Pattern Recognition and Machine Learning Assignment

Group 20

Willy Ayissi

Hugo Linsenmaier - 270 742

Kristian Järvenpää

Daniel Linjama

March 10, 2017

Contents

Introduction

We started by creating Kaggle accounts and formed a team called Team 20. Then, We downloaded the competition data and the template according to the assignment instructions.

After downloading the data and the template, the next step was to try to get to know more about our data (data structure) and try to understand the preprocessing done in the template.

* Data structure : in order to understand our data structure we analyzed the x_train.csv and x_test.csv files . We found that our data consists of 15485 genes for x_train and 3871 genes for x_test that are repeated 100 times and for each genes we have 5 histones markers. The rows of both files represent the genes and the columns the histones markers.

* Preprocessing of the classifier : the preprocessing consists on removing the first row that represent the name of each columns and reshaping the data (x_train and x_test) so that on each row represent only one gene for both files.

In this document we are going to present the differents steps of our progress in the competition since the beginning with the logistic regression until the end with the neural networks.

I. Logistic regression with default parameters

We introduced ourselves to Kaggle and the competition format by implementing a Logistic Regression classifier on top of the template. We used the default parameters and the code lines added are presented below.

```
LR_default = LogisticRegression()
LR_default.fit(x_train,y_train)
y_pred = LR_default.predict_proba(x_test)
#preparing the submission
y_hat = y_pred[:,1]
submit(y_hat,"LR_defaut")
```

In this part we just fit and predict with our logistic regression classifier. To generate the file for the submission we use a function called submit (see in annexes). In that first test that we made we obtain a score of 0.88834 on Kaggle (with the public score).

II. Experiments with parameter C and penalty

To further improve our prediction with logistic regression, we tested different C values and penalty. The first step was to do a loop where we were estimating AUC score for different values of C for L1 and L2.

```
#First let's split our training datas into two parts each
X_train2, X_test2, y_train2, y_test2 = train_test_split(x_train, y_train, test_size=0.

#first let test the impact of C parameter on the accuracy
result  = []; score = [[],[],[]]
ind = 0;ind2=0
C_range = np.array([1e-10,1e-9,1e-8,1e-7,1e-6,1e-5,1e-4,1e-3,1e-2,1e-1,1e-0,1e1,1e2,1e
rang = np.arange(-10,10)
clf  = LogisticRegression()
score.append(list(C_range))
for penalty in ["l1", "l2"]:
   clf.penalty = penalty
   for C in C_range:
        clf.C = C
        clf.fit(X_train2, y_train2)
        y_pred = clf.predict_proba(X_test2)
        y_f = y_pred[:,1]
        accuracy = 100*roc_auc_score(y_test2, y_f)
        score[ind+1].append(accuracy)
   ind += 1
```

Then we tried to find out the best parameters between all the combination of C and penalty. After that we fit and predicted with the parameters C and penalty with highest accuracy in order to improve our score and submit the result on kaggle. And as expected the score on kaggle increased up to 0.89603.

```
# we are going to select now the best params
selected_params = []
```

```
ind = 1
if (max(score[1])<max(score[2])):
    ind = 2
ind_best = returnIndexOfMax(score[ind])
selected_params.append(["l%s"%(ind),C_range[ind_best]])
#prediction with the best parameter
clf.penalty = selected_params[0][0]
clf.C = selected_params[0][1]
clf.fit(x_train,y_train)
y_sub1 = clf.predict_proba(x_test)
y_sub = y_sub1[:,1]
submit(y_sub,"Logistic_best_params")
```

The following graphs shows the variation of the accuracy depending on the C parameter.

