University of Liège
School of Engineering

Project 3 - Human activity prediction

ELEN0062: Introduction to machine learning

Julien Gustin, Joachim Houyon, Yassine Maziane

September 13, 2023

1 Evaluation

When starting a project like this, the most important thing is to evaluate our results, to do so, we looked at how our results will be evaluated on the set, and we saw that *TS file have been collected from 3 new subjects not represented in the LS sample.* Therefore our idea was just to do a **KFold** cross validation for which each fold represent all examples associated to one subject and in that way each fold is then used once as a validation while the 4 remaining folds form the training set.

The issue with that way to evaluate our model is that some subject do not have any example associated to a certain activity and others have a lot of **NaN** which does not help to have a totally unbiased evaluation, but by taking the mean over all the test fold we get something quite close to the score given by kaggle when submitting our results.

About the metric we use the same as on kaggle, the accuracy:

$$accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

This metric is interesting in our case because we have a well balanced class balance data set.

2 Data

2.1 Preprocess

In order to preprocess the data the first thing that we have done is to replace NaN values by the mean of the column training set for both the train and test sets. This allows not to discard all data associated with that row so that we keep as much data as possible.

We also standardize all features with **StandardScaler** train on the training sample to transform again the train and test set.

2.2 Features extraction

In the machine learning literature, feature extraction refers to the process of creating new features from an initial set of data. These features encapsulate the central properties of a data set and represent it in a low-dimensional space that facilitates the learning process[1].

This initial data set of raw time series is too large and contains noise that makes the learning task difficult. Moreover machine learning algorithm does not have an overview of the plot like us and a way to give it some insight about theses time series is to extract features from it.

2.2.1 Sliding window

Our first idea was to use a kind of sliding window of a certain size that allows overlap, this sliding window goes through each time series (each sensor for a given sample) in order to extract features from these small parts of time series.

[1]https://www.sas.com/content/dam/SAS/support/en/sas-global-forum-proceedings/2018/2020-2018.pdf

The features extracted were:

- **Mean**
- **Standard deviation**
- **Slopes**
- **Min value**
- **Max value**
- **Median**

Using that technique gives us really good results and sends us to the top 1 of the public leaderboard for a while.

2.2.2 Tsfresh

At some points we were stuck in a minima for which we were not able to improve our best models, therefore the only thing improvable was the features extracted. After some research we found the library **Tsfresh**. Given a time series (one per sensor), it returns all features selected associated to that time series and this for each sample.

We have therefore spent some time reading the documentation and literature in order to choose features that seemed useful to our problem[2]:

- *abs_energy*:
  - Sum over the squared values
- *maximum*:
  - The highest value of the time series
- *absolute_sum_of_changes*:
  - Returns the sum over the absolute value of consecutive changes in the series
- *fft_coefficient*:
  - Calculates the 20 first fourier coefficients of the one-dimensional discrete Fourier Transform for real input by fast fourier transformation algorithm
- *fft_aggregated*:
  - Returns the spectral centroid (mean), variance, skew of the absolute fourier transform spectrum.
- *linear_trend*:
  - Calculates a linear least-squares regression for the values of the time series versus the sequence from 0 to length of the time series minus one. (slope)
- *mean*
- *number_peaks*

[2]https://tsfresh.readthedocs.io/en/latest/api/tsfresh.feature$_{e}xtraction.htmlmodule - tsfresh.feature_{e}xtraction.feature_{c}alculators$

  - Calculates the number of peaks of at least support n in the time series x. (n=4)
- *variance*

We could obviously spend more time selecting the best features to extract and best parameter associated to it but it seems to be a good starting point.

2.3 Results

Here are some results with models that had been tuned via a grid search for each feature extraction techniques.

Models	**Raw data**	**Sliding Window (128)**	**TSFreshFeatureExtractor**
KNN	0.4689 | N/A	0.6968 | 0.7233	0.7569 | 0.8400
SVC (OvR)	0.5430 | N/A	0.7160 | 0.7400	0.8350 | 0.8471
Random Forest	0.5962 | N/A	0.7600 | 0.8142	0.7926 | 0.8686

Table 1. Accuracy score for different machine learning algorithms and features extraction techniques formatted as *cross-valdiation score | kaggle public score*. N/A are used for missing values

As the number of tries on kaggle is limited, we were not able to try all models with different ways of handling data, therefore we just tried the ones that seemed the most interesting and promising to us, i.e, the ones that gave the best score in cross-validation.

