Toolbox 3: Image and Patter Recognition

Project 1: Tool wear monitoring

Rida LEFDALI

1 Introduction:

The aim of this project is to build a classifier to classify tools used in mailing process into two categories: “worn tool” and “unworn tool”.
firstly, we are going to work on logistic regression as classifier, and explain all the steps to build this classifier, then, we will try other classifiers and compare between them in other to choose the best classifier. So, we are going to:

1. Describe a set of images by means of a set region descriptor.

2. Using the computed descriptors, we are going to simulate a classification experiment and calculate some performance metrics from the results of the Logistic Regression classifier. In this experiment, we are going to divide the dataset into two disjoint sets: training and test

2 The Dataset

The dataset that we dispose for this project is a set of binary images which contains the regions corresponding to cutting edges of insert tools used in milling processes.

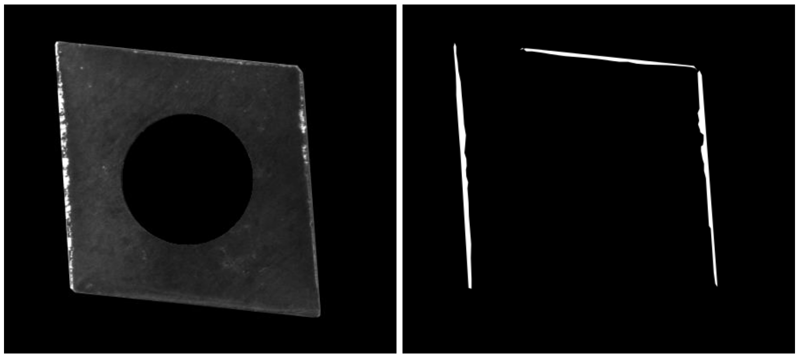
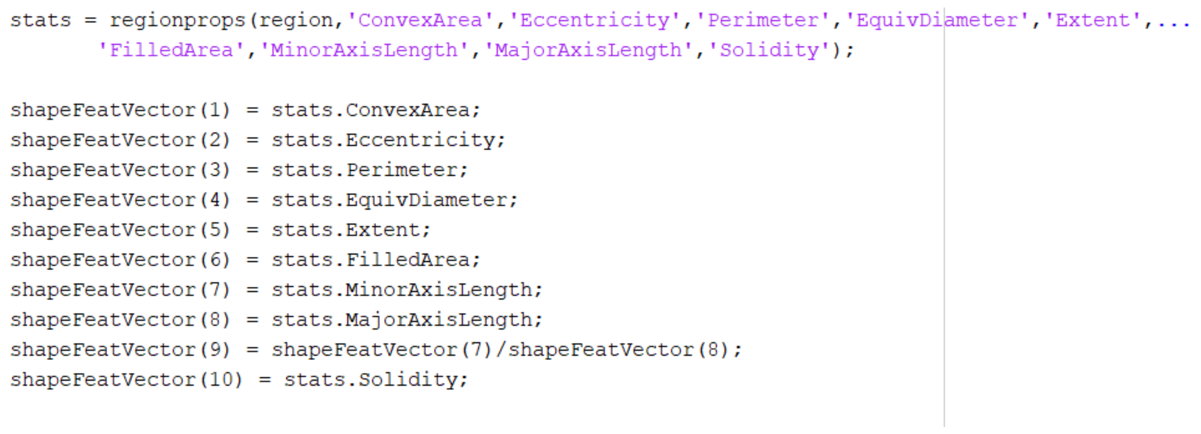


Figure 1: image of the cutting edges and its binary segmented image

For each binary image we will extract 10 features from a descriptor called *ShapeFeat*.

The extraction of these features is made by the following MATLAB function “***fGetShapeFeat.m”:***

