UNIVERSITÀ DI PISA

Data Mining II Project

Analysis of the "Human Activity Recognition Using Smartphones" dataset

Data Mining
AY 2021/2022

Cosimo Faeti 636812
Riccardo Galarducci 637763

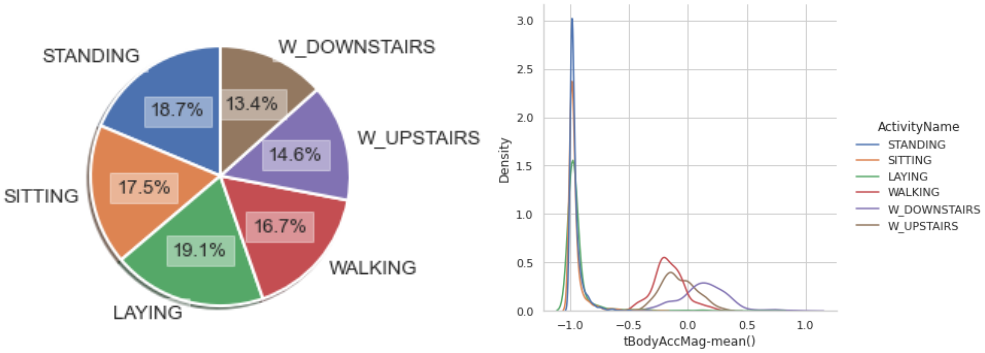
1 Exploratory Analysis & Data Cleaning

Human Activity Recognition Using Smartphones data set concern experiments that have been carried out with a group of 30 volunteers. Each person performed six activities (WALKING, WALKING UPSTAIRS, WALKING DOWNSTAIRS, SITTING, STANDING, LAYING) wearing a smartphone on the waist. Using the embedded accelerometer and gyroscope of the smartphone were captured 3-axial linear acceleration and 3-axial angular velocity. From the raw signals were extracted, using a sliding window, that correspond to a record in the data set, 561 features. Each record is also associated to the activity label and the identifier of the subject who carried out the experiment.

The data set is divided into a training set, composed by 7352 records, and a test set, constituted by 2947 records.

Data Quality. We performed a rapid data cleaning checking for duplicates and missing values. The results were negative.

Exploratory Analysis. We have then proceed an exploratory analysis in particular with respect to the target variable *Activity*. Figure 1a shows that the composition of the training set is quite balanced according to the six activity labels. Figure 1b shows that the activity labels can be divided into motion and static activities because this two groups produce very different distributions according to the 561 features.



(a) Activity composition

(b) Distribution plot

2 Simple Classification & Imbalanced Learning

In this section we have defined two classification tasks:

1. **Multiclass classification problem** with *Activity* as target variable ;
2. **Binary classification problem** with activity 2 which corresponds to *Walking Upstairs* as class 1 and all the others activities as class 0.

Then we have solved them using two classifiers, **Decison tree** and **k-NN**, and we have compared their performances on the test set using several evaluation metrics.

2.1 Multiclass classification problem

Decision tree classifier. We have used the *randomized search algorithm* in order to tune decision tree parameters: splitting criterion, maximum depth of the tree, minimum samples split and minimum samples leaf. The best parameters configuration found are entropy as splitting criterion max depth and min samples split equal to 3.

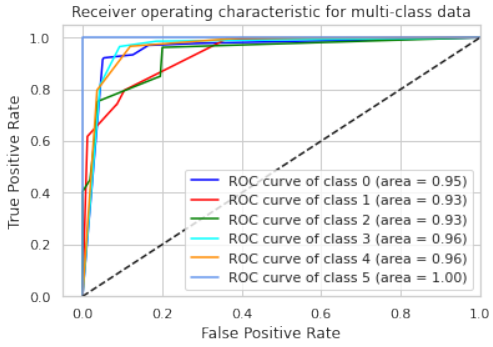
K-nearest neighbor. As well as for decision tree we have used the *randomized search algorithm* to tune k-NN parameters: k number of neighbours and the weight function used in prediction. The best parameters configuration found by the algorithm are k equal to 8 and weight points by the inverse of their distance.

Performance evaluation. We have evaluated the performances of the two classifiers on the test set using the *accuracy score*, and we have exploited the weighted average in order to take into account label imbalance, of *precision*, *recall* and *F1-score*. We have also compared the ROC curve of the two classifiers.

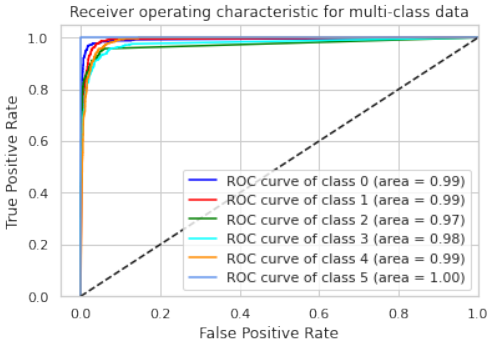
As can be observed both from Table 1 and from Figure 2, k-NN hits quite impressive results, and all its score are above the ones reached by the decision tree.

	precision	recall	f1-score	accuracy
Decision Tree	0.80	0.84	0.82	0.73
k-NN	0.92	0.90	0.91	0.90

Table 1: Performance evaluation metrics



(a) Decision Tree



(b) k-NN

Figure 2: ROC curves

2.2 Binary classification problem

In this second classification problem we are dealing with the binary classification problem defined above, where records that have activity *Walking Upstairs* belong to *Class 1* and all the other record belong to *Class 0*. As we can see in Figure 3 we are in an unbalanced scenario where there is a big disproportion between the two classes.

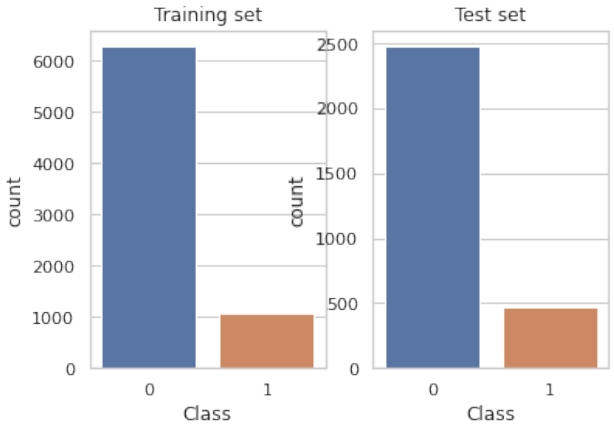


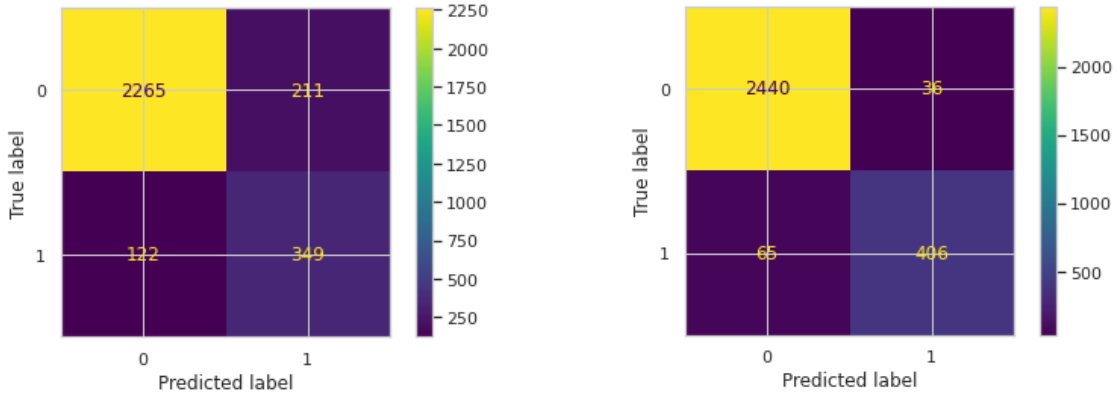
Figure 3: Countplot target variable

We have solved this task, as well as for the multiclass classification problem, with **Decison tree** and **k-NN**.

The first step is to tune the parameters for both the classifiers using the *randomized search algorithm*:

- **Decision tree** best parameters configuration is: entropy as splitting criterion, max depth equal to 3, min samples leaf equal to 5 and min samples split equal to 6;
- **k-NN** best parameters setting is: k equal to 8 and all points in each neighborhood with uniform weights .

In Figure 4 are reported the confusion matrices for the two classifiers. We can see at first glance that k-NN seems to get better results in the prediction of *Class 1* which is the class of interest.



