UNIVERSITÀ DI PISA

Data Mining report

Analysis of Human Activity Recognition Using Smartphones

Emanuele Sabatini
637756

Michele Papucci
544376

Federico Volpi
648069

Academic year 2021/2022

Contents

1 Module 1 - Imbalanced Learning, Dimensionality Reduction, Anomaly Detection

1.1 Data Understanding

After proper manipulation, the dataset appears to be composed of 563 attributes, of which one is our target y. These attributes are a series of statistical measures applied on the raw time series. It has no missing values.

If we plot the bar chart (Figure 1.1) in order to see the number of data points for each of the activities, we can see that we have more data points for activities 4, 5 and 6. These are the SITTING, STANDING and LAYING activities.

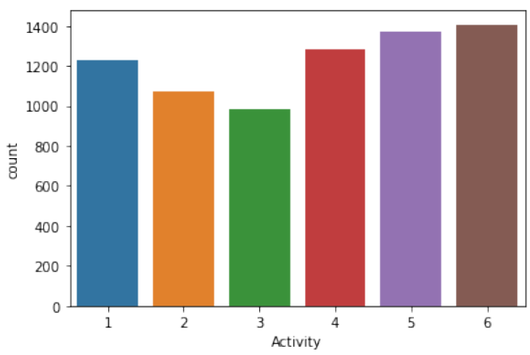

Figure 1.1: Bar chart of activity

If we proceed plotting a bar chart with the count of the data points of each activity for each Id, meaning each time window registered, we can easily observe a peak of data points for the 4, 5 and 6 activities from the 15 window on.

If we order the features by their variance, from the assumption that a higher variance means higher information, we find that the three attributes with the largest variance are those related to entropy and correlation. We proceed plotting the boxplots (Figure 1.2) of the first three variables by variance, all three related to entropy. Signal entropy, in information theory, leads to a higher value if we have higher information. Once we assume this, we can hypothesise a reason for the very different boxplots for the WALKING-related activities. The latter involve more information in comparison to SITTING, STANDING and LAYING, thus leading the instruments to record less information.

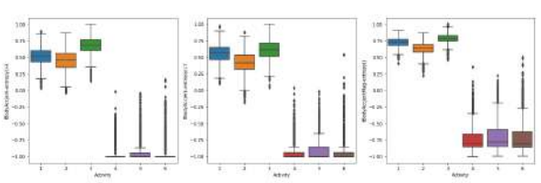

Figure 1.2: Box plots of the first three variables by variance

1.2 Baseline classifier

We defined a baseline classifier to see how the implementation of advanced techniques could improve it. We chose a Decision tree classifier which was fine tuned with a grid search on accuracy. The results are shown in Table 1.

Class	Precision	Recall	F-Score	Accuracy
1	.83	.95	.89	.87
2	.85	.78	81	
3	.89	.83	.86	
4	.87	.76	.81	
5	.80	.89	.84	
6	1	1	1	

Table 1: Decision Tree Classifier

To have a better understanding of the baseline classifier and to have more parameters to benchmark against with other classifiers we also decided to plot the ROC Curve and to calculate the area under the curve for each Activity against the others as shown in Figure 1.3.

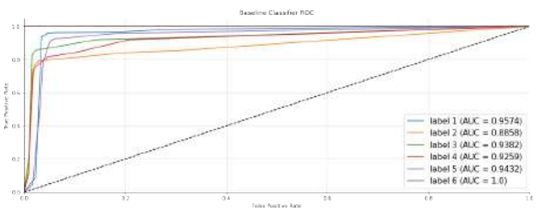

Figure 1.3: ROC Curve Baseline classifier

1.3 Outliers Detection

For Outliers detection we tested a host of different methods to try to find the best one, in particular the following methods, each belonging to a different kind approach for outlier detection, have been tested:

- K-nearest neighbour, or KNN, (distance-based approach)
- Local Outlier Factor, or LOF, (density-based approach)
- Angle Based Outlier degree, or ABOD, (angle-based approach)
- Isolation Forest, or IFO, (model-based approach)
- Intersection of the above

Beside the four classic algorithms we've also tried to classify as an outlier only the data point which have been identified as such by all four of the algorithms. The Figure 1.4 summarizes the results of the various outlier detection methods, both showing which point were classified as outliers and which percentage of the dataset has been identified as such.

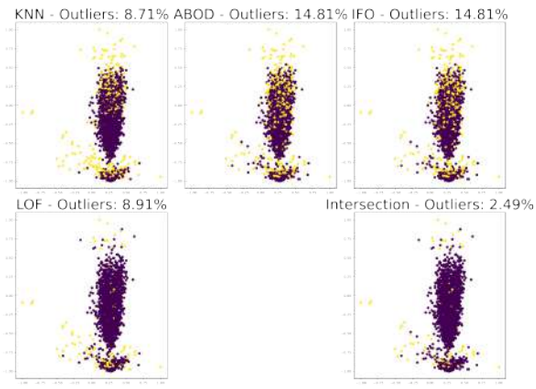

Figure 1.4: Scatterplot outliers

To decide which one of these five performed better we've created five different dataset each one with the outliers removed from one of the methods. Then, we trained another Decision Tree on that dataset and compared the performances.

While none of the approaches drastically improved the classifier performance, we saw a slight improvement in the accuracy and F-Score in the model trained with just the KNN outliers removed as shown in Table 2.

Class	F-Score	Accuracy
1	.87	.88
2	.79	
3	.86	
4	.84	
5	.87	
6	1	

Table 2: Decision Tree Classifier with KNN outliers removed

The ROC Curves of KNN (Figure 1.5) and of the Baseline model are similar, with only activities 4 and 5 having an area under the curve significantly larger than the baseline model.

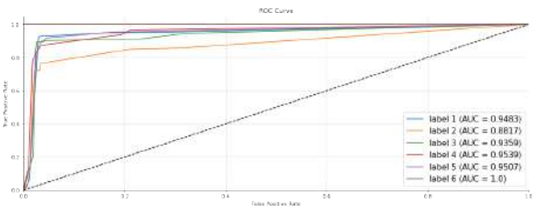

Figure 1.5: KNN - ROC curve

1.3.1 Conclusions

As we saw, the only technique that actually improved the baseline classifier was KNN, which is also one of the approaches that identified less outliers (8.7% of the dataset, against ABOD and IFO which identifies almost double the outliers).
This probably means that KNN correctly identified outliers that were negatevly affecting the performance of the baseline model, and kept records that were useful.
However "saving" more data alone can't be the reason it performed better, since the Intersection approach identified as outliers only the 2.49% of the datapoints but still performed worse than both KNN and the baseline model, with a 85% accuracy and lower F-Scores and Areas under the curve.
The low percentage of the Intersection approach also means that every technique identified largely different records as outliers, and for this dataset KNN is probably the better choice in identifying the outliers that actually matter.

1.4 Imbalance Learning

Since the class distributions in the original Training set are relatively well balanced with each other (Table 3), we initially tried to see how the distribution of the classes changed by removing all the outliers (655) identified by the KNN (Table 4).

Class 1	Class 2	Class 3	Class 4	Class 5	Class 6
1226	1073	986	1286	1374	1407

Table 3: Training set class distributions

Class 1	Class 2	Class 3	Class 4	Class 5	Class 6
1199	1044	941	1075	1219	1219

Table 4: Training set class distributions without KNN outliers

The classes that were most affected were class 4 and class 6, where about 61% of the overall outliers were found. Despite the removal of the outliers there was no significant imbalance in any class, consequently, in order to obtain a highly unbalanced dataset, records were randomly removed from both a static class (6 - LAYING) and a dynamic one (2 - WALKING UPSTAIRS), until we have 50 records in both. From class 6, 96% of records have been removed, and 95% of records from class 2. Table 5 shows the distribution of classes after the imbalance.

Class 1	Class 2	Class 3	Class 4	Class 5	Class 6
1199	50	941	1075	1219	50

Table 5: Training set class distributions after imbalance

The performance of the classifier (Figure 1.6) compared to the one trained with the initial training set is lower, even if class 6, despite the imbalance continues to have excellent results, just a 10% drop in recall and consequently also a decrease in the F1-score. The results of class 2, on the other hand, worsen considerably and the greatest decrease occurs in recall, from 78% to 37%.

```
Accuracy 0.7994570749915167
F1-score [0.8438061  0.50951684 0.75102041 0.81619938 0.82643172 0.94602552]
              precision    recall  f1-score   support

           1       0.76      0.95      0.84       496
           2       0.82      0.37      0.51       471
           3       0.66      0.88      0.75       420
           4       0.83      0.80      0.82       491
           5       0.78      0.88      0.83       532
           6       1.00      0.90      0.95       537

    accuracy                           0.80      2947
   macro avg       0.81      0.80      0.78      2947
weighted avg       0.81      0.80      0.79      2947
```

Figure 1.6: Decision Tree on the unbalanced dataset

The lower performance of the model in predicting class 2 is even more evident when comparing the ROC curves of class 2 (Figure 1.7) to those of the other classes.