But, given these results we can clearly see that extracting features from time series is a good idea and using TSFreshFeatureExtractor helps also a lot by ease the extraction of more complicated features.

For the next points we will only consider when using *TSFreshFeatureExtractor* as it gives on average a better score.

3 Models

3.1 Different models

In first approach, after pre-processing the data, we decided to try a few classifiers with a bare tuning of their parameters in order to get a broad idea of the classifiers that might work well from those that are not so useful to our goal. This idea is clearly not perfect since many models need their parameters to be very well tuned in order to return solid results, SVM being an example of those.
The most appropriate and interesting models among those studied will then be tuned thanks to a grid search and then be analyzed in details in the next sections.
Results may be observed at table 2.

Model	CV Score %
KNN()	71.1332
SVC(kernel="linear", C=0.025)	80.6875
SVC(gamma=2,C=1)	2.1891
SVC()	77.4377
DecisionTreeClassifier()	55.6654
RandomForestClassifier()	74.4094
GaussianNB()	61.5719
LinearDiscriminantAnalysis()	10.4007
QuadraticDiscriminantAnalysis()	10.4081
MLPClassifier()	80.9883

Table 2. Cross-Validation results of different models

We observe that KNN, RF, SVC and MLP offer quite good results. Furthermore, the 3 latter possess a lot of hyper-parameters that may be tuned. We decided to study KNN in order to benchmark the 3 complex models to a very simplistic one. Indeed, if the difference between complex and very simple models is not large, one may prefer to go with the simple one.

3.2 KNN

We first decided to start with KNN as it's a simple classifier. KNN mainly depends on 3 parameters:

1. *n_neighbors* : the number of neighbors considered
2. *weights* : the weights that will be given to each point of the neighborhood. It's either uniform or inversely proportional to the distances separating the sample currently being classified to all training points in the neighborhood.
3. *p* : p=1 corresponds to the Minkowski distance while p=2 corresponds to the euclidean distance.

After performing a grid search with many potential parameters, we observed that the optimal parameters were a number of neighbors equal to 10, weights that are defined in an inversely proportional way and the distance defined in the minkowski sense.
Let us now study the performance of this model with respect to its parameters. Notice all accuracy measures are computed using a mean over a cross-validation procedure.

As specified above, the "distance" weight seems to output better results and this is indeed what we see on figure 1. For any number of neighbors we observe that distant dependant weights lead to better results. This is quite expected and almost intuitive to rely more on the close neighbors than to rely identically on all neighbors. Furthermore, close neighbors being more likely to share the same class as the sample being studied, far neighbors are more likely to be of different class. Therefore, giving less importance to those that are far and more to those that are close seems like a reasonable choice.
We also specified a certain parameter such that when p=1, the minkowski distance is used while p=2 allows to use the euclidean distance. We observe on figure 2 that for any number of neighbors, the minkowski distance leads to better results.