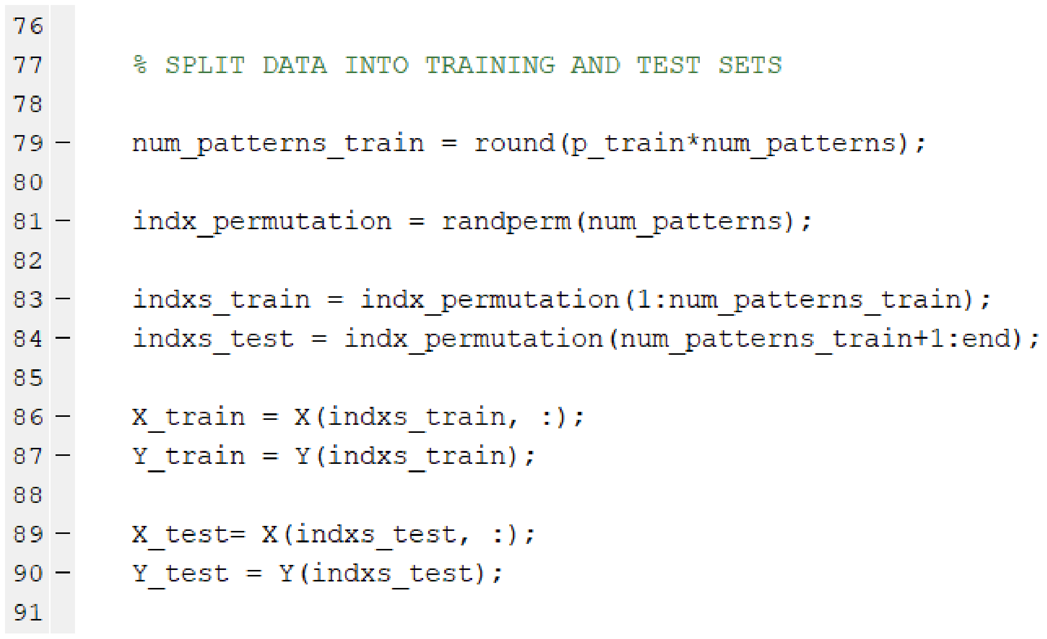


```
stats = regionprops(region,'ConvexArea','Eccentricity','Perimeter','EquivDiameter','Extent',...
       'FilledArea','MinorAxisLength','MajorAxisLength','Solidity');

shapeFeatVector(1) = stats.ConvexArea;
shapeFeatVector(2) = stats.Eccentricity;
shapeFeatVector(3) = stats.Perimeter;
shapeFeatVector(4) = stats.EquivDiameter;
shapeFeatVector(5) = stats.Extent;
shapeFeatVector(6) = stats.FilledArea;
shapeFeatVector(7) = stats.MinorAxisLength;
shapeFeatVector(8) = stats.MajorAxisLength;
shapeFeatVector(9) = shapeFeatVector(7)/shapeFeatVector(8);
shapeFeatVector(10) = stats.Solidity;
```

Figure 2: Script to extract the geometrical features

3 Division of the data between Train and Test sets

Now, we are going to split randomly our data, which is a matrix that the 10 features extracted for each image, into two disjoint sets. The first will be used to train our logistic classifier, and the second will be used to test the classifier and compute some metrics to evaluate the model (classifier).



```

% SPLIT DATA INTO TRAINING AND TEST SETS

num_patterns_train = round(p_train*num_patterns);

indx_permutation = randperm(num_patterns);

indxs_train = indx_permutation(1:num_patterns_train);
indxs_test = indx_permutation(num_patterns_train+1:end);

X_train = X(indxs_train, :);
Y_train = Y(indxs_train);

X_test= X(indxs_test, :);
Y_test = Y(indxs_test);

```

Figure 3: Script of splitting data into training set and test set

4 Classification:

4.1 Training of the classifier and classification of the test set:

Training:

The aim of the training phase of the logistic regression is to find the optimal parameter θ that are used in the hypothesis function to estimate the probability that a certain pattern $x^{(i)}$ belongs to the positive class. The hypothesis function used in logistic regression is sigmoid function:

$$h_{\boldsymbol{\theta}}(\mathbf{x}) = g(\boldsymbol{\theta}^T \mathbf{x}) = \frac{1}{1+e^{-\boldsymbol{\theta}^T \mathbf{x}}}$$

To find the optimal parameters θ we are going to minimize the cost function J(θ), which will yield the average error between the outputs of the function h for the training elements and their real classes.

$$J(\boldsymbol{\theta}) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \ln(h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)})) + (1-y^{(i)})\ln\left(1-h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)})\right)\right]$$

Now, in order to find the Minimum of J(θ), we are going to use the gradient descent algorithm, which is an iterative procedure in which each of the components of the vector are updated on each iteration:

$$\theta_j^{(k)} = \theta_j^{(k-1)} - \alpha \frac{1}{m}\sum_{i=1}^{m}(h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) - y^{(i)}) \cdot x_j^{(i)},$$

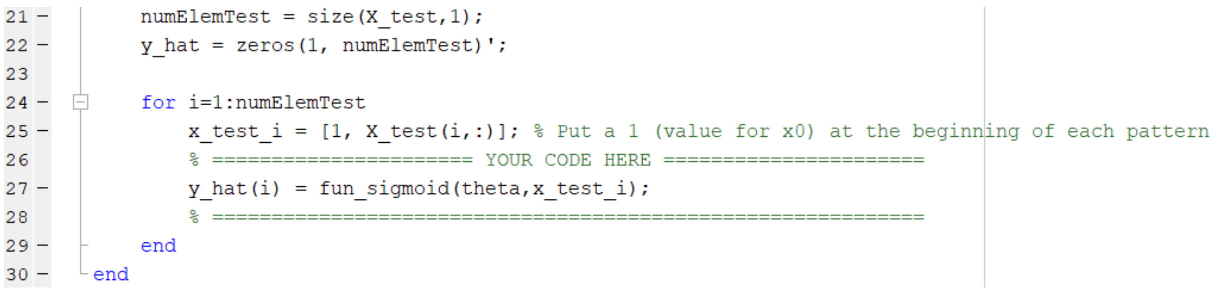
where the element $x_j^{(i)}$ is the jth feature of the feature vector that represents the ith object (i.e., the ith pattern). Be aware that the value of the element is always 1. In practice, it will be obtained adding the value 1 at the beginning of the feature vector.

