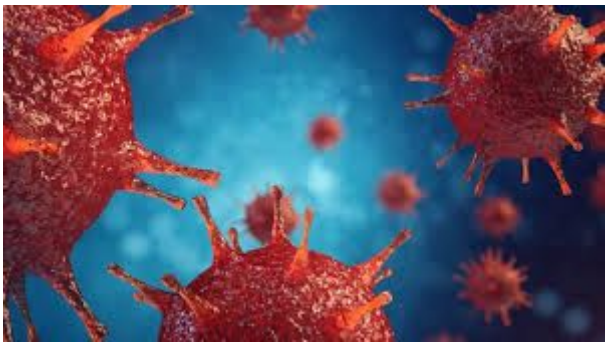


HADACA : Fight the Cancer

Team name : FACTOR
Group name : HADACA (Cancer research)
Team members : Omar Abdoulaye BADIANE (1bis), Aline BOUBEE DE GRAMONT (1), Iuliia LOGVINOVA, Noé ROBIN (1bis)
Challenge URL : https://codalab.lri.fr/competitions/333
N° of code submission : 8406
Github repo of the project : https://github.com/FACTOR-CR/FACTOR
Video URL : https://www.youtube.com/watch?v=MFn8fUTswKM&feature=youtu.be
Diapos URL :
https://github.com/FACTOR-CR/FACTOR/blob/master/FACTOR%20project.pdf

Introduction of the challenge and datas :

Cancer is one of the most widely distributed disease in the world. About 90.5 million people have cancer. Only in 2017, 400.000 new types of cancer were discovered. Unfortunately, even in age of technologies, there is still no absolute cure for it. This project is a way to help the research advance and continue to fight the scourge of 21st century. The idea is to use Machine Learning to automate the process of diagnosing the tumors on different stages and make it easier to define the exact stage of cancer.
This is a way to apply the theoretical knowledge that we learnt from the discipline ‘Artificial life’.

Data and problem description:

We have data generated from an original dataset of real cancer data of 5000 patients.

The main goal was to improve the classification results by building a predictive model and training it to eventually improve weak baseline, so we can predict the advancement of the stage. The prediction should be at least greater than 10 percents. The data is a matrix of lines (number of patients) * columns (number of features per patient) . The features correspond to methylation information related to the medical condition of each patient.
This challenge required substantial preprocessing, including a matrix factorisation(Figure 2 and 3).The wanted result of this project is a possibility to identify 10 different cancer stages among a specific population.

Used methods (Description of our algorithms) :

For the **preprocessing** : We use scikit-learn methods, here is the pseudo-code :

```
import pandas as pd                                  // Importing the librairies we need
From data.io import read as df

Data = read_as_df('data_dir + '/' + 'data_name')     // Importing the dataset

X = dataset.iloc[:, :-1].values                      // Create a matrix of features in our dataset

From sklearn.preprocessing import Imputer            // Taking care of missing data
Imputer = Imputer(missing_values = "NaN",            // With the class called imputer in sklearn.preprocessing
          strategy = "mean", axis = 0)                  we will search for missing data
Imputer = imputer.fit(X[:,1:3])
X[:, 1:3] = imputer.transform(X[:, 1:3])             // Now we will just replace the missing values with
                                                        the mean of the column by the method transform.

from sklearn.model_selection import train_test_split // We separe the data into three datasets, for training,
X_train, X_test, Y_train, Y_test =                      testing and validation.
           train_test_split(X,Y, test_size=0.2)

from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()

X_train = sc_X.fit_transform(X_train)                // We applied a feature scaling to the data :
X_test = sc_X.transform(X_test)                         It is a method used to standardize the range of
                                                        independent variables or features of data.
```

The product we got in the end after preprocessing is the final training set.