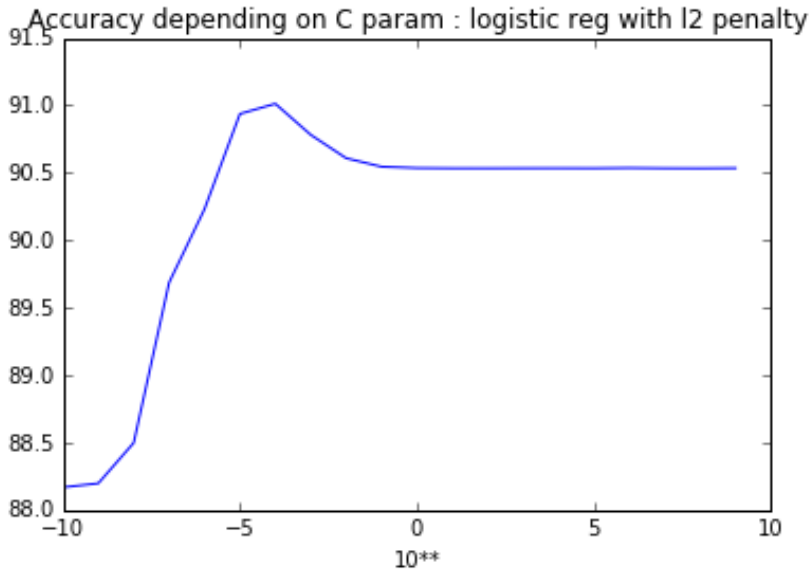

Figure 1: Variation of accuracy depending on C paramter with the l1 penalty.

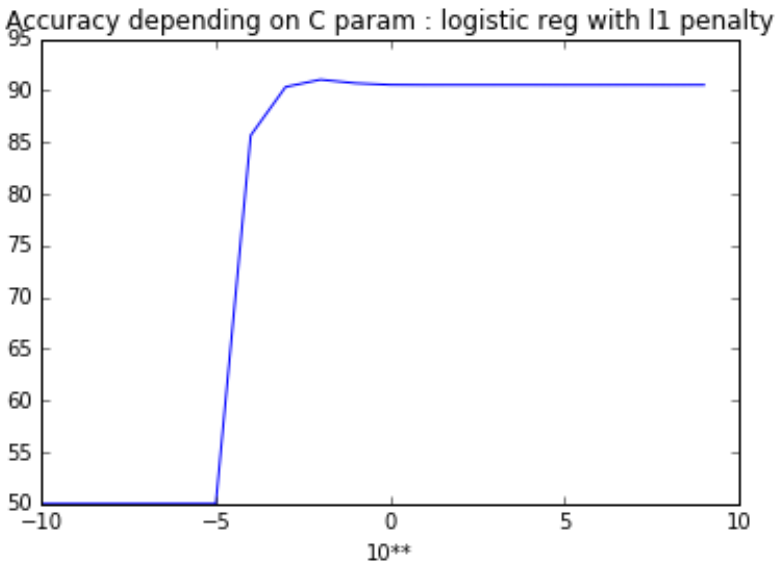

Figure 2: Variation of accuracy depending on C paramter with the l2 penalty.

From the two graphs above we can see that for the both of them the maximum is obtained between C = 10**-6 to C=1. That is why in the following table which give the accuracy depending on the C value and the penalty chosen (rounded after 3 decimals), we just focused on the value of C between 10**-6 to 1. In this table we can see that we have the best accuracy with the 'l1' penalty and the 10**-2 value for the C parameter.

Values of C	**L1 penalty**	**L2 penalty**
10e-6	0.5	0.900
10e-5	0.5	0.906
10e-4	0.860	0.907
10e-3	0.901	0.904
10e-2	0.910	0.902
10e-1	0.905	0.902
10e0	0.902	0.902