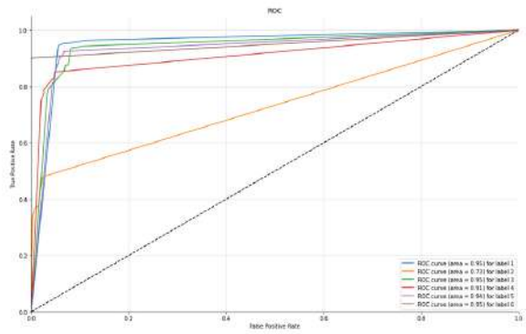

Figure 1.7: ROC curve - imbalance

1.4.1 Random Undersampling

The classes with higher frequency were reduced to the same frequency (50 records) as the two classes with lower frequency, extracting the records in a random way without repetition. Then this dataset was used to train a Decision Tree and the results obtained from the model in the Test set are shown in Figure 1.8.

Accuracy 0.7967424499491008
F1-score [0.78070175 0.70394737 0.6978022 0.76578411 0.78564684 0.99906803]

	precision	recall	f1-score	support
1	0.69	0.90	0.78	496
2	0.73	0.68	0.70	471
3	0.82	0.60	0.70	420
4	0.77	0.77	0.77	491
5	0.79	0.78	0.79	532
6	1.00	1.00	1.00	537
accuracy			0.80	2947
macro avg	0.80	0.79	0.79	2947
weighted avg	0.80	0.80	0.79	2947

Figure 1.8: Decision Tree - Random Undersampling

The accuracy of the model does not change, the recall of class 2 improves considerably even if the precision and recall of all the other classes are slightly lower, except for class 6 where precision and recall are 100%. The AUC of class 2 also improves, passing from 0.73 to 0.90 (Figure 1.9)

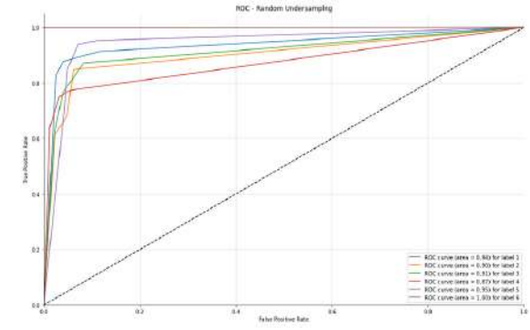

Figure 1.9: ROC - Random Undersampling

1.4.2 Condensed Nearest Neighbour

The results of the undersampling using Condensed Nearest Neighbor, with [1, 3, 4, 5] as target classes of resampling and number of neighbours = 15, are shown in the Table 6;

Class 1	Class 2	Class 3	Class 4	Class 5	Class 6
8	50	10	37	16	50

Table 6: Training set after CNN

The Decision Tree has 0.78 accuracy (Figure 1.10), while the performance of the minority classes improve considerably, those of the classes subject to resampling worsen especially in recall, in particular class 3 passes from a recall of 0.88 to 0.45 and the 4 from 0.80 to 0.52.

Accuracy 0.7760434340006787
F1-score [0.80561555 0.71803556 0.56586826 0.66407263 0.79748823 1.
]

	precision	recall	f1-score	support
1	0.87	0.75	0.81	496
2	0.60	0.90	0.72	471
3	0.76	0.45	0.57	420
4	0.91	0.52	0.66	491
5	0.68	0.95	0.80	532
6	1.00	1.00	1.00	537
accuracy			0.78	2947
macro avg	0.80	0.76	0.76	2947
weighted avg	0.81	0.78	0.77	2947

Figure 1.10: Decision tree - CNN

Also in relation to the AUC that of class two rises to 0.91 while for class 3 it drops to 0.72 and 0.76 for class 4 (Figure 1.11).

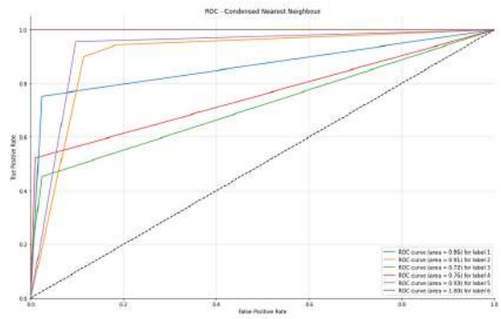

Figure 1.11: ROC - CNN

1.4.3 Random Oversampling

All the classes were over-sampled to a number of 1219 in such a way as to equal the frequency of the majority class (5), extracting records randomly with repetition.

Accuracy 0.837801153715643
F1-score [0.82631579 0.66141732 0.80504587 0.82352941 0.85251799 0.99906803]

	precision	recall	f1-score	support
1	0.73	0.95	0.83	496
2	0.87	0.54	0.66	471
3	0.78	0.84	0.81	420
4	0.87	0.78	0.82	491
5	0.82	0.89	0.85	532
6	1.00	1.00	1.00	537
accuracy			0.84	2947
macro avg	0.84	0.83	0.83	2947
weighted avg	0.85	0.84	0.83	2947

Figure 1.12: Decision tree - Random Oversampling

With this balancing technique, the accuracy of the Decision Tree improves from 0.79 to 0.84. Class 2 recall is still low, but improves from 0.37 to 0.54 (Figure 1.12). While there are no substantial improvements in the AUC which goes from 0.73 to 0.76 (Figure 1.13)

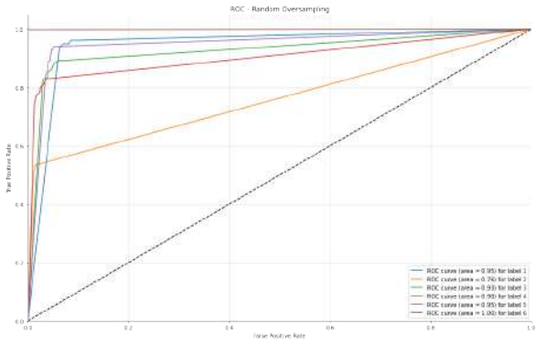

Figure 1.13: ROC - Random Oversampling

1.4.4 SMOTE

Also with this technique the frequency of each class is brought to 1219, in such a way as to equal the frequency of class 5, introducing synthetic records in each class by interpolation, constructed by randomly considering one of the k = 5 neighbours of the selected record.

The accuracy of the Decision Tree obtained after SMOTE (Figure 1.14) is the same as that of the model obtained after Random Oversampling, however, there is a higher AUC for class 2, equal to 0.81 (Figure 1.15).

Accuracy 0.8408551068883611
F1-score [0.79760888 0.66925065 0.8371532 0.83870968 0.85819521 0.99906803]

	precision	recall	f1-score	support
1	0.69	0.94	0.80	496
2	0.85	0.55	0.67	471
3	0.85	0.83	0.84	420
4	0.86	0.82	0.84	491
5	0.84	0.88	0.86	532
6	1.00	1.00	1.00	537
accuracy			0.84	2947
macro avg	0.85	0.84	0.83	2947
weighted avg	0.85	0.84	0.84	2947

Figure 1.14: Decision tree - SOMTE

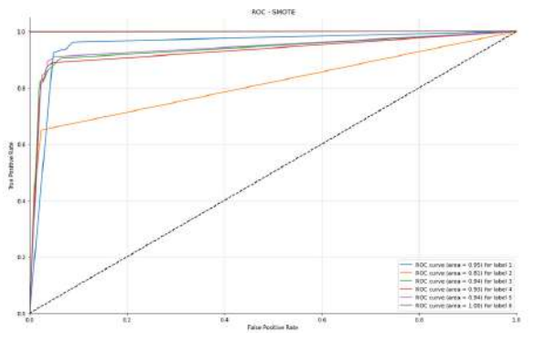

Figure 1.15: ROC - SMOTE

1.4.5 Adjust the class weights

We assigned a higher weight (15) to the minority classes and a lower weight (1) to all the others, leaving the number of records in each class unchanged. The Table 7 shows the weight assigned to each class:

Class 1	Class 2	Class 3	Class 4	Class 5	Class 6
1	15	1	1	1	15

Table 7: Class weights

The performance of the Decision Tree is shown in Figure 1.16, trained taking into account the different weights of the classes, are similar to those obtained with the Oversampling techniques, with an accuracy of 0.84 and an AUC for class 2 equal to 0.76.

