
UNIVERSITY OF PISA

DEPARTMENT OF INFORMATICS

DATA MINING

# Human Activity Recognition Using Smartphones Dataset

Chiara De Nigris (586013)

Chiara Giurdanella (560686)

Simona Sette (544298)

ACADEMIC YEAR 2021/2022

# CONTENTS

# 1. Data understanding and preparation

*Human Activity Recognition (HAR) Using Smartphones* dataset is a collection of data about the movements of 30 people performing different activities, recorded using smartphones tied to their waists with the help of sensors (accelerometer and gyroscope). The target variable is the feature Activity, composed by six different human activities: *Laying*, *Sitting*, *Standing*, *Walking*, *Walking Downstairs* and *Walking Upstairs,* each labeled with a value from 1 to 6. Each feature vector is a row on the set. The dataset is composed as shown in Table 1.

| | *Data type* | *Range of value* | *Variable type* |
|---|---|---|---|
| A 561-feature vector with time and frequency domain variables. | Float | From – 1 to 1 | Numerical |
| Its activity label. | Int | From 1 to 6 | Categorical (ordinal) |

*Table 1* Composition of HAR dataset

The dataset has been split in three sections: the train set, which has been used to train the models, the validation set (equal to the 30% of the train), used to define the optimal parameters tuning of the models and the test set (which in this case has been already provided by the creators of the dataset as a different file), used to test the model after completing the training.

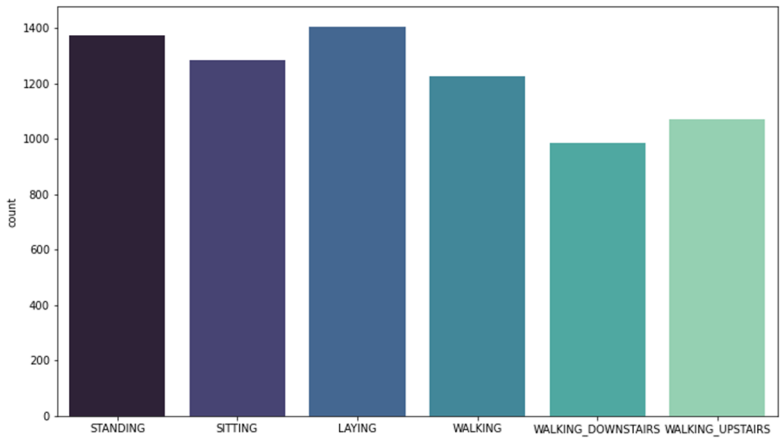



*Figure 1* Activities distribution

The data preparation phase has been focused on testing the conformity of the data set in order to perform deeper analysis. From the moment that the features have been already bounded in the range [-1,1], it hasn't been necessary any kind of normalization on the data.

Moreover, the set has shown no duplicated instances or missing values and a balanced distribution of the six activities (as shown in Figure 1), which have been chosen as target for the following classification tasks.

## 1.1 Outliers Detection

Afterwards, the analysis has been centered on outlier detection and on the treatment of anomalies spotted within the data. In particular, the goal has been to detect the top 1% of the outliers, using four different algorithms, representative of different approaches: *DB-Scan* and *LOF* (Local Outlier Factor) as density-based approaches, *ABOD* (Angle-Based Outlier Degree) as angle-based approach and *Isolation Forest* as model-based approach. The first method is the only labeling approach whereas the other three are scoring ones.

For what *DB Scan* is concerned, the knee method has been used to detect the best values for the parameters tuning, returning as optimal parameters *min points* equal to 10 and *epsilon* equal to 5. Remembering that *DB Scan* is mostly a clustering approach and not an outlier detection method per-se, outliers have been identified as the points labeled as -1, which are 185.

Taking into consideration *LOF*, after different trials to define the best parameters tuning, optimal results have shown setting the number of neighbors equal to 20, as default, and contamination at 0.05.
Considering the *LOF negative_outlier_factor*, 368 outliers were detected from which, rounding up, 4 have been considered for the analysis as the 1% and are shown in Table 2.

| *Index* | *LOF score* |
|---|---|
| 70 | -1.76 |
| 71 | -1.75 |
| 1904 | -1.74 |
| 4730 | -1.73 |

*Table 2* Top 1% outliers found using *LOF*

The third algorithm used has been *ABOD*, a high-dimensional approach and angle-based method which detects potential outliers by considering the variances of the angles between the data points. For the parameters tuning, the optimal values have shown to be 20 for the number of neighbors, 0.05 for the contamination and *fast* for the method, in order to reduce the algorithm complexity. Considering this tuning and the evidence that points with a lower *ABOD score* are considered to be anomalous, 358 outliers have been detected and the 1% is shown in Table 3.

| *Index* | *ABOD score* |
|---|---|
| 3934 | -8.28 |
| 2451 | -1.98 |
| 3939 | -2.05 |
| 3935 | -2.19 |

*Table 3 Top* 1% outliers found using *ABOD*

The last anomalies detection method tested has been *Isolation Forest*, a model-based approach which has as strength a very low computational cost compared to the previous tested approaches. Considering as inliers the points scored close to 0 and outliers the ones scored close to -1, this algorithm has detected 368 points as outliers, which 1% is shown in Table 4.

| *Index* | *IsoF score* |
|---|---|
| 3934 | -0.19 |
| 3935 | -0.19 |
| 3939 | -0.17 |
| 3936 | -0.17 |

*Table 4* Top 1% outliers found using *Isolation Forest*

The outliers detected have shown to be an exiguous number and to not deviate very much from the rest of the data; moreover, none of the tested methods have assigned a really high score to the outliers detected. Since from these observations it has been possible to do not consider outliers as a problem for the quality of the data or as erroneous data coming from wrong measurements, it has been chosen to not treat the anomalies.

## 2. Dimensionality Reduction

The next stage in the analysis has been to explore different dimensionality reduction techniques, part of different families of methods: *Recursive feature elimination* and *Univariate feature selection* as Feature Selection approaches, and *PCA* as a Feature Projection method. Table 5 summarizes the results found for what the number of dimensions obtained is concerned.

| *Technique* | *N. of dimension obtained* |
|---|---|
| Recursive feature elimination | 35 |
| Univariate feature selection | 10 |
| PCA | 3 |

*Table 5* Number of dimensions obtained through dimensionality reduction techniques

After the application of different methods, it has been tested if the outlier detection, already discussed in Section 1.1, will provide better results on the reduced dataset. Except for *DB Scan* method which doesn't detect any outlier, *LOF*, *ABOD* and *Isolation Forest* do not return any improvement on the reduced sets neither for what the scoring is concerned neither taking into consideration the number of outliers.

## 2.1 Classification on dimensionality reduced sets

The first analysis provided on the dimensionality reduced datasets has been the classification task adopting the *Decision Tree* algorithm as classifier. Results obtained on the validation set have been evaluated through Precision, Recall and f1 metrics and are reported in Table 6. Looking at the results, it is clear that *Laying* is the easier variable to predict.

| | *Recursive feature elimination* | | | *Univariate feature selection* | | | *PCA* | | | *Support* |
|---|---|---|---|---|---|---|---|---|---|---|
| | *Precision* | *Recall* | *f1* | *Precision* | *Recall* | *f1* | *Precision* | *Recall* | *f1* | |
| LAYING | 1 | 1 | 1 | 1 | 1 | 1 | 0.97 | 0.98 | 0.98 | 422 |
| SITTING | 0.93 | 0.96 | 0.95 | 0.71 | 0.73 | 0.72 | 0.63 | 0.65 | 0.64 | 386 |
| STANDING | 0.96 | 0.93 | 0.95 | 0.74 | 0.72 | 0.73 | 0.69 | 0.66 | 0.67 | 412 |
| WALKING | 0.93 | 0.93 | 0.93 | 0.77 | 0.77 | 0.77 | 0.74 | 0.76 | 0.75 | 368 |
| WALKING DOWNSTAIRS | 0.91 | 0.94 | 0.92 | 0.76 | 0.75 | 0.75 | 0.60 | 0.61 | 0.60 | 296 |
| WALKING UPSTAIRS | 0.93 | 0.91 | 0.92 | 0.78 | 0.79 | 0.78 | 0.76 | 0.72 | 0.74 | 322 |

*Table 6* Evaluation od Decision Tree classifier on validation sets reduced in different

*Decision Tree* classifier reaches the values of accuracy on the different sets shown in Table 7.

| | *Accuracy* |
|---|---|
| *Set reduced through Recursive Feature Elimination* | 0.95 |
| *Set reduced through PCA* | 0.74 |
| *Set balanced through Univariate Feature Selection* | 0.80 |

*Table 7* Accuracy on reduced datasets

Comparing the results to the ones obtained using the *Decision Tree* on the original dataset, shown in Section 4.1.1, it is possible to notice that the only case in which Accuracy value improves is for the classification on reduced set through *Recursive feature elimination*. Accuracy reached on the sets reduced through *PCA* and *Univariate Feature Selection* are considerable sufficient but do not raise performances on the original set.

## 2.2 Two-dimensional visualization

The second step of the testing on dimensionally reduced dataset has been define the separability of the activity labels from a visual point of view. To evaluate visually if the classes can be separable, it has been used *PCA* as reduction technique. As it is possible to observe from the scatter plot on the left in Figure 3, *PCA* is not able to clearly separate the data. For this reason, it has been implemented another method, *t-SNE*, which provides a clearer separation of the data from the moment that this algorithm is more suitable for this task.

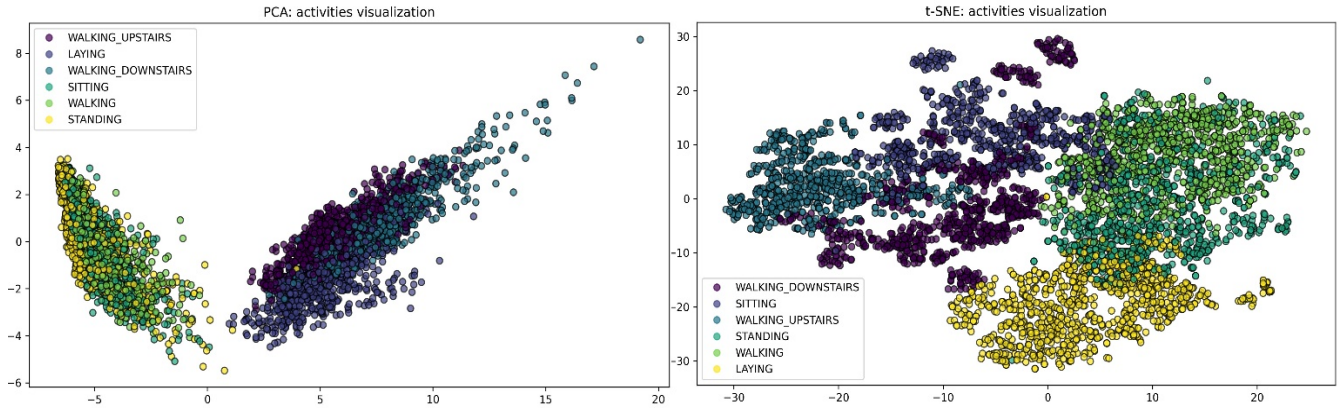



*Figure 2* Separation of Activity using PCA and t-SNE techniques

# 3. Imbalanced learning

This section is centered on testing imbalanced learning techniques. From the moment that, as explained in Section 1, the dataset presents a balanced distribution of the six different activities composing the *Activity* variable, in order to test the balancing techniques, it has been necessary to manually compose an unbalanced version of the data.

To provide a more original evaluation, it has been decided to take into consideration only two of the six activities labeled in the set: *Walking* and *Walking Upstairs*, composing a new subset of the original dataset. As shown on the left of Figure 3, also the new subset presented a balanced shape of the variables: considering the total of the instances, 1226 comes from *Walking* (51% of the subset) and 1073 from *Walking_Upstairs* (49% of the set). To achieve the goal of testing imbalance learning techniques, the subset has been manually unbalanced: the 1226 instances of *Walking* have been turned into the 96% of the set, reducing the *Walking_Upstairs* instances from 1073 to 49.
The resulting subset is composed by 1275 instances and the distribution of the two variables is shown in the right side of Figure 3.

