Project : Tool Wear Classification

ESSAYEGH Nour

December 2020

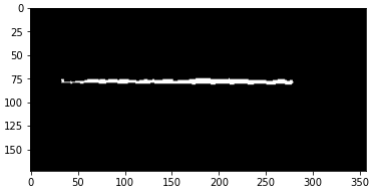
For this project, we are going to work on the classification of some binary images using machine learning and especially the method of logistic regression.

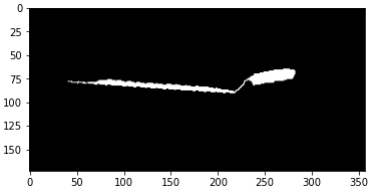
The data set we will be working with is a number of binary images that correspond to cutting edge tools.

The objective of this study is to classify those edges based on some features.

1 PART 1 : Image Description

Here are examples of images we will be describing.





We used the function regionprops to get all the features we need for our study.The function *fGetShape-Feat* compute the 10 features for a given region.The 10 features are the convex area, the eccentricity, the perimeter, the equivalent diameter, the extent, the filled area, the minor and major axis length, the ratio of the axis lengths and the solidity.

The features of each image are computed in an X vector. The Y vector represents the labels of each image. For a better interpretation, we binarize our labels so that class False corresponds to a low or medium wear level, whereas class True corresponds to a high wear level.

2 PART 2 : Classification

2.1 Training and testing sets :

To test the robustness of our model we have to split randomly our data into training and testing sets. We will use 70% of the data to train our model and the other 30% for testing for which we predict the results of the labeling using the logical regression model and we compare it with the labels we have. Therefore, evaluate the quality of the classifier. Since I'm working with python we can use the *train_test_split* but I decided to reproduce the Matlab script in the main program.

2.2 Construction of our classifier :

2.2.1 Construction and training

The objective of the training of the logical regression classifier is to estimate a parameter θ used to estimate if the binary image described by X_i belongs to the positive class. The model is based on the estimation of the parameter θ so that it will minimize a cost function J. The **hypothesis function** h used in logistic regression is the sigmoid function. The equation of this function is :

$$h(z) = \frac{1}{1 + e^{-z}} \tag{1}$$

```
def fun_sigmoid(theta, X):
    p=np.dot(theta,np.transpose(X))
    g=1/(1+np.exp(-1*p))
    return g
```

In our case z is a linear combination of θ and x.
In this model the cost function is given by the following equation :

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y_i * ln(h(x_i)) + (1 - y_i) * ln(1 - h(x_i))] \tag{2}$$

The optimisation is carried out by means of the **gradient descent** algorithm. As seen in the equation bellow :

$$\theta_j^k = \theta_j^{k-1} - \alpha * \frac{1}{m}\sum_{i=1}^{m}(h(x^i) - y^i) * x_j^i \tag{3}$$

We started by defining a function to determine the cost for each y_i and x_i called *fCalculate-CostLogReg* :

```
def fCalculateCostLogReg(y, y_hat) :

    cost_i=y*np.log(y_hat)+(1-y)*np.log(1-y_hat)
    return cost_i
```

We are using the 2 last functions into the loop inside the function $fTrain\_LogisticReg$ so that it will return the θ we will be using for prediction.

```
for num_iter in range(max_iter):
    # STEP 1. Calculate the value of the function h with the current theta
    gradient=np.zeros((m,n+1))
    x_i=np.zeros((m,n+1))
    x_i[:,0]=1
    for i in range(m):
        x_i[i,1:n+1] =X_train[i, :]
        h_train[0,i]=fun_sigmoid(theta, x_i[i])
        gradient[i,:]=(h_train[0,i]-Y_train[i])*x_i[i]

    # STEP 2. Update the theta values.
    theta=theta-alpha*(1/m)*sum(gradient)

    # STEP 3. Calculate the cost on this iteration and store it on vector J
    x_i=np.zeros((m,n+1))
    x_i[:,0]=1
    total_cost=0
    for i in range(m):
        x_i[i,1:n+1]  =X_train[i, :]
        h_train[0,i]=fun_sigmoid(theta,x_i[i])
        total_cost += fCalculateCostLogReg(Y_train[i], h_train[0,i])

    total_cost=(-1/m)*total_cost
    J[num_iter+1]=total_cost
```

2.2.2 Classification

The prediction is generated with the function $fClassify\_LogisticReg$ that builds the labels corresponding to the testing set (X_test) using the θ we extracted from $fTrain\_LogisticReg$.

