Final Project Assignment

ECG Classification for Arrhythmia Detection

Cyprien TORDO
6071112 - ctordo@tudelft.nl
Émilien COUDURIER
6066895 - ecoudurier@tudelft.nl

Summary

In this Final Project Assignment, our goal is to develop a Machine Learning model to detect various types of abnormal heartbeats (arrhythmias) from heart activity recorded in the form of an electrocardiogram (ECG).

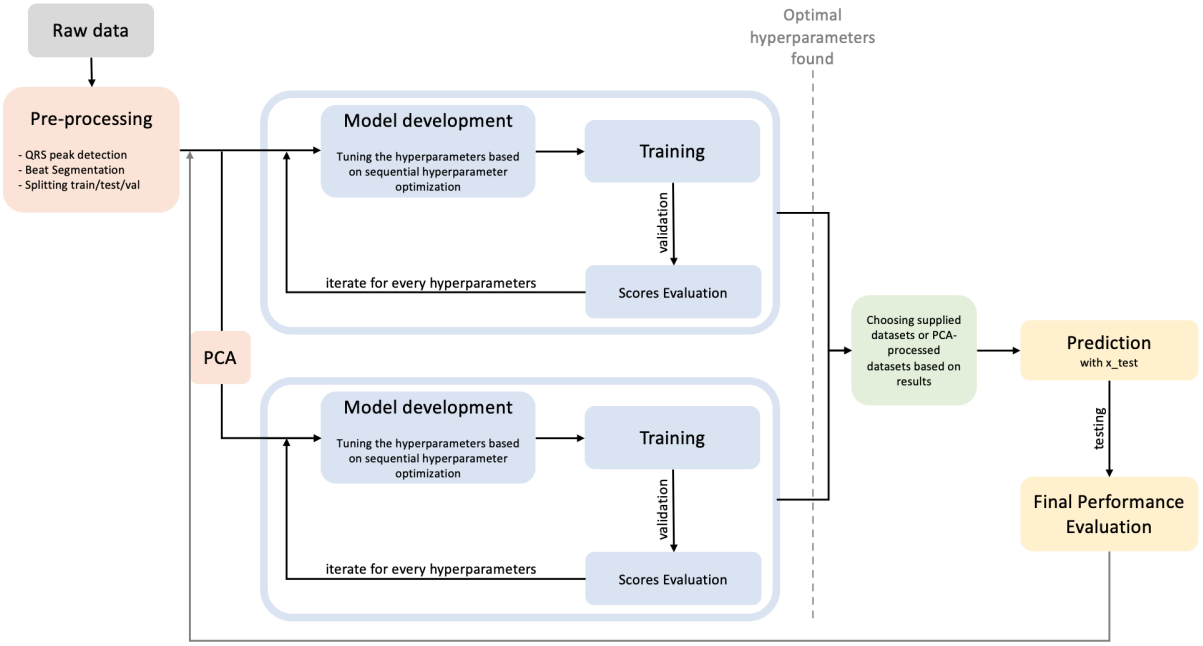
In concrete terms, we have a total of 109451 ECG samples. Each ECG sample is an ordered list of 250 numerical values, and is labeled with a specific type of arrhythmia.
In our application, there are 15 different types of arrhythmia, holding labels *0* to *14*.

In practice, the aim of this project is to predict the arrhythmia type to which an ECG sample belongs, based only on the 250 sample values.

As the role of this project is to put into practice all the theory assimilated during the quarter, we chose ECG sample classification as our subject, since the medical field seemed to us to be a very good concrete example of how Machine Learning can be put to good use.

This report presents our intellectual path and the methods we developed to achieve it, and describes the steps we took to orchestrate the whole project.

2. Detailed ML pipeline



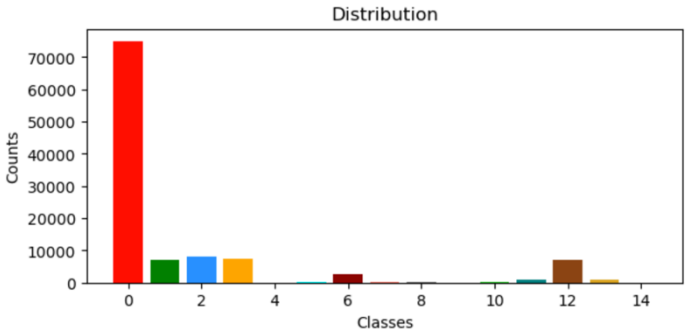
All the different stages presented in this workflow figure are described in the corresponding subsections of the report.
We will detail the motivations, developments and consequences of each of these parts and their coherence when put together.

