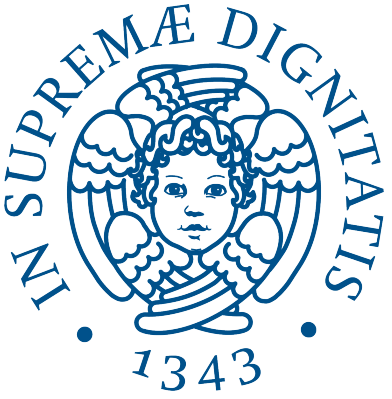


Data Mining 2 - Project

Jornea Ion [637765]
Macchia Alessandro [636679]
Stortoni Cristian [639184]

University of Pisa
A.Y. 2021/2022

Contents

1 Introduction

In recent years, data mining and machine learning techniques have found many applications in movement recognition, thanks to ever more advanced sensors providing a growing amount of data. The current project analyses the "Human Activity Recognition Using Smartphones" dataset, in order to showcase the application of several of such techniques.

The dataset considers data coming from 30 subjects, who wore a smartphone around their waist, and then performed six activities (1 - walking, 2 - walking upstairs, 3 - walking downstairs, 4 - sitting, 5 - standing, 6 - laying). For each record we have triaxal acceleration signals coming the accelerometer and triaxial angular velocity signals coming from the gyroscope, recorded in 2.56 second time windows, for a total of 10,299 records. For each of these, a total of 561 features were extracted, all of which are continuous variables, plus the class label indicating the activity to which that record is associated. All the variables were already normalised and scaled to the range [-1, 1]. The dataset was already divided into a training part and a test part using a 70%-30% split, which we maintained as such. Figure 1 shows the number of records in the training set for each of the classes, which are quite evenly distributed.

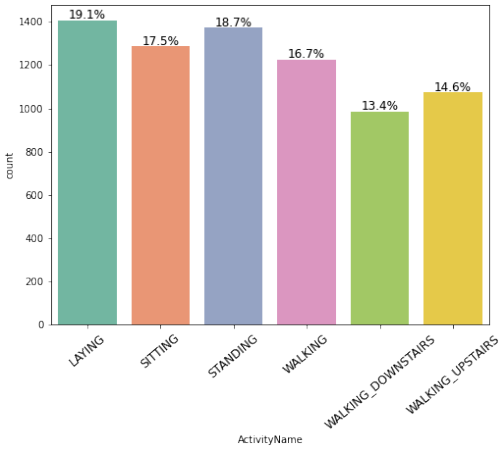


Figure 1: Distribution of the class attribute

2 Treating Anomalies, Imbalances and Dimensionality

We started by considering the whole dataset and running the basic decision tree and K-Nearest Neighbours classifiers. To be able to calculate more meaningful distances, we applied a Z tranformation to all the variables (i.e., subtracting the respective mean and dividing by the standard deviation). Surprinsingly, we immediately had quite strong performances with both methods.

On the test set, the KNN classifier considering 14 neighbours and points weighted uniformely returned an accuracy of 0.89, with a precision of 0.90 (ranging from 0.84 to 0.99 for the various classes) and a recall of 0.89 (range 0.77-0.97). To find this parametre configuration we used a 10-fold cross validation performing a grid search on all possible neighbours between 1 and 15, trying to weigh the points both uniformely and by distance. Even more surprising was the decision tree classifier. By using Gini as impurity measure and requiring at least 1,000 records per leaf, we obtain, on the test set, an accuracy of 0.83, precision of 0.83 (range 0.70-1.00), and recall of 0.82 (range 0.73-1.00). We arrived at this extreme configuration since we noticed that by increasing the number of leaves required in each node, the performance was not decreasing significantly, so we wanted to highlight just how much we could increase it and still have decent results.

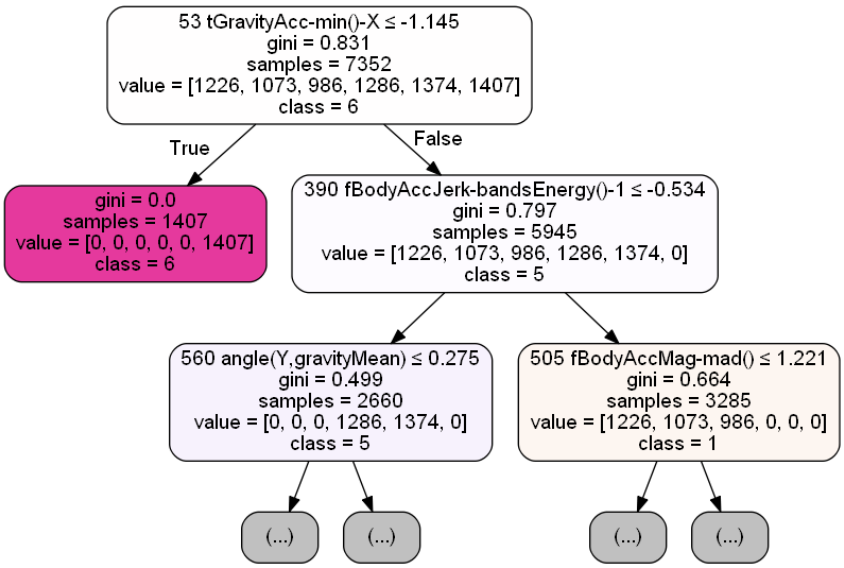


Figure 2: Decision tree

Figure 2 provides a representation of the first two layers of the decision tree built on the training set. In particular, after only one split, the classifier was able to perfectly distinguish class 6 (laying) from all other activities, as highlighted by the perfectly pure pink node in the image. At the second split, it separated classes 1, 2 and 3 (the various types of walking) from classes 4 and 5 (sitting and standing). Since the recognition of class 6 appeared to be trivial, as well as the division between the other two groups of classes, we decided to narrow down our classification tasks. For the remainder of this chapter we will therefore focus on a binary classification task involving classes 4 and 5, whereas classes 1, 2 and 3 will be considered in the chapter dedicated to advanced classification techniques. By considering only the sitting and standing activities, we are left with 2,660 training records and 1,023 test ones. Class 5 records represent 52% of the points, thus every classifier in this chapter should at least perform better than a trivial classifier (always labelling points as class 5) which would have an accuracy of 0.52 in order to be of any use.

2.1 Decision Tree and KNN

After selecting only records of classes 4 and 5, we re-run the KNN and decision tree classifiers and re-tuned the parametres. For the KNN, after testing all values from 1 to 15 in the same way as before, we found the best configuration resulted from choosing 14 neighbours and weighing the points uniformely. For the decision tree, we used Gini as impurity measure, a maximum depth of 5, and a minimum of 90 points per leaf. These latter parametres were found after running a 10-fold grid search cross validation on all possible depths from 5 to 10 and minimum number of nodes per leaf from 5 to 100 (going in increments of 5). The performances of the two classifiers on the test set of the restricted dataset are summarised in Table 1.

Overall, we can observe quite good results with both techniques, with a 0.84 accuracy for the decision tree and 0.91 for the KNN. In particular, we can observe higher F1 scores (0.82 for class 4, 0.85 for class 5) compared to the situation where we considered the whole dataset (0.81 for class 4, 0.80 for class 5).

Notably, in the decision tree only few (9 out of 561) variables have a feature importance greater than 0. This may signal the presence of redundant or highly correlated variables, in addition to the possibility that there are highly discriminat variables in the dataset. Dimensionality reduction will be covered in a later section and will address such concerns. By looking

at the feature importance returned by the decision tree, the most relevant attribute appears to be "angle(Y,gravityMean)", whose kernel density estimation is plotted in Figure 3. As easily observable, the two classes have a relatively moderate overlap in the central part; class 5 (standing) presents a more concentrated distribution around higher values, whereas class 4 (sitting) has longer tails both to the right and especially to the left, with a less defined peak. Overall, this variable already provides a good separation between the two attributes, which helps explain the nice performance of the estimator.

The analysis up to this point did not consider the presence of anomalies in the dataset, which may impact the results of classification. The effects of outliers and their treatment will be topic of the next section.

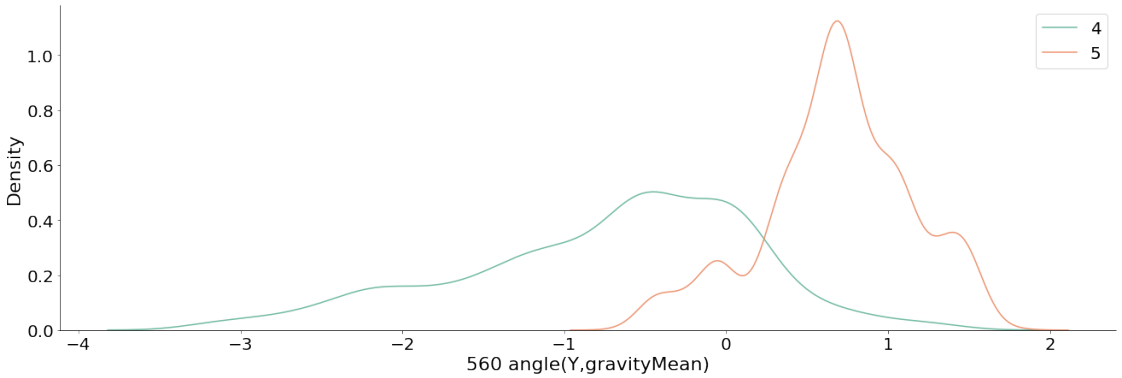


Figure 3: Kernel density estimation of the most important variable

Table 1: Performance of the basic decision tree and KNN classifiers (on test set)

	Decision tree	**KNN**
Accuracy	0.84	0.91
Precision	0.84	0.91
Recall	0.84	0.91
F1	0.84	0.91

2.2 Outlier Detection

As far as the identification of the outliers is concerned, we agreed to apply three different methods, coming from different approaches, in order to be able to spot the top 1% of them. Specifically, we used the "Local Outlier Factor" method which is density-based, the "Isolation Forest" method which is model-based and the "ABOD" method which is angle-based.

Once that all the techniques that have been presented were tuned precisely to our needs (selecting a contamination of 0.01), we were able to determine the top 1% outliers. Subsequently, we decided to identify those common points that were marked as outliers in all the methods that we applied (4 points in total), and then we deleted them from our dataset. Figure 4 depicts a scatter plot of two of the most important variables according to the decision tree feature importance, with the common outlier points marked in red. It is curious to note how they present rather average values for these two variables, implying that their status of outliers may depend on the other features. Thereafter we opted to perform again the classification methods that have been discussed in the previous section, to check if by removing the outliers, the classification performance would have increased or not. Unfortunately, we did not have get any significant improvement for both the decision tree and the KNN, reporting the same scores as before (see Table 1). Considering that most likely deleting four common outliers were not

enough to enhance the classification performances, we agreed on running again all the outlier detection's methods, looking for the common 1% of shared points. Namely we wanted to obtain the first 27 points in common that were detected as outliers, from the "LOF", the "ABOD" and the "isolation Forest" methods. Once again, reapplying both the decision tree and the KNN on a dataset without those common 27 points, did not allow us to gain any further improvement in terms of performances. This behavior could be justified, since even without the elimination of the outliers from the dataset, the general performances of the methods were still pretty good.