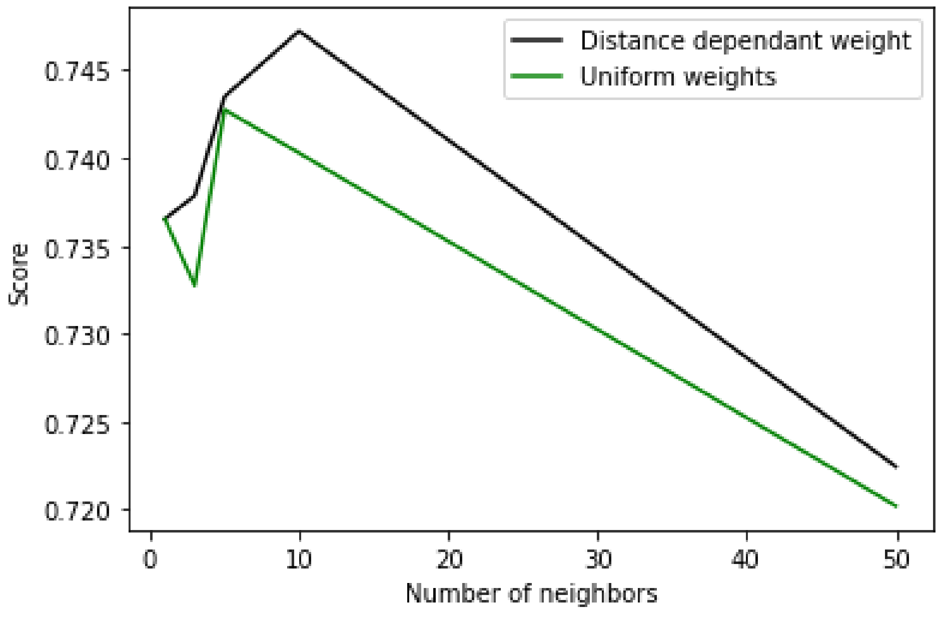

Figure 1. KNN accuracy score with different weights

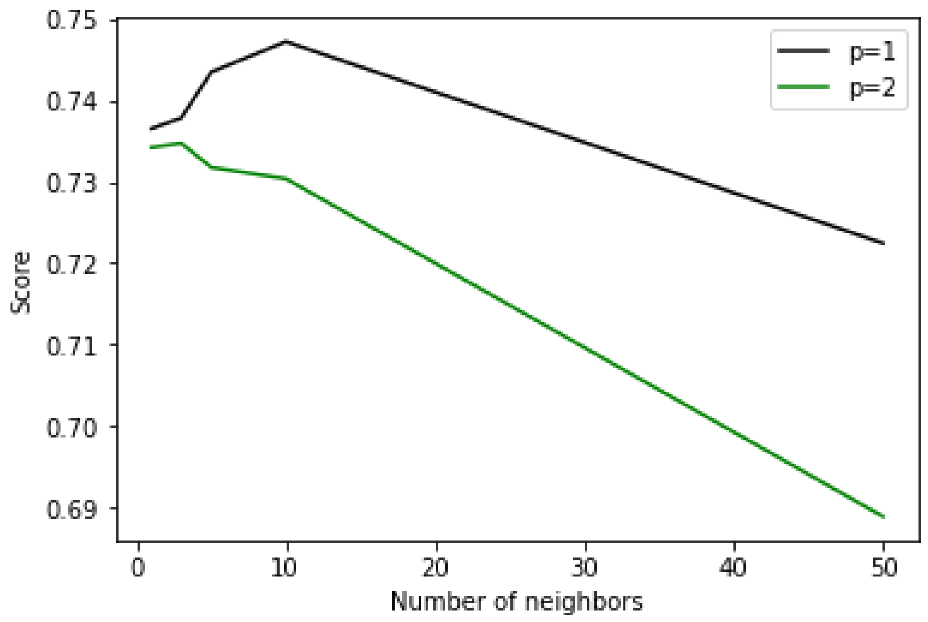

Figure 2. KNN accuracy score with different p

3.3 SVC

3.3.1 OvR and OvO

The idea is to use `Support vector machines` to classify our data. However, instead of having a single `SVC` that classifies everything, we used two strategies:

- **One-vs-the-Rest (OvR) multiclass strategy:** For each class c, a binary SVC is trained on the data and returns either 1 (is predicted as class c) or 0 (predicted as other than c)
- **One-vs-One (OvO) multiclass strategy:** For each class $c_1, c_2, c_1 \neq c_2$, a binary SVC is trained on the data and returns either 1 (predicted as class c_1), or 0 (predicted as class c_2)

These strategies are very common in multiclass classification. The output prediction is the output from the classifier with the highest score (i.e. the highest confidence).

3.3.2 SVC parameters tuning

A grid search has been applied on several parameters such as:

- **Kernel:** Either `Gaussian` or `Polyniomal`. If `Polyniomal`, we also tune the `degree` of the polynomial
- **C:** Find a good value of `C` such that we find the better compromise regarding bias and variance
- **coef0:** The independent term can vary with the chosen kernel (only used for the 'poly' kernel)

3.3.3 Experiment

The best parameters for our SVC were a `polynomial kernel of degree 3` with $C = 0.25$ and **coef0** $= 2$. Whether we chose a `OvO` or a `OvR` strategy, the results were quite the same. On our cross validation, on a `OvR` strategy, it gave a mean score of 0.8350. The classifier did a score of 0.8471 on `Kaggle`. The scores are very close to each other, the model is consistent.

