Project 3 on Machine Learning

Data Analysis and Machine Learning FYS-STK3155
Department of Physics, University of Oslo, Norway

Alatorre Flores Jessica

Abstract:

The main objective of this project is to put in practice everything learned during the course and try some of the different algorithms of machine learning discussed in class with real data in order to be able to visualize in a better way the scope that these methods have and thus also be able to compare the performance between them using the methods to measure the performance that we also discussed during the course.

For the development of this project the professor's proposed example was chosen, this is to analyze the bank's credit card data with the binary result of classification between credible and not-credible clients. This data is provided by the UCI site and is of special interest because there is an article that recently used the same data in an analysis of ML.

This article interests us because it can help us compare the results we are getting against the results obtained by the authors. Therefore, with this we can get a bigger picture about the methods we are using and have a broader discussion.

The specifications for this project were to use two of the machine learning methods that we discussed in the course, but I used three of them. First, I analyzed Logistic Regression, then a Neural Network based on the backpropagation algorithm and finally as an experimental way to make a different one from those treated in the article, I tried with Random Forest using the functionality provided by sckit-learn.

The results were observed based on the accuracy of the classifier, also the confusion matrix for the result between the actual data with the predicted data. Metrics like precision, recall and F1-score were also observed, and similarly to the article, the graphs of cumulative gain were also computed.

Introduction:

Nature of the problem:
With the data set that was chosen, about the information of the bank accounts, we find a problem of binary classification in which it analyzes information of customers of Taiwan through 23 explanatory variables to obtain the result that tells us if a client is reliable or not to pay their debts. This is explained with the probability of default that means that the borrower will be unable or unwilling to repay its debt in full or on time. It can also be seen

as a regression problem when calculating the result of predictive accuracy of the estimated probability.

For the analysis of this work I tried with 3 different machine learning methods. The first two, are methods that are also used in the article, and this was with the intention of compare my obtained results whit the results obtained by the authors. The third method random forest and this one was chosen as an experimental way to work with one different from those presented in the article.

Logistic regression:
It is a method used for classification problems that uses a logistic function to model a binary outcome (two possible values). The goal is to identify the classes to which new unseen samples belong where each datapoint is deterministically assigned to a category (i.e. $y_i = 0$ or $y_i = 1$). The advantage of this method is that it uses a simple function that gives the probability for each datapoint to belong to a category.
This approach is a case of linear regression model thus it cannot deal well with a non-linear problem.

Artificial Neural Network:
An artificial neural network is a computing system which is vaguely inspired by the biological neural networks, where the system itself learns to develop relationships between inputs and outputs by balancing some weights and using activation functions. An ANN is composed of connected units or nodes which are called the neurons, distributed generally with a structure of the input layer, one or more hidden layers, and the output layer.
Backpropagation distributes the error term back up through the layers, by modifying the weights at each node.
This method can handle non-linear problems, but it does not result in a simple probabilistic classification.

Random Forest:
This machine learning method is capable of regression and classification and it belong to a class of machine learning algorithms called ensemble methods that are the methods that combines several models to solve a single prediction problem. It works by generating multiple classifiers/models which learn and make predictions independently. Those predictions are then combined into a single (mega) prediction that should be as good or better than the prediction made by any one classifier.
Random forest is a brand of ensemble learning, as it relies on an ensemble of decision trees.

About the dataset:
As we can find in the site of UCI, the research employed a binary variable, default payment (Yes = 1, No = 0), as the response variable.
And the following 23 variables are used as explanatory variables:

- X1: Amount of the given credit (NT dollar): it includes both the individual consumer credit and his/her family (supplementary) credit.

- X2: Gender (1 = male; 2 = female).
- X3: Education (1 = graduate school; 2 = university; 3 = high school; 4 = others).
- X4: Marital status (1 = married; 2 = single; 3 = others).
- X5: Age (year).
- X6 - X11: History of past payment. We tracked the past monthly payment records (from April to September, 2005) as follows: X6 = the repayment status in September, 2005; X7 = the repayment status in August, 2005; . . .;X11 = the repayment status in April, 2005. The measurement scale for the repayment status is: -1 = pay duly; 1 = payment delay for one month; 2 = payment delay for two months; . . .; 8 = payment delay for eight months; 9 = payment delay for nine months and above.
- X12-X17: Amount of bill statement (NT dollar). X12 = amount of bill statement in September, 2005; X13 = amount of bill statement in August, 2005; . . .; X17 = amount of bill statement in April, 2005.
- X18-X23: Amount of previous payment (NT dollar). X18 = amount paid in September, 2005; X19 = amount paid in August, 2005; . . .;X23 = amount paid in April, 2005.