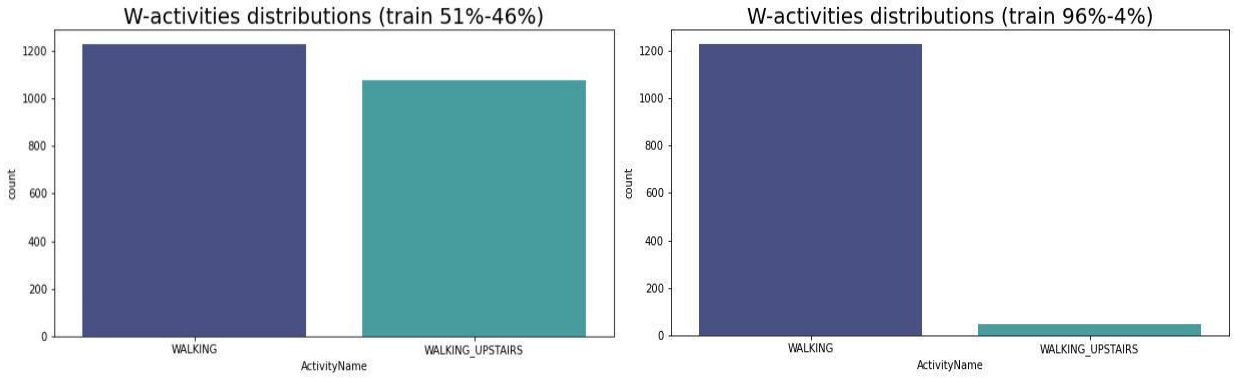



*Figure 3* On the left, balance training set; on the right, unbalanced version

At this point, three different algorithms of rebalancing were applied on the training set and the reshape of the train for the three methods is shown in Table 8.

| | *n. instances of Walking* | *n. instances of Walking_Upstairs* |
|---|---|---|
| *Condensed Nearest Neighbor* | 34 | 10 |
| *Random Undersampling* | 34 | 34 |
| *Random Oversampling* | 858 | 858 |

*Table 8* Reshape of the train set for three different rebalancing methods

For what classification is concerned, *Decision Tree* algorithm has been used to classify the two activities, training the algorithm on the balanced datasets and testing it on the unbalanced validation sets. Results provided on the validation sets have been evaluated through Precision, Recall and f1 metrics and are shown in Table 9.

| | *Random Oversampling* | | | *Random Undersampling* | | | *CNN* | | | *Support* |
|---|---|---|---|---|---|---|---|---|---|---|
| | *Precision* | *Recall* | *f1* | *Precision* | *Recall* | *f1* | *Precision* | *Recall* | *f1* | |
| WALKING | 1 | 0.99 | 0.99 | 1 | 0.95 | 0.97 | 1 | 0.95 | 0.98 | 368 |
| WALKING UPSTAIRS | 0.79 | 1 | 0.88 | 0.44 | 1 | 0.61 | 0.47 | 1 | 0.64 | 15 |

*Table 9 Decision Tree* performances evaluation on balanced datasets

A further analysis on the classification performances has been provided through the construction of ROC curves for all the methods:

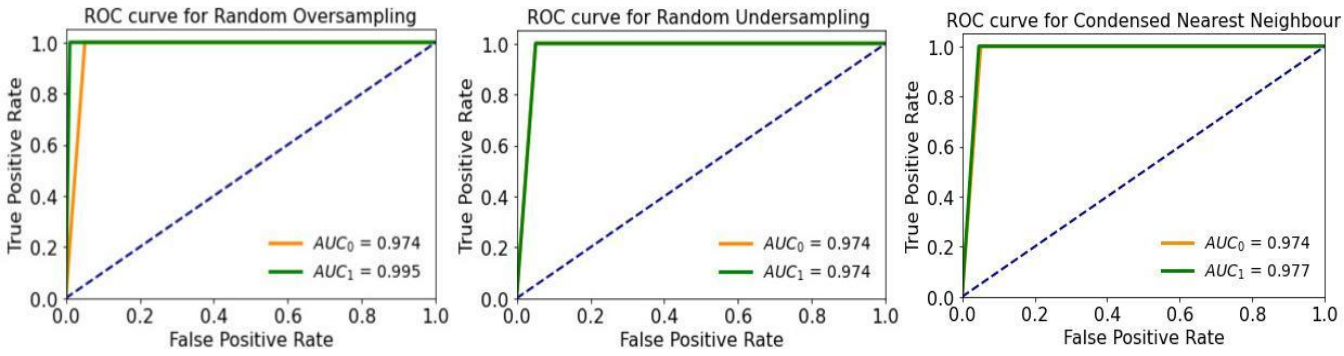



*Figure 4* ROC curves for the different sets reduced with different techniques. Class 0 is Walking

Looking at the ROC curves in Figure 4, it is possible to notice that *Random Oversampling* is the most performing algorithm. For what accuracy is concerned, *Decision Tree* classifier reaches values of accuracy shown in Table 10 on the different sets.

| | *Accuracy* |
|---|---|
| *Set balanced through Random Oversampling* | 0.99 |
| *Set balanced through Random Undersampling* | 0.95 |
| *Set balanced through CNN* | 0.96 |

*Table 10* Accuracy of DT on different reduced sets

Overall, taking also in consideration that *Random Undersampling* and *Condensed Nearest Neighbor* provide a very reduced version of the subset, *Random Oversampling* is definable as the best balancing method on this set of data, taking into account values reported in Tables 9 and 10.

# 4. Classification

This section is centered on the classification task, with the aim to build a model able to predict the activity labels. To achieve this goal, they have been analyzed the performances of both simple and advanced classifiers.

All the models have been trained on the training set while the validation set has been used to tune their optimal parameters. At the end of the training phase, all the classifiers have been tested on the test set.

Results provided in the tables and in the following section refer to the trials on the final test set and have been evaluated using Precision, Recall, f1 metrics and Accuracy. Moreover, for each classifier have been built the ROC curves (Receiver Operating Characteristic) for each of the six classes using the One versus rest technique and computed the AUC (area under the curve) score as an average of the AUC scores of the six curves obtained.

All the algorithms have been implemented using the *Scikit Learn Library* (except the *XGBoost,* exposed in Section 4.2.1.1, because it is not available from this library).

## 4.1 Basic Classifiers

As basic classifiers, they have been tested *Decision Tree*, *K-NN*, the *Naive Bayes classifier* in its *Gaussian* version and *Logistic Regression*.

### 4.1.1 Decision Tree

The first tested classifier has been the *Decision Tree*. To find the best parameters for the tuning of the algorithm, it has been used the Grid search technique, which returned as best setting to tune the max depth of the tree at 8, the min sample leaf at 5, the min sample split at 10 and to use as splitting criterium the *Gini* one.

After the tuning, the *Decision Tree* provided the results on the test set shown in Table 11 for what Precision, Recall and f1 score are concerned.

| | *Decision Tree* | | | |
|---|---|---|---|---|
| | *Precision* | *Recall* | *f1* | *Support* |
| LAYING | 1 | 1 | 1 | 537 |
| SITTING | 0.88 | 0.78 | 0.83 | 491 |
| STANDING | 0.82 | 0.90 | 0.86 | 532 |
| WALKING | 0.81 | 0.94 | 0.87 | 496 |
| WALKING DOWNSTAIRS | 0.85 | 0.75 | 0.79 | 420 |
| WALKING UPSTAIRS | 0.74 | 0.71 | 0.72 | 471 |

*Table 11* Evaluation of Decision Tree classifier

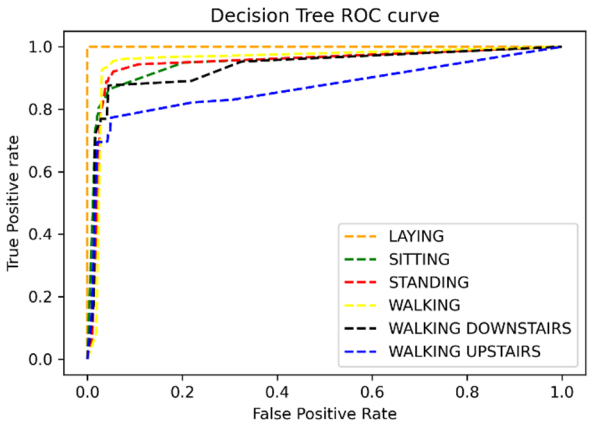



*Figure 5 ROC* curves for Decision Tree classifier

Considering Accuracy score, the model reaches 0.85 on the test set.

The following step has been to build the ROC curve for the classifier (shown in Figure 5) and compute the respective AUC (Area under the curve), which shows to be equal to 0.94.

The worst class to predict according to the ROC is *Walking Upstairs*.

### 4.1.2 K-nearest neighbors (K-NN)

The second algorithm to be tested has been the *K-nearest neighbors*, one of the simplest classification algorithms which computes the similarity between the new case and available cases and put the new case into the category that is most like the available categories. In order to maximize the value of accuracy it has been implemented a Grid search, which suggested to set the number of neighbors equal to 10 and use the '*Distance*' weights. Using these parameters, results on the test set are shown in Table 12. For what Accuracy is concerned, the model reaches satisfying results, providing 0.90 on the test set. Figure 6 shows the ROC curve for the classifier, whose AUC score is 0.98.

| | *K-NN* | | | |
|---|---|---|---|---|
| | *Precision* | *Recall* | *f1 score* | *Support* |
| LAYING | 1 | 0.99 | 0.99 | 537 |
| SITTING | 0.91 | 0.79 | 0.84 | 491 |
| STANDING | 0.83 | 0.93 | 0.88 | 532 |
| WALKING | 0.85 | 0.97 | 0.91 | 496 |
| WALKING DOWNSTAIRS | 0.93 | 0.80 | 0.86 | 420 |
| WALKING UPSTAIRS | 0.90 | 0.87 | 0.89 | 471 |

*Table 12* Evaluation of K-NN classifier

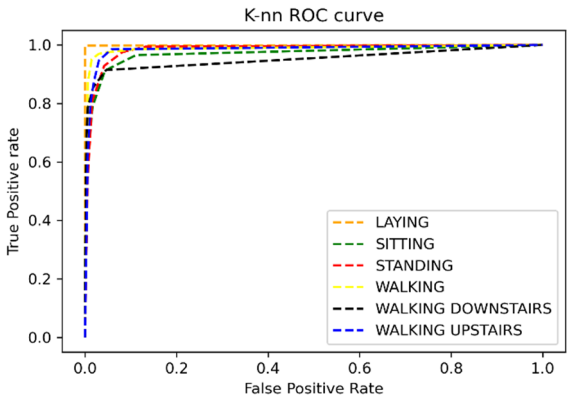



*Figure 6 ROC* curves for K-NN classifier

### 4.1.3 Naive Bayes Classifier

*Naive Bayes* is a supervised learning algorithm which is based on Bayes theorem, used for classification task. There are two assumptions behind the functioning of this probabilistic classifier: the independence between the features and the fact that each feature has the same importance (so none is irrelevant).

The Naive Bayes has been the third classifier to be tested, using its Gaussian Approach, which assumes that continuous features follow a normal distribution.

The model reaches a lower Accuracy score compared to the previous basic classifiers, since it is only 0.64. Other evaluations on the test set are shown in Table 13. As for the previous algorithms, to evaluate the performances of the classifier, it has been built the ROC curve, shown in Figure 7, and calculated the AUC score, which returned 0.94. The worst curve is the one of *Sitting*, which also shows a low value of Precision.

| | *GaussianNB* | | | |
|---|---|---|---|---|
| | *Precision* | *Recall* | *f1* | *Support* |
| LAYING | 0.85 | 0.18 | 0.29 | 537 |
| SITTING | 0.37 | 0.87 | 0.52 | 491 |
| STANDING | 0.81 | 0.40 | 0.53 | 532 |
| WALKING | 0.84 | 0.85 | 0.84 | 496 |
| WALKING DOWNSTAIRS | 0.84 | 0.63 | 0.72 | 420 |
| WALKING UPSTAIRS | 0.77 | 0.97 | 0.86 | 471 |

*Table 13* Evaluation of *Naïve Bayes classifier*

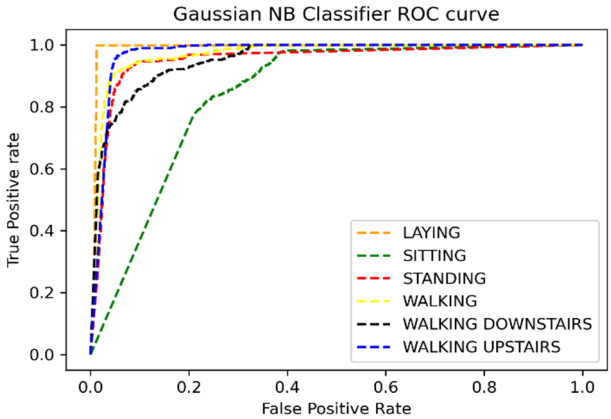



*Figure 7* ROC curves for *Naïve Bayes classifier*

### 4.1.4 Logistic regression

*Logistic Regression* is a supervised learning classification algorithm used to predict the probability of a target variable. As *Support Vector Machine*, which will be exposed in Section 4.2.3, this algorithm is designed for binary classification and do not natively support multi-class classification tasks. One approach to adapt these kinds of algorithms to multi-classification problems is the *One-vs-Rest* (shortened in OvR) strategy, a heuristic method which fits one classifier per class, then fits the class against all the other classes and, at prediction time, decide if a given sample belongs to a class or not applying each classifier (trained for each class). A possible downside of this approach is that it requires one model to be created for each class, which could be a problem for large datasets.

So, to achieve a multi-classification, the algorithm has been tuned specifying as the *multi-class* parameter to '*ovr*' value and, after different trials to detect the optimal values of the different parameters, leaving the default options for the other parameters.

After the tuning, the model has been evaluated on the test set, returning the values reported in Table 14 for what Precision, Recall and f1 are concerned and a value of Accuracy equal to 0.96.

