ELEN-0062: Introduction to Machine Learning

Project 3 - Human Activity Prediction

Maxime Goffart
180521

Olivier Joris
182113

Academic year 2021 - 2022

Contents

1 Introduction

In this report, we will present and justify the different techniques we used for the third project of the course. The goal is to provide the reasoning behind our choices and to present the results obtained on Kaggle.

2 Data pre-processing

We started by analyzing the data in order to pre-process them. Indeed, pre-processing is important to have good results when doing classification. This section includes the observations we made about the dataset and different approaches we tried in order to solve the problems linked to the dataset.

2.1 Filling missing values

The learning set contained a certain number of missing values represented by the `-999999.99` value. Thus, we started by looking at several methods proposed by the scikit-learn library[1] to replace these missing values. We tried several of these methods and finally decided to use the `KNNImputer` which completes missing values using k-nearest neighbors. Each missing value is imputed using the mean value from the nearest neighbors found in the training set.

2.2 Observations about the samples

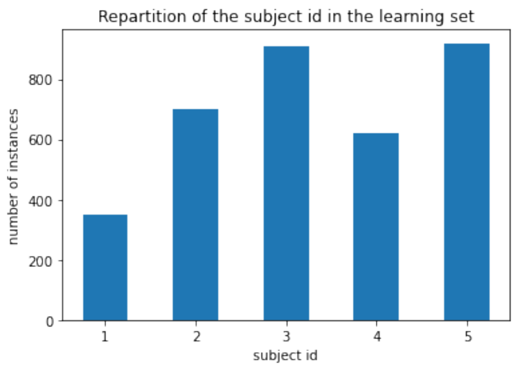


Figure 1: Repartition of the different subject ID's in the learning set.

We observed that the learning set seems to be balanced. The activity ID's are well distributed. They are each composed of 250 instances of the subjects in the training set. It is good because if this was imbalanced we could have a model which always predict the value of the majority class. This model would have a nice score on our learning set but when trying it on the testing set, it would lead to bad results.

However, when looking at the subject ID's of the learning set, we observed that some subjects have been less used to perform measurements. This can be observed on the figure 1. This can lead to overfit because all subjects have not the same behavior when perfoming an activity and the sensors measurements are correlated to one subject. We tried to make the sample relative to sensors and did feature selection on them to handle this problem. It was then difficult to

[1]`https://scikit-learn.org/stable/modules/impute.html`

predict the activity of the input subjects because the samples were not anymore linked to one subject. We thus, later in the project, realized that when doing cross-validation to measure the accuracy of our models, we had to split our training and testing sets according to the subject ID's in order to see if it can predict the activity of unseen subjects. We also added these subject ID's as feature.

2.3 Feature selection

The fact that the dataset was really huge (3500 * 31 * 512 values) and some values irrelevant, motivated us to perform features selection. We decided to perform feature selection using `SelectFromModel`[2]. It is performing ranking based on one estimator and then remove the less ranked features recursively. We chose the `ExtraTreeClassifier`[3] as estimator for this method. It is similar to a random forest classifier and only differ in the manner it is constructed which lead to the creation of multiple de-correlated decision trees. This seems to be a good model to rank the features because we have seen in the theoretical course that decision trees are nice to discriminate features but are too much related to the dataset. This is why the way the Extremely Randomized Trees are constructed reduce the variance of the dataset and reduce the chance of overfitting.

The implementation of this part can be found in the `feature_selection.py` python script.

2.4 Feature extraction

Because feature selection did not improve the scores of our model, we decided to extract new features from the ones of the dataset including the subject ID's which were not used in the initial dataset. First, we performed some statistics on the time series (mean, median, standard deviation, etc). We had the idea to perform these extractions because it could reduce the number of features and keep the meaning of the data. A null variance implies that the activity is a static activity, the number of negative values implies that it is an activity in a cold environment, etc. We thought that a good model could understand the meaning of these statistics and it improves our score.

Thereafter, because we did not had signal representation knowledge, we investigate[4] about the features we could add to feed our model and found that these statistics could also be expressed in a frequency domain instead of time domain. This is easy to do thanks to the discrete Fourier transforms which are integrated in the numpy library. We also extracted some purely physics features such as the energy of one signal, the average resultant acceleration, and the signal magnitude area. We also tried some libraries[5] which have this feature extraction as purpose but it was very slow and we thus quickly forgave them.