3. Task I : Data pre-processing

3.1 Visualizing data

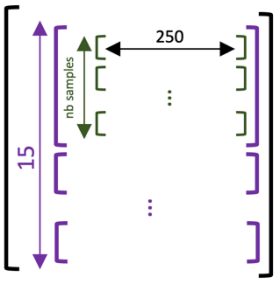
We started the project by performing a few non-required operations that gave us a good representation of the shape of the data we will be manipulating and classifying.

First, we counted how many ECG samples each of the 15 arrhythmia types had and plotted these counts on a bar plot. As expected, the type 0 (neutral) category is far more present than the others.



This observation gave us the idea of studying, for the models we'll be training, not only general accuracy but also accuracy without the neutral category (we'll come back to this later in the report).

Next, we segregated the ECG samples according to their arrhythmia type. Specifically, we created the ***ECGbyCat*** list, in the following form:



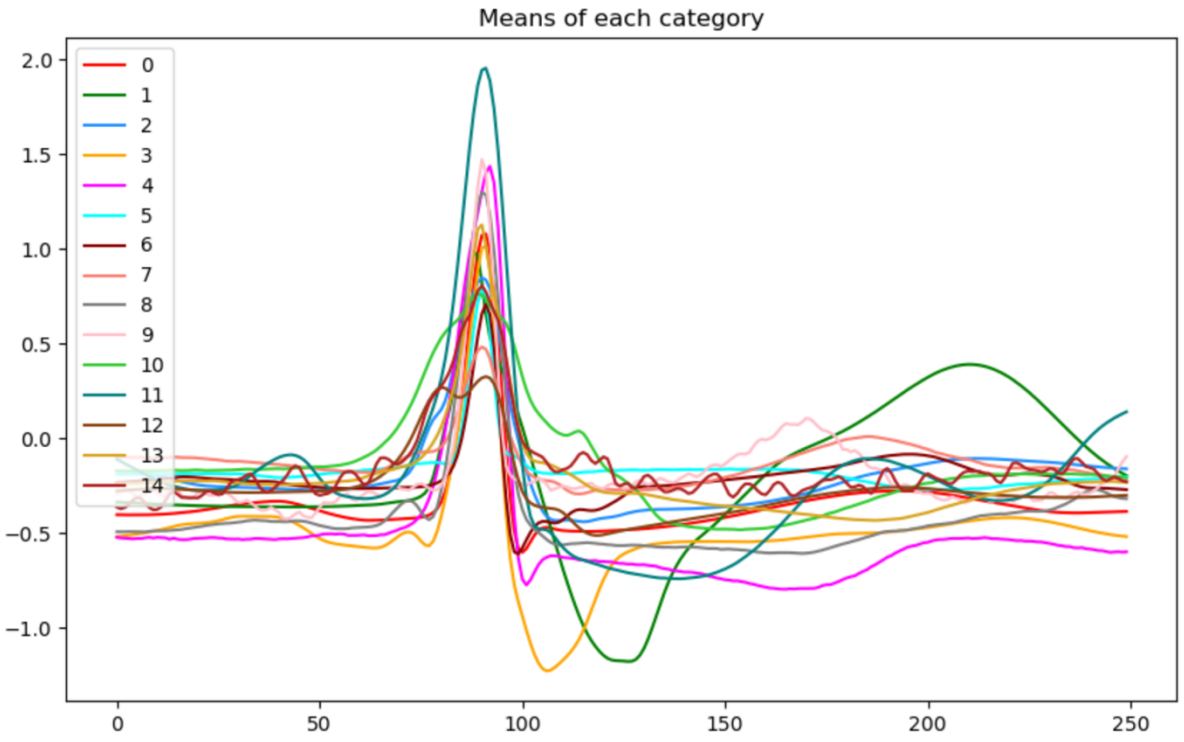
ECGbyCat contains 15 lists, each containing all the ECG samples of a given category.

The first list contains all type 0 ECG samples, the second list contains all type 1 ECG samples, and so on.