(a) Decision Tree

(b) k-NN

Figure 4: Confusion matrix

Our first impression is confirmed by the evaluation measures reported in Table 2, where precision, recall and F1-score are related to predictions of *Class 1*. We can see that k-NN score are higher in all the four measures.

	precision	recall	f1-score	accuracy
Decision Tree	0.62	0.74	0.68	0.88
k-NN	0.92	0.86	0.89	0.97

Table 2: Performance evaluation metrics

In Figure 5 are plotted the Precision-Recall curves with respect to *Class 1*. We can see that k-NN curve dominates Decision tree curve for each threshold.

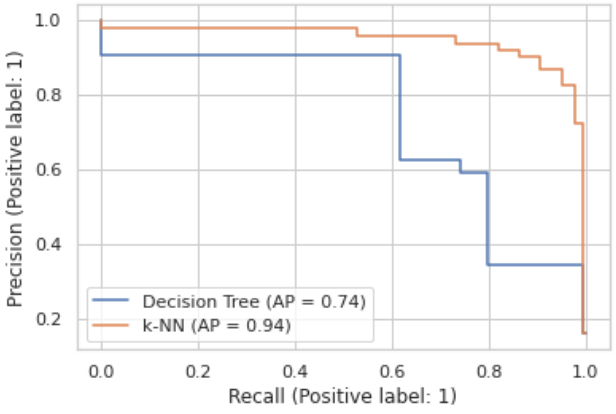


Figure 5: Precision-Recall curve

2.3 Imbalanced Learning

The binary classification problem of the previous section was performed in an unbalanced setting. In this section we have exploited three main methods for balancing the dataset and we have solved the same classification problem using the same classifiers as before in order to test if there is an improvement on their performances. The three balancing techniques we have used are:

1. **Random Undersampling**. Under-sample the majority class;
2. **Random Oversampling**. Over-sample the minority class;
3. **SMOTE**. It generates *synthetic* objects of the minority class in the neighborhood of existing positive instances.

	Decision Tree			
	Imbalanced	Undersampling	Oversampling	SMOTE
Accuracy	0.89	0.88	0.87	0.87
Precision	0.62	0.61	0.56	0.56
Recall	0.74	0.80	0.85	0.85
F1-score	0.68	0.69	0.67	0.67
	k-NN			
Accuracy	0.97	0.94	0.96	0.95
Precision	0.92	0.77	0.82	0.75
Recall	0.86	0.99	0.97	0.99
F1-score	0.89	0.86	0.89	0.85

Table 3: Imbalanced learning

In Table 3 we can capture clearly the variations over the evaluation measures. While accuracy and F1-score tend to remain stable, the behaviours of precision and recall is opposite: precision scores decrease, instead recall increase. This is what we were looking for because a classifier with a higher recall has a higher chance of correctly identifying the positive instances, even sacrificing a bit the precision score.

3 Outliers Detection

The purpose of this section is to identify the top 1% outliers using three different methods:

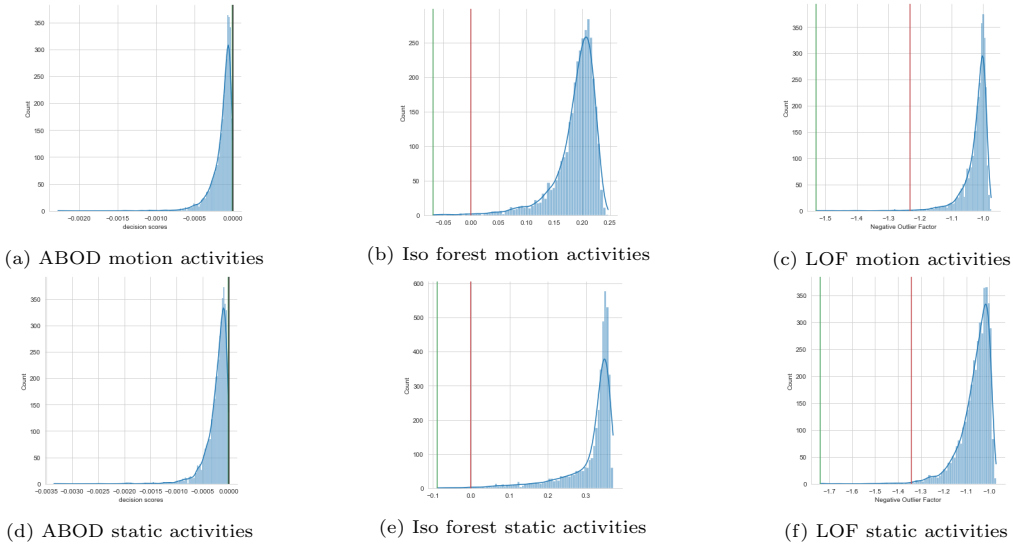
1. **Angle-Based Outlier Degree**: which belongs to the high-dimensional approaches;
2. **Isolation Forest**: which belongs to the model-based approaches;
3. **Local Outlier Factor**: which belongs to the density-based approaches.

We have chosen these three methods as they are the most suitable for operating with a dataset with such high dimensionality.

Before fitting the three algorithms to the training set, we have first grouped the instances belonging to the motion activities and the instances belonging to the static activities. We have done this preprocessing step because we realized that ABOD and Isolation Forest results were biased by this dichotomy present in the original dataset.

Then we have applied the three algorithms to both the two groups looking for the top 1% outliers scores. In Figure 6 we can see the distributions of the outlier score returned by the three algorithms. The portion of the distribution in between the green and red lines corresponds to the records with higher outlier scores.

We compared the records identified as outliers by the three methods. Regarding records belonging to motion activities, only 1 record out of 17 found is common among the three methods, while ABOD and Isolation Forest, which are probably the most suitable methods among the three, have 6 out of 17 records in common. For what concern static activities we have that 4 out of 21 record are in common among the three methods, but if we consider just ABOD and Isolation Forest the number of records in common increases to 7.



(a) ABOD motion activities

(b) Iso forest motion activities

(c) LOF motion activities

(d) ABOD static activities

(e) Iso forest static activities

(f) LOF static activities

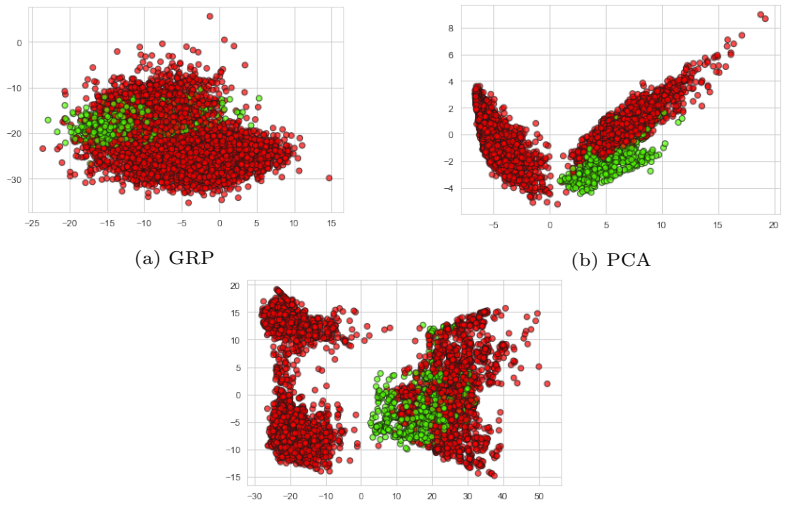
Figure 6: Outliers score distributions

We have decided to remove from the data set the records marked as outlier from both the Isolation Forest and the ABOD method because, as we have said previously, are the ones more suitable to work on a dataset with high dimensionality like the HAR dataset.

4 Dimensionality Reduction

In order to be able to visualize the dataset in two dimensions and to perform some tasks that will be discussed later, we have tested three dimensionality reduction techniques:

1. **Gaussian random projection** that belongs to random subspace projection techniques;
2. **Principal component analysis (PCA)**;
3. **IsoMap** which belongs to the broad family of multidimensional scaling.



(a) GRP

(b) PCA

(c) IsoMap

Figure 7: Dimensionality Reduction

In Figure 7 we can see that, both in PCA and IsoMap, points assume two separate distributions in the space; this is due to the dichotomy between static and motion activities. The two different colours green and red reflect the binary target variable we have defined previously in the Simple Classification section.

We have chosen the IsoMap reduced space, to perform outlier detection and solve the binary classification task because ensures to maintain the intrinsic geometry of the data.

