
UNIVERSITÀ DI PISA

# HUMAN ACTIVITY RECOGNITION USING SMARTPHONES

Antonio Calabrese (620370)
Chiara Menchetti (647629)
Carlo Volpe (646873)

DATA MINING 2 PROJECT

Academic Year 2021/2022

# Contents

# 1 Introduction

The UCI-HAR dataset was established for developing Human Activity Recognition algorithms using smartphones. In Artificial Intelligence, Human Activity Recognition (HAR) refers to the capability of machines to identify various activities performed by the users. The knowledge acquired from these recognition systems is integrated into many applications where the associated device uses it to identify actions or gestures and performs predefined tasks in response.

The experiments were carried out with a group of 30 volunteers (of age 19-48) performing six different activities: WALKING, WALKING UPSTAIRS, WALKING DOWNSTAIRS, SITTING, STANDING, LAYING. The data was collected through a smartphone (Samsung Galaxy S II), worn on the participants' waist, using its embedded accelerometer and gyroscope, which was able to capture 3-axial linear acceleration and 3-axial angular velocity at a constant rate of 50Hz. The experiments have been video-recorded to label the data manually.

# 2 Data Understanding & Preparation

The HAR dataset has been randomly partitioned into two sets, where 70% (21) of the volunteers was selected for generating the training data and 30% (9) the test data. For each record in the raw dataset it is provided: 3-axial acceleration from the accelerometer (total acceleration) and the estimated body acceleration; 3-axial angular velocity from the gyroscope; a 561-feature vector with time and frequency domain variables; its activity label and an identifier of the subject who carried out the experiment. All the data has already been normalized to an interval [-1, 1].

In order not to compromise the impartiality of the results, in this first analysis, only the training set is taken into account. The training data contains 7352 observations and 561 variables plus the target variable describing the activity. There are no missing information, so there is no need to implement any correction based on substituting or removing values from any record. The number of instances provided by each participant (Figure 1) is not the same across each different task, however this slight imbalance is not significant enough to be considered as an issue for the following steps.

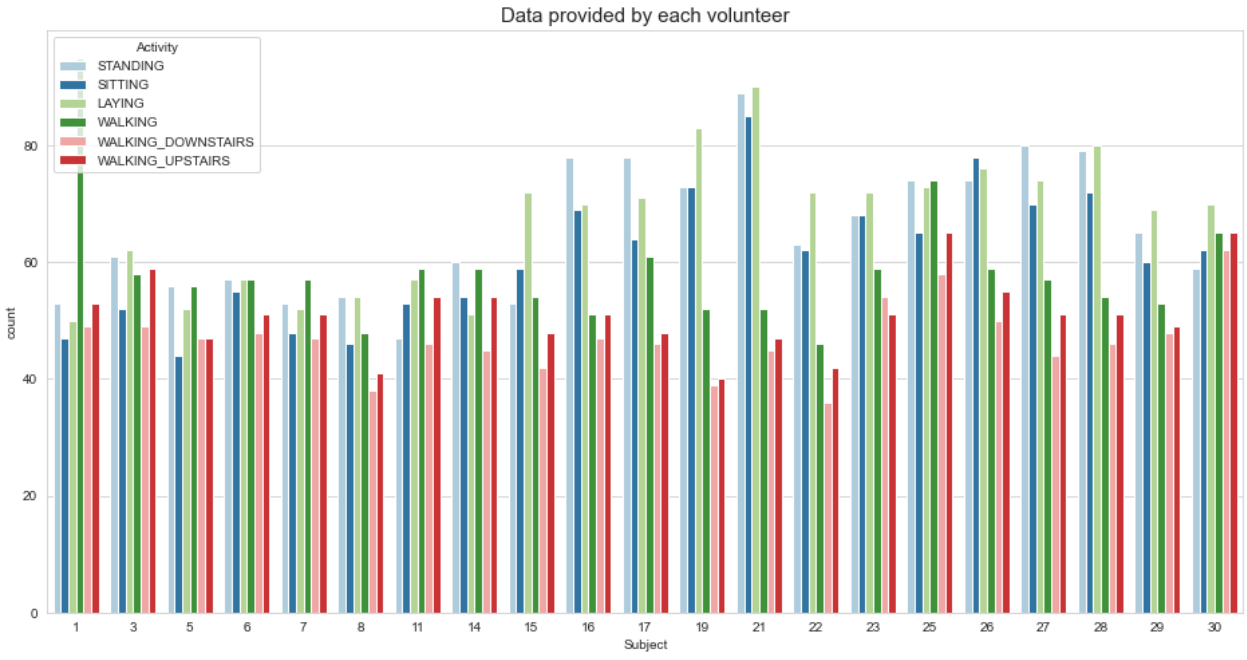



Figure 1: Data provided by each volunteer, belonging to the training data, divided by activity

The *Label* variable, identified as the target variable, indicates the type of activity the subject was performing during the collection of the different 561 parameters considered. As shown in Figure 2, the different

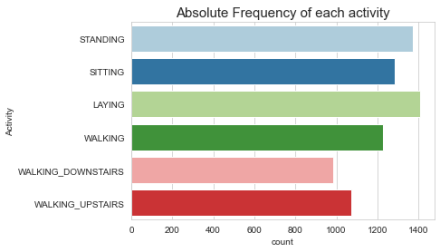
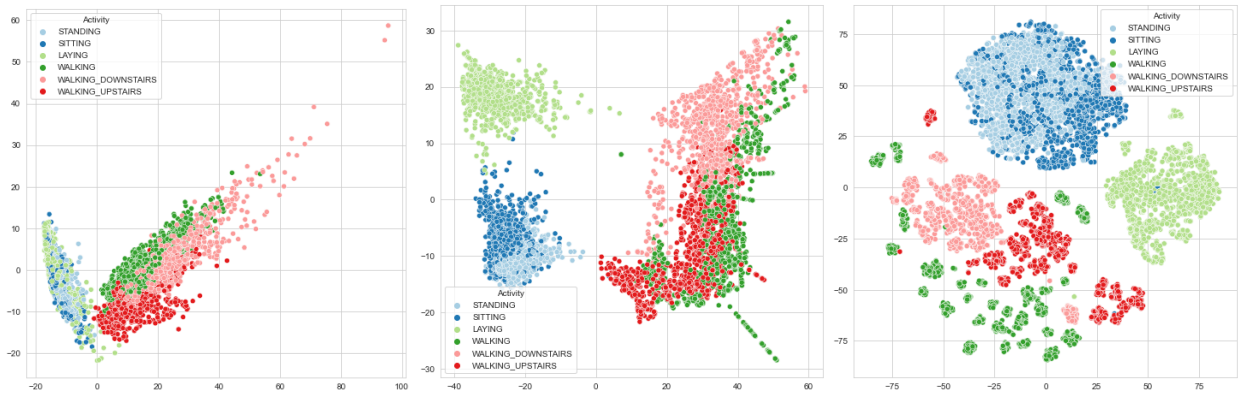



Figure 2: Absolute frequency of each activity in the training data

levels are quite balanced. The observations are divided into: WALKING (16.7%), WALKING UPSTAIRS (14.6%), WALKING DOWNSTAIRS (13.4%), SITTING (17.5%), STANDING (18.7%) and LAYING (19.1%).

Given the high number of dimensions present in the data, it is very difficult to analyse or visualise each one of them as they are. For this reason, some dimensionality reduction techniques were implemented to understand from a spatial point of view if there were any particular relations between the different activities. The feature projection methods implemented for these 2D representations (Figure 3) are respectively PCA (Principal Component Analysis), Isomap (Isometric Feature Mapping) and t-SNE (t-Distributed Stochastic Neighbor Embedding). For each of them the first two principal components were selected, independently from the fact, or not, that they represented most of the explained variability in the data.




(a) PCA representation (b) Isomap representation (c) t-SNE representation

Figure 3: 2-Dimensional representation of the data using feature projection

Given these illustrations, in a lower dimension, of the instances in the training data, divided by the activity labels, some observations can be drawn. In all of them there is a clear distinction between stationary tasks (SITTING, STANDING, LAYING) and non-stationary tasks (WALKING, WALKING UPSTAIRS, WALKING DOWNSTAIRS). Especially for the PCA, the first principal component (on the x axis), which explains 50.8% of the total variability in the data, most probably captures the acceleration of the person.

As the last step of the preprocessing, a new dataset was created by removing the variables with a correlation higher than, or equal to, 0.80.

# 3 Simple Classification

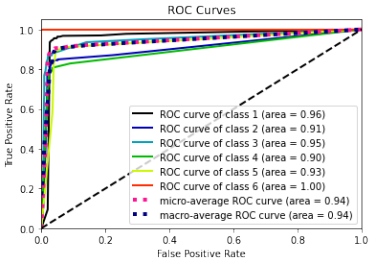
The objective of this analysis is to predict the activity each volunteer was undertaking given the parameters collected. Two methods of supervised classification, namely the Decision Tree classifier and the k-Nearest Neighbor classifier, are implemented in this section in order to create a sort of baseline for the comparison with more complex approaches, that will be taken into account at a later stage.

## 3.1 Decision Tree Classifier

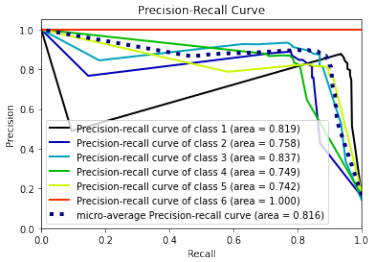
The first supervised learning method used is the Decision Tree classifier, which acquires knowledge in the form of a tree. No techniques for dimensionality reduction or transformation of the data of any kind were used before fitting the model. The parameters chosen, based on the analysis of a cross-validation grid search's output, were: $max\_depth = 10$, $min\_samples\_splits = 10$ and $min\_samples\_leaf = 7$. The results obtained by the classifier are reported below (Figure 4) along with the roc and precision-recall curves.

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 1 | 0.83 | 0.95 | 0.89 | 496 |
| 2 | 0.87 | 0.78 | 0.82 | 471 |
| 3 | 0.89 | 0.84 | 0.86 | 420 |
| 4 | 0.87 | 0.76 | 0.81 | 491 |
| 5 | 0.80 | 0.89 | 0.85 | 532 |
| 6 | 1.00 | 1.00 | 1.00 | 537 |
| accuracy | | | 0.88 | 2947 |
| macro avg | 0.88 | 0.87 | 0.87 | 2947 |
| weighted avg | 0.88 | 0.88 | 0.87 | 2947 |

(a) Classification report




(b) ROC curve




(c) Precision-Recall curve

Figure 4: Results for Decision Tree Classifier

Given the classifier and that no transformation was applied to the data, the results are optimal. Both the area under the roc curves and the areas under the precision-recall curves are quite high.

### 3.2 k-Nearest Neighbor Classifier

The second base classifier used is the k-Nearest Neighbor. The settings to implement this method, however, were chosen to be slightly different from the previous one. The data was standardized, as the classifier is based on a notion of distance, differently from the Decision Tree which is not. And, as it suffers from the curse of dimensionality, the variables were reduced based on their pairwise correlations. A function was used to drop the variables having a pairwise correlation of more than 0.80, which resulted in going from 561 to 151 features. The parameter k was chosen according to the one-standard-error rule (Figure 5) which indicates optimal $k = 13$. The results obtained by the classifier are an accuracy of 0.88 and an error of 0.12.

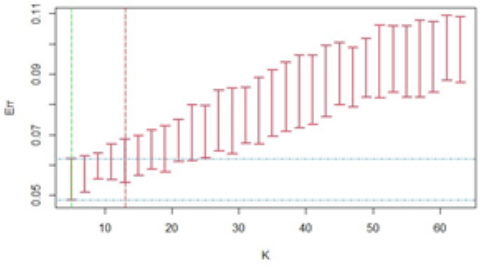



Figure 5: Parameter k for k-NN

## 4 Anomaly Detection

The next step of the analysis is outlier detection, whose goal is to identify anomalous observations in the data that could compromise the classification's results. Various techniques are implemented to ultimately identify the top 1% records having the highest outlier score (considering only the training data this would be 74 instances). All the following approaches were applied on the dataset with 151 non-correlated features given that correlation could influence some results.

### 4.1 Local Outlier Factor

The Local Outlier Factor (LOF) algorithm is an unsupervised anomaly detection method which computes the local density deviation of a given data point with respect to its neighbors. It considers as outliers the samples that have a substantially lower density than their neighbors. So, the advantage of using LOF is in identifying points that are outliers relative to a local cluster of points. However, the main drawback is that by using densities, the higher the number of dimensions, the less meaningful the approach will be. For the analysis, different values of the k (nearest neighbors) parameter were tested. Applying the LOF algorithm to our dataset with $k = 50$, 44 observations were identified as anomalies (Figure 6).