III. Preprocessing and Cross-validation

With the same objectives of increasing our score, the next step was to try to reduces the dimensionality of our data by using a pipeline of PCA (Principal Component Analysis) and Logistc Regression. Here is the following code:

```
pca = decomposition.PCA()
pipe = Pipeline(steps=[('pca', pca), ('logistic', clf)])
n_components = [10,20,30,40,50,60,70,80,90,100,120,150,200,250,300,350,400]
estimator = GridSearchCV(pipe, dict(pca__n_components=n_components), cv=5, scoring='roc_au
estimator.fit(x_train, y_train)
print(estimator.best_params_)
print(estimator.cv_results_['mean_test_score'])
print(estimator.cv_results_['std_test_score'])
```

After implementing the pipeline of PCA and logistic regression, we fit and predict with the best numbers components and we obtain a score of 0.89893 on Kaggle.

```
y_pred = estimator.predict_proba(x_test)
y_f = y_pred[:,1]
submit (y_f,"pca_pipelining")
```

The following tables present the score for each number of components estimated :

Dimension of PCA	**Score**	**Standard deviation**
10	0.887	0.013
20	0.908	0.0083
30	0.910	0.0082
40	0.911	0.0085
50	0.912	0.0078
60	0.911	0.0078
70	0.911	0.0078
80	0.911	0.0077
90	0.911	0.0078
100	0.911	0.0077
120	0.911	0.0077

IV. Improvement of the accuracy with different classifiers

Still with the same objectives of gaining more precision with our predictions, we thought about trying to use other classifier that could be more accurate that the logistic regression classifier. That's why we almost try all the classifier that we knew at the time. So, in this part we are going to present the other classifiers that we used (random forest, extra-trees, adaboost, xgboost, gradientboost, k-nearest-neighbors). But in our presentation we are going to focus in the presentation of the xgboost classifier which gave us our higher score (0.90759) on kaggle in a first time, then we are going to present the other classifiers that we used. (The full code can be seen in the annexes)

XGBoost Classifier

Another classifier that we implemented on our project was the XGboost classifier. But first we try to maximize our score with that classifier by choosing the best parameter that can influence the score (so we just focused on the max_depth parameter (which represent the maximum of tree depth for base learners) and the n_estimators (number of estimators of the classifier). First we try to see the variation of the accuracy by fixing different number of estimators with different value of max_depth. We obtain the following figure

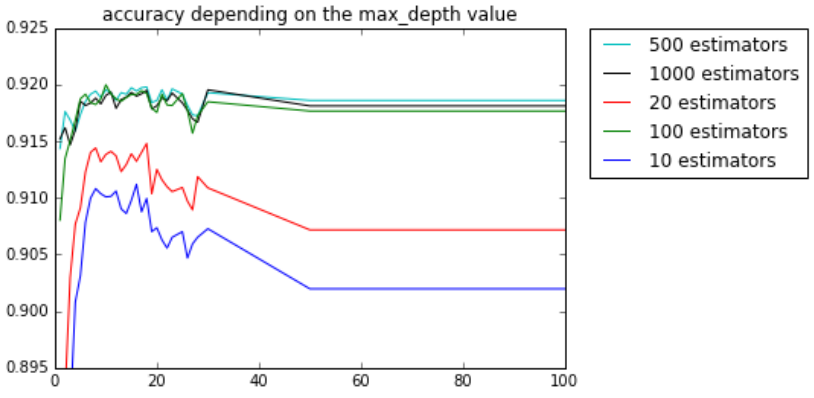

Figure 3: Variation of accuracy depending on max_depth for the XGBoost.

In this figure we can observe two things : -¿ The maximum of our accuracies is near 20 for all the classifiers and after a certain value (about 30) the accuracy decrease and stabilize itself after 50. We will choose as best parameter of max_depth the value of max_depth that maximize the accuracy for all the n_estimators values (here it was 18 for the n_estimators = 500). -¿ We can also observe in this figure that the accuracy of our classifier increase with the number of estimators until 500. After this value we can see that (the sky blue curve of the accuracy with 500 estimators is slightly above the black curve of 1000 estimators). We can conclude that the maximum accuracy is obtain with a number of estimators ¡ 1000 and above 1000 estimators we overlearn our data. In the following figure we can look that the accuracy starts to decrease after n_estimators = 1000.