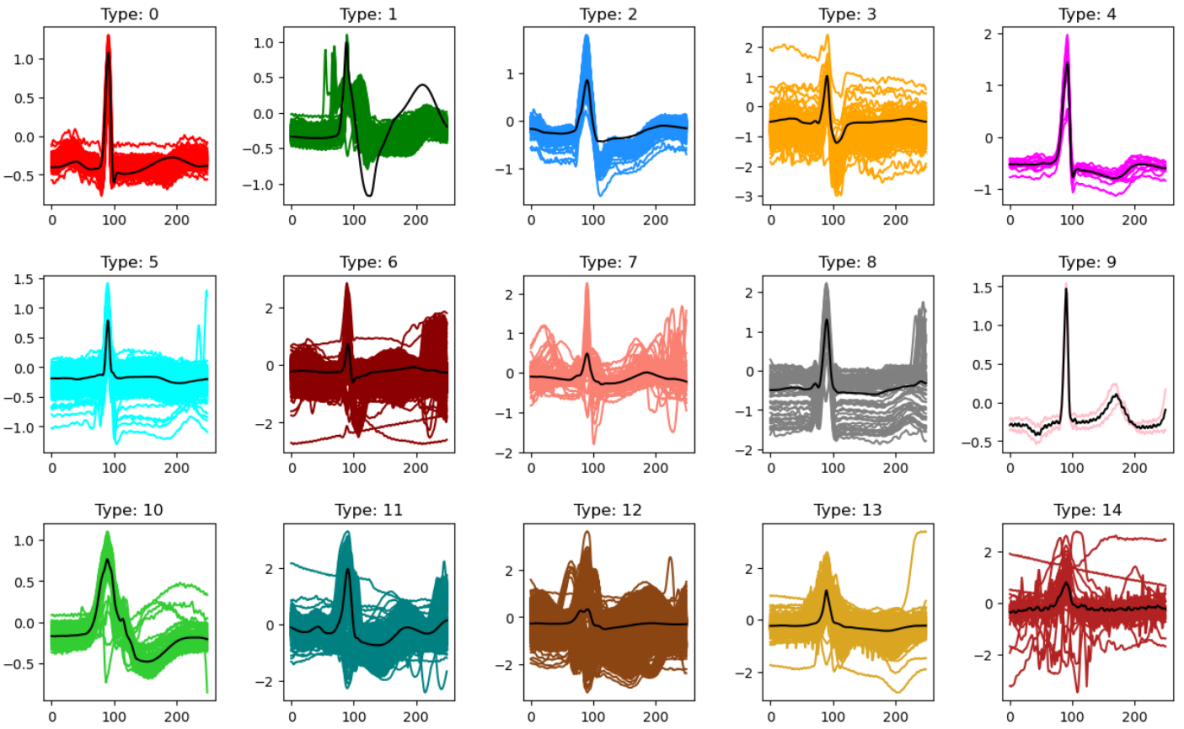
From this list, which, we grant you, has a rather complex form, we were able to derive the following lists.

The list ***sums*** contains 15 new ECG samples, obtained by summing all the ECG samples of a same category.
Then, the list ***means*** is obtained by dividing each ECG sample in ***sums*** by the number of original samples that have been summed (that number is stored in the list ***counts***, established for the counting operation). Roughly speaking, it weights the list ***sums*** to create a list of 15 new samples, each of which is the average of all samples of a specific arrhythmia type.

To ensure that the averages are consistent, and to check whether there are significant differences between different categories, we plot these 15 average ECG samples on a same graph :



Finally, for each of the 15 arrhythmia types, we have plotted on a small graph the first *N_max* samples of that category, and the average sample of that type (found in the list ***means***). This allows us to visualize the overall trends of samples according to their arrhythmia type, and to check that the distributions made previously are consistent.