In order to quantify the convergence of our classifier, we are able to see the evolution of the cost function. This function depends on two variables, the learning rate α and the number of iterations. For α in $]0, 3]$, we see the curve slowly converging to the value of its minimum and if we take α in $[2, 3]$ we can observe that the convergence is much quicker so it needs fewer iterations to stabilize around 0.33.

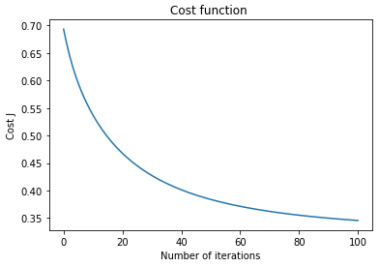


Figure 1: alpha=0.1

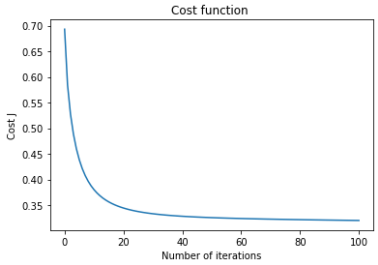


Figure 2: alpha=0.5

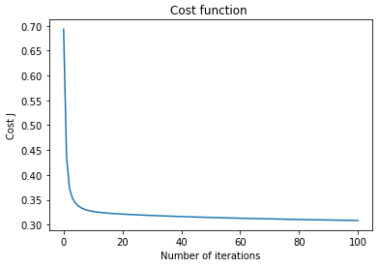


Figure 3: alpha=2

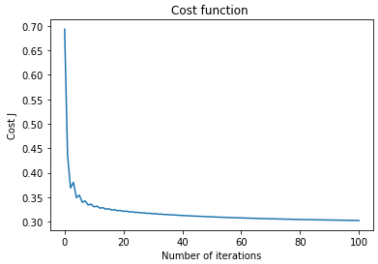


Figure 4: alpha=4

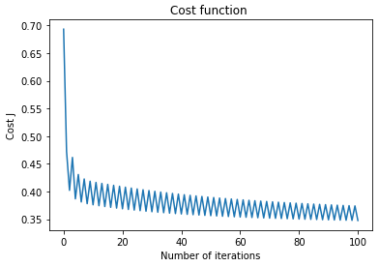


Figure 5: alpha=5

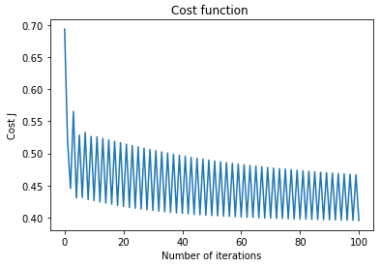


Figure 6: alpha=6

Exceeding $\alpha = 4$ we start to notice a few oscillations and for α exceeding 5 the convergence is impossible in less than 100 or even 1000 iterations and that is due to the gradient. For an important value of α the gradient does not converge to the minimum of the function and that leads to the important oscillations.

We notice that in this last case the accuracy is less important than the cases with the convergence and that's understandable from the structure of the algorithm.