Outlier detection. We have used Isolation Forest, both the standard and the extended versions on the new reduced dataset. The possibility to plot the data in two dimensions allowed us to plot the score maps of the standard and extended versions and to grasp visually the differences between the two. In Figure 8 we can see clearly the linear orthogonal cuts of the standard version and the more flexible cuts of the extended version.

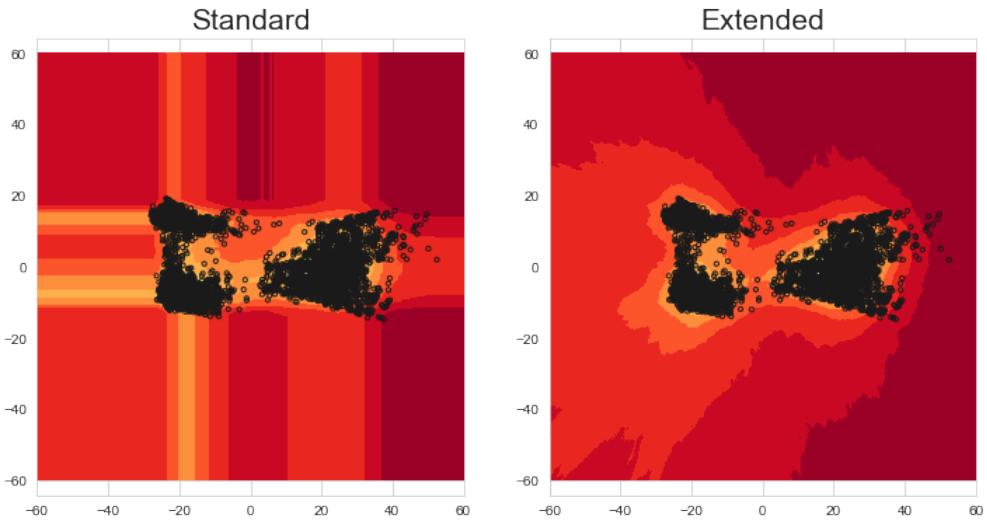
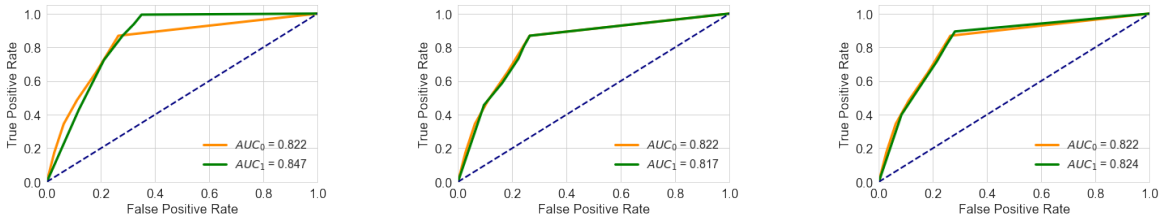


Figure 8: Isolation forest score map

Classification & imbalanced learning. We have repeated the steps we have done in the previous Imbalanced Learning section. We have first solve the binary classification task with a 5-NN with uniform weights on the imbalance dataset. Then we have solved the same classification tasks but balancing the dataset with *Random undersampling*, *Random oversampling* and *SMOTE*. We have analyzed the results comparing the ROC curves of the classifier on the unbalanced dataset (orange ROC curve) and on the balanced dataset (green ROC curve). As we can see in Figure 9 there are no improvements after balancing the data, the ROC curves overlap in all the three cases.



(a) Random undersampling (b) Random oversampling (c) SMOTE

Figure 9: ROC curve

5 Advanced Classification Methods

In the following section we solved the binary classification task defined in section *Simple Classification & Imbalanced Learning* with advanced classification methods.
In an early stage, the classification task was carried out on the original dataset which was an analysis done just out of curiosity as the result could not be sufficient due to the presence of many methods that are not suitable for working with unbalanced data; while methods that are able to handle them still provided excellent results, such as the Support Vector Machine. Therefore, as the dataset is strongly unbalanced, i.e. approximately 85% negative class and 15% positive class, it was deemed necessary to apply *Random Oversampling* in order to balance the training data.

5.1 Naive Bayes

The classifier discussed is the **Gaussian Naive Bayes**, in which the assumption is that the continuous values of the predictor variables, associated with each class, are distributed according to a Gaussian distribution.
We applied the *GaussianNB* algorithm on the training set composed by continuous variables, for the classification task.

The model shows an accuracy of 0.77 on the test set, moreover the AUC-ROC is 0.867 and the AUC-PrecisionRecall is 0.71.
Table 4 shows the results of the performance evaluation metrics for the test set. The results show a substantial difference for the two classes in precision and recall. The precision of the model is very low for the positive class, while it is perfect for the negative. As far as Recall is concerned, the classifier gives us important results for *Class 1* as a high value of Recall translates into a high chances of correctly identifying positive instances.

	precision	recall	f1-score
0	1.00	0.73	0.84
1	0.41	1.00	0.58
accuracy			0.77

Table 4: Results of Naive Bayes Classifier

Figure 10 shows the Precision-Recall curve of the model where it can be seen that the positive class curve is highly affected by the low precision values.

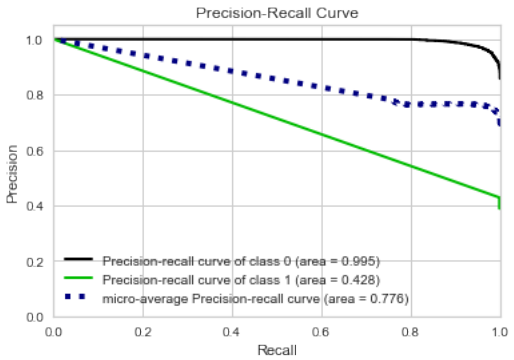


Figure 10: Precision Recall curve Naive Bayes

5.2 Linear Regression

In this section we solved two linear regression problems:

- **Simple linear regression**. We selected the variable *tBodyAcc-mean()-X* as independent variable X and *tBodyAcc-mean()-Y* as dependent variable Y (Figure 11 shows the plot);
- **Multiple linear regression**. We selected the variable *tBodyAcc-mean()-Y* as dependent variable and the remaining variables as independent variables. Moreover we regularized it using Ridge and Lasso methods.

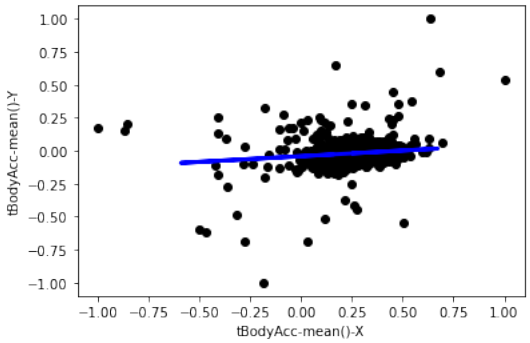


Figure 11: 2-dimensional Linear Regression

In Table 5 are reported the results in terms of *Coefficient of determination (R^2)*, *Mean Squared Error (MSE)* and *Mean Absolute Error (MAE)*.
As we can observe for the Simple Linear Regression model R^2 is negative which means that it does not follow the trend of the data (Figure 11), which is due to the fact that the variance predicted by the model is greater than the actual variance. On the other hand, for Multiple Linear Regression the model without regularization yielded the best results for all three metrics. The coefficient of determination (R^2) shows that the model explains 70% of the variability of Y; MSE remained the same value (0.001); MAE slightly increased (0.024).

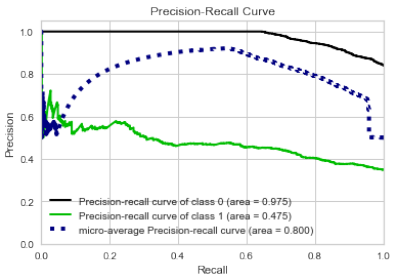
	Simple Linear Regressor	**Multiple Linear Regressor**		
		No Regularization	*Ridge*	*Lasso*
R^2	-0.237	0.703	0.698	0.000
MSE	0.166	0.001	0.001	0.004
MAE	0.178	0.024	0.024	0.033

Table 5: Results of Linear Regression

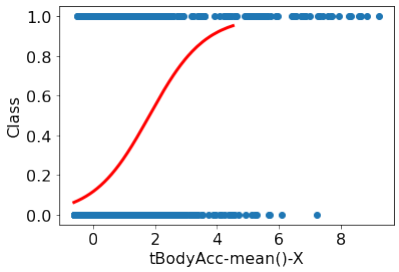
5.3 Logistic Regression

Afterwards, logistic regression was applied as classification method. In order to simplify the process, we left the default l_2 regularization in place and the default solver *lbfgs* which allows to get convergence more quickly.
We selected *tBodyAcc-mean()-X* as independent variable; as we can observe in Table 6 the logistic model obtained good result in terms of precision and recall only for the negative class, while for the positive class the results were worse, which means that the model, although working on a balanced dataset, has difficulty distinguishing when a person walking upstairs from the rest. This is clearly visible in the Precision Recall curve in Figure 12a, where once again the low precision values for the positive class heavily influence the curve.