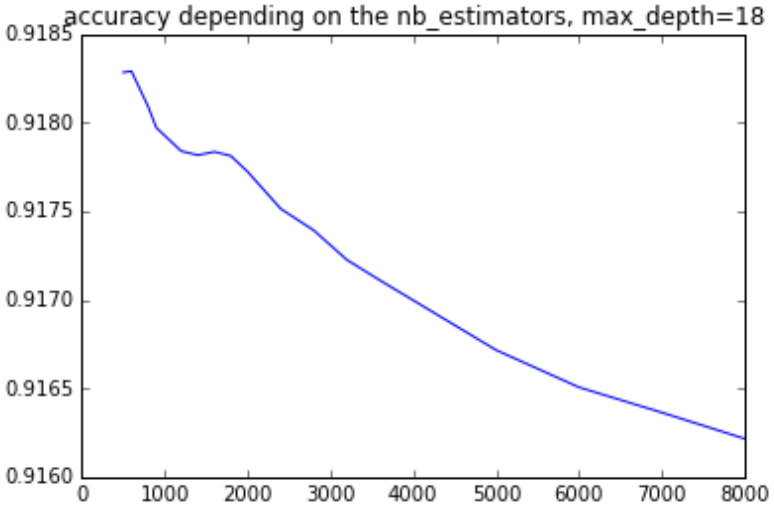

Figure 4: Variation of the accuracy with the number of estimators. for the XGBoost

The next step was then to find the value of estimators between 100 and 1000 estimators that maximize our accuracy with max_depth and submit to Kaggle. We found that the number of estimators that was maximizing this accuracy was 650 and we obtain a score of 0.90759. (see the code in annexes)

Other tested classifiers

In order to increase our chances to get a better score we try to implement the few many others classifiers in our project (ExtraTrees, Adaboost,grandientBoosting, KNeighbors). We start by forming groups of classifiers that have the same parameters.

```
#First step : training the classifier seen in class
    classifiers1 = [(RandomForestClassifier(), "Random Forest"),(ExtraTreesClassifier(), "
    classifiers3 = [(KNeighborsClassifier(),"KNeighbors")]
```

After that we made a loop for each group of classifiers like in part 3 were we try many values of our parameters in order to find the parameters that maximize our accuracy. At the end we choose the classifier that gives the best accuracy and submit the result on Kaggle

The following picture show the obtained results of the comparison of the classifier with their best parameters

Classifier	**roc_auc_score**
Random Forest	91.4188%
Extra-trees	91.2913%
AdaBoost	90.5734%
GB-Trees	91.6332%
KNeighbors	91.0068%

V. Customer Meetings report

First customer meeting took place 9.2.2017. The client introduced the biological background of the problem and how the measurements were done. For us, the most important take-away was first all to confirm us that our understanding of the provided data. In this meeting, our customer also define us what he was expecting from us. Indeed our tasks was to find the histone markers which has strongest relation to gene activation and to repression? In oder words the task here was to find out the histone markers which has a strong effect on the gene expression.

We try to answer to that question by making features selection as far as we knew that our columns represent histone markers in our data. And for all the selected features we try to find out the proportion of selected features belonging to each histone markers. And with that information we were able to classify the histones markers that has more impact in our genes.

After this first meeting, we discussed about the method to implement in order to make our features selection. We decided to use the RFECV which was the method that we was most used to. But we decided to use also the random forest features selector in order to compares our two results before generalizing.

The second meeting took place 16.2.2017. We introduced our findings so far and presented our future plans. The client seemed very satisfied with our progress.

And the last meeting was to present the result our results. From customers meetings we know that the most important histogram modifications should be H3K4me3,H3K9me3. In our data, each gene is represented as an array of 500 features, where every 5th element is from the same histogram modification. This means that, for example, the values corresponding to H3K4me3 our found at indicies 0,5,10,15,...,495,500. Now observe in figure 2 that the most important features our found at indices 253, 258, 263 and 248. These are all five apart, meaning they belong to same histogram modification which is H3K9me3. This supports the theory introduced by our client.