The training of our classifier is made by the function ***"fTrain_Logistic_Reg.m"***, the function ***"fCalculateCostLogReg.m"*** computes the cost function and the function ***"fun_sigmoid.m"*** computes the hypothesis function.

Classification:

Once our classifier has trained, we are going to classify the elements that are not used in the training phase i.e., we will classify the elements of the test set. This means that we will compute for each element of test set the probability that it belongs to the positive class.

The classification is done with following code in the file ***"fClassify_LogisticReg.m"***:



```
    numElemTest = size(X_test,1);
    y_hat = zeros(1, numElemTest)';

    for i=1:numElemTest
        x_test_i = [1, X_test(i,:)]; % Put a 1 (value for x0) at the beginning of each pattern
        % ===================== YOUR CODE HERE ======================
        y_hat(i) = fun_sigmoid(theta,x_test_i);
        % ============================================================
    end
end
```

Figure 3: Script of classification phase

One thing we should ensure before starting classification is the convergence of our cost function J(θ). This convergence depends on the learning rate alpha in the gradient descent. Now, we will see how the value of this learning rate α affect the convergence of our cost function.

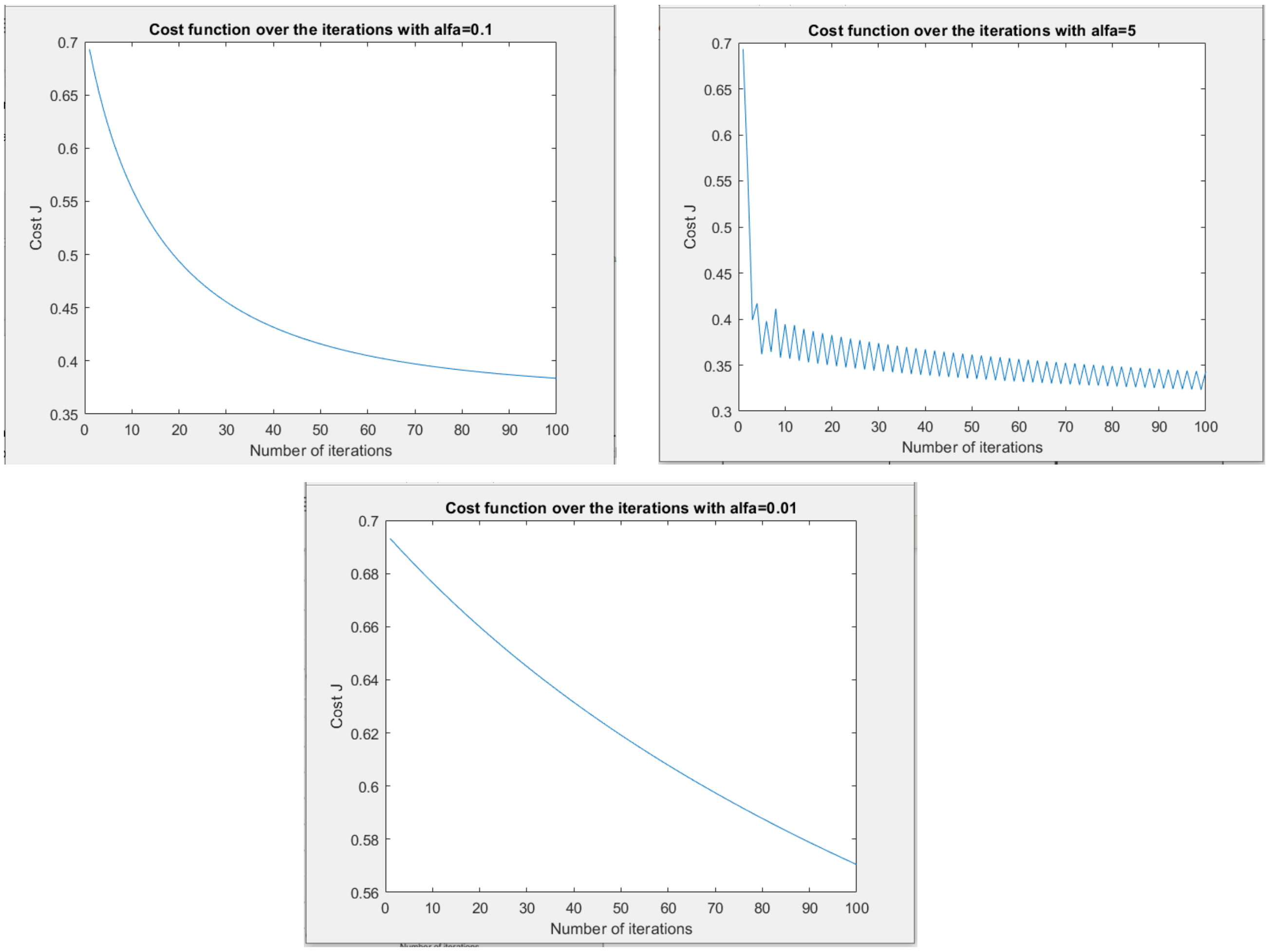
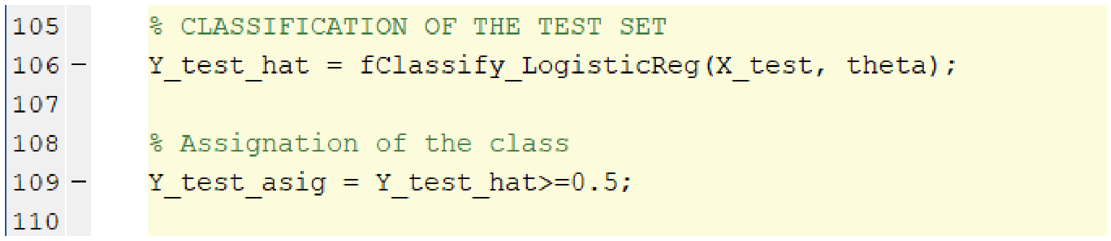