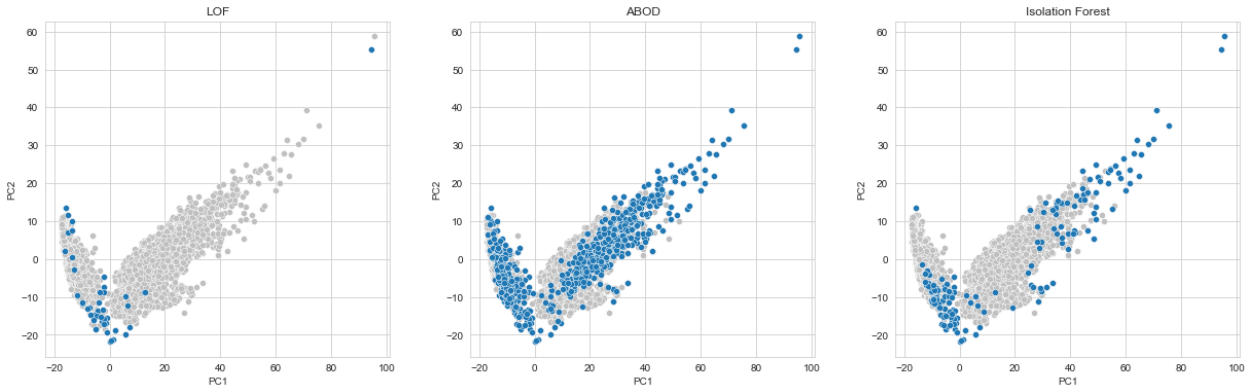
### 4.2 Angle Based Outlier Detection

Angle-based outlier detection (ABOD) is useful in outlier detection for high-dimensional data. Instead of examining neighborhoods as proximity-based concepts, it assesses the broadness of angle spectrum of a point as an outlier factor. The basic assumption is that outliers are at the border of the data distribution while normal points are in the centre of data distribution. The ABOD model evaluates the degree of outlierness on the variance of the angles (VOA) between a point and all other pairs of points in the data set. The intuition

is that the smaller the angle variance of the point, the more likely it is an outlier. The reason why ABOD does not substantially deteriorate in high-dimensional data is due to the fact that angles are more stable than distances. For the analysis, the parameter *n_neighbors* was set to 20, and thus 723 outliers were identified (Figure 6).

## 4.3 Isolation Forest

Isolation Forest is a model-based approach, which is very efficient and reliable. The main advantage of the algorithm is that it does not rely on building a profile for data in an effort to find samples that do not conform to this profile. Rather, it utilizes the fact that anomalous data are "few and different". Most anomaly detection algorithms find anomalies by understanding the distribution of their properties and isolating them from the rest of normal data samples. In an Isolation Forest, data is sub-sampled, and processed in a tree structure based on random cuts in the values of randomly selected features in the dataset. Those samples that travel deeper into the tree branches are less likely to be anomalous, while shorter branches are indicative of anomaly. As such, the aggregated lengths of the tree branches provide a measure "anomaly score" for every given observation. The records identified as outliers by the algorithm were 166 (Figure 6).




(a) LOF (b) ABOD (c) Isolation Forest

Figure 6: Visualization of outliers according to different approaches

## 4.4 Extended Isolation Forest

The Extended Isolation Forest, as the name suggests, is an extension of the previous anomaly detection approach. The main advantage here is that it allows the branching process to occur in non-orthogonal directions. The process of choosing branch cuts is altered so that at each node, instead of choosing a random feature along with a random value, we choose a random normal vector along with a random intercept point. As in the case of the standard Isolation Forest, the anomaly score is computed by the aggregated depth that a given point reaches on each tree. To implement this in pyhton the EIF library with the *iForest* function was used. The parameters chosen, based on the suggestion by the authors, are: *ntrees* = 200, *sample_size* = 256 and *ExtensionLevel* = 150. The extension level is set as N-1 (where N is the number of dimensions), which is the fully extended case. The outliers scores are represented in Figure 7.

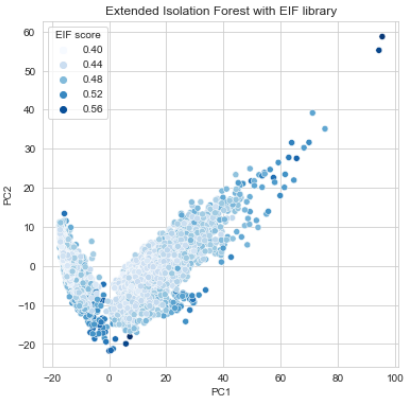



Figure 7: EIF outliers scores

### 4.5 Removal of outliers

The high dimensionality of the dataset (151 variables), and the objective of identifying only 1% of the observations (74) as outliers, suggested to proceed by using the outputs of the ABOD and Isolation Forest methods, as they are the ones able to handle such problems. The results of the these two approaches were compared in order to select the anomalies they had in common. A total of 143 outliers were in both outputs, which were then sorted according to the Extended Isolation Forest's scores. From here, the 74 observations with the highest anomaly score were selected and subsequently removed from the dataset (Figure 8). It was decided to remove them since they were only 1% of the observations.

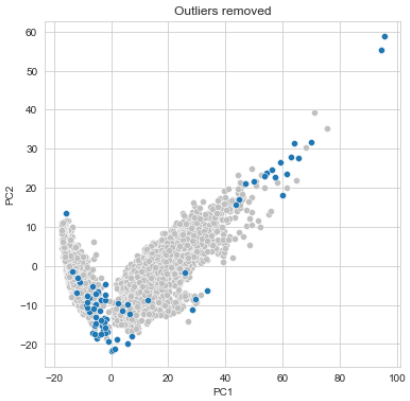



Figure 8: Outliers removed from the dataset

## 5 Imbalanced Learning

The value distribution of the class to predict, as illustrated in the data understanding section, is balanced. Given the didactic purpose of the project, the dataset will be turned into an imbalanced two-class version for this section. Starting from the training data, without outliers and with only the 151 non-correlated features, the classes were grouped as follows: 1 - SITTING (1270 observations) and 0 - NOT SITTING (WALKING + WALKING UPSTAIRS + WALKING DOWNSTAIRS + STANDING + LAYING = 6008 observations). Then, 1030 samples were randomly removed from the positive class (SITTING) to obtain a 96% - 4% imbalanced problem. By fitting a Decision Tree classifier, on this unbalanced data, the best results (Table 1) are obtained with the following parameters: $max\_depth = 10$ $min\_samples\_leaf = 1$ and $min\_samples\_split = 2$. The accuracy, which is very high (0.93), in this case is misleading. The recall and F measure for the positive class are quite low, so the model has not optimal predictive capacities, due to the imbalanced setting. Different strategies can be adopted to overcome this obstacle.

### 5.1 Undersampling

One possibility is undersampling the majority class. Both Random Undersampling and Condensed Nearest Neighbour were implemented. Random Undersampling brought both classes to 240 samples. The performance was tested through a Decision Tree classifier, just like before, and for the positive class, as the accuracy remained stable (0.93), the recall (0.88) and F1 (0.80) increased significantly, whereas the precision dropped to 0.74. However, the limitation in using this approach is the randomness in sampling, which affects the performance. Condensed Nearest Neighbour, on the other hand, is based on the notion of k-NN. It downsamples the majority class from 6008 to 440 observations. By fitting a Decision Tree on this data the performance was worse than the one in the imbalanced setting.

### 5.2 Oversampling

Another approach is oversampling the minority class. Two techniques were implemented: Random Oversampling and SMOTE. Random Oversampling brought both classes to 6008 observations. The performance of the Decision Tree classifier was overall more balanced than the previous ones: precision 0.87, recall 0.77, F1 0.81 and accuracy 0.94. SMOTE brought both classes to 6008 samples as well. The performances though were not as great: precision 0.83, recall 0.74, F1 0.78 and accuracy 0.93. All the results from the Decision Tree classifier are reported in Table 1.

| | Precision | Recall | F1 - score | Accuracy |
|---|---|---|---|---|
| Imbalanced data | 0.93 | 0.61 | 0.74 | 0.93 |
| Random Undersampling | 0.74 | 0.88 | 0.80 | 0.93 |
| CNN | 0.56 | 0.68 | 0.61 | 0.86 |
| Random Oversampling | 0.87 | 0.77 | 0.81 | 0.94 |
| SMOTE | 0.83 | 0.74 | 0.78 | 0.93 |

Table 1: Decision Tree classifier performance evaluations wrt the positive class

# 6 Dimensionality Reduction

Dimensionality reduction simply refers to the process of reducing the number of attributes in a dataset while keeping as much of the variation in the original dataset as possible. Dimensionality reduction can lead to several advantages, such as: less training time and less computational resources whilst maintaining or increasing the overall performance of machine learning algorithms; avoiding the problem of overfitting; data visualization (as shown in paragraph 2); removing noise in the data. There are mainly two types of dimensionality reduction techniques: feature selection and feature projection. Feature Selection methods only keep the most important features in the dataset and remove the redundant ones. There is no transformation applied. On the other hand, Feature Projection methods transform the data in the high-dimensional space to a space of fewer dimensions. The data transformation may be linear, or nonlinear. For all the feature selection methods the starting point is the 151-dimensional dataset, whereas for the PCA is the complete dataset.

## 6.1 Variance Threshold

The Variance Threshold technique removes all features with a training-set variance lower than some threshold. By default, it removes all zero-variance features, i.e. features that have the same value in all samples. The threshold chosen for this project was 0.1, resulting in 39 variables.

## 6.2 Univariate Feature Selection

Univariate feature selection works by selecting the best features based on univariate statistical tests. Each feature is compared to the target variable, to see whether there is any statistically significant relationship between them. It is also called analysis of variance (ANOVA). The drawback of this approach is that the number of features returned must be selected apriori. So, the parameter k (for the top statistically significant features) was set to 20 variables.

## 6.3 Recursive Feature Elimination

Recursive feature elimination attempts to eliminate dependencies and collinearity that may exist in the model. Features are ranked by recursively eliminating a small number of features per loop. Given an external estimator that assigns weights to features (e.g., the coefficients of a linear model), RFE selects features by recursively considering smaller and smaller sets of features. By considering a Decision Tree classifier, as model, this technique returns 12 variables.

## 6.4 Principal Component Analysis

Principal component analysis, differently from the previous three applications, is a feature projection approach. The objective of PCA is to find a small number of linear combinations of the observed variables which explain most of the variation in the data. Given the original 561-dimensions training data, the minimum number of components to reach an 80% of cumulative explained variance is 26.

### 6.5 Discussion on performance evaluation and outlier detection

Each dimensionality reduction technique returned a set of variables meaningful to that method. For each set of defined variables, a decision tree was constructed and performance calculated. For the feature selection approaches the results of the three classifiers were roughly the same. Whereas, by using the 26 components from PCA, the accuracy, even with different parameter settings, was not able to exceed 0.78. Then, it was decided to retain the variables from the Variance threshold approach to implement a second outlier detection using LOF, ABOD and Isolation Forest. As a result, the LOF method returned only 1 outlier, ABOD returned 712 outliers and Isolation Forest returned 1083 outliers.

## 7 Advanced Classification

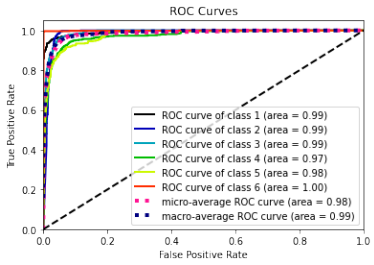
In this section various methods of advanced classification are implemented. For each one of these, hyperparameter tuning was performed, and the outputs were analysed. The datasets that are mainly going to be used are: the one containing all the 561 variables and the one containing the 151 non-correlated features (both without the 74 outliers identified in the previous module).

### 7.1 Naive Bayes Classifier

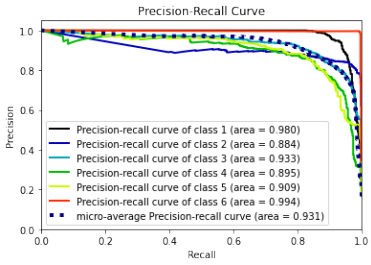
Naive Bayes is a probabilistic classification technique based on Bayes' theorem with the "naive" assumption of conditional independence between every pair of features given the value of the class variable. The function implemented is the *GaussianNB* which assumes the data is distributed following a Gaussian probability density function. And the mean and standard deviation parameters are estimated through Maximum Likelihood estimation. The features considered for this supervised learning task are the 151 non-correlated ones. There are not really any parameter to tune, so the results obtained are reported here (Figure 9).