For the **classifier** : We tried out several ways of performing supervised learning (via scikit-learn methods. First, we chose (among the methods that we tried) the method of the "Random Forest". In the end, we decided to use a multiclass perceptron (because it's a classifier that we know better). We chose to implement our own algorithm :

```
def fit(self,X,y) :
    '''
    This function train our model
    X: Training data matrix of dim num_train_samples * num_feat.
    y: Training label matrix of dim num_train_samples * num_labels.
    '''

    self.w = np.zeros((self.NBCLASSES,X.shape[1])) # initialization of wieghts

    arg_max, predicted_class = 0, 0 #intilization of argmax and predictions
    nberreur = 0
    for it in range(self.NBEPOCH) :
        #print("Epoque n° :",it,"Nb d'erreur :",nberreur)
        nberreur = 0
        for j in range(X.shape[0]):
            # Prediction of the classe with a scalar (for each class)
            arg_max = 0
            for c in range(self.NBCLASSES) :
                cur = np.dot(X[j], self.w[c])
                if cur >= arg_max :
                    arg_max, predicted_class = cur, c

            # If the prediction is not correct, adjust weights
            if (y[j] != predicted_class) :
                nberreur = nberreur + 1
                self.w[int(y[j])] += X[j]*self.ETA
                self.w[predicted_class] -= X[j]
    self.is_trained = True
```

Compared to standard multiclass classifier as we can see on the Figure 5 , our classifier has slightly better results.

To choose the best value for the training rate, we used a function which computes the score for the training set, for each value of the training rate. We made a graphic, with and without preprocessing :