Accuracy 0.8327112317611113
F1-score [0.82776801 0.66753927 0.81142857 0.81147541 0.83673469 0.98965193]

	precision	recall	f1-score	support
1	0.73	0.95	0.83	496
2	0.87	0.54	0.67	471
3	0.78	0.85	0.81	420
4	0.82	0.81	0.81	491
5	0.83	0.85	0.84	532
6	1.00	0.98	0.99	537
accuracy			0.83	2947
macro avg	0.84	0.83	0.82	2947
weighted avg	0.84	0.83	0.83	2947

Figure 1.16: Decision Tree - Class weight

1.4.6 Conclusions

Considering the various imbalance learning techniques, the best results in terms of accuracy were obtained by the two oversampling techniques (Random oversampling and SMOTE) and by the decision tree trained with the adjusted class weights. Focusing instead on the AUC, specifically that of the minority classes, the best result was obtained by Random Undersampling.

1.5 Dimensionality Reduction

The large number of attributes calls for a proper dimensionality reduction. For the scope of our analysis, we will consider the following techniques: Principal Component Analysis, ISOMAP, feature selection techniques such as Univariate Feature Selection, Variance Threshold and Recursive Feature Elimination.

1.5.1 Principal Component Analysis

The first algorithm that we use for it is PCA, the Principal Components Analysis.

Accuracy 0.7974211062097047
F1-score [0.84150943 0.79344262 0.70588235 0.6986987 0.75655431 0.96106363]

	precision	recall	f1-score	support
1	0.79	0.90	0.84	496
2	0.82	0.77	0.79	471
3	0.74	0.67	0.71	420
4	0.69	0.71	0.70	491
5	0.75	0.76	0.76	532
6	0.98	0.94	0.96	537
accuracy			0.80	2947
macro avg	0.80	0.79	0.79	2947
weighted avg	0.80	0.80	0.80	2947

Figure 1.17: Decision tree - PCA

On the test set, the algorithm brings us an accuracy of 79% (Figure 1.17), with an even better result in terms of F1-score, 84%. It's interesting to underline how LAYING is the easiest to predict.

To find a good compromise between the number of principal components and accuracy, we chose to consider for our analysis the first ten components. This choice was inferred from the percentage of explained variation of these components. While the first component alone explains 62% of the variation, the second the 4,8% and the fifth 1,7%, thus presenting an imbalance that could be caught with ten components. Plotting the graph of the first two principal components (Figure 1.18), we can clearly see two distinct cluster with an homogeneous incidence of the target variable.

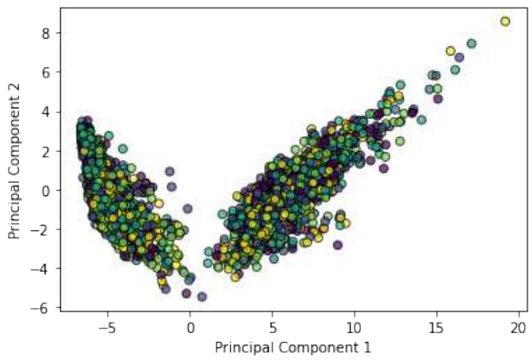

Figure 1.18: First two principal components

1.5.2 ISOMAP

The reason to utilise ISOMAP is to show the performance of a non-linear manifold algorithm that uses the geodesic distance instead of the usual Euclidean one to see if it delivers better results both in terms of reduction and accuracy. In the Figure 1.19, we see the graphical result of dimensionality reduction through this technique.

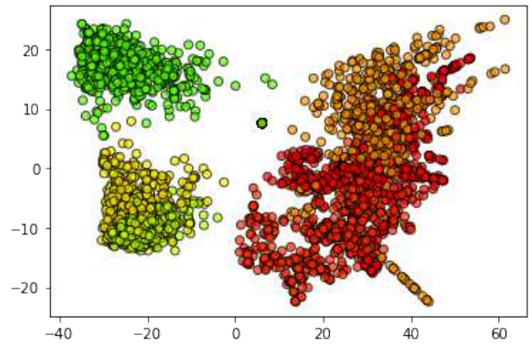

Figure 1.19: ISOMAP

The accuracy is 78,5% and the F1-score varies according to the different actions. While it is capable of strongly reducing the dimensionality to just 2 features with an adequate accuracy with the Decision Tree (Figure 1.20), it performs slightly worse than the PCA with a similar number of components, thus we conclude that to achieve our objectives in the dataset we will use other more conventional linear techniques.

```
Accuracy 0.7846781504986401
F1-score [0.77892031 0.8384     0.77031802 0.66248432 0.66666667 0.98472385]
              precision    recall  f1-score   support

           1       0.74      0.82      0.78       368
           2       0.86      0.81      0.84       322
           3       0.81      0.74      0.77       296
           4       0.64      0.68      0.66       386
           5       0.69      0.64      0.67       412
           6       0.98      0.99      0.98       422

    accuracy                           0.78      2206
   macro avg       0.79      0.78      0.78      2206
weighted avg       0.79      0.78      0.78      2206
```

Figure 1.20: Decision Tree - ISOMAP

1.5.3 Univariate Feature Selection

We move to the three feature selection techniques, starting from Univariate Feature Selection. We again chose a K=10 in order to be coherent with the choice of the PCA. On the Decision Tree Classifier (Figure 1.21), we have as results an accuracy of 70%, with an F1-score that is lower than the accuracy for all the activities except for the 6th one, that again is perfectly guessed. The result is, by the way, the worst one between the feature selection techniques.

```
Accuracy 0.7030878859857482
F1-score [0.6531401  0.68595927 0.65756824 0.58776329 0.60819828 1.
              precision    recall  f1-score   support

           1       0.63      0.68      0.65       496
           2       0.69      0.68      0.69       471
           3       0.69      0.63      0.66       420
           4       0.58      0.60      0.59       491
           5       0.62      0.60      0.61       532
           6       1.00      1.00      1.00       537

    accuracy                           0.70      2947
   macro avg       0.70      0.70      0.70      2947
weighted avg       0.70      0.70      0.70      2947
```

Figure 1.21: Decision Tree - Univariate Feature Selection

1.5.4 Variance Threshold

Shifting to Variance Threshold, the results are better. We set a variance threshold of 83%, in order to get the most relevant variables after a proper hyperparameter tuning. In terms of F1-score, we can also find a lower difference between the different activities as show in Figure 1.22. The accuracy may be higher, but the number of attributes is way too high: 97.

```
Accuracy 0.832711231761113
F1-score [0.8046875  0.7028754  0.8162762  0.81295716 0.83562902 1.
              precision    recall  f1-score   support

           1       0.78      0.83      0.80       496
           2       0.71      0.70      0.70       471
           3       0.85      0.79      0.82       420
           4       0.83      0.79      0.81       491
           5       0.82      0.86      0.84       532
           6       1.00      1.00      1.00       537

    accuracy                           0.83      2947
   macro avg       0.83      0.83      0.83      2947
weighted avg       0.83      0.83      0.83      2947
```

Figure 1.22: Decision Tree - Variance Threshold

1.5.5 Recursive Feature Elimination

The best result was achieved by Recursive Feature Elimination. With the Decision Tree Classifier and a proper tuning, we can achieve a 85% of accuracy with a shape of 35 variables as shown in Figure 1.23. We decided to use this dimensionality reduction technique, even if it involves a slight more number of variables, to maintain the maximum quantity of information possible while strongly reducing the number of variables to a much more manageable dataset.

```
Accuracy 0.8510349507974211
F1-score [0.88555347 0.76769912 0.83830846 0.78151261 0.80987203 1.
              precision    recall  f1-score   support

           1       0.83      0.95      0.89       496
           2       0.80      0.74      0.77       471
           3       0.88      0.80      0.84       420
           4       0.81      0.76      0.78       491
           5       0.79      0.83      0.81       532
           6       1.00      1.00      1.00       537

    accuracy                           0.85      2947
   macro avg       0.85      0.85      0.85      2947
weighted avg       0.85      0.85      0.85      2947
```

Figure 1.23: Decision Tree - Recursive Feature Elimination

Between the tested algorithms RFE had the best impact on classification. The same Decision

Tree trained on a reduced dataset, after a Grid search (Figure 1.24), improves across all class F1-score, and the accuracy gets better. RFE reduces the feature to 32.