Implementation:

For all the machine learning methods that were used, it was followed the same process to analyze the data and evaluate the classifier, which was:

1. Preliminaries
   - Import libraries
   - Load the dataset
2. Read and structure the data
   - Structure the data in a dataframe
   - Separate variables
3. Create training and test data sets
   - With the functionality of sckit-learn
4. Train the classifier
5. Apply the classifier to the test set
   - Compute the predictions
6. Evaluate the classifier
   - Get the accuracy of the classifier
   - Compute the confusion Matrix
   - Get some more metrics as precision, recall and F-score
   - Visualize the cumulative gain graph

Whit the first two methods, Logistic Regression and Neural Networks, the codes that were developed in project 2 were used, but they were also compared with the results obtained when using the functionalities of sckit learn.
And with the last method, Random Forest, only the sckit-learn functionality was used.

For the first steps the scripts are omitted in this report due it's almost the same as what we have worked in the las two projects besides in not focus of interest for this project. The complete source code can be consulted in the given [github repository.]

I structured the data using pandas and once they were structured, I separated between the explanatory variables and the responses.
In order to explore more the data, the distribution of the responses was visualized.

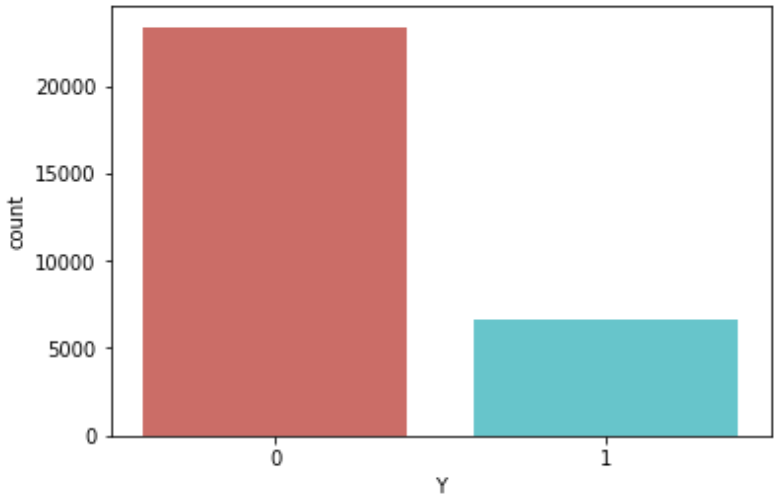

Fig. 1: Distribution of the binary outcome for the whole data set

As we can observe, the data is very unbalanced with the 77.88% for the value 0 which means that the clients are non-risky because there is no default payment, and only 22.12% for the risky clients.
This situation is an indicator that the methods may not perform very well, but it is also of interest look how each method face it and if any of them can make accurate predictions despite this issue.

Logistic Regression:

This method models the probability that each input belongs to a category. To generate this probabilities, logistic regression uses the Sigmoid function that gives outputs between 0 and 1, for all datapoint.

```
def sigmoid(z):
    return 1 / (1 + np.exp(-z))
```

where z is the dot prod between the explanatory variables (X) and the weights (theta)

```
z = np.dot(X, theta)
```

Then, the loss function is used to compute the measure for how well the algorithm is performing with the weights (theta) that have a random value to start.

```
def loss(h, y):
    return (-y * np.log(h) - (1 - y) * np.log(1 - h)).mean()
```

In order to minimize the loss function, the weights have to be varied, either increasing or decreasing the weights. For this we use the standard gradient descent deriving the loss function with respect to each weight. And then update the weights subtracting the derivate and repeat until reach the optimal values.

The predictions are computed by comparing the probabilities given by the sigmoid function with a threshold that usually is 0.5 what would be equivalent to round this probability.

