ELEN0062 : Introduction to machine learning
Project 3 report: Human activity prediction

Aldeghi Florian(s183157) Baguette Brice (s181482)
Dasnois Louis (s181779)

September 28, 2023

1 First model selection

First, we decided to choose a classifier as model since we had to classify a population into multiples activities with sensor measurements. To start, we tried, k-NN, decision tree, neural network with MLPClassifier, logistic regression and a random forest. With basic cross-validation on samples, as implemented in sklearn, we got quickly realised that we were getting biased results (over 90% in cross-validation, but only around 50% when submitting), so we switched to our own cross validation method based on subjects.

We obtained biased results that are in table 1 (each algorithm is used with the default parameters of sklearn)

k-NN	Decision tree	Logistic regression	Neural network	Random forest
0.8526	0.7914	0.1483	0.2971	0.9486

Table 1: Performances of learning algorithms with sklearn 5-fold cross-validation.

Even if results are biased, we can already see that three algorithms look way better than the others, although this can be because no attempt was made at tuning the hyper-parameters.

We used a 5-fold cross-validation where each fold is all the data of one subject. Since the objective is to test the model on other subjects, it makes more sense to do it in this way. Otherwise, we use the data of a subject to build the model and after that, we test this model on the same subject. It totally overestimates the accuracy of the model.

Results of the models we tested with our cross-validation method are in table 2.

k-NN	Decision tree	Logistic regression	Neural network	Random forest
0.4668	0.3313	0.0853	0.1631	0.4780

Table 2: Performances of learning algorithms with 5-fold cross-validation based on subjects rather than samples.

After our first few tests, we decided to go with k-NN and random forest, since they are very good "out of the box". We dropped decisions trees as they are already used and give better results within random forests. Then, we tried varying hyper-parameters for these two algorithms. We represented the perfomance of K-NN and random forest according to their parameter (see Figure 1 and Figure 2).