Features selection with RFECV

The first method that we implemented in our features selection was the RFECV method. The purpose here was to find out the proportion of each histone markers in the selected features.

For the implementation of our rfecv we choose to use the xgboost classifier since it was the classifier which was giving us the best accuracy. The following code shows the implementation.

```
rfecv = rfe(estimator=XGBClassifier(n_estimators=650,max_depth=18),step=50,verbose = 1,c
  rfecv.fit(x_train, y_train)
  rfecv_scores = rfecv.grid_scores_
```

After the RFECV implementation, the next step was now to find out the number of features selected from each histone marker for all our genes. This was done by using the mask of selected features which is stored in the support_ attribute of the rfecv. This mask of selected features consists on a vector in our case of 500 rows, which rows take the value "True" when the features (rows of our data) corresponding to this row indicies has been selected and "False" if not.

So knowing that in our data, each gene is represented as an array of 500 features, where every 5th element is from the same histogram modification. For example the features found at indicies 0,5,10,15,...,495 correspond to H3K4me3 marker and for those in indicies 1,6,11,16,...,496 correspond to H3K4me1 marker. With that basis we count the selected features for all the histone markers. An extract of the code for the histone maker H3K4me3 is shown in the following lines

```
support = rfecv.support_


nb_selected_features = 0
histone_marker = []
#histone marker H3K4me3
for i in range(0,500, 5):
    if (support[i]==True):
        nb_selected_features +=1
```

```
histone_marker.append(['H3K4me3',nb_selected_features])
#histone marker H3K4me1
```

The same thing has been done for the 4 other histone marker. The following table give the histones marker with the number and proportion of selected features belonging to each of them

Histone markers	**nb selected features**	**proportion**
H3K4me3	61	24%
H3K4me1	47	18%
H3K36me3	50	20%
H3K9me3	50	20%
H3K27me3	42	16%
Total	250	100%

Features selection with Random Forest features selector

The second method that we used to get the most important features was the built-in random forest feature selection. Random forest doesn't works the same way as RFECV. Actually random forest just rank all the features by giving to each of them a score going from 0 to 1. The highest is the score the more important the features is. And when we sum up all the scores of all the features we have obtain 1. The figure below show the 25th most important features that we figure out with this method for example

```
1. Histone 3 (0.024037)
2. Histone 3 (0.020952)
3. Histone 3 (0.020290)
4. Histone 3 (0.017409)
5. Histone 3 (0.017004)
6. Histone 3 (0.014524)
7. Histone 3 (0.014269)
8. Histone 3 (0.013546)
9. Histone 0 (0.011601)
10. Histone 0 (0.011080)
11. Histone 1 (0.009629)
12. Histone 0 (0.009378)
13. Histone 3 (0.009099)
14. Histone 1 (0.008742)
15. Histone 1 (0.008697)
16. Histone 0 (0.008367)
17. Histone 3 (0.008211)
18. Histone 3 (0.008008)
19. Histone 3 (0.007964)
20. Histone 2 (0.007902)
21. Histone 1 (0.007614)
22. Histone 0 (0.007438)
23. Histone 0 (0.007376)
24. Histone 3 (0.007193)
25. Histone 3 (0.007079)
```

Figure 5: 25 most important features achieved with random forest of 100 trees.