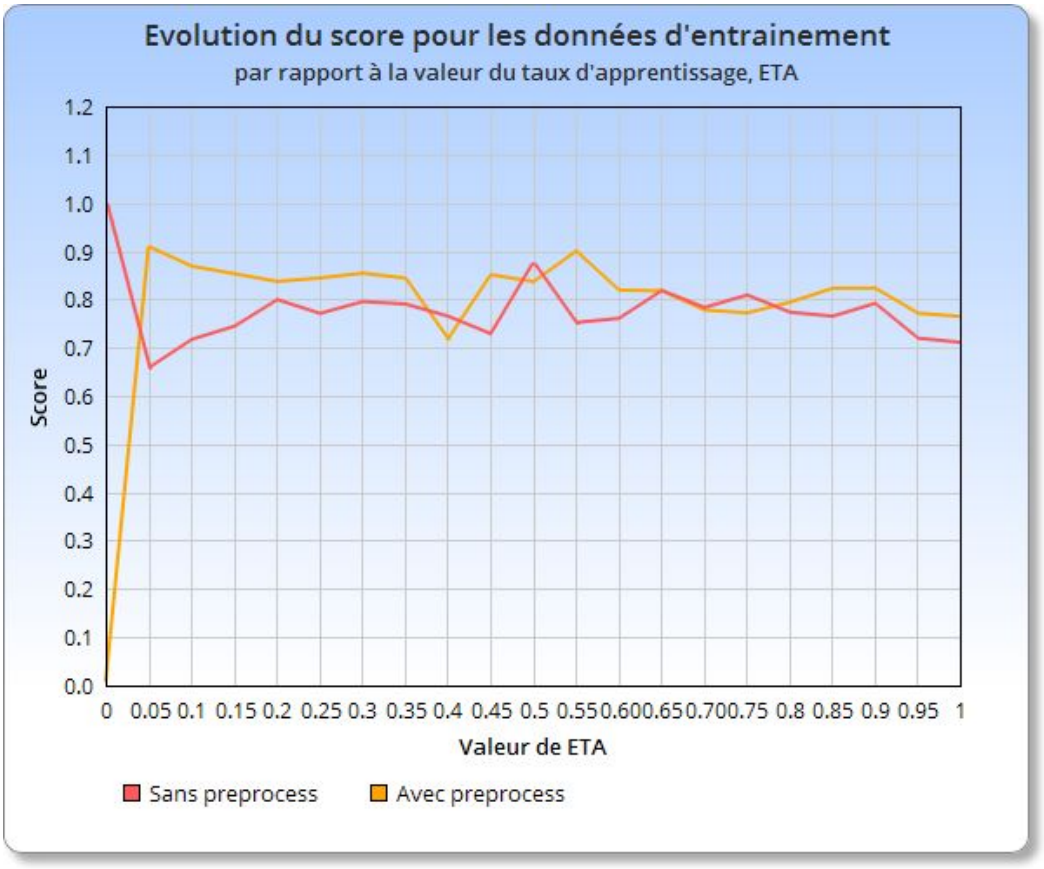


Figure 1

As we can see on this graphic(Figure 1) the results with preprocessing give us a higher final score than with the results without preprocessing which is logical because the goal of preprocessing is to filter the data.It includes cleaning, instance selection,feature extraction,normalization.

Results in the challenge :

Classifier given with the starting_kit (left matrix of Figure 2) :

With the test data, we can see that this classifier is not too efficient, unlike our classifier, because with a standardized confusion matrix, a classification system will be even better than its confusion matrix will approach a diagonal matrix.