Figure 3: plot of the cost Function with different values of α

We see from these figures that when the value of the learning rate is great (for example α = 5) the cost function does not converge but when the alpha is small (α = 0.1, 0.01) the cost function converges but for exceedingly small values of alpha the convergence becomes slow, we should choose a value of alpha that ensure a fast convergence the cost function, in our case, 0.1 seems to be a good value for α. Also, it is a good idea to try to modify the number of iterations to see the effect of the learning rate.

We have computed for each element of test set the probability that it belongs to the positive class. Now we are going to assign for each element the class that correspond to it using a decision threshold equal to 0.5.



```
% CLASSIFICATION OF THE TEST SET
Y_test_hat = fClassify_LogisticReg(X_test, theta);

% Assignation of the class
Y_test_asig = Y_test_hat>=0.5;

```

Figure 4: assigning a class for each element of the test set

4.2 Assessment of the performance of the classifier:

Once we have estimated the class of each element in our test set, we are going to compute some metrics to measure the performance of our classifier. The metrics that will be used to evaluate our classifier are the ***"accuracy"*** and the ***"F-score".***

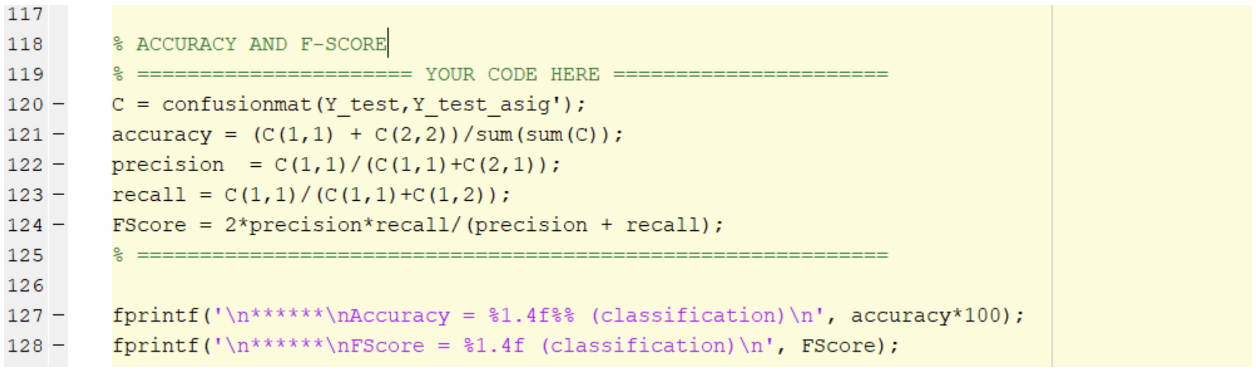
Accuracy: the percentage of the elements that have been well classified

F-score: the harmonic mean between the precision and the recall. Therefore, it gives an idea of the balance between both measurements

Precision: Ratio of the true positives and the total number of elements that the classifier has predicted as positives.

Recall: Ratio of the true positives and the elements that the number of elements whose real class is positive

All this metrics can be computed from the confusion matrix which contains the value of four categories: ***True Positive, False Negative, False Positive, True Negative***.



```
% ACCURACY AND F-SCORE
% ====================== YOUR CODE HERE ======================
C = confusionmat(Y_test,Y_test_asig');
accuracy = (C(1,1) + C(2,2))/sum(sum(C));
precision  = C(1,1)/(C(1,1)+C(2,1));
recall = C(1,1)/(C(1,1)+C(1,2));
FScore = 2*precision*recall/(precision + recall);
% ============================================================

fprintf('\n******\nAccuracy = %1.4f%% (classification)\n', accuracy*100);
fprintf('\n******\nFScore = %1.4f (classification)\n', FScore);
```

Figure 4: computation of the performance metrics

After Computing the confusion matrix, we have computed the accuracy and the F-score; we get the following results:

Accuracy = 80.2469%

FScore = 0.6190

We See that we get an accuracy that equal to 0.80 which is good, and we can assume that our classifier is efficient, but the accuracy can be a tricky metric especially when the test set is imbalanced. So, that is why we should compute the value of the F-score besides the accuracy to evaluate our classifier correctly. The value of the F-score that we have got (0.61) confirms that our data is imbalanced

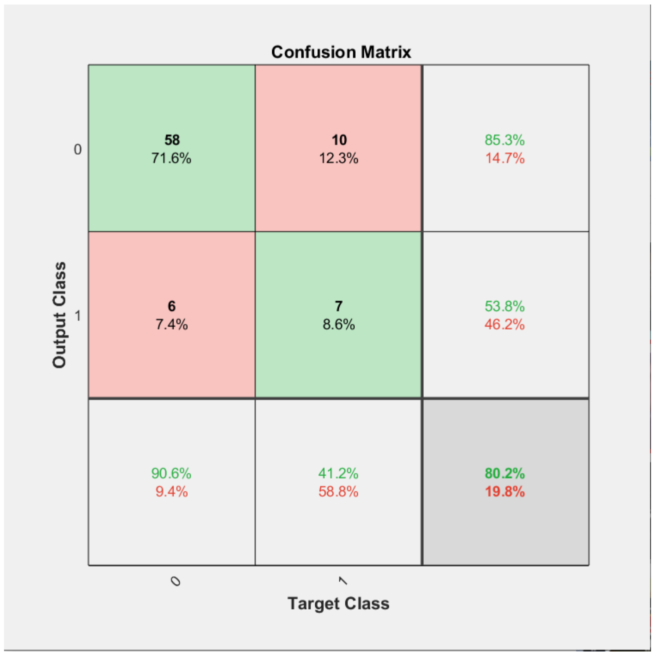
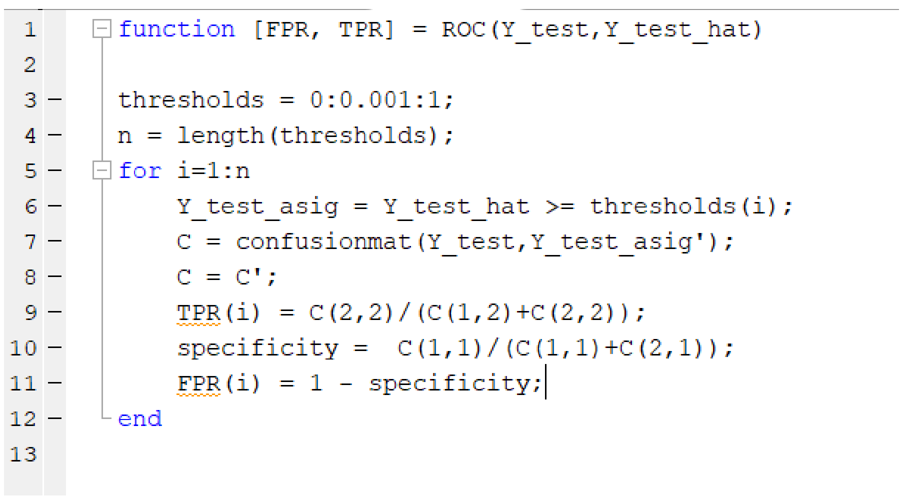


Figure 5: Confusion matrix

Note: You could get different results if you execute the code, and this because the split of the data is random. But you will get results close to the ones in this report.

5 ROC Analysis:

ROC curve is a performance measurement for classification problem at various thresholds settings. ***ROC*** is a probability curve; it shows the evolution of ***True Positive Rate (sensitivity or recall)*** depending on the ***False Positive Rate (1 – specificity).***



```
function [FPR, TPR] = ROC(Y_test,Y_test_hat)

thresholds = 0:0.001:1;
n = length(thresholds);
for i=1:n
    Y_test_asig = Y_test_hat >= thresholds(i);
    C = confusionmat(Y_test,Y_test_asig');
    C = C';
    TPR(i) = C(2,2)/(C(1,2)+C(2,2));
    specificity =  C(1,1)/(C(1,1)+C(2,1));
    FPR(i) = 1 - specificity;
end