Figure 12b shows the sigmoid logistic function considering the target variable *Class* and the independent variable *tBodyAcc-mean()-X*.



(a) PR curve for Logistic Regression



(b) Sigmoid logistic function

Figure 12: PR Curve Logistic Regression and Sigmoid logistic function

	precision	recall	f1-score
0	0.95	0.80	0.87
1	0.42	0.76	0.54
accuracy			0.80

Table 6: Results of Logistic Regression

5.4 Support Vector Machine

The Support Vector Machine model defines a hyperplane to differentiate two classes of objects in a multidimensional data space. The hyperplane can be either linear or non-linear depending on the selected kernel. We applied the Support Vector Machine algorithms on the original data space and on the reduced data space.
In both cases, *GridSearchCV* was used to find the best parameters for the implementation of the Support Vector classifier. The scoring on the parameter combinations was done taking into account the recall measured on a validation set of size 5. The hyperparameter tuning was performed for both Linear SVM and Non-Linear SVM, and also for both original data space and reduced data space.

The parameters used are:

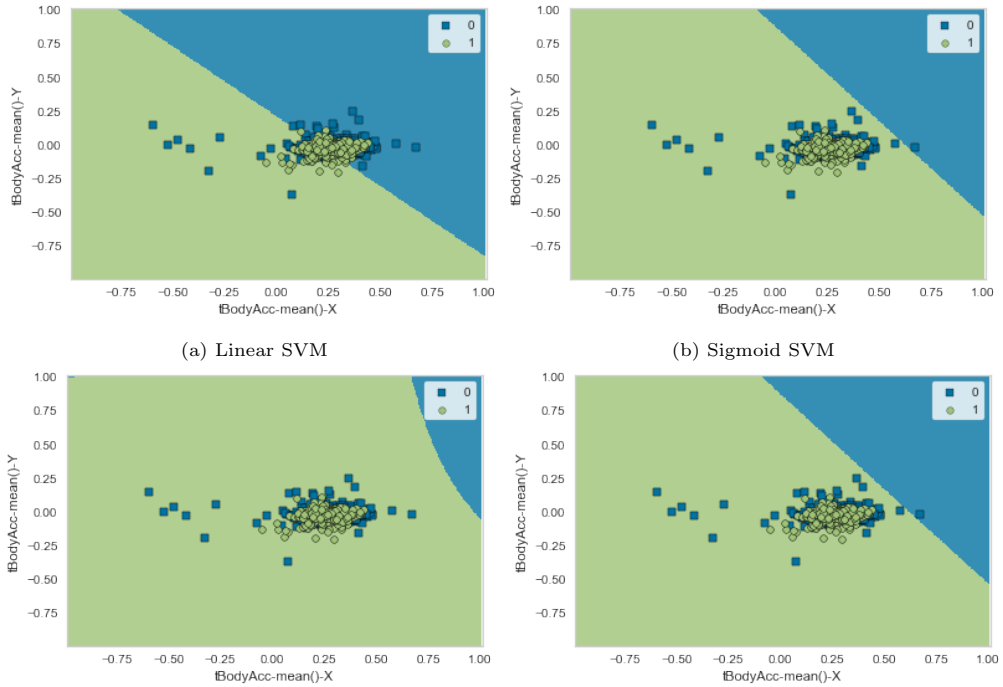
- **C**:[0.001, 0.01, 0.1, 1, 10, 100]. It is a regularization parameter;
- **gamma**: [1,0.1, 0.01, 0.001,'scale','auto']. It is a kernel coefficient used by different kernel function to regularize its shape.

The hyperparameter tuning was applied for each type of kernel functions. The kernel functions examined were: *Linear*, *Sigmoid*, *Polynomial* and *Radial basis function (rbf)*.

As we can see in Tables 7 and 8, all the models performed very well, but it can be noticed that the reduction of dimensionality to 19 components through PCA slightly reduced the performance of most models. This is due to the fact that the use of PCA can lead to a significant loss of information that could have been important for classification. In fact, a slight decrease in the accuracy can be observed.

Figure 13 shows the graphical representations of all SVM methods on the original dataset.

For simplicity, we reported in Figure 14 the Precision-Recall curve for the Linear SVM on the original data space, while Table 9 shows the AUC-ROC and AUC-PR for the same model.
Specifically, to summarise the performance of the model on the test set, we can notice that 98% of the records classified as *Class 0* belonged to this class and 100% of the records that actually belonged to the *Class 0* were classified as belonging to it (recall = 100%). The precision of the model is the same in both the classes, but the recall is lower in the positive class (recall = 91%) which means that the false negatives for the positive class are more than for the negative one and therefore the model is less permissive in classifying instances as positive. Overall, the model on the test set shows an accuracy of 0.98.



(a) Linear SVM

(b) Sigmoid SVM

(c) Poly SVM

(d) rbf SVM

Figure 13: Graphical representation of SVM methods on the original dataset

	Linear SVM	**Non-Linear SVM**		
C	0.1	100	0.001	10
Gamma	1	0.001	0.1	scale
Kernel	linear	sigmoid	poly	rbf
Precision	0.98 - 0.98	0.98 - 0.98	0.99 - 0.96	0.99 - 0.96
Recall	1.00 - 0.91	1.00 - 0.91	0.99 - 0.94	0.99 - 0.93
F1-Score	0.99 - 0.94	0.99 - 0.94	0.99 - 0.95	0.99 - 0.94
Accuracy	0.98	0.98	0.98	0.98

Table 7: Results of Linear and Non-Linear SVM on original dataset

	Linear SVM PCA	**Non-Linear SVM PCA**		
C	0.01	100	100	1
Gamma	scale	1	1	scale
Kernel	linear	sigmoid	poly	rbf
Precision	0.96 - 0.97	0.94 - 0.85	0.98 - 0.82	0.98 - 0.86
Recall	1.00 - 0.78	0.98 - 0.65	0.96 - 0.92	0.97 - 0.89
F1-Score	0.98 - 0.86	0.96 - 0.74	0.97 - 0.87	0.98 - 0.87
Accuracy	0.96	0.92	0.95	0.95

Table 8: Results of Linear and Non-Linear SVM with PCA

	Linear SVM (Original data)
AUC-ROC	0.95
AUC-PR	0.95

Table 9: AUC-ROC & AUC-PR Linear SVM model

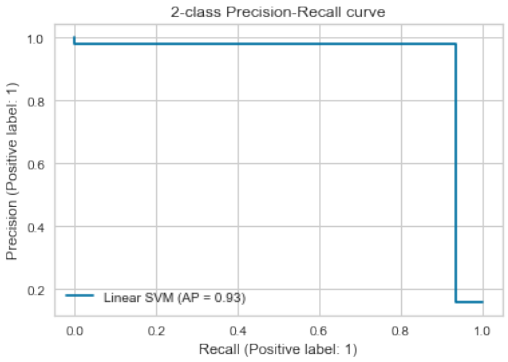


Figure 14: Precision-Recall Curve of Linear SVM

5.5 Neural Network

In this section we solved the classification task with two types of Neural Network models:

- **Perceptron** is a single-layer neural network model that is used when the problem is linearly separable;
- **Deep Neural Network** are multi-layer neural networks and can solve any type of classification tasks involving non linear decision surfaces.

Single Perceptron. It was perfomed via the Perceptron classifier of the *Scikit-Learn* library. We normalized the data and trained the model using the following parameters:

- max_iter = 150 . The maximum number of passes over the training data (aka epochs);
- tol = 0.001 . The stopping criterion. The iteration will stop when *loss >previous_loss - tol* ;
- eta0 = 0.01 . Constant by which the updates are multiplied.

The model returned a high accuracy of 98%, meaning that after performed the training, the model correctly predicts and classifies 98% of the record in the test set.