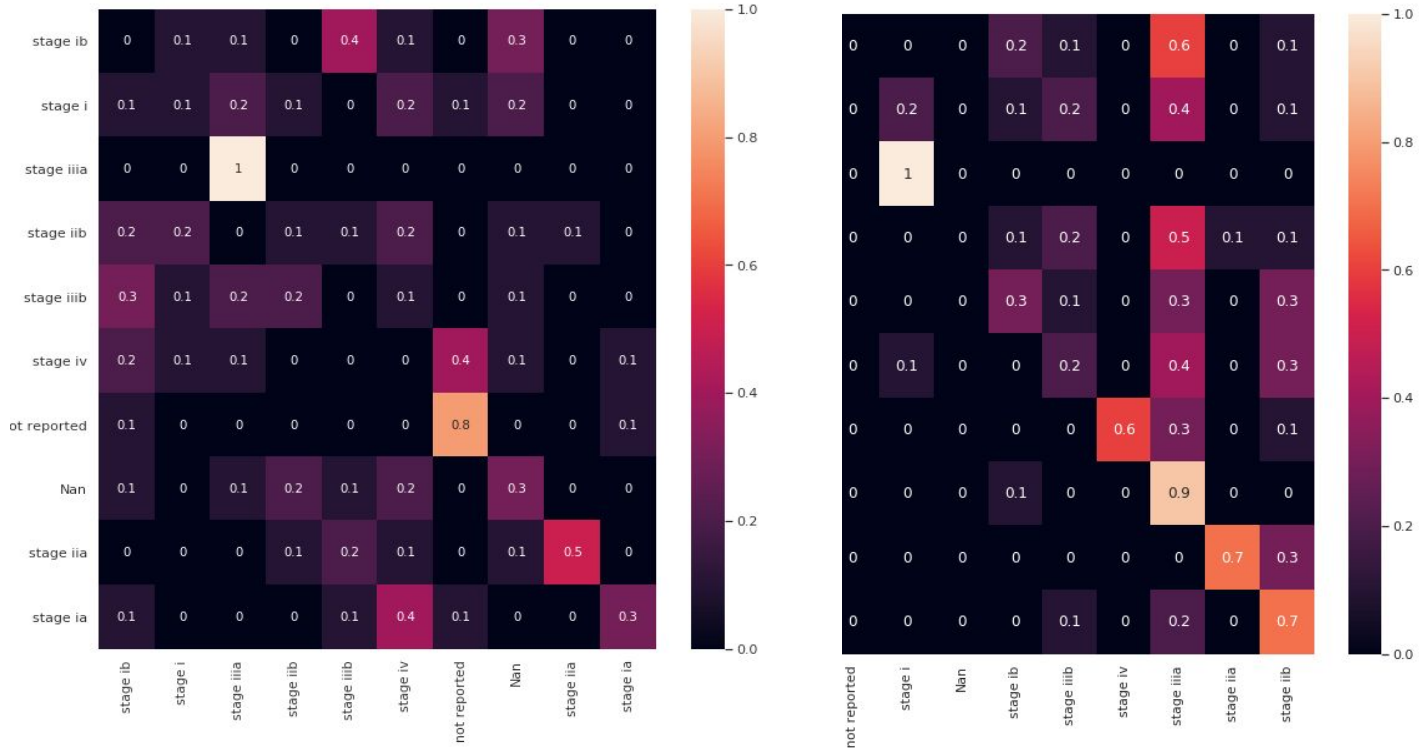
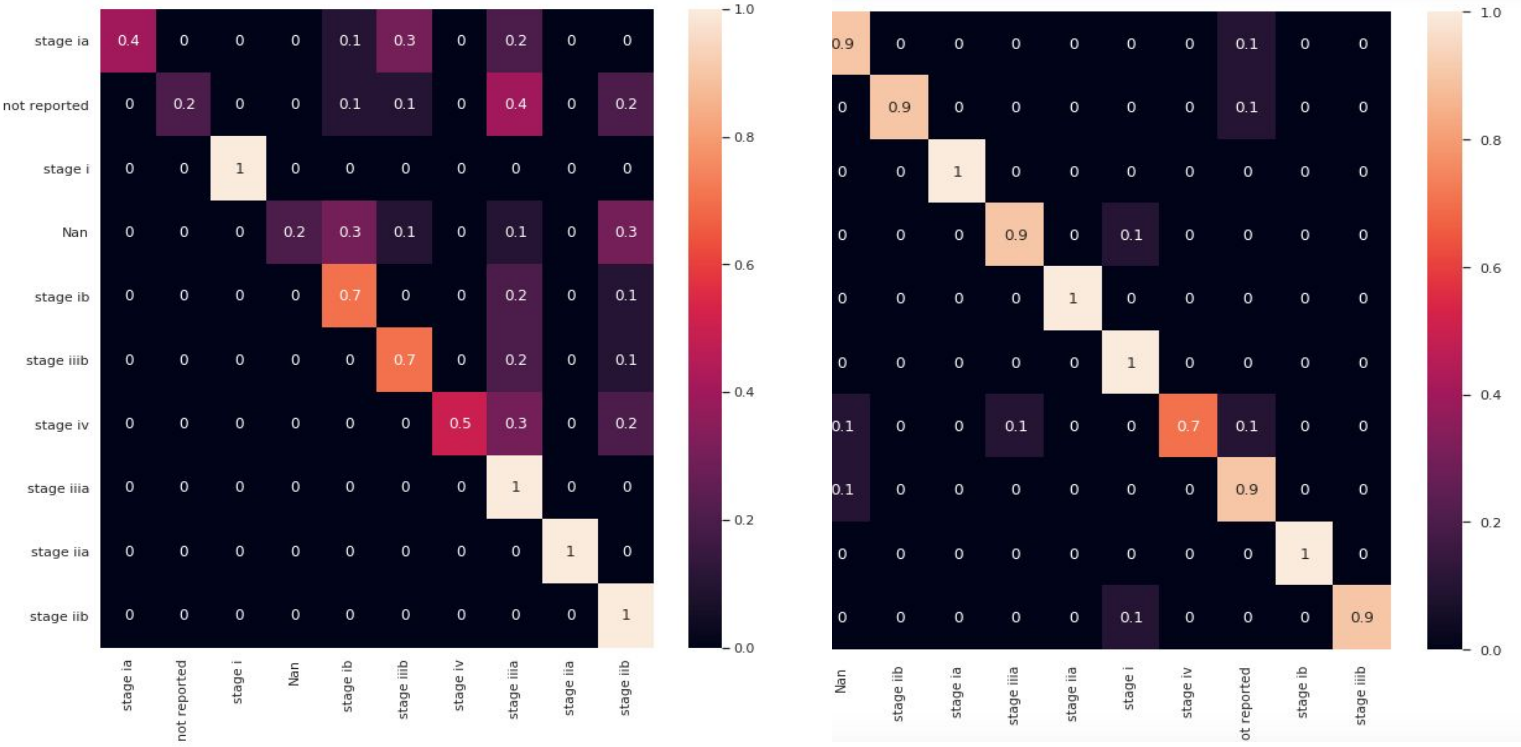


Figure 2

Our classifier (right figure of Figure 2) :

With our classifier, test Data is better ranked, because there is more zéros (0) outside the diagonal, which proves a better classification.