```

Figure 6: code of the function that compute the ROC curve

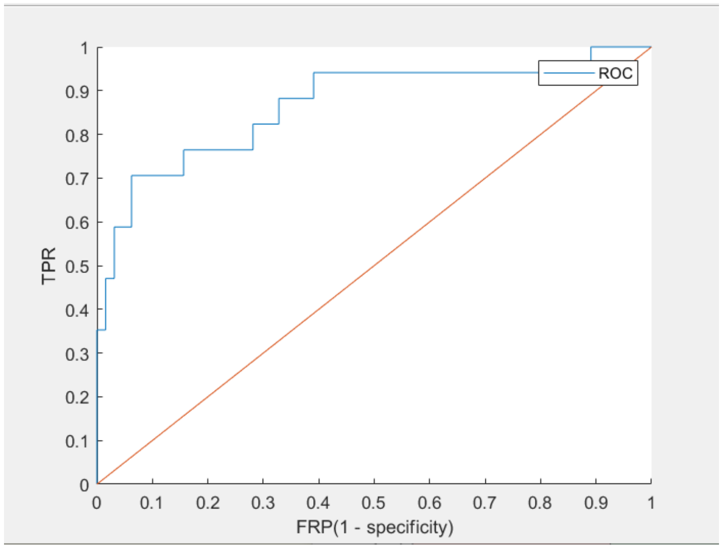
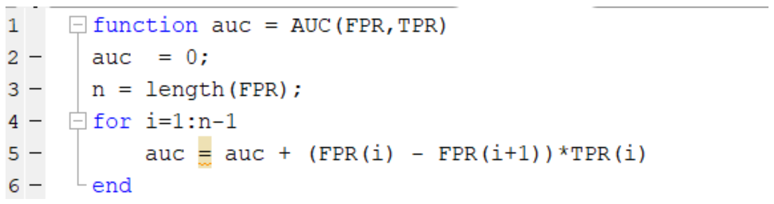


Figure 7: ROC curve of the classifier

After Computing the ***ROC Curve***, let us compute ***the Area Under Curve (AUC)***. ***AUC*** represents degree of separability. It tells how much model is capable of distinguishing between classes. Higher the ***AUC***, better the model is at predicting 0s as 0s and 1s as 1s.

The code of the function that compute the AUC.



```
function auc = AUC(FPR,TPR)
auc   = 0;
n = length(FPR);
for i=1:n-1
        auc = auc + (FPR(i) - FPR(i+1))*TPR(i)
end
```

Figure 8: code of the function that compute the AUC

We got an *AUC = 0.8500* which means that our classifier is quite efficient.

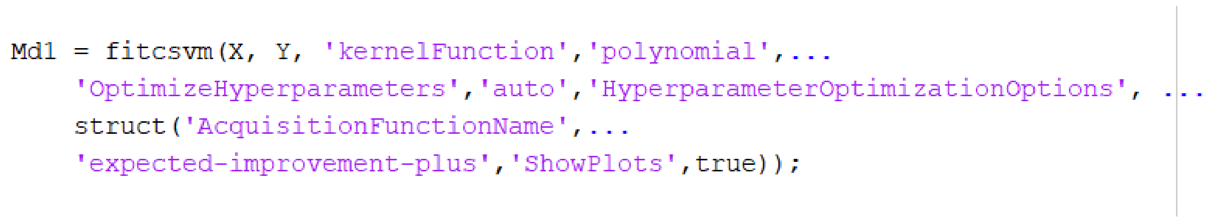
6 Other classifiers:

6.1 SVM:

The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space (N — the number of features) that distinctly classifies the data points.

To separate the two classes of data points, there are many possible hyperplanes that could be chosen. Our objective is to find a plane that has the maximum margin, i.e., the maximum distance between data points of both classes. Maximizing the margin distance provides some reinforcement so that future data points can be classified with more confidence.

The parameters that influence the choice of the optimal hyperplane for our SVM classifier are the kernel, the soft margin (Box Constraint) and the kernel scale. We are going to use a built-in function in MATLAB to find the optimal the Box Constraint and the Kernel Scale for a specific kernel function.

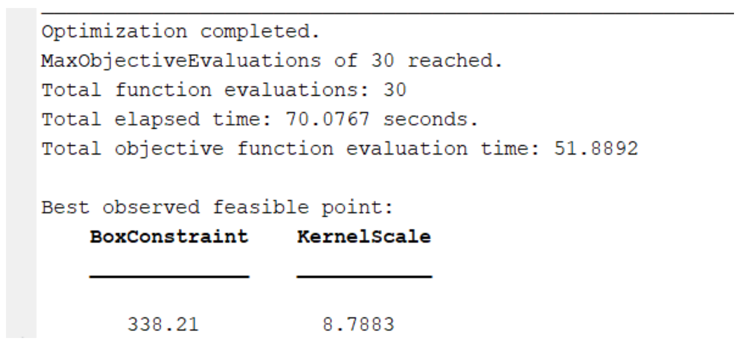


```
Md1 = fitcsvm(X, Y, 'kernelFunction','polynomial',...
    'OptimizeHyperparameters','auto','HyperparameterOptimizationOptions', ...
    struct('AcquisitionFunctionName',...
    'expected-improvement-plus','ShowPlots',true));
```

Figure 9: code to find the optimal parameter for the polynomial kernel

We have tried other kernel function such as the gaussian kernel and the linear kernel, but the polynomial kernel gave the best parameters.

