Faculty of Applied Sciences

Introduction to Machine Learning

ELEN062-1

Project 3:
Human activity prediction

Authors :
Lewin Sacha - s181947
Delcour Florian - s181063
Makedonsky Aliocha - s171197

Instructors :
Geurts Pierre
Wehenkel Louis
Project instructor :
Claes Yann

December 2021

1 Abstract

In this report, we describe our methods and thinking for building an efficient activity recognition classifier, using the Scikit-Learn library. Each step of the process of building this classifier is detailed, along with the classifiers tested, such as random forests, multi-layer perceptrons, k-nearest neighbors, and the various techniques attempted to improve the accuracy, such as boosting, bagging or voting. The resulting classifier's accuracy is evaluated using leave-one-out cross-validation, and achieves a mean accuracy of 90%. It reached an accuracy of 89.14% on the public leaderboard on Kaggle.

2 Introduction

The training data set is composed of 31 sensors. Each of them measured 3500 time series, with 512 measurements over 5 seconds. Each time series corresponds to one of 5 subjects. The physical sensors are :

2. Heart beat rate (bpm)
3. Hand temperature (°C)
4, 5, 6. Hand acceleration (3D, m/s^2)
7, 8, 9. Hand gyroscope (3D, rad/s)
10, 11, 12. Hand magnetometer(3D, μT)
13. Chest temperature (°C)
14, 15, 16. Chest acceleration (3D, m/s^2)
17, 18, 19. Chest gyroscope (3D, rad/s)
20, 21, 22. Chest magnetometer(3D, μT)
23. Foot temperature (°C)
24, 25, 26. Foot acceleration (3D, m/s^2)
27, 28, 29. Foot gyroscope (3D, rad/s)
30, 31, 32. Foot magnetometer(3D, μT)

The labeled activities are :

1. Lying
2. Sitting
3. Standing
4. Walking very slow
5. Normal walking
6. Nordic walking
7. Running
8. Ascending stairs
9. Descending stairs
10. Cycling
11. Ironing
12. Vacuum cleaning
13. Rope jumping
14. Playing soccer

Intuitively we directly see some links between the physical sensors and the activites. For example, it is logical that while playing soccer, the subject will have high foot accelerations every time he shoots in the ball, or when he is ironing the hand temperature and magnetometer will increase, when doing any sport his heart beat rate will increase, ...

Physical interpretation will be very important, because depending on the activity, we might get very special values for a time series. For example, rope jumping leads to fast increases of chest acceleration. These should not be considered as outliers by our model.

Our goal is therefore, for any given list of 31 sensors with 512 measurements each, to be able to guess the corresponding activity, no matter the individual. This means each observation has 15872 features. There are 3500 observations in the training set, and 3500 in the testing set.

3 Naive approach

At the beginning of the project, we didn't preprocess the data and we tested a `MLPClassifier`, which is a neural network, where we can choose the number of hidden layers and their respective sizes. We got horrible results on Kaggle (0.18285) with 169 as the number of neurons of the first and only layer used (chosen by computing the `accuracy_score` of the predictions, with hidden layer sizes between 1 and 200). Those bad results are explained by the fact that since the NA's are replaced by -999999.99 they greatly influence the results, making the classification impossible, even more since if there are NA's for one sensor at a time-serie, all the values of this sensor are NA's so they are really dominant. So the algorithm has a lot of difficulty to converge, and we don't get anything good.

4 Data pre-processing

Before training a model, it is extremely important to pre-process our data to avoid bad results which can come from NA's or outliers. Moreover, since the dataset is very large, we might perform feature extraction.

4.1 Missing values

In our dataset, missing values are represented by -999999.99. We started by replacing the NA's in a column (a sensor at a particular time between 0s and 5s) by the mean of this same column (so the mean for all types of activity and all subject, all together for a particular sensor) and we tested a `RandomForestClassifier`, which builds decision trees on different subsamples of the data set and uses averaging to improve the results. The score on Kaggle was 0.2, which was still really bad.
Thus we improved our way of pre-processing, by replacing the missing values by the mean of the values for this sensor at the same time with respect to the activity and the subject. For example, if there are NA's for the sensor 2 of the time-serie 5, which represents the

measure of the activity 6 done by the subject 4, we replace the NA at time t of this sensor by the mean of all the values of the sensor 2 at this same time t for the activity 6 performed by the subject 4 (so for every time-serie in those conditions). For some pairs activity-subject, all the values were NA's for one sensor, which make the way of pre-processing we used impossible. So for those pairs, we only respect the activity done to compute the mean and not the subject anymore.
We observed that there is no NA in the test data set so it causes no problem to use the activity to pre-process the train set which actually has 4 957 990 missing values (8.92% of train set).
With this pre-processing we got better results on Kaggle but still bad, 0.56571 with a `MLP` classifier with 167 neurons in the first and only layer used.