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 1 | 0.92 | 0.94 | 0.93 | 496 |
| 2 | 0.84 | 0.93 | 0.88 | 471 |
| 3 | 0.89 | 0.85 | 0.87 | 420 |
| 4 | 0.94 | 0.55 | 0.70 | 491 |
| 5 | 0.71 | 0.92 | 0.80 | 532 |
| 6 | 0.99 | 1.00 | 0.99 | 537 |
| accuracy | | | 0.87 | 2947 |
| macro avg | 0.88 | 0.86 | 0.86 | 2947 |
| weighted avg | 0.88 | 0.87 | 0.86 | 2947 |

(a) Classification report




(b) ROC curve




(c) Precision-Recall curve

Figure 9: Results for Gaussian Naive Bayes Classifier

As seen in the classification report, the evaluation parameters are not better than the ones of a simple decision tree classifier. However, the areas under the curves of both ROC and Precision-Recall curves are slightly higher.

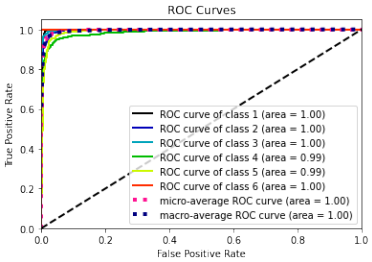
Another Naive Bayes application (*CategoricalNB*) is applied by considering categorical attributes. Each feature was discretized into four equal-sized buckets (0-0.25, 0.25-0.50, 0.50-0.75, 0.75-1). From the analysis, the results are similar to the ones obtained with the previous classifier.

### 7.2 Logistic Regression

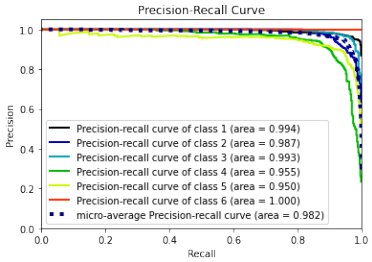
The next classification technique used is the Multinomial Logistic Regression, that generalizes the multiclass problem in logistic regression. Multinomial models are models in which the dependent variable is a qualitative variable that can take multiple values. The objective is to understand how we can explain/predict the probability that an observation belongs to a certain class. The results, obtained on the dataset containing non-correlated features, are reported in Figure 10.

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 1 | 0.93 | 0.99 | 0.96 | 496 |
| 2 | 0.96 | 0.92 | 0.94 | 471 |
| 3 | 0.97 | 0.95 | 0.96 | 420 |
| 4 | 0.93 | 0.87 | 0.90 | 491 |
| 5 | 0.89 | 0.94 | 0.92 | 532 |
| 6 | 1.00 | 1.00 | 1.00 | 537 |
| accuracy | | | 0.95 | 2947 |
| macro avg | 0.95 | 0.95 | 0.95 | 2947 |
| weighted avg | 0.95 | 0.95 | 0.95 | 2947 |

(a) Classification report




(b) ROC curve




(c) Precision-Recall curve

Figure 10: Results for Multinomial Logistic Regression

The evaluation measures are significantly improved with respect to the classifiers seen until now. Also, the curves are close to perfect.

## 7.3 Support Vector Machines

Support Vector Machines learning aims to construct, in a given feature space, a hyperplane to best separate training samples with different class labels. The hyperplane is derived on the basis of a limited number of training instances, called support vectors, to maximize a margin on each side of the plane. If the data are not separable by a hyperplane, the data can be projected into feature spaces of higher dimensionality where linear separation might be possible. Once found the maximal margin hyperplane that separates the data, it is used as a model for classification. In SVM, mapping into higher-dimensional feature spaces is accomplished through the use of kernel functions. The analysis was done considering different kernel functions (*linear*, *poly*, *rbf*, *sigmoid*) and different values of $C$ (0.01, 0.1, 1.0, 10.0, 100.0), which is a regularization parameter on the level of misclassification errors the model is allowed. The dataset chosen for this task is still the one made up of the non-correlated features. This due to the fact that some kernels do not work well with correlations and might then provide erroneous results. For instance, with a linear kernel the effect of multicollinearity is similar to the one of a logistic regression. The rbf kernel, instead, is based on distances between data points, similar to k-Nearest Neighbors, so if correlated variables are used, they will have more impact on the classification results. The data was standardised by using z-score (*StandardScaler* function).

### 7.3.1 Linear SVM

The classification was obtained through the *LinearSVC* function. Shifting the value of $C$ with the different values stated above, the classification results remained unchanged, except for 10.0, where the accuracy went down by approximately 0.01. In Figure 11 are reported the results with $C = 1.0$.

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 1 | 0.95 | 0.97 | 0.96 | 496 |
| 2 | 0.93 | 0.92 | 0.93 | 471 |
| 3 | 0.97 | 0.94 | 0.95 | 420 |
| 4 | 0.93 | 0.86 | 0.90 | 491 |
| 5 | 0.88 | 0.94 | 0.91 | 532 |
| 6 | 0.99 | 0.99 | 0.99 | 537 |
| accuracy | | | 0.94 | 2947 |
| macro avg | 0.94 | 0.94 | 0.94 | 2947 |
| weighted avg | 0.94 | 0.94 | 0.94 | 2947 |

Figure 11: Classification report for Linear SVM

### 7.3.2 Non-Linear SVM

The non-linear SVM where obtained through the *SVC* function, and specifying the kernel parameter. For the polynomial kernel, as C increases the performance of the classifier changes. For $C = 0.01$, the accuracy is as low as 0.23; for $C = 0.1$, it is 0.75; for $C = 1.0$, it is 0.91. The best results however are for 10.0 and 100.0 (Figure 12a) where the accuracy reaches 0.92. For the rbf kernel, as $C$ changes the pattern is similar to the previous kernel, with a slightly increase in the performance. For $C = 0.01$ the accuracy is 0.61; for 0.1 it is 0.93; and then it stabilizes to 0.95. In Figure 12b is the classification report for $C = 1.0$. For the sigmoid kernel, the shift in the value of $C$ does not follow the same fashion as the previous two. The higher performance results are obtained with $C = 0.1$ (accuracy = 0.93)(Figure 12c), followed by $C = 1.0$ (accuracy = 0.91). Whereas, for the other values of $C$ (0.01, 10.0, 100.0), the accuracy measures are respectively 0.85, 0.88 and 0.88.

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 1 | 0.84 | 1.00 | 0.91 | 496 |
| 2 | 0.95 | 0.85 | 0.90 | 471 |
| 3 | 0.98 | 0.90 | 0.94 | 420 |
| 4 | 0.90 | 0.90 | 0.90 | 491 |
| 5 | 0.89 | 0.94 | 0.91 | 532 |
| 6 | 1.00 | 0.93 | 0.96 | 537 |
| accuracy | | | 0.92 | 2947 |
| macro avg | 0.93 | 0.92 | 0.92 | 2947 |
| weighted avg | 0.93 | 0.92 | 0.92 | 2947 |

(a) Poly kernel

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 1 | 0.95 | 0.99 | 0.97 | 496 |
| 2 | 0.98 | 0.93 | 0.95 | 471 |
| 3 | 0.97 | 0.96 | 0.97 | 420 |
| 4 | 0.92 | 0.91 | 0.91 | 491 |
| 5 | 0.91 | 0.93 | 0.92 | 532 |
| 6 | 0.99 | 0.98 | 0.98 | 537 |
| accuracy | | | 0.95 | 2947 |
| macro avg | 0.95 | 0.95 | 0.95 | 2947 |
| weighted avg | 0.95 | 0.95 | 0.95 | 2947 |

(b) Rbf kernel

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 1 | 0.92 | 0.99 | 0.95 | 496 |
| 2 | 0.94 | 0.93 | 0.93 | 471 |
| 3 | 0.95 | 0.90 | 0.93 | 420 |
| 4 | 0.87 | 0.86 | 0.87 | 491 |
| 5 | 0.88 | 0.91 | 0.89 | 532 |
| 6 | 1.00 | 0.96 | 0.98 | 537 |
| accuracy | | | 0.93 | 2947 |
| macro avg | 0.93 | 0.92 | 0.92 | 2947 |
| weighted avg | 0.93 | 0.93 | 0.93 | 2947 |

(c) Sigmoid kernel

Figure 12: Classification reports for non-linear SVM

It is worth mentioning that this same process was applied also considering all the 561 features. The performance of the linear SVM was slightly better with an accuracy of 0.97 (for $C = 1.0$). Whereas, the other kernels produced results similar to the ones above, and in case of the sigmoid kernel the best accuracy obtained was 0.85 ($C = 0.1$), which is significantly less than the results obtained with a reduced set of features.

## 7.4 Neural Networks

Neural Networks, also known as Artificial Neural Networks (ANNs), are a series of algorithms that endeavours to recognise underlaying relationships in a set of data through a process which is inspired from the way the human brain operates. The 561-dimensional dataset was used for the following analysis. The data was standardized with z-score (*StandardScaler* function in Python) as Multi-layer Perceptron is sensitive to feature scaling. The libraries required for the evaluation and construction of the model are Sklearn, Tensorflow and Keras.

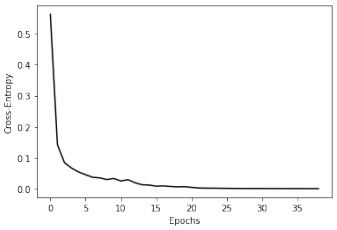
### 7.4.1 Neural Networks with Sklearn

The *MLPClassifier* from Sklearn implements a multi-layer perceptron (MLP) algorithm. It can solve any type of classification task involving non-linear decision surfaces.

For a multiclass classification the output function is *Softmax*, also known as normalized exponential function, which is a generalization of the logistic function to multiple dimensions. The *MLPClassifier* uses the cross-entropy loss function. The other parameters tuned with a grid search using a 3-fold cross validation were: *solver* (*sgd*, *adam*), *momentum* (0, 0.1, 0.2, 0.3, 0.5, 0.7,0.9), *hidden_layer_sizes* ((128, 64, 23,), (100, 69,), (128, 64, 32,)) and *learning_rate* (*constant*, *invscaling*, *adaptive*). As for the two different solvers: Stochastic Gradient Descent (sgd) updates parameters using the gradient of the loss function with respect to a parameter that needs adaptation, whereas Adam is also a stochastic optimizer, but it can automatically adjust the amount to update parameters based on adaptive estimates of lower-order moments. The best combination of parameters observed are: *solver* = *adam*, *momentum* = 0.1, *hidden_layer_sizes* = (128, 64, 32,) and *learning_rate* = *adaptive*. The classification report is shown in Figure 13a. Moreover, from Figure 13b, it can be seen that the loss curve converges.

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 1 | 0.96 | 0.99 | 0.97 | 496 |
| 2 | 0.95 | 0.96 | 0.95 | 471 |
| 3 | 0.98 | 0.93 | 0.95 | 420 |
| 4 | 0.96 | 0.90 | 0.93 | 491 |
| 5 | 0.88 | 0.97 | 0.92 | 532 |
| 6 | 1.00 | 0.96 | 0.98 | 537 |
| accuracy | | | 0.95 | 2947 |
| macro avg | 0.95 | 0.95 | 0.95 | 2947 |
| weighted avg | 0.95 | 0.95 | 0.95 | 2947 |

(a) Classification report




(b) Loss curve

Figure 13: MLP Classifier

### 7.4.2 Neural Networks with Keras

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK or Theano. It provides more freedom in the design of the network with respect to the Sklearn version. Models with Keras are defined as a sequence of layers, in such a way to create a sequential model where you can add layers one at a time until the optimal configuration is reached.

For this implementation the dataset setting is the same as in the previous subsection. Given the multi-class nature of the problem some form of encoding must be used for this deep learning algorithm. Since the dependent variable is categorical, and no such ordinal relationship exists between the different unique

values, integer encoding might not be enough. In fact, using this encoding and allowing the model to assume a natural ordering between categories may result in poor performances. In this case, a one-hot encoding can be applied to the integer representation, as to create a new binary variable for each unique integer value (in total 6 binary variables). Due to the need for reproducibility and exact comparison of the models built, the training data was split into training set (90% of the observations) and validation set (10% of the observations). The first model defined (starting from an educated guess from the results of the previous multi-layer perceptron classifier) is a sequential one with dense layers, where: the first layer has 128 neurons with 561 input dimensions and a *relu* activation function; the second layer has 64 neurons and activation function *relu*; the last layer (output layer) has 6 neurons and *softmax* as activation function. The compiler has the following parameters: *loss* = *categorical_crossentropy*, *optimizer* = *adam*, *metrics* = accuracy.