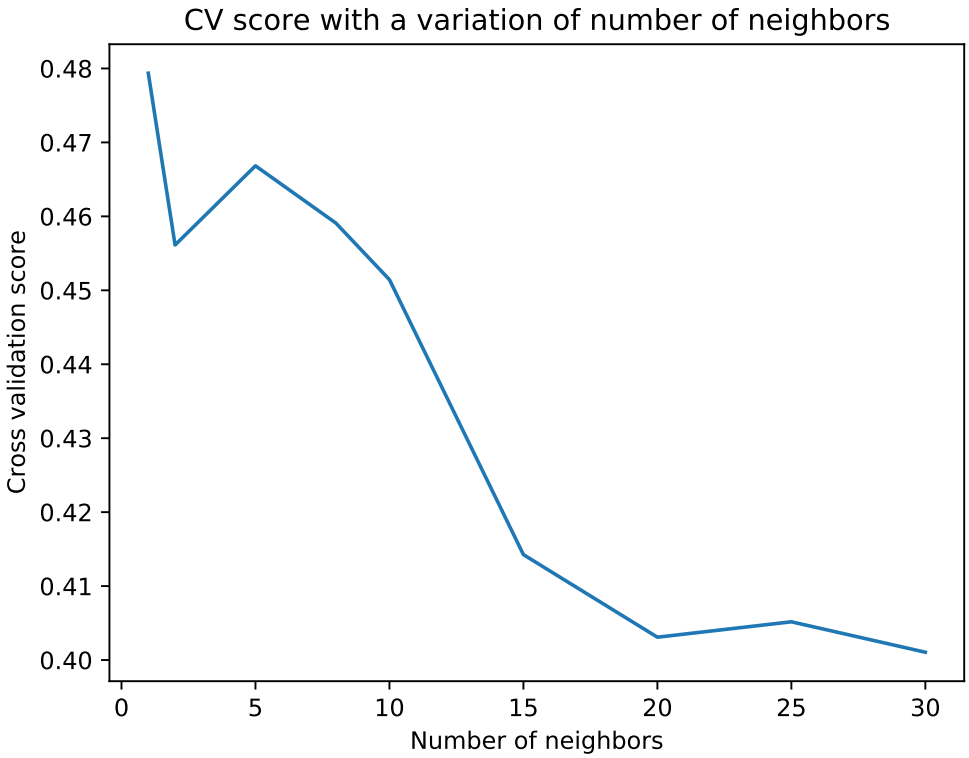

Figure 1: Performance of k-NN with regards to n_neighbors

For Figure 1, we observe an under-fitting when the parameter k grows, which reduces the accuracy of the model. The maximum stays at about 0.48, which could be a good model to train and obtain better results.

For Figure 2, as we expected, greater the parameter, the more accuracy is obtained. It leads to greater calculation time but it is not a problem for us since we just need to propose the best model possible without constraints on efficiency.

At this moment, we kept these two models in mind and knew the optimal parameter to use. The next step is to improve the model and we had some idea to test like preprocessing in order to see which model could be improve the most.

2 Preprocessing data

Now that we know which models we want to use, we can start looking at ways to improve their performances. One way could be to preprocess the data. Indeed, we give the raw times series as input, which results in a huge number of features that are often very correlated with one another. Maybe there is a way to transform the input in such a way that the different classes are better separated in the feature space, and maybe we can do that with (a lot) less features than we have initially.