Figure 3



For Figure 3,we are in a similar case to the previous one, except that we measure the efficiency of the classifiers with the training data. And you notice that the diagonal of the matrix of confusion (with our classifier) is filled with '1' (right's figure), unlike the old classifier (left's figure). Which proves that our classifier is better than the original one of the starting_kit.

Leaderboard codalab :

RESULTS

#	User	Entries	Date of Last Entry	Prediction score ▲	Duration ▲	Detailed Results
1	HEALTH	15	04/07/19	1.0000 (1)	0.00 (1)	View
2	SaraHammache	1	04/05/19	1.0000 (1)	0.00 (1)	View
3	Cancer	29	04/10/19	0.9962 (2)	0.00 (1)	View
4	luc.gibaud	4	01/24/19	0.9843 (3)	0.00 (1)	View
5	Malikkazi	3	01/24/19	0.9804 (4)	0.00 (1)	View
6	martin.bauw	32	01/24/19	0.9721 (5)	0.00 (1)	View
7	doctor	30	04/09/19	0.8663 (6)	0.00 (1)	View
8	Cure	23	04/10/19	0.8574 (7)	0.00 (1)	View
9	takfarinas.nait-larbi	143	04/09/19	0.8136 (8)	0.00 (1)	View
10	FACTOR	33	04/10/19	0.6850 (9)	0.00 (1)	View
11	Zhengying	4	02/08/19	0.3589 (10)	0.00 (1)	View
12	OmarAbdoulayeBADIANE	7	03/26/19	0.2399 (11)	0.00 (1)	View

Our final score was 0.6850 , which is not bad, but still we need a lot of improvement as the best score possible is 1.

Contribution and the distribution of the duties :

In the beginning we were supposed to be the group of 6 people, but in the end we were left only with 4 people. The hardest thing about working in a group is to make sure that everyone is participating.
Noé ROBIN and **Omar Abdoulaye BADIANE** were responsible for the Preprocessing part and Classifier. **Iuliia LOGVINOVA** and **Aline BOUBEE DE GRAMONT** - for data visualization.
At the start, we wanted to use the method of the "Random Forest" for the Classifier part , but we ended up using a Multiclass Perceptron as we are more familiar with this method.
Even with only half of group, we managed to organise the process and to finish the challenge.

Discussion and conclusion :
This project helped us learn about teamwork: how to manage the team, how to distribute tasks between members of group. It's quite hard to organise the teamwork well and to establish a good communication between members.

It gave a possibility of using our knowledges about Machine Learning and of actually applying them in a life-related situation. It helped to learn more about it, too.
One of the main difficulty of this project was to deeply understand all of the datas and the different ways to manipulate them.
The results we got are really encouraging, but we still have place to improve our algorithms.
There's no limit for improving the results, so the process is almost never-ending.

Message for future students : this project is quite interesting and motivating. We hope you will choose to work on this project next year too, because this is a project that can really help advance the researches for cancer treatments in future. And it will make you work in conditions similar to those in real companies : in a group of people with different opinions and etc. That's a good training for future, too.