The complete code for the logistic regression class is shown below.

```
class LogisticRegression:
    def __init__(self, lr=0.01, num_iter=100000,
      fit_intercept=True, verbose=False):
        self.lr = lr
        self.num_iter = num_iter
        self.fit_intercept = fit_intercept
        self.verbose = verbose

    def __add_intercept(self, X):
        intercept = np.ones((X.shape[0], 1))
        return np.concatenate((intercept, X), axis=1)

    def __sigmoid(self, z):
        return 1 / (1 + np.exp(-z))
    def __loss(self, h, y):
        return (-y * np.log(h) - (1 - y) * np.log(1 - h)).mean()

    def fit(self, X, y):
        if self.fit_intercept:
            X = self.__add_intercept(X)

        # weights initialization
        self.theta = np.zeros(X.shape[1])

        #Standard Gradient Descent
        for i in range(self.num_iter):
            z = np.dot(X, self.theta)
            h = self.__sigmoid(z)
            gradient = np.dot(X.T, (h - y)) / y.size
            self.theta -= self.lr * gradient
```

```
    def predict_prob(self, X):
        if self.fit_intercept:
            X = self.__add_intercept(X)
        return self.__sigmoid(np.dot(X, self.theta))

    def predict(self, X):
        return self.predict_prob(X).round()
```

Then, an object of this class is created to perform the training using the train set and the predictions for the test set are computed.

To evaluate how well the classifier performed the accuracy was computed by getting the mean of the values that corresponded to the responses for the test set.

```
model = LogisticRegression(lr=0.1, num_iter=100000)
model.fit(X_train,Y_train)
preds = model.predict(X_test)
probas = model.predict_prob(X_test)
preds = preds.astype('int')

accuracy = (preds == Y_test).mean()
print("accuracy with my lr:",accuracy)
```

```
[out]:  accuracy with my lr: 0.78
```

As we can observe, the value of accuracy is pretty good but we can not say that we have a good classifier only with this measure. To have a better overview of the results the confusion matrix and some metric as precision, recall and f-score were also computed.

```
conf_mat = metrics.confusion_matrix(Y_test, preds)
ig, ax = plt.subplots(figsize=(3,3))
sns.heatmap(conf_mat, annot=True, fmt='d', cmap='YlGn')
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
print(classification_report(Y_test, preds))
```

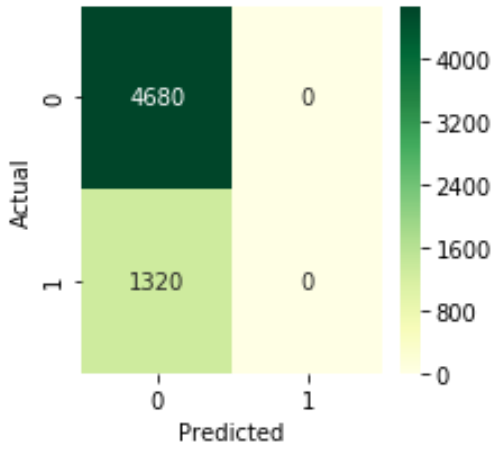

Fig 2. Confusion matrix for logistic regression.

	precision	recall	f1-score	support
0	0.78	1.00	0.88	4680
1	0.00	0.00	0.00	1320
micro avg	0.78	0.78	0.78	6000
macro avg	0.39	0.50	0.44	6000
weighted avg	0.61	0.78	0.68	6000

Table 1. metrics for logistic regression

As expected, with these values and especially with the confusion matrix we can see now that despite having a good value of accuracy, the classifier is not performing so well because is totally ignoring one class which is the class with less values and that is why is not reflected in the value of the accuracy.

Whit sckit-learn

Now, a comparation with sklear is performed to check if we have similar results.

```
from sklearn.linear_model import LogisticRegression

modelS = LogisticRegression(C=1e20)
modelS.fit(X_train, Y_train)
y_preds = modelS.predict(X_test)
probas = modelS.predict_proba(X_test)
```

```
[out]:  accuracy with my lr: 0.780
```

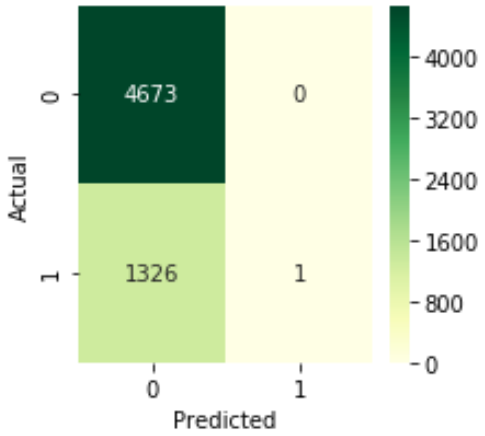

Fig 3. Confusion matrix for logistic regression with sklearn

As shown above, the results of logistic regression using the functionality of sckit-learn is the same as whit the developed code. So, we can conclude that it was not the implementation of the method but the distribution of the data that is not balanced.

Whit this conclusion is a bit difficult to compare the result with the result presented in the article because it is not informed if they performed a preprocessing in the data to balance the classes. However, it was convenient to graph the ROC curve and compute the area under the curve (AUC) which is the area between model curve and baseline curve.