In general, running too many epochs may overtrain the network and result in overfitting and perform poorly when given new data. This is one of the reasons it was decided to keep a hold-out validation set to test the accuracy (and error) after every epoch (simultaneously for the train set and validation set). Due to this tendency of neural networks to overfit, an early stopping strategy was adopted. This was achieved by the *EarlyStopping* function with parameters: *monitor* = *val_loss* and *patience* = 10. This meaning that, the network will run for some epochs and then it will decide to stop when the run of the next ten epochs will not increase, significantly, the performance in the validation data and then backtrack to the best results. The model was fitted on the training data, specifying the validation data, with *epochs* = 100, *batch_size* = 10 and *callbacks* equal to the early stopping function. With this configuration the network stopped after 18 epochs (Figure 14) with the following results:

```
Epoch 18/100
655/655 [==============================] - 1s 2ms/step - loss: 0.0232 - accuracy: 0.9951 - val_loss: 0.0862 - val_accuracy: 0.9835
```

Figure 14: Last epoch of Neural Network model

The performance was then evaluated on the test set:

```
93/93 [==============================] - 0s 1ms/step - loss: 0.3380 - accuracy: 0.9444
```

Figure 15: Evaluation of Neural Network model

As for the previous implementation with the *MLPClassifier*, another hidden dense layer with 32 neurons and activation function *relu* was added to this model, however the performance on the test data was similar (loss = 0.3799 and accuracy = 0.9447) and more epochs were necessary (34 epochs), so for the sake of simplicity this attempt was discarded.

To understand in a more detailed way how the model would have behaved if no early stopping approach were used, it was decided to let it run for 200 epochs without early stopping. The loss curve for the train and validation data is represented in Figure 16. As expected, the model overfits as the number of epochs increases. The evaluation on the test set gives an accuracy of 0.9420 and a loss of 1.1173, which is very high considering that the one from before was 0.3380.

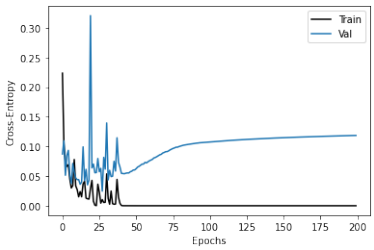



Figure 16: Loss curve of train and validation sets

In an attempt to improve the performances obtained from this model (considering the one with the early stopping criterion), and lower the complexity during training (thus avoiding eventual overfitting), the use of regularization techniques has been considered.

The first is the L2 regularization which adds to the loss function a penalization term. By including *kernel_regularizer* = l2(0.01) to both layers, the network stopped at epoch 68 with the following output:

```
Epoch 68/100
655/655 [==============================] - 1s 2ms/step - loss: 0.1148 - accuracy: 0.9777 - val_loss: 0.1264 - val_accuracy: 0.9739
```

Figure 17: Last epoch of Neural Network model with L2 regularization

The performance on the test set was then evaluated:

```
93/93 [==============================] - 0s 1ms/step - loss: 0.2232 - accuracy: 0.9447
```

Figure 18: Evaluation of Neural Network model with L2 regularization

The accuracy has remained the same with respect to the initial model (0.9444, Figure 15), however the loss has decreased by more than 0.1. Also other values for the L2 regularizer were tested, but with less fortunate results. With *kernel_regularizer* = l2(0.1): epochs = 33, accuracy on test = 0.9091 and loss on test = 0.3749.

With *kernel_regularizer* = l2(0.001): epochs = 53, accuracy on test = 0.9396 and loss on test = 0.2991.

The second technique is the Dropout regularization, which randomly disconnects units from the network during training. This prevents the problem of unit coadaptation i.e. avoids that different neurons are learning the same information. So, different percentages of connections to drop were tested (0.1, 0.2, 0.3, 0.4). The one resulting in the highest accuracy and lowest loss was 0.3. By adding, to the same model defined previously, *Dropout*(0.3) between the first and second layers, the performance on the training data is the following:

```
Epoch 32/100
655/655 [==============================] - 1s 2ms/step - loss: 0.0224 - accuracy: 0.9911 - val_loss: 0.0363 - val_accuracy: 0.9890
```

Figure 19: Last epoch of Neural Network model with Dropout regularization

The network stops at epoch 32. The evaluation on the test set is:

```
93/93 [==============================] - 0s 1ms/step - loss: 0.2861 - accuracy: 0.9484
```

Figure 20: Evaluation of Neural Network model with Dropout regularization

After having observed the behaviour of the network with the different regularizations, it can be concluded that all the models reported have an accuracy around 0.94. In terms of loss, the ones with regularizations have a lower value (0.223 for L2 and 0.286 for Dropout). There could probably be other configurations to get a higher accuracy score and lower loss, but this is the best we could obtain. It is also worth mentioning that other layer sizes (like 100 x 4 layers) and/or activation functions were tested but the results were similar or worse, so it was decided to stick to one configuration to then try the different regularizations techniques on.

## 7.5 Ensemble Methods

Ensemble methods combine the predictions of several base estimators built with a given learning algorithm in order to improve generalizability/robustness over a single estimator. The ensemble methods applied are: Random Forest, Bagging and Boosting.

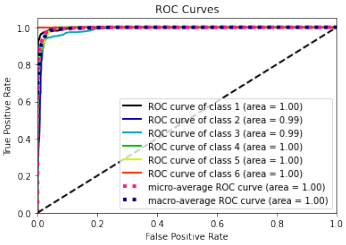
### 7.5.1 Random Forest

The Random Forest classifier operates by constructing a multitude of decision trees at training time which, usually, tend to correct decision trees' habit of overfitting to their training data. The analysis was implemented on the 561-dimensional dataset.

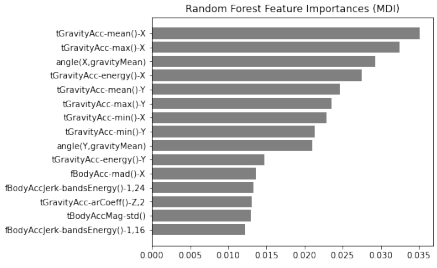
The parameters tuned were: *max_depth* (3, 5, 10, 15, 20) and *n_estimators* (50, 100, 150, 200). The optimal results were obtained through the combination *max_depth* = 20 and *n_estimators* = 100. Above are reported the measure performances, the roc curves and the top 15 most important features. The feature importance is measured with the Mean Decrease in Impurity (MDI) or Gini Importance which is calculated as the sum over the number of splits (across all trees) that include the feature, proportionally to the number of samples it splits. However, this impurity-based measure can inflate the importance of numerical features and it suffers from being computed on statistics derived from the training data. On the other hand, the only thing that can be observed on the test data is the permutation importance (Figure 21d).

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 1 | 0.89 | 0.97 | 0.93 | 496 |
| 2 | 0.89 | 0.90 | 0.90 | 471 |
| 3 | 0.97 | 0.85 | 0.90 | 420 |
| 4 | 0.92 | 0.88 | 0.90 | 491 |
| 5 | 0.89 | 0.92 | 0.91 | 532 |
| 6 | 1.00 | 1.00 | 1.00 | 537 |
| accuracy | | | 0.93 | 2947 |
| macro avg | 0.93 | 0.92 | 0.92 | 2947 |
| weighted avg | 0.93 | 0.93 | 0.92 | 2947 |

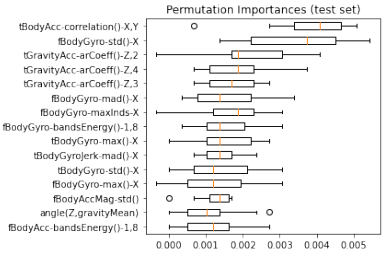
(a) Classification report




(b) Roc curve




(c) Feature importance




(d) Permutation importance

Figure 21: Results for Random Forest

### 7.5.2 Bagging

A Bagging classifier is an ensemble meta-estimator that fits base classifiers each on random subsets of the original dataset and then aggregates their individual predictions to form a final prediction. Different models with different parameters were fitted. Three on the 561-variables dataset: a Bagging classifier with a default Decision Tree as base estimator and *n_estimators* = 100 (Figura 22a); a Bagging classifier with a Random Forest as base estimator with *n_estimators* = 100 (Figura 22b); and a Bagging classifier with a *LinearSVC* with $C = 1000$ and *n_estimators* = 10 (Figura 22c). Another one was fitted on the 151-non-correlated dataset with base estimator *SVC* with $C = 1.0$ and *n_estimators* = 10 (Figura 22d). Comparing the different classification reports, the one obtaining the higher accuracy is the one built upon the Linear Support Vector Machines classifier.

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 1 | 0.88 | 0.96 | 0.92 | 496 |
| 2 | 0.89 | 0.85 | 0.87 | 471 |
| 3 | 0.93 | 0.87 | 0.90 | 420 |
| 4 | 0.88 | 0.80 | 0.84 | 491 |
| 5 | 0.83 | 0.90 | 0.86 | 532 |
| 6 | 1.00 | 1.00 | 1.00 | 537 |
| accuracy | | | 0.90 | 2947 |
| macro avg | 0.90 | 0.90 | 0.90 | 2947 |
| weighted avg | 0.90 | 0.90 | 0.90 | 2947 |

(a) Decision Tree

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 1 | 0.90 | 0.98 | 0.94 | 496 |
| 2 | 0.90 | 0.92 | 0.91 | 471 |
| 3 | 0.96 | 0.84 | 0.89 | 420 |
| 4 | 0.91 | 0.89 | 0.90 | 491 |
| 5 | 0.90 | 0.92 | 0.91 | 532 |
| 6 | 1.00 | 1.00 | 1.00 | 537 |
| accuracy | | | 0.93 | 2947 |
| macro avg | 0.93 | 0.92 | 0.93 | 2947 |
| weighted avg | 0.93 | 0.93 | 0.93 | 2947 |

(b) Random Forest

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 1 | 0.95 | 1.00 | 0.97 | 496 |
| 2 | 0.98 | 0.95 | 0.96 | 471 |
| 3 | 1.00 | 0.98 | 0.99 | 420 |
| 4 | 0.97 | 0.88 | 0.92 | 491 |
| 5 | 0.91 | 0.97 | 0.94 | 532 |
| 6 | 1.00 | 1.00 | 1.00 | 537 |
| accuracy | | | 0.96 | 2947 |
| macro avg | 0.97 | 0.96 | 0.96 | 2947 |
| weighted avg | 0.96 | 0.96 | 0.96 | 2947 |

(c) Linear SVC

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 1 | 0.94 | 1.00 | 0.97 | 496 |
| 2 | 0.95 | 0.92 | 0.94 | 471 |
| 3 | 0.97 | 0.94 | 0.96 | 420 |
| 4 | 0.94 | 0.90 | 0.92 | 491 |
| 5 | 0.91 | 0.94 | 0.93 | 532 |
| 6 | 1.00 | 1.00 | 1.00 | 537 |
| accuracy | | | 0.95 | 2947 |
| macro avg | 0.95 | 0.95 | 0.95 | 2947 |
| weighted avg | 0.95 | 0.95 | 0.95 | 2947 |

(d) SVC

Figure 22: Bagging with different base classifiers

### 7.5.3 Boosting

Boosting generates more models consecutively, giving more and more weight to the errors made in the previous models, so that they are gradually more attentive to the aspects that caused their inaccuracies. Finally obtaining an aggregate model having better accuracy than each model that constitutes it. The function used is *AdaBoostClassifier*, whose output is given by the weighted sum of the predictions of the individual models. The dataset considered is the full one. The two parameter settings reported are: *base_estimator* = *DecisionTreeClassifier* with *max_depth* = 15 and *n_estimators* = 100 (Figura 23a); *base_estimator* = *RandomForestClassifier* with *n_estimators* = 100 (Figure 23b). Given this two configurations, the higher accuracy score is obtained with the Decision Tree classifier as base estimator.