Best Parameters: {'max_depth': 9, 'min_samples_leaf': 1, 'min_samples_split': 2}

	precision	recall	f1-score	support
1	0.95	0.93	0.94	368
2	0.94	0.93	0.93	322
3	0.92	0.97	0.94	296
4	0.94	0.93	0.93	386
5	0.93	0.95	0.94	412
6	1.00	1.00	1.00	422
accuracy			0.95	2206
macro avg	0.95	0.95	0.95	2206
weighted avg	0.95	0.95	0.95	2206

Figure 1.24: Decision Tree - RFE after Grid

1.6 Applied Dimensionality Reduction to improve previous tasks

1.6.1 Dimensionality Reduction and Visualization

Using dimensionality reduction we selected a few algorithms that could compress the dataset in just two dimension to visualize it. For this purpose we selected ISOMAP, PCA e t-SNE.
From Isomap we can see two macro-cluster: one composed by activities 1, 2 and 3, and the other composed by activities 4, 5 and 6 as shown in Figure 1.25.

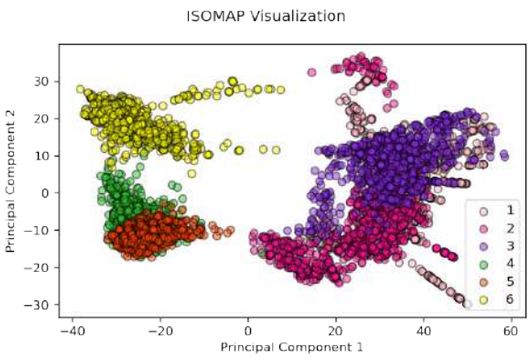

Figure 1.25: ISOMAP visualization

This division make sense from a semantic point of view. In fact, the first three activities are all walking-related activities so it is plausible that the datapoints are similar. The same can be said for the other cluster which is composed by the sitting, standing and laying activities (4, 5 and 6). These are all activities that require no movements, which is one of the major dimension recorded by the sensors from which the dataset has been created, and so it make sense that they should be relatively similar.

The same cluster behaviour discussed in the ISOMAP visualization is even more clear in the PCA visualization (Figure 1.26).

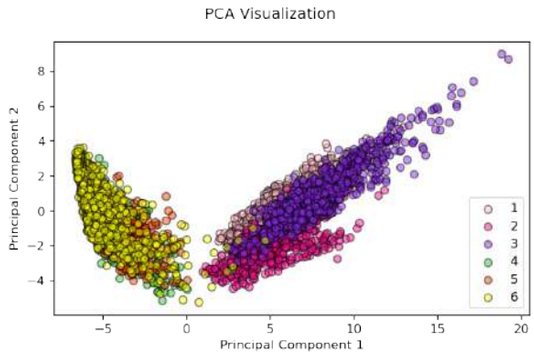

Figure 1.26: PCA visualization

In the t-SNE visualization (Figure 1.27) we can see two dense cluster, one formed by the laying activity (6) and one formed by the sitting (4) and standing (5) activity. The cluster formed by the laying activity is well separated from the cluster formed by the sitting and standing activities. Both these two clusters aver very compact.

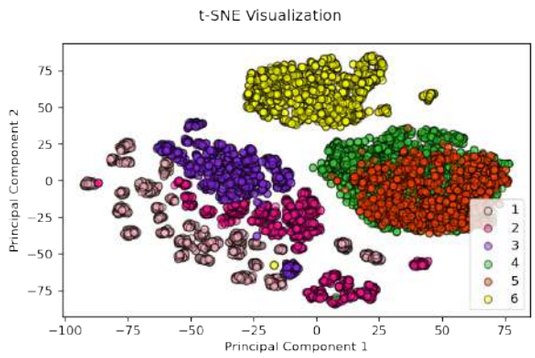

Figure 1.27: t-SNE visualization

Since t-SNE tries to maintain the original distances is plausible to say that the laying activity is in some way distant from the other two activities, probably in the dimensions that regards the angles of the gyroscope.
The same can't be said for the walking-related activities which could be seen as one cluster with a very uneven density. The only activity in it which is quite compact is walking downstairs (3).

1.6.2 Dimensionality Reduction to improve classification

We tried to improve classification tasks using dimensionality reduction techniques. The best performing one was RFE which reduced the dataset to just 35 features. After a grid search on a classifier trained with the reduced dataset we lost some accuracy and the f-score were overall worse than the baseline classifier trained on the original dataset (Table 8).

Class	Precision	Recall	F-Score	Accuracy
1	.86	.93	.90	.85
2	.88	.73	.80	
3	.75	.82	.79	
4	.89	.69	.78	
5	.76	.92	.83	
6	1	1	1	

Table 8: Decision Tree Classifier - Classification on a RFE reduced dataset

While accuracy and F1-scores got worse, the ROC curves and their areas under the curve improved significantly from the baseline model (Figure 1.28).

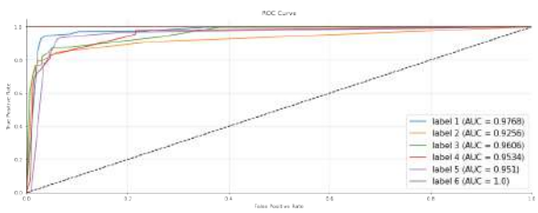

Figure 1.28: ROC - Classification on a RFE reduced dataset

1.6.3 Dimensionality Reduction to improve outliers detection

We tried to improve the outlier detection task by using our best performer outlier detection algorithms, KNN, on the reduced dataset. KNN finds less outliers, 8.5% against the original 8.7%, and after removing the flagged outliers the classification task from 1.6.2 is just sightly better, with a higher accuracy and the F-scores of classes 4 and 5 getting better. From this point of view it still perform worse than the baseline model as shown in Table 9.

Class	Precision	Recall	F-Score	Accuracy
1	.81	.96	.88	.86
2	.83	.80	.82	
3	.88	.74	.80	
4	.80	.81	.80	
5	.82	.81	.81	
6	1	1	1	

Table 9: Decision Tree Classifier - outlier detection on a RFE reduced dataset

The Roc Curves (Figure 1.29) are slightly worse than the ones from the classifier in 1.6.2, but still better than the ones from the baseline model.

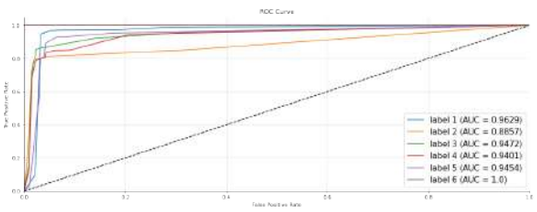

Figure 1.29: ROC - outlier detection on a RFE reduced dataset

2 Module 2 - Advanced Classification Methods

2.1 Classification

2.1.1 Random Forest

For the Random forest we performed the hyperparameter tuning both with the Grid Search and with the Random Search. The performance of the various parameters was evaluated taking into account the accuracy for both methods. The parameters analyzed were:

- "criterion" : Gini and entropy;
- max_dept": with a value between "None" e 20;
- min_sample_split": 2, 5, 10, 20, 30, 50, 100;
- "min_sample_leaf": 1, 5, 10, 20, 30, 50, 100;
- "n_estimators": 10, 20, 30, 40, 50, 60, 70, 80, 90, 100.

The Grid Search found the best parameters: {'criterion': 'gini', 'max_depth': None, 'min_samples_leaf': 20, 'min_samples_split': 50, 'n_estimators': 100} and in the Validation set it has an Accuracy 97%, while the Random Search with 20 iterations identified as parameters: {'criterion': 'entropy', 'min_samples_split': 30, 'min_samples_leaf': 10, 'max_depth': 9, 'n_estimators': 100}, and the Accuracy in the Validation set is the same as that obtained with the Grid Search. In the Test set, on the other hand, the model built with Random search parameters achieved slightly better performances with Accuracy of 91%, while Grid Search of 90%. The performances of the two models are shown in Figure 2.1.

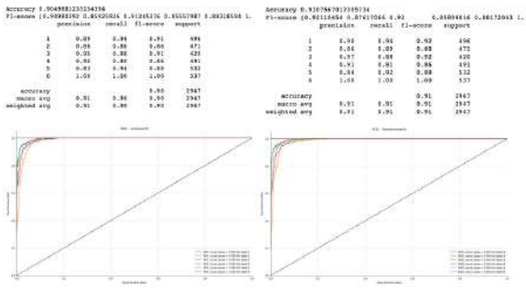

Figure 2.1: Performance Test set GridSearch(left) and Randomsearch(right)

For both methods we have reported the first 15 most important features. For the Grid Search model the most important parameter is "tBodyAccjerk-entropy () - X", while for Random Search it is "angleX.gravityMean()" (Figure 2.2).

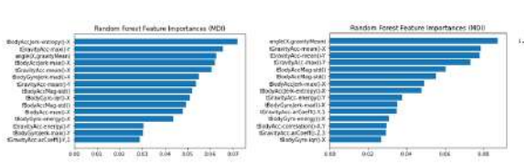

Figure 2.2: Feature importance Grid Search(left) and Random Search(right)

2.1.2 Naive Bayes Classifier

For the Naive Bayes Classifier we used both the Gaussian Naive Bayes and the Categorical Naive Bayes.