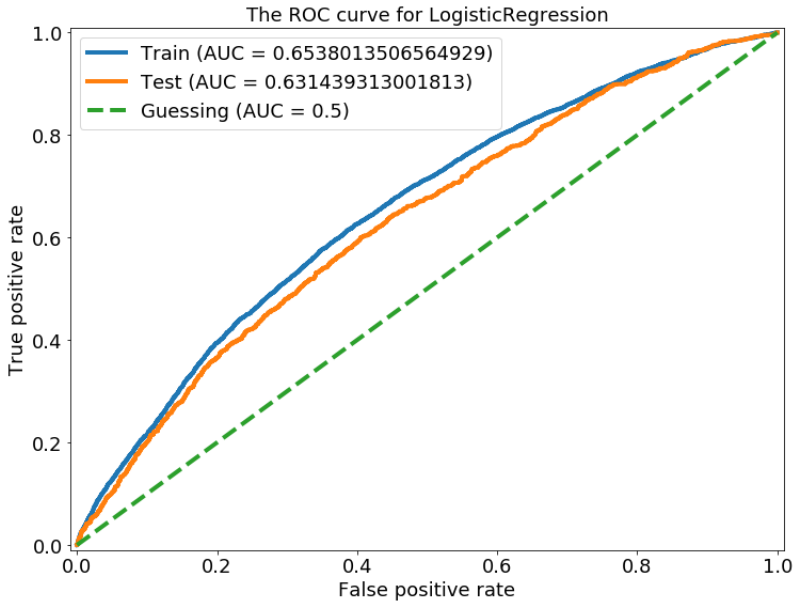

Fig. 4. ROC curve for Logistic Regression

Artificial Neural Networks:

The neural network connects linear and non-linear elements so that using different layers approximates any mathematical function which allows to separate classes with any type of boundary, either linear or non-linear.
The components of this method are firstly, the input data, then the neurons that are grouped by layers, starting with the input layer, one or more hidden layers and an output layer where we find the output data.

The process to follow is that the input that is a block of information, is transformed through the neurons that perform the linear combination of the weights with the inputs.

```
# Initialize weights:
np.random.seed(88)
w1 = 2.0*np.random.random((input_dim, hidden_dim))-1.0
#w0=(2,hidden_dim)
w2 = 2.0*np.random.random((hidden_dim, output_dim))-1.0
#w1=(hidden_dim,2)

#Calibratring variances with 1/sqrt(fan_in)
w1 /= np.sqrt(input_dim)
w2 /= np.sqrt(hidden_dim)
```

The weights give us to understand how important the information is, that is, how much it will affect to generate a result.

The other main component of a neural network is an activation function that in this case is the Sigmoid function. This function is to add a non-linear behavior to the information to compute the output probabilities.

```
def sigmoid(z, first_derivative=False):
    if first_derivative:
        return z*(1.0-z)
    return 1.0/(1.0+np.exp(-z))
```

To obtain the output values what is done is to obtain the results of the multiplication of the data by the weights, in all the neurons of each layer and using the activation function to obtain the output value by layers that result take it as input of the next layer and repeat the process until you reach the output layer. This procedure is called forward step.

```
h1 = sigmoid(np.matmul(X_train[index], w1))
logits = sigmoid(np.matmul(h1, w2))
probs = np.exp(logits)/np.sum(np.exp(logits), axis=1,  keepdims=True)
h2 = logits
```

We want to know whether the neural network is working well or not, therefore it is necessary to have a way to compute this measure, this is the so-called loss function.

```
L = np.square(Y_train[index]-h2).sum()/(2*N) +
reg_coeff*(np.square(w1).sum()+np.square(w2).sum())/(2*N)
```

The Backpropagation step is how the neural network learn to update the weights of a neural network to minimize the loss function. It is called backpropagation because is going from the last layer that is the output layer to the first layer, computing the gradient with respect of the weights to find the minimum error.

```
# Backward step: Error = W_l e_l+1 f'_l
#dL/dw2 = dL/dh2 * dh2/dz2 * dz2/dw2
dL_dh2 = -(Y_train[index] - h2)                                    #(N, 2)
dh2_dz2 = sigmoid(h2, first_derivative=True)             #(N, 2)
dz2_dw2 = h1                                  #(N, hidden_dim)
#Gradient for weight2:   (hidden_dim,N)x(N,2)*(N,2)
dL_dw2 = dz2_dw2.T.dot(dL_dh2*dh2_dz2) + reg_coeff*np.square(w2).sum()
dL_dz2 = dL_dh2 * dh2_dz2                                    #(N, 2)
dz2_dh1 = w2                                                 #z2 = h1*w2
dL_dh1 =  dL_dz2.dot(dz2_dh1.T)  #(N,2)x(2,hidden_dim)=(N, hidden_dim)
dh1_dz1 = sigmoid(h1, first_derivative=True)            #(N,hidden_dim)
dz1_dw1 = X_train[index]                                       #(N,2)
#Gradient for weight1:  (2,N)x((N,hidden_dim)*(N,hidden_dim))
dL_dw1 = dz1_dw1.T.dot(dL_dh1*dh1_dz1) + reg_coeff*np.square(w1).sum()
```