The ROC curves computed for all the classes are shown in Figure 8; the AUC is equal to 0.99.

| | *Logistic Regression* | | | |
|---|---|---|---|---|
| | *Precision* | *Recall* | *f1* | *Support* |
| LAYING | 1 | 1 | 1 | 537 |
| SITTING | 0.97 | 0.89 | 0.93 | 491 |
| STANDING | 0.91 | 0.97 | 0.94 | 532 |
| WALKING | 0.93 | 0.99 | 0.96 | 496 |
| WALKING DOWNSTAIRS | 0.99 | 0.97 | 0.98 | 420 |
| WALKING UPSTAIRS | 0.98 | 0.94 | 0.96 | 471 |

*Table 14* Evaluation of *Logistic Regression* classifier

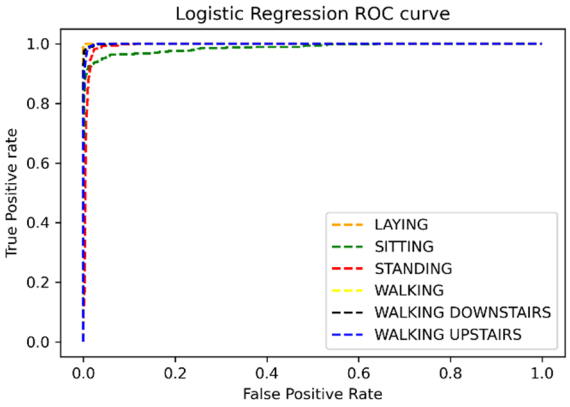



*Figure 8 ROC* curves for *Logistic Regression*

### 4.1.5 Comparison

The basic classifiers reach different performances on the data. For *Logistic Regression*, *K-NN* and *Decision Tree*, *Laying* results to be the easier activity to predict, an expected observation since such activity presents the more distinguishable values respect to the other variables. Instead, for the *Naive Bayes*, *Laying* obtains really low values of Recall and f1, but a perfect ROC curve, as for the other three classifiers.

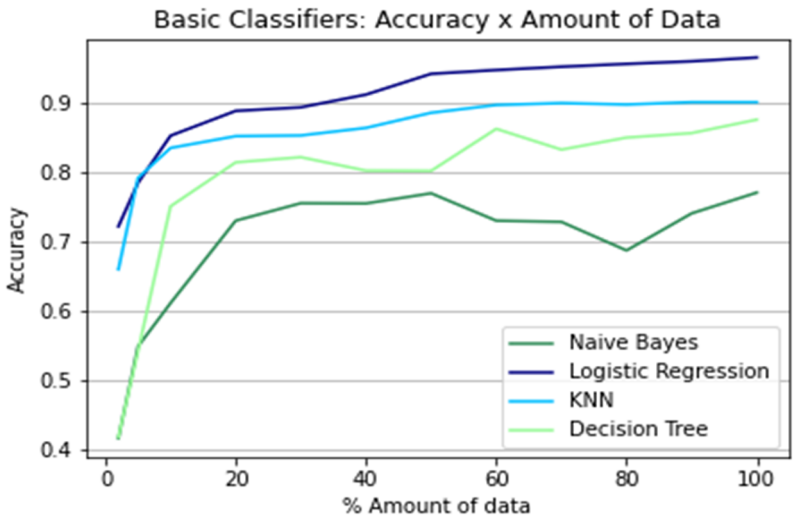



*Figure 9* Variance of accuracy based on training record provided for basic classifiers

To understand how much the amount of data provided to train the models actually affects the performances for different classifiers, it has been plotted the variance of accuracy based on the percentage of training records provided in the train phase for each of them. Figure 9 shows that a decent level of accuracy is reached almost by all the classifiers when at least a 20% of training set is provided. The worst performing one on the long term is *Naive Bayes*, whose performance can't be compared with the other classifiers, which return optimal and constantly improving-over-data performances.

Overall, looking also at the tables and at the accuracy values exposed in the previous sections, *Logistic Regression* classifier reveals to be the more performing classification algorithm on the data, with an accuracy of 0.96.

## 4.2 Advanced Classifiers

As advanced classifiers, they have been tested *Gradient Boosting, Neural networks, Support Vector Machine* and *Ensemble classifiers*.

### 4.2.1 Gradient Boosting

The first advanced classifier to be tested has been the *Gradient Boosting* algorithm. It is a machine learning algorithm that combines many weak learning models together to create a strong predictive model. The objective of *Gradient Boosting* classifier is to minimize the loss, or the difference between the actual class value of the training example and the predicted class value.

| | *Gradient Boosting Machine* | | | |
|---|---|---|---|---|
| | *Precision* | *Recall* | *f1* | *Support* |
| LAYING | 1 | 0.99 | 1 | 537 |
| SITTING | 0.97 | 0.88 | 0.92 | 491 |
| STANDING | 0.90 | 0.98 | 0.94 | 532 |
| WALKING | 0.93 | 0.98 | 0.95 | 496 |
| WALKING DOWNSTAIRS | 0.98 | 0.86 | 0.91 | 420 |
| WALKING UPSTAIRS | 0.90 | 0.93 | 091 | 471 |

*Table 15* Evaluation of *Gradient Boosting Machine*

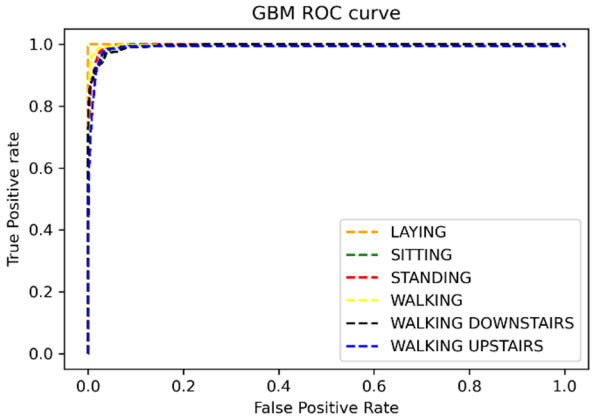



*Figure 10 ROC* curves for *Gradient Boosting Machine*

The first step has been to define the best *Learning Rate* for the algorithm, computing the accuracy score on the train and on the validation sets for a range of learning rate from 0.05 to 1. It appears that the value of Learning Rate which maximizes the value of accuracy on both the train and the validation sets is 1. After implementing several trials, it has been decided to leave the other parameters as default settings.

Successively, performances on the test set have been evaluated also through precision, recall and f1 metrics and are reported in Table 15. The model reaches an accuracy level of 0.94 on the test set. As for the other classifiers, it has been built the ROC curve (shown in Figure 10) and computed the AUC score, equal to 0.99.

#### 4.2.1.1 XGBoost

To provide a more interesting evaluation, it has been tested also the *XGBoost* (*eXtreme Gradient Boosting*), an implementation of *Gradient Boosting* designed for improve speed and performances. For what the parameter tuning is concerned, the objective defined has been the *multi:logistics*, as tree method the *'hist'* one (from the moment that it is recommended for high dimensional datasets) and the others parameters have been left as default. Performances on the test set have been evaluated through the same metrics used for *Gradient Boosting* and are contained in Table 16. *XGBoost* achieves very good performance, with a level of accuracy of 0.93. Like for the other classifiers, it has been built the ROC curve (shown in Figure 11) and computed the AUC score, equal to 0.99.

| | *XGBoost* | | | |
|---|---|---|---|---|
| | *Precision* | *Recall* | *f1* | *Support* |
| LAYING | 1 | 1 | 1 | 537 |
| SITTING | 0.94 | 0.84 | 0.89 | 491 |
| STANDING | 0.87 | 0.95 | 0.91 | 532 |
| WALKING | 0.90 | 0.96 | 0.93 | 496 |
| WALKING DOWNSTAIRS | 0.97 | 0.90 | 0.93 | 420 |
| WALKING UPSTAIRS | 0.90 | 0.90 | 0.90 | 471 |

*Table 16* Evaluation of *XGBoost*

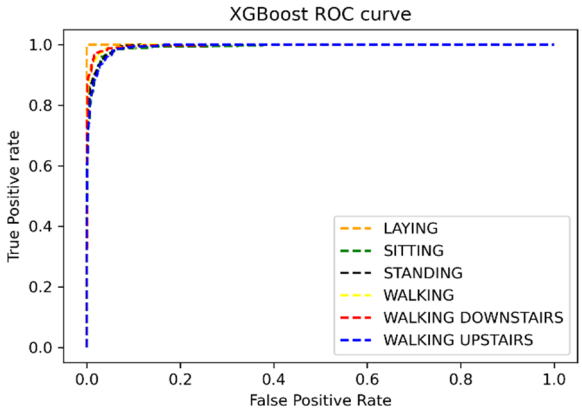



*Figure 11* ROC curves for *XGBoost*

### 4.2.2 Neural Networks

This section is focused on the construction of an *Artificial Neural Network Classifier*.

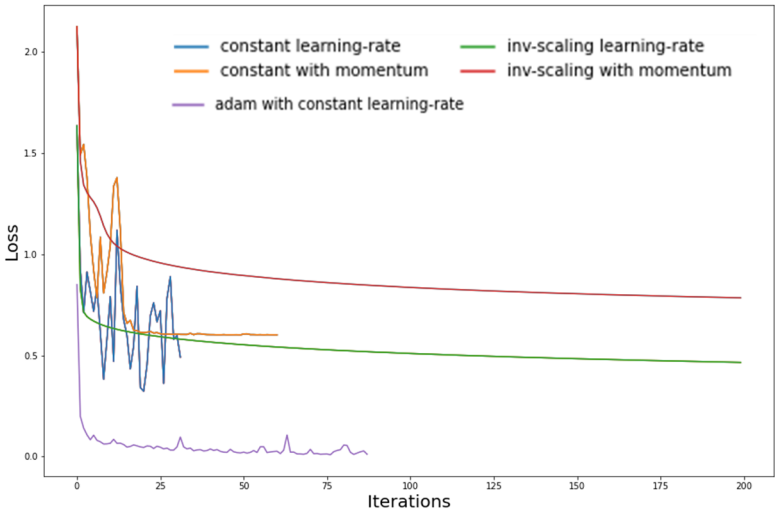



*Figure 12 Loss* curves for different parameters of *MLPC*

The training set was used in all its integrity, then split by the early stopping parameter for multilayer perceptron classifier, that uses a 10% of the training set to create a validation set; the model built was evaluated on the official test set. In order to avoid overfitting problems the early stopping parameter was included inside the model constructor, which does a monitoring on the model performance using the validation set and stops the training when validation score is not improving in a significant way through consecutive iterations.

For different configurations of *Multiple Linear Perceptron Classifier,* different loss curves were built in order to make a comparison and understand which type of solver and learning curve settings better suited our set of data.

As shown in Figure 12, the best performing set of parameters that provides the lowest value of loss is the setting of an adaptive learning rate with initialization value equal to 0.01 and the appliance of the *'Adam'* solver, an expected result since this stochastic gradient-based optimizer is actually suited for classification task on big sets of data. Those two settings were employed in the construction of the *NN* classifier, together with a single hidden layer composed of 100 units, a number of maximum iterations fixed to 200 and the rectified linear unit activation function: those other parameters were chosen according to a Grid Search application on various combinations of parameters. The other parameters have been left as the default settings. The *NN* classifier provided an accuracy of 0.95 on the validation and of the 0.94 on the test set. Table 17 reports the Precision, Recall and f1 values on the test set.

| | *Neural Network* | | | |
|---|---|---|---|---|
| | *Precision* | *Recall* | *f1* | *Support* |
| LAYING | 1.00 | 0.97 | 0.99 | 537 |
| SITTING | 0.95 | 0.88 | 0.91 | 491 |
| STANDING | 0.88 | 0.96 | 0.92 | 532 |
| WALKING | 0.93 | 0.98 | 0.95 | 496 |
| WALKING DOWNSTAIRS | 0.97 | 0.92 | 0.95 | 420 |
| WALKING UPSTAIRS | 0.93 | 0.93 | 0.93 | 471 |

*Table 17* Evaluation of *NN*

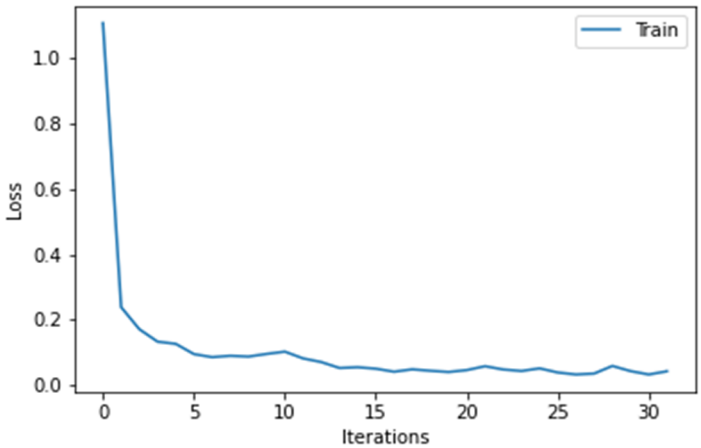



*Figure13* Loss curve on training set

As shown in Figure 13, the drastic descent of the loss values through iterations on the training set shows that our model is more than suitable in the prediction of the different movement classes, resulting in a computed current loss of 0.043.