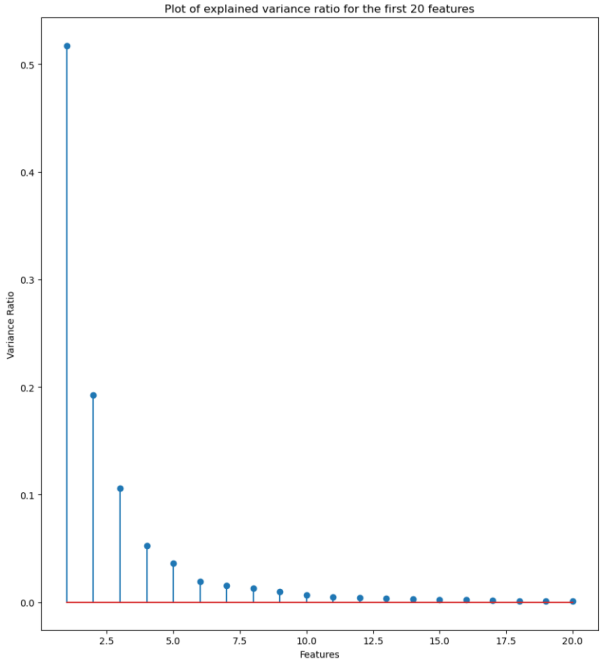
3.2 Pre-processing

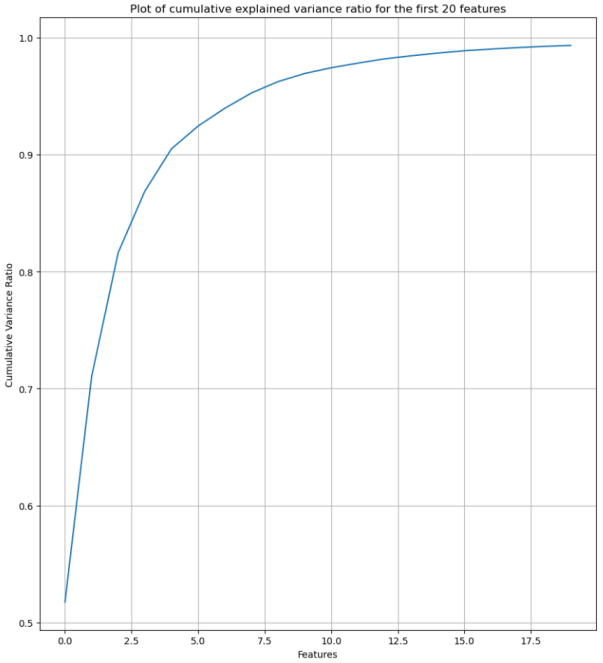
Our approach has been to use 2 different sets of data:

- The first dataset we use is the already provided data. As the QRS peak detection, Beat segmentation and train/val/test Splitting data operations are already performed, no additional pre-processing is required and the data is ready for model training.

- The second dataset we will be using is obtained by applying a PCA to the already provided data. We have applied a PCA to the train, test and val data sets supplied by the project data : the new sets generated are named *train_pca*, *test_pca* and *val_pca*. We have therefore retained the desired distribution of 60, 20, 20%.

In order to interpret the results of the PCA, we followed the procedure studied in Lab and plotted the evolution of the variance ratio on a graph. We concluded that the first 6 features could be retained for the rest of the project, and that the others were irrelevant.





Now that this is done, let’s move to classification.

4. Task II : Classification

We have decided to use a **multilayer perceptron** model for our computations. This choice was motivated by its very good application to multi-class classification problems. As stated previously, this approach will be applied twice : first, with the already given data sets ; then, on PCA-processed data.

First, we will present the different tunable hyperparameters we decided to optimize. Then, we will explain how we determined the criteria for best modeling. Finally, we will show and interpret the obtained results

4.1 Tuning parameters

Since we used a MLP model, there are several hyperparameters we can change in order to find the best model. The hyperparameters we have kept for optimization are the followings :

4.1.1 Alpha

The alpha parameter corresponds to the regularization strength in our neural network. A smaller alpha means a smaller regularization strength and a closer fit to the data train set. This also means that a too small alpha may lead to overfitting.
We are typically looking for the best alpha in the list :

$$\alpha = [0.1, 0.01, 0.001, 0.0001]$$

4.1.2 Activation function

The activation function is basically what makes a neuron activate or not, depending on the input. This is thus a very important part of the neural network. Among others, we can recognise the 3 best activation functions as followed :

$$f_{activation} = [\text{relu}, \text{logistic}, \tanh]$$

4.1.4 Solver

The solver is the numerical method used to optimize the set of weights in our model, in example the gradient descent mentioned during the lectures. Speaking of which, we used the stochastic gradient descent and the adam solver during our model tuning :

$$solver = [\text{Adam}, \text{Stochastic gradient descent}]$$

4.1.4 Width and number of hidden layers

Finally, the most important parameter we tune in our process is the size of our layers in the neural network. The width and depth of the neural network determine how efficient and how expensive (time, iterations) it will be.

$$size = [(50), (25, 25), (50, 50), (100, 100), (100, 75, 50)]$$

We have decided to fix each of those parameters in a list, and try them independently from each other; i.e. we tune one parameter at a time. Concerning the choice of the other parameters while tuning one, we simply take the best value for the parameter found before. If it has not been tuned before, we simply take the first value of its list. We assume that this method gives the clearest results for each parameter, since those parameters can be taken as non-correlated (for example, choosing one or another solver will not affect the best activation function).
This corresponds to a **Sequential hyperparameter optimization** which could have more bias than a random search and could be less effective than a grid search. However, taking into account the low complexity of our model and data, a grid search would be very expensive for a very small improvement and a random search would be almost as efficient. Furthermore, based on the already very good performance obtained in the end, we will make the choice to keep this method.