After applying all those calculations, the weights must be updated by subtracting the derivate multiplied by the learning rate. This process is repeated a certain number of times to find the optimal parameters for the neural network.

```
w2 += -learning_rate*dL_dw2
w1 += -learning_rate*dL_dw1
```

The complete code for the neural network is shown below.

```
def load_data(N=300):
    rng = np.random.RandomState(0)
    X = rng.randn(N, 2)
    y = np.array(np.logical_xor(X[:, 0] > 0, X[:, 1] > 0),
dtype=int)
    y = np.expand_dims(y, 1)
    y_hot_encoded = []
    for x in y:
        if x == 0:
            y_hot_encoded.append([1,0])
        else:
            y_hot_encoded.append([0, 1])
    return X, np.array(y_hot_encoded)

def sigmoid(z, first_derivative=False):
    if first_derivative:
        return z*(1.0-z)
    return 1.0/(1.0+np.exp(-z))
```

```
def tanh(z, first_derivative=True):
    if first_derivative:
        return (1.0-z*z)
    return (1.0-np.exp(-z))/(1.0+np.exp(-z))

def predict(data, weights):
    h1 = sigmoid(np.matmul(data, weights[0]))
    logits = np.matmul(h1, weights[1])
    probs = np.exp(logits)/np.sum(np.exp(logits), axis=1,
keepdims=True)
    return np.argmax(probs, axis=1)

N = 1
input_dim = int(X_train.shape[1])
hidden_dim = 10
output_dim = 1
num_epochs = 10000
learning_rate= 1e-3
reg_coeff = 1e-6
losses = []
accuracies=[]

# Initialize weights:
np.random.seed(88)
w1 = 2.0*np.random.random((input_dim, hidden_dim))-1.0
w2 = 2.0*np.random.random((hidden_dim, output_dim))-1.0

#Calibratring variances with 1/sqrt(fan_in)
w1 /= np.sqrt(input_dim)
w2 /= np.sqrt(hidden_dim)

for i in range(num_epochs):
    index = np.arange(X_train.shape[0])[:N]
    #is want to shuffle indices: np.random.shuffle(index)

    # Forward step:
    h1 = sigmoid(np.matmul(X_train[index], w1))
    logits = sigmoid(np.matmul(h1, w2))
    probs = np.exp(logits)/np.sum(np.exp(logits), axis=1,
keepdims=True)
    h2 = logits

    # Definition of Loss function: mean squared error plus Ridge
regularization
    L = np.square(Y_train[index]-h2).sum()/(2*N) +
reg_coeff*(np.square(w1).sum()+np.square(w2).sum())/(2*N)

    losses.append([i,L])

    # Backward step: Error = W_l e_l+1 f'_l
    #       dL/dw2 = dL/dh2 * dh2/dz2 * dz2/dw2
    dL_dh2 = -(Y_train[index] - h2)
    dh2_dz2 = sigmoid(h2, first_derivative=True)
    dz2_dw2 = h1
    #Gradient for weight2:   (hidden_dim,N)x(N,2)*(N,2)
    dL_dw2 = dz2_dw2.T.dot(dL_dh2*dh2_dz2) +
reg_coeff*np.square(w2).sum()

    dL_dz2 = dL_dh2 * dh2_dz2
    dz2_dh1 = w2
    dL_dh1 =  dL_dz2.dot(dz2_dh1.T)
dh1_dz1 = sigmoid(h1, first_derivative=True)
    dz1_dw1 = X_train[index]
    #Gradient for weight1:  (2,N)x((N,hidden_dim)*(N,hidden_dim))
    dL_dw1 = dz1_dw1.T.dot(dL_dh1*dh1_dz1) +
reg_coeff*np.square(w1).sum()
```

Now we can compute the predictions for the test set with method predict().
The accuracy is computed with the mean of the comparison between the responses of the test set and the prediction to check how well is performing the classifier.

```
y_pred = predict(X_test, [w1, w2])

accuracy = (y_pred == Y_test).mean()
print("accuracy of the classifier:",accuracy)
```

```
[out]:  accuracy of the classifier: 0.7788333333333334
```

In the same way as the previous method, we can observe that the value of accuracy is "good" but again this is not enough to evaluate the behavior of the classifier.