The ROC curves shown in Figure 14 are satisfying for all the six classes. Moreover, it is provided also an optimal value of AUC, 0.99.

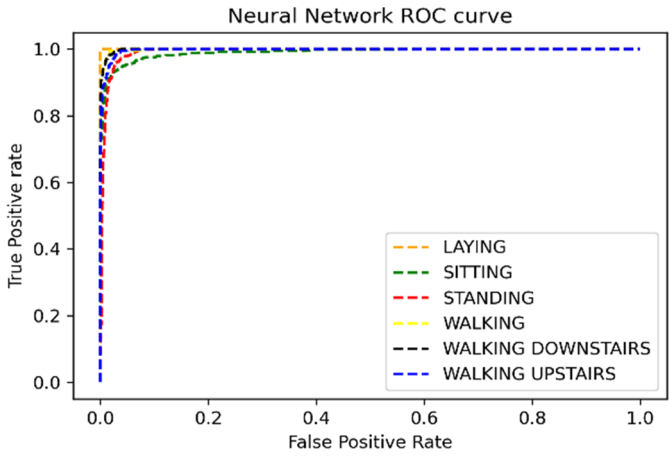



*Figure 14* ROC curves for *NN* classifier

### 4.2.3 Support Vector Machine

*Support Vector Machine* (shortened in SVM) is a supervised machine learning algorithm that tries to find an optimal boundary (known as hyperplane) between different classes using a subset of the training examples, called support vectors.

Natively, SVM doesn't support multiclass classification so, as already explained for the Logistic Regression in section 4.1.1, it has been used the *One-vs-Rest* strategy, setting '*ovr*' as value of the parameter *decision_function_shape* of the scikit learn algorithm.

SVM algorithm use a set of mathematical functions that are defined as the kernel, which take data as input and transform it into the required form. For this analysis the SVM algorithm has been tested with three different types of kernel functions: *linear, polynomial, and radial basis function (RBF).*

For what the parameter tuning is concerned, after different trials, in order to maximize Accuracy score, it has been set the value of C equal to 1, gamma equal to '*auto*' and 3 as degree of the *polynomial* kernel function. The other parameters have been left as the default settings.

After the tuning, performances on the test set for the algorithm set with the three different kernels have been evaluated using Precision, Recall and f1 and are contained in Table 18.

| | *Linear kernel* | | | *Polynomial kernel* | | | *RBF kernel* | | | *Support* |
|---|---|---|---|---|---|---|---|---|---|---|
| | *Precision* | *Recall* | *f1* | *Precision* | *Recall* | *f1* | *Precision* | *Recall* | *f1* | |
| LAYING | 1 | 1 | 1 | 1 | 1 | 1 | 0.99 | 1 | 1 | 537 |
| SITTING | 0.97 | 0.89 | 0.93 | 0.93 | 0.88 | 0.91 | 0.93 | 0.89 | 0.91 | 491 |
| STANDING | 0.91 | 0.98 | 0.94 | 0.90 | 0.94 | 0.92 | 0.91 | 0.94 | 0.93 | 532 |
| WALKING | 0.95 | 0.98 | 0.97 | 0.93 | 0.99 | 0.96 | 0.96 | 0.98 | 0.97 | 496 |
| WALKING DOWNSTAIRS | 0.98 | 0.94 | 0.96 | 0.99 | 0.90 | 0.94 | 0.98 | 0.90 | 0.94 | 420 |
| WALKING UPSTAIRS | 0.94 | 0.95 | 0.95 | 0.94 | 0.96 | 0.95 | 0.92 | 0.97 | 0.94 | 471 |

*Table 18* Evaluation of *SVM*

After this evaluation, they have been computed the ROC curves for all the classes, represented in Figure 15.

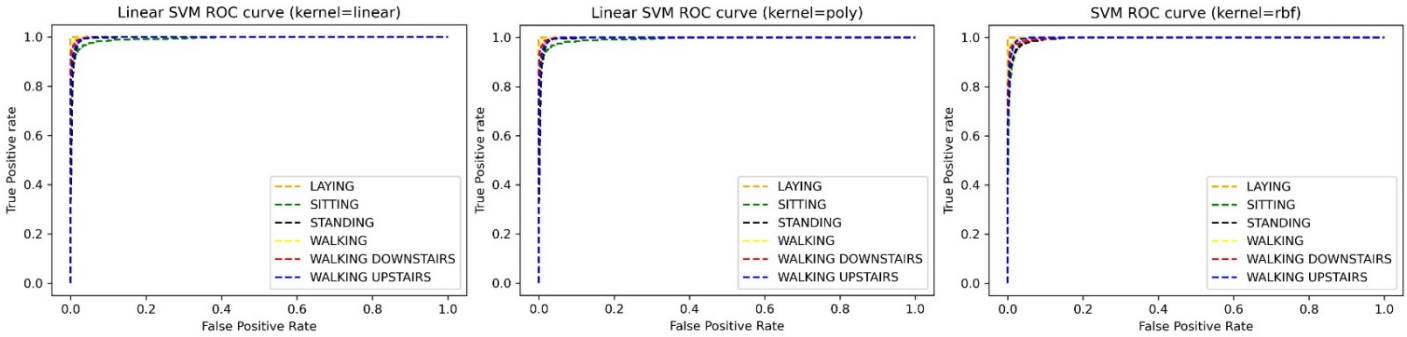



*Figure 15* ROC curves for *SVM* classifier

For what Accuracy and AUC score are concerned, the different scores are contained in Table 19.

| | *Accuracy* | *AUC* |
|---|---|---|
| Linear kernel | 0.949 | 0.99 |
| Polynomial kernel | 0.947 | 0.99 |
| RBF kernel | 0.957 | 0.99 |

*Table 19* Accuracy and AUC scores

The Support Vector Machine classifier, returned optimal values on the classification task, considering both the Accuracy values and the ROC curves provided for the different classes.

### 4.2.4 Ensemble Classifier

This section covers the following ensemble techniques: *Random Forest, Bagging* and the adaptive version of boosting (*AdaBoost*). The basic idea of ensemble classifiers is to aggregate the predictions of several independent base classifiers in order to obtain an optimal resulting model and an improvement in the level of accuracy. The final output is based on majority voting after combining the results of all models. The first method is grounded on a manipulation of input features, the other two affect the original data distribution to create a sample of values by manipulating the records.

The first model to be tested has been Random Forest, which combines the predictions made by multiple decision trees built on different samples.
A Grid Search was implemented in order to obtain the best settings for specific parameters, returning as optimal tuning to set '*Gini*' as splitting criterium with a max depth of 10, a max features value set on auto, min samples leaf of 5, 25 estimators and min samples split of 5.

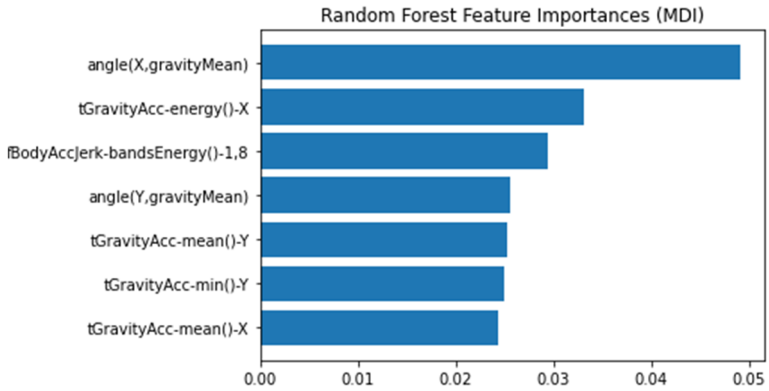



*Figure 16* Feature importance for *Random Forest*

As is shown in the plot in Figure 16, none of the features covers a great importance in the classification task, probably due to the high number of variables considered. In any case, the most important variable turns out to be *angle(X.gravityMean).*

*Bagging*, or *Bootstrap AGGregatING*, has been the second model to be inspected, using the *Naive Bayes,* shown in Section 4.1.3, as base classifier, as it presented the worst performance among the basic classifiers. Nevertheless, it hasn't been recorded any improvement compared to the performances obtained from a single Naive Bayes classifier. To provide a more complete evaluation, this algorithm has also been analyzed with *Decision Tree* and *Random Forest*. For what Decision Tree is concerned, it has been tested a range of trees from 1 to 100, achieving the best results with 25 trees and an oscillation of the values of accuracy from 88 to 90% for the different trials. However, the best performance was obtained using *Random Forest* as base classifier, which achieved an accuracy value of 0.92.

The last classifier rated has been *AdaBoost*, using *Random Forest* as base estimator. For what the parameter tuning is concerned, in order to improve Accuracy and define the best trade-off between the number of estimators and the learning rate, they have been implemented different trials. As can be seen from Figure 17, the optimal value of accuracy, 0.93, has been obtained tuning 10 as *n_estimators* and a *learning rate* of 0.75.


*Figure 17* Parameters tuning for *AdaBoost*

Performances of the Ensemble models have been evaluated using Precision, Recall and f1 score and are reported in Table 20. The resulting ROC curves are comparable in Figure 18.

| | *Random Forest* | | | *Bagging (Random Forest)* | | | *AdaBoost (Random Forest)* | | | *Support* |
|---|---|---|---|---|---|---|---|---|---|---|
| | *Precision* | *Recall* | *f1* | *Precision* | *Recall* | *f1* | *Precision* | *Recall* | *f1* | |
| LAYING | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 537 |
| SITTING | 0.94 | 0.85 | 0.89 | 0.93 | 0.87 | 0.90 | 0.94 | 0.88 | 0.91 | 491 |
| STANDING | 0.87 | 0.95 | 0.91 | 0.89 | 0.94 | 0.91 | 0.90 | 0.94 | 0.92 | 532 |
| WALKING | 0.89 | 0.98 | 0.93 | 0.88 | 0.97 | 0.92 | 0.90 | 0.97 | 0.93 | 496 |
| WALKING DOWNSTAIRS | 0.97 | 0.84 | 0.90 | 0.95 | 0.83 | 0.89 | 0.96 | 0.85 | 0.90 | 420 |
| WALKING UPSTAIRS | 0.99 | 0.91 | 0.90 | 0.91 | 0.90 | 0.91 | 0.90 | 0.92 | 0.91 | 471 |

*Table 20* Evaluation of Ensemble classifiers

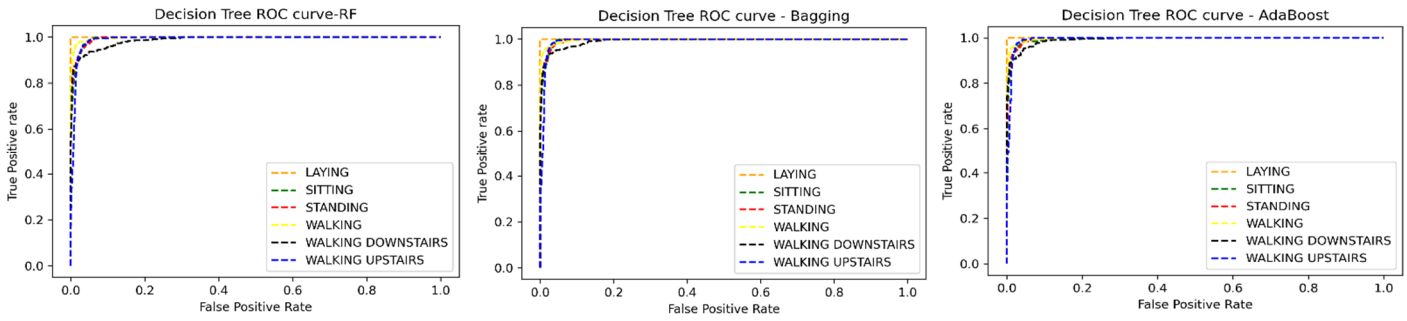



*Figure 18* ROC curves for Ensemble classifiers

Table 21 shows the respective AUC values and compares the accuracy of the optimal ensemble models.

| | *Accuracy* | *AUC* |
|---|---|---|
| Random Forest | 0.92 | 0.994 |
| Bagging + Random Forest | 0.92 | 0.995 |
| AdaBoost + Random Forest | 0.93 | 0.995 |

*Table 21* Accuracy and AUC scores for Ensemble classifiers

### 4.2.5 Comparison

The advanced classifiers reach different performances on the data. For all the classifiers the easier class to predict has been *Laying*. Comparing the final values of Accuracy, the most performing model results to be *Support Vector Machine* with *RBF kernel*, with an Accuracy of 0.96.