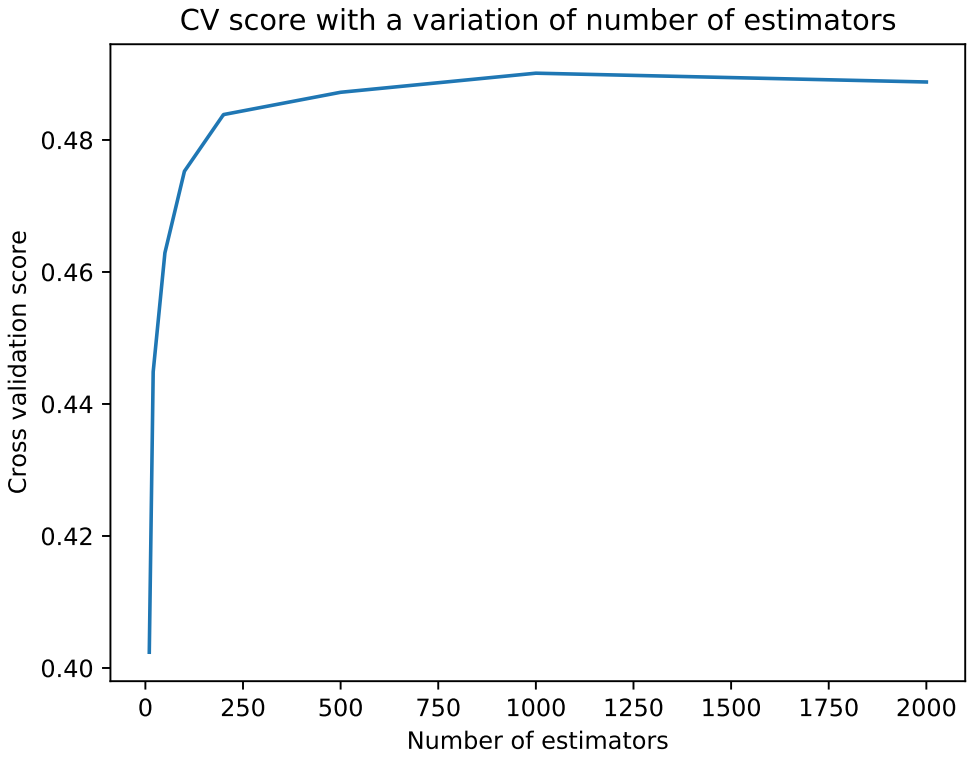

Figure 2: Performance of Random forest with regards to n_estimators

2.1 Normalization and missing values

First of all, we knew that there was missing data. We decided to replace each missing value by the average value a subject got for that feature (i.e. the average value the sensor measured at that specific time for that specific subject).
We also tried to normalise the data in a way that makes it less dependent on each specific subject. To do that, we computed the mean and the standard deviation of all measurements of the same kind (i.e. from the same sensor) from the same subject, and used that to normalise the data. For example, for each bpm measurement for subject one, we subtract the mean bpm of subject one across all measurements, then divide by the standard deviation of the heart rate of subject one across all measurements. Doing both of these improved the results slightly for k-NN, but not for random forest, (see table 3) so we decided to continue without. In hindsight, we probably should have tried that for a linear method, such as the logistic regression, as they work much better when the input is normalized. We didn't think of this at the time, but a quick test reveals that we would have obtained a cross-validation score of 0.5352, which is actually pretty good.

k-NN	0.4668
Random forest	0.4780
k-NN + normalization	0.4865
Random forest + normalization	0.4461

Table 3: Cross-validation scores of k-NN and Random forest on raw data and on normalized data where missing values have been taken care of.

2.2 Fast Fourier transform

After doing some research in the litterature, we came across an article (`https://www.frontiersin.org/articles/10.3389/fbioe.2020.00664/full`) talking about a very similar problem. They were also trying to do activity recognition based on time series coming directly from sensors. Before giving the data to their models, they were extracting different features from the time series and from their Fourier transforms. We tried following their approach. To start, we simply applied a fast Fourier transform on each input time series, without doing any feature extraction. This didn't achieve much for k-NN, but resulted in a huge performance jump for random forests, as can be seen in table 4.

k-NN	0.4668
Random forest	0.4780
k-NN + FFT	0.4196
Random forest + FFT	0.6775

Table 4: Cross-validation scores of k-NN and Random forest on raw data and after fast Fourier transform.

2.3 Feature extraction

Our next approach was to extract some features from the time series (such as their mean, standard deviations, etc) and use that as input to our models. Since there was no way for us to think of all the features that could be interesting by ourselves, we decided to used a library that specialises in feature extraction from time series, namely `tsfresh`. At first, the features we extracted were the ones included in their `MinimalFCParameters`. Once again, this resulted in a huge improvement for random forest, but not for k-NN (see table 5).

k-NN	0.4668
Random forest	0.4780
k-NN + feature extraction	0.4203
Random forest + feature extraction	0.7123

Table 5: Cross-validation scores of k-NN and Random forest on raw data and after feature extraction.

3 Support vector machines

Since our models were getting pretty good, our next idea was to try stacking different models. Stacking works best when the models are very different. However, stacking random forests with k-NN was taking way too long, so we tried looking for other methods. Since our input was now very different, we had a go at linear models again, in particular SVM since the number of features was now small enough to try. We quickly found out that the cross-validation results were much better if we applied the normalization steps and took care of the missing values (as described in section 2.1) before doing the feature extraction. Then we got to tuning the C regularization parameter, both with L1 and L2 norms. The results are in figures 3 and 4. Using L1 norm gives much better results, and we settled with a value C equal to 200, giving a cross-validation score of 0.7346.