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 1 | 0.94 | 0.98 | 0.96 | 496 |
| 2 | 0.90 | 0.93 | 0.92 | 471 |
| 3 | 0.96 | 0.89 | 0.92 | 420 |
| 4 | 0.95 | 0.86 | 0.90 | 491 |
| 5 | 0.88 | 0.95 | 0.92 | 532 |
| 6 | 1.00 | 1.00 | 1.00 | 537 |
| accuracy | | | 0.94 | 2947 |
| macro avg | 0.94 | 0.93 | 0.94 | 2947 |
| weighted avg | 0.94 | 0.94 | 0.94 | 2947 |

(a) Decision Tree

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 1 | 0.90 | 0.97 | 0.94 | 496 |
| 2 | 0.88 | 0.92 | 0.90 | 471 |
| 3 | 0.97 | 0.84 | 0.90 | 420 |
| 4 | 0.91 | 0.89 | 0.90 | 491 |
| 5 | 0.90 | 0.92 | 0.91 | 532 |
| 6 | 1.00 | 1.00 | 1.00 | 537 |
| accuracy | | | 0.93 | 2947 |
| macro avg | 0.93 | 0.92 | 0.93 | 2947 |
| weighted avg | 0.93 | 0.93 | 0.93 | 2947 |

(b) Random Forest

Figure 23: Boosting with different base classifiers

## 7.6 Gradient Boosting Machines

Gradient Boosting trains many models in a gradual, additive and sequential manner. The major difference between *AdaBoost* and Gradient Boosting Algorithm is in how they identify the shortcomings of weak learners (eg. decision trees). While the *AdaBoost* model identifies the shortcomings by using high weight data points, gradient boosting performs the same by using gradients in the loss function, which is a measure indicating how good are model's coefficients at fitting the underlying data. Different Gradient Boosting functions with a set of different parameters were used for classification purposes. All were implemented on the full dataset. The models implemented are: *GradientBoostingClassifier* with *loss* = 'deviance', *n_estimators* = 100, *learning_rate* = 0.1 and *max_depth* = 3 (Figure 24a); *HistGradientBoostingClassifier* with *loss_function* = 'categorical_crossentropy', *learning_rate* = 0.1 and *max_depth* = 3 (Figure 24b); *XGBClassifier* with *objective*='multi:softprob', *max_depth* = 6 and *learning_rate* = 0.1 (output not included because very similar to the *GradientBoostingClassifier* one); *LXGBClassifier* with *boosting_type*='gbdt', *max_depth* = 6, *num_leaves* = 31, *n_estimators* = 100 and *subsample_for_bin*=200000 (Figura 24c).

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 1 | 0.93 | 0.98 | 0.96 | 496 |
| 2 | 0.93 | 0.92 | 0.92 | 471 |
| 3 | 0.97 | 0.92 | 0.94 | 420 |
| 4 | 0.92 | 0.84 | 0.88 | 491 |
| 5 | 0.86 | 0.94 | 0.90 | 532 |
| 6 | 1.00 | 1.00 | 1.00 | 537 |
| accuracy | | | 0.93 | 2947 |
| macro avg | 0.94 | 0.93 | 0.93 | 2947 |
| weighted avg | 0.94 | 0.93 | 0.93 | 2947 |

(a) GradientBoostingClassifier

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 1 | 0.95 | 0.99 | 0.97 | 496 |
| 2 | 0.93 | 0.94 | 0.94 | 471 |
| 3 | 0.98 | 0.93 | 0.95 | 420 |
| 4 | 0.93 | 0.86 | 0.90 | 491 |
| 5 | 0.88 | 0.95 | 0.91 | 532 |
| 6 | 1.00 | 1.00 | 1.00 | 537 |
| accuracy | | | 0.95 | 2947 |
| macro avg | 0.95 | 0.94 | 0.94 | 2947 |
| weighted avg | 0.95 | 0.95 | 0.94 | 2947 |

(b) HistGradientBoostingClassifier

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 1 | 0.93 | 0.98 | 0.96 | 496 |
| 2 | 0.94 | 0.93 | 0.93 | 471 |
| 3 | 0.98 | 0.93 | 0.95 | 420 |
| 4 | 0.93 | 0.84 | 0.88 | 491 |
| 5 | 0.87 | 0.94 | 0.90 | 532 |
| 6 | 1.00 | 1.00 | 1.00 | 537 |
| accuracy | | | 0.94 | 2947 |
| macro avg | 0.94 | 0.94 | 0.94 | 2947 |
| weighted avg | 0.94 | 0.94 | 0.94 | 2947 |

(c) LXGBClassifier

Figure 24: Classification reports for gradient boosting

Between the classifiers and parameter combinations tested, the one giving the higher performance is the *HistGradientBoosting* classifier with an accuracy of 0.95.

# 8 Regression

## 8.1 Univariate Linear Regression

For simple linear univariate regression, *tBodyAcc-entropy()-X* was used as the target variable, and *tBodyAcc-arCoeff()-X,4* was used as the independent variable.

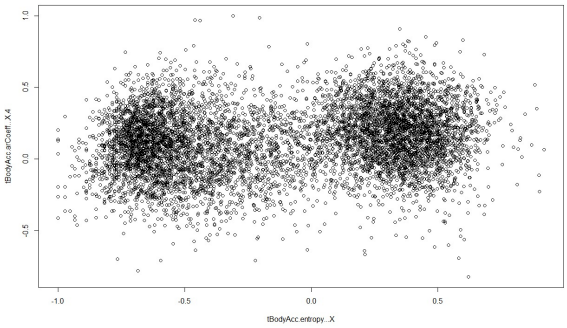
```
lm(formula = tBodyAcc.entropy...X ~ ., data = X1)

Residuals:
    Min      1Q  Median      3Q     Max
-0.9358 -0.4517  0.0465  0.4126  1.1507

Coefficients:
                         Estimate Std. Error t value Pr(>|t|)
(Intercept)            -0.157024   0.006211  -25.28   <2e-16 ***
tBodyAcc.arCoeff...X.4  0.406091   0.023524   17.26   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.4597 on 7276 degrees of freedom
Multiple R-squared:  0.03935,   Adjusted R-squared:  0.03922
F-statistic:   298 on 1 and 7276 DF,  p-value: < 2.2e-16
```

(a) Regression summary




(b) 2D representation

Figure 25: Simple linear univariate regression

The variable x is statistically significant for the model given that it has a lower p-value than alpha. The presence of low $R^2$ indicates that the two variables are not correlated; on average a unit change in *tBodyAcc-arCoeff()-X,4* results in a 0.4 increase in *tBodyAcc-entropy()-X*. Also, from the scatterplot, it can be seen that there is no linear relationship between the two variables.

## 8.2 Multiple Linear Regression

A multiple linear regression was carried out on the entire dataset, i.e. on all 151 variables. Then, having identified the statistically significant variables for an alpha value of 0.05, a reduced model was created, which was subsequently compared with the complete model. Three measures (AIC, BIC, ANOVA) were used in the comparison. The results indicated the reduced model as the best of the two. $R^2$ is equal to 0.90 for both models.

Then, the Breusch & Pagan test (hypothesis $H_0$ homoschedasticity of the data) was performed on the reduced regression model which showed that the data are heteroschedastic as the variance of the errors is not equal to a constant, it is a quantity that varies as the statistical units vary. The variance $\sigma_i$ depends on $i$. To say that $\sigma_i^2$ is no longer a constant is to say that $\sigma_i^2$ has its own model that goes beyond the MLR model; it has its own structure that depends on external and internal variables. To estimate the variance, it was used a linear regression model called Auxiliary Regression derived from the estimation of $e_i$ (random errors of the regression model). Attributing appropriately calculated values to $e_i^2$ allows us to derive estimates for unknown parameters that are useful in deriving an estimate for $\sigma_i^2$.

Once the auxiliary regression model has been estimated, it is necessary to find out whether or not heteroschedasticity really exists in the data. If there is heteroschedasticity a linear dependency link exists between

$e_i^2$ and the variables involved in the regression. Therefore the significance of the parameters is tested (BP test). If you do not reject H0 then it means that the parameters are statistically null, so $e_i^2$ does not depend on the variables specified in the model, hence there is no heteroschedasticity. If at least one parameter is non-zero, it means that the variance actually depends on one or more variables, so there will be heteroschedasticity. The result of the BP test on the auxiliary regression created reports a p-value greater than alpha, so the $H_0$ (null hypothesis) is not rejected and, therefore, contrary to what had initially emerged, there is no heteroschedasticity.

```
        studentized Breusch-Pagan test

data:  reg.aus
BP = 82.855, df = 82, p-value = 0.4528
```

Figure 26: BP test

The next phase of the analysis involves the use of Lasso, Ridge and Elastic Net Regressions (with alpha values of 0.2, 0.4, 0.6, 0.8) using both *lambda min* and *lambda 1se*.

| | RMSE | Rsquare |
|---|---|---|
| en 1 | 0.1573095 | 0.8755702 |
| en 0.8 | 0.1575043 | 0.8753062 |
| en 0.6 | 0.1572412 | 0.8756412 |
| en 0.4 | 0.1573171 | 0.8755498 |
| en 0.2 | 0.1573389 | 0.8755096 |
| en 0 | 0.1595820 | 0.8716975 |

(a) EN with lambda min

| | RMSE | Rsquare |
|---|---|---|
| en 1_1se | 0.1576445 | 0.8749367 |
| en 0.8_1se | 0.1577303 | 0.8747894 |
| en 0.6_1se | 0.1577888 | 0.8746846 |
| en 0.4_1se | 0.1578302 | 0.8746162 |
| en 0.2_1se | 0.1579696 | 0.8743694 |
| en 0 | 0.1604642 | 0.8703434 |

(b) EN with lambda 1se

| | |
|---|---|
| n_1_min | 11 |
| n_1_se | 57 |
| n_8_min | 3 |
| n_8_se | 53 |
| n_6_min | 14 |
| n_6_se | 51 |
| n_4_min | 10 |
| n_4_se | 50 |
| n_2_min | 9 |
| n_2_se | 44 |

(c) Shrinkage penalty

Figure 27: Elastic Net performance evaluation

Given that the results were similar between Lasso and Elastic Net, the regressor with the greatest shrinkage penalty was preferred. Figure 27 shows that using a *lambda 1se*, the Lasso regression is the one which penalises the most (57 variables set to zero).

Finally, using the variables derived from the Lasso Regression shrinkage, a final multiple linear regression model was created which, once defined, was compared with the initial reduced regression model. The two give roughly the same results, the reduced model being slightly better as it has a smaller RMSE value (0.26 compared to the 0.28 of Lasso).

The last test applied is the Jarque & Bera's test on the residuals of the reduced model. As the p-value is smaller than alpha, it is concluded that the model does not fit the data well.

```
        Jarque Bera Test

data:  reg_lm_lass$residuals
X-squared = 276.55, df = 2, p-value < 2.2e-16
```

Figure 28: Jarque & Bera's test

### 8.3 Gradient Boosting Machines for Regression

Now, the same regression problem, on the *tBodyAcc-entropy()-X* variable, will be applied by using Gradient Boosting. The *GradientBoostingRegressor* function with *loss_function* = 'squared_error', *learning_rate* = 0.1, *n_estimators* = 100 and *max_depth* = 5 yielded the following performance measures on the test test: $R^2$ = 0.949, MSE = 0.010 and MAE = 0.070. Other parameters were tested, however this was the best solution obtained. Comparing the two models, the $R^2$ and MSE scores are better for the Gradient Boosting regression.

## 9 Time Series Analysis

Time Series analysis was made using data collected from 3 sensors (*Total Acceleration*, *Body Acceleration*, *Body Gyroscope*) on 3 axes ($x$, $y$, $z$), used in the form of time series. There are 9 files, each containing 7352 observations per 128 variables indicating the time stamps, therefore each record is a time series.

For the following analysis two dataset will be used, a univariate one and a multivariate one. The univariate data corresponds to the *Total_acc_x* file, chosen due to the higher performance on a starter KNN than the other ones. Whereas the multivariate data includes all the 9 files.

### 9.1 Simple Classification with KNN

A common task for time series machine learning is classification. As the first algorithm the K-Nearest Neighbors classifier was used on both datasets with Euclidean and Dynamic Time Warping (DTW) distances.

### 9.1.1 Univariate KNN

For the univariate case the results reported are the ones obtained by considering the dataset containing the *Total_acc_x* variable. The Euclidean distance metric is not suitable for time series data because it ignores the time dimension of the data. On the other hand, the DTW distance metric measures the similarity between two subsequences that may not align exactly in time, speed, or length. can be observed from the KNN classifiers' outputs that the better results are the ones with DTW. Such inconsistency of the Euclidean distance can also be observed from the results of the classifier which performs better with the DTW (DTW accuracy = 0.77; Euclidean accuracy = 0.60).