Our different attempts of implementation can be found in the `feature_extraction_v*.py` python scripts where * refer to the version of this feature extraction.

[2]`https://scikit-learn.org/stable/modules/feature_selection.html`
[3]`https://www.geeksforgeeks.org/ml-extra-tree-classifier-for-feature-selection/`
[4]`https://towardsdatascience.com/feature-engineering-on-time-series-data-transforming-signal-data-of-a-smartp`
[5]Like `sktime` and `tsfresh`.

3 Performance measure

We initially performed cross-validation inside our training set splitting it blindly. This lead us to high scores but when posting our predictions on Kaggle, the score inordinately decreased. Some results we present in the section 4 are based on this technique because we only, late in the project, realized that another method could be used to have more realistic scores. As introduced in the section 2.2, this method consists in doing cross-validation but taking into account the subject ID's. We decided, at each iteration of our k-cross-validation, to split our training set randomly such the split training set is composed of three of the subject ID's samples and the split testing set of the two remaining. This method is better to approximate predictions on the real testing set because this set is composed of never seen subjects.

4 Studied models

4.1 K-nearest neighbors

First, we decided to use the K-nearest neighbors method because it was the one provided by the pedagogical team with the assignment. Yet, the reasons that made us trying multiple versions of this technique are the facts that it is easy to interpret and to use.

4.1.1 1-NN

Our first submission was the one provided with assignment[6] that was generated with the 1-nearest neighbor model[7].
This submission yields a score of 0.52 on Kaggle.

4.1.2 25-NN

Based on 4.1.1, we decided to increase the value of K of the K-nearest neighbors method to 25. The motivation behind the choice of this value for K was theoretical. Since the dataset is huge and can contain errors in the measured values, we though that increasing the value of K would increase the precision of the model because each prediction will not depend on a single neighbor that could have been misclassified.
This technique is implemented in the file `knn_basic_25.py`. This submission yields a score of 0.54 on Kaggle.

4.1.3 55-NN and 49-NN

We decided to keep using the K-NN method to study how well it could perform on the assignment. We decided to study the accuracy of the KNN technique by using a 10-fold cross validation technique with the negative log loss function as a scoring measure. We obtained the following graph of accuracies depending on the value of K:

[6]File `example_submission.csv`.
[7]Files `toy_script.py` or `knn_basic_1.py`.

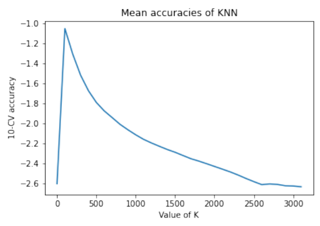


Figure 2: 10-CV accuracy of KNN depending on value of K

We could see that there is a peak. By using a divide and conquer technique, we found that the peak was obtained for values of K around 50. Thus, we decided to study the accuracy in a small range of K values around $K = 50$. We obtained the following graph:

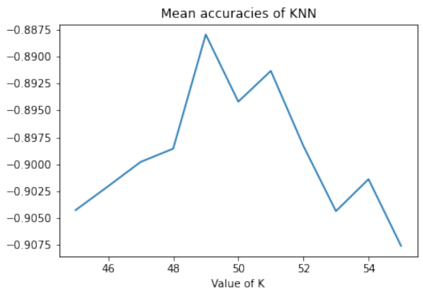


Figure 3: 10-CV accuracy of KNN depending on value of K

Based on the divide and conquer approach performed, we decided to use 55 neighbors. The choice of this value was motivated by the fact that we observed a peak around this value.
This submission yields a score of 0.53142 on Kaggle and the implementation of it is inside the file `knn_basic_55.py`.

Then, based on the graph of figure 3, we decided to use 49 neighbors. The choice of this value was motivated by the fact that it yields the highest score on the learning set by using a 10-fold cross validation strategy with the negative log loss function as a scoring measure.
This submission yields a score of 0.52571 on Kaggle which is less than the one for 55-NN. Yet, we expected a higher score based on the graph of figure 3. The implementation is available inside the file `knn_basic_49.py`.

Based on the two previous results, we came to the conclusion that using "vanilla" K-nearest neighbors was not sufficient. Thus, we decided to find new techniques explained in the following sections.