The GaussianNB has an accuracy of 85% in the Test set, and a minimum AUC value of 97% for classes 4 and 5 (Figure 2.3).

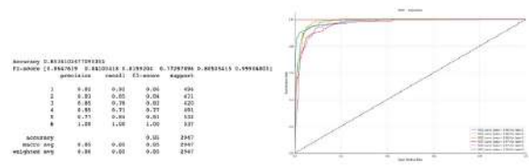

Figure 2.3: GaussianNB

For the CategoricalNB we have discretized the dataset with "qcut" in 4 equal sized groupings of the data using the quartiles. The accuracy in the Test set is 87% and the lowest AUC value is always 97% for classes 4 and 5 as for the Gaussian NB. The performances of the CategoricalNB are shown in Figure 2.4.

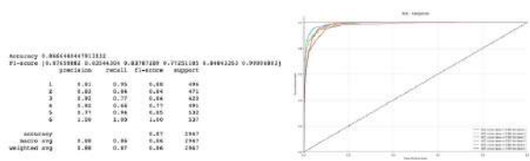

Figure 2.4: CategoricalNB

2.1.3 Neural Network

For the Neural Network we use the sklearn library. To avoid over fitting and to achieve the highest performance possible we decided to do a Grid Search focused on accuracy, testing the following parameters:

- 'hidden_layer_sizes': [(364, 128, 64, 32,), (200, 100,), (100,)];
- 'alpha': [0.1, 0.2];
- 'solver': ['adam', 'sgd'];
- 'activation': ['identity', 'logistic', 'tanh', 'relu'];
- 'batch_size': [100, 200, 250];
- 'power_t': [0.1, 0.3, 0.5];
- 'validation_fraction': [0.1, 0.2, 0.3];
- 'learning_rate': ['constant', 'invscaling', 'adaptive'].

The following parameters have been found to be the best to achieve a high accuracy: {'activation': 'identity', 'alpha': 0.2, 'batch_size': 200, 'early_stopping': True, 'hidden_layer_sizes': (200, 100),' learning_rate ':' constant ',' power_t ': 0.1,' solver ':' sgd ',' validation_fraction ': 0.2}.

The model obtained using these parameters has the accuracy of 95% in the Validation set and 88% in the Test set. The values obtained from the model in the Test set specifically and the relative Loss curve are shown in Figure 2.5.

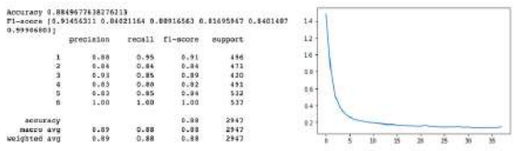

Figure 2.5: Neural Network performance

The minimum AUC obtained from the model is 0.98 for classes 4 and 5, (Figure 2.6).

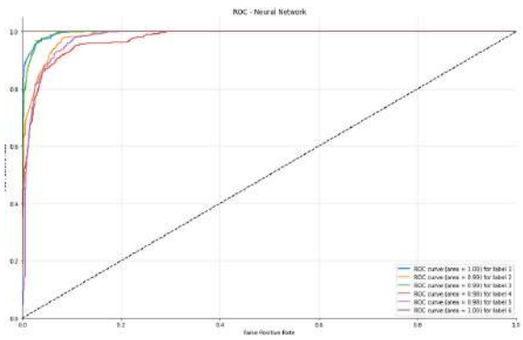

Figure 2.6: ROC - Neural network

2.1.4 Logistic Regression

To implement Logistic Regression, we first pre-processed data properly through StandardScaler(). Then, we introduced the LogisticRegression() classifier with hyperparameter tuning. We analysed a variety of different combinations of arguments, starting from three different solvers: 'newton-cg', 'lbfgs' and 'liblinear'. We considered different values for the C parameter: 100, 10, 1.0, 0.1 and 0.01. We achieved the best result with C=100 and the solver 'lbfgs', thus with the largest regularisation and a non-default solver. The accuracy on the Test set was 89,9%, with a strong record of f1-scores for the different activities as shown in Figure 2.7.

Accuracy 0.8992195453003053
F1-score [0.92706334 0.86401674 0.94146949 0.82781457 0.85915493 0.97621313]

	precision	recall	f1-score	support
1	0.88	0.97	0.93	496
2	0.85	0.88	0.86	471
3	0.99	0.90	0.94	420
4	0.90	0.76	0.83	491
5	0.81	0.92	0.86	532
6	1.00	0.96	0.98	537
accuracy			0.90	2947
macro avg	0.91	0.90	0.90	2947
weighted avg	0.90	0.90	0.90	2947

Figure 2.7: Logistic Regression performance

We also proceeded plotting and analysing the ROC curve (Figure 2.8), in which each activity was plotted against all the rest. We show a consistent record of good scores, in particular regarding activity 1.

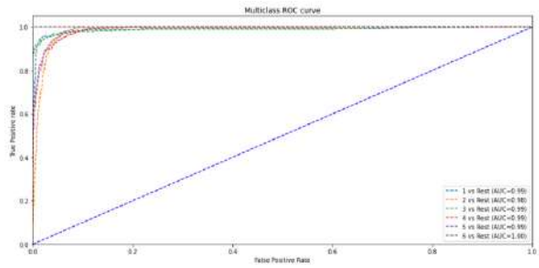

Figure 2.8: ROC - Logistic Regression

2.1.5 Support Vector Machine

We started the implementation of the SVM algorithm through a part of pre-processing through StandardScaler(), in order to standardise data. We used a validation set properly split from the original one in order to tune the hyperparameters. The tuning happened through gridsearch, with a total of 240 fits of a param_grid that acted mostly on three components:

- 'C' = [0.1,1,10,100];
- 'gamma': [1,0.1,0.01,0.001];
- 'kernel': ['rbf', 'poly', 'sigmoid'].

The 'C' parameter is the misclassification cost, the 'gamma' parameter is used by kernel functions to regulate their form and finally the 'kernel' parameter that chooses the best kernel function. A smaller C decreases the penalty of misclassification, a larger one increases it. Instead, a larger gamma gives to the points that are closer to higher weight. The best fit was found being C=100, gamma=0.01 and kernel='rbf'.

Accuracy 0.9060061079063454
F1-score [0.90218424 0.85197018 0.92738854 0.86089814 0.88672566 1.]

	precision	recall	f1-score	support
1	0.85	0.96	0.90	496
2	0.85	0.85	0.85	471
3	1.00	0.87	0.93	420
4	0.93	0.80	0.86	491
5	0.84	0.94	0.89	532
6	1.00	1.00	1.00	537
accuracy			0.91	2947
macro avg	0.91	0.90	0.90	2947
weighted avg	0.91	0.91	0.91	2947

Figure 2.9: Support Vector Machine performance

The accuracy is 90,6% with F1-scores that, again, moderately vary according to the different activity considered (Figure 2.9). We move to the ROC curve (Figure 2.10), in order to have a better graphical representation of the explanatory power of the model, according to the different activities in a one vs all context.

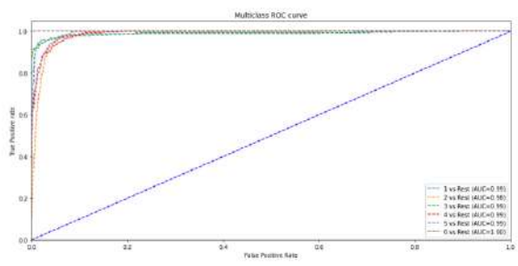

Figure 2.10: ROC - Support Vector Machine

2.1.6 Bagging

We proceeded with another Ensemble Classifier, Bagging, that is based on bootstrapping the dataset with the implementation of a baseline classifier, that we decided it to be Decision Tree for coherence. In order to do so, we tuned the hyperparameters in the following way.

- 'n_estimators': [2,4,8,10,20,50,100];
- 'max_features':[2,4,6,8,10].

GridSearchCV showed as the best result max_features: 2 and n_estimators: 100. Moving on with the classification, we had the following result, with an accuracy of 89,2% and a F1-score that oscillates between 0.97 for activity 6 and 0.84 for activity 4, in line with variations saw before (Figure 2.11).

```
Accuracy 0.8924329826942654
F1-score [0.90640394 0.86419753 0.90978399 0.84088514 0.85821596 0.96925859]
              precision    recall  f1-score   support

           1       0.89      0.93      0.91       496
           2       0.84      0.89      0.86       471
           3       0.98      0.85      0.91       420
           4       0.87      0.81      0.84       491
           5       0.86      0.86      0.86       532
           6       0.94      1.00      0.97       537

    accuracy                           0.89      2947
   macro avg       0.90      0.89      0.89      2947
weighted avg       0.89      0.89      0.89      2947
```

Figure 2.11: Bagging performance

We conclude the analysis with the ROC curve shown in Figure 2.12.