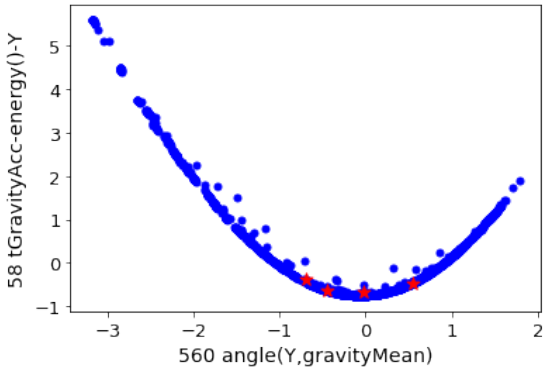


Figure 4: Scatter plot of two most important variables, with outliers highlighted

2.3 Unbalanced Learning

In this section we decided to increase the challenge creating an unbalanced version of our dataset, upon which we applied different techniques of imbalanced learning. The first step was to create a "base" unbalanced variant, which was generated through the utilisation of a random under sampler.

Table 2: Performance of the decision tree and KNN classifiers after different imbalanced learning methods (on test set)

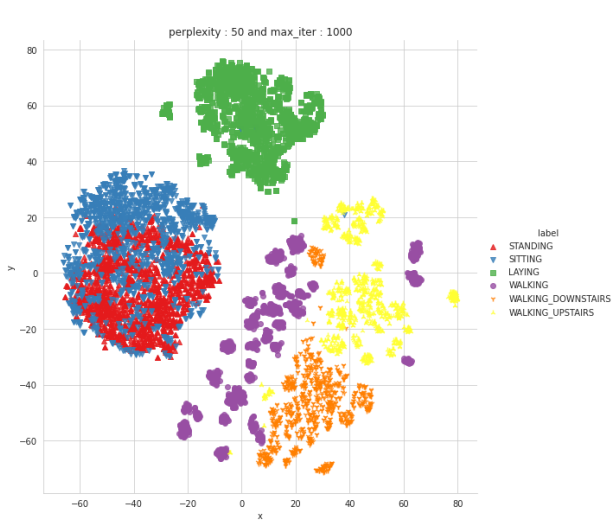
	Unbalanced dataset		**CNN**		**ADASYN**	
	DT	**KNN**	**DT**	**KNN**	**DT**	**KNN**
Accuracy	0.73	0.70	0.75	0.77	0.82	0.82
Precision	0.79	0.79	0.80	0.77	0.83	0.82
Recall	0.72	0.68	0.74	0.76	0.82	0.82
F1	0.71	0.66	0.73	0.76	0.82	0.82

The new dataset was composed by 57 instances of class 4 and 1374 instances of class 5 (implying a 4 to 96 unbalancing). Starting from this new version of the dataset we performed on it two different techniques, respectively one which was targeted to the under sampling of the majority class thanks to the Condensed Nearest Neighbors approach, while the second one was aimed to the over sampling of the minority class through the ADASYN method. As a result, we obtained two versions of the previous unbalanced version of the dataset. In the first case (CNN) we got a new dataset composed by 57 instances of class 4 and 134 of class 5 (we settled for a 1 to 2 rebalancing), while in the second case (ADASYN) we achieved a variation formed by 1382 instances of class 4 and 1374 of class 5 (an almost 1 to 1 rebalancing). The next step was to perform a decision tree and a KNN on all the three variants that we created, to be able to compare the performances of the methods. As we can see from Table 2, the best dataset in term of scores for both the DT and the KNN was the one where we applied the ADASYN approach. In fact, the latter was also the one where the performances were the most similar to

the results obtained in the original dataset, although still lower. Another important thing that must be noticed is the fact that the worst scores that we got appeared to be the ones which were performed on the dataset derived from the random under sampler. This trend could have been predicted due to the lack of instances of class 4 in that specific dataset which caused some values, such as the recall, to be low for both the classification method. Furthermore, the advantage in performance that the KNN had in the balance setting is now lost, as we have a smaller gap between the two.

2.4 Dimensionality Reduction

As far as dimensionality reduction is concerned, we decided to start by using t-SNE, a non-linear, unsupervised method to represent points from a large space, as in our case, to a two-dimensional space, while preserving local structures. Our goal was to understand how the t-SNE was able to represent the data according to the classes it belongs to, and whether there was a good separation and distinction. For this reason, considering that the classification data already hinted at a possible observation overlap between standing and sitting, we tried to run the algorithm in both the whole (Figure 5a) and the reduced dataset (Figure 5b).



(a) On whole dataset

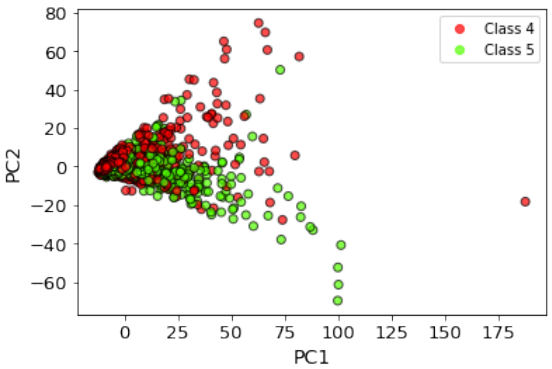
(b) On classes 4 and 5

Figure 5: t-SNE visualisation

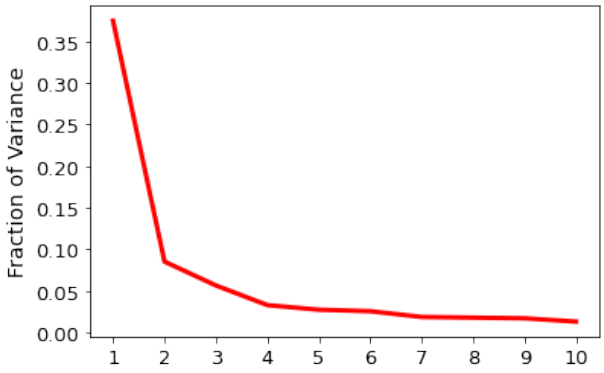
As suggested by the classification models, in Figure 5 unlike the other classes, which turn out to be quite distinguishable and separated in their clusters (which is why as pointed out at the beginning we decided to focus only on standing and sitting), for the latter two there is an almost complete overlap. We therefore tried to run t-SNE only in the reduced dataset, with the results shown in Figure 5b. Although we have iterated the t-SNE for different values of perplexity, the result is almost always the same, an overlapping situation. Since perplexity is the probability that models the way t-SNE sets the actual number of neighbours each point is attracted to, we have seen that for both cases the value of 50 was the best. Too low a value will lead to small disconnected clusters, while larger values will lead to circular clusters with roughly equidistant points. This result with these parameters set shows us that the data for these classes have similar values in terms of observations and features, both in multidimensional space and also in reduced space, which is why it is difficult to classify them. As a last procedure we tried to check if the dataset that the t-SNE generated with 2 dimensions could improve the classification. As far as the test classification is concerned, as we can see in figure 3 the

performances are much worse with an accuracy that went from 0.84 to 0.57, only 0.05 better than the trivial classifier. For the KNN instead the accuracy decreased from 0.91 to 0.59. In the prediction of the training set, the models performed sensibly better: we may note it as an overfitting phenomenon. Considering these observations for both classifiers we can guess that the reduced dimension space has reduced the prediction in the test and we would not use t-SNE as a suitable reduction method to improve classification performance.

We then performed a principal component analysis. Considering a number of components equal to 2, the performance of the classifiers greatly decreases again, especially for the KNN, which fares even worse than the decision tree (see Table 3). A possible reason for such drop in performance could be the low fraction of variability explained by the components considered (0.375 for the first, 0.085 for the second - Figure 6b). As we could see from the graph in Figure 6a, some points really stand out from the others, being very detatched from the the mass of points. We reran the same outlier detection methods we previously discussed. These five points were deemed as outliers by at least 2 methods out of 3 (two points were classified as outliers by all methods). PCA allowed us to finally have a graphical representation were outliers are easily detectable visually. In addition, the three methods recognised two common outliers among all of them, which were also deemed outliers before applying the dimensionality reduction techniques. Thus, we have a strong case for labelling these two records as anomalies.



(a) Scatter plot of first two principal components



(b) Fraction of explained variance by each principal component

Figure 6: Principal component analysis

Another dimensionality reduction method we wanted to try is univariate feature selection. Using ANOVA, or analysis of variance, we first selected the five best features for dimensionality reduction using the Fisher test. This time, the result of the classification brought an increase in performance compared to the original situation, with an accuracy in the test set of 0.85 instead of 0.84. We can say that with only 5 features chosen by the analysis of variance, our decision tree classifier has improved. We had however mixed results, as the KNN performance decreased.

Table 3: Performance of the decision tree and KNN classifiers after different dimensionality reduction methods (on test set)

	t-SNE		**PCA**		**ANOVA**	
	DT	**KNN**	**DT**	**KNN**	**DT**	**KNN**
Accuracy	0.57	0.59	0.65	0.57	0.85	0.84
Precision	0.58	0.59	0.65	0.57	0.85	0.84
Recall	0.57	0.58	0.65	0.57	0.85	0.84
F1	0.57	0.58	0.65	0.57	0.85	0.84

2.5 Final discussion

We started by considering a version of the dataset restricted only to classes 4 and 5 to get a more challenging and focused task. The basic decision tree and KNN classifiers were already pretty decent, much better than the trivial classifier, due to the presence of highly discriminant features. The outlier detection methods identified unanimously only a few outliers, which we have then been able to also recognise graphically when applying dimensionality reduction techniques such as PCA. The latter proved useful for several interesting visualisations, but they did not generally provide improvements in the classification, except for the ANOVA method.

3 Advanced Classification

This second chapter is dedicated to the application of several more advanced classification techniques. For this purpose, a multiclass task has been selected. In particular, we considered records of classes 1, 2 and 3 (three types of walking activities), and then standardised them. The rationale behind this choice of task is that different types of walking activities could be harder to recognise between themselves, rather than compared to other activities. We have a total of 3,285 training records and 1,387 test ones. The distribution of classes does not present any significant unbalancing, with 1,722, 1,544 and 1,406 records respectively.

In order to set a reference point for more advanced classification methods, we tuned (in the same way we described before) and ran the decision tree and KNN classifiers. The performances of the two models are summarised in Table 4, and they do not numerically differ from what we already saw considering only classes 4 and 5. Notably, the decision tree reported sensibly better overall performance on class 1 (with an F1 score of 0.89) compared to the other two classes (F1 score of 0.79 and 0.80 respectively). The KNN instead returned more homogenous results (F1 scores of 0.91, 0.92 and 0.88 respectively). Ideally, we would like methods presented in this chapter to not perform worse than the two presented baseline models.