4.2 Comparing scores

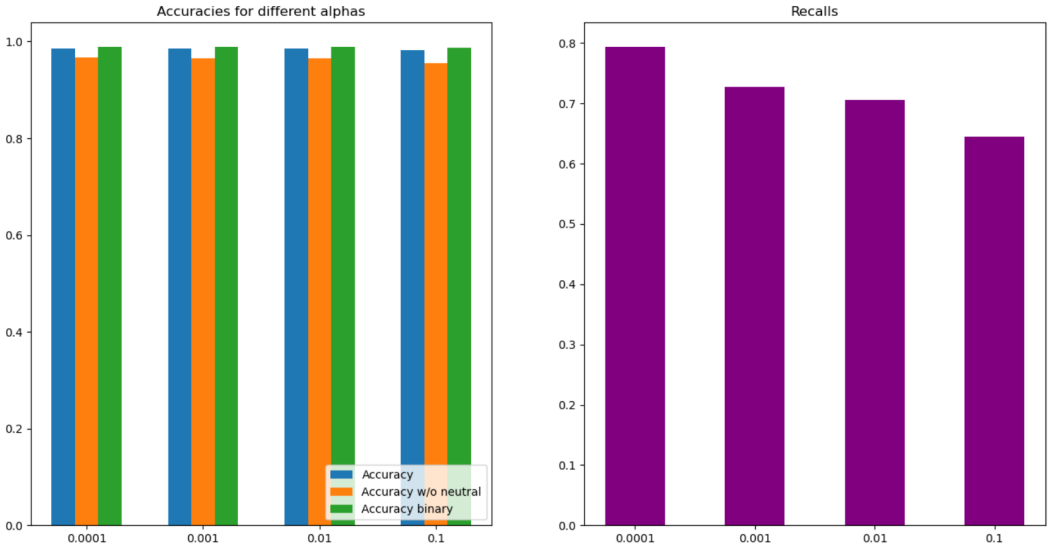
In order to compare our different models, we use four different scores : **overall accuracy**, accuracy in between non-neutral classes (named **'NN-Accurary'** from now on), accuracy in between the neutral class and every other (**binary accuracy**) and the **recall** score.
While the overall accuracy tends to give a very high score since most of the data (70%) is neutral, the NN-accuracy gives a good representation of how good different arrhythmias are differentiated. The binary accuracy is here to see how good we can tell if someone is sick or not, no matter what kind of arrhythmia the patient has. Finally, the recall score gives a good indication of the amount of False Negatives, which are in our case something we want to minimize compared to the False Positives (the consequences are way less dramatic in this way).
Here are the **formulas** for each of the score we are using:

- $A_{overall} = \frac{number\ of\ correctly\ classified\ instances}{number\ of\ classified\ instances}$
- $A_{NN} = \frac{number\ of\ correctly\ arrhythmia-classified\ instances}{number\ of\ \ arrhythmia-classified\ instances}$
- $A_{binary} = \frac{number\ of\ correctly\ neutral-classified\ instances}{number\ of\ classified\ instances}$
- $Recall = \frac{1}{C}\sum_{i=1}^{C} Recall_i$ where C : total number of instances ; $Recall_i$: recall for class i