Bibliography :

- Cours de Vie Artificielle L2, Aurélien Decelle, 2018
- scikit documentation : https://scikit-learn.org/stable/documentation.html
- pandas and pandas.Dataframe documentation : https://pandas.pydata.org/pandas-docs/version/0.23.4/generated/pandas.DataFrame.html
- files README.ipynb given with TP1, TP2, and the starting kit.
- http://test.egc.asso.fr/wp-content/uploads/egc2013_atelier_cidn.pdf
- Random Forest : http://pages.cs.wisc.edu/~matthewb/pages/notes/pdf/ensembles/RandomForests.pdf
- https://fr.wikipedia.org/wiki/Perceptron
- https://openclassrooms.com/fr/courses/4470406-utilisez-des-modeles-supervises-non-lineaires/4730716-entrainez-un-reseau-de-neurones-simple
- The challenge itself: https://codalab.lri.fr/competitions/333?secret_key=5a2afb10-c5a0-49f2-a8b5-2a57370e39f6
- https://en.wikipedia.org/wiki/Random_forest

Bonus page :

The macro-average precision metric :
The submissions are evaluated using the macro-average precision metric. A macro-average will compute the metric independently for each class and then take the average (hence treating all classes equally) contrary to whereas a micro-average which will aggregate the contributions of all classes to compute the average metric.

The evaluation does not involve the matrix factorization. However, the quality of this factorization is both important to obtain good classification results and to study biological insights.

Cross-validation: Cross Validation is a technique which involves reserving a particular sample of a dataset on which you do not train the model. Later, you test your model on this sample before finalizing it.

Here are the steps involved in cross validation:

1. You *reserve* a sample data set
2. Train the model using the remaining part of the dataset
3. Use the reserve sample of the test (validation) set. This will help you in gauging the effectiveness of your model's performance. If your model delivers a positive result on validation data, go ahead with the current model. It rocks!

Learning the parameters of a prediction function and testing it on the same data is a methodological mistake: a model that would just repeat the labels of the samples that it has just seen would have a perfect score but would fail to predict anything useful on yet-unseen data. This situation is called overfitting. To avoid it, it is common practice when performing a (supervised) machine learning experiment to hold out part of the available data as a test set X_test, y_test. Note that the word "experiment" is not intended to denote academic use only, because even in commercial settings machine learning usually starts out experimentally. Here is a flowchart of typical cross validation workflow in model training. The best parameters can be determined by grid search techniques.

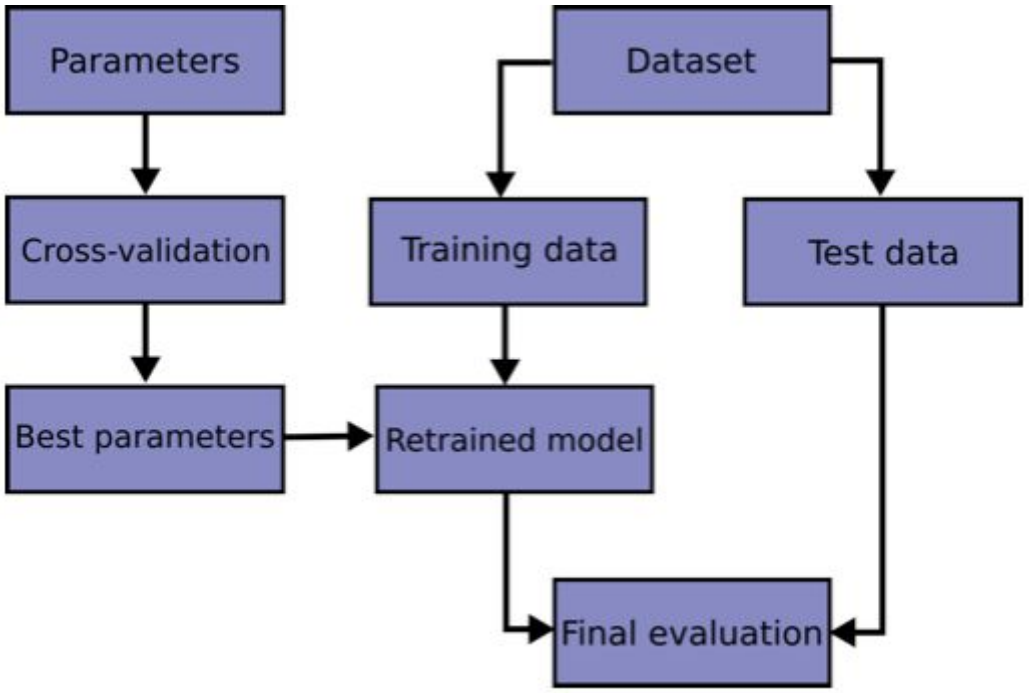


Figure 4

Overfitting :

