Università di Pisa

Data Mining II Final Project

Ondrej Krasnansky
Alberto Bessone
Luca Santolillo

Human Activity Recognition Using Smartphones

Academic Year 2022/2023

1 Introduction

In this report we are going to analyze the data obtained from the experiment: Human Activity Recognition Using Smartphones Dataset. This experiment was performed by analyzing a sample of 30 volunteers aged 18-48 years, who performed 6 activities: WALKING, WALKING UPSTAIRS, WALKING DOWNSTAIRS, SITTING, STANDING, LAYING. For each task, data of 3-axial linear acceleration and 3-axial angular velocity at a constant rate of 50Hz were extracted. In addition, to obtain two sets (one train set and one test set), 70% of the participants were randomly selected to form the train set and 30% to form the test set. The original Dataset therefore consists of a train set of 7352 rows by 561 columns, and a test set of 2947 rows by 561 columns.

1.1 Data Preparation and Preprocessing

For this first part of the project, we sought to understand how the dataset to be analyzed was constituted, and then later process it to make it clearer and more compliant with the tasks we would have to peruse on it. From the documentation in the folder from which we extracted the dataset, it was described how in part there had already been data cleaning work, in how many features had been extracted from time series values, which had been normalized and transformed to remove noise. the dataset is thus presented with features that had already been normalized and were between -1 and 1 in value. for first objective we tried to highlight if there were any unique values to be eliminated, however, no values showed up from our search. Next, an algorithm was implemented to eliminate features that had more than 0.85 or less than -0.85 as a correlation value, this was done to avoid redundancies in the data. finally, it was checked which features had the least variance, but putting them in order it was found that, among those remaining after the previous elimination processes, there were none with enough variance to allow further elimination of that variable. The final dataset is presented with 152 features, to which an additional 3 columns were added that represented the target variable and the subject who had performed the action. More specifically, the columns: activityName, activity, subject were added. The total size of the dataset is thus [7352 rows $\times$ 155 columns] as far as the test dataset is concerned, and [2947 rows $\times$ 155 columns] as far as the test is concerned.

2 Imbalanced Learning & Dimensionality Reduction

2.1 Data preprocessing

For this part, it was decided to perform the imbalanced learning and dimensionality reduction tasks only on the "*Train*" file. In this phase, the "Train" dataset preprocessed in the previous step was used. The features "*ActivityName*" and "*Subject*" were removed as they were unnecessary for this experiment. For the classification, '*Activity*' was considered as the *target variable* (on which the label encoding had already been performed). The distribution of the classes appeared as in the figure 1.

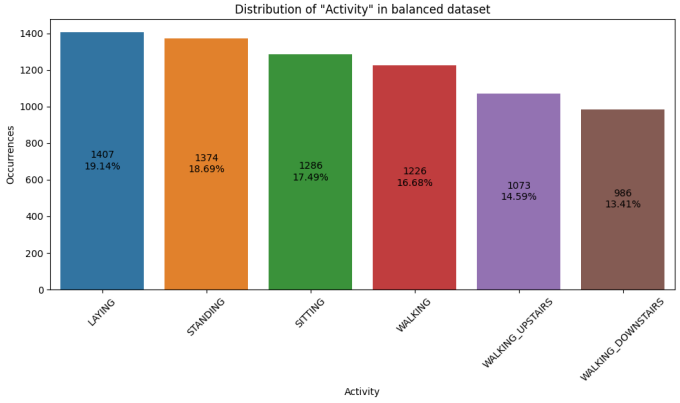
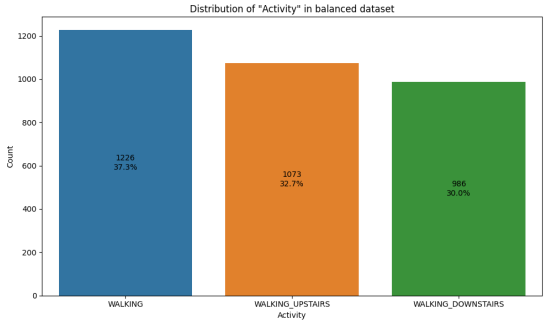
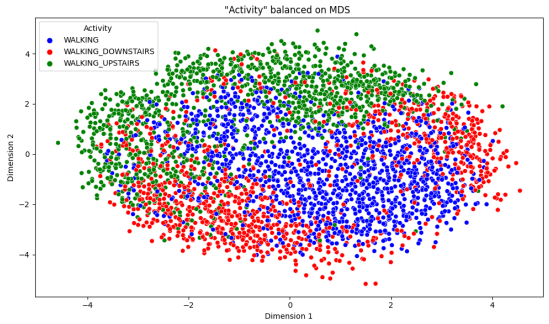


Figure 1: "Activity" in balanced dataset.

As the classes were fairly balanced and distinct, it was decided to eliminate the least overlapping classes ("SITTING" "STANDING" "LAYING"), creating a new dataset containing only the most overlapping classes ("WALKING", "WALKING_UPSTAIRS", "WALKING_DOWNSTAIRS"), obtaining the results visible in the figures below.



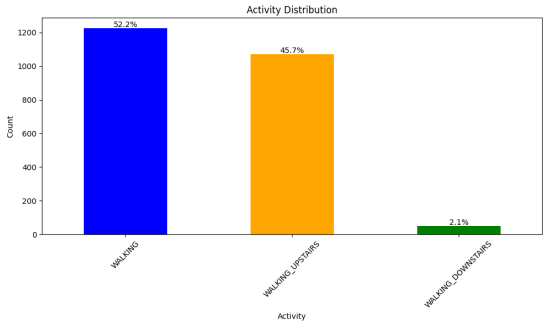
(a) Distribution of "Activity" with 3 class on balanced dataset.



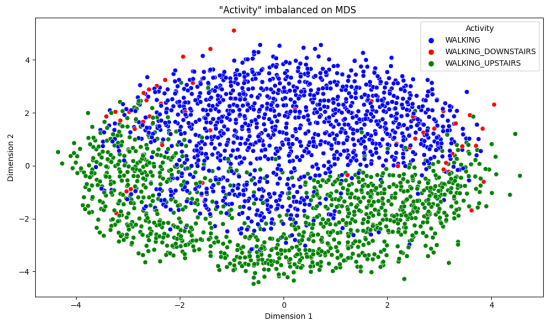
(b) "Activity" with 3 balanced class on MDS.

Figure 2: "Activity" reduced on balanced dataset.

Finally, the dataset was unbalanced by reducing the least present class ('WALKING_DOWNSTAIRS'), bringing its occurrences to 2.1% of the total classes.



(a) Distribution of "Activity" with 3 class on imbalanced dataset.



(b) "Activity" with 3 imbalanced class on MDS.

Figure 3: "Activity" reduced on imbalanced dataset.

Once the unbalanced dataset was obtained, an initial classification was performed on it with the *Decision Tree*, using all features (152) to determine the values of the categorical variable *'Activity'*. The optimal values for the DT parameters were found with a GridSearch with RepeatedStratifiedKFold. The values for the parameter in the GridSearch are as follows:

GRIDSEARCH DT WITH CV=5	Parameter
min_samples_split	2, 5, 10, 15, 20
min_samples_leaf	1, 5, 10, 15, 20
max_depth	None, 2, 5, 10, 15, 20
criterion	"gini", "entropy", "log_loss"

Table 1: Parameters for Decision Tree GridSearch

The results obtained with the best possible DT parameters ('criterion': 'entropy', 'max_depth': 20, 'min_samples_leaf': 1, 'min_samples_split': 2, 'random_state=42') on the entire dataset (152 features) are shown in the *classification report* and *confusion matrix*.

Classe	Precision	Recall	F1-Score	Support
WALKING	0.97	0.95	0.96	368
WALKING_UPSTAIRS	0.96	0.96	0.96	322
WALKING_DOWNSTAIRS	0.43	0.67	0.53	15
Accuracy			0.95	705
Macro Avg	0.79	0.86	0.81	705
Weighted Avg	0.95	0.95	0.95	705

Table 2: Classification report on 152 dimensions.

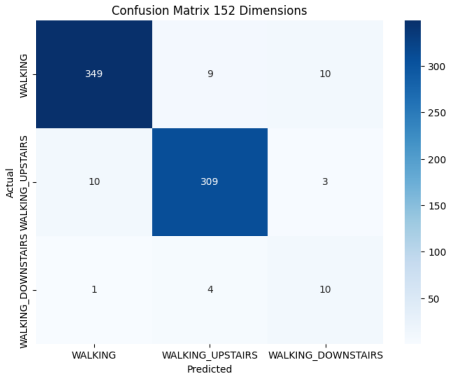


Figure 4: Confusion Matrix DT 152 Dimensions

2.2 Select From Model

As a first dimensionality reduction technique, it was decided to proceed with the *SelectFromModel* using the DecisionTree as estimator on the 152-dimensional dataset. The SelectFromModel selected 14 dimensions. Before proceeding with *oversampling* and *undersampling* techniques, it was decided to perform a GridSearch with the same previous specifications to find the optimal values of the DT parameters (criterion='entropy', max_depth=20, min_samples_leaf=1, min_samples_split=2, random_state=42) on the dataset reduced to 14 features. The results are shown in the *classification report* and *confusion matrix* below.

Class	Precision	Recall	F1-Score	Support
WALKING	0.97	0.98	0.97	368
WALKING_UPSTAIRS	0.97	0.96	0.96	322
WALKING_DOWNSTAIRS	0.83	0.67	0.74	15
Accuracy			0.96	705
Macro Avg	0.92	0.87	0.89	705
Weighted Avg	0.96	0.96	0.96	705

Table 3: Classification Report on 14 Dimensions.

Comparing the DT run on 152 dimensions and the one run on 14, we can see that the latter has almost double the precision of the initial dataset on the unbalanced class (from 43% to 83%) while recall is unchanged (67% on both datasets).

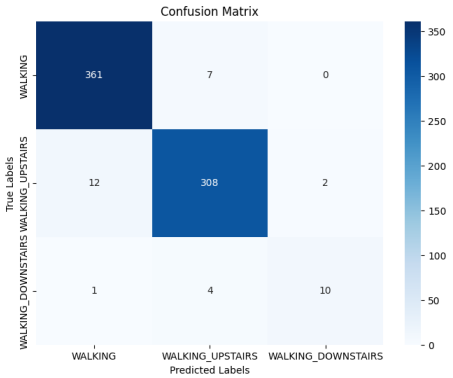


Figure 5: Confusion Matrix DT 14 Dimensions

2.2.1 Oversampling

Next, starting from the dataset reduced to 14 dimensions with SelectFromModel, oversampling techniques were applied, specifically *SMOTE* and *ADASYN*. Then, 2 GridSearches were launched on the oversampled data (one on the *oversampled train with SMOTE* and the other on the *oversampled train with ADASYN*), in order to find the best values for the DT parameters. Once the best parameters were obtained, predictions were made with the trained models on the *Test* part obtaining the results in the table 4.

Select From Model	**Precision**	**Recall**	**F1-Score**	**F1-Score Macro Avg**
SMOTE	64%	93%	76%	89%
ADASYN	54%	87%	67%	86%

Table 4: Oversampling comparison with Select From Model[1]

As shown in the table 4, on the reduced dataset with 14 dimensions with SelectFromModel, the oversampling technique that performs best is *SMOTE*. The model trained with SMOTE has a better F1 Score on the minority class "WALKING_DOWNSTAIRS" (76% vs. 67% of *ADASYN*), which leads to an improved overall performance (*F1-Score Macro Avg* 89% Vs 86%). When comparing these results with the DT on 152 dimensions (*F1-Score unbalanced class*: 53%; *F1-Score Macro Avg*: 81%), it can be seen that both techniques lead to better results on both the unbalanced class and overall performance. Regarding the comparison with DT on 14 dimensions without oversampling, it can be seen that SMOTE manages to obtain better *F1-Score* results on the unbalanced class (76% vs 74%).