Table 4: Performance of the basic decision tree and KNN classifiers (on test set)

	Decision tree	**Decision Tree**
Accuracy	0.83	0.91
Precision	0.83	0.91
Recall	0.83	0.91
F1	0.83	0.91
AUC	0.9029	0.9758

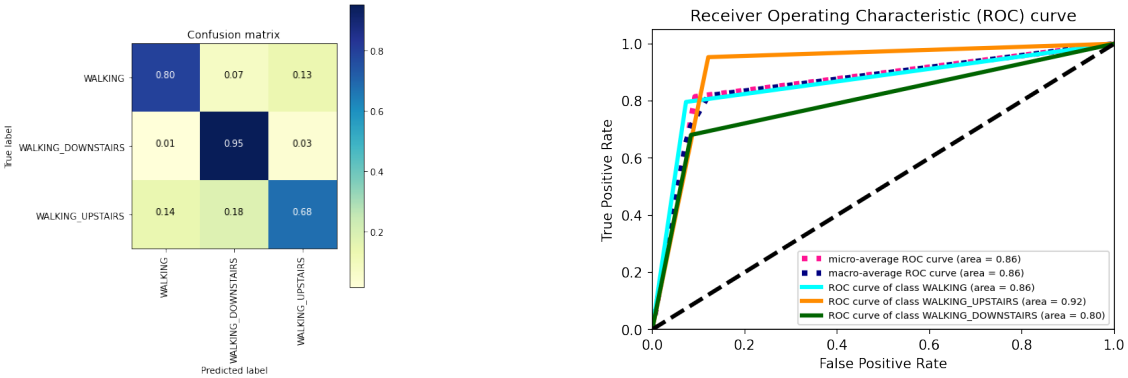
3.1 Naive Bayes Classifier

The Naive Bayes classifier is a probabilistic machine learning model based on Bayes' theorem. It is called "Naive" because the various features of the data are considered independently one of each other, and it does not consider the correlation between the characteristics of the instance. It calculates the probability of each label for a given object by looking at its attributes. Then, it chooses the label with the highest probability. Gaussian Naive Bayes is a variant of Naive Bayes that assumes data follows the Gaussian distribution and it supports continuous data. Furthermore, features are assumed to be uncorrelated with each other. This model can be fit by simply finding the mean and standard deviation of the points within each label, which is all what is needed to define such a distribution.

As the only parametre for cross validation we used variance smoothing, i.e. the portion of the largest variance of all attributes that is added to the variances to ensure more stable calculations.

After carrying out a 5-fold cross validation performing a grid search, we tested the model on the test set. We get an overall accuracy of 0.81, but looking at the accuracies of individual classes we can observe some more interesting results. As we can notice from Figure 7a which shows the confusion matrix, the classifier predicts with a respective accuracy of 0.83 and 0.87 the classes 1 and 2, namely 'walking' and 'walking upstairs' while the class 'walking downstairs' with an accuracy of 0.73, which worsens the overall result. From Figure 7b, we can observe how the general area under the curve is 0.86. Trying also the categorical version of the Naive Bayes, we get similar but slightly inferior results.

Naive Bayes is based on the often erroneous assumption of independent characteristics. In this dataset we hardly find independent characteristics among the signals (many of them have high correlation coefficients between them). This could affect the classification and as we will see we will get much better results with other classifiers that require less assumptions and/or are more robust to this issue.



(a) Confusion matrix (b) ROC curve

Figure 7: Some visualisations of the Gaussian Naive Bayes classifier

Table 5: Performance of the Naive Bayes classifiers (on test set)

	Gaussian NB	**Categorical NB**
Accuracy	0.81	0.78
Precision	0.81	0.79
Recall	0.81	0.78
F1	0.81	0.78
AUC	0.8562	0.8336

3.2 Logistic Regression

Logistic regression is a statistical model that is extensively used for binary classification, but it can be generalised also for multiclass classification. In our case, we used the scikit-learn library logistic regression because it allowed us to approach our situation as a multiclass problem thanks to the "one over the rest" scheme. As for the previous models we used the same behaviour, namely we performed a grid search to get a clearer view of the parameters which could perfectly fit our case. An important thing that must be noticed, is the fact that in this case (like many other classifiers, e.g. the SVC) we found the parametre "C", which indicates the penalisation

to apply for misclassified training instances. Thanks to the utilisation of the grid search, the model has been tuned through a 10-fold cross validation, which allowed us to avoid any potential overfitting behaviour. The general performances of the model on the training set were perfect (with an accuracy equal to 1). Almost the same situation happened for the test set, where the accuracy resulted in 0.98. In fact, as we can see from the Table 6, even the performance on the test set were significantly high.

Furthermore, just to have a double check of the great result of this model, we decided to also compute the AUC score, which resulted in 0.9989, a further confirmation of the high performance of the model.

Table 6: Performance of logistic regression classifiers (on test set)

	Logistic regression
Accuracy	0.98
Precision	0.98
Recall	0.98
F1	0.98
AUC	0.9989

3.3 Support Vector Machines

Support vector machines are a technique which tries to depict the decision boundaries of a set of records by selecting a particular subset of them, called support vectors. We are basically looking for the hyperplane which better separates the data into the various classes: in this situation the support vectors represent the points closest to such boundary. We experimented with both the linear and non-linear SVM (using as kernel the radial basis function). As easily understandable, our problem is non-separable, thus we will have to pay attention to the parametre C indicating the penalisation to apply for misclassified training instances. This regularisation parametre is also very important to avoid overfitting and assuring a solid test performance (although, being a user-specified parametre it is still susceptible to possible faults on our behalf).

Both models have been tuned through a 5-fold cross validation using a grid search for the value of C (in both models) and the type of kernel to use (for the non-linear SVM). The results of the two models on the test set are summarised in Table 7. Both present stronger performances than the baseline models, with the linear classifier performing slightly better. No significant differences emerge among the three classes.

Figure 8 provides a graphical representation of the support vectors extracted from the non-linear SVM (denoted by a black circle) on the scatter plot of the first two principal components extracted from the dataset through PCA. Although not perfectly identifiable, we can get an idea of the boundary separating the classes into three more homogenous areas.

Table 7: Performance of the support vector machine classifiers (on test set)

	Linear SVM	**Non-linear SVM**
Accuracy	0.98	0.96
Precision	0.98	0.96
Recall	0.98	0.96
F1	0.98	0.96
AUC	NA	0.9962

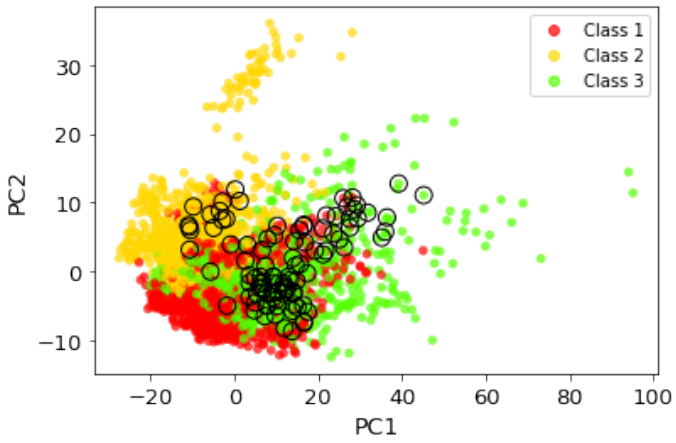


Figure 8: Scatter plot of first two principal components, with support vectors highlighted

3.4 Neural Networks

For what it concerns the development of a neural network, we opted for the utilisation of the MLP classifier library, which is released by scikit-learn. In general terms a multi-layer perceptron (MLP) is a supervised learning algorithm which, given a set of features and a target variable, can learn a non-linear function approximator for either classification or regression. In a neural network, we are forced in some way to set a lot of parameters (i.e., hidden layers, alpha, learning rate, etc.) to achieve a good result, and for this reason we agreed on performing a grid search. The latter has been performed also using a 10-fold cross validation. As a result, we obtain that some of the best parameters were hidden layers equal to 128, 64 and 32, alpha (the L2 penalty parameter, needed to avoid overfitting) equal to 0.1 and as activation function 'tanh' (the hyperbolic tangent function). This suite of configuration led us to gain a good performance of the model. In fact, as we can see from Table8 the general accuracy on the test set is 0.98.

An important thing to notice is the fact that MLP classifier trains iteratively, and at each time step the partial derivatives of the loss function, with respect to the model measures, are computed to update the parameters. For the training of the neural network, when updating the weights, we used a minibatch strategy, with a batch size equal to 200, which, after some trials, represented a good compromise between performance and computational complexity.

In Figure 9 we can see the loss curve which gives us a snapshot of the training process and the direction in which the network learns. We can observe how the loss decreases very rapidly in the first few epochs (up to the fifth one), then starts decreasing very slowly until epoch 60, point from which the decrease becomes imperceptible.

Table 8: Performance of the Neural Network (on test set)

	Neural Network
Accuracy	0.98
Precision	0.98
Recall	0.98
F1	0.98
AUC	0.9990

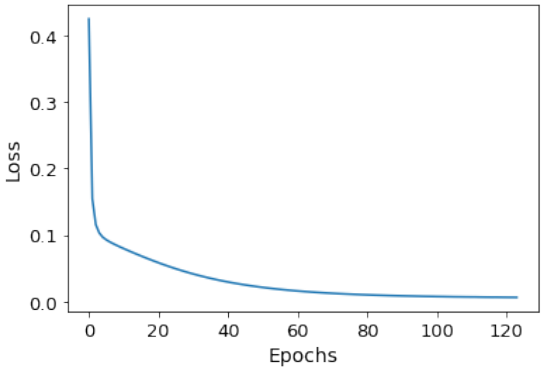


Figure 9: Visualisation of the features ranked by importance according to the random forest classifier

3.5 Random Forest

Random forest is an ensemble method which combines the predictions of several base estimators (in this case decision tree) to improve generalisability/robustness over a single estimator. Usually, we distinguish between two different families of ensemble models, namely the averaging methods and the boosting methods. The random forest belongs to the first category, because the driving principle is to build several estimators independently and then to average their predictions. An important aspect that must be noticed is the fact that in Random Forest, each tree in the ensemble is built from a sample, drawn with replacement (i.e., a bootstrap sample) from the training set. Furthermore, when splitting each node during the construction of a tree, the best split is found either from all input features or from a random subset of features, injecting randomness in the model. The general principle behind the utilisation of random forest is that by taking an average of the different predictions (coming from the different trees), some errors can cancel out, hence yielding to an overall better model. In our case our model has been tuned through a grid search, using a 3-fold cross validation. Some of the most important parametres that we got, resulted in number of estimators set to 400, no limit on the maximum tree depth and Gini as a criterion for the split. As we can see from Table 9 the overall performance on the test set was pretty good, leading to a general accuracy of 0.93. One observation that we can point out is the score regarding the class 3, which is the lowest for both precision (0.90 against 0.97 and 0.91) and recall (0.89 against 0.92 and 0.96), and consequently for F1 score.

In Figure 10 we can see a plot referring to the future importance of the model, which will be used in further analysis (simple and multiple linear regression). We can observe how many variables in the top 15 by feature importance refer to some concept of deviation (standard deviation for the variables ending in "std()" and mean absolute deviation for the ones ending in "mad()") or some autoregressive coefficients (variables containing "arCoeff()"). This suggests us that we can discriminate between the three types of walking by considering the variability of the signals and by how much they depend on past values of themselves.