3 PART 3 : Quality of the classifier

3.1 Confusion matrix and accuracy :

Since we have the predictions for our X_test we can compare the results with the real labels stored in Y_test to test if our classification model is efficient. A good tool to visualize the errors in the classification is the confusion matrix.

```
M=confusion_matrix(Y_test, Y_test_pred);
print(M)

# ACCURACY AND F-SCORE

accuracy=np.trace(M)/sum(sum(M))
Precision=M[0,0]/(M[0,0]+M[1,0])
Recall=M[0,0]/(M[0,0]+M[0,1])
FScore=2*((Precision*Recall)/(Precision+Recall))

print('the accuracy :', accuracy*100);
print('the Fscore :', FScore);
```

The result of one random splitting between the training and testing sets is :

```
[[59  7]
 [ 7  7]]
the accuracy : 82.5
the Fscore : 0.8939393939393939
```

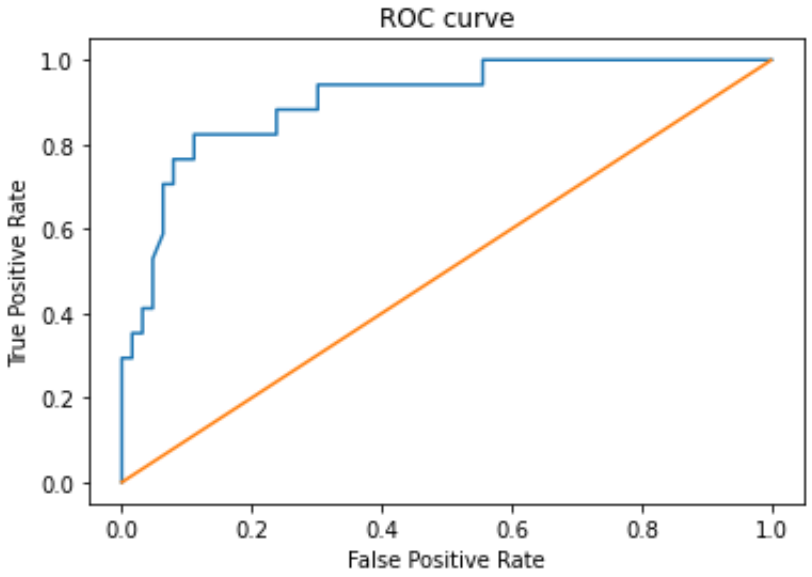
The confusion matrix is built in a way where we can see how many observations are falsely classified. We can see here that we have 14 miss classified objects 7 false-positives and 7 false-negatives.

The accuracy is calculated based on the confusion matrix and it's given in percentage.Looking at the value of the accuracy (82%), we can say that our model is efficient, but we can't fully trust the accuracy rate because it can be a very false criterion depending on the data.

3.2 The ROC Curve :

To be sure of our classifier we decided the use the ROC curve and the Area under the curve. The ROC curve is a curve that shows the evolution of the sensitivity comparing it to a threshold t=0.5. the area under this curve is very significant in terms of the efficiency of the classifier. For this we computed the function in our function file and plotted the result.

```
def ROC(Y_test,Y_test_hat):
    '''
    L=np.arange(0,1,0.001)
    T=[]
    F=[]
    for j in L:
        Ypred=Y_test_hat>=j
    #matrice de confusion
        M=np.zeros((2,2))
        for i in range (len(Y_test)):
            if Y_test[i]==True and Ypred[i]==True:
                M[0,0]+=1
            elif Y_test[i]==False and Ypred[i]==True:
                M[1,0]+=1
            elif Y_test[i]==True and Ypred[i]==False:
                M[0,1]+=1
            elif Y_test[i]==False and Ypred[i]==False:
                M[1,1]+=1

        tpr=M[0,0]/(M[0,0]+M[0,1])
        fpr=M[1,0]/(M[1,0]+M[1,1])
        T.append(tpr)
        F.append(fpr)
    return T,F
```



We also used the rectangular integral method to estimate the value of the Area Under the Curve (AUC) since an excellent model has AUC near to the 1 which means it has good measure of separability. A poor model has AUC near to the 0 which means it has worst measure of separability. In fact it means it is reciprocating the result. It is predicting 0s as 1s and 1s as 0s. And when AUC is 0.5, it means model has no class separation capacity whatsoever.

```
#the ROC curve and the area under the curve

T,F=ROC(Y_test,Y_test_hat)
L=np.arange(0,1,0.001)
plt.plot(F,T)
plt.plot(L,L)
plt.show()

#now we can the area under the curve using the rectangle rule
m=len(L)
AUC=0
for i in range(m-1):
    AUC+=(F[i]-F[i+1])*T[i]
print("The AUC is",AUC)
```

```
The AUC is 0.9202508960573477
```

For our classifier the value of the AUC is near 1. We can say that our classifier is strong and very efficient for the type of data we worked with.