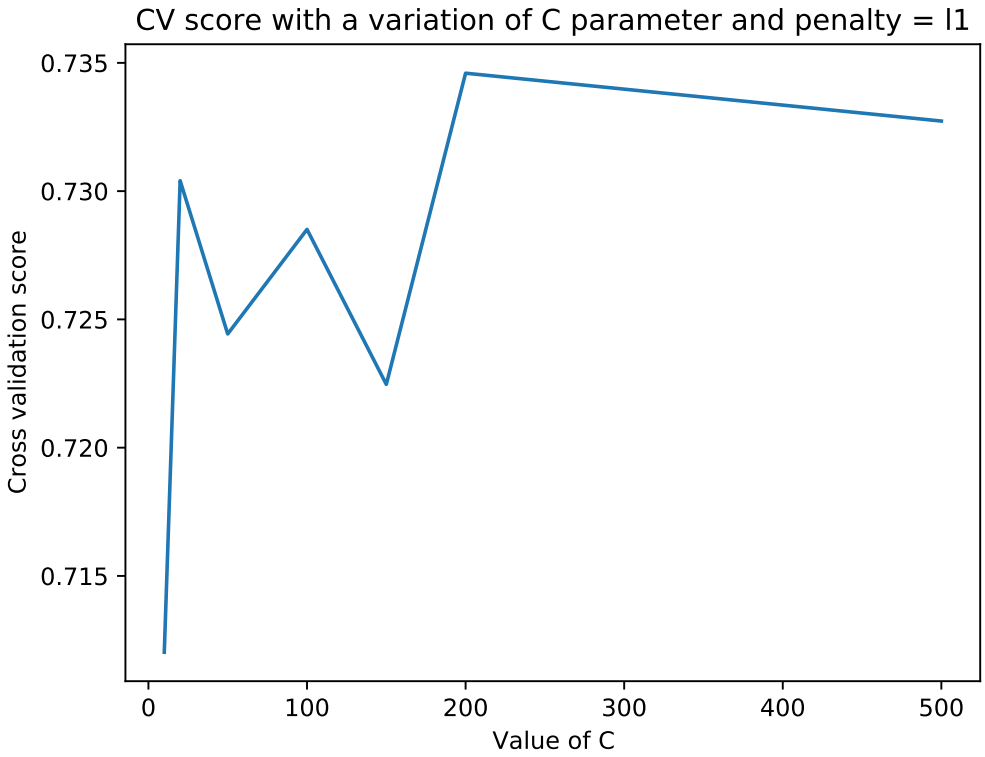

Figure 3: Performance of SVM classifier with regards to C, using L1 norm

4 Stacking

Next, we tried stacking our two best models (SVM and random forest). To do this, we used the `StackingClassifier` class provided by sklearn. We use our SVM and random forest with extracted features (normalized for SVM) as the base models. The default meta model is a logistic regression, but it had some trouble with convergence, so we decided to used another random forest instead, as random forests are very powerful and require little to no hyper-parameter tuning (the default value for `max_features` is usually pretty good). This meta model uses as input the predicted probabilities of the base models, and is trained using cross-validation. This method achieved a cross-validation score of 0.7486. This is the best cross-validation score we were able to achieve, but we chose not to use this model in the end for reasons explained in the next section. In addition to the concerns raised in the next section, this scores contains selection bias, as the parameters used by the base models (which correspond to the ones chosen in the previous sections) were selected using the whole training set, outside of our cross-validation loop.

5 Final model selection

A natural choice would be to select the model that achieves the best overall cross-validation score. This is what we would do if we had no further data, and if none of the scores were biased. However, in our testing, we saw huge variations in the scores obtained for each cross-validation fold. Accuracies went up to 90% for some subjects, but less than 50% for others. These variations tend to be consistent for each type of model: for example, random forests with feature extraction are consistently good at predicting the activities for subject 1, but not for