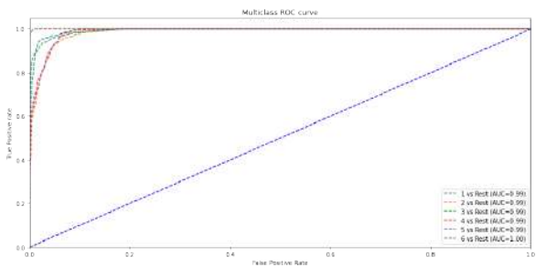

Figure 2.12: ROC - Bagging

2.1.7 Gradient Boosting

We tried various gradient boosting algorithms to make a better classificator from our baseline model, in particular we tried Gradient Boosting, Histogram-based Gradient Boosting, XGBoost e LightGBM. The Scikit Learn implementation of Gradient Boosting, LightGBM and XGBoost performed all really well.
While the results of the three were all really similar, Gradient Boosting performed slightly better regarding the F-Scores (Table 10).

	GBoost		XGBoost		LightGBM	
Class	F-Score	Accuracy	F-Score	Accuracy	F-Score	Accuracy
1	**.96**	.93	.95	.93	.95	.93
2	**.93**		.92		**.93**	
3	**.95**		.94		**.95**	
4	.87		**.89**		.87	
5	.88		**.91**		.89	
6	**1**		**1**		**1**	

Table 10: Gradient Boosting algorithms performance

The Areas under the curve of the ROC's were all pretty similar, with the best one being the LightGBM's one shown in Figure 2.13.

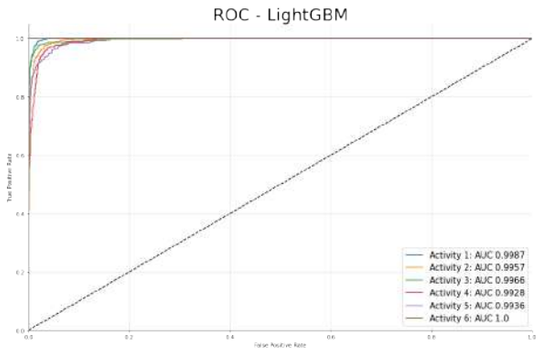

Figure 2.13: ROC - LightGBM

2.2 Conclusions

We've tested various classification techniques, from the tests that we did we can say that the best performing classification method on this dataset is Gradient Boosting, which achieved the highest accuracy and the best areas under the ROC curves for each activity.
The second best algorithm was another ensemble method, Random Forest.

Regarding the Ensemble classifiers, we can see how an increase in the number of estimators refines the classification, for example with Bagging classifier, where the number of estimator implemented by GridSearchCV was the largest defined: 100.

Since both our best performing models were ensemble, we decided to further show the impact in the number of estimators by plotting how the accuracy level of a model rises when the number of estimators rises, for both Random Forest and Gradient Boosting.

Figure 2.14 reports the change in accuracy to the Random Forest model, as the number of estimators varies from 10 to 1000 increasing by 10 each time, with all the other parameters being the ones identified by the Grid Search.

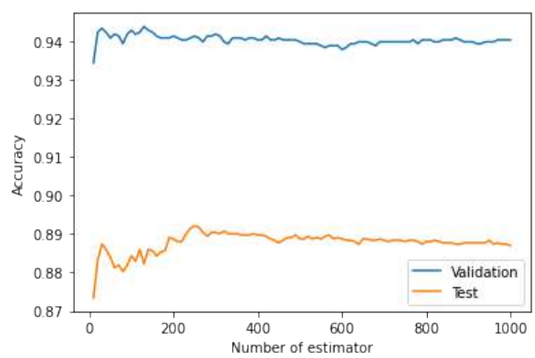

Figure 2.14: Change in accuracy as the number of estimators varies in Random Forest

While in the Validation set the accuracy level does not change particularly, in the Test set there is a peak in the curve at a number of estimators equal to 250.

A similar result was obtained for the Gradient Boosting model, where around a hundred estimators it sharply rises in accuracy and remains constant after that as show in Figure 2.15. This is interesting because while Random Forest seems to have some high spots and low spots even when the number of estimators gets bigger, Gradient boost

doesn't seem to being negatively affect by a higher number of estimators.

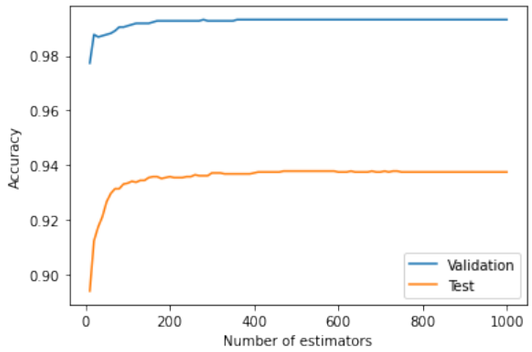

Figure 2.15: Change in accuracy as the number of estimators varies in Gradient Boosting

For both models, we've also tested which was the minimum number of records in the training set to achieve an acceptable level of accuracy.
As we can see in Figure 2.16, for Random Forest, when the training set gets around 500 records, the accuracy level jumps rapidly towards 85%, and from that the accuracy grows much slower.

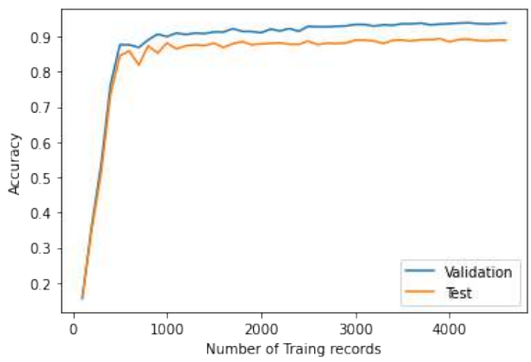

Figure 2.16: Accuracy level on different number of Training records in Random Forest

We've got similar results for Gradient Boost too, but it's important to note how even with a very small dataset, it performed significantly better than Random Forest, as Shown in Figure 2.17.

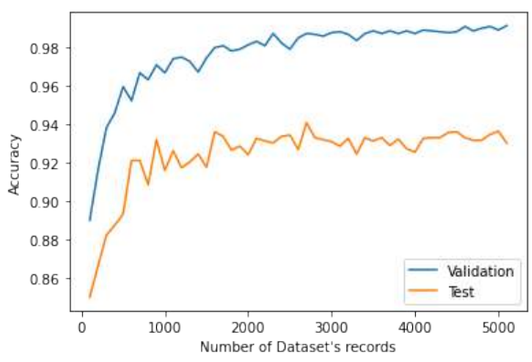

Figure 2.17: Accuracy level on different number of Training records in Gradient Boost

2.3 Linear Regression

We analyzed the simple linear regression, the multiple linear regression with two regularizers (Lasso and Ridge). To compare the results, we used the R2, also called the coefficient of determination. Finally, the MSE, that is the Mean Squared Error and the MAE, the Mean Absolute Error.

We start from the simple linear regression. We've decided to choose two variables: 'fBodyAccJerk-entropy()-X' and 'tGravityAcc-correlation()-X,Z'. This were chosen because of their high feature importance. The former acted as independent variable, while the latter as dependent variable. We've found a very low R2 of 0.005, pointing to a very low degree of variability of Y that can be explained by X. This was confirmed by a high MSE of 0.503 and MAE of 0.637 (Figure 2.18).