4.2.1 Original data

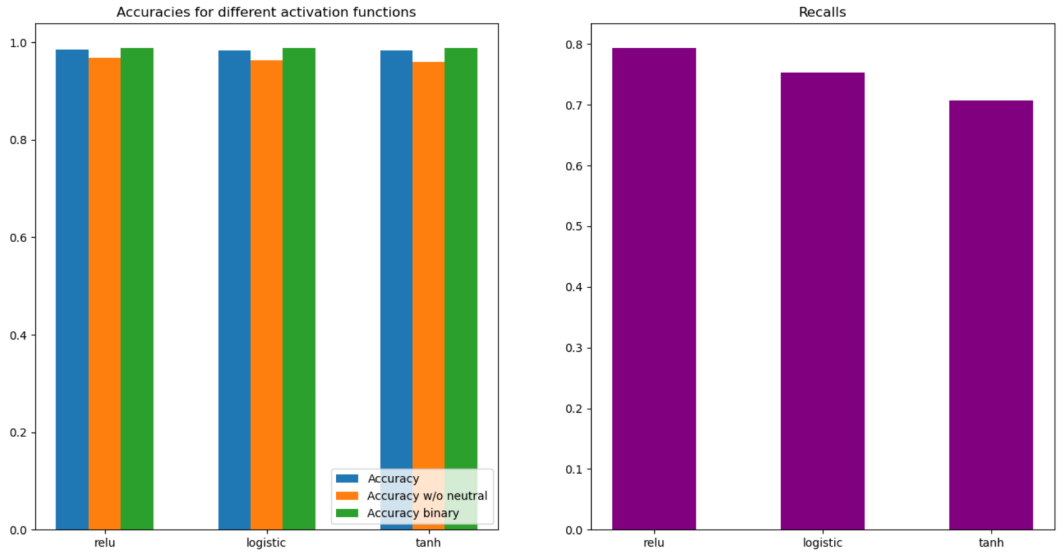
Our results will be presented as bar plots, since they seem to be the most visual way of showing them. As mentioned in the legends, the blue bar is the overall accuracy, the orange one represents the NN-accuracy and the green one is for the binary accuracy. In addition, we plot the recalls of each option in order to distinguish them in case they have very close scores.

a) Best Alpha :



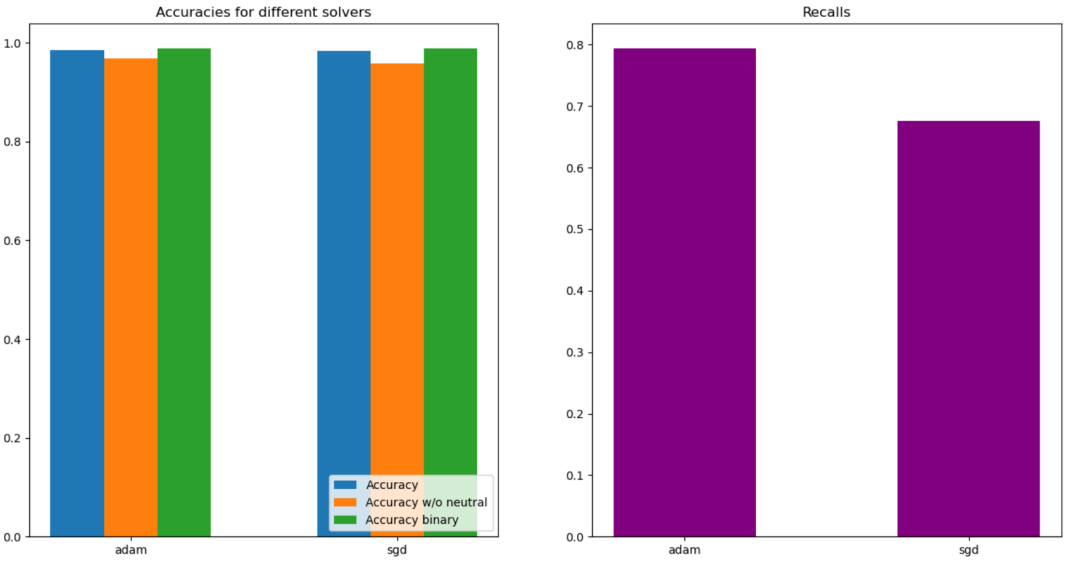
We can see in the plot that the overall accuracies of every α are very close to 0.98, but the best recall is given for **α = 0.0001**, hence this is the value we will keep in the next computations. Although, we need to be careful with using such a small value for α since it could lead to overfitting during the training process.

b) Best Activation Function :