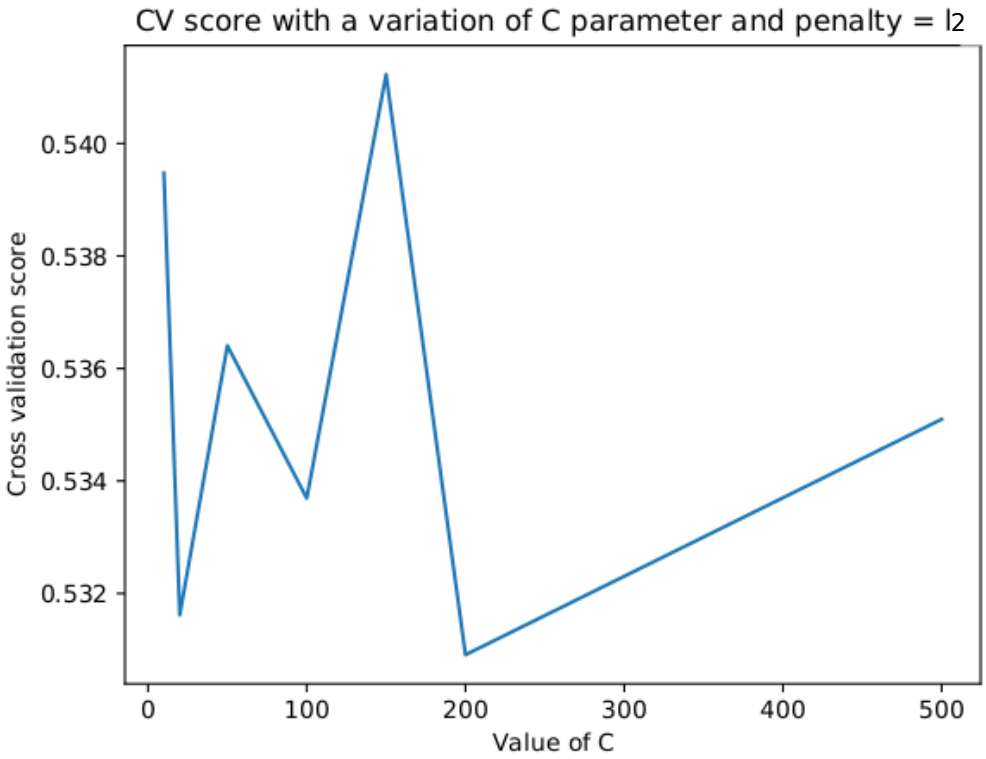

Figure 4: Performance of SVM classifier with regards to C, using L2 norm

subject 4, while SVM is more balanced. In addition to that, some models give much better scores when submitted to Kaggle (and evaluated against the 10% of the test set) than other models that are better or equal in cross-validation. This seems to indicate that those models performing well on the reduced test set are good at predicting activities for the subjects in the test set. With random forests consistently doing multiple percent better than SVM on submission, the chances of it being a fluke are low and, assuming all 3 test subjects are present in the reduced test set, we can conclude that random forests are the best models we have for predicting the activities of subjects in the test set.

6 Fine tuning random forests

We weren't able to get significant improvements by playing with the `max_features` parameter, so we left it as default, but we could still try to select the best features to extract before using the random forest. We tried to find a minimal set of features that performed well in cross-validation (as well as when submitted, but that was mainly as a last check). We ended up with the following six features: median, standard deviation, spectral centroid, mean absolute difference between consecutive values, maximum and minimum, giving a cross-validation score of 0.7137. This is our final model, that we chose to use for the final score.

7 Recapitulative table of the performances of each model

All scores in Table 6 are computed with our 5-fold cross-validation.

learning algorithm:	k-NN	Decision tree	Logistic regression	Neural network	Random forest
score with sklearn CV	0.8526	0.7914	0.1483	0.2971	0.9486
score with our CV	0.4668	0.3313	0.0853	0.1631	0.4780
score with normalization of input and missing value	0.4865	/	/	/	0.4461
score with FFT on input	0.4196	/	/	/	0.6775
score with feature extraction	0.4203	/	/	/	0.7123

Decision tree	0.3313
Neural network	0.1631
Logistic regression	0.1483
Logistic regression + normalization	0.5352
k-NN	0.4668
k-NN + normalization	0.4865
k-NN + FFT	0.4196
k-NN + feature extraction	0.4203
Random forest	0.4780
Random forest + normalization	0.4461
Random forest + FFT	0.6775
Random forest + feature extraction	0.7123
SVM + normalization and feature extraction	0.7346
Stacking (SVM + RF)	0.7486 (biased)
Final model	0.7137

Table 6: Cross-validation scores of each model attempted.