Table 9: Performance of Random Forest classifiers (on test set)

	Random Forest
Accuracy	0.93
Precision	0.93
Recall	0.93
F1	0.93
AUC	0.9808

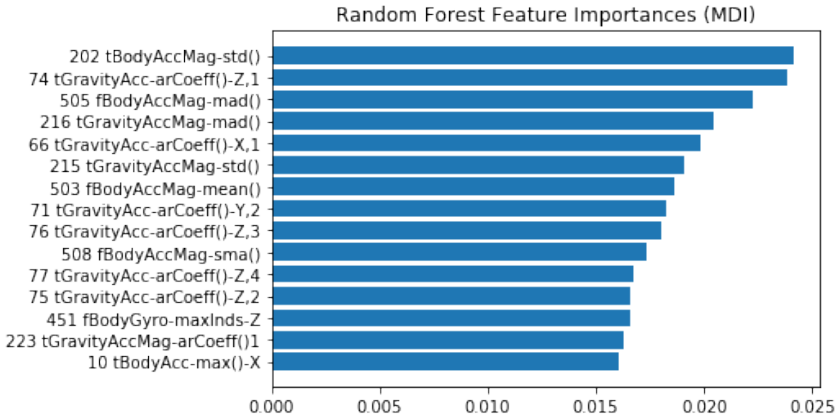


Figure 10: Visualisation of the features ranked by importance according to the random forest classifier

3.6 Gradient Boosting Machines

Another ensemble (this time of the boosting family) method we applied is the so called Gradient Boosting. This involves the use of many weak learners (in our case decision trees), which sequentially improve the results of classification (note that each tree is based on the results of the previous one). We tested 3 different versions of this method: the standard implementation by sklearn, the XGBoost variant, and the LightGBM variant. For each we performed a 3-fold cross validation using a grid search over several key parametres, such as the number of estimators, the regularisation parametre, the number of leaves per node and maximum number of tree levels. The regularisation and tree pruning parametres are crucial to avoid overfitting: by choosing a higher values for the first and lower values for the second we can build a more conservative but more reliable model. As far as the number of estimators is concerned, the higher it is, the better the performance will, although increasing the increase in accuracy becomes marginally smaller every weak estimator we add. As a confirmation of this, by looking at Table 10, we can observe how the classical method and XGBoost have basically the same performance, despite the latter having five times more estimators than the former (500 against 100), and thus taking a lot more time in model building. We can also note how LightGBM loses only one percentage point in accuracy compared to the other two methods, but it gains a huge advantage in terms of speed (in our trials, about 10 times faster in terms of wall time).

Compared to random forests (the other ensemble method we already discussed), gradient boosting machines perform slightly better in terms of accuracy and, considering the LightGBM method, they are less expensive in terms of time spent building the model.

Table 10: Performance of the gradient boosting machine classifiers (on test set)

	Classical GB	**XGBoost**	**LightGBM**
Accuracy	0.96	0.96	0.95
Precision	0.96	0.96	0.95
Recall	0.96	0.96	0.95
F1	0.96	0.96	0.95
AUC	0.9958	0.9953	0.9939

3.7 Linear Regression

For this section, we decided to tackle a regression problem. Since the dataset presents no obvious regression task we first needed to choose some variable as our objective. We considered as a target the feature '12 tBodyAcc-max()-Z'. This variable was chosen as to not be too correlated with other variables (its maximum correlation with another variable was less than 0.70). We first tried predicting it in a univariate simple regression setting and then in a univariate multiple regression one.

3.7.1 Simple Linear Regression

In simple linear regression, our aim is to accurately predict the value of a dependent variable (or regressand) given one independent variable (or regressor). To choose the regressor we looked at the feature importance generated by the random forest and chose one of the first variables ('215 tGravityAccMag-std()' in particular). We avoided autoregressive coefficients to avoid incomprehensible analyses. For each of the models used, we looked for the intercept, i.e. the value that y assumes when x = 0, the point at which the line cuts the y-axis, and then the angular coefficient of the line, which represents its slope and describes the change that Y undergoes when X increases. After finding these two values we tried to predict using the model found in the test set. Figure 11 provides a graphical representation of the regression line fitted to the data. To evaluate the performance of the model we used the R^2 score, which in relative terms represents the index of the goodness of fit of our regression. The higher the R^2, the better our model predicts. The errors in prediction were measured using both the mean squared error (MSE) and mean absolute error (MAE).

As visible from Table 11, in the case of linear regression with the simple model we have an R^2 of 0.177 with a quite high MSE of 0.823, this shows us that our model with these characteristics and unique variables cannot predict very well. This could have been however expected, since the regressand is not much correlated with any variable. Considering that we only used one regressor (a highly discriminant variable) the performance may not be considered too bad.

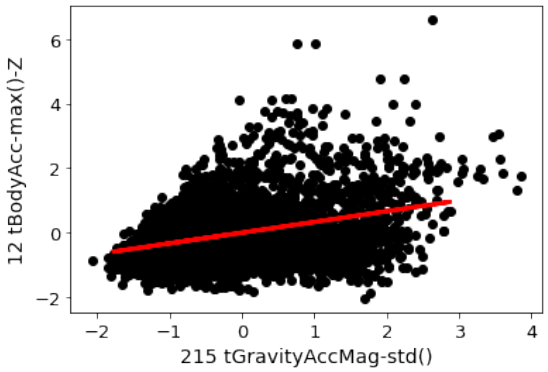


Figure 11: Scatter plot of the two considered variables for simple regression, with fitted regression line (in red)

Table 11: Performance of the linear regression models (on test set)

	Simple	**Multiple - Lasso**	**Multiple - Ridge**	**Multiple GB Regressor**
R^2	0.177	0.680	0.625	0.566
MSE	0.823	0.320	0.375	0.434
MAE	0.680	0.432	0.468	0.490

3.7.2 Multiple Linear Regression

Since our performance was exceptional in the simple regression, probably due to the fact that the predicted variable could depend on more variables than just the one considered, we extended the regression process using all the remaining variables excluding the target to see if the performance would improve.

By running the basic multiple linear regression, we got negative R^2, which is the result of the problem of multicollinearity. To solve this issue we need apply some regularisations, like those provided by the Lasso and Ridge regressions. Lasso takes a parametre α (which corresponds to the L1 penalty term) and performs variable selection, so that many coefficients will be set to zero, thus addressing the multicollinearity problem. Ridge regression uses an L2 penalty term instead, but does not perform variable selection. Another way to tackle the problem is through a Gradient Boosting Regressor, which uses the same approach seen in the Gradient Boosting Machines section. We performed a 10-fold cross validation with a grid search on the α parametre for the Lasso and Ridge regressors (settling for a final value of 0.003 and 0.5 respectively), while for the GB regressor we considered the learning rate and number of estimators (0.09 and 200 respectively). The number of estimators again proved useful to increase the performance of the regressor, although after a certain point the improvement became less and less tangible.

As visible from Table 11, the best result comes from the Lasso regressor, with an R^2 equal to 0.680 and the lowest mean errors. In particular, the Lasso regressor set 381 out of the 560 total regressor coefficients equal to zero, which helped reduce the effect of multicollinearity. The GB regressor did not prove to be any better than the traditional regressors. Overall, we can also state that by using many more variables, the performance greatly improves compared to the simple regression setting, where our fit produced an R^2 which was almost four times lower.

3.8 Final discussion

The use of advanced classifiers generally proved to improve the classification performance, reaching extremely high values, with a peak of 0.98 accuracy for several methods (neural network, logistic regression, linear SVM). The only method against this trend were the Naive Bayes classifiers.

After setting up all the models and collecting the results, we decided to try and see what would happen if we provided less data to the various models. Using a random undersampling, we considered only a quarter of the data and we observed only a very slight decrease in accuracy for all methods (from 0.01 to 0.03). In an even more restricting scenario, with only 5% of the data, we observed a greater drop in accuracy (up to 0.12), although some methods were able to perform almost at the same level. This shows the efficacy of the methods presented even in conditions of scarcer data, given also that we did not re-tune the parametres of the models.

In conclusion, we can say that the classification of the three types of walking resulted in very good results, thanks also to the presence of highly discriminant variables, which can effectively distinguish among the three classes. A useful insight provided by the random forest feature importance is that variables expressing deviations and autoregressive coefficients proved particularly useful for this point. This fact could reveal itself to be interesting and useful for the next chapter concerning the analysis of time series.

4 Time Series Analysis

The current chapter deals with time series and their analysis. Apart from the dataset we considered previously, we were also provided with 9 time series datasets (for each of the three measurements taken, a dataset for each axis was available) for such task. Following the results of the classification task (considering in particular the feature importance resulting from the random forest classifier), which highlighted many features of gravity acceleration connected to the X axis, we decided to analyse the dataset regarding total acceleration on the X plane (total_acc_x). We again have 7,352 training instances and 2,947 test ones, with each record representing a time series with 128 timestamps and being associated to a specific class label indicating the activity (as before). For this new task, all classes have been considered and kept. The time series were already scaled to a normal range: we tried to apply a mean-variance transformation, but this had no impact on the performance of the various models presented hereinafter.

4.1 Basic classification

We begin our analysis by trying to understand how far the classification in the previous section can differ or improve by using the original time series, from which the features dataset was extracted. Our aim will therefore be to see if through the time series the algorithms can identify factors that can help to better identify and classify different types of activities.

For this first part, in addition to working on the raw time series, we used Symbolic Aggregate Approximation (SAX) as an approximation method for our analysis. This was helpful to us both computationally and also to reduce the complexity of our time series, making them easier to interpret. In particular, through the SAX, which performs a transformation of data regions into symbols using the PAA coefficients, we have the assurance that every symbol will have the same probability of appearing in the approximated time series dataset, while managing to lose a minimal amount of information. We can see an example in Figure 12; in this case we chose a combination of 16 PAA segments and 8 SAX symbols. From the graph, we can see that in this time series, considering the different variations through the SAX, the size is reduced through the representation of 4 symbols: the first in the region of value 5, the second for value 7, and so on.

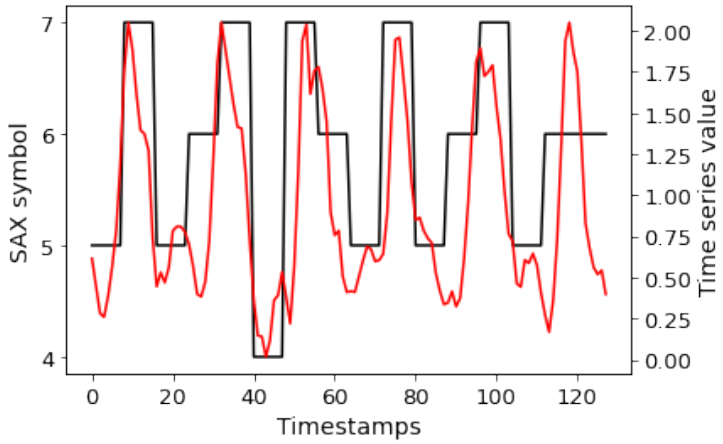


Figure 12: Selected time series (red) with SAX approximation superimposed (black)

Starting with the classifications, we used the K-Nearest Neighbours classifier first on the original series using both the Manhattan and Euclidean distance. As we have found out, in both cases, but as we have already pointed out in the previous chapters, class 6 (laying) is detected

perfectly by the classifiers, and as we have already seen it has more difficulty in detecting the activity of sitting (class 4), and standing (class 5), in which there is little motion on the X plane. This is to be expected as we were analysing the total acceleration on such axis. The end result is that the Manhattan distance has one more accuracy point than the Euclidean one. The results of all classifiers are summarised in Table 12.

Still in the raw time series for the KNN classifier we tried using Dynamic Time Warping (DTW) as a method for calculating ideally better distances. We do in fact get slightly better results, with an accuracy of 0.79, although at a much higher cost in terms of time complexity (despite having used the Sakoe-Chiba band).