3.4 Deep neural network

By looking at the results in table 2 we have seen that MLP gives quite good results. However, it is quite difficult to build an *almost* fully custom neural network with scikit-learn. Therefore we have searched for another library and we found **Keras**, a model that allows to build relatively quite custom deep neural networks. Our architecture is composed of

- **Input layer**:
  - # features neurons
- **Hidden layer**
  - 300 neurons - activation = *relu* - l2 normalisation
  - 100 neurons - activation = *relu* - l2 normalisation
- **Output layer**:
  - 14 neurons - activation = *softmax*

Note that before every dense layer, while training we perform a **dropout** of 20% of the neurons (found with grid-search) in order to regulate our neural network and to prevent over-fitting. The idea of the dropout is such that at every training step, each neuron (excluding output neurons) has a probability p of being ignored during this training step.

L2 normalization is also used to prevent over fitting.

Regarding the loss function, we have used something relatively and widely used for multiclassification : *sparse_categorical_crossentropy* using a stochastic gradient descent as optimizer (not the best one, but the one seen in class).

Concerning the number of layers and number of neurons at each layer, we have performed different combinations and this is the one that gives the best results with cross-validation.

When looking at the figure 3 we can see the accuracy and loss during the training and we can clearly see that subject 3 and 4 give really poor results on the test set, which is expected because they are the two subjects with the most N/A data. Moreover, by looking at the accuracy curve we can pretend that we have reached a *local* minimum as it does not seem to improve a lot after 30 epochs.

Finally this neural network gives an accuracy score of **0.8289% on our cross-validation** and a score of **0.9285% on kaggle** ! This model leads us to the top of public leaderboard. However, we are almost sure that we are just lucky and we should just take into consideration the local score.