Overfitting is a modeling error which occurs when a function is too closely fit to a limited set of data points. Overfitting the model generally takes the form of making an overly complex model to explain idiosyncrasies in the data under study.A key challenge with overfitting, and with machine learning in general, is that we can't know how well our model will perform on new data until we actually test it.

To address this, we can split our initial dataset into separate training and test subsets.

Table 1 : Statistic on the data :

Dataset	**Training**	**Validation**	**Test**
Number of Examples	4000	500	
Number of Features	20 000 000	2 500 000	
Has Categorical variables ?	No		
Sparsity	0.00045 %	0.000435 %	

Has missing data ?	Yes (Class 6, 8)	No
Number of examples in each class	400 examples per classes	Unknown

- A categorical variable is a variable that can take on one of a limited, and usually fixed number of possible values. Here, it is not the case.
- To calculate sparsity, we used a simple program we wrote on jupyter to count the zeros in the matrices, and we divided by the total number of elements.
- We have no information about classes 6 and 8 when we describe the datas.
- The number of examples in each class is known for the training set, but not for the two others sets because we don't have the labels.

Table 2 : Results of Classics (classifiers and our classifier) :

Dataset	**Training**	**Cross-Validation**	**Validation**
Basic Estimator	0.8991	0.6872	0.87
Naive Bayes Classifier	0.3860	0.8233	0.81
SVM	0.8964	0.834	0.93
Decision Tree	0.8996	0.8677	0.86
Random Forest	0.9980	0.8666	0.95
Perceptron	0.2100	0.6581	0.75
Our classifier (perceptron)	0.9090	0.7866	0.6850

- We show here the proportion of error, 1 is the best score.
- To obtain our results, we have modified the file "sample code submission", and chose the appropriate method of scikit learn.
- Using cross-validation(Figure 4), the training data is split into multiple training/test folds.
- Validation refer to the score on Codalab (1 is a perfect score)
- For perceptron and Naive Bayes Classifier for training the results seem to be lower than they should be, so probably there was an error.