Then we run the classification on the SAX-transformed dataset, using both a decision tree and the KNN classifier (exploiting DTW). Due to the approximation, the KNN model lost some considerable performance going down to 0.68 accuracy, while the decision tree benefited and performed better than before. The drop in KNN performance is explained by the terrible recall it had on class 5 (barely 0.01): it looks as if the approximation through SAX made it easier to recognise class 4 (whose recall in particular jumped from 0.51 to 0.99) at the expense of class 5.

All these classifiers presented perfect performance on class 6, but struggled particularly with class 4 in the raw series (which had good precision but terrible recall) or class 5 in the SAX transformed series.

Table 12: Performance of basic time series classifiers (on test set)

	DT raw	**DT** SAX	**KNN** raw Manhattan	**KNN** raw Euclidean	**KNN** raw DTW	**KNN** SAX DTW
Accuracy	0.67	0.72	0.78	0.77	0.79	0.68
Precision	0.67	0.75	0.79	0.78	0.79	0.71
Recall	0.67	0.72	0.78	0.77	0.79	0.68
F1	0.67	0.69	0.78	0.77	0.79	0.62

4.2 Shapelet and advanced classification

After having performed the basic classification methods, we then moved on to shapelet extraction and tried to exploit this technique for classification purposes. We later explored other techniques, such as convolutional neural networks and MINIROCKET. Structure-based classifiers are not considered in this section, as they would require a dataset of extracted features upon which to perform the classification task, which is essentially what we have already done in the previous chapter. Hence, such analysis will not be repeated again.

Shapelets are particular subsequences in a dataset, whose aim is to represent in the best way as possible a particular class. Once we find such representatives, we can use them as a tool for classification. We extracted shapelets using the ShapeletModel provided by the tslearn library, looking for one shapelet for each of the six classes, each having a length of 1/8 of the time series. We used the same parameters as those proposed by the authors of the model, using a total of 1,000 iterations. After fitting the model on the training set and trying to predict the test set, we obtained some interesting results, which are reported in greater detail in Table 13.

The overall accuracy of 0.78 is already on par with the basic classifiers we discussed in the previous section. Unlike in Chapter 3, we have however a lot of variability in the single class measures, so we deemed it appropriate to report them in full, as they are an exemplification of the other performances we obtained with other future classifiers in this section, whose detailed

Table 13: Performance of the shapelet model (on test set)

Class	Precision	Recall	F1 score	Accuracy
1	0.68	0.99	0.81	-
2	0.92	0.57	0.71	-
3	0.95	0.85	0.90	-
4	0.71	0.38	0.49	-
5	0.60	0.86	0.71	-
6	1.00	1.00	1.00	-
Average	0.81	0.78	0.77	0.78

performance will be omitted. Once again class 6 demonstrates the best performance, being completely isolated from the others. We can also observe some rather huge precision-recall discrepancies, particularly for classes 1, 2, 4, the first one having a very high recall, while the latter have relatively small recall values. Class 4 in particular seems quite hard to predict, with an F1 score of only 0.49, much lower than all of the others. Having extracted the shapelets, we now also had a dataset of the distances of each time series from each shapelet. This allows us to use the decision tree and KNN classifiers on such representation. For the KNN model, we again used both the Euclidean distance and the DTW. The tuning of these models followed an analogous process ad described in Chapter 3. Results are summarised in Table 14.

Table 14: Performance of the decision tree and KNN classifiers on shapelets (on test set)

	DT	KNN (Euclidean)	KNN (DTW)
Accuracy	0.77	0.78	0.80
Precision	0.79	0.79	0.81
Recall	0.77	0.78	0.80
F1	0.76	0.77	0.79

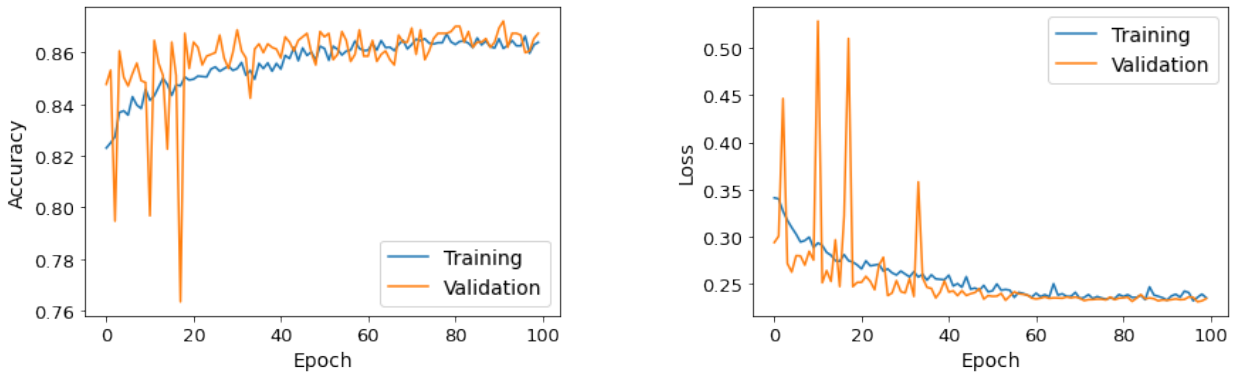
The three model present very similar performances, in line with the original shapelet classifier. The KNN classifier using DTW distances performed the best with an accuracy of 0.80. All of these models presented noticeable precision-recall discrepancies, especially for class 4, which continued to have very low recall (the situation for the other classes slightly improved instead).

Moving towards more advanced classifiers, we tried to implement a simple convolutional neural network, a type of network which is very popular for image analysis but has also many successful applications on time series. We used three hidden layers using a 'relu' activation function and an output layers using the 'sigmoid' function. We used a minibatch strategy, updating the weights after each 10% of the records, and we trained the network for a total of 100 epochs, keeping 20% of the data as a validation set every time. The model was trained on the original data, so the shapelets extracted previously have not been considered.

Table 15 reports a brief summary of the performance of such network, which outperforms the classifiers we have seen so far. Notably, the performance on classes 1, 2 and 3 improved significantly, whereas we did not see any improvement on classes 4 and 5, which remain the hardest two to predict (which gives more meaning to the choice of the task adopted in Chapter 2).

Figure 13 depicts the accuracy and loss curves for the convolutional neural network for the 100 epochs of training, highlighting the training performance versus the validation one. We can see how both measure improve visibly upon reaching a plateau around epoch 50. What is also interesting to note is the presence of some huge spikes in both graph during the first 40 or so epochs. This could be explained by the use of data shuffling at the beginning of each epoch, a

strategy used to avoid excessive overfitting of the model and help to converge faster. Another possible explanation regards the use of the dropout strategy, which randomly disconnects some neurons in between layers. We can also observe that validation performance is sometimes better than training one. Although it may look odd, it is possible due to the regularisation parameters being applied only on training (which increases the loss, aiming to reduce overfitting) and the fact that training loss is computed during the whole epoch (this is relevant since weights change during the epoch thanks to the minibatch strategy) while the validation one is computed only at the end of each epoch. After epoch 40 we can observe a neat convergence between the two measures in both graphs.



(a) Accuracy history (b) Loss history

Figure 13: Some visualisations of the convolutional neural network

As a final model for this task, we report in Table 15 the results of a ridge classifier (with built-in cross validation) executed on a version of the dataset transformed with MINIROCKET. This model revealed itself to be the best one of all we have test, with an accuracy of 0.85, just slightly above that of the convolutional neural network (the detailed results are almost identical, with the same patterns we discussed before repeating themselves). Most notably, the MINIROCKET tranformation performed incredibly also in terms of time complexity, being faster than the neural network and its original version (ROCKET), which we have tested and from which we obtained worse results (accuracy 0.73) despite taking more than twice the time. We also tried to run a Canonical Interval Forest classifier, but the performance was quite poor (worse than the basic KNN model). Hence its results are not reported, as for the ROCKET.

Table 15: Performance of the convolutional neural network on the raw series and ridge classifier on MINIROCKET transformation (on test set)

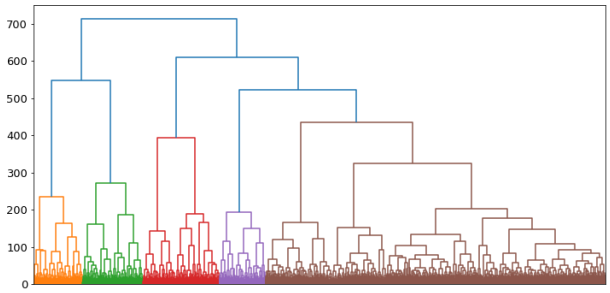
	CNN	**MINIROCKET**
Accuracy	0.84	0.85
Precision	0.86	0.85
Recall	0.84	0.85
F1	0.83	0.85

4.3 Clustering

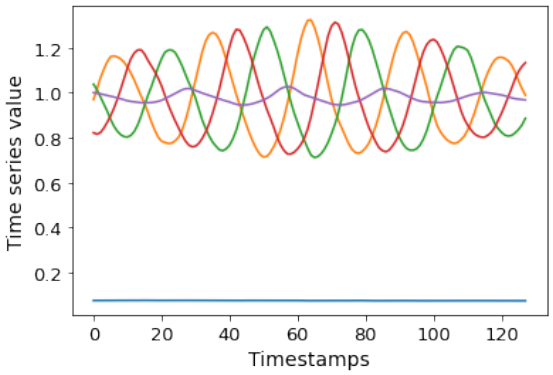
In order to get a better understanding of the data which we were manipulating, we decided to investigate some more knowledge through the utilization of clustering techniques. Our strategy has been founded on the goal of applying just a few clustering algorithms, but with different versions of the dataset in input, to be able to compare and interpret the different results

obtained. As we will discuss later on this section, we applied different approximations to the original dataset, to better comprehend how the alteration of the values and the dimensions of the input dataset, would have affected or not the distribution of the instances of a specific type of activity, within the clusters themselves.

Our approach started with the exploitation of a hierarchical clustering algorithm, which allowed us to gain some spendable insights on the data, usable also for the further analysis. We first applied the algorithm to the original dataset, using different linkage measures. After some different attempts we proceeded using the Ward's method also due to the fact that, knowing that we had in mind of using also K-means as a second clustering algorithm, as we recall from the DM1 project, the interpretation of a dendrogram resulting from the utilization of Ward's method, could be a good starting point to set the number of centroids for K-means (due to the similar approach both methods have with the SSE). As it can be seen in Figure 14a it seemed quite clear to us how the algorithm distinguished five different clusters (they can be seen through the different usage of colours).