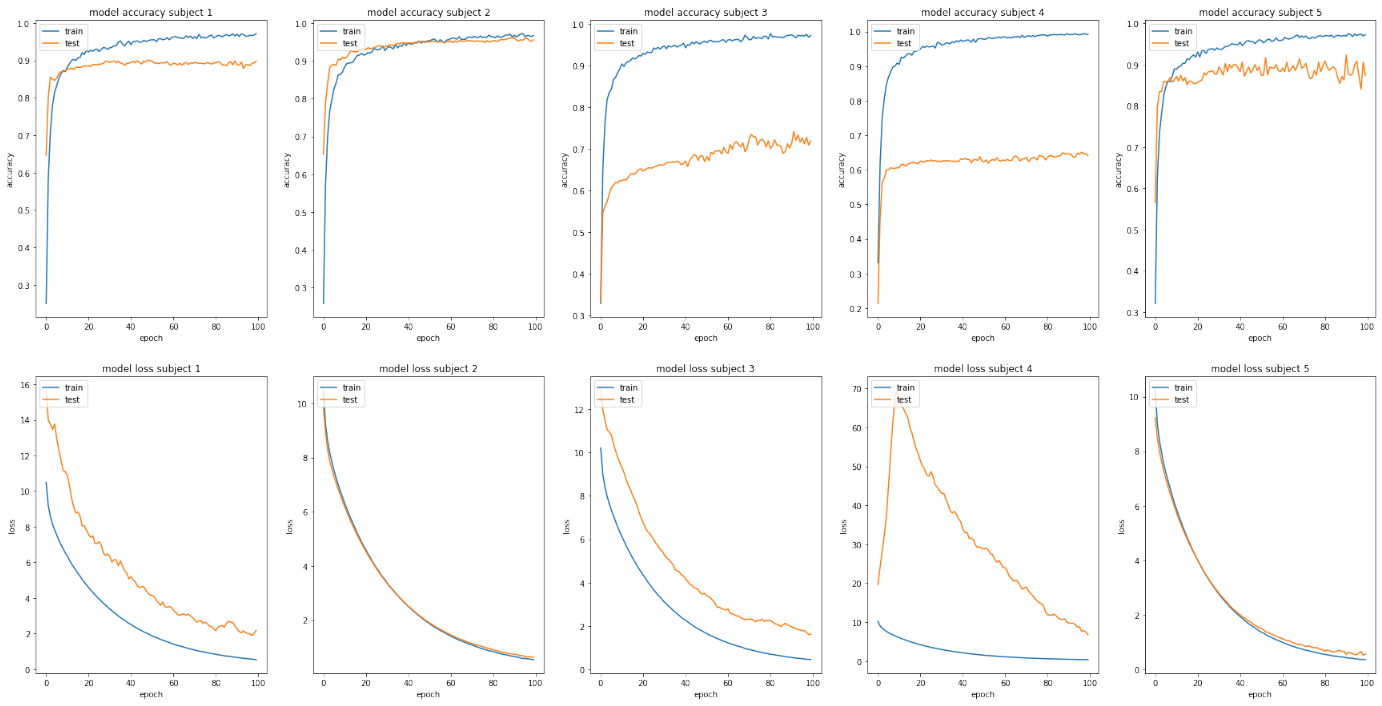

Figure 3. Accuracy on loss during training with 100 epochs

3.5 Ensembles Learning

3.5.1 Random Forest

Basically, a random forest is a set of trees which were built by choosing the best attribute out of a random subset of all attributes at each step. We call it random because the subset of attributes considered at each step is randomly drawn from the set of all attributes. Furthermore, the data used to build all trees is not the original data set but a bootstrapped version which typically uses only $\frac{2}{3}$ of the original data set . To classify a test sample, the latter will be executed on all trees and a classification rule will allow to classify the test sample. Most often, the most frequent class is used. Since a third of the original data set, also called the out-of-bag data set, is not used for building the forest, we may use them on the forest as a test. The proportion of out-of-bag samples that the forest misclassified is known as the out-of-bag error.

Unfortunately, this error was not used in the context of this project as the original data set is made out of measurements coming from 5 persons. Thus, as the bootstrap data set randomly selects samples with replacement, it is highly probable that the same person will have some samples used for the LS and some used for the TS. This not something we want, that's why we use a cross-evaluation technique leaving one person out at each fold.

Random forest mainly depends on 5 parameters:

1. Number of trees in the forest
2. Bootstrap boolean : whether we use a bootstrap strategy or not
3. Bootstrap ratio : the proportion of samples that will be used for creating the bootstrapped data set
4. Max features : the number of features that will be randomly drawn at each step
5. Max depth : the maximum depth of the tree

After running a grid search on multiple possible values for each of the enumerated parameters, we obtained the following optimal parameters:

1. Bootstrap = true
2. Bootstrap ratio = 0.4
3. Max features = 4
4. Max depth = 15

The grid search did not take into account the number of trees as the more trees the forest contains, the better it tends to work. The optimal depth is not the largest possible one as one could have expected it to be.

It is a well known result but still we checked that indeed a larger number of trees lead to better results, see figure 4.

3.5.2 Adaboost

We want to see if a boosting method can be useful in order to explain our data. Therefore, we have tried an `Adaboost Classifier`, with a `Decision Tree classifier` as a base estimator.

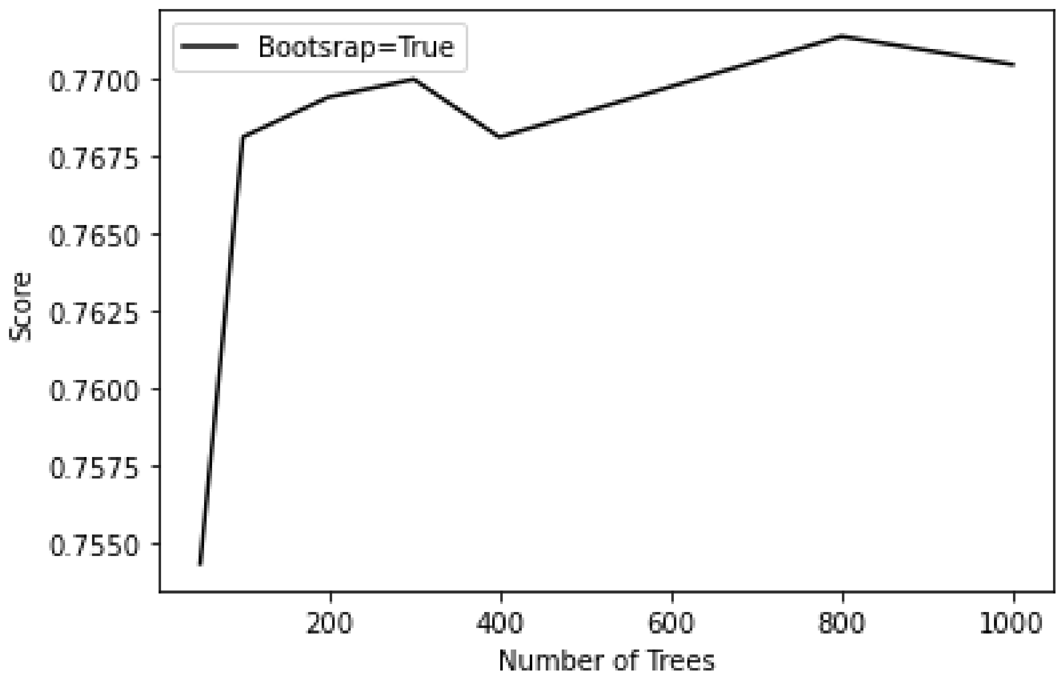

Figure 4. Score with number of trees

For doing this, 200 `Decision Tree classifiers` with a maximum depth of 10 are used.