4.1.4 Multiple K-nearest neighbors models

We took a look back at the slides provided with the assignment and we focused on the features (sensors) of the datasets. We noticed that the features are not in the same units. Thus, we thought about using multiple 1-nearest neighbor models with one 1-NN model per feature. This idea was motivated by the fact that we though we would obtained a better accuracy by using one 1-NN per feature because measurements in different units would not be mixed.
We implemented this approach in the file `knn_splitted_1.py`. This yields a score of 0.52 on Kaggle which is the same as the toy script provided with the assignment.

4.1.5 Mutiple K-nearest neighbors models and samples modification

After the disappointing result obtained in 4.1.4, we decided to take a better a look at the datasets. We noticed that some samples were vectors full of -999999.99. We though that these measurements would badly influenced the models. Thus, we decided to replace each sample that was a vector full of -999999.99 by a vector full of 0.
We implemented this approach in the file `knn_splitted_1_filtered.py` and the score obtained on Kaggle was 0.53142.

4.1.6 Conclusion on K-nearest neighbors

After all the different tries perform using K-nearest neighbors models, we came to the conclusion that K-NN was not sufficient for this project.
The codes for the different plots are available inside the Jupyter notebooks `knn_basic.ipynb` and `knn_splitted.ipynb`.

4.2 Decision trees

Secondly, we decided to use the decision tree method because its interpretability is very easy and the effect of the parameters is easy to understand.

4.2.1 Decision tree with min_sample_split=2

First, we try a very simple decision tree with the parameter `min_sample_split` set to 2. We decided to choose this value for that parameter because we wanted to have a first result for the decision tree on which we could based our further study of the approach.
This model has been implemented in the file `dt_basic_2.py`. It has an accuracy of 0.8074285714285715 measured with a 10-fold cross validation strategy and provided a score of 0.38571 on Kaggle.

4.2.2 Decision tree with post-pruning

We decided to study the accuracy of the decision tree depending on the value of the `min_sample_split` parameter. In order to do, we decided to study the accuracy for values of the parameter ranging from 2 to 15500. We obtained the following graph:

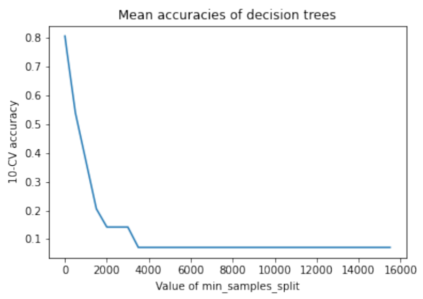


Figure 4: Accuracy of decision tree depending on value of parameter

Based on figure 4, we can see that the value of the parameter, that results in the highest accuracy, is smaller than 2000. Thus, we decided to study the accuracy of the model for values of the parameter between 2 and 2000. We obtained the following graph:

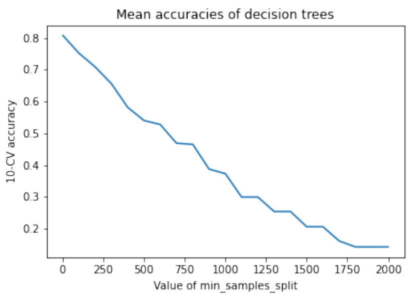
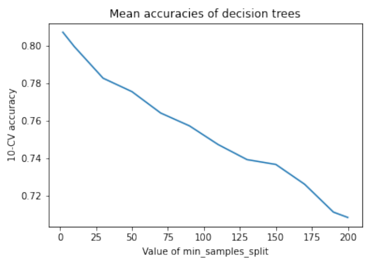
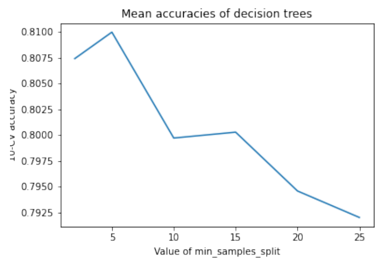
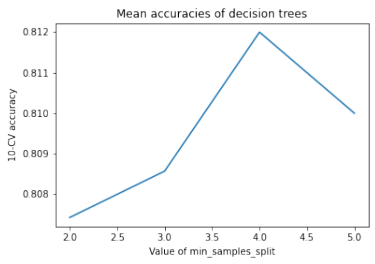


Figure 5: Accuracy of decision tree depending on value of parameter

We continue this approach and study the accuracy of the model for values of the parameter [2,200], [2,25], and [2,5]. We obtained the following graphs:

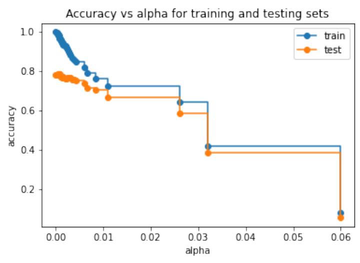


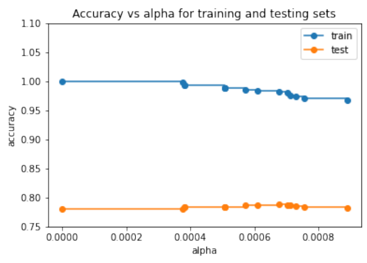




It turns out that using that using a small value for the parameter `min_sample_split` provides a higher accuracy using a 10-fold cross validation strategy. Yet, we might overfit on unseen data. Thus, we decided to study the overfitting. We based ourselves on the approach presented in the documentation of scikit-learn[8].

We split the learning sample in 2 using proportions 75% and 25% where the 75% will be our new learning set and the 25% will be our new test set. Then, we computed the path during the minimal cost-complexity pruning and we got the alphas for the sub-trees. Afterwards, we build a tree with the different values of alphas obtained in the previous step[9]. Finally, we computed the score of the LS and the TS on the different tress and plotted it. We obtained the following graphs[10]:





Based on the obtained data, we found out that the value of alpha that provides the highest score on the test set was 0.0006772486772486773.
So, for our next model, we built a decision tree with this particular value of alpha for pruning.

[8]https://scikit-learn.org/stable/auto_examples/tree/plot_cost_complexity_pruning.html#
[9]We actually build only a fifth because there was too much possible alphas.
[10]The graph on the right is a zoom on the graph on the left.

This particular model provided an accuracy of 0.8025714285714285 using a 10-fold cross validation approach and a score of 0.40285 on Kaggle, which is small improvement compared to the score 0.38571 obtained without pruning.
The analysis performed for this subsection is available in the Jupyter notebook `dt.ipnyb` and the implementation of the model in the file `dt_pruned.py`.

4.2.3 Decision tree with filtered data

As explained in section 2, the data provided is not ideal. Thus, we decided to try a decision tree with pre-processed data.
This model is implemented in the file `dt_filtered.py`. This particular model provided an accuracy of 0.7928571428571429 using a 10-fold cross validation approach[11] and a score of 0.43428 on Kaggle. This is an improvement compared to the 0.40285 obtained with pruning.

4.2.4 Conclusion on decision trees

After the study performed on decision trees, we concluded that decision trees were not adequate for this project due to the complexity of the dataset. We decided to move to more complex models.

4.3 Random forests

Next, we decided to use random forests because they should provide better accuracies than decision trees. Yet, their interpretability is still easy.

4.3.1 Random forest with 40 trees

First, we decided to try a few forests and measuring their accuracies using a 10-fold cross validation strategy. We obtained the following results:

#	**Number of trees**	**min_sample_split**	**10-CV accuracy**
1	10	200	0.7354285714285714
2	50	200	0.7702857142857142
3	50	100	0.8608571428571429
4	100	50	0.9062857142857143

Table 1: 10-CV accuracy of different random forests

Based on table 1, the first conclusion we draw was that increasing the number of trees and diminishing the value of `min_sample_split` increases the accuracy measured with a 10-fold cross validation approach.

Then, we decided to study the accuracy of the forest based on the number of trees with a fixed value for `min_sample_split` (arbitrarily set to 25). We obtained the following graph:

[11] Available in the Jupyter notebook `dt_filtered.ipynb`.

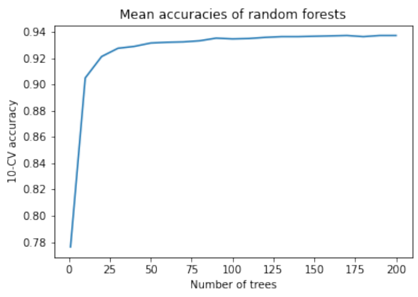


Figure 6: Accuracy of random forests depending on number of trees

Based on figure 6, we can see that, for a number of trees greater or equal to 30, we reach a plateau. Now, we will study the accuracy of random forests depending on the value of `min_sample_split` for a number of trees fixed to 40. We obtained the following graph:

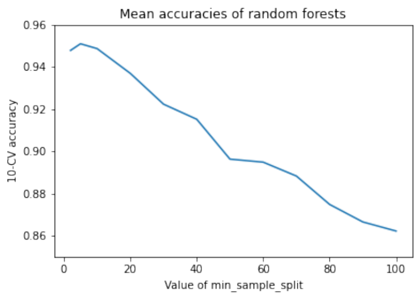


Figure 7: Accuracy of random forests depending on `min_sample_split`

As we can see on figure 7, the smaller the value of `min_sample_split`, the higher the accuracy using a 10-fold cross validation measure.