To continue evaluating the performance of the method, the confusion matrix and some metrics as precision, recall and f-score are also computed.

```
conf_mat = confusion_matrix(Y_test, y_pred)
fig, ax = plt.subplots(figsize=(3,3))
sns.heatmap(conf_mat, annot=True, fmt='d', cmap='YlGn')
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
print(classification_report(Y_test, y_pred))
```

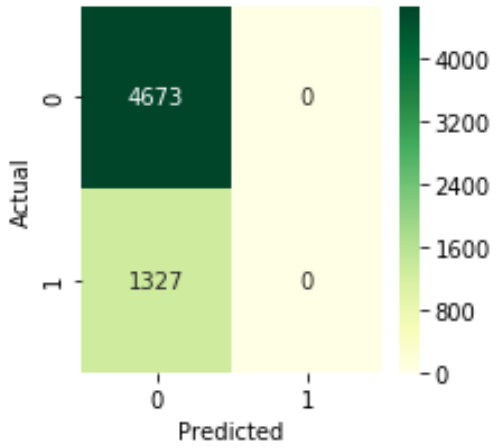

Fig 5. Confusion matrix for neural network method

	precision	recall	f1-score	support
0	0.78	1.00	0.88	4673
1	0.00	0.00	0.00	1327
micro avg	0.78	0.78	0.78	6000
macro avg	0.39	0.50	0.44	6000
weighted avg	0.61	0.78	0.68	6000

Table 2. Matrics for Neural Network

We can observe that again we are facing the same problem, the classifier is having a good value of accuracy, but it is misclassifying completely the minority class, this is another proof that shows that the problem is with the dataset and not with the classifier.

Once again is convenient to use the functionality of sckit-learn to compare the performance and check if there is some difference.

```
from sklearn.neural_network import MLPClassifier
clf = MLPClassifier(random_state=1)
clf.fit(X_train, Y_train)
Y_pred = clf.predict(X_test)

accuracy = (Y_pred == Y_test).mean()
print("accuracy of the classifier using Sckit:",accuracy)
```

```
[out]:  accuracy of the classifier using Sckit: 0.7281666666666666
```

Computing the confusion matrix and the metrics…

```
conf_mat = confusion_matrix(Y_test, Y_pred)
fig, ax = plt.subplots(figsize=(3,3))
sns.heatmap(conf_mat, annot=True, fmt='d', cmap='YlGn')
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
print(classification_report(Y_test, Y_pred))
```

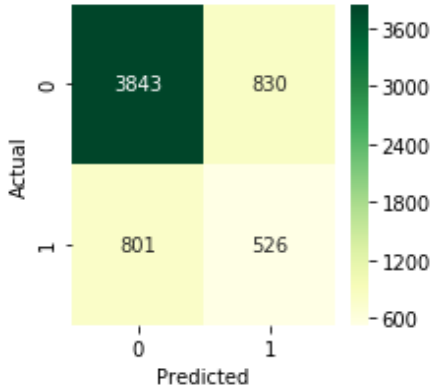

Fig. 6. Confusion matrix for Neural Network using Sklearn

	precision	recall	f1-score	support
0	0.83	0.82	0.82	4673
1	0.39	0.40	0.39	1327
micro avg	0.73	0.73	0.73	6000
macro avg	0.61	0.61	0.61	6000
weighted avg	0.73	0.73	0.73	6000

Table 3. Metrics for Neural Network using Sklearn

In this occasion, the problem of the classifier ignoring completely one class is not presented but looking at the confusion matrix in Fig 6, we can observe that the classification is not optimal, there are more datapoints misclassified that the correctly classified in the minority class. Actually, we can also observe that the value of the accuracy is lower than the case where one class is ignored.

In order to continue with the same line of analysis, the graph of the ROC curve is computed too.