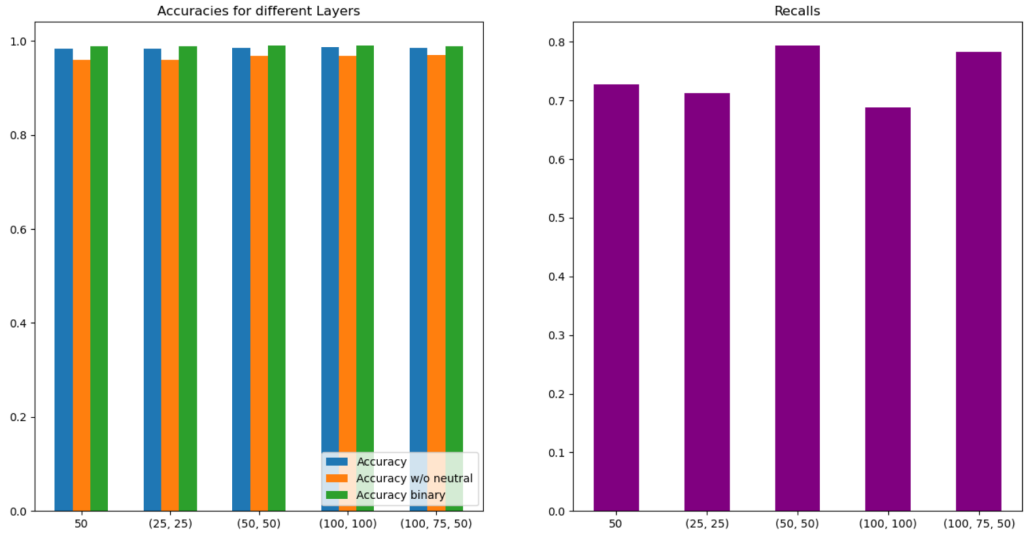
It can be hard to tell which one is better than another by just looking at the accuracies (everyone of them is around 0.985), however the relu function seems to have a better recall score (0.79 > 0.75 > 0.71). Even if the difference is almost non-noticeable, we can confidently say that the **relu** activation function is the best for our model.

c) Best Solver :



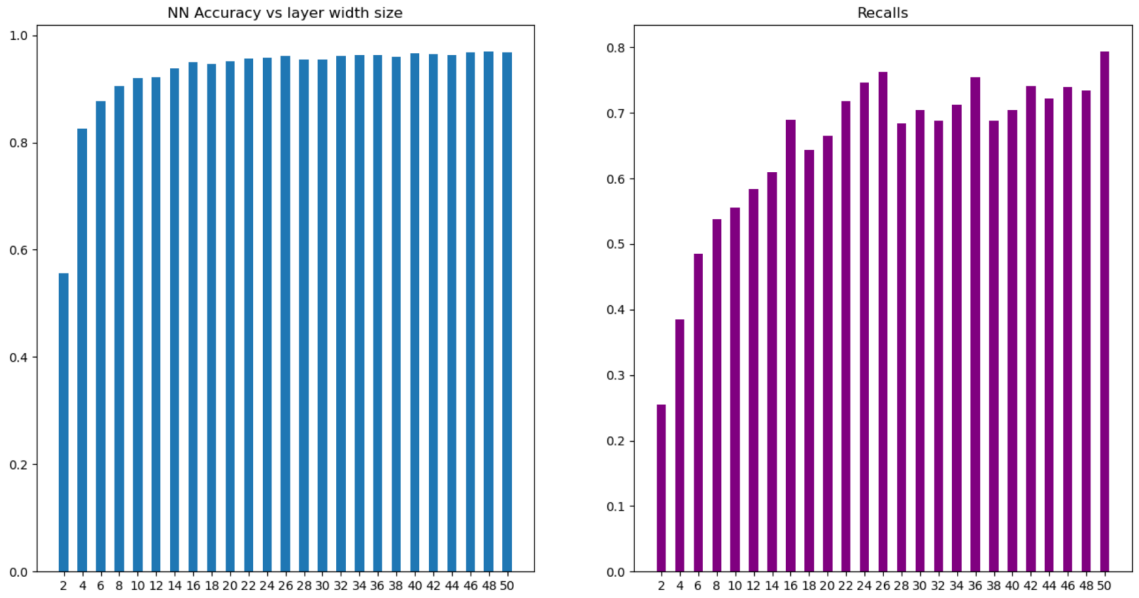
Regarding the best solver, the only noticeable difference is once again given with the recall score. **Adam** solver looks better than the Stochastic Gradient Descent (0.79 > 0.68)

d) Best layer size :



Here, we finally can look at the most important hyperparameter in our model. Surprisingly enough, even the one layer neural network gives good results, and increasing to a 3 layers neural network or increasing the width of each layer does not increase the accuracies a lot. Concerning the recall score, we can see a peak for a 2 layers neural network with size 50 for each, which seems to be the best option for us.