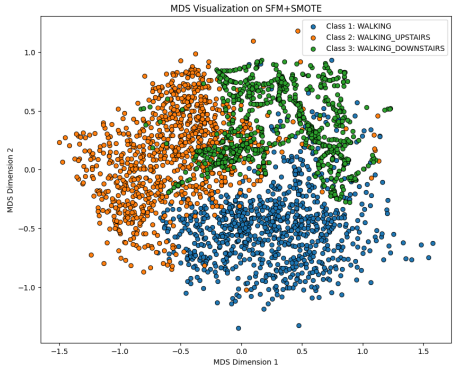


Figure 6: MDS with SFM+SMOTE Oversampling Class WALKING_DOWNSTAIRS

In the following, it was deemed important to include the ROC curves of the Select From

[1]Note: *Precision*, *Recall* and *F1-Score* refer to the performance metrics for the WALKING_DOWNSTAIRS class, while *F1-Score Macro Avg* is a general metric.

model in combination with oversampling for the sake of completeness. However, it is important to emphasise that on an imbalanced dataset, the use of these metrics as a performance parameter may lead to misleading results.

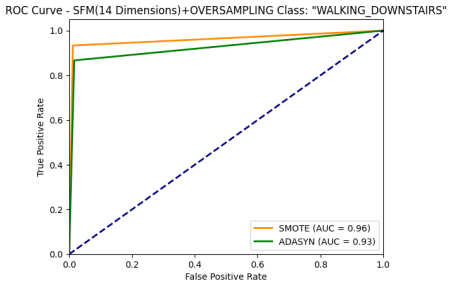


Figure 7: ROC Curves SFM+Oversampling comparison

2.2.2 Undersampling

As regards the undersampling techniques, the same steps were performed as for oversampling. Condensed Nearest Neighbors(CNN), TOMEK LINKS, Edited Nearest Neighbors (ENN) and Cluster Centroids (CC) were used on the reduced dataset with 14-dimensional SelectFromModel. Then, for each Train on which an undersampling technique was applied, a GridSearch was performed (with the same values as those used in the data preprocessing) to find the best values for the DT parameters. The results obtained are shown below:

Select From Model	Precision	Recall	F1-Score	F1-Score Macro Avg
CNN	25%	93%	39%	72%
TOMEK LINKS	83%	67%	74%	89%
ENN	92%	73%	81%	91%
CC	16%	93%	27%	65%

It was considered to evaluate the models according to the *F1-Score* because it was considered to be the ideal metric summarising precision and recall. In this case, note how *ENN* is the *undersampling technique* that returns the best results (F1-Score on the underrepresented class: 81%, F1-Score Macro Avg: 91%) while the worst result is from *CC* (F1-Score on the underrepresented class: 16%, F1-Score Macro Avg: 65%).

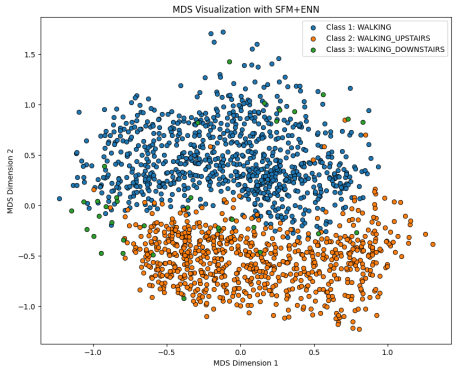


Figure 8: MDS with SFM+ENN Undersampling Class WALKING_DOWNSTAIRS

In the following, it was deemed important to include the ROC curves of the Select From model in combination with undersampling for the sake of completeness. However, it is important to emphasise that on an imbalanced dataset, the use of these metrics as a performance parameter may lead to misleading results.

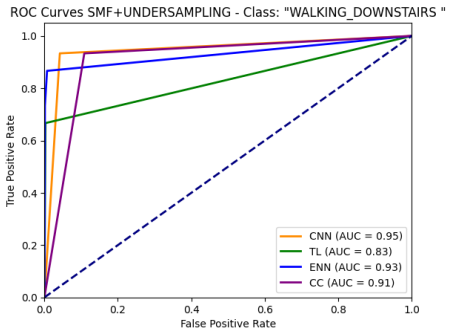


Figure 9: ROC Curves SFM+Undersampling comparison

2.3 Recursive Feature Elimination

In this second experiment, *Recursive Feature Elimination*(RFE) was used to reduce the preprocessed dataset with 152 dimensions. Before running the RFE, a GridSearch was performed with RFE and Decision Tree as estimator, and using the F1-Score as scoring. The RFE has selected 10 dimensions. Next, a GridSearch was performed on the reduced dataset to find the best values for the DT parameters.

Class	**Precision**	**Recall**	**F1-Score**	**Support**
WALKING	0.95	0.98	0.97	368
WALKING_UPSTAIRS	0.98	0.95	0.96	322
WALKING_DOWNSTAIRS	0.75	0.60	0.67	15
Accuracy			0.96	705
Macro Avg	0.89	0.84	0.87	705
Weighted Avg	0.96	0.96	0.96	705

Table 5: Classification Report on 10 Dimensions.

It can be observed how the F1 Score of the unbalanced classes rises from 53% of the dataset with 152 dimensions to 67% of the dataset reduced to 10 dimensions with RFE.

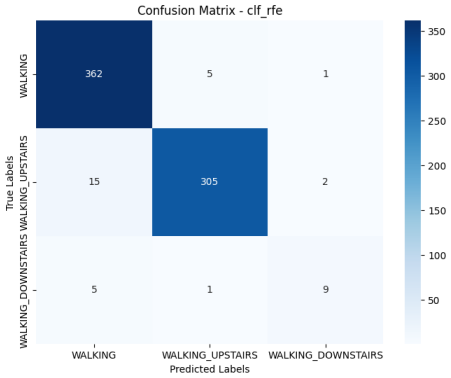


Figure 10: Confusion Matrix DT on RFE (10 Dimensions)

2.3.1 Oversampling

During this step, the dataset with RFE was used and two oversampling techniques were performed on it: *SMOTE* and *ADASYN*. After that, GridSearch was performed on each oversampled dataset to find the best values for the DT parameters and the results visible in the table below were obtained:

As can be seen from the table 6, the oversampling technique that returns the best results on the reduced 10-dimensional dataset with RFE is *ADASYN* (F1-score: 71%, F1-Score Macro Avg:

Recursive Feature Elimination	Precision	Recall	F1-Score	F1-Score Macro Avg
SMOTE	59%	67%	62%	84%
ADASYN	69%	73%	71%	87%

Table 6: Comparison of oversampling techniques using Recursive Feature Elimination

87%), which also performs better on the unbalanced class with respect to the classifier used with RFE without any oversampling technique (F1-score: 67%).

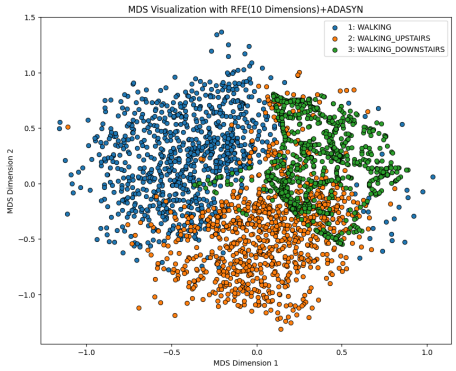


Figure 11: MDS with RFE+ADASYN Oversampling Class WALKING_DOWNSTAIRS

In the following, it was deemed important to include the ROC curves of the RFE model in combination with oversampling for the sake of completeness. However, it is important to emphasise that on an imbalanced dataset, the use of these metrics as a performance parameter may lead to misleading result.

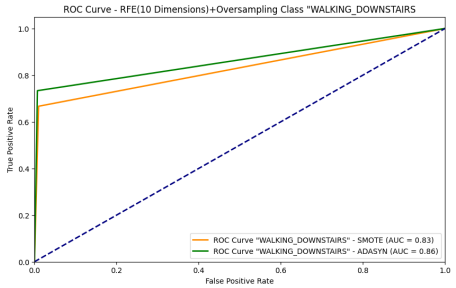


Figure 12: ROC Curves RFE+Oversampling comparison

2.3.2 Undersampling

For this part, the reduced RFE dataset with 10 dimensions was taken and then undersampled with *CNN*, *TOMEK LINKS*, *ENN* and *CC*. A GridSearch was then performed with each undersampled dataset to find the best values for the paremeters of the DT. The results obtained can be seen in the following table:

Recursive Feature Elimination	Precision	Recall	F1-Score	F1-Score Macro Avg
CNN	9%	60%	16%	58%
TOMEK LINKS	75%	60%	67%	87%
ENN	78%	47%	58%	83%
CC	20%	87%	32%	70%

As can be seen in the table, the best results are obtained with TOMEK LINKS (F1-Score:67%, F1-Score Macro Avg 87%) and ENN(F1-Score:58%, F1-Score Macro Avg: 83%). In this case, the

DT on the RFE with ENN has the same performance as the DT on the RFE (F1-Score:67%, F1-Score Macro Avg 87%).

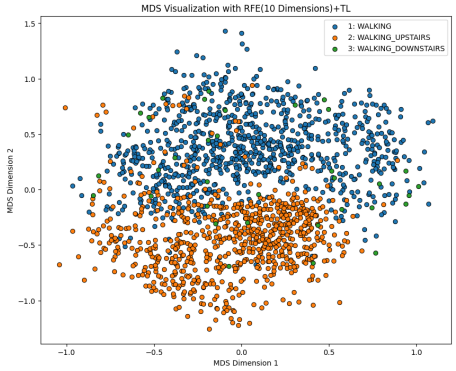


Figure 13: MDS with RFE+TL Undersampling Class WALKING_DOWNSTAIRS

In the following, it was deemed important to include the ROC curves of the RFE model in combination with undersampling for the sake of completeness. However, it is important to emphasise that on an imbalanced dataset, the use of these metrics as a performance parameter may lead to misleading result.

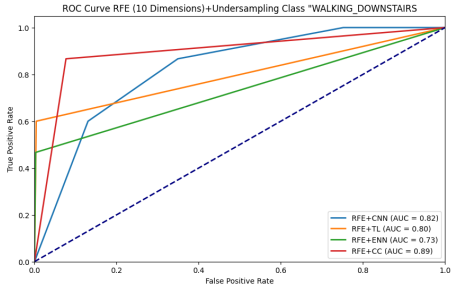


Figure 14: ROC Curves RFE+Undersampling comparison

2.4 Final considerations Imbalanced Learning & Dimensionality Reduction

It can be stated that, with regard to dimensionality reduction with *Select From Model* (14 dimensions), the highest F1 score on the minority class is obtained with *ENN* (81%). In this case, a better overall performance of the model on all classes is also confirmed (F1-Score Macro Avg: 91%). Regarding dimensionality reduction with *Recursive Feature Elimination* (10 dimensions), the highest F1 score on the minority class is obtained with *ADASYN* (71%). In general, it can be said that dimensionality reduction techniques in combination with oversampling and undersampling techniques (except for *CNN* and *CC*) perform better than the dataset reduced to 152 dimensions.