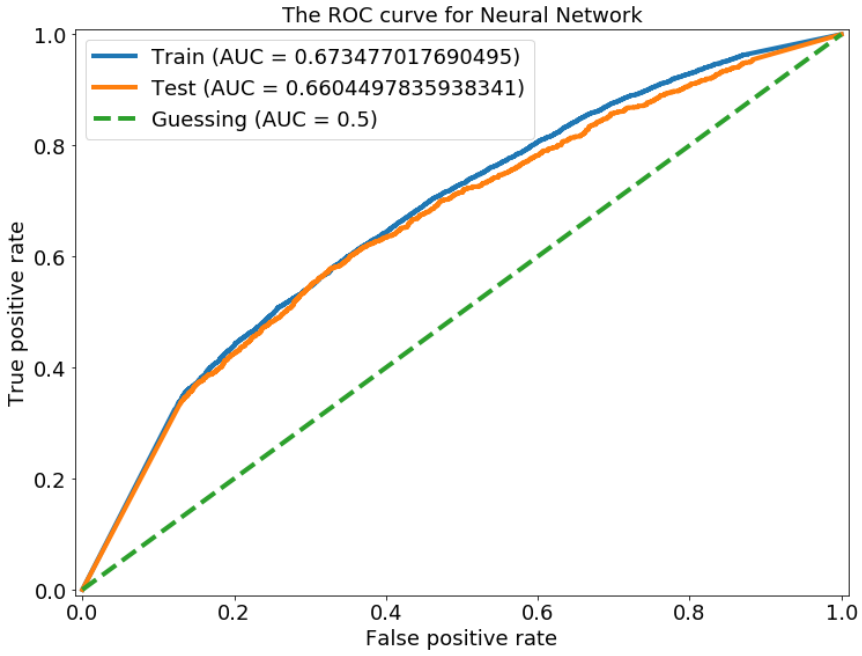

Fig. 7. ROC curve for the Neural Network

This curve in fig 7, was computed with the probabilities from the sckit-learn method of the Neural Network. We can observe the curves from the train and the test and the baseline or the guessing with the value of 0.5.

Random Forest:

For this machine learning method, the function sklearn.ensemble.RandomForestClassifier was used.
This function fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. (see documentation) .

The process is again the same, first import the model, instantiate the model an then train with the train set. After this we use the model to predict the values for the test set.

```
# Import the model we are using
from sklearn.ensemble import RandomForestClassifier
# Instantiate model with 1000 decision trees
rf = RandomForestClassifier(n_jobs=2, random_state=0)
# Train the model on training data
rf.fit(X_train, Y_train);

# Use the forest's predict method on the test data
predictions = rf.predict(X_test)
```

Then, we proceed to evaluate how was well was the performance for this classifier. The accuracy, the confusion matrix and the same metrics as the used previously were computed.

```
from sklearn import metrics
print("Accuracy:",metrics.accuracy_score(Y_test, predictions))

from sklearn.metrics import confusion_matrix
conf_mat = confusion_matrix(Y_test, predictions)
fig, ax = plt.subplots(figsize=(3,3))
sns.heatmap(conf_mat, annot=True, fmt='d', cmap='YlGn')
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
print(classification_report(Y_test, predictions))
```

```
[out]: Accuracy: 0.813
```

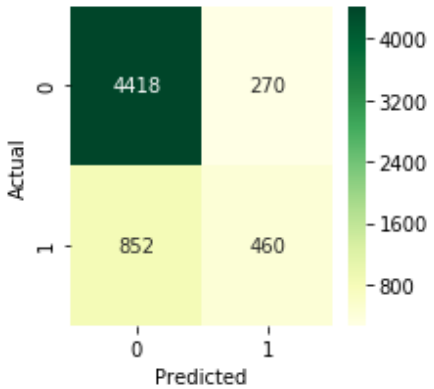

Fig. 8. Confusion matrix for Random Forest using sklearn

	precision	recall	f1-score	support
0	0.84	0.94	0.89	4688
1	0.63	0.35	0.45	1312
micro avg	0.81	0.81	0.81	6000
macro avg	0.73	0.65	0.67	6000
weighted avg	0.79	0.81	0.79	6000

Table 3. Metrics for Random Forest using sklearn

This time, the results were a bit better, we can observe that the value of accuracy very good, it is the highest of all of the used methods and this time the classifier is not ignoring the minority class, we can observe this also on the table 3 with the metrics for both classes, for example, the precision is lower for the value of the minority class (y=1, risky clients) but is not that bad compared with the other method that were used before.

The graph with the ROC curve was again computed for this case.