To look further in the optimization of our model, we plot the accuracies and recall for hidden layers of size (***i, i***) where ***i*** goes from 2 to 50 by steps of 2. This gives us a good idea of when the accuracy stops getting relevantly better, in an attempt to choose the best trade-off in between performance and cost.



From this plot, we can see that an interesting trade off point would be for layers (26, 26), since after this the accuracies only get slightly better and the recall has a peak around 0.77. Hence, we will take the best size for our layers as **(26, 26)**.

To put it in a nutshell, by optimizing the hyperparameters we found :

α = 0.0001 ***activation function*** **: relu**

solver **: adam** ***layer size*** **: (26, 26)**

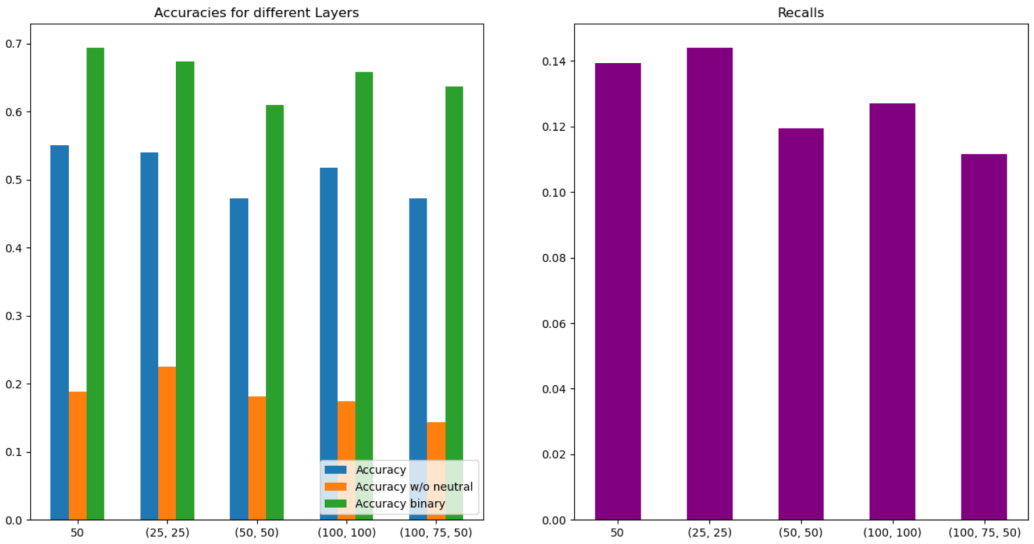
These gave us a model which has very promising performances (given from testing our model with the test set) :

$A_{overall} = 0.98172$ $A_{NN} = 0.95933$ $A_{binary} = 0.98720$ $Recall = 0.72787$

4.2.2 PCA-processed data

The main goal here is to compress the needed features and see if a much smaller PCA dataset would give at least decent results compared to a full size deep neural network. If so, then it would be an interesting path to follow as it would really reduce the cost of the training and therefore make it more accessible. Thereby, we apply a PCA transformation on the train, test and validation sets independently to make sure each validation and testing operation are unbiased. From there, it is just a matter of training different models for each option in our parameter lists, and comparing them.

After tuning the model and using the best hyperparameters when tuning the layers, PCA data shows disappointing accuracies :



The overall accuracies can be listed in the interval : **[0.61, 0.69]**
The NN-accuracies can be listed in the interval : **[0.47, 0.55]**
The binary accuracies can be listed in the interval : **[0.15, 0.23]**
The best recall is around **0.145**.

These are indeed very bad, and are completely **unacceptable for medical purposes**.

Conclusion

In conclusion, we are satisfied with our work and proud to have completed this project.

In terms of results, the best model we've come up with is a multilayer perceptron model with hyperparameters $\alpha = 0.0001$, $f_{activation} = Relu$, $solver = Adam$, and $Layers = (26, 26)$. This model is destined to be trained with the supplied, already pre-processed data.

With an overall accuracy of 0.981 and a recall of 0.727, this model is consistent with the medical application aimed by this project: we designed our model search so that we prioritize the avoidance of false negatives over false positives : from a human point of view, it seems acceptable to classify a neutral ECG sample as arrhythmic, whereas it's essential not to classify an arrhythmic ECG sample as neutral.

This project proved to be an excellent pedagogical approach. It gives concrete form to the theory studied in class, which can sometimes seem rather abstract to students. The practical and autonomous aspect of the project enabled us to assimilate the course properly.