3 Outliers Detection

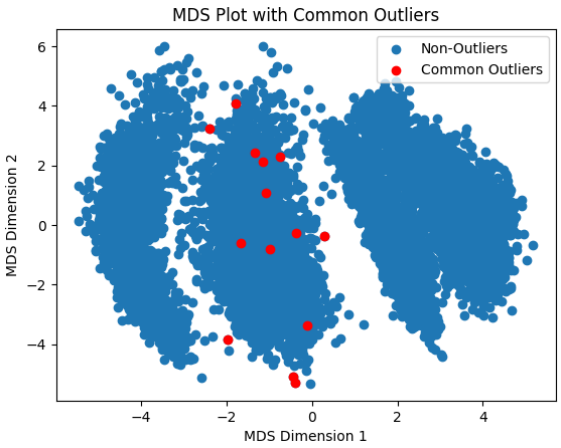
3.1 Detecting Outliers with different Techniques

In this initial phase of *Outlier Detection*, several methods and approaches were used:

- Density-based (COF)
- Angle-based (ABOD)
- Model Based (Isolation Forest)

Methods	N_Outliers
ABOD_5N	185
ABOD_10N	77
COF_5N	74
COF_10N	74
ISOLATION FOREST	74

(a) Methods for Outliers Detection



(b) Outliers (n_neighbors=10) common to all methods on MDS

All *Outlier Detection* methods were set with *contamination* equal to 1%, while for *COF* and *ABOD*, 2 values of N_neighbors were tested: 5 and 10. Next, 2 experiments were performed to find the common outliers: initially *COF* and *ABOD* were set *with n_neighbors=5* and Isolation Forest, and 14 common outliers were found; then *ABOD* and *COF* were set with *n_neighbors=10* and *Isolation Forest*, and again 14 outliers were found.

3.2 ABOD: a Qualiquantitative Analysis

In this experiment, it was considered interesting to analyse the results obtained with *ABOD*. This technique was chosen specifically due to the high number of dimensions (152). ABOD by measuring the variance of the *angular spectrum* gives more accurate results than the *distance of the points*. ABOD with 10 neighbors and 1% of contamination found 77 outliers. The results obtained have the following characteristics:

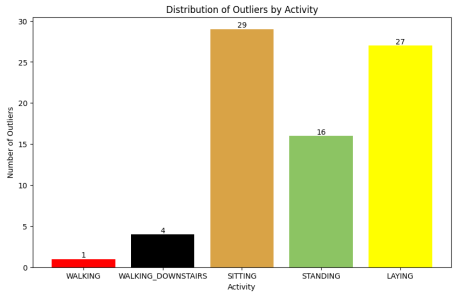


Figure 16: Outliers qualiquantitative analysis with ABOD

As can be seen in the figure 16, the dominant classes among the outliers are "SITTING" (29 outliers) and "LAYING" (27 outliers) followed by "STANDING" (16 outliers), while "WALKING_DOWNSTAIRS" (4 outliers) and "WALKING" (1 outlier) rank in the bottom 1% of outliers. It is important to note that in the top 1% of outliers identified with ABOD, there is no data with class "WALKING_UPSTAIRS ".

Finally, we decided to visualize a reduced 2-dimensional dataset with MDS, first the outliers with respect to the data (Fig.17), and then the outliers labelled according to '*activity*' with respect to the rest of the data(Fig.18).

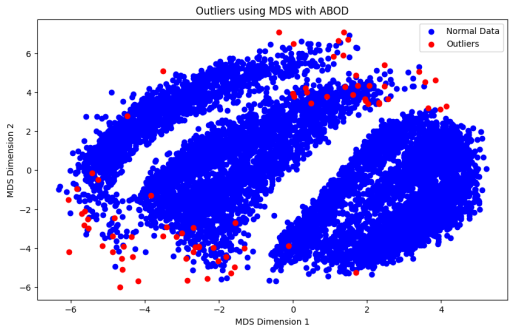


Figure 17: Outliers with respect to data

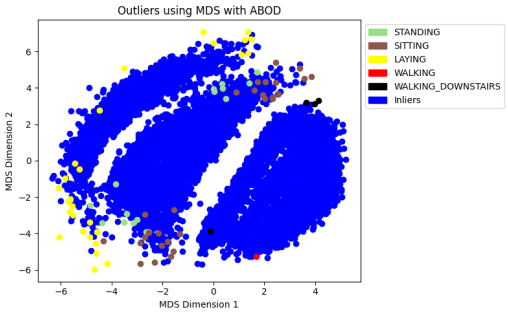


Figure 18: Outliers labeled as "Activity" with respect to data

3.3 ABOD & SFM+ENN on Imbalanced Dataset

For this experiment, the unbalanced dataset from *section 2* was taken, on which the top 2% of outliers were removed, reduced in dimensionality with *SelectFromModel*, and to which *ENN* was applied as an undersampling technique. The aim was to test whether the techniques learned so far and used in combination could improve the DT classifier. *ABOD* was performed with n_neighbors=10 and contamination 1% on the unbalanced dataset in section 2. This eliminated 48 outliers. Subsequently, the clean dataset was reduced with *SelectFromModel* with *DT* as estimator, and 10 dimensions were selected. Finally, *ENN* was performed and, on the imbalanced dataset, a *GridSearch* was carried out to find the optimal values for the *DT classifier parameters*. Finally, classification was performed on the *test* part.

SFM+ W/OUT 2% OUT+ENN	
Dimensions:	10
F1-Score"Walking_Downstairs":	62%
F1-Score Macro Avg:	85%

Table 7: Performance Evauation ABOD+SFM+ENN on Imbalanced Dataset

As can be observed from the results in the table 7, in this case the DT classifier on the imbalanced dataset has a lower performance (F1-Score imbalanced class: 62%, F1-Score Macro Avg: 85%) than the DT on the *Select From Model* with the *ENN* alone (F1-Score on the imbalanced class: 81%, F1-Score Macro Avg: 91%). Most likely, using *ABOD* compromises the DT as it detects and eliminates data from the unbalanced class (which goes from having 50 occurrences to 27 occurrences), resulting in the first class in terms of number of *outliers found*. It can be stated that, in this case, the elimination of the *2% outliers* in combination with *Select From Model* and *ENN* worsens performance compared to classification alone on a reduced dataset with *SFM* and *ENN*.

4 Advanced Classification

In this section, we are going to present various advanced classifiers which were seen during the semester. As an target variable was used *Activity* as the scope of this dataset is to predict this variable. To preprocess the data was not necessary to do any further modifications as the dataset was almost fully prepared to work with. We have just dropped *subject* as we have considered this feature not important during the training.

4.1 Logistics Regression

The first classifier used during this project was Logistics Regression. To train this classifier we have used *LogisticsRegression()* from scikit-learn library. As this classifier can be more sensitive to outliers we have decided to train two different models. First model trained on train test and

tested on test set without any specific modification. The second one was using *ABOD* algorithm to drop 1% of outliers from the train test. For both models it was used 5-fold cross-validation *RandomizedSearchCV* for hyperparameter tuning.

Parameter	**Values**	**Description**
C	0.1, 0.01, 1.0, 10.0	Inverse of regularization strength
solver	newton-cg, lbfgs, liblinear, sag, saga	Algorithm for optimization
penalty	l1, l2	Regularization type
max_iter	1000, 1500	Maximum number of iterations
tol	1e-4, 1e-5, 1e-6, 1e-3, 1e-2	Tolerance for convergence
class_weight	balanced, None	Weight associated with classes

Table 8: Hyperparameters for RandomizedSearchCV

C	**max_iter**	**penalty**	**solver**	**tol**	**class_weight**
10	1000	l2	lbfgs	0.01	None

Table 9: Best parameters without dropping outliers

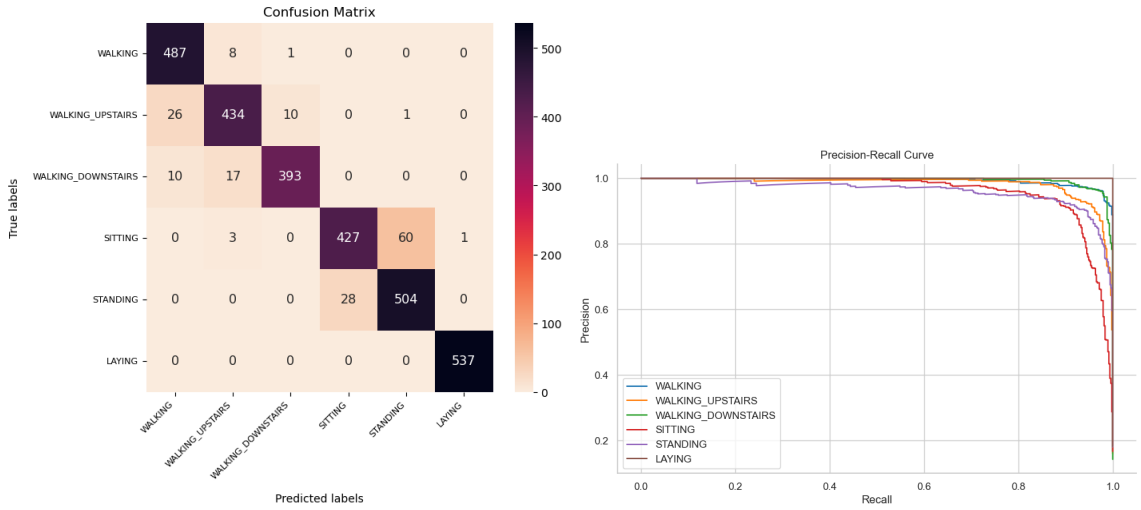
After computing *ABOD (Angle-Based Outlier Detection)* algorithm which have found 185 outliers. These ones were dropped from the train set and it was repeated the RandomizedSearchCV in order to find the best match of hyperparameters. The result of this tuning it can be seen in 10

C	**max_iter**	**penalty**	**solver**	**tol**	**class_weight**
10	1500	l1	liblinear	0.0001	balanced

Table 10: Best parameters dropping outliers

Performances of these two models were quite similar, but dropping outliers was not helpful to reach better results which can be seen in the table 11.

In the table 12 is presented complete classification report for the best model tested. Overall performance of this classifier can be considered satisfying with high precision and recall for every class. The *LAYING* is the class predicted in all its cases with recall 100%. In the figure 19 can be observed more clearly this behaviour. The precision of this class decreased because of the missclassification of *SITTING* as *LAYING* in one case.



(a) Confusion Matrix

(b) Precision-Recall curve

Figure 19: Best Logistics Regression model characteristics

Model	Accuracy	F_score
With outliers	94.40	94.35
Outliers dropped	94.00	93.96

Table 11: Model performances with and without outliers

Table 12: Classification Report for better Logistic Regression model

Class	Precision	Recall	F1-Score	Support
WALKING	0.931	0.982	0.956	496
WALKING_UPSTAIRS	0.939	0.921	0.930	471
WALKING_DOWNSTAIRS	0.973	0.936	0.954	420
SITTING	0.938	0.870	0.903	491
STANDING	0.892	0.947	0.919	532
LAYING	0.998	1.000	0.999	537
Accuracy			0.944	2947
Macro Avg	0.945	0.943	0.943	2947
Weighted Avg	0.945	0.944	0.944	2947

4.2 Support Vector Machine

For this classifier we have tried to use the same approach as in the previous section. As support vector machines are more robust for outliers we have decided to train the model without using any outlier detection algorithm. In order to prevent overfitting and find the best match of hyperparameters we have used 5-fold cross-validation using GridSearchCV from sklearn library. Firstly, the model was trained on the full dimension dataset. Then we have tried to reduce these dimensions to 40 using RFE (Recursive Feature Elimination). Also it was tested to implement oversampling and undersampling techniques and observe if it could be potentially useful in order to improve performances. In the table 13 we present parameters used for the grid search and it's best values selected.

