train*' dataset as it had a higher variance than the others. In particular, it was used for the first transformations, Dynamic Time Warping, and the different approximation techniques. Clustering analysis was performed using '*body_gyro_y_train*', as well as for Shapelets and Motif discovery. Lastly, '*total_acc_x_train*' was again used for the time series classification, which, although it reported excellent results in terms of performance it also highlighted a common misclassification when predicting the 5th class.

While testing the different advanced clustering techniques, the dimensionality reduction methods previously exploited, such as PCA, played a big role in giving us an optimal visualization of the clusters of each different cluster method. Each of the visualizations, obtained with a number of components equal to 2, resulted in a structure of 2 well separated clusters.

The Explainability field has been our last focus mining the HAR dataset, using the LIME and SHAP algorithms it has been possible to explain, in a human comprehensible form, how the black box models (ExtraTrees, XGBRegressor) made their decisions based on the given features. The methods explained in a very clear way, especially through visualization methods, how the different features impacted the classification choice during the calculus.

References

[1] Domingos, P., Pazzani, M. 1997. On the optimality of the simple Bayesian classifier under zero-one loss. Machine learning, 29(2-3), pp.103-130.

[2] Zhang, H. 2005. Exploring conditions for the optimality of naive Bayes. International Journal of Pattern Recognition and Artificial Intelligence, 19(02), pp.183-198.

[3] Dorugade, D., Kashid, N. Alternative Method for Choosing Ridge Parameter for Regression. Applied Mathematical Sciences, Vol. 4, 2010, no. 9, 447 – 456.

[4] Breunig, M. M.; Kriegel, H.-P.; Ng, R. T.; Sander, J. R. (1999). "OPTICS-OF: Identifying Local Outliers" (PDF). Principles of Data Mining and Knowledge Discovery. Lecture Notes in Computer Science. Vol. 1704.