The following lines show the code of this method :

```
clf_rf = RandomForestClassifier(n_estimators = 100)
clf_rf.fit(x_train, y_train)
importances = clf_rf.feature_importances_
std = np.std([tree.feature_importances_ for tree in clf_rf.estimators_],axis=0)
indices = np.argsort(importances)[::-1]

print("Feature ranking:")
proba = [0] * 5
for f in range(x_train.shape[1]):
    #each line in proba correspond to one histone marker
    print("%d. Histone %d (%f)" % (f + 1, indices[f] % 5, importances[indices[f]]))
    proba[indices[f] % 5]+=importances[indices[f]]
print(proba)
```

```
print(sum(proba))
```

In figure 6 below, the black histograms represent the feature importance and the blue error-bars represent the inter-trees variability. Observe that the inter-tree variability is very large. We believe this is due to very small number of estimators, only 1000. Similarly, in figure 7 we present the feature importance of 50 most important features and their variability.

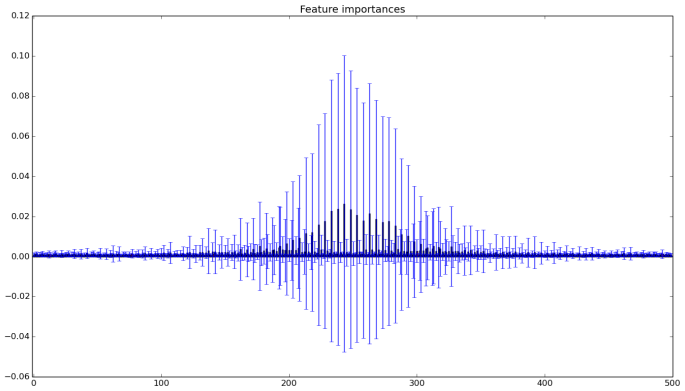

Figure 6: Feature importances achieved with random forest of 1000 trees.

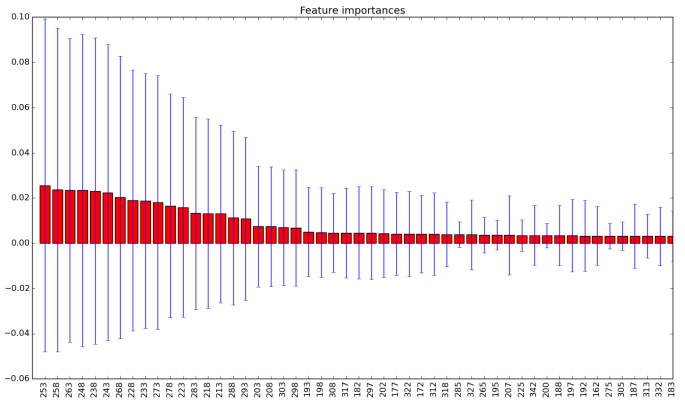

Figure 7: 50 most important features achieved with random forest of 1000 trees.

Though this method was giving us a good overview a ranking of our features according to their importance. That wasn't the what we wanted, indeed as we

said in the beginning the objective was to rank the histone markers. So we decide to sum up the score of the features of each histone marker and compare this histones markers using that cumulative score. The table below show the score that we obtain for each Histone marker.

Histone markers	**score**
H3K4me3	0.2472
H3K4me1	0.1759
H3K36me3	0.2033
H3K9me3	0.2965
H3K27me3	0.0771
Total	1

Conclusion the customer tasks

From the two method (the two tables) we can draw the conclusion that the histones markers that has a has strongest relation to gene activation was the genes H3K4me3 and the H3K9me3 because in the two case their are the two ones with highest score. And the histone marker with the least relation to gene activation was the H3K27me3 still according to our results with our two methods.

In our final meeting on the 23rd of February, we presented to our client our results and from a biological point of view he was quite satisfied and it made sense to him as far as the markers H3K4me3 and H3K9me3 respectively has the role of promoter mark and repressor mark, which are the function with the most important impact.