Based on a what we have observed, we decided to try a random forest composed of 40 trees with `min_sample_split` set to 25.
This model is implemented in the file `rf_40_25.py` and the analysis performed is available in the Jupyter notebook `random_forest.ipynb`.
This model provided an accuracy of 0.9288571428571428 using a 10-fold cross validation strategy and a score of 0.52 on Kaggle. This score on Kaggle is the same as the one obtained for 1-NN. Yet, this is an improvement compared to the scores obtained for the decisions tree in the part 4.2 of this report. This is what was expected because random forests are ensemble methods that are using multiple trees.

4.3.2 Random forest with feature selection

As explained in section 2, the data provided is not ideal. Thus, we decided to try a random forest with pre-processed data. We kept the same values for the number of trees and `min_sample_split`.
This model is implemented in the file `rf_filtered.py`. This particular model provides an accuracy of 0.9074285714285713 using a 10-fold cross validation approach[12] and a score of 0.48285 on Kaggle. This is less than what we obtained for a random forest without filtering the data. This is certainly linked to the fact that our first approach for feature selection was not ideal as explained in section 2.

[12] Available in the Jupyter notebook `random_forest_filtered.ipynb`.

4.3.3 Random forest with feature extraction

The, we used the feature extraction of the section 2.4 and obtained these results locally using the cross-validation and splitting according to the subject ID's:

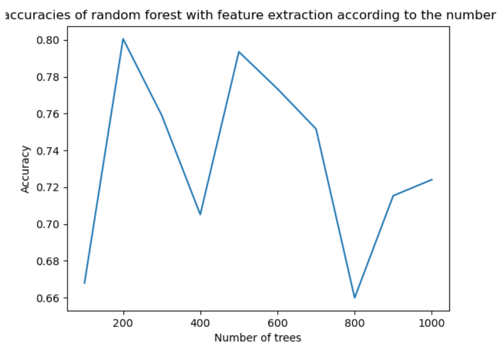


Figure 8: Accuracy of random forests depending on the number of trees.

The way the features were generated is well appropriate to trees. They can, for example, split according to the feature which represents the variance of the timesteps. The tree can divide the ones having a null variance from the others which represents splitting static activities from dynamic ones. It is why we chose this model as final model. We have less chance to overfit because we understand our model unlike models which can become too complex to understand. It also performs the best score when doing cross-validation splitting according to the subject ID's which motivated this choice.

The implementation of this model is available in the `random_forest_feature_extraction.py` python script.

4.3.4 Conclusion on random forests

Random forests which is an ensemble method using "bagged" trees fill the trend of the trees to overfit and is thus well adapted to our feature extraction. As observable in the table of section 5, random forest is the model that provides the highest accuracy locally and on Kaggle.

4.4 Logistic regression

We tried to use the logistic regression model on our extracted features. Indeed, logistic regression being a linear classifier, it relies upon the quality of the features in order to perform good predictions. When trying this model, we observed a warning concerning the convergence saying that it was better to standardize the data. Thus, we did it but the logistic regression did not result in good predictions. It was really often predicting the running activity. Thus, we concluded that our features were too complex for this model.

The implementation of this part can be found in the `logistic_regression_feature_extraction.py` python script.

4.5 Multilayer perceptron

We tried to use single and multi layers perceptron classifiers as models, both with feature selection and feature extraction because the initial dataset was too huge to train this model.

The main parameters of this model are the number of neurons and the number of layers. Thus, we studied the model according to these two parameters.

4.5.1 Feature selection

The evolution of the model with feature selection according to the number of neurons and layers can be observed on the figures 9 and 10.

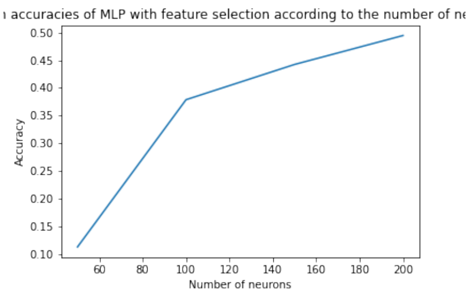


Figure 9: Accuracy of MLP depending on the number of neurons (single layer).