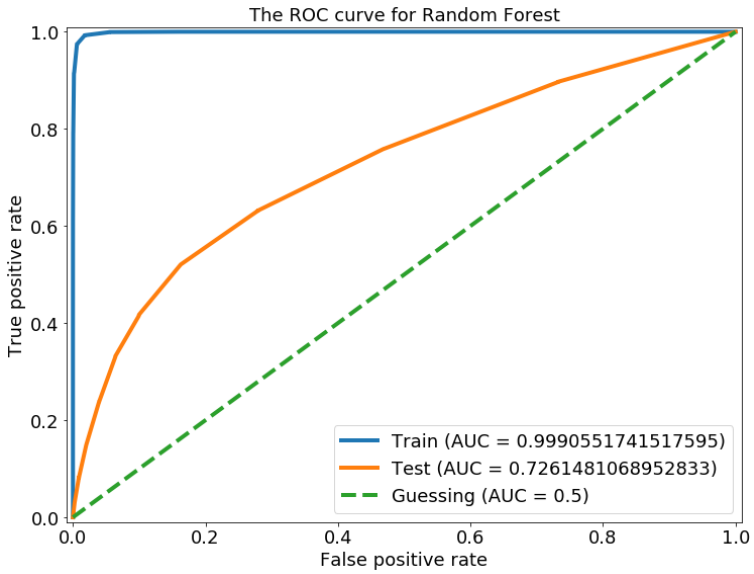

Fig. 9. ROC curve for the Random Forest using sklearn

We can observe in fig 9 that this time a different graph was gotten. For the train set, the area under the curve is almost 1, and for the test, the area under the curve is 0.76 which is a good value, or al least is much better that with the other classifiers.

Conclusions:

After use the three different method and comparing the first two with the performance using sklearn, we can conclude in first place, that the data is not optima, and this has a serious impact on the efficiency of the classifier. Being a large data set with a very unbalanced values for the responses variables is not cool because the classifier does not train properly and as we observed that for the model there is not apparent problem because it still has a good value of accuracy.

There are some methods to solve this kind of problem, where one modified the original data either using over-sampling the minority class or under-sampling the majority class, there is also methods that use a combination of both. I didn't apply one of these techniques because I wanted to focus more in the study of the classifiers and use the different methods with the original data to check if any of them could present an accurate classification despite this situation.

Related with this issue, the next conclusion would be that none of the method managed to achieve a very good classification. The best performance was provided by Random Forest but although it has a good accuracy and it predict for the two classes better that the other methods, we can see that the predicted values the minority class still have a significant incorrect classification percentage.

Another advantage of the Random Forest classifier was the execution time, this was significantly lower compared to the other models.

It was not very possible to compare the obtained results with the results from the article due the problem with the data. In the article is not mentioned how did the authors deal with the situation, if they used a transformation to the data or if they used the models without consider that they were classifying only for one class.

The general conclusion of the project is that the main objective was reached because different machine learning methods were used and the understanding of them was clearer and deeper by using the real data of the credit card data set and there machine learning methods were evaluated and compared between themselves in a way of continuation to the course of applied data analysis and machine learning.

References.

- Hastie, Trevor, Tibshirani, Robert, Friedman, Jerome . (2009). The Elements of Statistical Learning. Springer.
- Lecture Notes for the course FSY-STK 4155, Morten Hjorth-Jensen, Department of physics, University of Oslo (2018). https://compphysics.github.io/MachineLearning/doc/web/course.html
- Susan Li, Building A Logistic Regression in Python (2017). https://towardsdatascience.com/building-a-logistic-regression-in-python-step-by-step-becd4d56c9c8
- Michael Nielsen, Oct 2018, *Neural Networks and Deep Learning*, http://neuralnetworksanddeeplearning.com/
- Tavish Sriva Satava, Important Model Evaluation Error Metrics Everyone should know, (2016). https://www.analyticsvidhya.com/blog/2016/02/7-important-model-evaluation-error-metrics/
- Ricco Rakatomalala, Universitè Lyon, Machine Learning with scikit-learn, http://eric.univ-lyon2.fr/~ricco/cours/slides/PJ%20-%20en%20-%20machine%20learning%20avec%20scikit-learn.pdf
- Ligdi Gonzalez, (2018), Aprendizaje Supervisado: Logistic Regression, http://ligdigonzalez.com/aprendizaje-supervisado-logistic-regression/
- Joshep Eddy, New York University, (2017), Can deep learning handle imbalanced data?, https://www.quora.com/Can-deep-learning-handle-imbalanced-data
- Random Forest, https://en.wikipedia.org/wiki/Random_forest#Algorithm
- Will Koehrsen,(2017), Random Forest in Python, https://towardsdatascience.com/random-forest-in-python-24d0893d51c0