4.2 Handling outliers

Exploring the data, we also noticed the existence of some outliers in both training set and test set. Indeed, the maximum values in X_{train} and X_{test} are respectively 1713.45 and 1597.99 while the minimum values are -32091.38 and -110348.95, which make no physical sense.
First we tried to train some models to identify outliers like `Isolation Forest, Minimum Covariance Determinant`[1]`, Local Outlier Factor and One-Class SVM` but these outliers classifiers did not convince us. In fact, depending on the hyper-parameters, either some valid measurements were detected as outliers, for example the vertical acceleration peak of the jumping rope, or not enough values were detected as outliers.

Thus we decided to perform the detection by ourselves. Figure 1 represents the boxplot of the median of X_{train} grouped by activities. Thus we can see that for each activity, the maximum median does not exceed 200 and the minimum median does not exceed -200. Then all values of the dataset outside the interval [-200, 200] will be considered as outliers. To handle these outliers, we will replace them by mean of the first previous valid value and the next one.
A better solution for dealing with outliers would have been to consider outliers for each activity and subjects independently. Unfortunately, we did not have access to the test set so it was not possible.

4.3 Feature extraction

The time-series dataset is extremely large and so computationally expensive. In addition, model trained on this type of dataset cannot make the right links to perform the classification. Then we understood that we had to reduce the dimension of the dataset in an efficient way.
We thought about taking only 1 column over 10, because since there are 512 measurements for 5 seconds of time, it makes 1 measurement every $10ms$, even though the average