(a) Hierarchical clustering dendrogram



(b) K-Means cluster centroids

Figure 14: Some visualisations for clustering

As we have just expressed before this first step helped us to set up the second clustering algorithms that we used, namely K-means (we used as metric the Euclidean distance). In order to have a second check on the choice of using five as number of clusters, we decided also to compute the "Silhouette Score" and the SSE for various values of K, to apply the "Elbow Method". The results that we obtained with those two methods seemed quite coherent with our decision so far, so we decided to proceed. The different dataset that we processed have been: the original dataset, a dataset on which we applied the SAX approximation and the one where we applied the PAA approximation. As we can see in Table 16 we got some interesting results. First of all as it can be seen in Figure 14b, from the representation of the centroids it can be deducted that we have a completely detached cluster located at the bottom of the graph, three clusters that share the same trend and a cluster which is placed among them. If we observe the distribution of the instances belonging to the same activity within those clusters, we can see from Table 16 how we got a cluster basically composed by just activity of "laying" (cluster 5), which we can say to be the detached cluster; three clusters which share similar distribution of activities such as walking, walking upstairs and walking downstairs, which we assume to be the ones with a similar trend; and one cluster which contains all the instances of sitting and standing plus some instances of the other activities, which we deduce to be the cluster with the centroid located among the other three. We found this result consistent with what we have already stated, namely the fact that even the classifiers had some problems to predict the instances representing activity 4 and 5 (sitting and standing) probably due to a kind of similarity that these signal have in common. Subsequently we proceed applying the

K-means algorithm also to the SAX and PAA approximated dataset. Also in this case the results that we obtained, they were coherent with what we got before, apart from the fact that in both of the approximated cases we achieved one more "trivial" cluster, containing basically just instances of activity laying. In fact, in both cases we had two clusters composed mostly by instances of activity laying and three more "mixed" clusters, but always with one of those containing by far the majority of instances of type sitting, and always with one containing all the instances of type standing.

We also tried to retrieve the clustering in the feature dataset analysed in the previous chapter, to see what features characterised each cluster, by looking at the centroids. The detached cluster and the low variability one presented significantly different values for the variables 'tGravityAcc-mean()-X' and 'tGravityAcc-max()-X', which are features extracted from the time series dataset we analysed. The three clusters with similar trend where instead differentiated by the variables 'tBodyAccJerk-mean()-X' and 'angle(tBodyAccMean,gravity)', which instead refer to the body acceleration, which was contained in a time series dataset we did not analyse, and this is why we were not able to fully detach these clusters.

Table 16: Composition of clusters returned by K-Means on raw dataset

	Activity 1	**Activity 2**	**Activity 3**	**Activity 4**	**Activity 5**	**Activity 6**
Cluster 1	305	342	240	1271	1374	/
Cluster 2	309	230	235	/	/	/
Cluster 3	314	244	249	/	/	/
Cluster 4	298	257	262	/	/	/
Cluster 5	/	/	/	15	/	1407

4.4 Motif/anomaly discovery

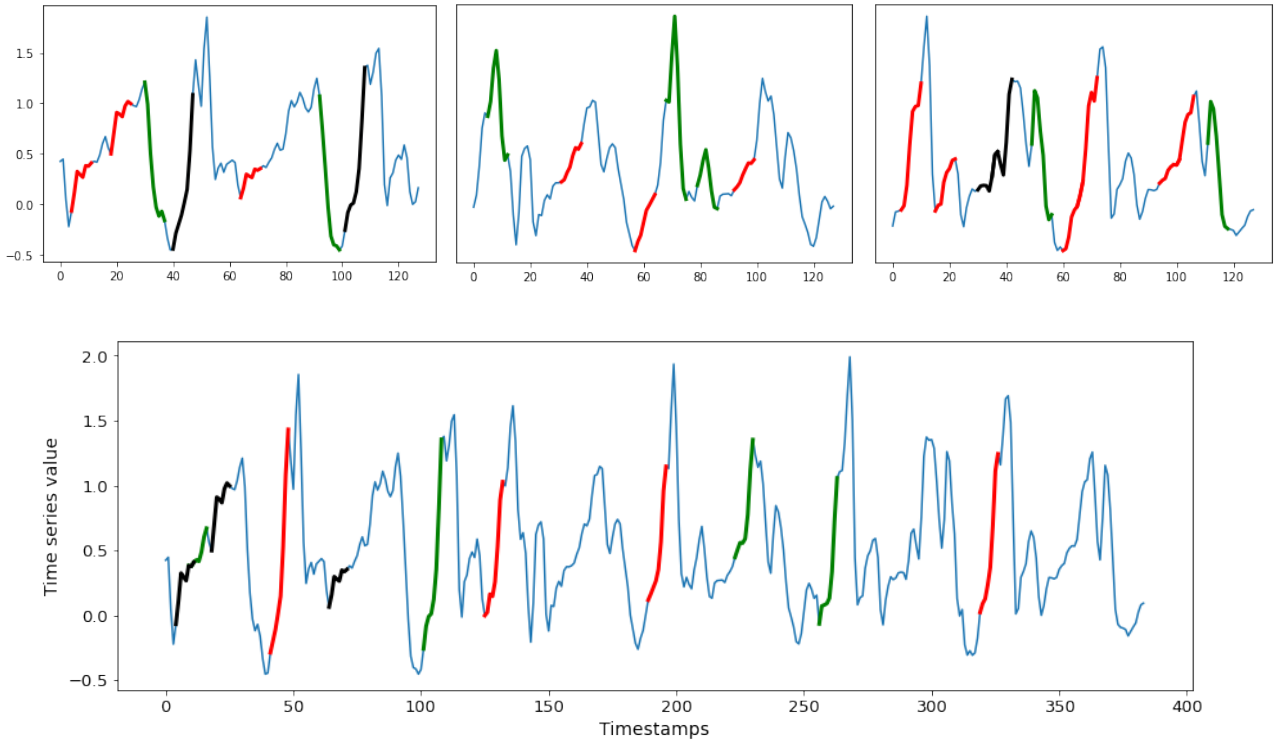


Figure 15: Motifs extracted on selected time series and their union

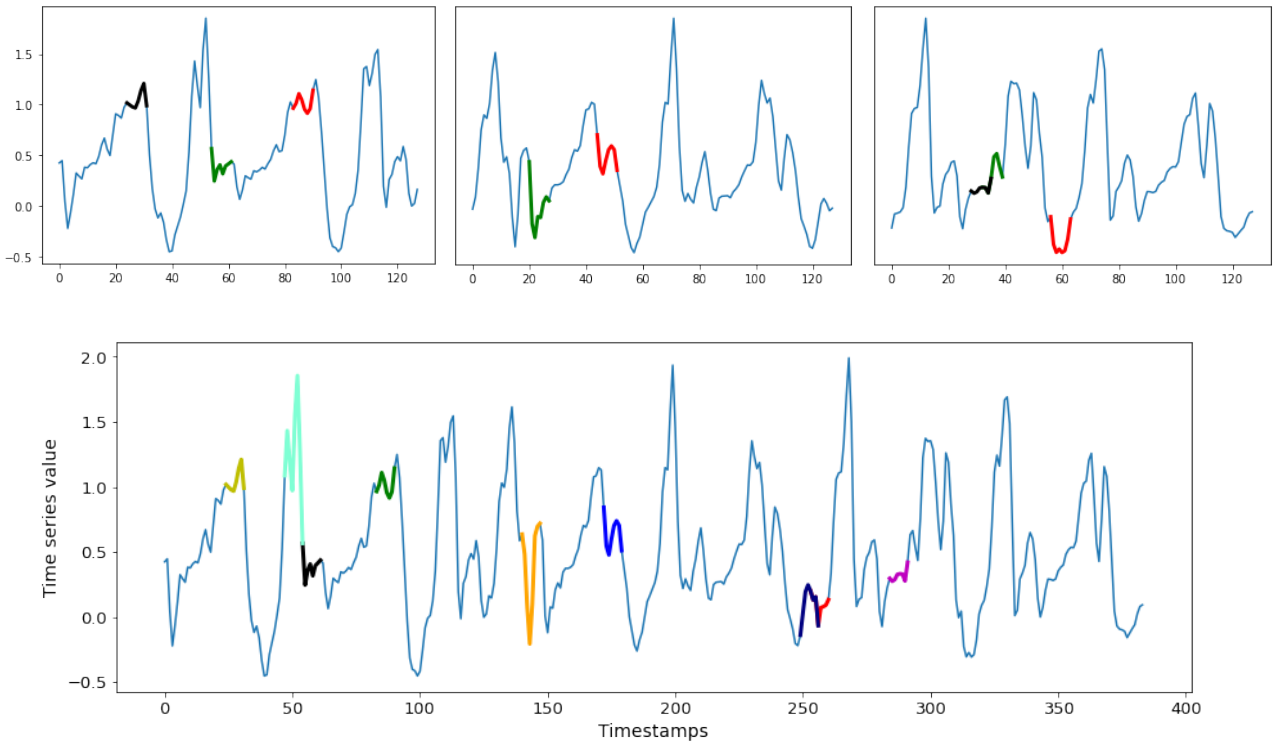


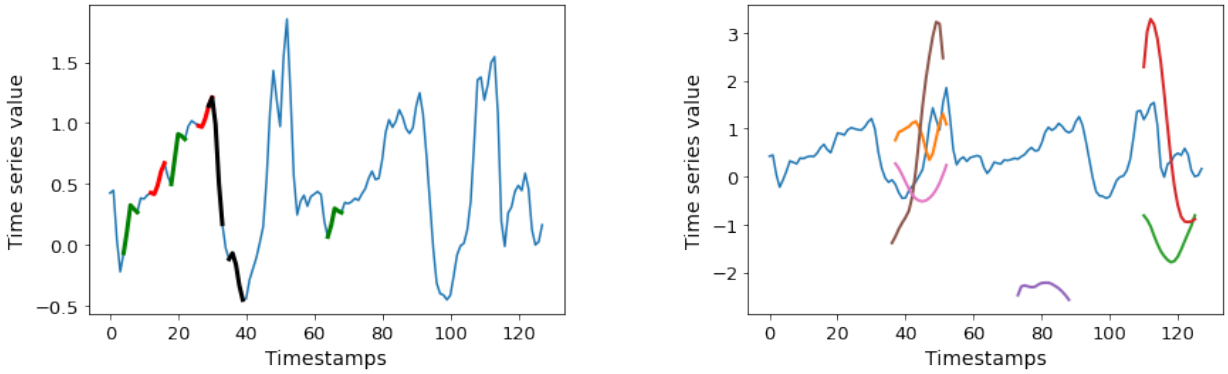
Figure 16: Discords extracted on selected time series and their union

Turning to motif analysis, we want to find out whether there are systematic patterns in our time series that allow us to recognise observations that repeat themselves with a certain cadence, or through discords sequences that represent very dissimilar elements from the entire time series, through the use of the matrix profile. In our analysis we set up a sliding observation window of length 8, as the algorithm calculates the distances for the windowed sub-sequence with respect to the entire time series.

For each single time series we analysed, we extracted the motifs and its anomalies. Since during the analysis we considered one time series at a time, out of a large set that we could draw from, we tried to perform an experiment, namely that of understanding whether by using a larger time series (by concatenating more series of the same subject performing the same activity), we could still systematically identify the same anomalies and motifs. We tried concatenating 3 random time series in order to be able to avoid overly complex interpretations. We deemed this idea interesting also because the records we have in the time series are actually fractions of a larger observation, so we were performing the inverse operation. Figure 15 shows the three separate time series we have chosen and their union. As we can see, as far as the motifs are concerned, we find the same 3 over almost the entire series, which means that they recur systematically.