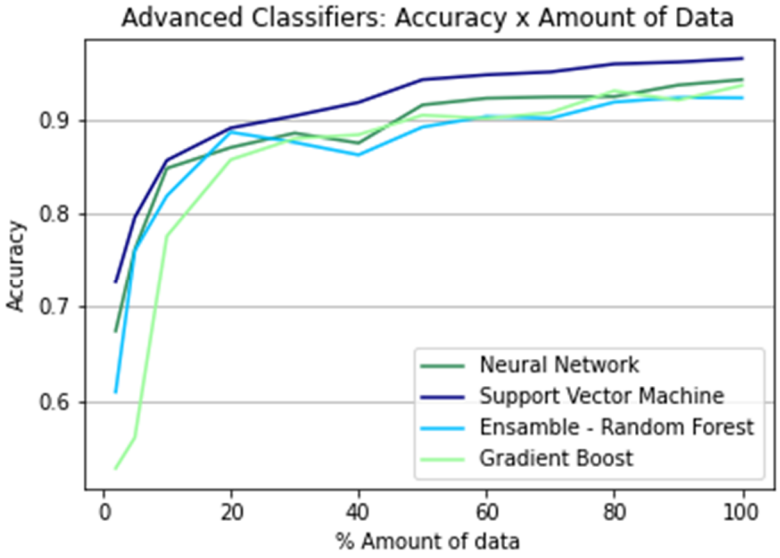



*Figure 19* Variance of accuracy based on training record provided for advanced classifiers

As for Basic Classifiers (Section 4.1.5), has been realized an incremental procedure (in terms of percentage) on the train set, in order to compare different accuracy results for different amounts of data provided.
Observations are similar to the ones done for basic classifiers: an appreciable level of accuracy is reached almost by all the classifiers when at least a 20% of training set is provided. For this amount of data, all the classifiers show a good level of accuracy, from 0.85 to 0.90.
Overall, the best performing one on the long term is *Support Vector Machine*, whose performances constantly grow up, tending to 1.

# 5. Regression

In this section is tackled a simple *Linear Univariate Regression Problem* based on two continuous attributes, using various regression techniques such as *LinearRegression*, its regularized version *Ridge*, *Huber* and *Gradient Boosting Machine*. The choice of variables has been based on the correlation index computed between all the possible pairs of attributes, which, in this specific dataset, appear to be all very correlated.
The model has been set using the variable *fBodyBodyGyroMag-mean()* as dependent and *tBodyGyroMag-std()* as independent variable. For what the parameters tuning is concerned, they have been left as default setting

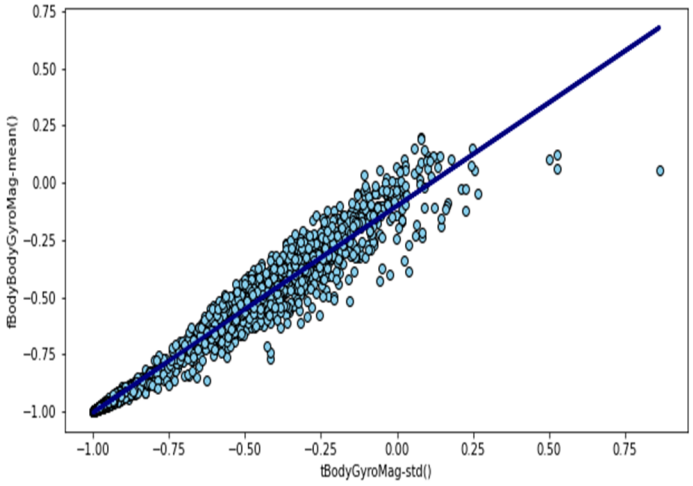



*Figure 20* Simple Linear Regression

since returned optimal results. To evaluate the models they have been used the following evaluation metrics:
R2, Mean Squared Error and Mean Absolute Error, whose values are shown in Table 22.

| | *R2* | *MSE* | *MAE* |
|---|---|---|---|
| Linear | 0.965 | 0.003 | 0.033 |
| Ridge | 0.965 | 0.003 | 0.033 |
| Huber | 0.965 | 0.003 | 0.032 |
| Gradient boosting | 0.964 | 0.003 | 0.032 |

*Table 22* Evaluation metrics for simple regression problem

The following step has been to implement a *Multiple Linear Regression*, adding the variable *fBodyBodyGyroMag-min()* among the independent ones. In order to measure the performances of this model, in Table 23 are presented the same methods and metrics applied to evaluate the simple regression. Likewise to the previous simple models, it can be noticed that all tested methods return almost equal results; an observation which confirms the strategy to do not remove outliers exposed in Section 1.1, from the moment that, even if Huber is a method less sensitive to anomalies, presents the same values of the other techniques.

| | *R2* | *MSE* | *MAE* |
|---|---|---|---|
| Linear | 0.967 | 0.003 | 0.033 |
| Ridge | 0.967 | 0.003 | 0.033 |
| Huber | 0.969 | 0.003 | 0.031 |
| Gradient Boosting | 0.968 | 0.003 | 0.030 |

*Table 23* Evaluation metrics for multiple regression problem

As can be observed looking at the outcomes in Figure 20 and 21, both the simple and multiple models reached optimal performances, as expected because of the high correlation among the variables. Therefore, we can consider both of them suitable to solve the analyzed linear regression problem.

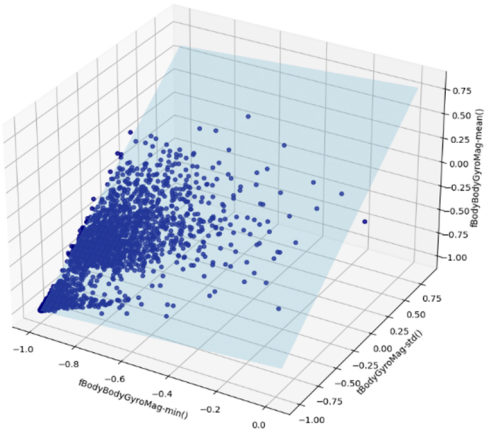



*Figure 21* 3D representation of the variables in multiple regression model

## 6. Time Series Analysis

This section is centered on the Time Series Analysis, focusing on the signal *total_acc_x*. This selection has been made after testing the *pyts K-nearest neighbors* classifier performances using the *Euclidean* distance on the supplied test set of all the nine signals provided by the authors of the dataset. The classification algorithm has been tested both on the non-normalized data and on data normalized through the *tslearn TimeSeriesScalerMinMax*, providing the evidence of a decreasing of performances on the normalized set. Results reported in Table 24 have been obtained on the non-normalized data and evaluated through Accuracy score, after the detection of the optimal parameters setting for all the signals using a Grid Search.

| *Signal* | *Accuracy* |
|---|---|
| *body_acc_x* | 0.59 |
| *body_acc_y* | 0.49 |
| *body_acc_z* | 0.46 |
| ***total_acc_x*** | **0.73** |
| *total_acc_y* | 0.62 |
| *total_acc_z* | 0.53 |
| *body_gyro_x* | 0.58 |
| *body_gyro_y* | 0.43 |
| *body_gyro_z* | 0.52 |

*Table 24* K-nn's accuracy scores for the test set of all the nine signals

The dataset used for the analysis is made by 7352 time series, each composed by fixed width sliding windows of 2.56 sec, for a total of 128 time stamps for each time series. The selected

signal has been used to provide clustering, motifs and discords discovery, shapelets analysis and classification, reported in the following sections.

For shapelets extraction and classification tasks, the dataset has been split in train and validation sets, respectively composed of 5146 and 2206 records.

For what normalization is concerned, as previously explained for *K-nn*, analysis have been executed both on normalized and non-normalized data, adopting the best set depending on the results. In order to improve performances, after different trials it has been determined to use non-normalized data for shapelets discovery and classification and normalized ones for clustering and motifs/discords detection.

## 6.1 Time Series Clustering

For clustering task it has been used the *TimeSeriesKmeans* algorithm from the *tslearn* python library, applied on the normalized data, since they shown to provide a better clustering of the time series.

The optimal *k* number of clusters has been detected through a visual approach using the Silhouette metric and the SSE score; considering a range of k from 1 to 10 the optimal number of clusters resulted to be 2. It has also inspected the possibility to divide the data in more clusters, but the algorithm did not provide optimal results for what the balancing and the division of the clusters were concerned. Moreover, it has also taken into consideration the fact that the six target activities could be divided in two macro activities, static (*Laying*, *Standing* and *Sitting*) and dynamic (*Walking, Walking_Downstairs* and *Walking_Upstairs*) and this dichotomy might be highlighted also by resulting clusters' composition.

The algorithm has been tested using *Euclidean* distance metric with different clustering approaches: *shape-based*, *feature-based* and *approximation-based*. Feature-based approach has been implemented extracting statistic descriptive features as mean, standard deviation, variance, median and others. Approximation-based approach has been set defining the number of segments as 13 and using the *Piecewise Aggregate Approximation* (*PAA*). The different results are reported in Figure 22.

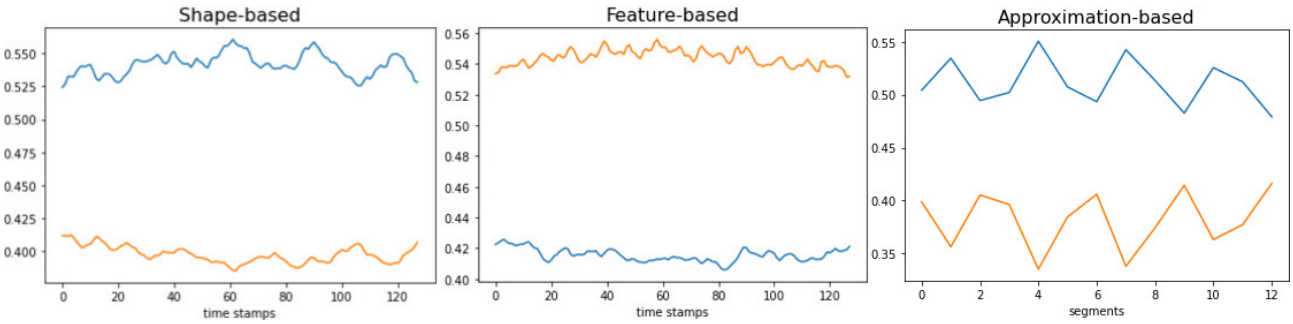



*Figure 22* Clustering approaches comparison

As it can be noticed from the Figure, all the three methods provide a good separation among the two clusters. To provide a deeper analysis of the quality of the clustering approaches, it has been inspected the distribution of the six target activities between the clusters, which are displayed in Figure 23.

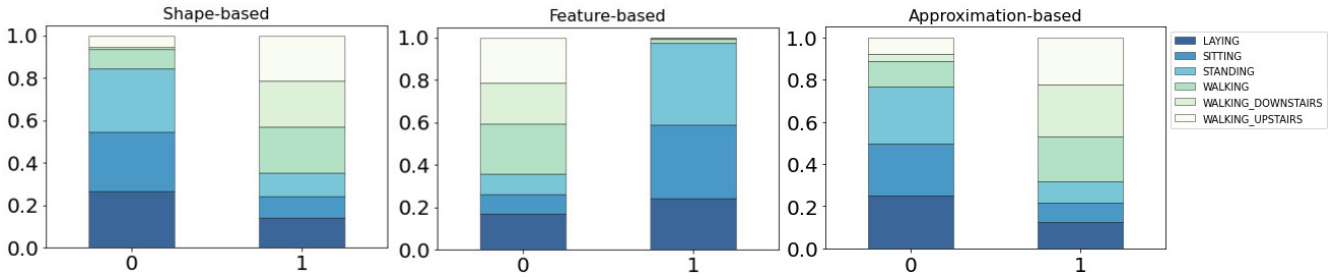



*Figure 23* Distribution of the activities among the clusters

As is possible to observe from the images, the algorithm seems to be able to distinguish in an acceptable way between static and dynamic activities in all the three approaches. The best cluster resulted to be cluster 1 obtained with feature-based approach, which is quite totally composed by static activities.

## 6.2 Motifs and Anomalies

The following step of the study has been motifs and anomalies discovery, which can be respectively defined as frequently appearing patterns in a time series and subsequences that are maximally different to all the rest of the signal.

To provide a representative analysis of motifs and discords, they have been detected through the Matrix Profile of the mean of the components of cluster 0 and 1 obtained using the Feature-based clustering method.

As shown in Figure 23 in the previous section, cluster 1 is quite totally composed by *Laying*, *Sitting* and *Standing*, reason why it has been judged interesting to inspect its trend as a representation of static activities. On the other hand, cluster 0 does not present a sharp division of the activities as cluster 1: it primarily contains time series labeled as *Walking*, *Walking_Downstairs* and *Walking_Upstairs* but it is not strictly representative of dynamic activities. For this reason, it has been decided to extract three time series for each dynamic activity and to compare them with results obtained on cluster 0.

The first analysis has been on the static cluster 1, which Matrix Profile has been built using a Time Window equal to 3. At this point, they have been detected motifs and discords: setting the parameter *max_motifs* equal to 10, five pairs of motifs have been discovered at indexes [21, 43], [37, 99], [113, 119], [39, 69], [31, 78]. Anomalies have been detected fixing the number of *k* equal to 5 and the *ex_zone* parameter equal to three and appear to be at the following time stamps:  95, 57,121, 26, 73. Both motifs and anomalies for cluster 1 are highlighted in Figure 24.