[1]Need a Gaussian dataset which is not really our case

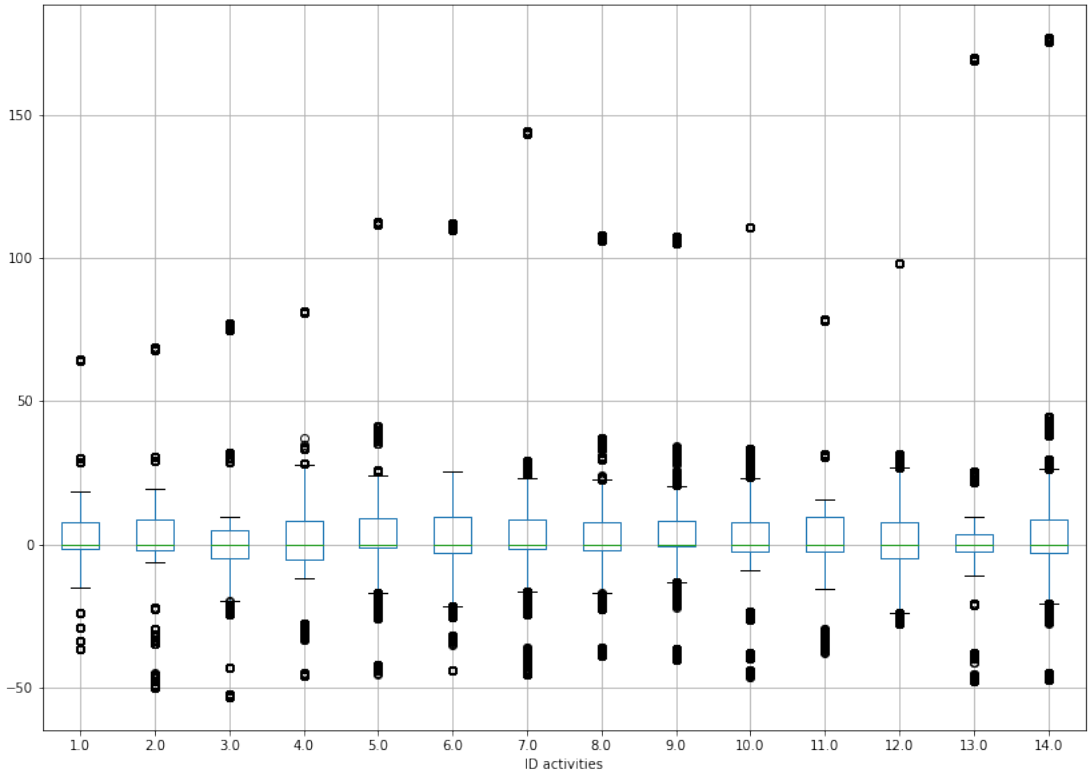

Figure 1: Median boxplots grouped by activities

human reaction time is about $400ms$, so there was really too much measurements, but this approach didn't improve our naive results.
Then we replaced the 512 columns of each sensor on each time-serie by the mean and the standard deviation of the 512 measurements, which gave us 62 columns (since there are 31 sensors), and it allowed us to train and test our models far quicker. It didn't only reduce the time of computation, but it improved a lot our accuracy ! We got 0.81428 on Kaggle with the `RandomForestClassifier` with a maximum depth of 6 (chosen to optimize our local `accuracy_score`) and 200 trees (more there are, bigger the accuracy will be), which was by far better than previous results. This can be explained by the fact that those statistical measures have a real physical meaning and are great synthetizers of information.
For example, with the acceleration of the different types of walking we will have a low mean, assuming that the speed of the subject is approximately constant, and a low variance for the same reason. But comparing with playing soccer, which will have a low mean too (accelerations are directly compensated with deceleration, since there are short sprints) but a huge variance because the player runs and stop running frequently so the acceleration increases really quickly and decreases quickly too, which causes huge differences and so huge variance.
In a second step, we added the median, the kurtosis, the first five point of the one-dimensional Fourier transform and in a third step the interquartile range. However, these will be explained later with model improvement.

4.4 Data scaling

We started scaling our data for the `MLPClassifier`, it was not necessary for random forest classifier (or any model that is based on tree algorithm) because it's only needed for distance based algorithms. However we had to choose which scaler to use, because, as you can see on Figure 2, there are different types which all use their own method to scale the data. After having tried them all, the best one was that we chose `PowerTransformer(method='yeo-johnson')`. Ideally, we should have tried them all for all the parameters of all our classifiers but it would have been really time-consumming for results that seemed pretty close.
This technique improved a lot the different distance based classifiers that we tried like k-Nearest-Neighbors, Neural-Network, SVM, ...

Name	Sklearn_Class
StandardScaler	StandardScaler
MinMaxScaler	MinMaxScaler
MaxAbsScaler	MaxAbsScaler
RobustScaler	RobustScaler
QuantileTransformer-Normal	QuantileTransformer(output_distribution='normal')
QuantileTransformer-Uniform	QuantileTransformer(output_distribution='uniform')
PowerTransformer-Yeo-Johnson	PowerTransformer(method='yeo-johnson')
Normalizer	Normalizer

Figure 2: Different types of scaler

5 Model validation

The first testing method of our model was bad because we did not separate the train and test sets by subjects. Then we used the last subject as test set and the others as training sets. However, a rigorous approach is to perfom cross-validation.

Thus we started to improve our local tests of accuracy, since we use those tests to choose the value of our tuning parameters of the different classifiers. To improve those, we compute the leave-one-out cross validation, taking 4 out of our 5 subjects as the training set, and the last subject being the testing set, and doing that 5 times to compute an accuracy score with each of the different subject as testing set, and finally doing the average of the scores. We tried it on the `MLPClassifier` with 199 and 31 as the sizes of our two hidden layers, but we obtained results that are a little bit higher in local than on Kaggle but it's still a good measure of magnitude accuracy. The cross validation that we perfomed allowed us to test a lot of combinations that the Kaggle platform could not accept due to the restricted number of submissions.

6 Model optimization

In order to perform the best possible classification, we need to explore the hyper-parameters space of all possible classifier algorithms. However, it takes long time to perform this operation so we tried some techniques that we understood from the course or from papers on internet to guide our testing methods.

6.1 Different classifiers

- ❉ It was time to test new classifiers, and we started with the **support vector machine classifier** (`SVC`). The principle is to draw a linear bound between the data, and this bound tries to maximize the margin, which is the sum of the distances from the observed data to the margin. We optimized the regularization parameter `C` and the best score obtained was 0.887 with `C = 0.95`.
- ❉ We also tried the **Decision Tree classifier** but our best score was only 0.697 with `max_depth=7`.
- ❉ **K-Nearest-Neighbors classifier** gave us a score of 0.835 with optimized parameter `n_neighbors=10`.
- ❉ We also tried **MLP classifier** so the multi-layer perceptron also called neural network where we tried to optimize the number of hidden layers and the number of neurons in each of them. We did not save the results, however we used `MLP` in our final solution which is better and described below.

These classifiers alone did not show great accuracies and this is why we tested ensemble classifiers.

6.2 Ensemble Classifiers

The main advantage of ensemble classifiers is that it combines several learning algorithms to improve the predictive performance. Especially the averaging classifiers, that train multiple different classifiers, or a single type of classifier but on multiple subsets of the data, and take the average of the results. The main drawback is the computation time which is longer.

As said above, we firstly tried **Random Forest classifier** that uses many decision tree classifiers on subsamples of the dataset to improve the predictive accuracy. Higher is the number of estimators, better are the results but computation time also increases. Our best score with cross-validation was 0.826 with `max_depth=7`.

We tried the **VotingClassifier**, which combines different classifiers. The classifiers must vote on which class will be attributed to the observation, and we can set a weight for each classifier. There are two types of voting, `hard` that uses predicted class labels for majority rule voting, and `soft` that predicts the class label based on the *argmax* of the sums of the predicted probabilities.