4 PART 4 : Other classification algorithms

4.1 SVM

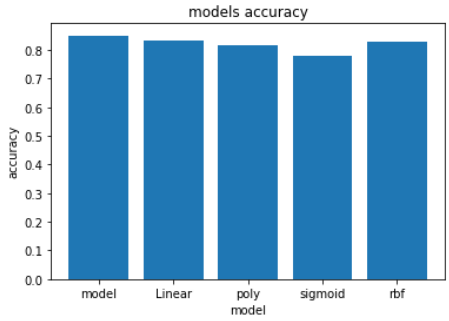
A support vector machine (SVM) is a machine learning algorithm that analyzes data for classification and regression analysis. It's a supervised learning method that looks at the data and sorts it into one of two categories. If the two categories cannot be separated linearly with a wide margin between the categories we use a nonlinear kernel as the Gaussian or the exponential. We can also add a soft margin parameter to make the classifier fit the data in a more precise way. There are multiple variations to the SVM method depending on the kernel used in each one. To compare the multiple kernels we trained and predicted 100 different training and testing sets with the 4 kernels. We visualize with a bar plot the mean accuracy and the AUC of each SVM model and the Logical regression, we built in part 1. We can see that our model has higher accuracy and AUC.

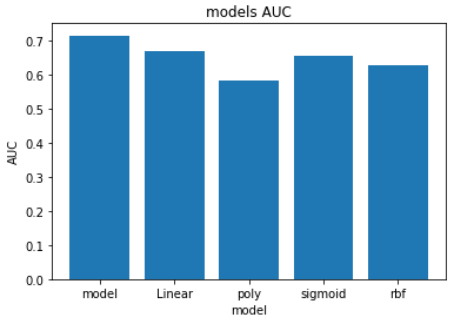
```
#linear svm
linear=svm.SVC(kernel='linear')
linear.fit(X_train,Y_train)
Y_pred1=linear.predict(X_test)

#polynomial svm
poly=svm.SVC(kernel='poly')
poly.fit(X_train,Y_train)
Y_pred2=poly.predict(X_test)

#sigmaid svm
sigmoid=svm.SVC(kernel='sigmoid')
sigmoid.fit(X_train,Y_train)
Y_pred3=sigmoid.predict(X_test)

#rbf svm
rbf=svm.SVC(kernel='rbf')
rbf.fit(X_train,Y_train)
Y_pred4=rbf.predict(X_test)
```

Figure 7: SVM without soft margin

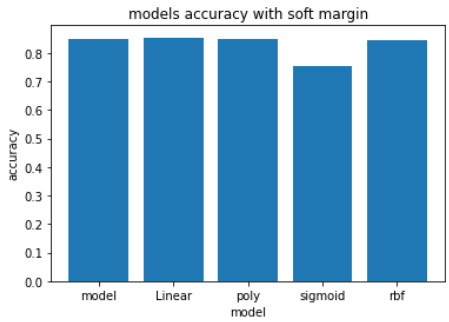
```
#linear svm
linear=svm.SVC(kernel='linear',C=10)
linear.fit(X_train,Y_train)
Y_pred1=linear.predict(X_test)

#polynomial svm
poly=svm.SVC(kernel='poly',C=10)
poly.fit(X_train,Y_train)
Y_pred2=poly.predict(X_test)

#sigmaid svm
sigmoid=svm.SVC(kernel='sigmoid',C=10)
sigmoid.fit(X_train,Y_train)
Y_pred3=sigmoid.predict(X_test)

#rbf svm
rbf=svm.SVC(kernel='rbf',C=10)
rbf.fit(X_train,Y_train)
Y_pred4=rbf.predict(X_test)
```