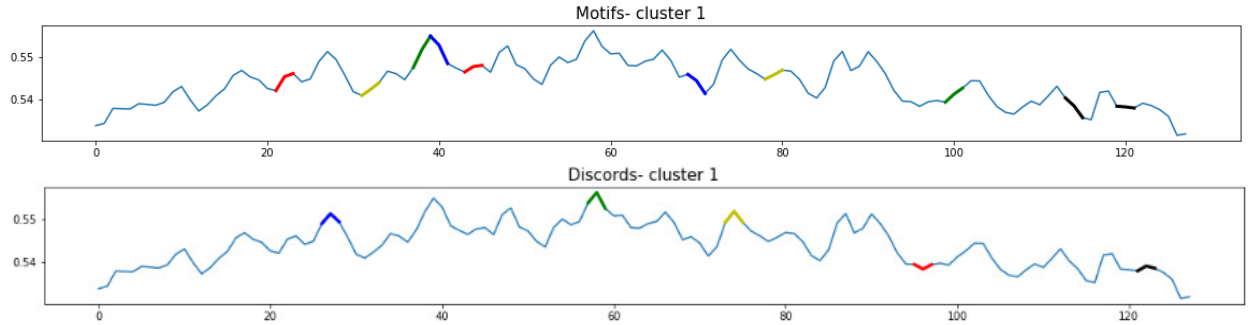



*Figure 24* Motifs and discords for cluster 1

Assuming that observed analysis on cluster 1 could be considered representative for static activities, the second part of the study has been focused on dynamic classes. As for cluster 1, Matrix Profile has been built using a Time Window equal to 3 and has been used to discover motifs and discords, which are reported in Figure 25.

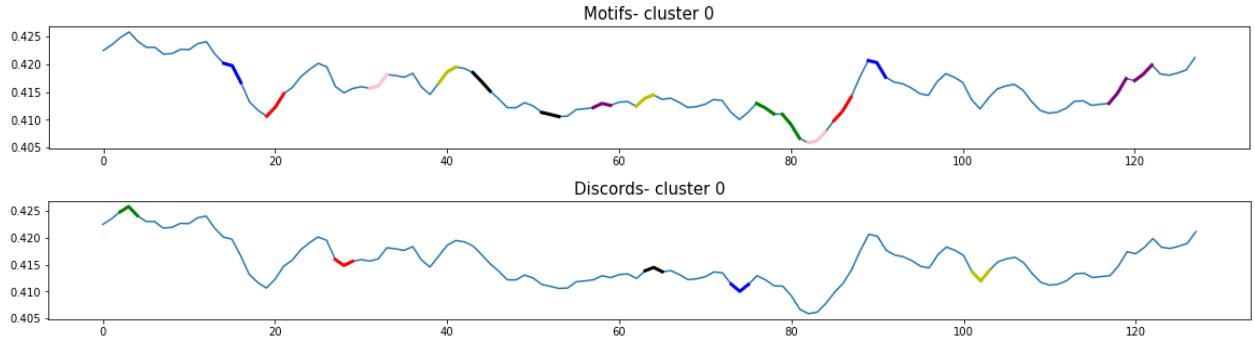



*Figure 25* Motifs and discords for cluster 0

To obtain the presented results, parameters have been left as for cluster 1. They have been detected seven motifs at indexes [19, 85], [76, 79], [43, 51], [14, 89], [39, 62], [31, 82], [57,117, 120] and five anomalies at time stamps 27, 2, 63, 73 and 101.
At this point it has been selected from the three extracted time series of dynamic activities one of class *Walking_Downstairs*, considered to be the "more dynamic" activity among all, in order to compare it with the results on cluster 0.

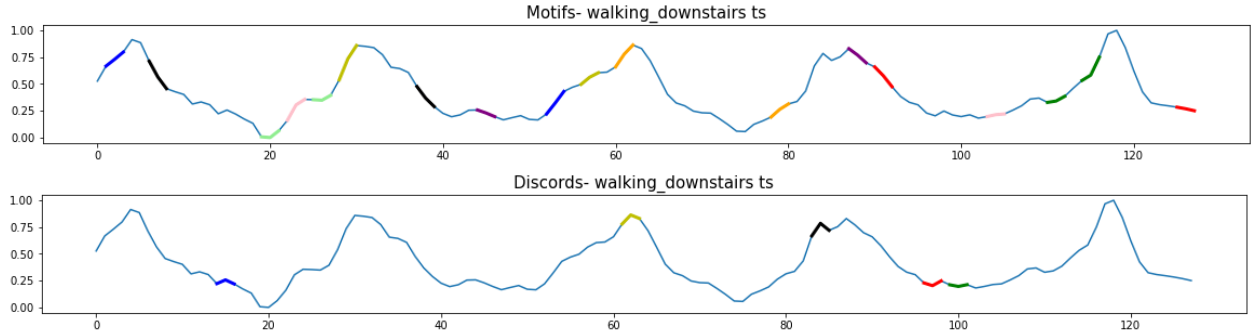



*Figure 26* Motifs and discords for *Walking_Downstairs* ts

Also in this case parameters have been left as the previous settings. As observable from Figure 26, nine pairs of motifs have been detected at [90, 125], [110, 114], [6, 37], [1, 52], [28, 56], [22, 103], [44, 87], [60, 78], [19, 25]; whereas 5 anomalies have been located at 96, 99, 83, 14, 61. Observing motifs discovered in the *Walking_Downstairs* time series, it is possible to assume that a dynamic activity as the analyzed one presents long ascend and descendent repetitive patterns, evidence that is possible to partially notice also in cluster 0's motifs (in pairs [19, 85] or [14, 89]). On the other hand, it might be assumed that the smoother behavior in the other intervals of cluster 0 is influenced by the static activities' contribution to the cluster composition.

## 6.3 Shapelets extraction

Shapelets, which are parts of the time series that are discriminative for a certain class, have been extracted in this part of the analysis in order to use them for classification task (Section 6.4) and to analyze their more interesting aspects. The number and length of shapelets to extract has been computed using the *grabocka_params_to_shapelet_size_dict* from the *tslearn* library, defining r=5 and l=0.1 and obtaining as shapelets dictionary {12: 6, 24: 6, 36: 6, 48: 6, 60: 6}. This dictionary has been used to learn shapelets through *tslearn ShapeletsModel* using as optimizer *Adam* from *Keras*. From the moment that extracted subsequences had five different lengths, for a clearer visualization it has been applied K-means clustering on the ones of the

same length. To discriminate the shapelets in a more efficient way, it has been identified 3 as the best number of k clusters, in order to erase the ones which presented the same shape and the same mean value. Clustering results are shown in Figure 27.

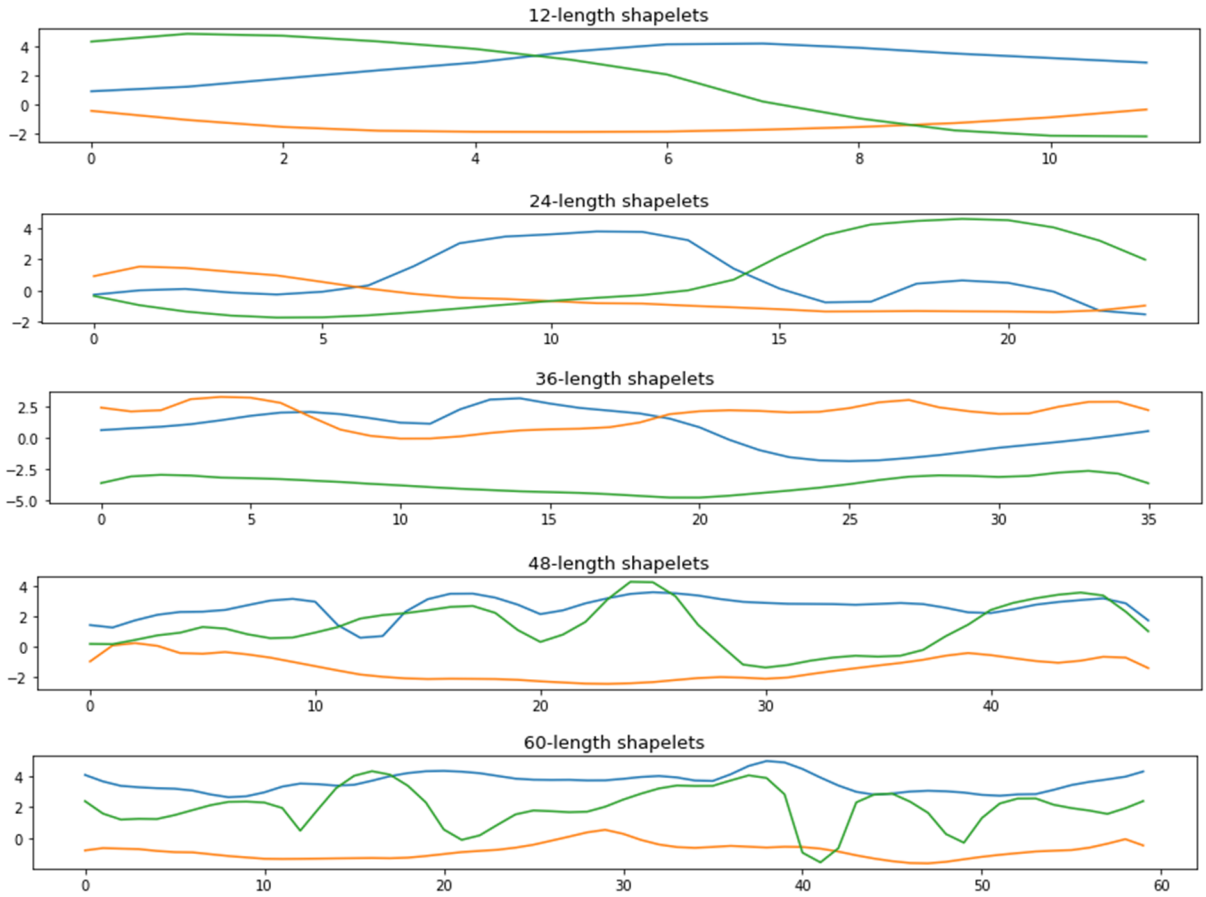



*Figure 27* Clustering on shapelets of different length

With the aim to inspect the more discriminative shapelets respect to the six activities classes, the distances computed with the *transform* method of the *ShapeletModel* have been used to plot the heatmap shown in Figure 28. On the x axis have been represented the 30 extracted shapelets, whereas on the y one the 5146 records of the split training set, which have been sorted using the *argsort* function respect to the six target variables. According to the labels assigned from the creators of the dataset, activities have been ordered in the following way: *Walking*, *Walking_Upstairs*, *Walking_Downstairs*, *Sitting*, *Standing* and *Laying*.The assumption behind the heatmap is that if a shapelet is close to records belonging to one class and far from all the records of the other classes, it will be more discriminative and influential for that specific class. From the Figure, where closeness is represented by darker colors, is particularly clear the strong influence that some shapelets have on the last class,

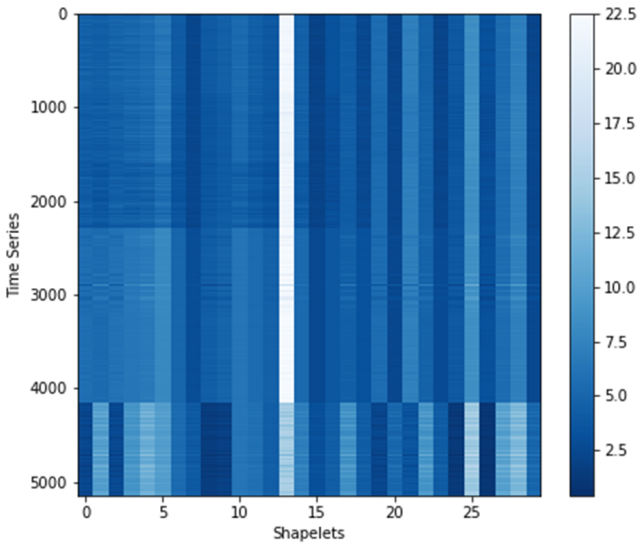



*Figure 28* Heatmap of distances between shapelets and time series

*Laying*, respect to the other five, as expected from the moment that this class is the one which provides better classification results.

## 6.4 Time Series Classification

The first classifier tested for the Time Series Classification has been the *K-nearest neighbors (K-nn)* from the *pyts* library, comparing the usage of three different distance functions: *Euclidean, Manhattan* and *Distance Time Warping* (*DTW*). For the hyper parameters tuning, they have been implemented three different Grids Search resulting the ones shown in Table 25 as best parameters. Results have been evaluated through accuracy score, which is 0.73 with *Euclidean*, 0.74 with *Manhattan* and 0.79 with *DTW*. The best metric to use has shown to be *DTW*, even if the other distances are less expensive from a computational point of view and obtain quite its same performances. For this reason, for successive classification trials with *K-nn* it has been considered the *Manhattan* distance.

| *Distance* | *N of neighbors* | *Weights* |
|---|---|---|
| *Euclidean* | 4 | distance |
| *Manhattan* | 9 | distance |
| *DTW* | 20 | distance |

*Table 25* Best parameters tuning for *K-nn* with different distances

The second classification task has been to classify the series according to a shapelet-based representation, using the extracted shapelets exploited in section 6.3. It has been used the *ShapeletModel* classifier, both in the basic approach than in the distance-based one, using *Decision Tree* and *K-nn* as classifiers. Successively it has been tested the usage of a *Convolutional Neural Network* (*CNN*) to classify the time series, with a *batch size* of 250 and 150 *epochs*. The last algorithm to be tested has been *Rocket*, which stand for Random Convolutional Kernel Transform. With this model, feature are transformed to train a linear classifier, which in this case is *RidgeClassifier* set with *alphas=np.logspace(-3, 3, 10)* and *normalize=True*.