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 1 | 0.78 | 0.84 | 0.81 | 496 |
| 2 | 0.63 | 0.77 | 0.70 | 471 |
| 3 | 0.69 | 0.82 | 0.75 | 420 |
| 4 | 0.37 | 0.22 | 0.28 | 491 |
| 5 | 0.47 | 0.32 | 0.38 | 532 |
| 6 | 0.54 | 0.69 | 0.61 | 537 |
| accuracy | | | 0.60 | 2947 |
| macro avg | 0.58 | 0.61 | 0.59 | 2947 |
| weighted avg | 0.58 | 0.60 | 0.58 | 2947 |

(a) Euclidean distance

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 1 | 0.84 | 0.97 | 0.90 | 496 |
| 2 | 0.75 | 0.80 | 0.77 | 471 |
| 3 | 0.93 | 0.70 | 0.80 | 420 |
| 4 | 0.56 | 0.57 | 0.57 | 491 |
| 5 | 0.60 | 0.59 | 0.59 | 532 |
| 6 | 1.00 | 1.00 | 1.00 | 537 |
| accuracy | | | 0.77 | 2947 |
| macro avg | 0.78 | 0.77 | 0.77 | 2947 |
| weighted avg | 0.78 | 0.77 | 0.77 | 2947 |

(b) DTW distance

Figure 29: Univariate KNN with k = 5

### 9.1.2 Multivariate KNN

In the multivariate setting, as expected, the performance of the KNN classifier improves, especially by considering the DTW distance. There is an increase in the accuracy of 0.11.

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 1 | 0.96 | 0.53 | 0.68 | 496 |
| 2 | 0.96 | 0.54 | 0.69 | 471 |
| 3 | 0.97 | 0.17 | 0.29 | 420 |
| 4 | 0.61 | 0.78 | 0.68 | 491 |
| 5 | 0.35 | 0.77 | 0.48 | 532 |
| 6 | 1.00 | 1.00 | 1.00 | 537 |
| accuracy | | | 0.65 | 2947 |
| macro avg | 0.81 | 0.63 | 0.64 | 2947 |
| weighted avg | 0.80 | 0.65 | 0.65 | 2947 |

(a) Euclidean distance

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 1 | 0.97 | 0.91 | 0.94 | 496 |
| 2 | 0.99 | 0.95 | 0.97 | 471 |
| 3 | 1.00 | 0.82 | 0.90 | 420 |
| 4 | 0.77 | 0.79 | 0.78 | 491 |
| 5 | 0.66 | 0.80 | 0.72 | 532 |
| 6 | 1.00 | 1.00 | 1.00 | 537 |
| accuracy | | | 0.88 | 2947 |
| macro avg | 0.90 | 0.88 | 0.88 | 2947 |
| weighted avg | 0.89 | 0.88 | 0.88 | 2947 |

(b) DTW distance

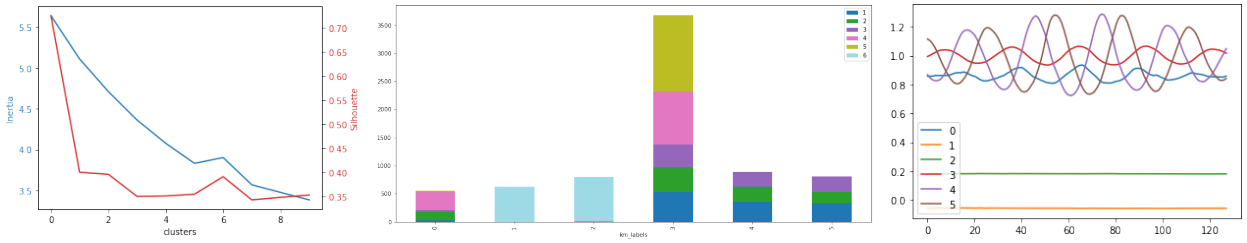
Figure 30: Multivariate KNN with k = 5

## 9.2 Clustering

In this section unsupervised clustering will be implemented with the K-Means algorithm. In general, time series clustering identifies distinct clusters in which its members have similar time series characteristics. This task was performed on the univariate dataset and on a SAX approximated one by considering both the Euclidean and DTW similarity measures.

### 9.2.1 K-Means on the univariate dataset

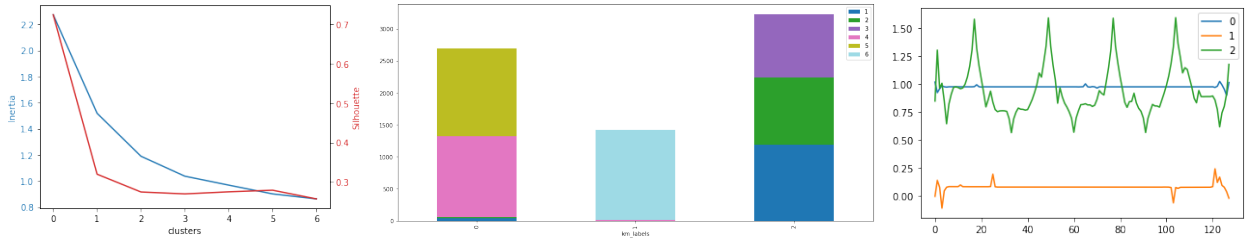
**Euclidean** Considering the Euclidean distance, the SSE and Silhouette measures were calculated for different values of k (Figure 31a), which was then chosen to be 6. In Figure 31b are represented the clusters with respect to the class label. From their analysis, it could be possible to think of obtaining a smaller number of clusters by grouping them into 3 (i.e. 1 & 4; 2 & 3; 5 & 6). The centroids of the 6 clusters are represented in Figure 31c.




(a) SSW and Silhouette (b) Clusters wrt the class label (c) Centroids

Figure 31: Kmeans with Euclidean distance

**DTW** The same is done with the DTW distance. By considering the elbow method for SSE and the not too different silhouette values it was chosen k = 3, in conjunction with the previous analysis. The clusters, as shown in Figure 32b, are well divided with respect to the different classes. Cluster 0 contains classes 4 (SITTING) and 5 (STANDING); Cluster 1 contains class 6 (LAYING); Cluster 2 contains classes 1 (WALKING), 2 (WALKING UPSTAIRS) and 3 (WALKING DOWNSTAIRS). The centroids are reported in Figure 32c.




(a) SSW and Silhouette (b) Clusters wrt the class label (c) Centroids

Figure 32: Kmeans with DTW distance

### 9.2.2 K-Means on the univariate SAX approximated dataset

Due to the complexity generated from the usage of the DTW metric, approximation techniques were introduced. Four types of clusterings were implemented by using two different types of approximation: Symbolic Aggregate Approximation (SAX) and 1D Symbolic Aggregate Approximation (1D-SAX). The number of segments chosen for the Piecewise Aggregate Approximation (PAA) is 30, the number of SAX symbols is 8, the number of average 1D-SAX symbols is 8 and the number of slope 1D-SAX symbols is 4. In Figure 33 is reported an example of the approximations on the first time series in the dataset.

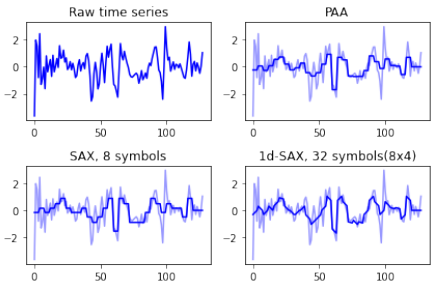
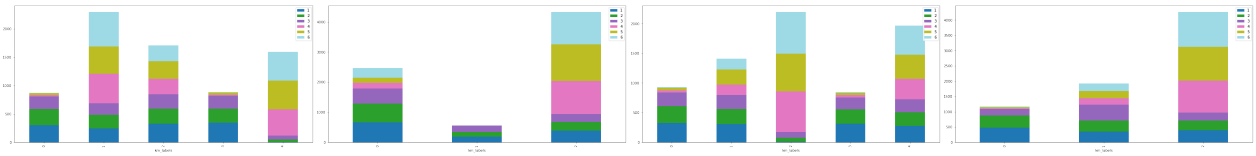



Figure 33: Approximation example

From the approximated data, the clusterings generated present heterogeneous clusters, each cluster contains indiscriminately observations for every class. It can be deduced that, by using these approximated versions of the dataset, K-Means is not able to distinguish between the different activities. For each combination, the SSE and Silhouette curves were calculated to then choose the value of k. The different clusterings are shown in Figure 34.



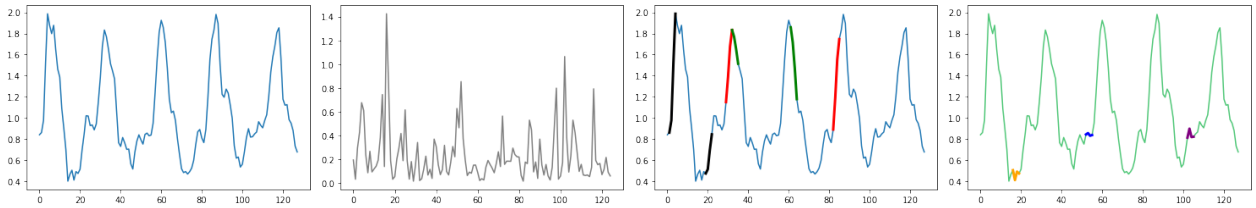
(a) SAX & Euclidean (b) SAX & DTW (c) 1D-SAX & Euclidean (d) 1D-SAX & DTW

Figure 34: K-Means clusters wrt the class label

## 9.3 Motifs & Discords Discovery

For the Motifs & Discords Discovery the two most dissimilar time series were taken into consideration (ts 953 e 5138) for the *total_acc_x* variable. This two records were then also analysed with respect to the other two sensors on the same axis.

**Total_acc_x** TS 953

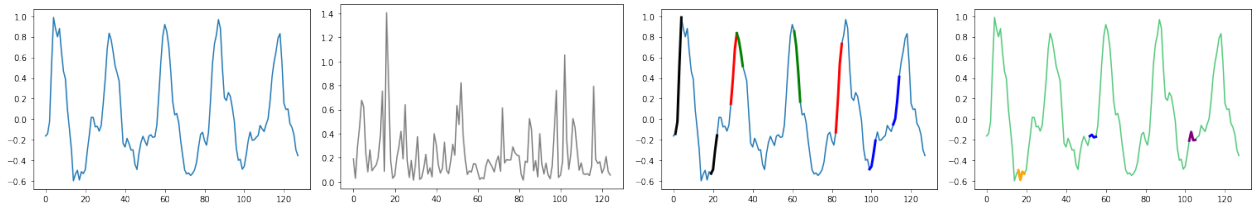


(a) Time series (b) Matrix Profile (c) Motifs (d) Discords

Figure 35: Total_acc_x TS 953

From the above representation, three motifs, which are associated to the valleys in the matrix profile, and three discords, which are associated to the picks, are discovered.

**Body_acc_x** TS 953

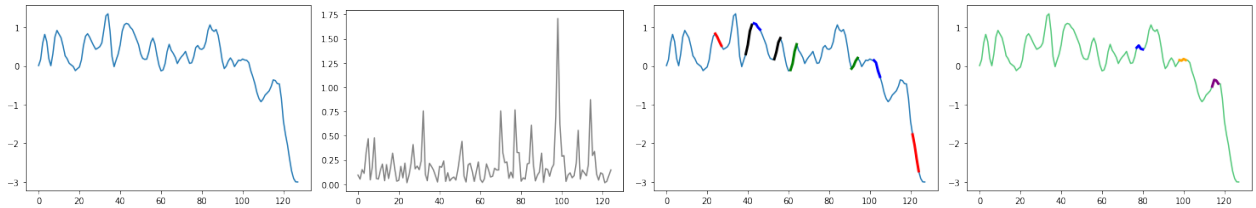


(a) Time series (b) Matrix Profile (c) Motifs (d) Discords

Figure 36: Boby_acc_x TS 953

From the above representation four motifs and three discords are discovered.

**Body_gyro_x** TS 953

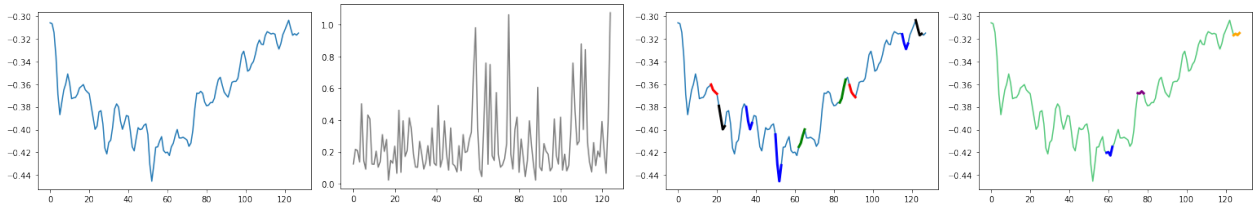


(a) Time series (b) Matrix Profile (c) Motifs (d) Discords

Figure 37: Boby_gyro_x TS 953

There are four motifs and three discords.