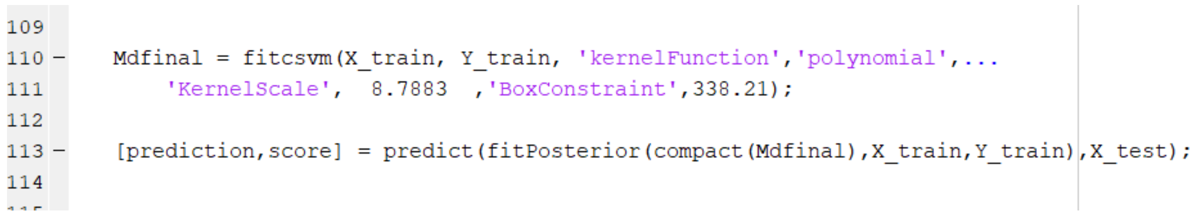
So, we got the following result of the hyperparameters optimization process:



```
Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
Total elapsed time: 70.0767 seconds.
Total objective function evaluation time: 51.8892

Best observed feasible point:
    BoxConstraint    KernelScale
    _____________    ___________

       338.21          8.7883
```

Figure 10: result of the hyperparameters optimization

Now, we are going to build our SVM classifier with the hyperparameters found previously and evaluate its performance:



```

    Mdfinal = fitcsvm(X_train, Y_train, 'kernelFunction','polynomial',...
        'KernelScale',  8.7883  ,'BoxConstraint',338.21);

    [prediction,score] = predict(fitPosterior(compact(Mdfinal),X_train,Y_train),X_test);

```

Figure 11: code to train the SVM classifier and classify the test

The measure of the performance of our SVM classifier gives the following results:

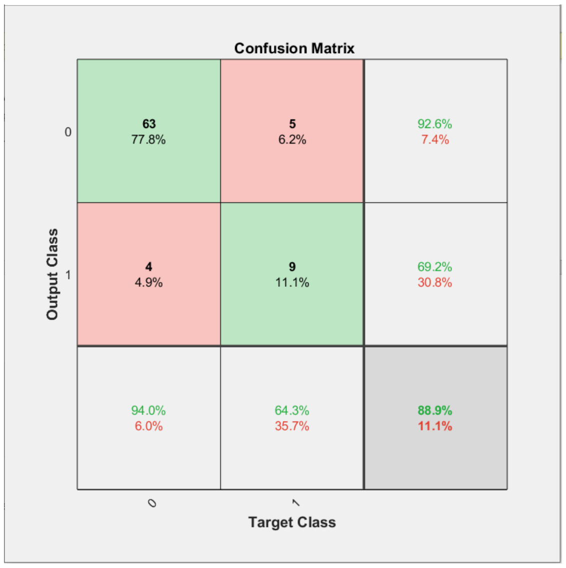


Figure 12: Confusion matrix

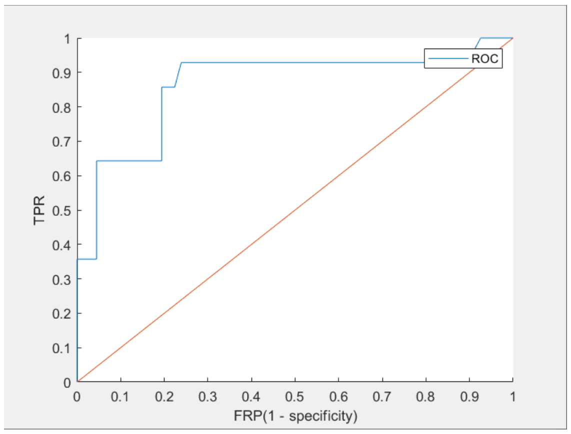


Figure 12: ROC curve

Accuracy = 88.8889%

FScore = 0.6667

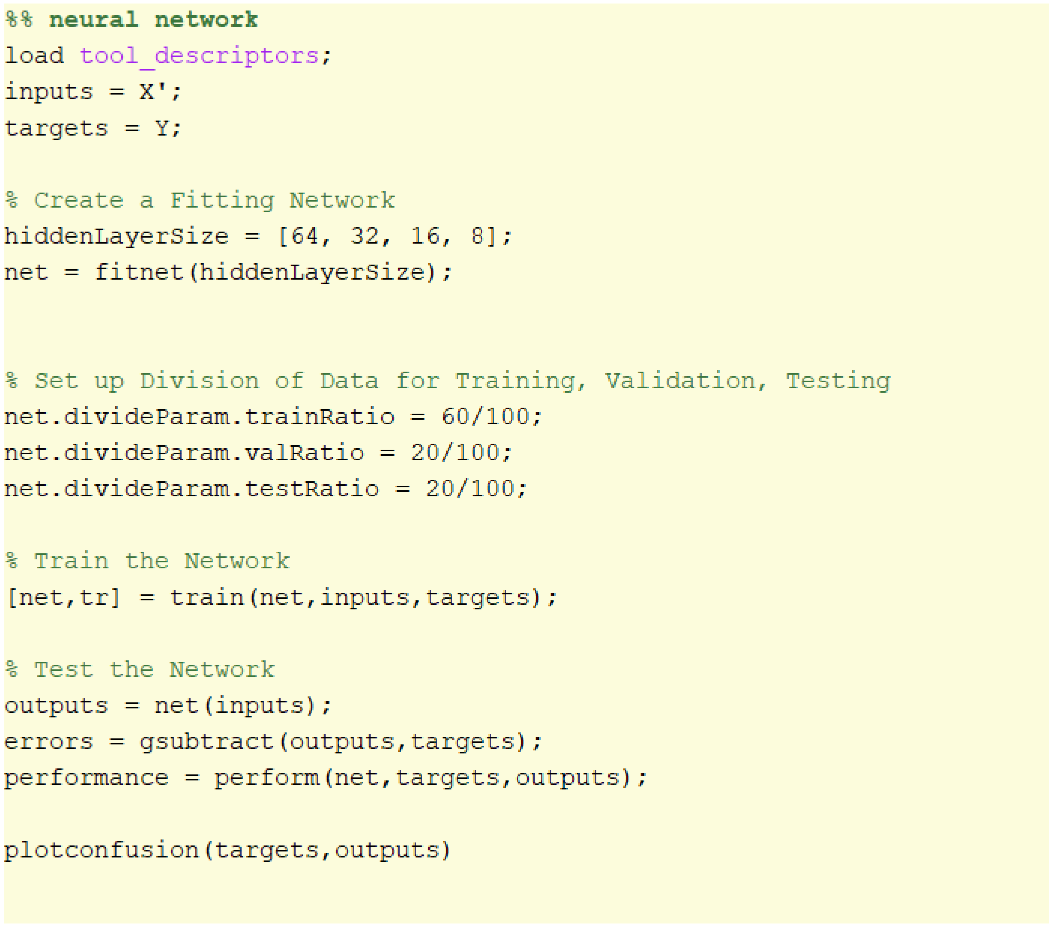
AUC = 0.8657

From these results, we can conclude that ***SVM*** classifier perform much better than the ***logistic regression*** classifier.

6.2 Neural Network:

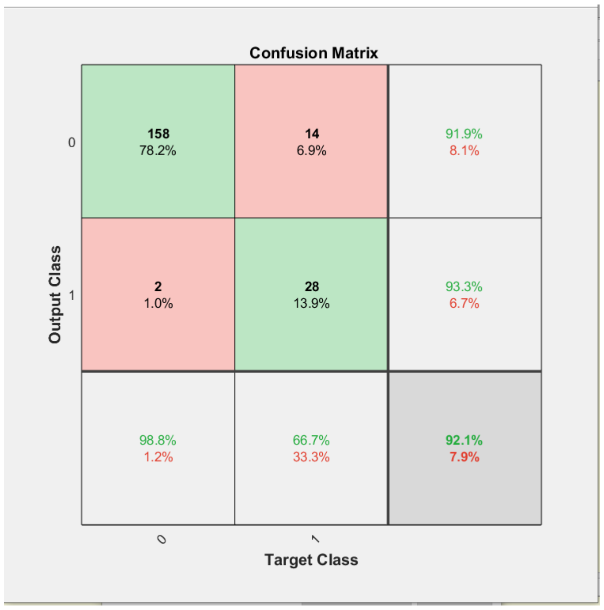
Now, we are going to use the neural network as classifier. Neural Network is a strong algorithm in machine learning, especially in binary classification. In MATLAB, there is a tool called ***"nntraintool"*** helps to build a neural network.

In a neural network, the choice of the number of hidden layers and their sizes plays is important and influences the performance of the neural network. In our case, to choose the number of hidden layers and their size has been made after many tries



```
%% neural network
load tool_descriptors;
inputs = X';
targets = Y;

% Create a Fitting Network
hiddenLayerSize = [64, 32, 16, 8];
net = fitnet(hiddenLayerSize);


% Set up Division of Data for Training, Validation, Testing
net.divideParam.trainRatio = 60/100;
net.divideParam.valRatio = 20/100;
net.divideParam.testRatio = 20/100;

% Train the Network
[net,tr] = train(net,inputs,targets);

% Test the Network
outputs = net(inputs);
errors = gsubtract(outputs,targets);
performance = perform(net,targets,outputs);

plotconfusion(targets,outputs)
```

Figure 13: Script to build a neural network



With neural network as classifier, we got an *accuracy = 92%* which is remarkably high, thus we can conclude that neural network is more accurate than other classifiers.

7 Conclusion:

Through this project we have seen many algorithms that are used in binary classification and the metrics that are used to evaluate the performance of any classifier such as ***the accuracy, F-score, confusion matrix*** and ***the ROC-curve.***

In our case, we have found that neural network is the best classifier for our data which gave the highest accuracy, but also the other algorithms (***logistic regression, SVM***) work very well with good accuracy. And this because our data is simple and not big.

Sometimes, certain algorithms cost a lot in term of the **CPU** usage and require a lot of time to build the model, so instead of using these algorithms to get high accuracy, we can use simple algorithms such as logistic regression and perform a **PCA** (Principal Component Analysis) which helps to reduce the dimensionality of the data set (in other words to reduce the number of features) and then get a high accuracy.