Parameter	Values	Best Parameter
C	1, 0.1, 0.01, 0.001	1
gamma	auto, scale	scale
kernel	linear, rbf, poly, sigmoid	poly
class_weight	None, balanced	balanced
tol	1e-4, 1e-5, 1e-6, 1e-3, 1e-2	0.001
degree	0,1,2,3,4,5,6,7	4

Table 13: Grid Search Parameters with Best Parameters without any dimensional reduction

Kernel parameters permitted us to find out the best function which permits to fit the data in non-linear space. AS he best kernel was *poly*, we have computed another grid search in order to find the best *degree* of the hyperplane for the polynomial function which resulted in to be *4*. The *C* parameter is to define the rate of penalty of missclassifying.

As techniques of undersampling we have used *TomekLinks* and for oversampling *SMOTE*. Firstly, we have computed the basic grid search on the full dimension dataset and tried all the performances using mentioned techniques. After the dataset was reduced by RFE it was repeated the 5-fold cross-validation grid search in order to find new parameters in this dimension and tested the model on the test set.

It can be observed that the optimum *kernel* function changed to *rbf* and it's not *poly* anymore.

In the following table 15 we are presenting the performances of all the models tested and their accuracy and f-score.

As we can see, using dimensionality reduction algorithm, in this case, did not provide better performances as without it. But on the other hand using tools for balancig the data set especially *TomekLinks* was beneficial for the performance of the model as its accuracy and f-score has increased by 0.1 %. In the following table 16 we are providing the complete classification

C	gamma	kernel	class_weight	tol
1	scale	rbf	balanced	0.0001

Table 14: Best hyper-parameters in reduced dimension

Table 15: Performances of all trained models

Dimensionality	Sampling	Accuracy (%)	F1-score (macro) (%)
Full	Without	95.25	95.18
	SMOTE	95.32	95.25
	Tomek Links	95.35	95.28
Reduced	Without	94.44	94.44
	SMOTE	94.03	94.00
	Tomek Links	94.40	94.39

report for the Support Vector Machine algorithm using *TomekLinks* as undersampling technique for imbalanced learning.

Table 16: Classification Report

Class	Precision	Recall	F1-Score	Support
WALKING	0.932	0.998	0.964	496
WALKING_UPSTAIRS	0.962	0.907	0.933	471
WALKING_DOWNSTAIRS	0.964	0.955	0.959	420
SITTING	0.941	0.910	0.925	491
STANDING	0.926	0.945	0.936	532
LAYING	0.998	1.000	0.999	537
Accuracy			0.954	2947
Macro Avg	0.954	0.953	0.953	2947
Weighted Avg	0.954	0.954	0.953	2947

As expected, the class *LAYING* is the best one predicted which will be seen also in the following sections. It's recall tells us that has been predicted correctly in all the cases which can be also seen in the confusion matrix and Precision-Recall curve in the graph 21. The precision of this class it's not 100% because as can be seen *SITTING* was once incorrectly predicted as *LAYING* which we can say that holds the logic of this data. The two classes which have the worst performance are *STANDING* and *SITTING* which are more often incorrectly predicted between them.

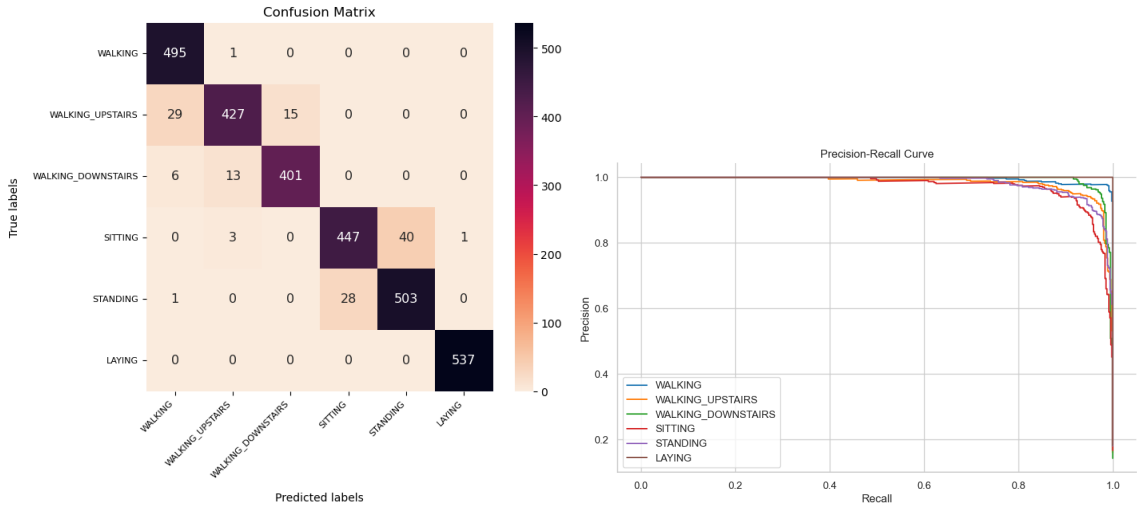
4.3 Neural Networks

Nowadays, Neural Networks are the most common machine learning algorithm known. In this study, we are going to build various models using two different libraries. Firstly, we will train a basic *Perceptron* from scikit-learn library. Using the same library we will use Multi-layer Perceptron which permitted us to introduce hidden layers and the number of neurons. And finally, we will try to use Keras from *Tensorflow* library which will be combined with *KerasClassifier* from scikit-learn library which have permitted us to use RandomizedSearchCV for hyper-parameter tuning.

4.3.1 Perceptron

For this model we have used GridSearchCV with 5-fold cross-validation as its computational expensivness is not too high it was possible to check every possible combination of the parameters in the table 17.

The performance of this model was surprisingly quite good with accuracy and f-score of 93.48 %. For educational purposes this study was more focused on deep neural networks using Keras and KerasClassifier and MLP Classifier from scikit-learn which is more flexible and complex and permits to train models with more complexity.



(a) Confusion Matrix

(b) Precision-Recall curve

Figure 20: Best SVM model characteristics

Parameter	Values	Best Value
penalty	l1, l2, elasticnet	l1
alpha	0.0001, 0.001, 0.01, 0.1	0.001
max_iter	100, 200, 500	100
tol	1e-3, 1e-4, 1e-5, 0.1	0.0001
class_weight	balanced , None	balanced

Table 17: Parameters for the Perceptron Classifier

4.3.2 MLP Classifier

To train this type of neural networks we have used RandomizedSearchCV as it's more difficult and time consuming to train them. We have computed 3 types of hyper-parameter tuning changing the number of hidden layers in order to find the best parameters for the model with 1, 2 and 3 hidden layers respectively. The number of neurons used was 16,32,64,100,128,256 and 512. These numbers were combined in the models with 2 and 3 hidden layers (512,256,128 for example). In the following tables (18 and 19) it can be observed all the hyperparameters chosen for the tuning using RandomizedSearchCV because of high computational complexity and its best parameters chosen for 3 different number of hidden layers.

Parameter	Values
activation	logistic, tanh, relu
solver	sgd, adam
batch_size	16, 32, 64
momentum	0.1,0.5,0.9
learning_rate	constant, adaptive
alpha	0.001, 0.01, 0.1, 0.2

Table 18: Parameters for RandomizedSearchCV of MLP Classifier

As can be seen, the best model performance is for 2-hidden layer MLP classifier with 128 and 64 number of layers with 93.76 % accuracy and macro F-score respectively.

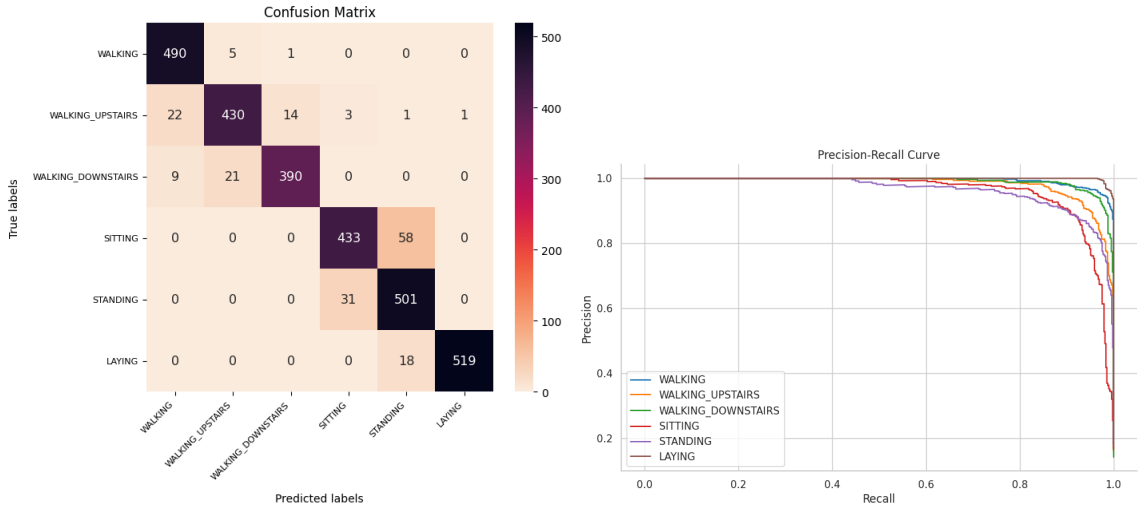
In comparison with support vector machine we can clearly see that the overall performance has decreased. Mainly difference which can be observed is that the class *LAYING* in this case is not predicted in 100% cases. Its precision remains the same as before with difference that is not *SITTING* which is predicted as *LAYING* but *WALKING_UPSTAIRS* which can't be considered that logical as in the previous case.

Parameter	**(256)**	**(128,64)**	**(64,32,16)**
activation	logistic	tanh	tanh
solver	adam	adam	sgd
batch_size	32	64	64
momentum	0.1	0.1	0.1
learning_rate	constant	constant	adaptive
alpha	0.01	0.01	0.2
Accuracy (%)	93.45	93.76	93.38
F_score(%)	93.49	93.76	93.28

Table 19: Best hyperparameters of MLP Classifier with different hidden layers

Table 20: Classification Report

Class	Precision	Recall	F1-Score	Support
WALKING	0.940	0.988	0.964	496
WALKING_UPSTAIRS	0.943	0.913	0.928	471
WALKING_DOWNSTAIRS	0.963	0.929	0.945	420
SITTING	0.927	0.882	0.904	491
STANDING	0.867	0.942	0.903	532
LAYING	0.998	0.966	0.982	537
Accuracy			0.938	2947
Macro Avg	0.940	0.937	0.938	2947
Weighted Avg	0.939	0.938	0.938	2947



(a) Confusion Matrix

(b) Precision-Recall curve

Figure 21: Best MLP model characteristics

4.3.3 Deep Neural Networks

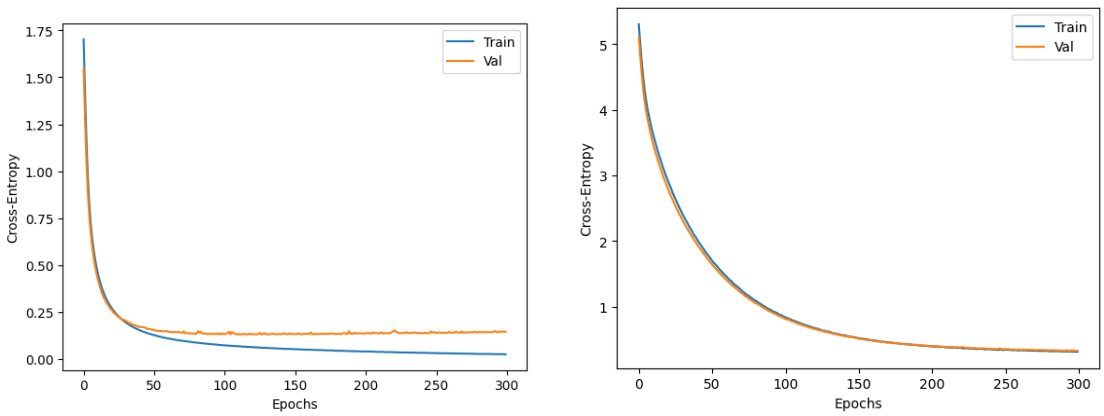
For this type of neural networks, we have used the same approach as in the previous section. It was trained 3 different models varying the number of hidden layers during the hyperparameter tuning. It was used RandomizedSearchCV in order to reduce the time of training as this models require the highest amount of time to train. As we have already mention during the presentation of the section of Neural Networks. To train these models we have used *Keras* from *Tensorflow* library and *KerasClassifier* to combine these two types of libraries after the hyperparameter tuning. As we are doing multiclass classification we have used *Sparse_categotical_crossentropy* as a loss function. As an activation function we have selected *softmax* as this function is commonly used for multiclass classifications.