Deep Neural Network. It was performed via the *Keras library*, in particular the Sequential and Dense models. The first layer was set as Sequential and the other 3 models as Dense with 128, 64 and 1 units respectively. As optimizer, the parameter 'ADAM' was chosen due to the fact that it performs better in the case of large datasets; as loss function the 'binary_crossentropy'.
We chose to set the Epochs equal to 100, in order to have a more accurate evaluation even if it was computationally time-consuming. Finally, the model was evaluated according to the batch size, which is between 10 and 50. The variation of the batch size made no difference in terms of accuracy, while we can observe a decrease of the loss (Table 10).

	Batch 10	Batch 50
Accuracy	0.97	0.97
Loss	0.21	0.14

Table 10: Table of accuracy and loss

Afterwards, we evaluated the model by splitting the data into a training and a validation set: 80% of the instances were used to train the model and the remaining 20% to validate it. In this case, the model has the same accuracy value as before (0.97) while the Loss value has increased significantly.
In order to avoid possible overfitting problems, it was decided to apply the Early Stopping and L2 Regularization approaches.

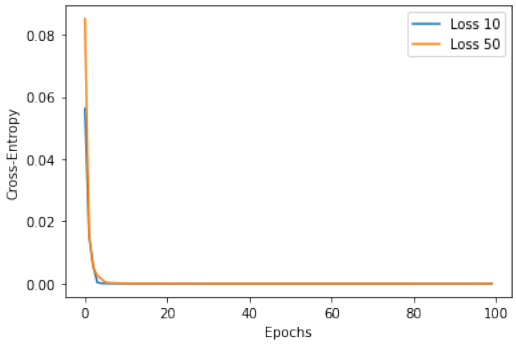


Figure 15: Difference Batch size

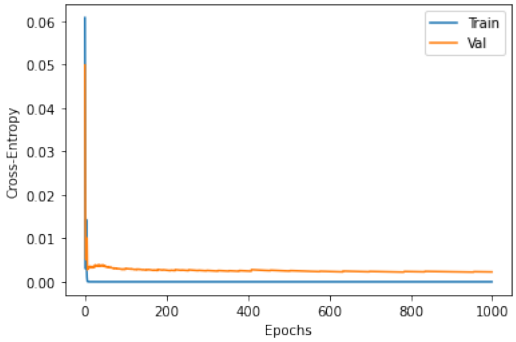


Figure 16: Evaluation of cross-entropy

As Table 11 shows, the best result was achieved using Early Stopping approach.

	Early Stopping	L2 Regularization
Accuracy	0.97	0.84
Loss	0.15	2.46

Table 11: Early Stopping - L2 Regularization

Finally, we evaluated the hyperparameter of the Keras model. The *RandomizedSearchCV* algorithm was used to search for the best parameters; the parameters to be evaluated were:

- n_layer = [1, 2, 3] ;
- h_dim = [32, 64, 128] ;
- activation = ['relu', 'linear','sigmoid','tanh'] ;
- optimizer = ['adagrad', 'adam'] .

The best result obtained was:

- n_layer = 1 ;
- h_dim = 64 ;
- activation = 'relu' ;
- optimizer = 'adam' .

5.6 Ensemble Classifier

Ensemble classifiers construct a set of base classifiers from the training data, and then predict the class label of the test records by combining the predictions made by several classifiers.
Before implementing any Ensemble methods, it has been performed the hyperparameter tuning using the *GridSearchCV* algorithm on the training set in order to find the best estimation for the parameters.

Random Forest. Random Forest constructs an ensemble (forest) of decorrelated decision trees, in Figure 17 are reported the feature importance returned by the random forest.

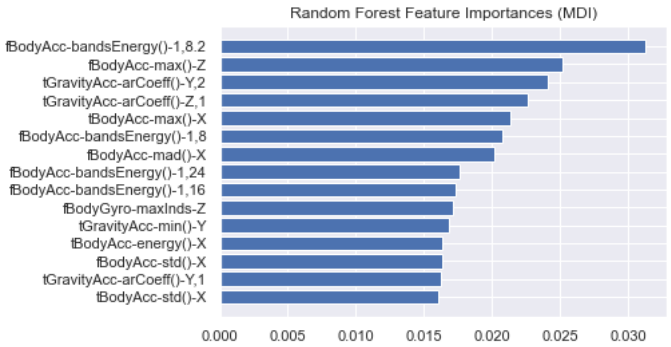


Figure 17: Top 15 most relevant features

Table 12 compares the performance obtained by the Decision Tree we trained in the Imbalanced learning subsection using Random Oversampling and Random Forest on the test set. It is possible to see a increase in performance in the Random Forest model. Specifically, precision and recall as well as F1-score increased as a result of the higher accuracy of the model. Random Forest on the test set have an AUC-ROC of 0.99 and an AUC-PrecisionRecall of 0.96. It can be concluded that the model is able to predict well when a person is walking upstairs or not.

	Decision Tree	**Random Forest**
Precision	0.95 - 0.56	0.95 - 0.96
Recall	0.87 - 0.75	0.95 - 0.75
F1-Score	0.92 - 0.67	0.97 - 0.84
Accuracy	0.86	0.95
AUC-ROC	0.93	0.99
AUC-PR	0.95	0.96

Table 12: Decision Tree - Random Forest

Bagging. It increased the performance of both the Decision Tree and the Support Vector Machine. Table 13 compares the performance of the original models (Linear SVM on the original dataset and Decision Tree) with the Bagging implementations. The performance of the Bagging SVM on the test set was roughly the same as that obtained using the original SVM, while Bagging Decision Tree significantly increased the performance of the base classifier.

AdaBoost is a classifier used in combination with other classification algorithms. The output of these algorithms is combined by a weighted sum to obtain the final output. In our case, it will considered two previously implemented classifiers: the Decision Tree Classifier and the Random Forest Classifier.

Table 14 compares the performance of the classifiers. The AdaBoost implemented using the Decision Tree performs better than using the Random Forest.

	Original SVM	Bagging SVM	Bagging Decision Tree
Precision	0.98 - 0.98	0.99 -0.95	0.95 - 0.87
Recall	1.00 - 0.91	0.99 - 0.95	0.98 - 0.76
F1-Score	0.99 - 0.94	0.99 - 0.95	0.96 - 0.79
Accuracy	0.98	0.98	0.93
AUC-ROC	0.95	0.98	0.98
AUC-PR	0.95	0.97	0.97

Table 13: Original SVM - Bagging SVM - Bagging Decision Tree

	AdaBoost Decision Tree	AdaBoost Random Forest
Precision	0.97 - 0.92	0.96 -0.87
Recall	0.99 - 0.86	0.98 - 0.80
F1-Score	0.98 - 0.89	0.97 - 0.83
Accuracy	0.96	0.94
AUC-ROC	0.98	0.98
AUC-PR	0.95	0.94

Table 14: AdaBoost for Decision Tree and Random Forest

5.7 Gradient Boosting

Gradient Boosting method produces a prediction on model in the form of an ensemble of weak prediction models.
We solved two kinds of tasks:

- **Gradient Boosting for Classification** finds the class prediction that occurs the most frequently;
- **Gradient Boosting for Regression** takes the average of the results obtained by weak learners.

In both cases, *GridSearchCV* algorithm was used to find the best parameters for the implementation of the Gradient Boosting. The scoring on the parameter combinations was done on a validation set of size 5. The parameters obtained from hyperparameter tuning are:

- Gradient Boosting for Classification: n_estimators = 150, max_depth = 5 and learning_rate = 0.1 ;
- Gradient Boosting for Regression: n_estimators = 100, max_depth = 4, learning_rate = 1.0 .

Tables 15 and 16 show the results obtained by the Gradient Boosting models with the corresponding metrics for evaluation.

	GradientBoosting Classification
Precision	0.96 - 0.91
Recall	0.98 - 0.79
F1-Score	0.97 - 0.84
Accuracy	0.95
AUC-ROC	0.99
AUC-PR	0.94

Table 15: Results of Gradient Boosting for Classification

	GradientBoosting Regression
R2	0.261
MSE	0.099
MAE	0.127

Table 16: Results of Gradient Boosting for Regression

Both Gradient Boosting methods performed very well, being reliable classifiers able to distinguish whether a person is walking upstairs or not.

In addition, **XGBoost** and **LightGBM** were implemented, which are alternative methods to Gradient Boosting used to improve performance.