In our case we tried initially the soft voting, with 3 classifiers (**MLP, Random Forest and SVM**), giving a bigger weight (2 while the weights of the two others are 1) to the `RandomForestClassifier`, with a maximum depth of 6 and a number of estimators of 125, that gave us the best results so far. This improved greatly our results, to 0.86571. We tried to not scale the data to see if it improved the results but it did the opposite.

We then tried a **BaggingClassifier** based on the **RandomForestClassifier**, that fits the base classifier on different subsets, and then proceeds to a vote to decide the class that will be attributed to the observation. We got 0.83142 on Kaggle, which is less good than the **VotingClassifier**.

6.3 Improving `VotingClassifier`

Since the **VotingClassifier** was our best classifier so far, we tried to improve it.
Instead of replacing the columns of the data only by the mean and the standard deviation, we also added as features the median, the one-dimensional discrete Fourier Transform (5 first values), and the kurtosis (expressed by $Kurtosis(x) = E[(\frac{x-\mu}{\sigma})^4]$). Kurtosis is a measure of whether the data are heavy-tailed or light-tailed relative to a normal distribution while Fourier Transform provides frequency information of the data. With those new features, and `hard` instead of `soft` voting, we managed to obtain a score on Kaggle of 0.88857.

To improve it again, we replaced the **SVM** used for the voting, by **KNeighborsClassifier** with 3 nearest neighbors. Thanks to this change, we obtained a result of 0.89142 which is our best one so far.

We then tried to add robustness[2] by computing the mean and the standard deviation without taking into account the 25 biggest values and 25 lowest values, to keep only 90% of the initial data. The results weren't improved with this technique, we got 0.88 on Kaggle.

This decrease might be caused by a loss of information, so we tried to replace outliers (we replaced, by the mean of the following and preceding values, every value above 200 in absolute value because those values don't make any physical sense) to compute the mean and the standard deviation. In addition we used only the 3 highest values of the Fourier transform and we also used the interquartile range as features. Thus, we got a score of 0.91428 on Kaggle which was way better !

Still using the voting classifier with **MLPClassifier**, **RandomForestClassifier**, **KNeighborsClassifier**, we managed to get our best result on Kaggle, which is 0.92000. We used `hidden layers sizes = (132,40)`, `n_neighbors = 3`, `n_estimators = 500` and `max_depth=7`. The weight for the `hard` voting was respectively 2, 3, 2 associated to **MLP, RF, KNN**.

[2]Initially, we did not achieve to handle outliers so we tried other ways to do this

7 Results summary

The Table 1 resumes the different results we got on Kaggle with the classifiers we described in the previous sections.
Legend :

- RF = Random Forest
- MLP = neural network with Multi-Layer Perceptron
- cross val = cross validation
- SVC = support vector machine classifier
- scaling = done with `PowerTransformer(method = 'yeo-johnson')`
- new data = when we added the median, Kurtosis measure, one-dimensional Fourier Transform
- KNN = K-Nearest Neighbors
- 90% = mean and standard deviation computed after keeping 90% of the data
- no outliers = handling outliers, and new features (IQR and 3 highest values for Fourier Transform)
- iqr = interquartile range

Adter the line "MLP cross val" to the end we always used leave-one-out cross validation to tune our parameters.

Classifier	Private score	Public score
RF unscaled	0.79238	0.81428
MLP unscaled	0.76730	0.79142
MLP cross val	0.73809	0.76000
Voting MLP, RF, SVC	0.80222	0.82000
Voting MLP, RF, SVC scaled	0.83841	0.86000
Bagging unscaled	0.82444	0.83142
Voting new data	0.87396	0.88857
Voting MLP, RF, KNN	0.87142	0.89142
Voting MLP, RF, KNN 90%	0.87269	0.88000
Voting MLP, RF, SVC no outliers	0.89523	0.91428
Voting MLP, RF, KNN no outliers	0.88698	0.92000

Table 1: Results summary

8 Conclusion

The model[3] that gave us the best local score and on Kaggle platform was the **Voting classifier** composed of **MLP**, **Random Forest** and **K-Nearest-Neighbors**. Cross validation was a great tool to test our many models even though it overfits a bit the data as we have seen on Kaggle.
Through this project, we realized that tuning hyper-parameters in order to have efficient

[3] 6.3

models was an hard and time-consuming task. Indeed, hyper-parameters space are often very large due to the models complexity.
We also noticed that pre-processing (feature selection, outliers and missing values handling) plays a big role in the accuracy of the models, and in the computation time. We can easily go from hours to minutes of computation.
As well as analysing the physical meaning of the data we treat is important to understand how to represent it.