Parameter	Values	Best Value
optimizer_learning_rate	0.001, 0.01, 0.1	0.01
model_hidden_layer_sizes	(100, 100), (128, 64), (256, 128), (512, 256)	(256,128)
model_activation	relu, tanh	relu
epochs	200, 300, 400	300
batch_size	32, 64, 128	128
momentum	0.1, 0.3, 0.5, 0.9	0.1
model_optimizer	SGD, Adam	SGD
model_dropout_rate	0.05, 0.2, 0.5	0.5

Table 21: Parameters for the Deep Neural Network

In the table 21 we can see all the parameters used for the hyperparameter tuning. In this table is presented especially the case for 2 hidden layers as it has resulted the best one after all the 3 models was trained.

In this model we have focused our training to prevent the overfitting as this type of neural networks are the most flexible once and really likely to overfitt. To prevent this phenomenom we have implemented one *dropout layer* before the output layer with value 0.5 which means that randomly drops 50 % of nodes from the last hidden layer. Second tool used to prevent overfitting was to implement *l2 regularization* equals to 0.01 in every layer.



(a) Before regularization and dropout layer

(b) After regularization and dropout layer

Figure 22: Overfitted vs. corrected loss function

Table 22: Classification Report for Deep Neural Network

Class	Precision	Recall	F1-Score	Support
WALKING	0.909	0.990	0.948	496
WALKING_UPSTAIRS	0.949	0.870	0.908	471
WALKING_DOWNSTAIRS	0.942	0.929	0.935	420
SITTING	0.914	0.866	0.889	491
STANDING	0.887	0.930	0.908	532
LAYING	0.998	1.000	0.999	537
Accuracy			93.25%	2947
Macro Avg	0.933	0.931	93.13%	2947
Weighted Avg	0.933	0.932	0.932	2947

We can see that the performance of this model is a little lower than the previous neural network using MLP Classifier, but still remains high. It can be observed that this type of neural network has predicted the *LAYING* 100 % times which was not true for the previous one where it was confused 18 times by *STANDING* and its recall was a little lower.

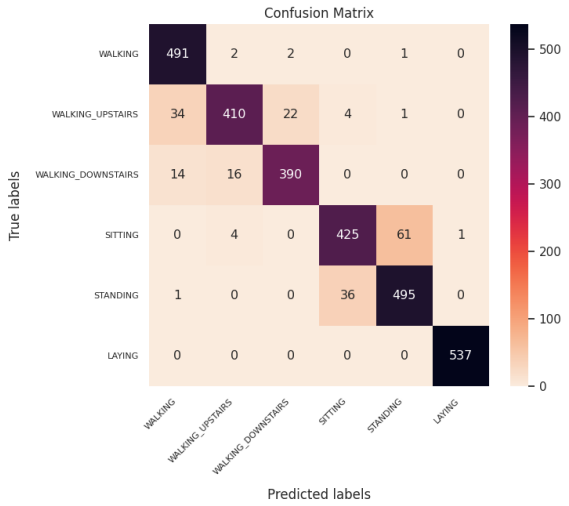


Figure 23: Deep Neural Network Confusion Matrix

4.4 Ensembled Methods

4.4.1 Random Forest

For this classifier we have used the same approach as in the previous sections. Firstly, it was trained in the full dimension space without any dimensionality reduction algorithm. For hyperparameter tuning it was used GridSearchCV from scikit-learn library using 5-fold cross-validation. After this model was trained we have tried to reduce the space to 40 dimensions using Recursive Feature Elimination and repeat the tuning in order to find the best match of hyper-parameters and reduce the overfitting. The third approach tested was to reduce dimensions using feature selection algorithm *SelectFromModel* where the algorithm has selected 44 features and test the model on the test set. In the following table 23 we are presenting all the hyper-parameters used for the tuning and its best values for every approach already described.

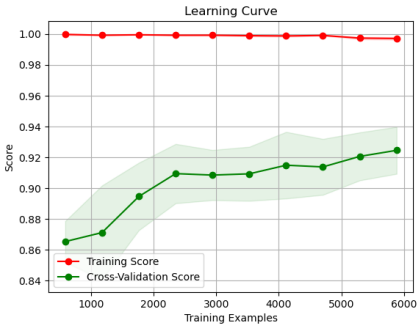
Parameter	**Values**	**Full Dimension**	**SelFMod**	**RFE**
n_estimators	50, 100, 150	150	50	50
max_depth	6, 7, 8	8	8	8
min_samples_split	3, 4, 5	3	3	3
min_samples_leaf	4, 6, 8	4	4	6
criterion	gini, entropy	gini	gini	entropy
max_features	sqrt, log2	log2	log2	log2
max_samples	0.4, 0.5, 0.6, 0.7	0.7	0.5	0.7

Table 23: Parameters for the Random Forest Classifier

These parameters was selected especially because of preventing the overfitting what we have detected during the training. We were obligated to reduce the max_depth to smaller numbers because of this problem. In the figure 24 we are presenting two learning curves obtained before and after reducing it. The first parameters *max_features* reduce the number of maximum of features considered for the best split and we have used two parameteres recommended by the library of scikit-learn. *Max_samples* parameter was also used in order to prevent overfitting as much as possible and it reduce the maximum number of samples in each tree.

Model	**Accuracy**	**F_score**
Full dimension	92.70	92.58
SelFmod	90.77	90.62
RFE	91.41	91.36

Table 24: Model performances with different approaches



(a) Overfitted Random Forest

(b) Better fitted Random forest

Figure 24: Overfitted model vs. Corrected model

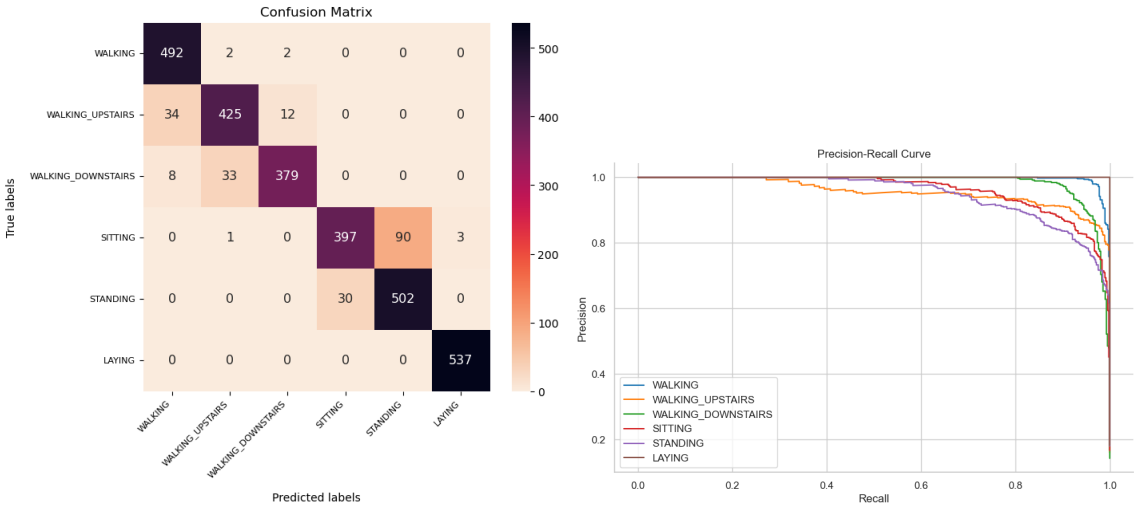
In the table 24 we can see how these models have performed which can take us to conclude that the best model was without using any algorithm for dimensional reduction.

Table 25: Classification Report for the best Random Forest

Class	Precision	Recall	F1-Score	Support
WALKING	0.921	0.992	0.955	496
WALKING_UPSTAIRS	0.922	0.902	0.912	471
WALKING_DOWNSTAIRS	0.964	0.902	0.932	420
SITTING	0.930	0.809	0.865	491
STANDING	0.848	0.944	0.893	532
LAYING	0.994	1.000	0.997	537
Accuracy			0.927	2947
Macro Avg	0.930	0.925	0.926	2947
Weighted Avg	0.929	0.927	0.926	2947

In the table 25 we can observe the overall classification report for this classifier. It can be conclude that also in this case remains the same phenomenom as in the previous classifier where *STANDING* and *SITTING* have the biggest missclassification between them where *SITTING* has the lowest recall between all of classifiers. Also in this case, *LAYING* has 100% recall which means that has been classified correctly in all its cases.

In the figure 25 can be observed that *SITTING* was predicted 3 times as *LAYING* which is a slightly worse as for SVM or deep neural network. In conclusion, this classifier works still good in overall F-score but is a little worse than the previous models.