**Total_acc_x** TS 5138

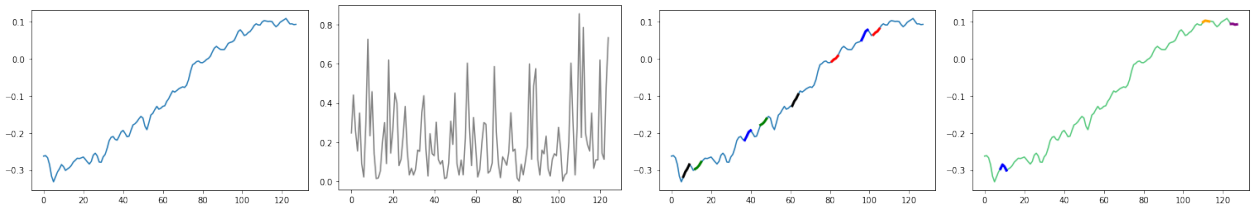


(a) Time series (b) Matrix Profile (c) Motifs (d) Discords

Figure 38: Total_acc_x TS 5138

There are four motifs and three discords.

**Body_acc_x** TS 5138

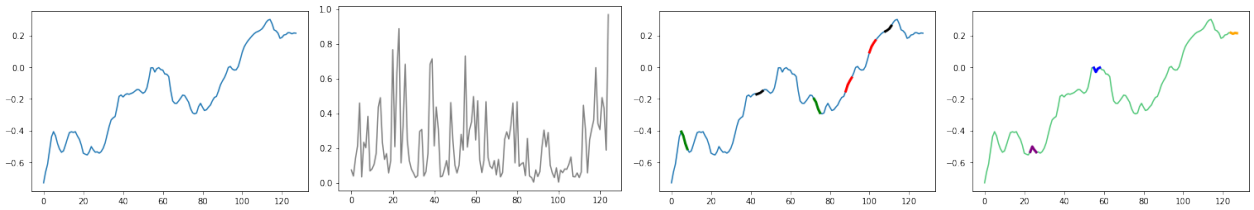


(a) Time series (b) Matrix Profile (c) Motifs (d) Discords

Figure 39: Boby_acc_x TS 5138

There are four motifs and three discords.

**Body_gyro_x** TS 5138

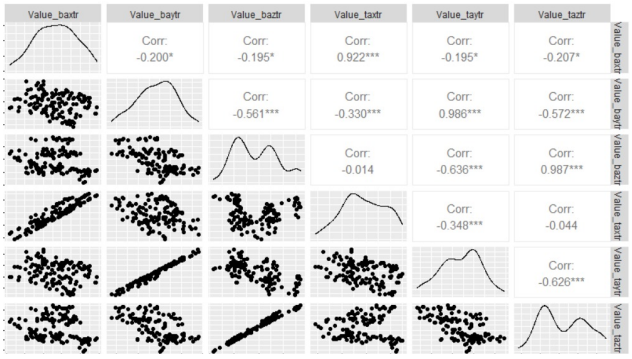


(a) Time series (b) Matrix Profile (c) Motifs (d) Discords

Figure 40: Boby_gyro_x TS 5138

There are three motifs and three discords.

From the analysis of the motifs and components of the time series in the two variables *Total_acc_x* and *Body_acc_x*, it could be possible to deduce that there will be a linear dependence between the two sensors.




(a) Correlation

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 1 | 0.93 | 0.90 | 0.91 | 496 |
| 2 | 0.99 | 0.91 | 0.95 | 471 |
| 3 | 0.99 | 0.74 | 0.85 | 420 |
| 4 | 0.78 | 0.80 | 0.79 | 491 |
| 5 | 0.62 | 0.80 | 0.70 | 532 |
| 6 | 1.00 | 1.00 | 1.00 | 537 |
| accuracy | | | 0.86 | 2947 |
| macro avg | 0.88 | 0.86 | 0.87 | 2947 |
| weighted avg | 0.88 | 0.86 | 0.87 | 2947 |

(b) KNN with DTW on 6 variables

Figure 41

Once calculated the pairwise correlation of the variables, it is clear that there is linear dependence between the sensors Total Acceleration and Body Acceleration.

Uncovered this new information, it was decided to implement again the KNN classifier with DTW by excluding the Body Acceleration sensor. The output is reported in Figure 41b. The accuracy is 0.86, which is only 0.02 less than the one on the original dataset. For this reason, it is preferred to the previous one given the lower complexity.

## 9.4 Advanced Classification

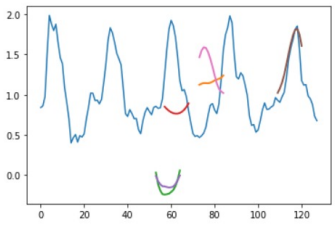
Standard machine learning algorithms are not always well suited to work on raw time series data. Many different categories of algorithms have been investigated to tackle this specific data type. Some consist in extracting features from time series that can then be used as input of a standard machine learning classifier (shapelet-based classifiers), while others work on raw time series. For this analysis different types of classifiers will be implemented.

### 9.4.1 Shapelet-Based Classifiers

A shapelet is a subsequence of a time series which is highly discriminative to recognise among different classes.

**Learning-Based Approach** .

The first step is to calculate the number and length of the shapelets with the grabocka-params-to-shapelet-size-dict function, the results obtained are: shapelet sizes 12: 6. Then it is used the function ShapeletModel with parameters: *n_shapelet_sizes* = 12: 6, *optimizer* = 'sgd', *weight_regularizer* = .01, *max_iter* = 200 and *verbose* = 1. The shapelets found are represented in Figure 42a together with the record 953. The results obtained from this classifier are shown in Figure 42b.




(a) Shapelet representation on TS 953

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 1 | 0.28 | 0.63 | 0.39 | 496 |
| 2 | 0.00 | 0.00 | 0.00 | 471 |
| 3 | 1.00 | 0.00 | 0.01 | 420 |
| 4 | 0.94 | 0.10 | 0.18 | 491 |
| 5 | 0.45 | 1.00 | 0.62 | 532 |
| 6 | 1.00 | 1.00 | 1.00 | 537 |
| accuracy | | | 0.49 | 2947 |
| macro avg | 0.61 | 0.46 | 0.37 | 2947 |
| weighted avg | 0.61 | 0.49 | 0.39 | 2947 |

(b) Classification report

Figure 42: Learning Based Approach

**Shapelet Distance-Based Classifier** .

This classification approach consists in calculating the distances between each shapelet and time series, which are then stored in a new dataset. The classifiers implemented are the KNN (with k = 5) and the Decision Tree (with *max_depth* = 8).

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 1 | 0.68 | 0.91 | 0.78 | 496 |
| 2 | 0.59 | 0.51 | 0.55 | 471 |
| 3 | 0.81 | 0.62 | 0.70 | 420 |
| 4 | 0.58 | 0.49 | 0.53 | 491 |
| 5 | 0.59 | 0.67 | 0.62 | 532 |
| 6 | 1.00 | 1.00 | 1.00 | 537 |
| accuracy | | | 0.71 | 2947 |
| macro avg | 0.71 | 0.70 | 0.70 | 2947 |
| weighted avg | 0.71 | 0.71 | 0.70 | 2947 |

(a) KNN

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 1 | 0.71 | 0.87 | 0.78 | 496 |
| 2 | 0.60 | 0.57 | 0.59 | 471 |
| 3 | 0.77 | 0.61 | 0.68 | 420 |
| 4 | 0.69 | 0.27 | 0.39 | 491 |
| 5 | 0.57 | 0.88 | 0.69 | 532 |
| 6 | 1.00 | 1.00 | 1.00 | 537 |
| accuracy | | | 0.71 | 2947 |
| macro avg | 0.72 | 0.70 | 0.69 | 2947 |
| weighted avg | 0.72 | 0.71 | 0.70 | 2947 |

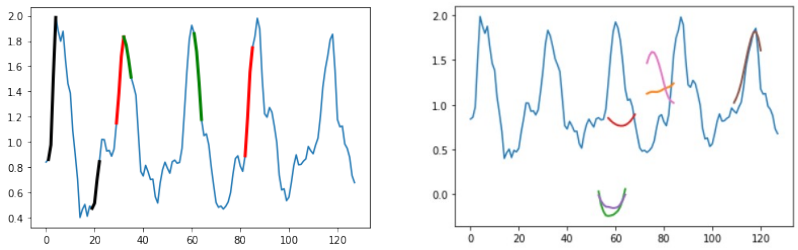
(b) Decision Tree

Figure 43: Shapelet Distance-Based Classifier classification reports

Both classifier have an accuracy of 0.71, which is much higher than the one of the previous approach.

**Similarities on Shapelets & Motifs** .

From a graphic analysis of the shapelets and the motifs of time series 953, it can be noticed that the motifs do not follow the trend of the shapelets with the exception of the last one. Since it is the only shapelet findable in the motifs it could be defined as the class corresponding to the time series, in this case class 3 (WALKING DOWNSTAIRS).



(a) Motifs (b) Shapelets

Figure 44: Time series 953

### 9.4.2 ROCKET

Random Convolutional Kernel Transform (ROCKET) extracts features from time series by using a large number of random convolutional kernels, meaning that all the parameters of all the kernels (length, weights, bias, dilation, and padding) are randomly generated from fixed distributions. The transformed features are then used to train a linear classifier, which in this case is a Ridge Regression. By implementing this classifier with the multivariate dataset as input data, the accuracy is 0.94, which is the higher obtained so far. The complete output is reported in Figure 45.

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 1 | 1.00 | 1.00 | 1.00 | 496 |
| 2 | 1.00 | 0.97 | 0.98 | 471 |
| 3 | 1.00 | 1.00 | 1.00 | 420 |
| 4 | 0.82 | 0.91 | 0.86 | 491 |
| 5 | 0.90 | 0.92 | 0.91 | 532 |
| 6 | 0.95 | 0.87 | 0.91 | 537 |
| accuracy | | | 0.94 | 2947 |
| macro avg | 0.94 | 0.94 | 0.94 | 2947 |
| weighted avg | 0.94 | 0.94 | 0.94 | 2947 |

Figure 45: ROCKET classification report

### 9.4.3 MiniROCKET

Several extensions of ROCKET have been proposed in the literature. One is MiniROCKET, which reduces the randomness of the parameters of the kernels by using a fixed value or sampling from smaller distributions. These modifications allow for more optimization and lead to a much lower computational complexity while maintaining similar performance. The output obtained on the multivariate dataset is the one in Figure 46. The accuracy is 0.98, which is even higher than ROCKET one.

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 1 | 1.00 | 1.00 | 1.00 | 496 |
| 2 | 1.00 | 1.00 | 1.00 | 471 |
| 3 | 1.00 | 1.00 | 1.00 | 420 |
| 4 | 0.96 | 0.90 | 0.93 | 491 |
| 5 | 0.91 | 0.96 | 0.94 | 532 |
| 6 | 1.00 | 1.00 | 1.00 | 537 |
| accuracy | | | 0.98 | 2947 |
| macro avg | 0.98 | 0.98 | 0.98 | 2947 |
| weighted avg | 0.98 | 0.98 | 0.98 | 2947 |

Figure 46: MiniROCKET classification report

# 10 Sequential Pattern Mining

Sequential Pattern Mining helps to extract the sequences which reflect the most frequent behaviours in the sequence dataset, which in turn can be interpreted as domain knowledge for several purposes. For this part of the study the *total_acc_x* sensor was used as in the previous section. To convert the time series data into a discrete format it was used SAX (Symbolic Aggregate Approximation) with 20 symbols and 40 segments. The parameters used for the *PrefixSpan* function are: *length of the subsequence* = 14 and *support* = 5000. With this setting two sequential patters are found in the data.