As far as the anomalies are concerned (see Figure 16), when uniting the three time series, we find again some of the anomalies found in the individual series, or ones with similar behaviour but in different positions,

As a final note, we tried to compare the results from the shapelet classifier with the motifs extracted, to see if what is recurrent inside a single time series (motif) could have some connection with what distinguished time series of a certain class (shapelet). Figure 17 depicts the motifs extracted for a certain time series of class 1 and the predicted locations of the extracted shapelets. In this instance, we observe a lack of overlap between the two concepts. Repeating this operation on many records yielded pretty similar results. The position of the shapelets reflects a similar behaviour to what we have seen during the clustering and classification tasks:



(a) Motifs (b) Predicted shapelet locations

Figure 17: Visualisation of motifs and predicted shapelets on selected time series

classes 1, 2 and 3 are very close together (the three shapelets around timestamp 37), as well as classes 4 and 5 (around timestamp 120), and class 6 is isolated.

4.5 Final discussion

In this chapter we concentrated on the total acceleration signals on the X plane, and analysed the relative time series dataset. Such inertia signal proves itself to be quite effective to classify the different types of walking and laying, while a much greater uncertainty is present for sitting and standing. We confirmed some previous results and insights, while recognising that to more successfully complete a classification task on the Human Activity Recognition dataset, a single inertia signal is not enough, although already pretty useful.

5 Sequential Pattern Mining and Advanced Clustering

In this chapter we will deal with sequential pattern mining techniques and advanced clustering algorithms. For what it concerns the first task, it has been performed on the same dataset used in the previous module, covering the time series analysis. Specifically, the dataset involved, refers to to the total acceleration on the X plane (total_acc_x), where each record is assigned to a specific class label and it represents a time series with 128 timestamps. Otherwise, the advanced clustering algorithms have been performed on the original feature dataset which was used in the first two chapters. This choice is reasonable since, thanks to the analysis applied, we were able to spot and to confirm insights, which were previously formulated through the classification analysis.

5.1 Sequential pattern mining

We begin our analysis applying a Symbolic Aggregate Approximation (SAX) to the dataset, using a combination of 16 PAA segments and 8 SAX symbols (as in the time series analysis). This was helpful in order to understand and to interpret the patterns deriving from the analysis resulting from the exploitation of the PrefixSpan algorithm. We decided to extract the top three closed frequent patterns, with their relative support, distinguishing the results based on which class they belonged. This kind of approach allowed us to compere the results and to make some interesting observation. First of all, as we can see from Table 17 it is clear how, for what it concerns the classes 1, 2 and 3 the closed frequent patterns are quite different among them (despite they share the same symbols). Possibly we can spot some similarity between records

of classes 1 and 2, while it appears evident how, the patterns which characterise records of class 3, are different. Another situation is presented for records of classes 4, 5 and 6. In fact while the first two are equal, meaning that we always have the same symbol in both cases, for what it concerns records of class 6 we have patterns which are always composed by the symbol 0 (representing flat series)

All the conclusions that we can derive from these analysis were quite predictable. In fact as we had already widely explained in the previous module, the time series of class 6 are isolated, and this is why it is reasonable the fact that the closed frequent patterns of that class, are composed just by the 0 symbol. As well, the same explanation could be expressed for the other classes. In fact, as we had already seen, the classes 4 and 5 were very similar (even for the advanced classification methods), and this would explain why they share the same symbol (and the same closed frequent pattern). The same situation is shown for classes 1, 2 and 3 which did not share the same exact patterns despite some similarity (exactly the same circumstances that have been described during the time series analysis).

Table 17: Most frequent patterns with their support, divided by activity class

Activity 1	**Activity 2**	**Activity 3**
(83.93%, [4, 6, 4, 6]), (82.54%, [6, 4, 6, 6]), (81.89%, [4, 6, 6, 6])	(60.76%, [3, 6, 4, 6]), (60.01%, [6, 4, 3, 6]), (59.92%, [4, 6, 4, 6])	(61.86%, [3, 7, 3, 7]), (61.86%, [7, 3, 7, 3]), (61.76%, [7, 3, 7, 7])
Activity 4	**Activity 5**	**Activity 6**
73.64%, [5, 5, 5, 5, 5, 5]	98.47%, [5, 5, 5, 5, 5, 5, 5]	99.86%, [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

5.2 Advanced clustering

In this section we move on to clustering techniques, an unsupervised task we performed on the original dataset used in the first two chapters. The algorithms we adopted are X-Means and K-Modes. We started with X-Means, which does not require an initial guess of the number of clusters (we just set an upper bound of 100), as the algorithm will keep splitting up clusters in a bisecting K-Means fashion until it no longer observes improvements in its BIC score. After running the algorithm, it returned us a total of 53 clusters, with a total SSE of 1282058 and a silhouette score of 0.0743. The clusters were quite unbalanced, with sizes ranging from a couple dozen points to over 300 (no singleton clusters were formed). In Figure 18 we plotted the clusters we obtained on a PCA projection of the dataset. We get a very messy picture with little readability and very hard to interpret.

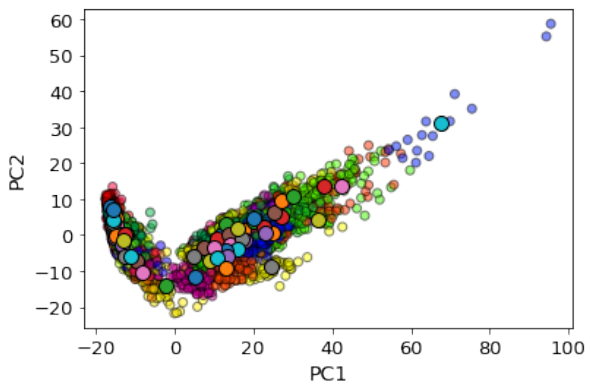
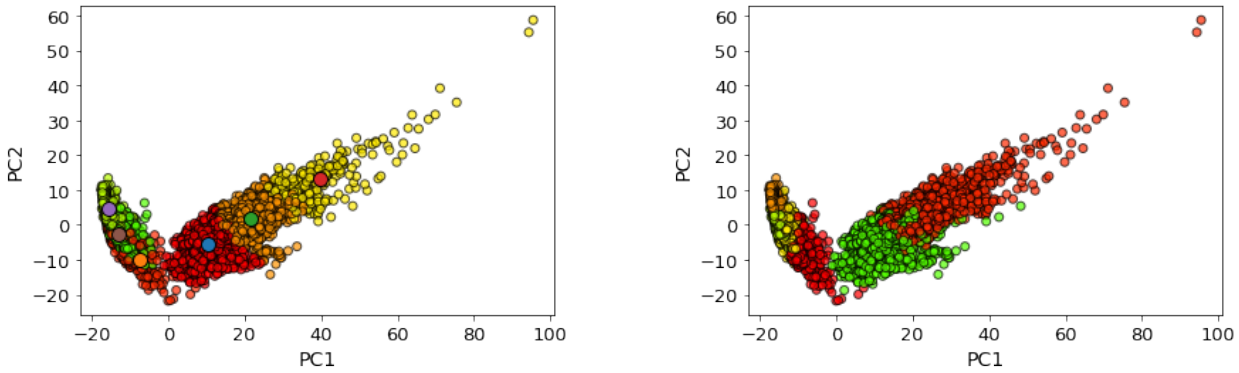


Figure 18: Clusters returned by X-Means with centroids visualised on PCA projection

To gain a more more informative insight, we decided to greatly reduce the maximum number of clusters down to 6 (equal to the number of classes). We wanted to see what natural structure in the data the algorithm was able to capture, trying to get the best possible global configuration thanks to X-Means. We expected a higher SSE (which was 1829855) but we got a higher silhouette score (which was 0.1798). A graphical representation of the clustering can be found in Figure 19a. Table 18 breaks down the composition of the clusters: we can see a neat distinction between the first 3 and the last 3 classes. Notably, class 6 is not as isolated as it emerged from previous tasks, but it was mixed up with records of classes 4 and 5.

We then tried to use another algorithm, which was originally designed for categorical data: K-Modes. We wanted to test whether by discretising the dataset and applying K-Modes we would have obtained similar results. After discretising each feature into quartiles, we proceeded using 6 as the number of clusters. Figure 19b gives a visualisation of the clustering. Comparing it to that of X-Means, we can see how the two are similar although there are some visible differences: X-Means presents 3 clusters along each principal component, while K-Modes has 2 clusters along the first component and 4 along the other. Table 19 displayes the composition of the clusters: again we see a separation between the first 3 and last 3 classes, although the former now appear in only two clusters out of six. The sizes of the clusters present less variability, class 6 is even more dispersed, whereas class 2 is more concentrated in cluster 5. The silhouette score for the K-Modes clustering was 0.11, but we expected a lower score since the algorithm was not designed for our continuous data.



(a) X-Means (b) K-Modes

Figure 19: Comparison of X-Means and K-Modes clusterings visualised on PCA projection

Table 18: Composition of clusters returned by X-Means

	Clust. 0	**Clust. 1**	**Clust. 2**	**Clust. 3**	**Clust. 4**	**Clust. 5**	**Total**
Class 1	645	0	513	68	0	0	1226
Class 2	858	0	210	5	0	0	1073
Class 3	200	0	619	167	0	0	986
Class 4	1	166	0	0	652	467	1286
Class 5	0	49	0	0	395	930	1374
Class 6	3	362	0	0	982	60	1407
Total	1707	577	1342	240	2029	1457	7352

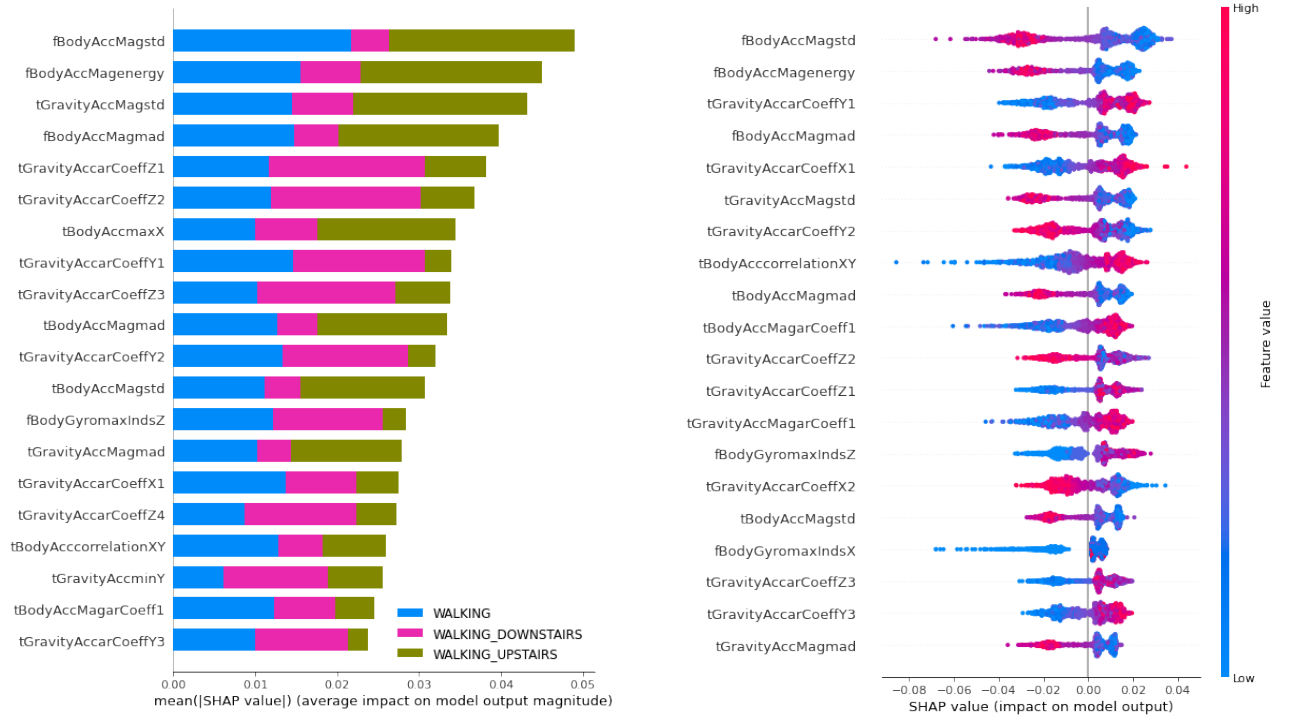
Table 19: Composition of clusters returned by K-Modes