(a) Confusion Matrix

(b) Precision-Recall curve

Figure 25: Random Forest model characteristics

4.4.2 BaggingClassifier

In this experiment, the preprocessed dataset with 152 dimensions was initially used as Train, on which a *RandomizedSearchCV* with *CrossValidation=5* and *10 iterations* was performed. to find the best values for the *C* and *Kernel* parameter for *SVC* and the number of estimators for Bagging. The result of the *RandomizedSearchCV* was *'C': 770, 'kernel': 'linear', 'n_ estimators': 28*. Next, the *BaggingClassifier* was trained with *SVM* with the values found and was used to make the prediction on the '*Test*' set. Next, the *Train* and the *Test* set were reduced with *SelectFromModel* using *SVC* as *estimator*, resulting in two 42 dimensions datasets. Afterwards, oversampling was performed with *SMOTE* on the *Train*. Finally, *RandomizedSearch* was performed on the latter to train the *BaggingClassifier* with *SVC* with the best values. The result was '*C': 5192, 'kernel': 'linear', 'n_ estimators': 25*. Lastly, the prediction was performed on the 'Test' set with the BaggingClassifier with the best values drawn on the trainset without 2% outliers, reduced with SFM and oversampled. The results can be seen in the figure 26:

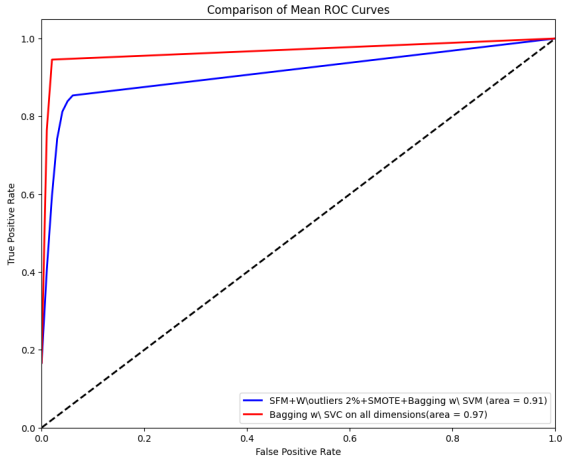


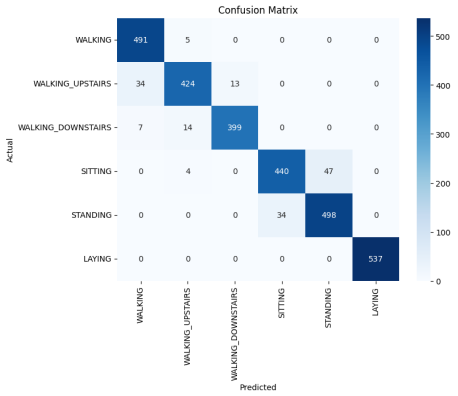
Figure 26: Mean ROC Curves for Bagging with SVM

As can be seen in the ROC curves above, the best results are obtained with the *BaggingClassifier* with *SVC* as estimator and trained on the "*train*" set with 152 dimensions. In this case, the average *AUC* is 0.97 with the *accuracy* of 95% and the *F1-Macro Avg* of 95%. As a matter of synthesis, only the results of the Bagging Classifier with SVC trained and tested on the 152-dimensional datasets will be analyzed.

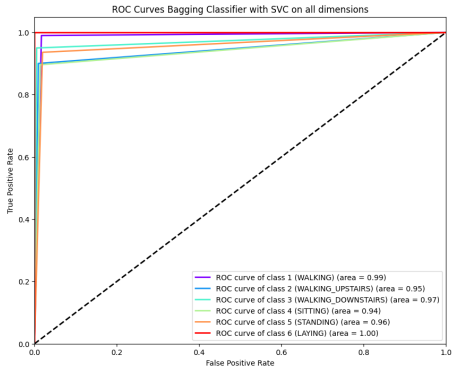
Activity	Precision	Recall	F1-score	Support
WALKING	0.92	0.99	0.96	496
WALKING_UPSTAIRS	0.95	0.90	0.92	471
WALKING_DOWNSTAIRS	0.97	0.95	0.96	420
SITTING	0.93	0.90	0.91	491
STANDING	0.91	0.94	0.92	532
LAYING	1.00	1.00	1.00	537

Table 26: Classification Report Bagging w/SVC on 152 dimensions

From the *classification report*, it can be seen that the *Bagging classifier with SVC* on 152 dimensions always correctly predicts the class "LAYING" (F1-Score:100%), the predictions on "WALKING"(F1-Score:96%) and "WALKING_DOWNSTAIRS"(F1-Score:96%) are also great. Instead, "SITTING"(F1-Score:91%) and "STANDING"(F1-Score:92%) are often misclassified as visible in the *confusion matrix* below. It can be said that the performance of the classifier is excellent overall.



(a) Confusion Matrix Bagging w/SVC on 152 dimensions



(b) Roc curves Bagging w/SVC on 152 dimensions

Figure 27: Best model for Bagging w/SVC

4.4.3 AdaBoost Classifier

In this experiment, we used the "*Train*" dataset reduced to 152 dimensions, on which a *RandomizedSearch* was initially performed with *Cross Validation=5* and *10 iterations* to find the best parameters for *n_ estimators* and *learnig rate* for *ADABOOST* and *max depth* for *Random Forest*. The best values for the parameters are *max_ depth: 10* for the *Random Forest*, *learning rate: 0.5* and *n_ estimators:149*. Next, ADABOOST with Random Forest was trained on the "*Train*" set with the optimal values and prediction was performed on the "*Test*" set. Next, 2% outliers were removed from the *Train* file with *ABOD* with *n_ neighbors=10*. After that, *Train* and *Test* files were reduced with *SelectFromModel* with *Random Forest* as *estimator*, obtaining 37 dimensions. Then the Train file without outliers and with reduced dimensionality was overampled with *SMOTE*. Finally, the latter dataset was used to train *ADABOOST with Random Forest*, and prediction was performed on the *Test* file. The results can be seen in the figure 28:

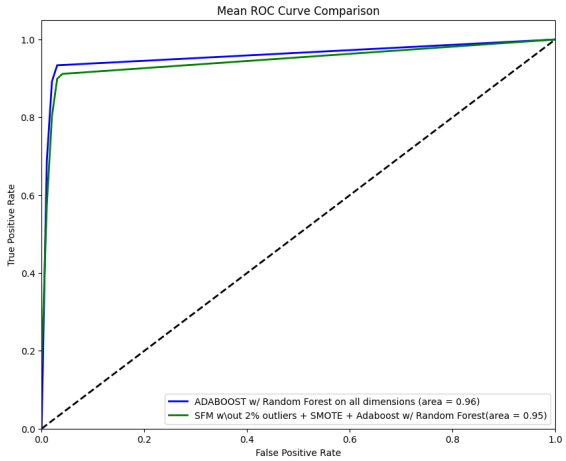


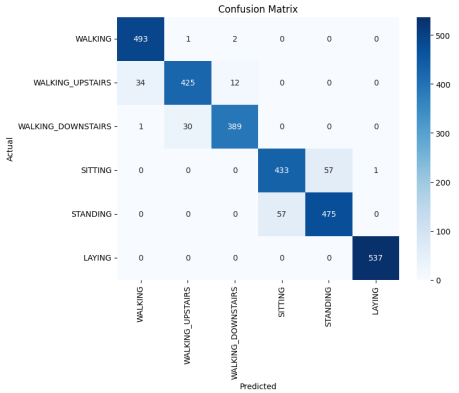
Figure 28: Mean ROC Curves for ADABOOST with Random Forest

As observable from the *mean Roc Curves*, *ADABOOST* classifier with *Random Forest* on the 152-dimensional preprocessed dataset performs slightly better than the dataset without 2% outliers, reduced to 37 dimensions, and oversampled with *SMOTE*. In fact, the *AUC* on the 152-dimensional dataset is 0.96 with *accuracy* of 93% and *F1 Macro Avg* of 93% while on the 37-dimensional dataset the *accuracy* is 91% as well as the *F1 Macro Avg*.

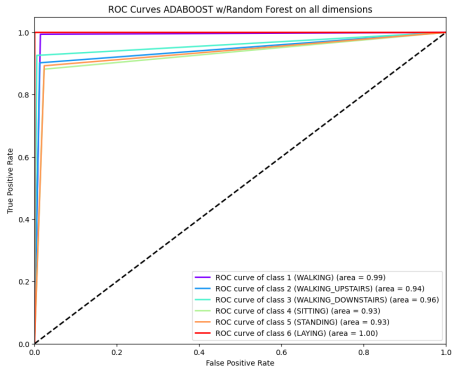
Activity	**Precision**	**Recall**	**F1-score**	**Support**
WALKING	0.93	0.99	0.96	496
WALKING_UPSTAIRS	0.93	0.90	0.92	471
WALKING_DOWNSTAIRS	0.97	0.93	0.95	420
SITTING	0.88	0.88	0.88	491
STANDING	0.89	0.89	0.89	532
LAYING	1.00	1.00	1.00	537

Table 27: Classification Report ADABOOST w/Random Forest on 152 dimensions

From the classification report, it can be seen that the best predicted classes are "LAYING" (F1 Score: 100%), "WALKING" (F1 Score: 96%) and WALKING_DOWNSTAIRS (F1 Score: 95%). "SITTING" (F1 Score: 88%) and "STANDING"(F1 Score: 89%) are the classes with the lowest F1-score and are often misclassified (as visible in the Confusion Matrix below). Despite this, it can be said that the performance of the classifier is excellent.



(a) Confusion Matrix Adaboost w/Random Forest on 152 dimensions



(b) Roc curves Adaboost w/Random Forest on 152 dimensions

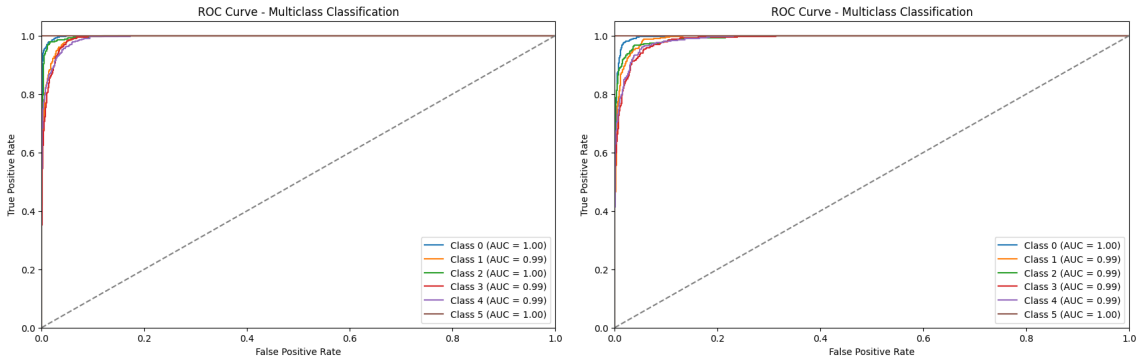
Figure 29: Best model for Adaboost w/Random Forest

4.4.4 Gradient Boosting Classifier

in this section we use Gradient Boosting Classifiers, and some of its variants, as methods, they are based on the gradient principle to optimize the loss function. Initially for hyperparameters, some were selected through gridsearch and then those that obtained better performance with low computational cost were selected. The XGBClassifier and LGBMClassifier methods are variants that speed up the computation but are slightly more imprecise. Regarding the parameters used for the Gradient Boosting Classifier, learning rate of 0.1, n estimators equal to 100 and max depth=3 were chosen. In the table below, it can be seen that the performance of LAYING is maximal, and on the other hand, the performance of WALKING DOWNSTAIRS is more lacking. Overall, the classifier performs optimally, obtaining the best results among all the algorithms used so far.