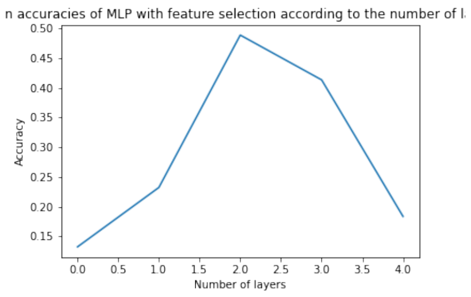


Figure 10: Accuracy of MLP depending on the number of layers with 50 neurons per layer.

The implementation of this part can be found in the `MLP_feature_selection.py` python script.

4.5.2 Feature extraction

The evolution of this model with feature extraction according to the number of neurons and layers can be observed on the figures 11 and 12.

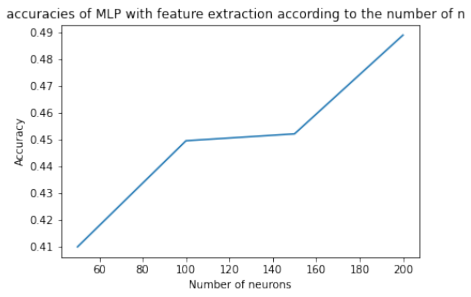


Figure 11: Accuracy of MLP depending on the number of neurons (single layer).

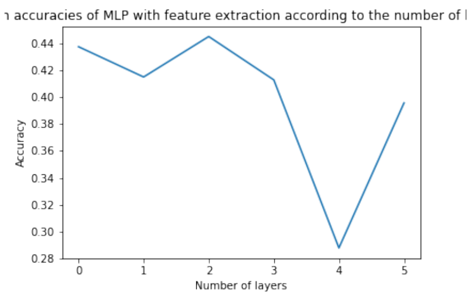


Figure 12: Accuracy of MLP depending on the number of layers with 50 neurons per layer.

The implementation of this part can be found in the `MLP_feature_extraction.py` python script.

4.5.3 Conclusion on Multilayer perceptron

We did not investigate much on multilayer perceptron classifier because we observed that this model was hard to tune. The graphics we presented in this section do not give real information about what the optimal parameters are.

4.6 Support vector machines

For completeness, we wanted to try support vector machines.

4.6.1 Support vector classification

This approach is based on the model SVC of scikit-learn[13]. We decided to study the accuracy of this model by splitting the learning set based on the ID's of the 5 subjects and by using feature selection.
For our first study, we decided to consider a wide range for the value of the regularization parameter[14]. We choose the range [1.0, 1000.0] has a starting point and obtained the following graph of accuracies depending on the value of the parameter:

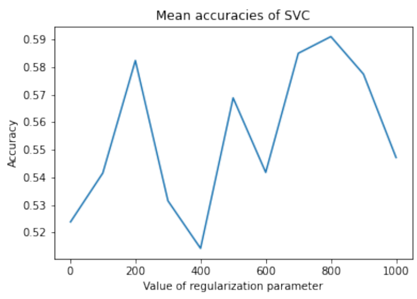


Figure 13: Accuracy of SVC depending on regularization parameter

As we can see on the previous plot, it seems there is not an obvious relation between the regularization and the accuracy of the model. Thus, we decided to study the accuracy of the model for values of the regularization in the range [1.0, 100.0]. We obtained the following graph:

[13]sklearn.svm.SVC.
[14]Denoted C in scikit-learn.

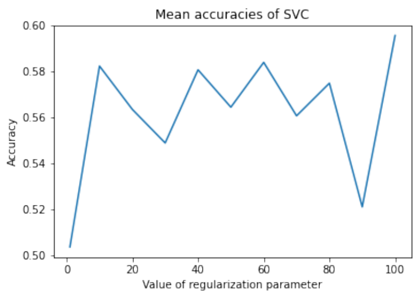


Figure 14: Accuracy of SVC depending on regularization parameter

Once again, it seems there is no obvious relation between the regularization and the accuracy. Yet, the values for the previous graphs were obtained as averages on 10 tries. We decided to increase the number of tries to 20. We obtained the following graph:

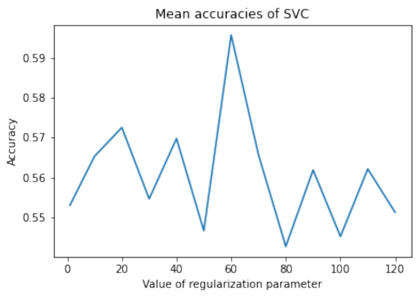


Figure 15: Accuracy of SVC depending on regularization parameter

The peak observed on the graph is for a value of the regularization parameter of 60. Thus, we decided to use a SVM (specifically `sklearn.svm.SVC`) with a value of 60 as the regularization parameter. This models has an accuracy 0.5956468368018641 locally obtained by splitting the learning set based on the ID's of the 5 subjects and by using feature selection[15]. This model gives a score of 0.49428 on Kaggle.
The study performed for this model is available in the Jupyter notebook `svc-feature-extraction.ipynb` and the implementation of the model in the file `svc-reg-60.py`.

4.6.2 Support vector classification - ν-SVC

This approach is based on the model ν-SVC of scikit-learn[16]. As always, we decided to study the accuracy of the model depending on the value of the parameter ν. To study the accuracy, we split the learning set based on the ID's of the 5 subjects and by calculating average accuracies over multiples splits. We used feature selection to reduce the size of the learning set.
We study the accuracy of the model for values of ν in [0.1, 0.6]. We obtained the following graph:

[15]The accuracies are averages over 20 tries.
[16]sklearn.svm.NuSVC.

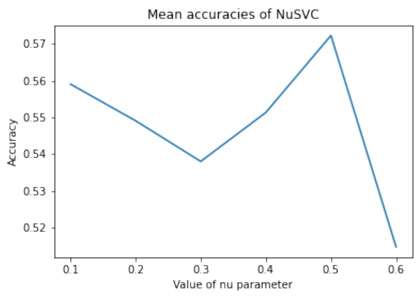


Figure 16: Accuracy of ν-SVC depending on value of ν

The peak observed on the graph is for a value of ν of 0.5. Thus, we decided to use a ν-SVC with $\nu = 0.5$. This model has an accuracy 0.5723277462764307 locally measured as explained before and a score on Kaggle of 0.56571.
The code for the generation of the graph is available in the Jupyter notebook `nusvc-feature-extraction.ipynb` and the implementation of the model in the file `nusvc.py`.

4.6.3 Conclusion on SVM

Support vector machines provide accuracies that are nowhere near the ones provided by random forests. Thus, we decided to stop our exploration of support vector machines and focus on random forests and improving our feature extraction technique.

5 Summary table

The following table summaries the techniques we have tried and the scores we have obtained on Kaggle.

Approach	Report section	Kaggle score
1-NN	4.1.1	0.52
25-NN	4.1.2	0.54
55-NN	4.1.3	0.53142
49-NN	4.1.3	0.52571
Multiple 1-NN	4.1.4	0.52
Multiple 1-NN with filtered data	4.1.5	0.53142
Decision tree - min_sample_split=2	4.2.1	0.38571
Decision tree with pruning	4.2.2	0.40285
Decision tree with filtered data	4.2.3	0.43428
Random forest - 40 trees - min_sample_split=25	4.3.1	0.52
Random forest - filtered data	4.3.2	0.48285
MLP - 1 inner layer - 500 neurons	4.5	0.48285
MLP - 1 inner layer - 100 neurons	4.5	0.43428
MLP - 1 inner layer - 100 neurons - feature extraction	4.5.2	0.22
25-NN with feature extraction	Not presented (too poor)	0.45714
Random forest - 1000 trees with feature extraction	4.3.3	0.87428
Logistic regression with feature extraction	4.4	0.48285
SVM with feature extraction	Not presented (too poor)	0.07142
SGD with feature extraction	Not presented (too poor)	0.07142
SVM with feature extraction	4.6.1	0.49428
NuSVC with feature extraction	4.6.2	0.56571
Random forest - 1000 trees with improved feature extraction	4.3.3	0.91142
Random forest - 400 trees with improved feature extraction	4.3.3	0.91714

Table 2: Approaches with respective Kaggle scores

6 Conclusion

This project allowed us to apply a lot of the techniques and models studied in the theoretical course. It helped us to understand certain issues related to a classification problem.

The model we chose seemed to well perform on the public leaderboard but a bit overfitted it. However, this model is the second best one we had submitted with a difference of 0.004 of accuracy comparing it to our best model. We thus not regret our choice.