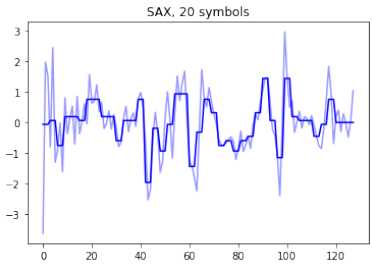


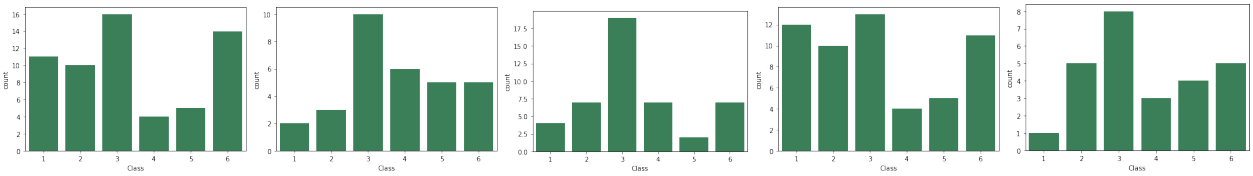

Figure 47: Example of SAX approximated time series

```
With minlen=14, maxlen=14, support=5000,
I found 2 frequent patterns. Here you go:
[(5009, [3, 3, 3, 4, 4, 4, 10, 10, 10, 10, 10, 10, 10, 10]),
(5006, [6, 6, 6, 7, 7, 7, 10, 10, 10, 10, 10, 10, 10, 10])]
```

Given the few sequential patterns found, it was decided to just print the top ten most frequent ones, with still a length of 14.

```
[(5009, [3, 3, 3, 4, 4, 4, 10, 10, 10, 10, 10, 10, 10, 10]),
 (5006, [6, 6, 6, 7, 7, 7, 10, 10, 10, 10, 10, 10, 10, 10]),
 (4991, [4, 4, 4, 4, 4, 4, 10, 10, 10, 10, 10, 10, 10, 10]),
 (4987, [3, 3, 3, 3, 3, 3, 10, 10, 10, 10, 10, 10, 10, 10]),
 (4987, [6, 6, 6, 4, 4, 4, 10, 10, 10, 10, 10, 10, 10, 10]),
 (4982, [5, 5, 5, 5, 5, 5, 10, 10, 10, 10, 10, 10, 10, 10]),
 (4980, [4, 4, 4, 5, 5, 5, 10, 10, 10, 10, 10, 10, 10, 10]),
 (4975, [6, 6, 6, 5, 5, 5, 10, 10, 10, 10, 10, 10, 10, 10]),
 (4972, [4, 4, 4, 7, 7, 7, 10, 10, 10, 10, 10, 10, 10, 10]),
 (4963, [4, 4, 4, 3, 3, 3, 10, 10, 10, 10, 10, 10, 10, 10])]
```

From these first ten frequent sequential patterns it can be seen that the last eight symbols are equal to 10 for each one of them. Interpreting these sequential patterns might not be feasible because they are patterns of symbols extracted from motion sensors' time series. One way to get information from sequential pattern mining might be to interpret the specific patterns found. Taking the fist five sequential patterns considering them as uninterrupted sequence, it was checked in how many records they were (Pattern 1 is present in 60 TS; Pattern 2 is present in 31 TS; Pattern 3 is present in 46 TS; Pattern 4 is present in 55 TS; Pattern 5 is present in 26 TS). For each pattern a barplot with respect to the class is reported below.



(c) Pattern 1 (d) Pattern 2 (e) Pattern 3 (f) Pattern 4 (g) Pattern 5

Figure 48: Frequency of the frequent sequences with respect to the class

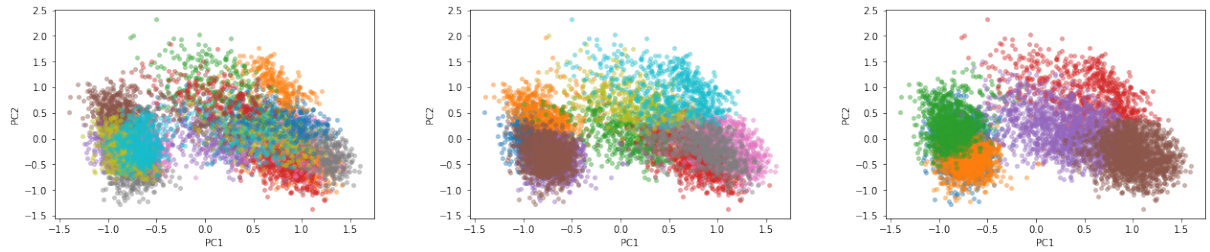
Patterns 1, 2, 3 and 5 belong mainly to records of the third class, whereas for pattern 4 the they belong to classes 1, 2, 3 and 6.

# 11 Advanced Clustering

In this section advanced classification techniques, namely X-Means and OPTICS, are implemented. X-Means on the 151-dimensional dataset and on the first 10 Principal Components (obtained through PCA) of the 561-dimensional dataset, whereas OPTICS only on the first 10 PC of the 561-dimensional dataset to simplify the computational complexity of the algorithm. The data was normalized with *MinMaxScaler*.

## 11.1 X-Means

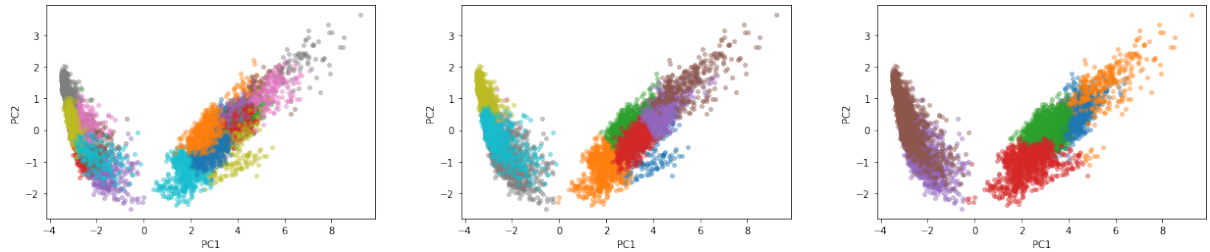
Three different values of K were considered: K = 20, K = 10 and K = 6. Initially the algorithm was executed on the 151-dimensional dataset, obtaining the following clusterings (Figura 49).




(a) K = 20 (b) K = 10 (c) K = 6

Figure 49: Clusterings on the 151-dimensional dataset

Then it was decided to run the algorithm also on the first 10 PC of the 561-dimensional dataset, which retained more than 80% of the total variability of the data. The clusterings are reported in Figure 50.




(a) K = 20 (b) K = 10 (c) K = 6

Figure 50: Clusterings on the first 10 PC of the 561-dimensional dataset

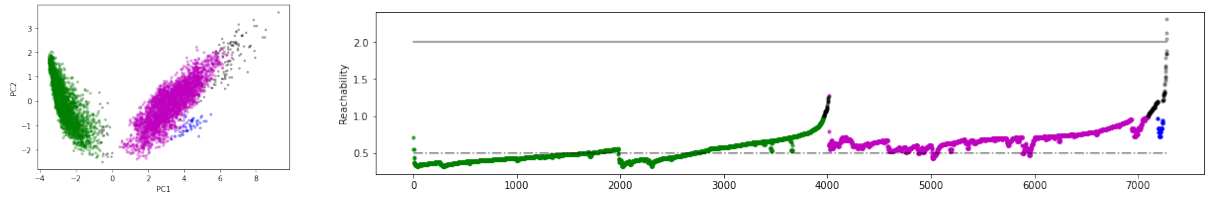
The WCE and Average Silhouette score were computed for all the clusterings obtained (Table 2). The ones using PCA attain better results for both measures. The result that better satisfies a trade-off between WCE and Silhouette is the clustering with PCA and K = 10.

| | K | WCE | Silhouette |
|---|---|---|---|
| X-Means | 20 | 14861.28 | 0.11 |
| | 10 | 16236.02 | 0.12 |
| | 6 | 17595.79 | 0.14 |
| X-Means with PCA | 20 | 8090.04 | 0.34 |
| | 10 | 10806.19 | 0.39 |
| | 6 | 14689.44 | 0.42 |

Table 2

## 11.2 OPTICS

The cluster method chosen for this algorithm is dbscan. Several tests with different values of *min_samples* and *eps* were carried out. The final configuration is with *min_samples* = 15 and *eps* = 1.0. The clusters obtained are three (cluster 0: 3964 observations; cluster 1: 3086 observations; cluster 2: 48 observations) plus 180 outliers (Figure 51a). Cluster 0 and Cluster 1 seem to represent the division between stationary and non-stationary tasks. In Figure 51b it is represented the reachability distance. The average silhouette score is 0.41.




(a) Clustering (b) Reachability plot

Figure 51: OPTICS on the first 10 PC of the 561-dimensional dataset

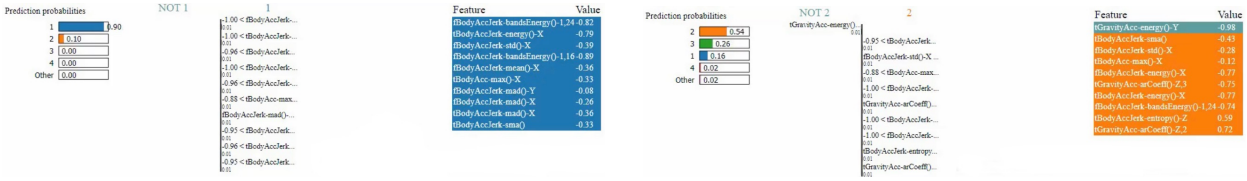
# 12 Explainability

In recent years, the statistical assumptions have become lighter and the power in the analyst's hands has been reduced due to the introduction of more and more complex models, the so-called black box models. With this term, it is referred to those models whose internals are either unknown to the observer or they are known but uninterpretable by humans. The most recent machine learning models give better results than the classical statistical models, that is, they have greater interpretability. However, if we are interested in explaining why a certain variable is more important than another in the classification, it becomes fundamental the use of explainability.

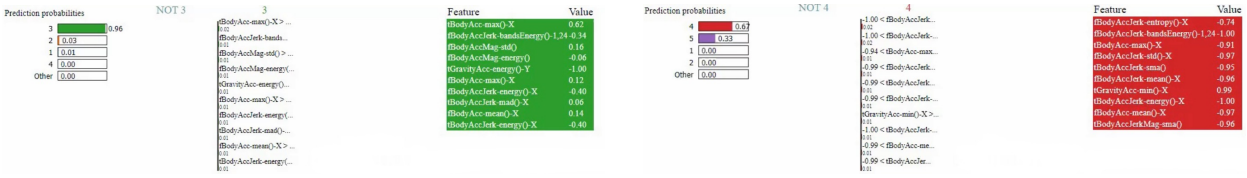
The two explainability techniques used are LIME and SHAP. The black box model chosen for this task is the Random Forest classifier previously implemented with *max_depth* = 20 and *n_estimators* = 100.

## 12.1 LIME

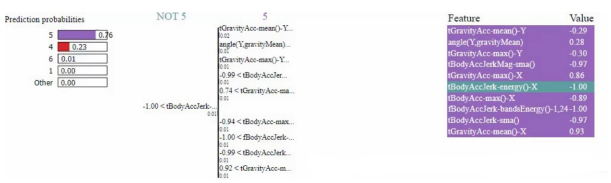
LIME is a local and agnostic explainability technique that takes any machine learning model as input and returns an explanation about the contribution of each feature in determining the class in the form of a score. The following representations contain the explainability for a record of each class of the target variable.




(a) Record of class 1 (b) Record of class 2




(c) Record of class 3 (d) Record of class 4




(e) Record of class 5


(f) Record of class 6

Figure 52: LIME on different records

## 12.2 SHAP

SHAP is an explainability technique that assigns an importance value to each feature that represents their impact on the model prediction. The difference with LIME is that it tests several subgroups of features holding one of them fixed and calculates the average marginal contribution of the fixed variable, to then return the explanation in the form of feature importance. The following charts are a density plot and a bar plot respectively showing the top ten most significant SHAP values.

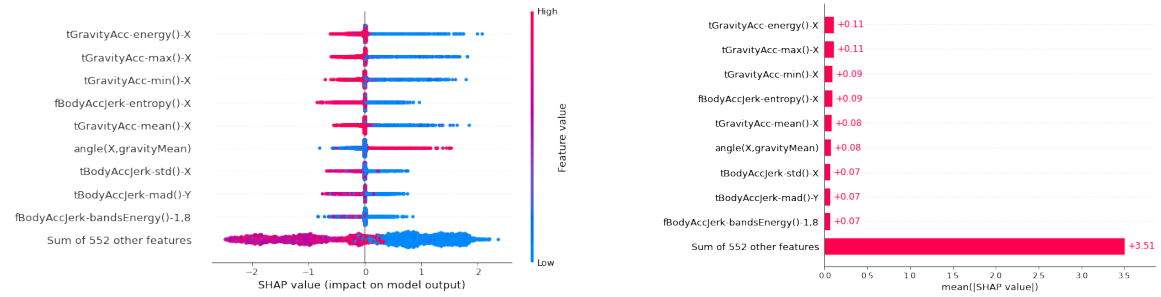



Figure 53: SHAP

From Figure 53 it can be inferred that there is no variable having a marginal contribution far superior than the others. *tGravityAcc-energy()-X* and *tGravityAcc-max()-X* have a slightly higher shapely value than the others.

Comparing these results with the feature importance derived from the Random Forest classifier in Figure 21c, it can be observed that they have five variables in common in their top ten.