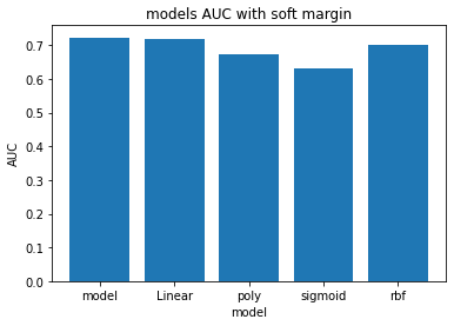
Figure 8: SVM with soft margin c=10





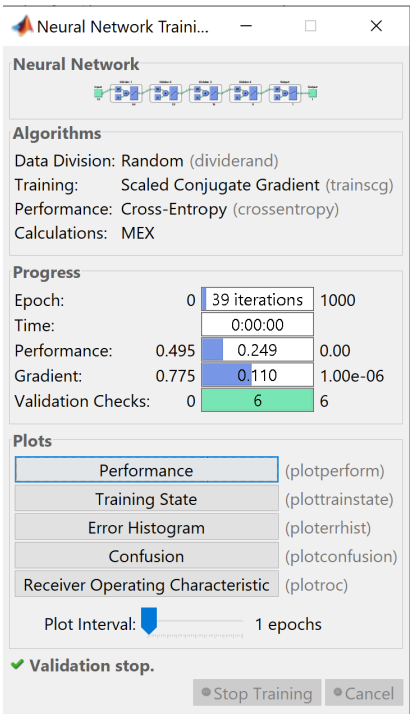
We repeated the same operation with the soft margin and with that, we can say that the linear kernel and the rbf kernel are very close to the efficiency of our model. They might even be a slightly more efficient.



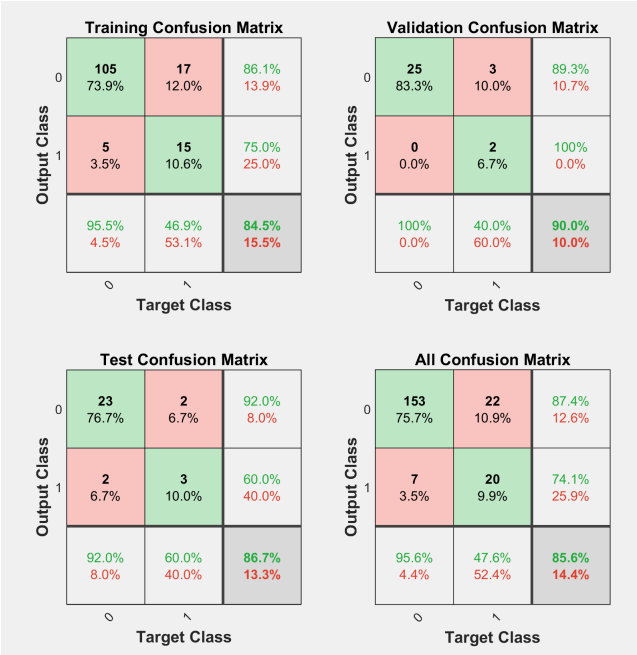


4.2 neural network

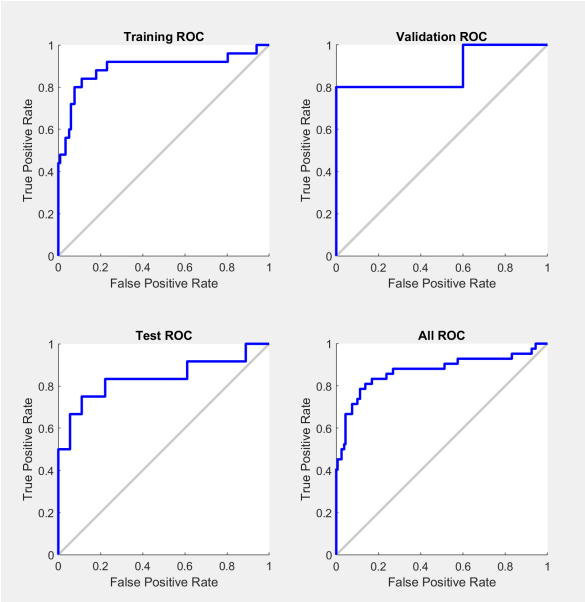
The classical shallow neural networks are one of the strongest tools in binary classification. I tried to build a neural network in python but I had some problems with the importation of the library tensorflow due to an incompatibility with python 3.8 and higher versions. Therefore I decided to work with MATLAB for this part. I defined the function fGetshapeFeat and filled the gaps to construct my X ans Y vectors since it was all I needed for the modelisation. The neural network summary is given as bellow :