VI. Neural network

In this final part we tried first of all to experiment using simple neural networks. But the challenge here was :

* find the number of layers that increase our accuracy

* find the number of epoch and batch_size that will give us the best accuracy

Number of layer

So, for the first problem we didn't have enough choice rather than implementing same neural networks with different number of classifier and see from which number the accuracy starts decreasing. In our case, it started to decrease from layers. So we decided to use layers

Number of nodes, epoch and batch_size

Here we decided to proceed the same as other classifier that we used such as XGBoost in order to find our hyper-parameters. So we made an imbrication of 3 loop :

* in the 1st : we choose different value of the node * in the 2nd : for each number node, we try different values of the epoch * in the 3rd : for each number of epoch, we try different values of batch_size. in this loop we also fit, predict and generate file (which name contains the number of nodes,the epoch value, the batch_size value.

The following code illustrate the method used

```
# Add layers one at the time.
nodes_range = [50,100,150,200,250]
epoch_range = [10,15,20,25,30,35,40]
batch_range = [1,2,3,4,5,6,7,8,9,10]
result_per_nodes = []
for nb_nodes in nodes_range :
    clf_keras = Sequential()
    clf_keras.add(Dense(nb_nodes, input_dim=X_train2.shape[1], activation = "sigmoid")
    clf_keras.add(Dense(nb_nodes, activation = "sigmoid"))
    clf_keras.add(Dense(nb_nodes, activation = "sigmoid"))
```

```
        clf_keras.add(Dense(nb_nodes, activation = "sigmoid"))
        clf_keras.add(Dense(1, activation = "sigmoid"))
        clf_keras.compile(loss="mean_squared_error", optimizer="sgd")
        result = []
        for epoch_value in epoch_range:
            roc_score = []
            for batch_value in batch_range:
                clf_keras.fit(X_train2, y_train2, nb_epoch=epoch_value, batch_size=batch_v
                keras_pred = clf_keras.predict_proba(X_test2)
                submit(keras_pred[:,0],"NN_params_%f_%f_%f"%(nb_nodes,epoch_value,batch_va
                roc_value = 100*roc_auc_score(y_test2,keras_pred)
                roc_score.append(roc_value)
            result.append(roc_score)
        result_per_nodes.append(result)
```

After that , it was question for us to find the parameters with the best accuracy in order to submit the file corresponding to that parameters. The following lines shows the code used.

```
 #get the node with the best score
max_score2= []
for i in range (len(nodes_range)):
    max_score1= []
    for j in range(len(epoch_range)):
        ind_best_batch = returnIndexOfMax(result_per_nodes[i][j][:])
        max_score1.append(result_per_nodes[i][j][ind_best_batch])
    ind_best_epoch = returnIndexOfMax(max_score1)
    max_score2.append(max_score1[ind_best_epoch])
ind_best_node = returnIndexOfMax(max_score2)

#get the number of epoch of that node which give this best score
max_score1 = []
for k in range(len(epoch_range)):
   ind_best_batch =  returnIndexOfMax(result_per_nodes[ind_best_node][k][:])
   max_score1.append(result_per_nodes[ind_best_node][k][ind_best_batch])
```

```
ind_best_epoch = returnIndexOfMax(max_score1)

#get the batch_size corresponding
ind_best_batch = returnIndexOfMax(result_per_nodes[ind_best_node][ind_best_epoch][:])

print ("the best parameter are : \n node : %d \n epoch : %d \n batch_size : %d"%(nodes
print("\n with the score %f"%(result_per_nodes[ind_best_node][ind_best_epoch][ind_best
```

Despite our efforts, neural networks did not seem to be the best approach to this problem, the best score that we get with this approach was 0.90295. We would have liked to experiment with deeper networks with constitutional neural networks but unfortunately we ran out of time.

Conclusion

At the due date of the competition, we finished with the 17th rank, we would had like to have a better rank and win the competition but we didn't. But the most important for us is the knowledge that we gained on this project and in general from this course. We particularly also like the meeting with the customer because it was kind of a simulation of a real life working experience with it constraints. This meeting taught us how to work with people with different studies background than ours.

Annexes

The full code can be found with this link : https://github.com/wdechanelo/ML-Project/blob/master/FinalCode.py