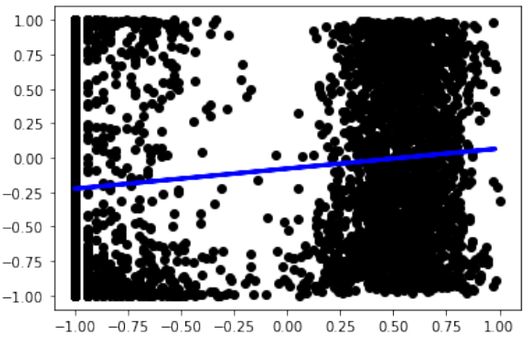

Figure 2.18: 2-D plot of the simple linear regression

Moving to the multiple linear regression, we've decided to maintain 'tGravityAcc-correlation()-X,Z' as dependent variable Y, while filtering it out the rest of the dataset that has been introduced as independent variable X. The regression is perfect, with an R2 of 1 and MSE of 0. This result must be managed carefully. To do this task, the dataset wasn't cleaned of all the redundant variables, opening up to a problem of multicollinearity. When we remove the most correlated variables, we see the R2 slowly reducing, and the MSE increasing. We applied both the Lasso and Ridge regularizers, but without relevant results. In particular, Ridge showed, in fact, the same results as the original model.
We've also applied Machine Learning solution to this regression problem. We tried the same linear problem using a LightGBMRegressor, that confirmed that very little variability of the dependent variable can be explained through the chosen independent variable. We had the same R2 of 0.005, while the error were still significant but a bit lower, with an MSE of 0.497 and an MSA of 0.619.
We then tried a multiple linear regression by using the whole dataset as X, but the results were not as good as the one found previously. We had a R2 value of 0.730, while the error were pretty low with an MSE of 0.130 and an MSA of 0.272. In Figure 2.19 is shown the difference between the original y value and the predicted one by our model.

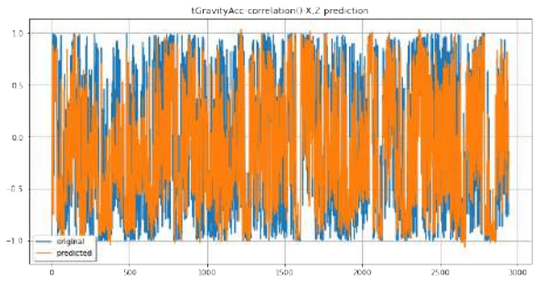

Figure 2.19: Prediction of our dependent variable in a multiple linear regression problem by a LightGBMRegressor

3 Module 3 - Time Series Analysis

3.1 Time series KNN

Before analysing the time series we create a Multivariate time series dataset with all the nine signals, and we decided to not normalize the data because the normalization negatively affected the accuracy of the various classifiers. The dataset consists of 9 signals, and for each signal we have 7352 time series composed of 128 time stamps in the Training set and 2947 time series for each signal in the Test set, always with 128 time stamps. Figure 3.1 shows the first time series for each signal.

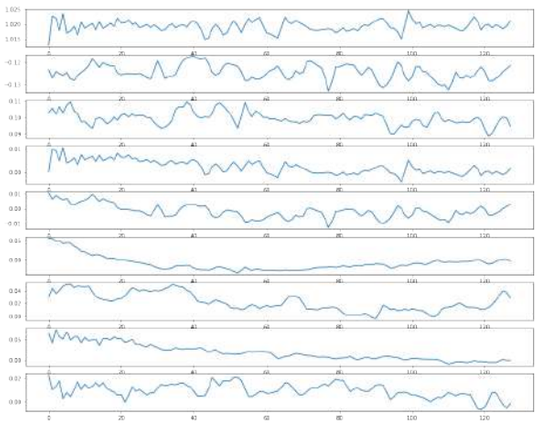

Figure 3.1: First time series for each signal

We did a classification using KNN with k = 5. For this task we tested both the Euclidean and the Manhattan distance and the accuracy obtained by the classifier in the Test set is 75% with the Euclidean and 76% with the Manhattan.

We then decided to focus the analysis only on one signal, so for each signal, taken individually, we performed a KNN with k = 5 using both the Euclidean and Manhattan distance. We chose the best performing one in terms of accuracy and it was "total_acc_x" (Total acceleration X).

Figure 3.2 reports the results obtained with the KNN on the chosen signal in the Test set. With the Euclidean distance we obtain 72% of accuracy and 73% with the Manhattan one.

Figure 3.2: KNN performance euclidean (left) and manhattan (right)

We also performed a further KNN (k = 5) using as distance metric the "Dynamic Time Warping" with sakoe - chiba constraint. The results are similar to those obtained previously with an accuracy of 78% (Figure 3.3).

Accuracy 0.7750254496097726
F1-score [0.90639481 0.78024194 0.78520626 0.55693582 0.6037037 1.]

	precision	recall	f1-score	support
1	0.84	0.99	0.91	496
2	0.74	0.82	0.78	471
3	0.98	0.66	0.79	420
4	0.57	0.55	0.56	491
5	0.59	0.61	0.60	532
6	1.00	1.00	1.00	537
accuracy			0.78	2947
macro avg	0.79	0.77	0.77	2947
weighted avg	0.78	0.78	0.77	2947

Figure 3.3: KNN performance with DTW

Therefore, comparing the results obtained by the three KNNs, the highest level of accuracy is obtained using the Dynamic Time Warping.

3.2 Clustering

We have decided to implement Shape-based, Features-based and Approximated clustering techniques. As before, we selected a single signal to analyze, "total_acc_x". In the Shape-based clustering part, we started to plot TimeSeriesKMeans with the Euclidean distance to start with. To better analyze our results, we've plotted the centroids in Figure 3.4.

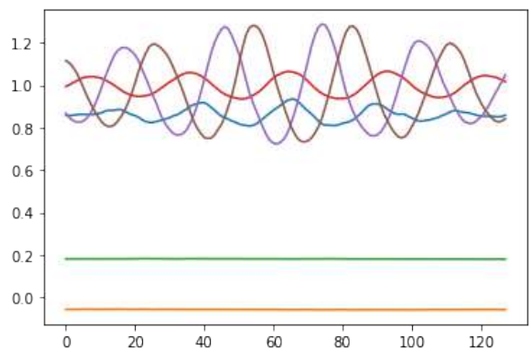

Figure 3.4: TimeSeriesKMeans euclidean - centroid

We've decided to compare the results with another metric, the Dynamic Time Warping. The results of the centroids are shown in Figure 3.5.

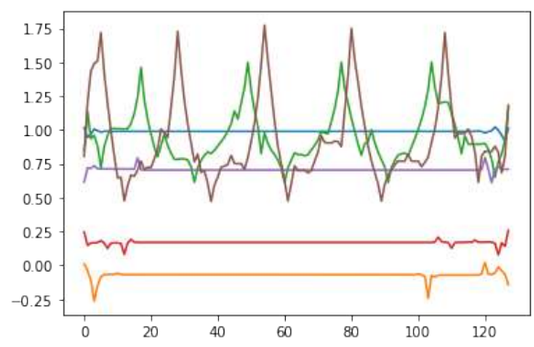

Figure 3.5: TimeSeriesKMeans DTW - centroid

In both cases, we can clearly see the different behaviours between the clusters and we can infer the difference between the actions that involve more movement to those who don't. Comparing the inertia, we can see that KMeans with Euclidean distance provides a 4.24 inertia, while Dynamic Time Warping metric just 0.89, pointing to a reliable result.

In the case of features-based clustering, we implemented in the analysis a wide range of statistical measures. We can still see, plotting the centroids of KMeans with six clusters (Figure 3.6), the difference between the activities. However, the extraordinarily high value of inertia made us discard this method for a more precise one.

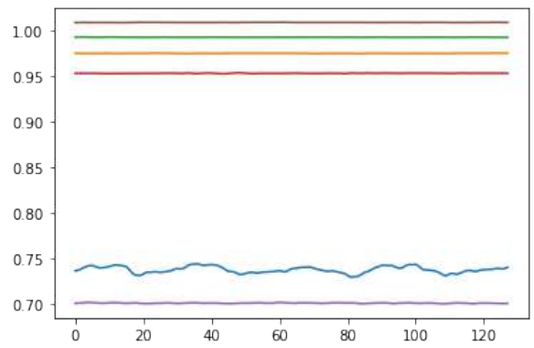

Figure 3.6: centroids of KMeans with six clusters

Finally, in the case of Approximated Clustering we implemented the PieceWiseAggregateApproximation (PAA). The process was implemented with KMeans, both Euclidean and Dynamic Time Warping metric. The graphs are similar to those saw until now, in the case of the PAA with DTW, for example we have the following shown in Figure 3.7.

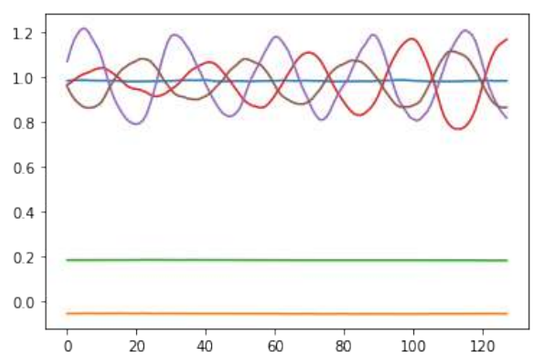

Figure 3.7: PAA with DTW

Inertia is very interesting, as the Euclidean distance brought a value of 0.126, brought down to 0.09 by the Dynamic Time Warping metric. Thus, in the end we can say that we have achieved the best result with PAA with DTW.