In the case of XGBoost, the parameters were set as follows: objective = 'binary:logistic'; n_estimators = 150; booster = 'gbtree'; max_depth = 4; learning_rate = 1.0; gamma = 0.0 (Minimum loss reduction required to make a further partition on a leaf node of the tree); reg_lambda = 1 (L2 regularization term on weights); tree_method = 'exact' (XGBoost considers all candidates from data for tree splitting, but underlying the objective is still interpreted as a Taylor expansion) .

On the other hand, for LightGBM, the parameters used are: boosting_type = 'gbdt' (traditional Gradient Boosting Decision Tree) ; max_depth = 4 ; num_leaves = 31 ; n_estimators = 100 ; learning_rate = 0.1 ; objective = 'binary' ; reg_alpha = 0.0 (L1 regularization term on weights) ; reg_lambda = 1 (L2 regularization term on weights) .

The results of the models are summarised in Table 17 which shows that both models have very good predictive capabilities, being reliable in distinguishing the two classes.

	XGBoost	**LightGBM**
Precision	0.97 - 0.91	0.98 - 0.93
Recall	0.99 - 0.83	0.99 - 0.87
F1-Score	0.98 - 0.87	0.98 - 0.90
Accuracy	0.96	0.96
AUC-ROC	0.99	0.99
AUC-PR	0.96	0.96

Table 17: Results of XGBoost and LightGBM

5.8 Conclusions on the best classifier

In conclusion, we can state that almost all the analyzed classifiers provide good performance. In particular, the best results were obtained with Support Vector Machine models; both Linear and Non-Linear methods performed very well, although the latter was the best overall, which is due to the fact that SVM Non-Linear methods work comparably well when there is an understandable margin of dissociation between classes, i.e. they are more productive in high dimensional spaces, as was in our case. This result underlines the excellent ability of the model to distinguish when a person is walking upstairs or not.
In addition, good performance was achieved by the Ensemble and Gradient Boosting methods, although they performed worse in terms of Recall for the positive class, resulting in a considerable loss of records with that class.

6 Time Series Analysis

In order to perform time series' tasks we have chosen to use the **total acceleration of the x-axis** to get the widest possible insight into the raw signals. The training set is composed by 7352 univariate time series, each composed by 128 time-stamps. The test set is composed by 2947 of 128 time-stamps. Each time series is labeled with the *activity* during which it was recorded.

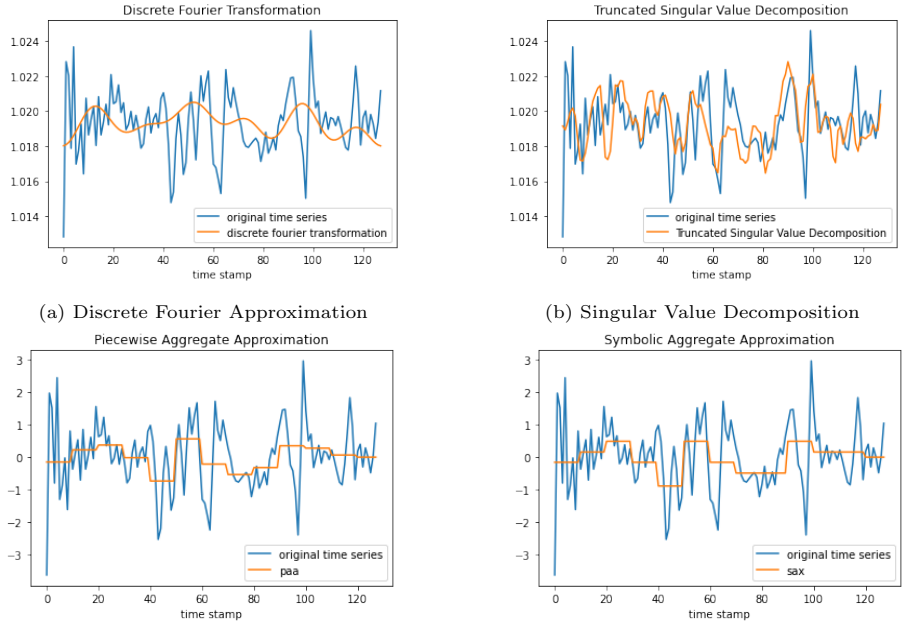
6.1 Clustering

In this section are reported the clustering results on the time series data set. In particular, we've performed **K-means** using both *euclidean* and *dynamic time warping* distance and **hierarchical clustering** computing

time series pairwise distance with *dynamic time warping* distance with a sakoe-chiba band restriction and using the average linkage.

Time series approximation . Before operating the clustering, we have reduced the dimensionality of the time series using **Symbolic Aggregate Approximation**. SAX approximation was applied to the data set dividing time series in 12 segments, and using an alphabet size equal to 8. We choose SAX approximation (Figure 18d) to maintain consistency with other sections of the project, in particular with the *Sequential Pattern Mining* section.

In Figure 18 are shown the representation of the other approximation techniques we had performed. Discrete Fourier Approximation Figure 18a, Piecewise Aggregate Approximation Figure 18c, Singular Value Decomposition Figure 18b.



(a) Discrete Fourier Approximation

(b) Singular Value Decomposition

(c) Piecewise Aggregate Approximation

(d) Symbolic Aggregate Approximation

Figure 18: Time Series Approximation

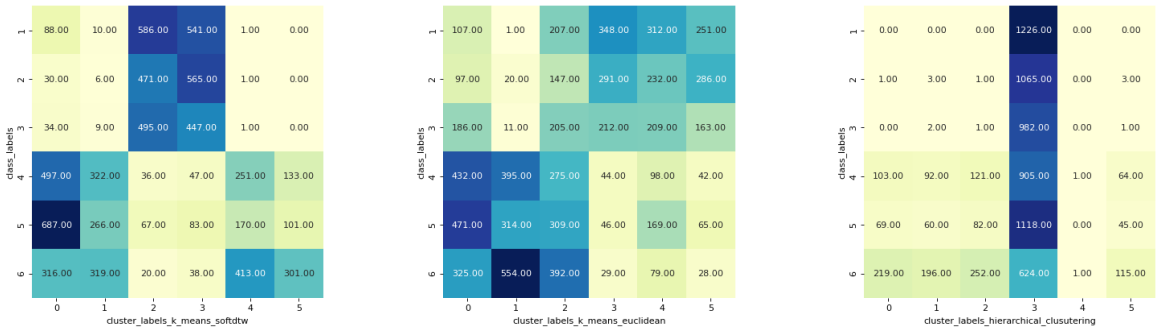
Clustering We exploited the fact that time series were originated by performing six activities, so we fixed parameter k equal to 6 in the k-means and cut the hierarchical clustering to obtain 6 clusters. We have evaluated the results returned by the three clustering algorithms using contingency matrices, that allows us to understand the composition of the clusters with respect to the true class labels of the training set instances.

In Figure 19 are shown contingency matrices produced by the three clustering algorithms performed. Results are not exciting, mainly due to the heavy approximation of time series. Among the three the most satisfactory results are returned by k-means using *dtw distance*. Instances belonging to motion activities, which correspond to class labels 1, 2 and 3, are captured by clusters 2 and 3; instead instances belonging to static activities, 4, 5 and 6, are grouped into cluster 0, 1, 4 and 5.

6.2 Motifs and Anomalies discovery from Matrix Profile

We chose to extract motifs and anomalies from the centroids derived from the time series k-means with *dtw distance*, that are plotted in Figure 20.

In particular we have focused on clusters 2 and 3, which have a very similar compositions according to class labels 1, 2 and 3. In this way we had tried to extract additional information that could be useful to differentiate between the two.



(a) K-means dtw distance (b) K-means euclidean distance (c) Hierarchical clustering

Figure 19: Contingency matrices

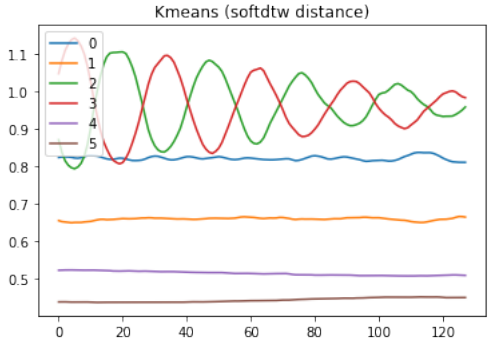
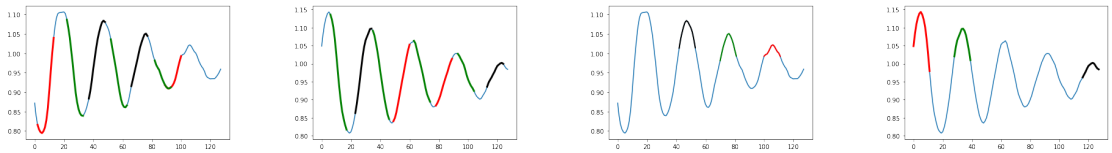


Figure 20: Centroids

We extracted top three motifs and anomalies using matrix profile method with a subsequence's length of twelve timestamps. In Figure 21 are are highlighted on the two time series centroids the top-3 motifs and anomalies retrieved. Although it is difficult to compare the results with each other, motifs and anomalies seem quite similar between the two.