To have a clear visualisation of the performance of our model we can have a close look to the confusion matrix for the training, validation and testing sets.



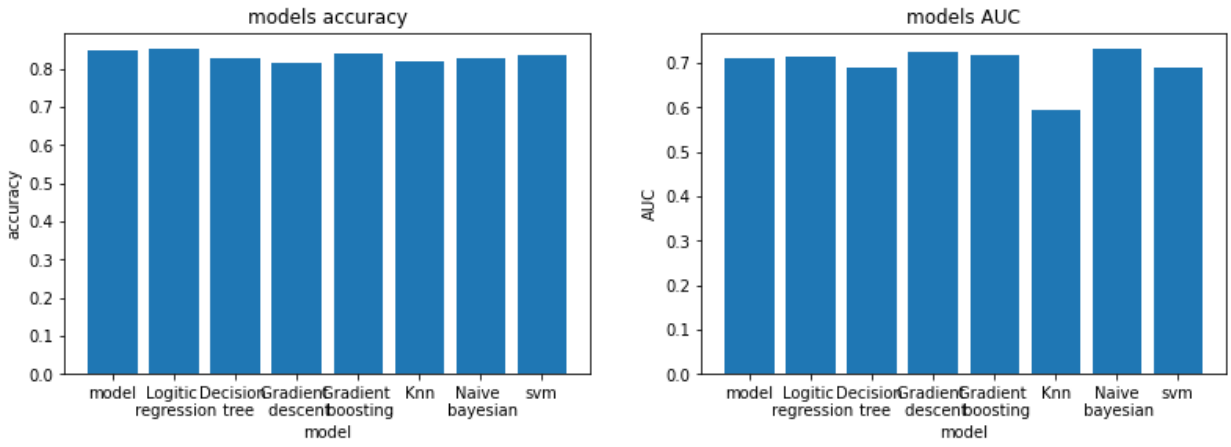
We can see that the total accuracy is 90% for it validation set and 85,6% globally. It's a slightly higher value comparing to our model. We can also visualise the efficiency of this model from it ROC curves.



4.3 Comparison with other classifier

We can compare our classifier with some other popular classifiers, we saw in the major Data science.For that, we created another file for the comparison named "other classifier.py".In this file, we imported the multiple libraries that correspond to the different algorithms. We chose to work with the generic function for Logical regression in python and it should give the same results as the method we coded, we also chose to work with decision tree, stochastic gradient descent, k nearest neighbors, Naive Bayesian and SVM.

To evaluate all the different algorithms we used the cross-validation method where we run the classification multiple times for $n = 100$ different samples and we compute the mean of the accuracy and the area under the curve for all the classifiers.We plotted the accuracy and the AUC to have a general view of the performance of each classifier.



From those two bar plot, we can see that the classification of the data with our method is very efficient since it has one of the highest accuracy and area under the curve in comparison to all the other models we can also see that the function of logical regression has nearly the same results as our model the slight difference might due to the α coefficient.

5 Conclusion

In comparison with other very popular and most efficient models, we can say that logistic regression is an efficient way to classify our sets of data regarding the geometrical and topological features we extracted from our binary images. We can imagine a way to upgrade the efficiency of our program with a grid search to fix the parameter alpha. We can also think of more feature descriptors that would make the gap between the two classes of our classifier more relevant.