Method	Accuracy	F1 score
GradientBoostingClassifier	0.93	0.93
XGBClassifier	0.92	0.92
LGBMClassifier	0.92	0.92

Table 28: Gradient Classifiers scores



(a) GradientBoostingClassifier ROC curve

(b) LGBMClassifier ROC curve

Figure 30: ROC curves of Gradient Classifiers

Table 29: Classification Performance of GradientBoostingClassifier

Class	Precision	Recall	F1-Score	Support
WALKING	0.89	0.99	0.93	496
WALKING_UPSTAIRS	0.93	0.85	0.89	471
WALKING_DOWNSTAIRS	0.97	0.93	0.95	420
SITTING	0.89	0.88	0.89	491
STANDING	0.89	0.90	0.90	532
LAYING	1.00	1.00	1.00	537
Accuracy			0.93	2947
Macro Avg	0.93	0.93	0.93	2947
Weighted Avg	0.93	0.93	0.93	2947

5 Advanced Regression

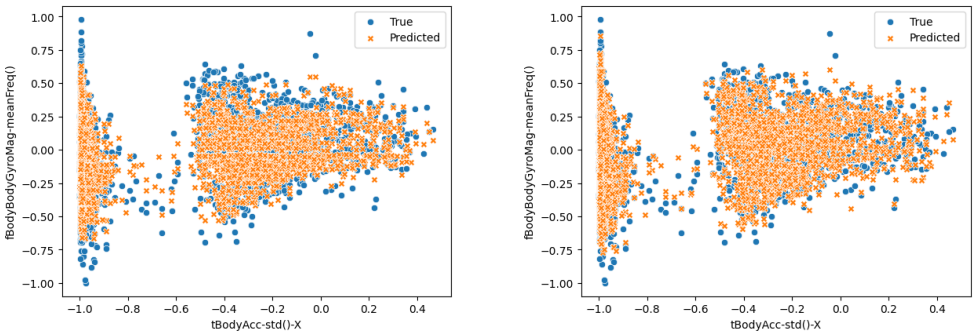
To address the Advanced Regression task, we decided to use fBodyBodyGyroMag-meanFreq() as the target feature. To perform this task, preprocessing steps were necessary in order to eliminate other features that are directly or indirectly related to the target measure, such as: fBodyBodyGyroMag-skewness(), fBodyBodyGyroJerkMag-min(), etc... After this cleaning, we used two algorithms to predict the values of the target variable, namely Gradient Boosting Regressor and the Adaboosting regressor, set to the following parameters:

- Gradient Boosting: n_estimators = 200, random_state= 123
- AdaBoost: n_estimators = 200, random_state = 42

Method	R2 score	MSE
Gradient Boosting regressor	0.88	0.009
AdaBoost regressor	0.81	0.013

Table 30: Regressor scores

As can be seen in the table with R2 score and MSE(Mean Square Error), the best performance was obtained by the Gradient Boosting regressor, which obtained a higher value of both R2 and a lower score of MSE. It can be seen from the results that both models succeed in predicting the outcome with good accuracy, and with relatively little error, this can also be seen from the following images that show a representation of the two regression algorithms with the predicted values and the actual values. Looking at the distribution of the data, it can be seen that the values that have the greatest error are those related to the prediction of the points furthest from the densest zone.



(a) ADA Boost Regressor

(b) Gradient Boost Regressor

Figure 31: Plots of predicted labels

6 Time Series Analysis

6.1 Data preparation and understanding

in this part of our analysis, we extracted the time series from the project folder. the time series dataset was already normalized and the noise had been cleared, so no further transformations needed to be made to the time series. the dataset was presented as 9 files representing the triaxial data of: Total acceleration, body acceleration, and the angular velocity vector. Three values taken from the accellerometer and the gyroscope of the smartphone used as the data collection tool. Each of these values was sampled in 2.56-second windows, thus being divided into 128 windows for each sample. Since to obtain the body acceleration the gravity constant was subtracted from the total acceleration, the two values would be redundant, so it was decided to take the body acceleration as the values and sacrct the total acceleration values, greatly reducing the computational complexity of the subsequent tasks. also given the high dimensionality, it was decided to reduce the sample length from 128 to 32 by using two compression methods, Discrete Fourier Transformation and Piecewise Aggregate Approximation.

6.2 Clustering

- *Partitional: K-means with 4 clusters and Dft and Sax metrics were used for this method, and euclidean distance as a metric*
- *Hierarchical: the agglomerative algorithm with different linkages was used for this method*

6.2.1 K-means

In order to find the best K, a search was carried out in a range of k values between 1 and 9, in order to find the elbow of the curve, with a number of clusters that can minimize the error and that uses a minimum number of clusters, so we decided to take 4 clusters for the analysis. in addition we also checked which among 3 or 4 clusters could be the best choice, for this evaluation we relied on the observation of the silouhette with these two values of k, and it turned out that 4 clusters possess a better silouhette. To make comparability reliable, we chose to apply the same number of clusters to both DFT-approximated and PAA-approximated time series.

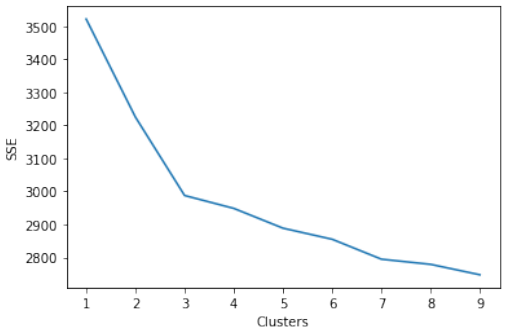
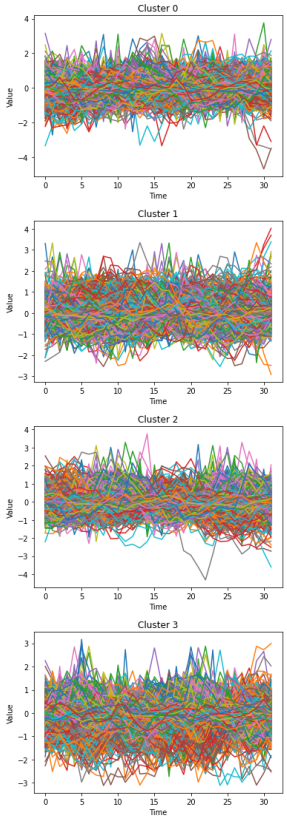
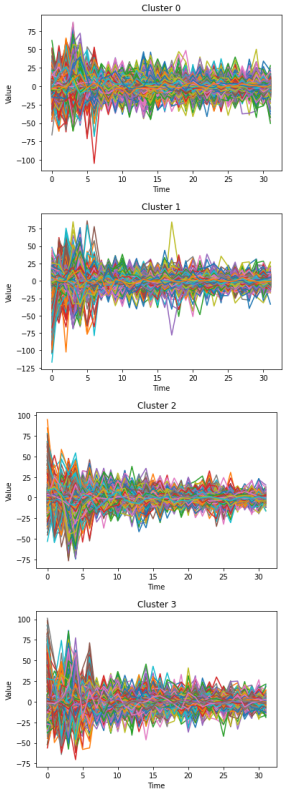


Figure 32: Trade off SSE - n. of K

It can be seen from the figure that the DFT and PAA approximations returned quite different clusters in the distribution of the data, this can also be seen from the related cross-tables, which show the different activities performed by each object. Looking at the distribution of activities in the various clusters shows that the three Walking activities are present in all 4 clusters, while the other activities are for the vast majority contained in cluster 0, this is very evident in cluster 2 contains exclusively Walking values. This peculiarity is common to both the Kmeans performed with DFT and those performed with PAA, so we can conclude that there is a substancial difference between the 'Walking' group and the others activities. However, both possess a silouhette of 0.39 demonstrating not good separation between clusters.



(a) Cluster with PAA



(b) Cluster with DFT

Figure 33: Best model for MLP classifier

	0	1	2	3
WALKING	1027	59	86	54
WALKING UPSTAIRS	362	274	119	318
WALKING DOWNSTAIRS	392	332	25	237
SITTING	1268	18	0	0
STANDING	1371	1	0	2
LAYING	1372	8	0	27

Table 31: CrossTab with Clusters and Labels using DFT

	0	1	2	3
WALKING	778	54	316	78
WALKING UPSTAIRS	265	359	219	230
WALKING DOWNSTAIRS	312	255	111	308
SITTING	1268	0	1	17
STANDING	1371	2	0	1
LAYING	1375	27	0	5

Table 32: CrossTab with Clusters and Labels using PAA

6.2.2 Hierarchical

For this clustering method, an agglomerative algorithm was used by testing different types of linkage and data approximations. To be more specific, the experiment that gave the best results was the one with DFT and single linkage. The complete general dendogram is shown in the figure, and the table describes the distribution of the data after selecting 4 clusters as a parameter of the algorithm so that it is comparable with the clusters in the kmeans. the silouhette value obtained

is 0.61, however, the clusters were found to be insignificant because, except for the densest cluster, the other clusters contain single values that make the distribution of the clusters inconsistent and insignificant. The DFT method was found to be better because it has both a slightly higher silouhette than the PAA method, which possesses a value of 0.59, and also has a slightly better cluster distribution. the only significant observation is that only the label Walking Upstairs is present in all 4 clusters.

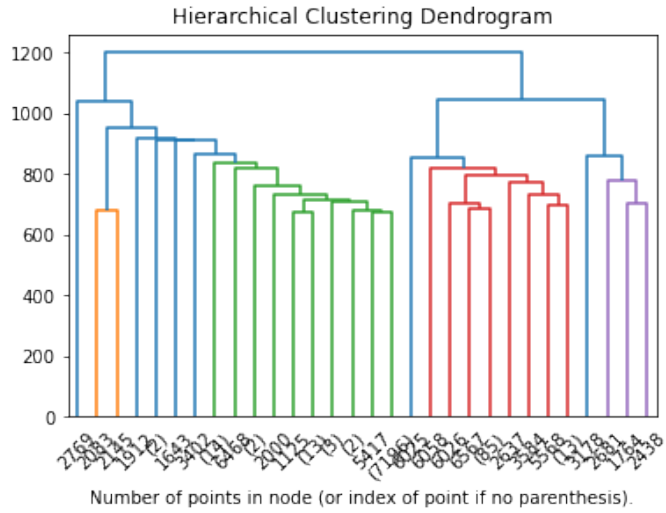


Figure 34: Hierarchical Dendrogram

	0	1	2	3
WALKING	1226	0	0	0
WALKING UPSTAIRS	1073	0	0	0
WALKING DOWNSTAIRS	983	1	1	1
SITTING	1286	0	0	0
STANDING	1374	0	0	0
LAYING	1407	0	0	0

Table 33: CrossTab with Clusters and Labels using Hierarchical clustering

6.3 Classification

For the classification task,we decided to use some classification algorithms trained on part of the data to be able to predict the target variable, which in this task corresponded to the activities performed by the test subjects. trained the classifiers and analyzed their performance, we were able to compare which among them was most accurate in predicting the actual value. Whereas, having 6 possible outcomes, the value of the random classifier, that is, of guessing a class by randomly assigning it, was 1/6, or 0.166. The six target values to be identified correspond to: WALKING, WALKING UPSTAIRS, WALKING DOWNSTAIRS, SITTING, STANDING, LAYING. Because of computational problems,due to the high number of values to be analyzed we used the dataset approximated with DFT, which gave a better performance than PAA. In order to better visualize and compare the various methods, a comparative table was prepared in which Accuracy performance and the average F1 score(i.e., the harmonic mean between accuracy and recovery) are shown.

- Shapelet Classifier
- Shapelet-based KNN
- Shapelet-based Decision Tree
- KNN euclidean distance

- KNN manhattan distance
- Convolutional Neural Network
- Rocket Classifier

to be precise, it should be said that for the shapelet parameters, a search was made to find the ideal number of subsequences that best classified the model, and then the distance between the 6 shapelets found(present in the first model) and the records present in the train set was calculated, and in turn these distances were used as training data.