Evaluation through Accuracy score and parameters tuning for all the tested classifiers are reported in Table 26; all the performances indicators refer to the original test set.

| | *Parameters* | *Accuracy* |
|---|---|---|
| K-nn | {n_neighbors=20, weights='distance', metric='dtw'} | 0.79 |
| Shapelet classifier | {optimizer=Adam, weight_regularizer=.01, max_iter=500, verbose=1} | 0.80 |
| Shapelet distance-based classifier + Decision Tree | {criterion: 'entropy', max_depth: 10, min_samples_leaf: 3, min_samples_split: 4} | 0.81 |
| Shapelet distance-based classifier + K-nn | {n_neighbors=4, weights='distance', metric='manhattan'} | 0.80 |
| CNN | {monitor='loss', factor=0.1, patience=50, min_lr=0.0001} | 0.84 |
| Rocket | {kernels=10.000} | 0.71 |

*Table 26* Evaluation of different Time Series Classifiers on original test set

As is possible to ascertain from the Table, the best performing algorithm is *CNN*, with an accuracy of 0.84. Nevertheless, even if the model was expected to reach more performing results, *CNN* returned performances really similar to the ones obtained with simpler classifiers, resulting to be unnecessary complicated. Unexpectedly, in this default setting *Rocket* came out to be the worst classifier on time series, providing an accuracy of 0.71, lower than the standard *K-nn* which reaches 0.79.

Comparing the classification results on the Time Series dataset with the ones on the structural features dataset used in Section 2, it is notable that classifiers perform better on the second one. The best classifier on this dataset reaches an accuracy of 0.84, a value which is similar for all the other models, whereas mostly all the classifiers on the structural dataset reach an accuracy higher than 0.90, with *SVM* and *Logistic Regressor* which reach the most accurate results (0.96).

# 7. Sequential Pattern Mining

This section of the project has been focused on the research of sequential patterns, which have been detected among the elements of Cluster 1 (composed by 2328 time series) obtained using the Feature-based clustering method explained in Section 6.1. Patterns have been searched inside Cluster 1 for two main reasons: first of all, the high dimension of the original dataset implied a great number of irrelevant observations and, moreover the cluster presents a balanced composition of static activities which could provide interesting outcomes. In order to make the sequential pattern mining process possible it was necessary to provide an approximation of the normalized time series, so that it would be easier to discover patterns in a more generic representation. The approximation has been made through the *SymbolicAggregateApproximation (SAX)* which requires two parameters: the number of segments to approximate the length of the time series and the number of symbols to use to represent the different trends of the curves. The number of segments has been set to 60 (less than the half of the original length) and the number of symbols has been defined equal to 30, from the moment that those values have shown the ability to capture the natural behavior of the time series without loss of information about the frequent little oscillations present in the curves.

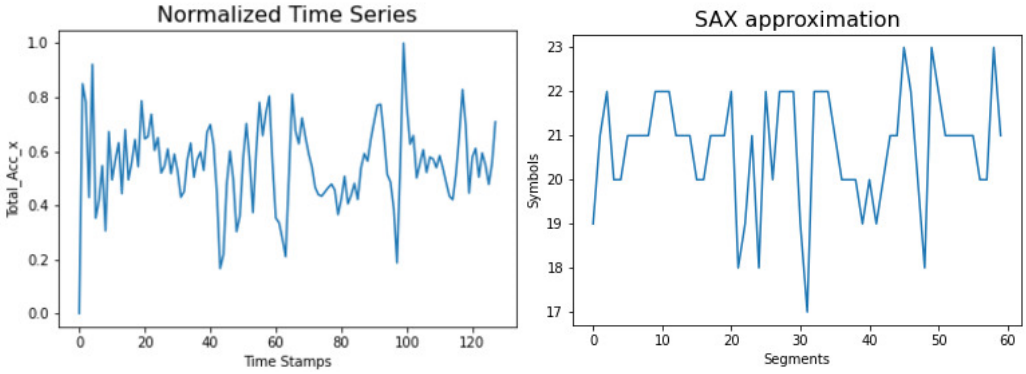



*Figure 29* Comparison between the first time series in Cluster 1 and the same one approximated by *Sax*

In order to discover if it was already possible to have a preview of the presence and the locations of the patterns in the selected signal, they have been extracted different sets of approximated time series in a range from 5 to 50 with an interval equal to 5. In Figure 30 are shown trials on 5 and 15 time series, from which is notable an obvious concentration of minor swings in the symbol's interval 20-22 on the y axes.

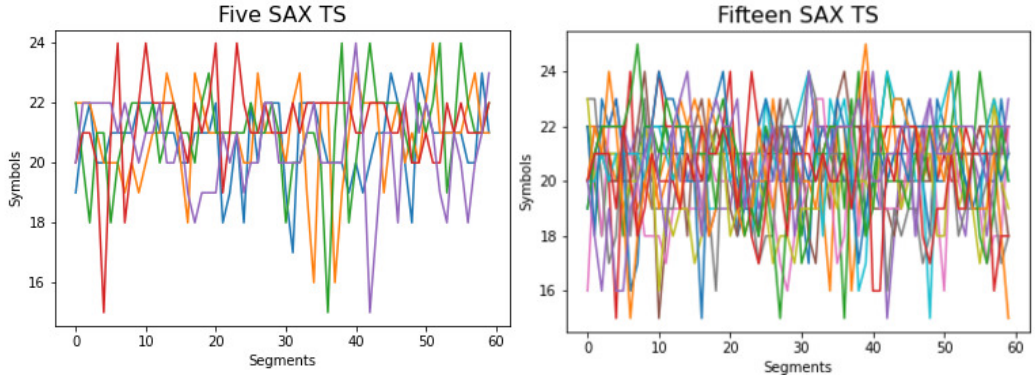



*Figure 30 SAX* approximated time series shown in different set dimensions

Pattern research has been implemented using the *PrefixSpan* library. The sequential closed patterns have been extracted considering different *min_support* and *min_lenghts* thresholds, providing results illustrated in Table 27.

| | N. of patterns | | |
|---|---|---|---|
| | *Min_length=3* | *Min_length=5* | *Min_length=6* |
| *Min_support=80%* | 1473 | 1134 | 729 |
| *Min_support=90%* | 105 | 29 | 2 |

*Table 27* Different number of closed patterns returned with different *min_length* and *min_support* thresholds

Sequential patterns which contained less than 3 elements have been ignored because considered meaningless. It has been chosen to overlook patterns with a *min_support* threshold lower than 80% because of the massive numbers of patterns detected (with a *min_support* of 50% they have been extracted hundreds of thousands patterns). As expected, from Table 27 it is deducible that the number of frequent sequential patterns grows at the decreasing of both the *min_support* and the *min_length* thresholds. On the other hand, a high value of *min_support* implies more frequent patterns but also less significant ones. It is interesting to notice that the most frequent patterns are composed by different combination of the same symbols (19, 20, 21, 22), which are the ones composing the interval containing the minor swings on the y axes shown in Figure 30. Table 28 contains samples of the extracted patterns, each preceded by its frequency.

| | *Min_length=3* | *Min_length=5* | *Min_length=6* |
|---|---|---|---|
| *Min_support=80%* | (2182, [22, 21, 20]) | (2112, [21, 22, 21, 21, 21]) | (2123, [21, 21, 21, 21, 21, 21]) |
| *Min_support=90%* | (2121, [19, 21, 20]) | (1967, [19, 21, 20, 21, 21]) | (1873, [21, 22, 21, 20, 21, 21, 20]) |

*Table 28* Sample of extracted closed pattern with different *min_length* and *min_support* thresholds

# 8. Advanced Clustering

For the advanced clustering task they have been tested the *X-means* and the *OPTICS* algorithms on the structural dataset (same used in the first five Sections) normalized using *MinMaxScaler()*.

The first approach to be implemented has been *X-means* from the *pyclustering* library, a *K-means* extension which improves its performances and has as advantage the fact that the algorithm is able to define the optimal number of clusters on its own. For what the setting is concerned, they have been set the number of clusters from a minimum of 2 to a maximal of 6.

From the moment that the visualization of the output needed a 2-dimensional representation, in order to provide more significant results it has been preferred to reduce the set of data in two dimensions using the *t-SNE* method, rather than randomly select two variables.

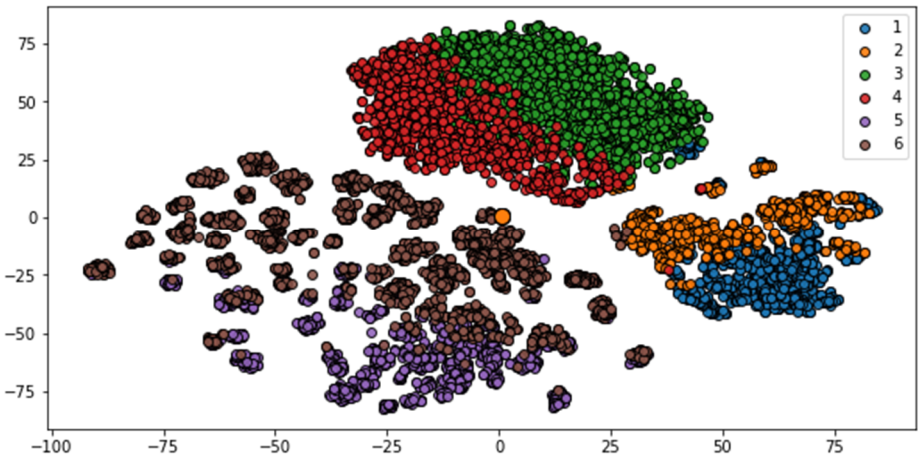
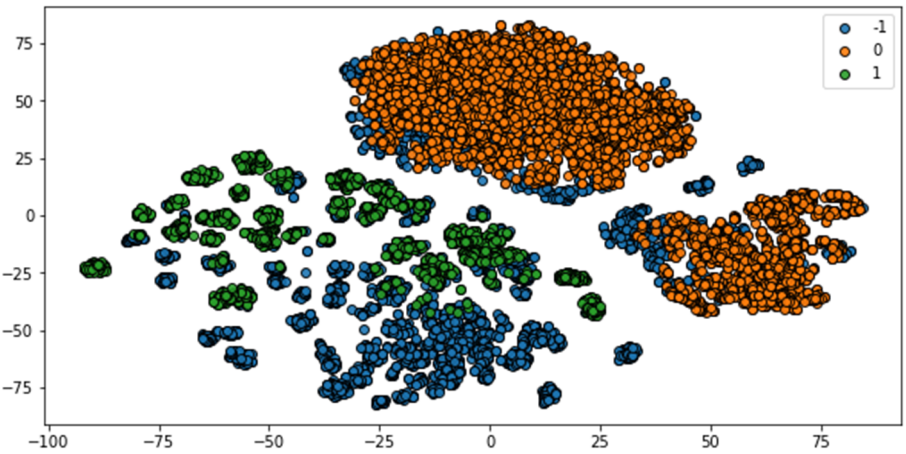



*Figure 31 X-means* clusters represented on 2-dimensional dataset through *t-SNE*

Observing Figure 31, it is possible to ascertain that the six clusters have a distinguishable distribution and a quite insignificant overlap.

The second advanced clustering algorithm to be applied has been *OPTICS*, a density-based approach implemented using the *sklearn* library. *OPTICS* is an evolution of the *DBSCAN* algorithm which uses the reachability to measure inter-points distances and solve the problem of managing different densities regions.

Taking into consideration the parameters tuning, *OPTICS* has been tested both with *xi* and *dbscan cluster_method* using *Euclidean* metrics. Both the clustering methods returned the same outcomes, so evaluation that will be provided refer to both of them. For what the remaining parameters are concerned, after different trials optimal results have been reached setting *epsilon* equal to 2.0 and *min_samples* as 30; other parameters have been left as default settings.




Considering the line of reasoning previously explained for *X-means*, the visualization of the clusters obtained through *OPTICS* has been provided reducing the dataset in a bidimensional way using the *t-SNE* method. Algorithm returned three clusters, shown in Figure 32.

*Figure 32 OPTICS* clusters represented on 2-dimensional dataset through *t-SNE*

To provide a more complete evaluation, they have been computed the Silhouette scores for both the algorithms and analyzed the composition of the clusters, as reported in Table 29.

| *Algorithm* | *Silhouette score* | *Composition of clusters* |
|---|---|---|
| *X-means* | 0.28 | [828, 627, 1590, 1011, 1213, 2083] |
| *OPTICS* | 0.50 | [2616, 3394, 1342] |

*Table 29* Comparison between advanced clustering algorithms

As evaluable from the Table, *OPTICS* returned a higher silhouette value and a sufficient balanced distribution of the elements inside the three clusters, even if 2616 records have been considered as noise.