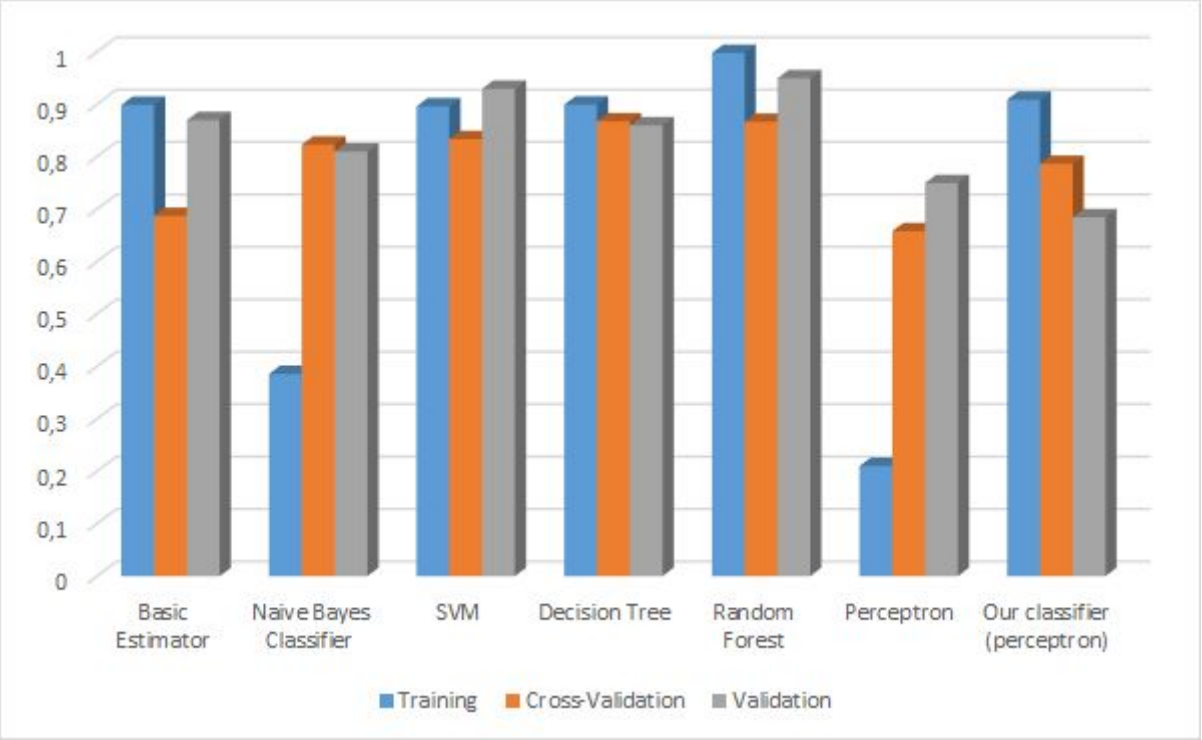


Figure 5

With histogram we can compare the efficiency of different algorithms much easier. So, Figure 5 is a better representation of results of the Table 2. We wanted to use Random Forest in the beginning, so here is a short explication of what this algorithm does. Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set. But in the end we used a perceptron which we studied before in Artificial Intelligence class. And here is a little explication of it(Figure 6).

The Perceptron is inspired by the information processing of a single neural cell called a neuron. A neuron accepts input signals via its dendrites, which pass the electrical signal down to the cell body. It is closely related to linear regression and logistic regression that make predictions in a similar way (e.g. a weighted sum of inputs).The weights of the Perceptron algorithm must be estimated from your training data using stochastic gradient descent.

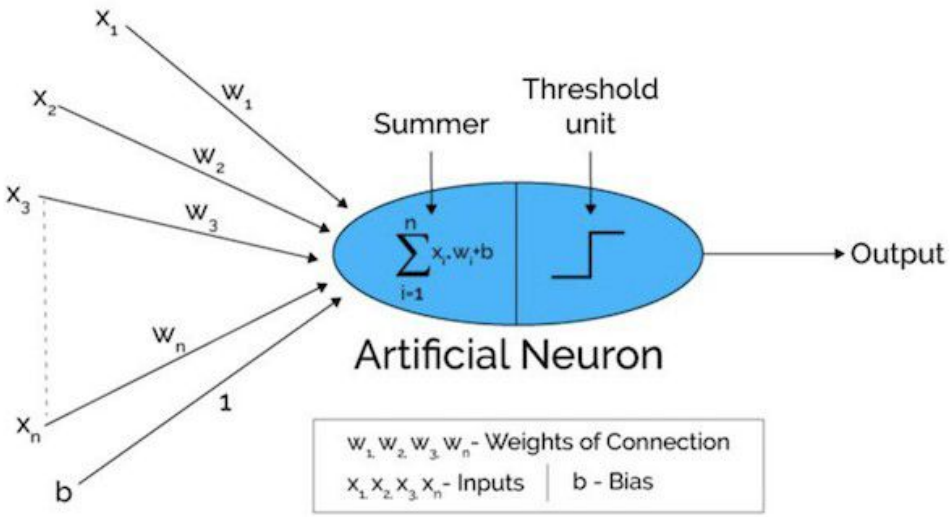

x1
x2
x3
xn
w1
w2
w3
wn
b
1
Summer
Threshold unit
Output
Artificial Neuron
w1, w2, w3, wn - Weights of Connection
x1, x2, x3, xn - Inputs
b - Bias

Figure 6