The results are quite close to the `Random Forest classifier`, the cross validation score is equal to 0.7581 and the Kaggle score is equal to 0.8428

3.5.3 Voting Classifier

"Unity is strength", this is what we are going to see with the `Voting` classifier. Indeed, the idea is to assemble the models that worked the best on the cross validation score[3], and make the prediction based on how confident they are when predicting a class.

The classifier will be composed of all the classifiers we presented previously, in this section. Let m_i be the ith model and N, the number of models. The prediction is done as follows:

$$argmax_j \sum_{i=1}^{N} \mathbb{P}(c = j | m_i, \theta)$$

Where $\mathbb{P}(c = j|m_i, \theta)$ quantifies how certain the model m_i classifies the data θ as representing the class $c = j$.

This give us an accuracy of 0.8618 in our CV and 0.9343 on kaggle. Meaning that indeed aggregating results of different estimator lead us to better predictions!

[3]In particular KNN, RF, DNN, AdaBoost and SVC

3.6 Performance summary

Below are presented the models that gave good scores, using the `tsfresh` extracted features, using accuracy as scoring.

Model	Mean CV score	Pub. Kaggle	Weighted score	Pri. Kaggle
OVR Polynomial SVC	0.8350	0.8471	0.8327	0.8714
RandomForest	0.7926	0.8686	0.7995	0.8657
DNN	0.8289	0.9285	0.8379	0.9067
KNN	0.7569	0.8400	0.7644	0.8415
AdaBoost	0.7917	0.8428	0.7658	0.8428
VotingClf	0.8618	0.9343	0.8683	0.9105

The third column corresponds to a weighted sum of the *mean CV score* and *Public kaggle score* computed as follows:

$$\frac{3500}{3500+350} \times \{cv\ score\} + \frac{350}{3500+350} \times \{Public\ kaggle\ score\}$$

The idea of that score is to take into account the results given by the public score on kaggle and our cv score in order to be closer to our real accuracy score on unknown data and be less biased[4], the cv score is computed with 3500 samples and the one of kaggle with 350 samples therefore using a weighted sum like that seems to make sense. This is the score that we will use in order to choose our final model!

By looking at the result we can see that the *mean CV score* was really a lower bound compare to the value of the private score on kaggle, this is easily explainable by the fact that two subjects has lot of N/A and when one of these subject is the test fold, it give quite bad results. Also by the fact that the training was done on smaller part of the training data set.

We can notice that the `Random Forest` model did a good score on the `Kaggle` set. However, its score is a lot worse on the cross validation set. At first glance we though that it was just lucky but by looking at the private score it seems quite close to the one of the public score. On the other hand, the `OvR` strategy on a `Polynomial SVC` model gave consistent scores: the cross validation score and the `public Kaggle` score are very close from each other, which can be expected that this model would perform the same on the entire set.

`Adaboost`, in general, gave worse results than other ensemble method such as `Random Forest` and `voting`. This can confirm that we are likely to improve our score by reducing the variance instead of trying to reduce the bias.

Regarding `DNN`, is the best non-ensemble learning by looking at the *weighted score* (and even on the private test set of kaggle). It helps a lot the voting classifier to achieve great score.

Finally the voting classifier method is the one that have the highest **weighted score** and therefore the one that we will choose as final model!

[4]Note that true by looking at the private score of kaggle

4 Conclusion

For the final leaderboard, we used the method which gave the best results **locally**. Therefore, the `Voting classifier` has been chosen. This is the model which has given the best score among all the other groups on the final test set.

To conclude, the project helped us to understand the importance of data processing: indeed, if we were to tell you which feature of the solution contributed to these results, this is clearly the data extraction. Indeed, with no feature extraction, whatever the model we would use, we would have ended up with a model that gives poor predictions (less than 60%). Moreover, even though some models perform better than others, there is no very huge difference. The main difference was really in the way we were extracting features in our time-series.