Although *X-means* silhouette appears to be lower than the *OPTICS* one, the allocation of the points inside clusters is considerable satisfying.

# 9. Explainability

This section is centered on providing explanations for the classification obtained on the structural dataset using the *Random Forest* classifier exposed in Section 4.2.4. For what the hyperparameters tuning of the algorithm is concerned, the setting has been kept the same as

for the previous application. Explanations provided have been extracted using both global and local methods; all the following results are referred to validation set.

## 9.1 Global explanation

Global explanations have been provided using *Partial Dependence* from the *Skater* library as model agnostic global interpretation algorithm, in order to describe the marginal impact of a feature on model prediction. For this analysis, it has been used the *plot_partial_dependence* method, which computes partial dependence of one or more selected variables. In this case, it has been chosen to inspect the effect on the classification algorithm of the variable *tBodyAcc-mean()-Y*, which is the mean of the Y acceleration from an accelerometer signal related to body motion.

Partial Dependence has been plotted for all the six variables; in Figure 33 are presented the plot of *Walking_Upstairs* and *Walking_Downstairs* which are the most interesting to comment. From the Figure it is possible to notice that the two classes return specular results: at the increasing of the mean acceleration of Y axis there is a decreasing probability to predict *Walking_Upstairs* and a growth of the probability to predict *Walking_Downstairs*; moreover, at the same point on the x axis (between -0.05 and -0.04) the prediction probability become steady for both classes.

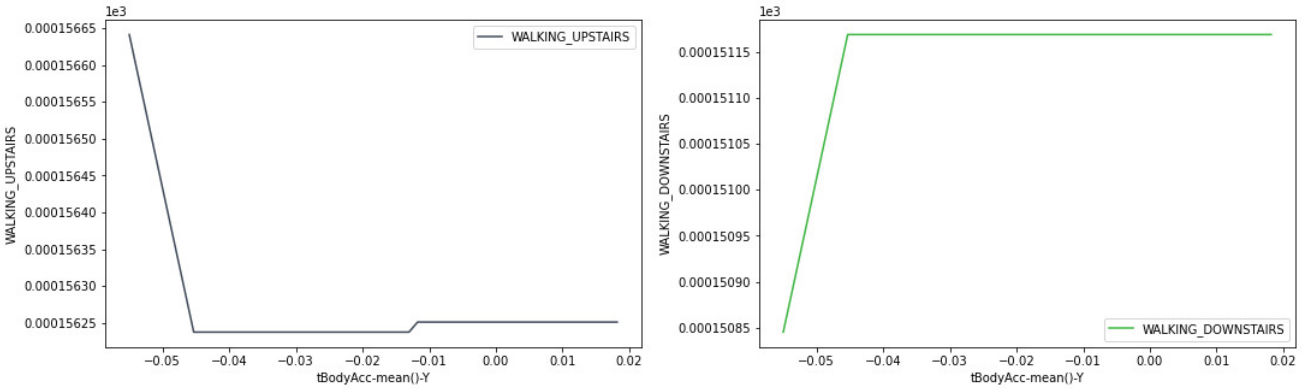



*Figure 33* Effect of t-BodyAcc-mean()-Y on the prediction of classes *Walking_Upstairs* and *Walking_Downstairs*

Similar symmetry, even if in a less sharp way, has been exploited in the partial dependency plots of *Sitting* and *Standing*, shown in Figure 34.

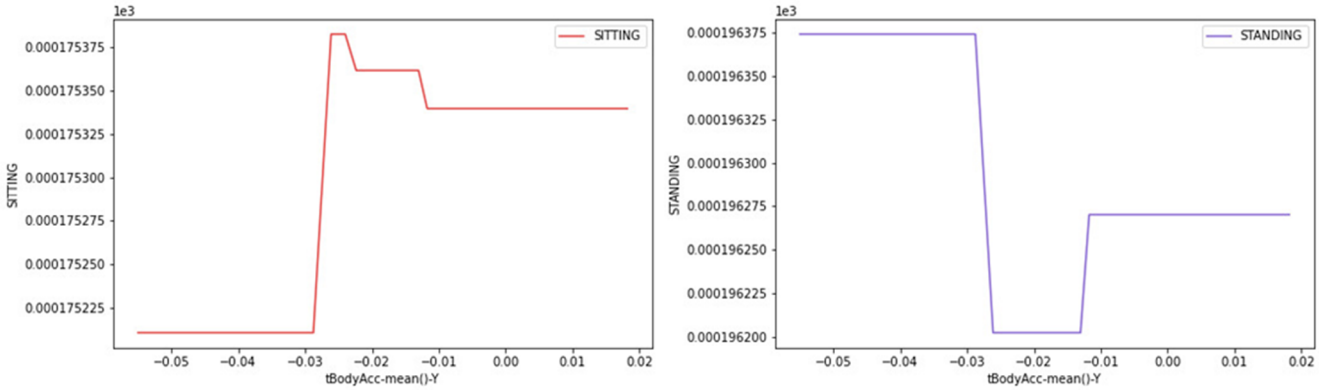



*Figure 34* Effect of t-BodyAcc-mean()-Y on the prediction of classes *Sitting* and *Standing*

At this point it has been tested another model agnostic global explainer, *TREPAN*, which approximates a BlackBox using a *Decision Tree* as surrogate model. The first three levels of *Decision Tree* classifier are shown in Figure 35.

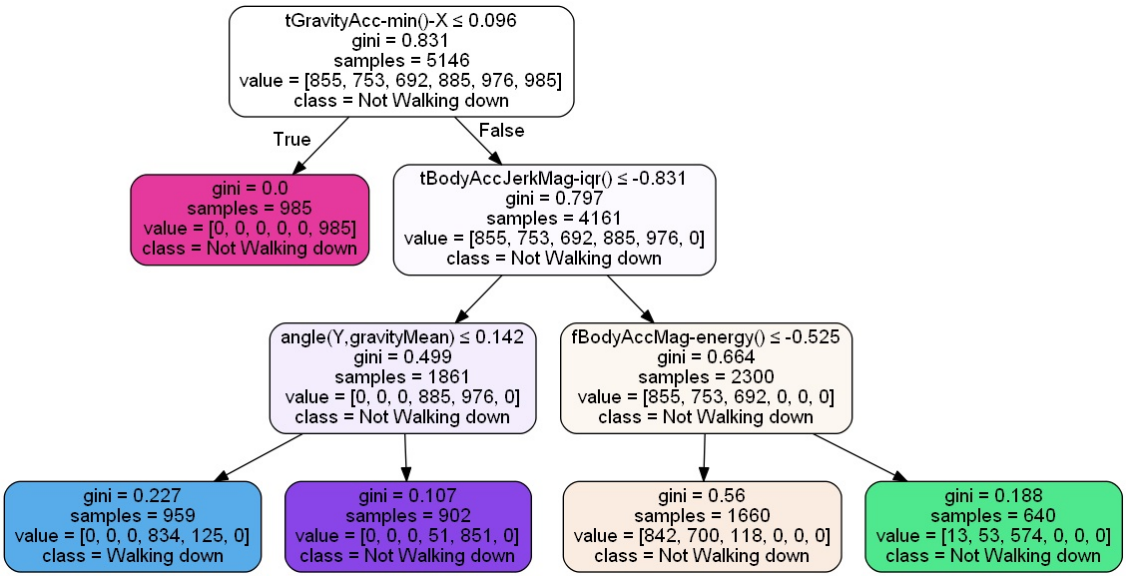



*Figure 35* First three levels of *Decision Tree* which approximates the *Random Forest* behavior

From the Figure it is possible to observe that the most informative variable to bisect records classified as *Walking_Downstairs* or labeled with other classes is *tGravityAcc-min()-X*, which defines the first split.

## 9.2 Local explanation

As local explanation methods they have been used *LIME* (*Local Interpretable Model-agnostic Explainer*) and *LORE* (*LOcal Rule-based Explainer*) algorithms.

The first one to be tested has been *LIME*, which locally approximates the Blackbox's behavior using a Linear Regressor as surrogate model. The algorithm returns explanation through feature-importance, establishing which variables affect more the classification.

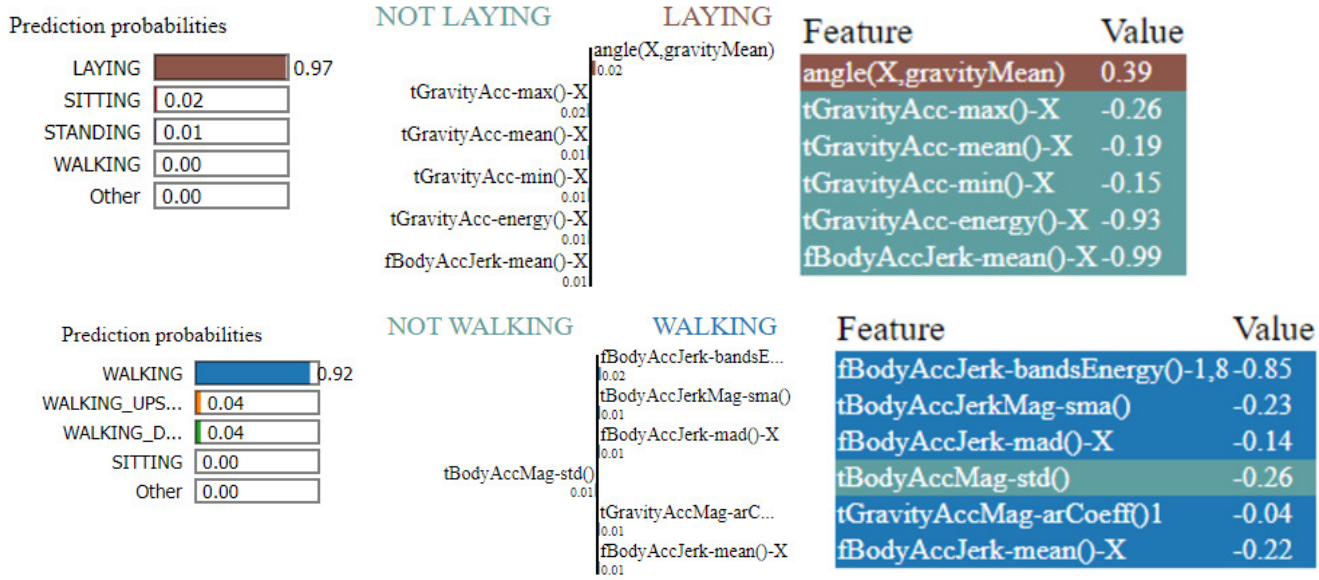



*Figure 36* Feature Importance for two records belonging respectively to classes *Laying* and *Walking*

After different trials to detect variables which have a high effect on the instance's classification, the explanations displayed in Figure 36 have been provided for two records correctly predicted and randomly selected from the validation set. Results have been obtained using the

*LimeTabluarExplainer*, taking into consideration the six most informative features for each example.

Comparing the evidence displayed in the Figure with the features defined as informative returned by *Random Forest* (shown in Figure 16 in Section 4.2.4), it is possible to observe that some variables found for the examples have been detected also by the ensemble algorithm, even if with slightly different weights. Anyway, the difference in informativeness is quite indiscernible: *Random Forest* doesn't return features more informative than 0.05 and the most informative feature in the *LIME* example has a value of 0.02. These low values are justified by the great number (561) of variables in the dataset.

The last implemented explanation algorithm has been *LORE* which extends *LIME* by adopting a decision tree classifier as a local surrogate. The explanation is expressed in logical form, through a single rule to classify the instances (factual rule) and a set of counterfactual rules, which highlight what are the possible changes to undo the results. *LORE* explainer was applied on the same randomly selected records used to test *LIME* with a neighborhood sample of 1000.

| *Factual rule* |
| --- |
| { angle(X,gravityMean) > -0.02, tGravityAcc-max()-X <= 0.46, fBodyAcc-min()-Z <= -0.58, tBodyGyroJerk-arCoeff()-Z,2 <= 0.66 } --> { Activity: 6 } |
| { fBodyAccJerk-mean()-X > -0.26, fBodyAccJerk-bandsEnergy()-1,8 > -1.00, tGravityAcc-arCoeff()-X,2 <= 0.79, fBodyAccMag-energy() <= -0.54, tBodyGyro-mean()-Y > -0.29, tGravityAcc-correlation()-Y,Z <= 1.07, fBodyGyro-bandsEnergy()-41,48.1 > -1.13, fBodyGyro-energy()-X <= -0.56 } --> { Activity: 1 } |

*Table 30* Rules extracted by *LORE*

The first rule in Table 30 refers to the instance classified as *Laying* while the second to the record labeled as *Walking*. Extracted rules show to be coherent with the results displayed using *LIME*, sharing approximately the same important features to prove the classification of the records. *LORE* did not return any counterfactual rule for the considered instances.