(a) motifs cluster 2 (b) motifs cluster 3 (c) anomalies cluster 2 (d) anomalies cluster 3

Figure 21: Motifs and Anomalies

6.3 Classification

We solved multiclass time series classification task with two types of classifiers:

- **Shapelet-based classifiers** which use as input a new representation of the dataset where each time series is described by a vector containing the distance with respect to the top six shapelets extracted;
- **Structural-based classifiers** which use as input the raw data, where each time-stamp is considered as an attribute of the time series.

6.3.1 Shapelet-based classifiers

We have extracted the top six shapelets (Figure 22), the same number as the class labels, using the Learning Shapelet model proposed by Josif Grabocka. We specified a shapelets length of 12 timestamps.

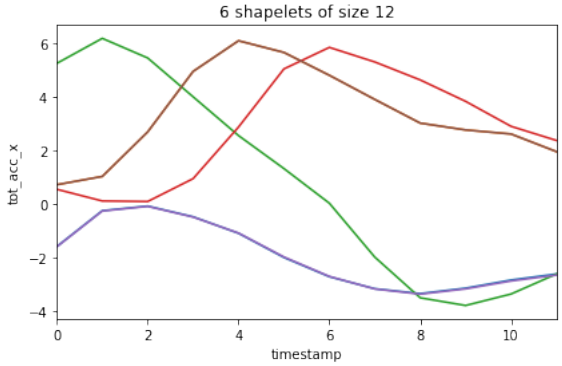


Figure 22: Top six shapelets plot

Two pairs of shapelets have very similar values as we can see in Table 18, this is why there are only four time series subsequences in the graph. This reflects the very similar behavior of time series belonging to different class labels.

Shapelet 1	Shapelet 2	Shapelet 3	Shapelet 4	Shapelet 5	Shapelet 6
-1.584	0.717	5.235	0.546	-1.593	0.717
-0.249	1.024	6.166	0.110	-0.262	1.024
-0.077	2.683	5.435	0.092	-0.088	2.683

Table 18: Shapelets as time series (first three values)

After we have transformed the original test set into distances to the learned shapelets, we trained two classifiers on the new dataset:

1. 5-NN ;
2. Decision Tree Classifiers .

In Table 19 are reported the results in terms of accuracy of the 5-NN and of the decision tree but also of the Learning Time-Series Shapelets model used to predict the test set.

	Shapelet model	5-NN	Decision Tree
accuracy	0.70	0.74	0.75

Table 19: Shapelet-based classifiers accuracy

6.3.2 Structural-based classifiers

In order to speed up the calculus we have performed two preprocessing steps:

- **Step 1**: We have reduced the size of the training set keeping a hundred time series for each of the six class labels;
- **Step 2**: Down sampled each time series and kept 32-time stamps (Figure 23).

Then we have trained three different classifiers:

1. **K-NN** that is the kind of a benchmark for time series calssification for its effectiveness; we trained a 5-NN both with euclidean distance and DTW distance. Each neighbor is weighted by the inverse of their distance;
2. **ROCKET** with 10000 kernels, and with the transformed time series we trained a Ridge classifier;
3. **Canonical Interval Forest** composed by thirty trees estimators .

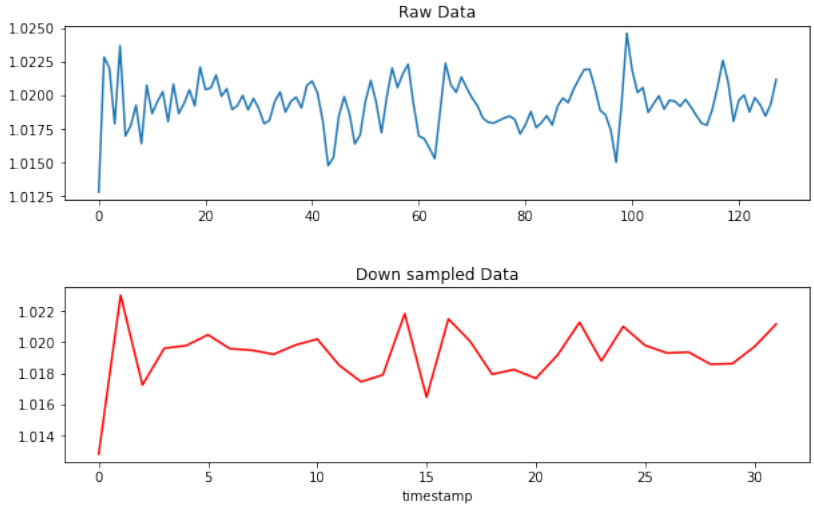


Figure 23: Down sampling

In Table 20 are reported the results in terms of accuracy of the classifiers, as we can see both ROCKET and CIF yield better results than K-NN with DTW distance. We can also notice how using DTW distance lead to better results with respect to euclidean distance in the 5-NN classifier.

	5-NN Euclidean distance	5-NN DTW distance	ROCKET	CIF
accuracy	0.63	0.72	0.80	0.76

Table 20: Structural-based classifiers accuracy

7 Sequential Pattern Mining

The aim of this section is to mine sequential pattern, that are frequent subsequences, from the time series data set, used in the *Time Series Analysis* section, through the *Generalize Sequential Pattern algorithm*. The algorithm only works in a discrete scenario, because of that, before applying GSP, we have done the following preprocessing steps:

1. Scaled time series using the MeanVariance scaler;
2. Approximate time series into 12 segments using SAX with an alphabet length equal to 8.

We have performed GSP with a minimum support threshold of 10%, looking for sequential pattern composed by 4 transactions.
In Table 21 are reported the top 10 sequential patterns mine by the algorithm. We can observe that we have sequential pattern which contains transactions with items 4 and 3 and sequential pattern which contains items 2 and 6. This facts outline once again the differences with time series belonging to static activities with low variability and time series with high variability, belonging to the motion activities group.

Frequency	**Sequential Pattern**
1250	[4, 4, 4, 4]
1239	[2, 2, 2, 2]
1221	[4, 4, 4, 3]
1218	[3, 3, 3, 3]
1209	[4, 4, 3, 3]

Frequency	**Sequential Pattern**
1209	[2, 2, 2, 6]
1209	[2, 6, 2, 2]
1198	[4, 3, 3, 3]
1197	[2, 2, 6, 2]
1189	[6, 2, 2, 2]

Table 21: Top-10 sequential patterns

8 Advanced Clustering

In this section will be analyzed the performance of two advanced clustering algorithms: **X-Means** and **OPTICS**. The clustering was performed on the tabular dataset.

8.1 X-Means

X-Means clustering algorithm is a K-means extansion that automatically determines the number of clusters, using the *Bayesian Information Criterion*.
We have used the implementation of X-Means of the *pyclustering* python library. X-Means found 20 clusters (the default value specified as upper bound of number of clusters) of different sizes. In order to represent the clustering, we have reduced the dataset space to 2 dimensions using the PCA approximation. We have fit the training set and then we have transform the clustering centroids.

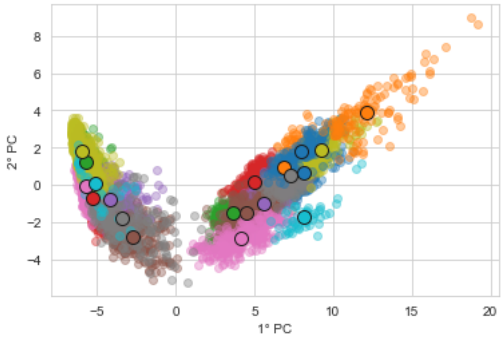


Figure 24: X-means clustering

8.2 OPTICS

OPTICS clustering algorithm is an evolution of the DBSCAN. OPTICS returns the points in a particular ordering, annotated with their smallest reachability distance. This output allow us to plot the reachability-plot (that is a special kind of dendrogram), from which we can obtained the hierarchical structure of the clusters.