3.3 Motifs and Anomalies discovery

We focused our efforts in the motifs and anomalies extraction on the first of the nine signals "total_acc_x". First of all we loaded the dataset and removed the overlap between the windows. Every window of 2.56s had 128 records of the signal with a 50% overlap with the next window.
After that we had around 150 hours of records, that is 463176 records for that signal. To extract motifs we did a bunch of trial and error to find the perfect amount of time to analyze to calculate the matrix profile. After experimenting with it we settled on an value of records to compare of 19200, which is around a 5 minute window. With the matrix profile we could see the highest value for the motifs and the lowest for the anomalies, which are shown in Figure 3.8.

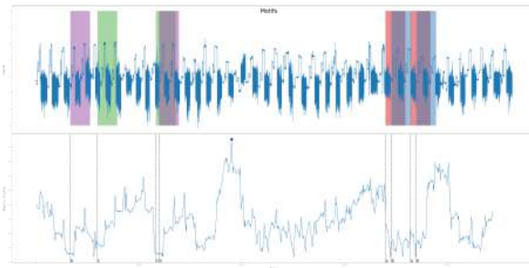

Figure 3.8: Motifs and Anomalies discovery

As we can see we have multiple motifs and a big anomaly, with some other maybe small anomalies towards the end. Is interesting to note that the four motifs towards the end, could also be seen just as two bigger motifs in sequence.

3.4 Time series classification

3.4.1 Brute force approach

For this task we tried both the Brute force approach and the learning approach to apply the shapelet transformation.

3.4.2 Brute force approach

To do the Brute force approach with ShapeletTransform we use only a subset of out training set due to the high time to get the results in our computer, for this reason after dividing the train set into train and validation set, we randomly removed 1546 values from the train thus passing from 5146 records to 3600.
We then ran the ShapeletTransform with the following parameters {'window_sizes' = [4, 12], 'window_steps' = [4, 12], 'random_state' = 42, 'sort' = True} and applied the transformation to the three sets, Train, Validation and Test. The figure 3.9 shows the classes after the transofrmation. On these datasets we have trained a Decision tree, with accuracy of 51% in the Validation and 50% in the Test set, and we built a KNN (k = 5) and we have obtained an accuracy of 56% in the Validation and 54% in the Test set.

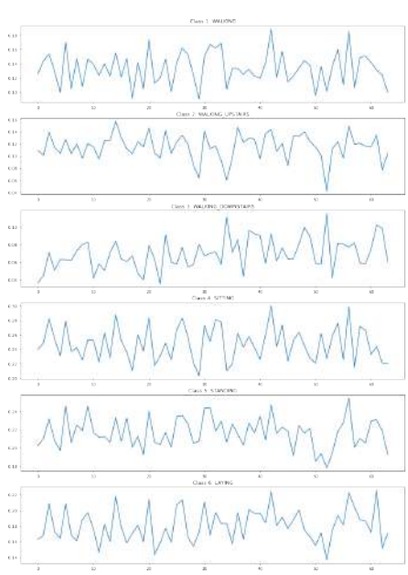

Figure 3.9: The classes after the ShapeletTransform

3.4.3 Learning approach

Unlike the previous approach, in this case we used the entire signal. First we defined the "shapelet sizes" with "grabocka params to shapelet size dict" and the highest levels of accuracy in the various classifiers we obtained with the parameters: {'n_ts' = 5146, 'ts_sz' = 128, 'n_classes' = 6, 'l' = 0.05, 'r' = 15}, where 'l' is the fraction of the length of time series to be used for base shapelet length, while 'r' is the number of different shapelet lengths to use . The result is "shapelet_sizes" = {6: 6, 12: 6, 18: 6, 24: 6, 30: 6, 36: 6, 42: 6, 48: 6, 54: 6, 60: 6, 66: 6 , 72: 6, 78: 6, 84: 6, 90: 6}.

After defining the shapelet size we have performed the tuning of the 'ShapeletModel' and the best parameters we have identified are: {'n_shapelets_per_size' = shapelet_sizes, 'optimizer' = Adam, 'weight_regularizer' = 0.01, 'max_iter' = 1000 }, and we extracted 90 shapelets and the first 10 are shown in Figure 3.10.

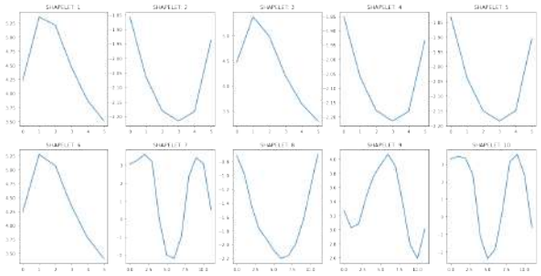

Figure 3.10: First 10 extracted shapelets

Figure 3.11 reports the results obtained by the 'ShapeletModel' in the Test set with 82% accuracy, (84% in the Validation).

Accuracy 0.8181201221581269
F1-score [0.90346084 0.86492622 0.92578616 0.34983498 0.72638889 1.]

	precision	recall	f1-score	support
1	0.82	1.00	0.90	496
2	0.93	0.81	0.86	471
3	0.98	0.88	0.93	420
4	0.92	0.22	0.35	491
5	0.58	0.98	0.73	532
6	1.00	1.00	1.00	537
accuracy			0.82	2947
macro avg	0.87	0.81	0.80	2947
weighted avg	0.87	0.82	0.79	2947

Figure 3.11: ShapeletModel's performance on Test set

We then applied the shapelet transformation on all the sets. The classes after the transformation are shown in Figure 3.12. On the transformed dataset we performed a classification using KNN and Decision tree and for both we used the Grid Search to perform the hyperparameter tuning.

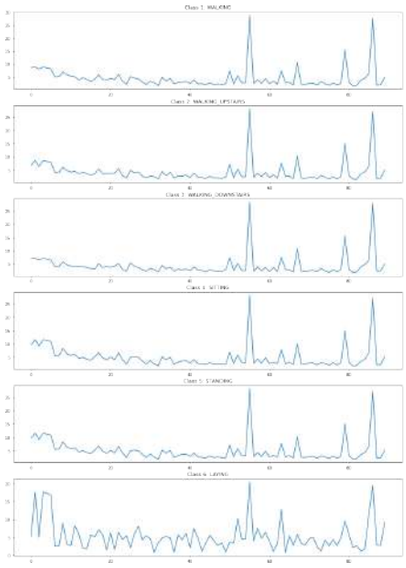

Figure 3.12: The classes after the shapelet transformation

For the KNN the parameters with which we have obtained the highest level of accuracy are: {'metric': euclidean, 'n_neighbors': 9, 'weights': 'uniform'}. The classifier with these parameters has an accuracy of 89% in the Validation and 84% in the Test set. The performance of the KNN are better than that obtained by the ShapeletModel both in the Validation set and in the Test.

In the Decision tree, however, it was trained using the following parameters: {'criterion': gini, 'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 50}. The performance of the Decision tree are similar to the one of the other two classifiers both in the Validation set, with 86% accuracy, and in the Test with 83%.

Comparing the two approaches, the best results were obtained from the learning approach using the KNN as a classifier.

3.4.4 Feature-based Classifier

Starting from the signal 'total_acc_x' we have extracted 13 features (Table 11) creating a new dataset. On the new dataset we repeated the classification task using the KNN and Decision Tree, using the GridSearch for both to perform the hyperparametrer tuning.

'avg'	mean
'std'	standard deviation
'var'	variance
'med'	median
'10p'	10 percentile
'25p'	25 percentile
'50p'	50 percentile
'75p'	75 percentile
'90p'	90 percentile
'iqr'	interquartile range
'Cov'	covariance
'Skw'	skew
'kur'	kurtosis

Table 11: Extracted features

The best parameters for KNN are: {'metric':

manhattan, 'n_neighbors': 6, 'weights': uniform}, while for the Decision Tree the parameters with which the best accuracy is obtained are: {'criterion': entropy, 'max_depth': 2, 'min_samples_leaf': 5, 'min_samples_split': 30}. KNN has an accuracy of 76% in the Validation and 70% in the Test , while Decision Tree has accuracy values equal to 80% and 75% respectively in the Validation and in the Test.

3.4.5 CNN - Convolutional Neural Network

For this classification task we first trained CNN only on "tot_acc_x" and then using all the signals.

The univariate CNN with 100 epochs has an accuracy of 83% in the Validation set and 77% in the Test. On the other hand, the multivariate CNN always with 100 epochs has much better performances, with 97% and 92% accuracy in their respective sets. Figure 3.13 reports the performances obtained by the two classifiers

Figure 3.13: CNN univariate (left) and multivariate (right)