	Accuracy	F1 score
Shapelet Classifier	0.41	0.33
Shapelet-based KNN	0.44	0.44
Shapelet-based Decision Tree	0.46	0.46
KNN euclidean distance	0.68	0.68
KNN manhattan distance	0.71	0.71
Convolutional Neural Network	0.78	0.76
Rocket Classifier	0.27	0.26

Table 34: Performance of Classifiers

Optimization functions were used to determine the parameters of the shapelets, which however do not use gridsearch, given the large number of possible combinations, from what we can see the shapelets have a peculiar shape, given the three-dimensionality of the dataset. The length of the sequences is only 3 values, this since the length of each sample after the approximation has been applied is 32 time stamps. This low length of the shapelets might explain why the classifiers based on them are not very performing, it is worth mentioning however how having 6 classes to predict, and not a binary classification, it is more complex to achieve high performance. the use of Dynamic Time Warping as a measure of distance was attempted, however the high computational difficulty did not make its use possible. In any case, even using KNN with Euclidean and manhattan distances produced appreciable results. The value of k used was 3, as using higher values affected the computation time while not leading to improved performance. As for the convolutional neural network, 5 layers were used: Conv1D, MaxPooling1d, Flatten, Activation Relu, Activation Softmax. In addition, the training dataset was further divided into train set and validation set, so as to improve the accuracy of the result. Finally, two dense nodes were used to process the data and return the output, the first one with 128 units and ReLU activation, which preprocesses the data and the second node with 6 units and softmax activation that returns the probability of belonging to one of the 6 classes. For the rocket classifier were set as the 5000 kernel value, yet despite the speed of this classifier, the results were very sloppy, with the worst performance of all.

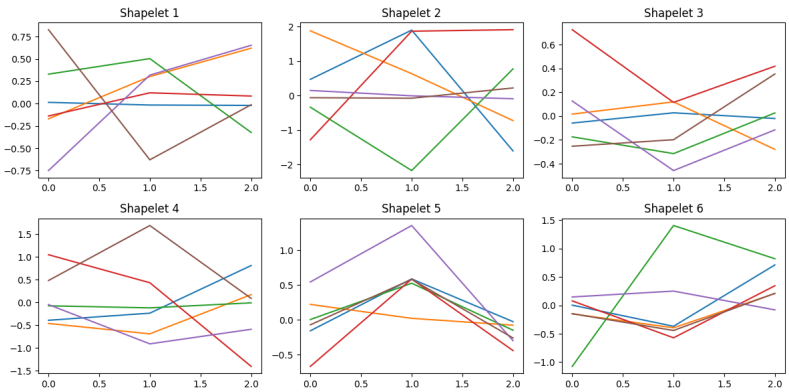
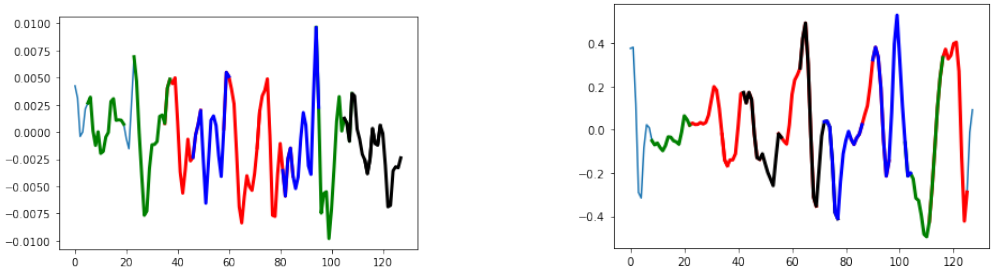


Figure 35: Shapelets

6.4 Motifs and Discords discovery

the task to discover the motifs and and Discords was performed by taking the original dataset without approximations. for this task we selected two sets of the acceleration x-axis, belonging to the same subject, one of which represented the LAYING action and the other the WALKING action. this selection was made to analyze the subsequences of two theoretically opposite actions, and to see if there are any different subsequences or interesting anomalies that might better describe a motion activity from a static activity. trying different lengths for each subsequence, it was decided to set the window size to 15. From this parameter, patterns for the LAYING file and 4 patterns for the WALKING file were processed, with their positions in the time series:

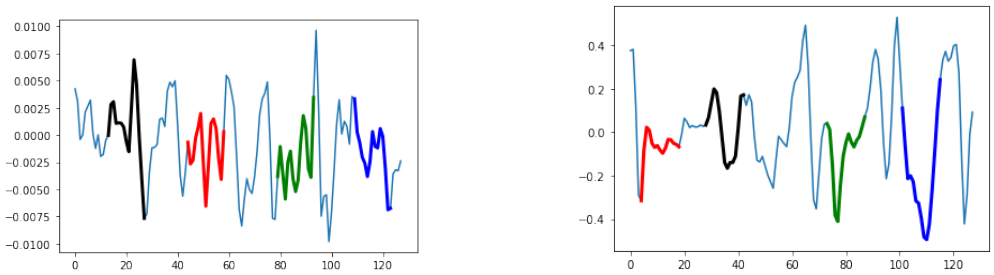
- LAYING: [35, 58, 71], [5, 23, 94], [105, 113], [46, 81]
- WALKING:[22, 34, 55, 82, 111], [8, 102], [42, 63], [72, 90]



(a) LAYING motifs discovery

(b) WALKING motifs discovery

Figure 36: Motifs discovery



(a) LAYNG discords discovery

(b) WALKING discords discovery

Figure 37: Discords discovery

for each Time series there are different patterns both in number of them and in the frequency of each of them within the sequence. undoubtedly the acceleration as far as the subject is concerned when he moves presents more uniform and repeated patterns over time, in contrast the subsequences of the static file are more, but also less repeated. this leads us to theorize that the acceleration data of a person who is walking presents more frequent patterns while those of a person who is stationary are slightly more irregular. The discords, on the other hand, are irregular for both files, is what leads us to believe that they are peculiar to the la time series and do not represent a common feature of the 'activity. They are 4 for each time series and are mainly placed between one Motifs and another type of Motifs.

7 Explainable AI

In the last part of this project, we are introducing two models of explainable AI which in the last few years are always gaining more attention in the world of data science. These algorithms permit to explain the "black-box" of classifier where it can't be seen inside and reasons of the

output are unknown. We have tried to compute two different explainable methods like **LIME (Local Interpretable Model-Agnostic Explainer)** which has permitted us to explain a local prediction and **SHAP (SHapley Additive exPlanations)** as a global approach. As these two models are agnostic and can explain various classifiers we have chosen our best support vector machine model.

7.1 LIME (Local Interpretable Model-Agnostic Explainer)

This explainer permitted us to see a local explanation of chosen classes. We have focused this algorithm to see an explanation of the prediction of *SITTING* and *STANDING*. The behaviour and contribution of various features for these predictions can be seen in the figure 38 and 39 where we have chosen two different instances the first one where *SITTING* was incorrectly classified and the second one where the prediction was correct.

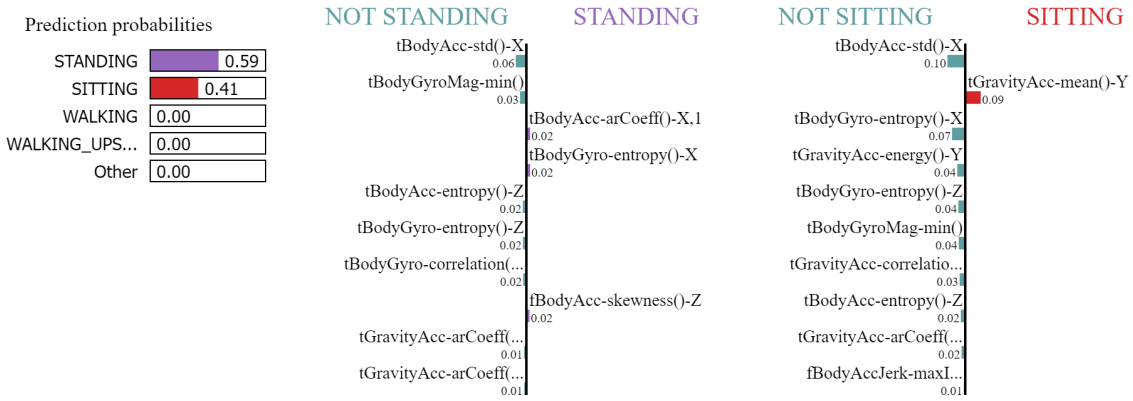


Figure 38: Incorrect prediction of *SITTING*

Figure 39: Correct prediction of *SITTING*

We can notice that in the first case where the target class was missclassified the probability to be correctly classified is just 41% but was missclassified by *STANDING* with the probability of 59%. On the other hand, in the second case where the prediction was correct, no confusion was observed as the probability of its correct prediction is 89% and just 11% for *STANDING*. The major impact on predicting *SITTING* has *tBodyGyro-arCoeff()* which we have also confirmed checking various instances where this phenomenom stayed unchanged.

7.2 SHAP (SHapley Additive exPlanations)

To compute SHAP we have used *KernelExplainer* which is the universal algorithm which can be used for all the classifier that's why we say it's agnostic. This explainer showed us the feature

importance for the prediction and helped us to see which features are important for the prediction of every class. As this algorithm is quite slow we have used just 1500 coalitions in order to speed up the process.

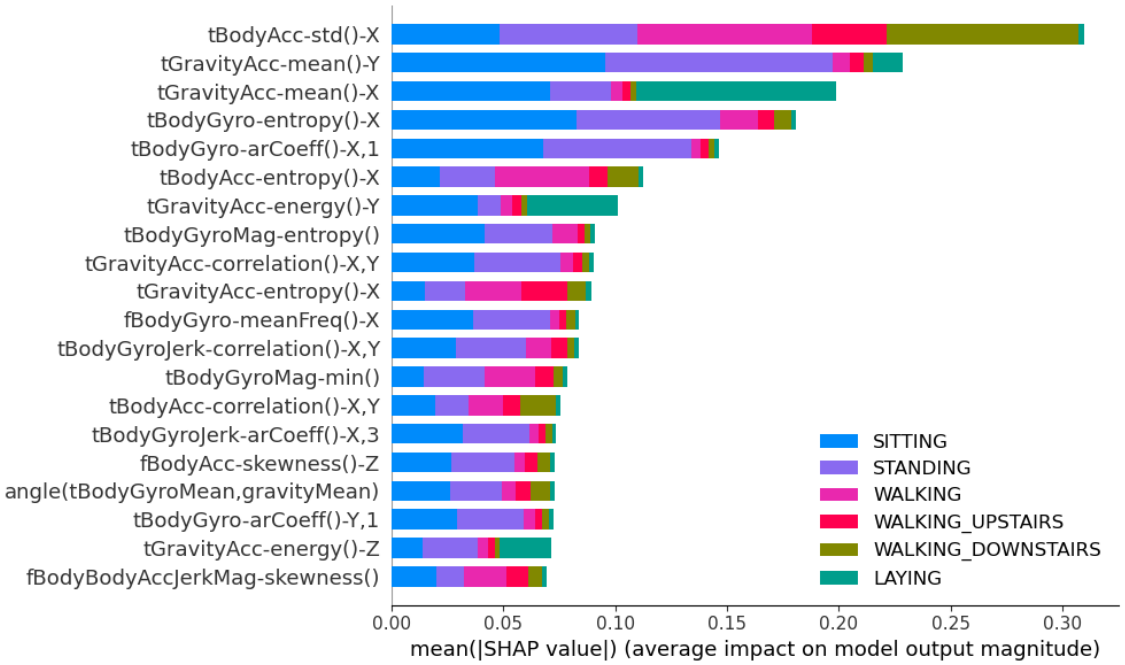


Figure 40: SHAP (SHapley Additive exPlanations)

In the figure 40 we can see the impact of the features on every class. What can be noticed that for predicting *WALKING_DOWNSTAIRS* the most important feature is *tBodyAcc-std()-X*, also this feature has the major impact on this class as it's almost not observed in other classes. Second important observation is that for predicting the most separated and the best predicted class *LAYING* is specially because of *tGravityAcc-mean()-X*. The impact of observed features to predict *SITTING* and *STANDING* is quite equally distributed which can explain the common missclassification of these classes.