We have used the implementation of OPTICS of the *scikit-learn* python library. We have first used OPTICS with its Xi cluster detection method with eps, the maximum distance between two samples for one to be considered as in the neighborhood of the other, setted as *inf* and min samples equal to 5 (The number of samples in a neighborhood for a point to be considered as a core point).
OPTICS return 20 clusters with very few points (59 max number) and 7054 record as outliers.

Then we have exploited the reachability plot in Figure 25 to set specific thresholds (3.5, 3.7 and 4) on the reachability, which corresponds to perform a DBSCAN.

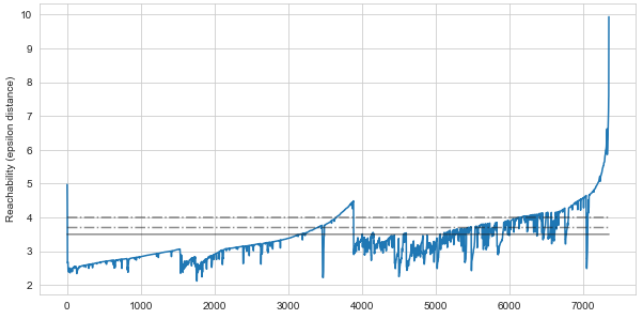
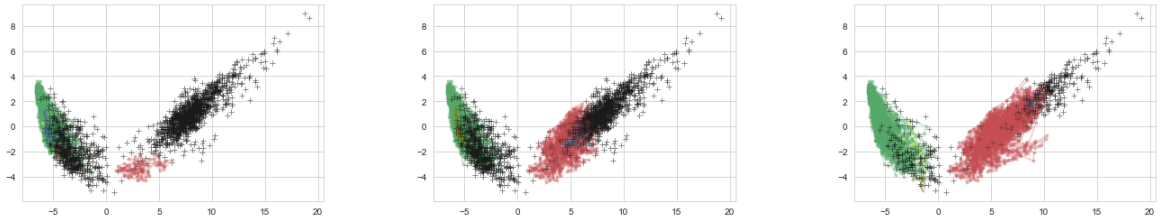


Figure 25: Reachability plot

Also in this case we have reduced the dataset in 2 dimensions using PCA approximation to plot the clustering results with eps parameter with the three previously specified values.
In Figure 26 We can see that the different clustering are retrieved with different choices of thresholds in DBSCAN.



(a) eps = 3.5 (b) eps = 3.7 (c) eps = 4

Figure 26: cluster_optics_dbscan clustering

9 Explainability

In conclusion, we conducted an explainability analysis by using LIME and SHAP methods. Our analysis focused on *Random Forest* which was one of the models with good results, as seen in Advancedd Classification Methods section.
We applied the two methods to selected records which showed interesting results in terms of prediction probability. The two selected records belong to the positive and negative class respectively.

9.1 SHAP

SHAP explains the predictions based on game theory, where the shapley value represents the contribution of a feature to the final outcome of the model.
From a global perspective, the summary plot in Figure 27 shows the positive and negative relationship of the top-10 features, in terms of importance and impact on the model, with the target variable. We note that some features, such as *tGravityAcc-min()*, *tGravityAcc-max()*, *tGravityAcc-std()* and *fBodyAcc-bandsEnergy()-1.8*, also appear in the plot of the feature importance of Random Forest in *Advacned Classification Methods* section, which means that they have a significant impact on the classifier's decisions.

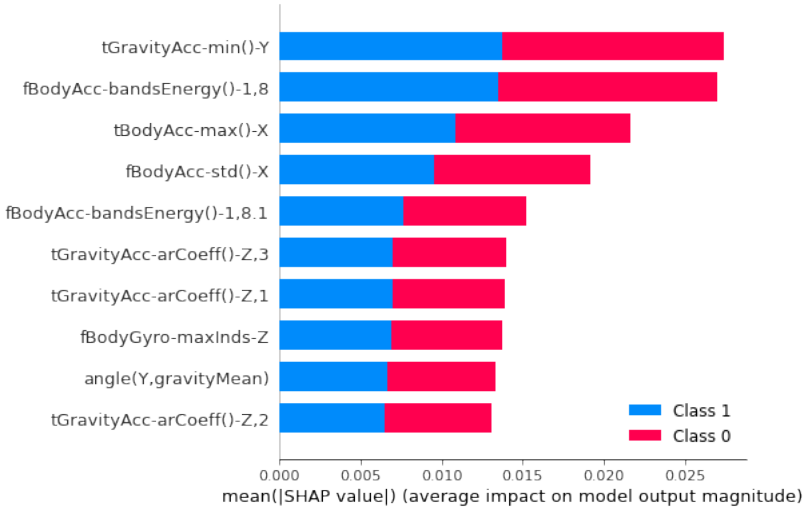


Figure 27: Summary plot shap feature value

On the other hand, considering a local point of view, two test samples were selected, one belonging to the positive class and the other to the negative class.

Figure 28 shows the force plot for the test record with negative class. The force plot is a special plot displaying the shapley values for features. Feature values in red increase the prediction, while those in blue decrease it. It can be seen that feature such as *fBodyAcc-max()-X* have a positive effect on prediction, in fact its value is, as shown in the figure, -0.225 which is larger than its average value -0.647, so it pushes the prediction to the right. On the other side, features such as *tGravityAcc-min()-Y* have a negative effect, in fact its value is -0.099 which is smaller than its average value 0.023, so it pushes the prediction to the left. In summary, the output value is 0.13 which represents the prediction for that observation, while the base value is 0.15, thus resulting in the negative class prediction.

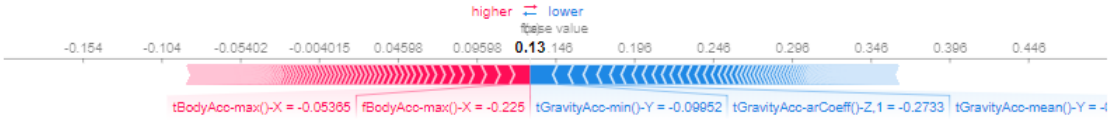


Figure 28: Force plot Class 0 instance

For the record test with a positive class, the process is specular (Figure 29). We note a positive effect on prediction by feature such as *tGravityAcc-arCoeff()-Y,1*. The output value is 0.68 while the base value is 0.1469, thus the prediction results as positive class.

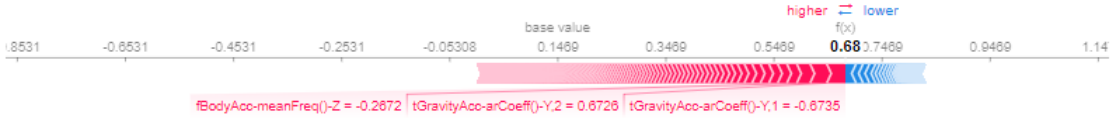


Figure 29: Force plot Class 1 instance

9.2 LIME

The output of **LIME** is a list of explanations, reflecting the contribution of each feature to the prediction of a data sample; it also allows to determine which feature changes will have most impact on the prediction. As already done for SHAP, the same test records with both negative and positive class were selected.

With regard to the negative class, it can be seen that none of the features heavily influenced the decision towards *Class 0*, indeed all features displayed in the plot in Figure 30 had the same influence on the final result, which results as negative class with a probability of 91%.

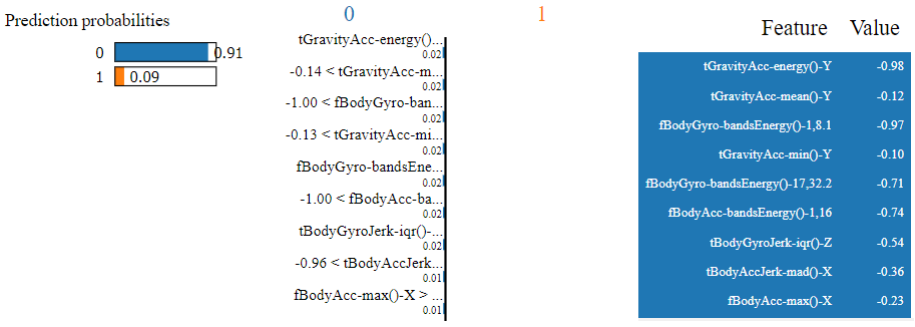


Figure 30: LIME Class = 0

The same can be said for the positive class, none of the feature heavily influences the classifier's decision, in fact the features considered in figure 31 are equally important. The prediction results as a positive class with a probability of 68%.

Figure 31: LIME Class = 1