	Clust. 0	Clust. 1	Clust. 2	Clust. 3	Clust. 4	Clust. 5	Total
Class 1	0	551	0	0	0	675	1226
Class 2	0	98	0	0	0	975	1073
Class 3	0	813	0	0	0	173	986
Class 4	110	0	277	254	644	1	1286
Class 5	145	0	173	338	718	0	1374
Class 6	143	0	520	226	512	6	1407
Total	398	1462	970	818	1874	1830	7352

6 Explainability

6.1 SHAP

As far as explainability is concerned, we wanted to better understand the choices made by our classifiers, to focus on the classification concerning only those classes belonging to the "walking" types in order to further understand and seek confirmation. We were interested in starting from a global interpretation (given by the aggregation of SHAP results), so we used the random forest with the TreeExplainer, which is faster compared to the kernelExplainer, and this allowed us to run it on the entire dataset instead of on a subsample, obtaining an accuracy of 0.92. From Figure 20a, we can easily see the impact that SHAP has attributed to each feature while classifying. Visually we can see that the accentuated tendency for the 3 most important features is to have a greater impact on classes "walking" and "walking upstairs", unlike "walking downstairs" which had lower classification results. This could mean that "walking" and "walking upstairs" tend to be movements recognisable by the same features but characterised by different values.



(a) Summary plot of mean SHAP value impact for classes 1, 2, and 3

(b) Summary plot of SHAP value impact for class 1 ("walking")

Figure 20: Some visualisations for SHAP

For example, looking at Figure 20b, which refers only to the walking class in particular, we see how high values of the feature "Fbodyaccmagstd" negatively affected the determination of the assigned target class. For TgravityAccarCoeffX1 it is the opposite, so we can say that it is negatively correlated with the target class. For the "walking upstairs" class instead, Figure 21 shows that there is a very approximate S-shaped relationship between "fbodyaccmagstd" and the target variable, which interacts with "tgravityaccrcoeffZ1" frequently (as found by SHAP). For the class "walking downstairs", which is the one for which the features analysed so far had the weakest impact, we tried to perform a local interpretation on an observation belonging to this class in order to understand the explanation for the prediction of a single instance of it.

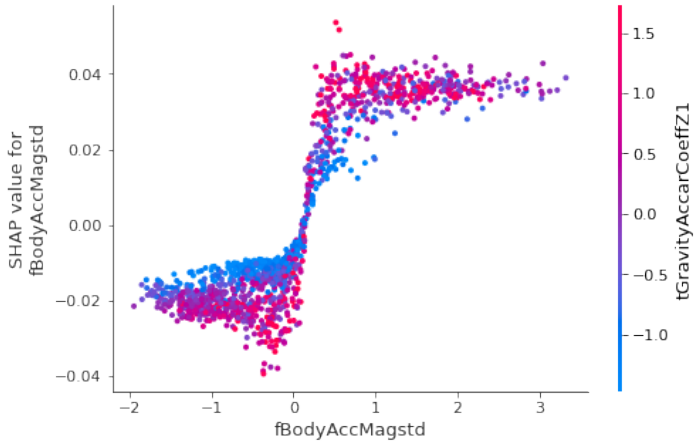


Figure 21: SHAP dependency plot for class 2 ("walking upstairs")

In Figure 22, the bold 1.00 is the model score for the considered observation. The features that were important to make the prediction for this observation are shown in red and blue, with red representing features that pushed the model score higher, and blue representing features that pushed the score lower. In particular, we note that in contrast to "walking" and "walking upstairs", for this observation of class "walking downstairs", we have an important role of the variables "tgravityaccentropyz" that made the score go up and "tgravityaccentropyz", which made it go down.

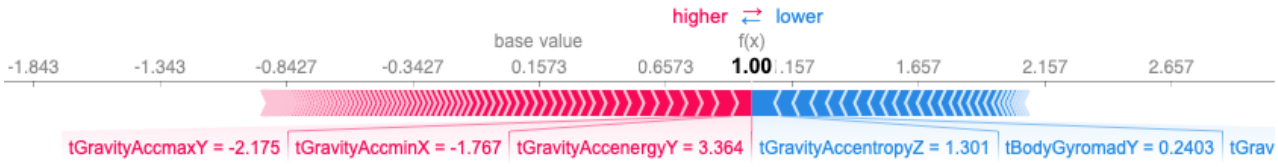


Figure 22: SHAP force plot - local explainability of a record of class 3 ("walking downstairs")

6.2 LIME

As the last step for this chapter, we wanted to focus on the other two classes we studied in the classification part where motion is not present, namely "standing" and "sitting" (as seen in time series analysis, the "standing" class is quite hard to predict). For the classification we used LIME via linear SVM, which was one of the best performing classifiers, where in this case we obtain an accuracy of 0.94 for the predictions of these two classes.

The output of LIME is a list of explanations, reflecting the contribution of each feature to the prediction of a data sample. This provides local interpretability, and it also allows us to determine which feature changes will have most impact on the prediction.

With regard to Figure 23, we can see what information LIME has extracted concerning one instance of class "standing", to which the model assigns a probability prediction of 0.99. The values of the variables in blue in this case are those for which this observation belongs to the "standing" class, while the orange values are those for which with those values the activity are associated more with "sitting". As we can notice for standing, the most important variables belong to the t/fbody family.

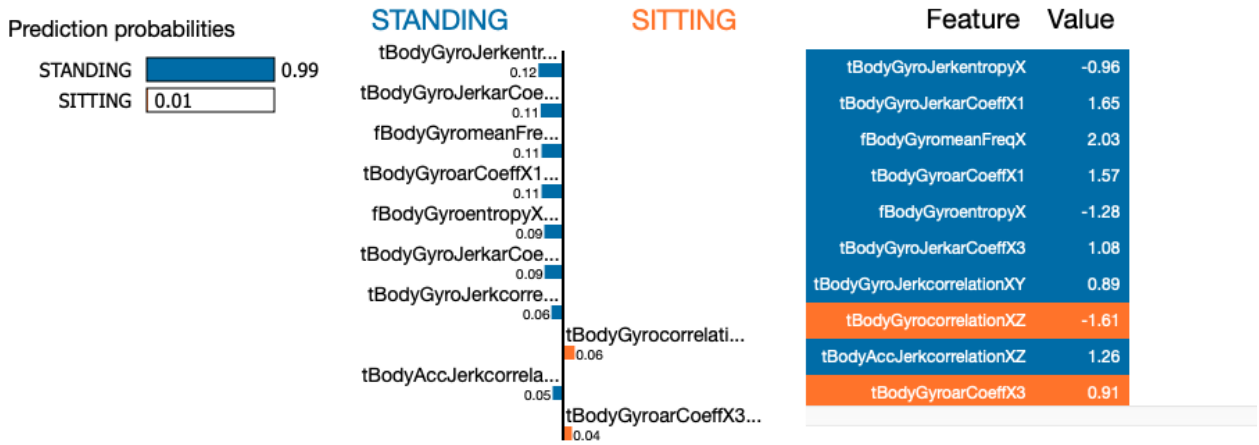


Figure 23: LIME explanation for a record of class 4 ("standing")

While taking an observation belonging to the sitting class, Figure 24 we see that the variables that weighed most on the determination of the target class belong to the gravity-related family. We found some variables that were also present for the determination of the "standing" class and many others dedicated only to "sitting" such as "TGravityAccminY", which is the most important.

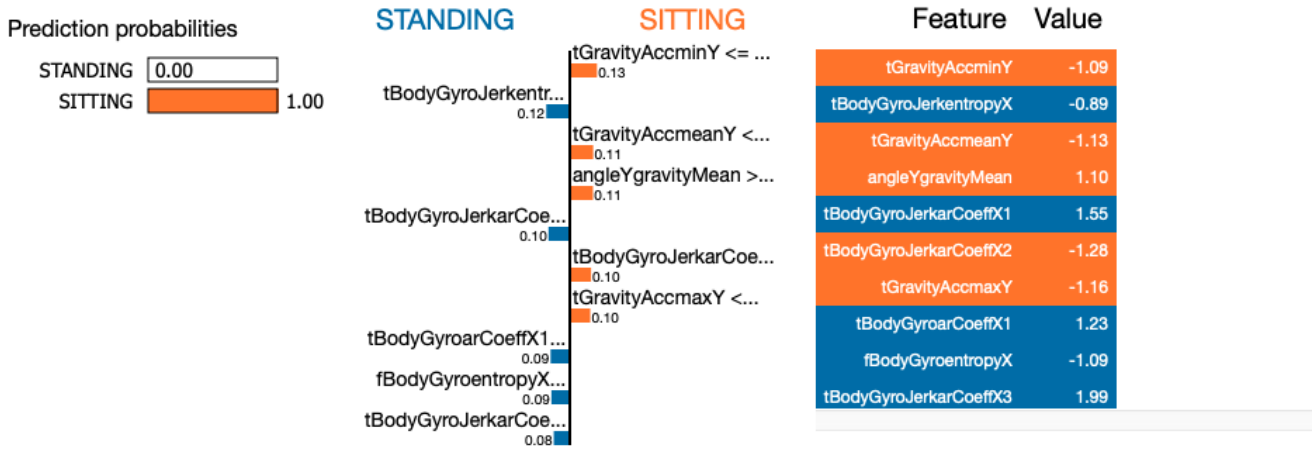


Figure 24: LIME explanation for a record of class 5 ("sitting")

Considering that there are 561 features related to motion sensor values, all of them with minimum deviations and of a particular and difficult knowledge domain, it is important that the model identifies as many different features as possible for class determination as well as focusing on the values they can take. The conclusion, therefore, is that explainability methods provide us with a good explanation of the choices made by the classification models and could be a good starting point for optimising certain phases of the analysis by paying particular attention to the most crucial variables and their values.

7 Conclusion

To summarise what we have carried out along this project, we remark that we dealt with different versions of the same dataset depending on the type of analysis that we wanted to compute. None of these versions presented needed any significant preprocessing steps and since the beginning, it has been relatively easy to apply with success, some classification algorithms to distinguish the different activities. Due to this fact, in more than one occasion, we decided to challenge ourselves by complicating the initial dataset, removing records which resulted useless to our aim of boosting our knowledge of advanced data mining techniques. Nevertheless, we apparently achieved nice results and thanks to the explainability methods, we were able to understand more deeply and to clarify some of the hidden processes behind the scenes. In conclusion we can state that there are some type of signals which are more determinant than other when it comes to classification, both in balanced and imbalanced situations. Moreover, we can affirm that all along the entire project we found consistency in the results obtained, confirming the same knowledge about the dataset, through different analyses.