Birla Institute of Technology and Science-Pilani, Hyderabad Campus

Second Semester 2019-2020

Machine Learning (BITS F464)

Assignment 2

Task - 1

Logistic Regression

Task - 2

Neural Networks

Under the guidance of

Prof. N L Bhanumurthy

Tenneti Hari Kiran	2017A2PS1462H
Kasuba Badri Vishal	2017A7PS0270H

Task 1 : Logistic Regression :

Aim :

To Detect Forged Bank Notes using BInary Logistic Regression Classification Model

Concept :

Logistic Regression is used to model the probability of a certain class or event that is to predict a given testing point with its given features belonging to class 1 or 0 or multiple classes. We use a sigmoid function to model a binary dependent variable as sigmoid function gives output ranging from 0 to 1 for any possible input

Given dataset (X_1, t_1) , (X_2, t_2) , (X_3, t_3) , (X_4, t_4) , ……. (X_N, t_N) N datapoints with each X_i having m training features $(X1_i, X2_i, X3_i, X4_i, ...Xm_i)$. We wish to predict whether for a given testing point X_k , whether that point belongs to $t_k = 0$ negative class or $t_k = 1$ positive class.
Since the resultant output is binary , we use sigmoid function of which the value we get if output>0.5 , then point belongs to +ve class else in -ve class

Sigmoid Function : $\delta(z) = \frac{1}{1+e^{-z}}$

Now , we wish to find vector, $W = (W_1, W_2, W_3, ...W_m)$ such that $W^T X$ maximizes the probability of an element belonging to one particular class

Let $Z = W^T X$ and using sigmoid function , $\delta(z) = \frac{1}{1+e^{-(W^T X)}}$ now $0 \le \delta(z) \le 1$

$$P(y_{test} == 1 || W, X) = \frac{P(W,X || y_{train} == 1) \times P(y_{train})}{P(W,X)}$$

$$P(y_{test} == 0 || W, X) = \frac{P(W,X || y_{train} == 0) \times P(y_{train})}{P(W,X)}$$

$P(W, X)$ and $P(y_{train})$ constant and are not bothered

$P(W, X || y_{train} == 1)$, we have to take that W which maximize this probability which is equal to
$\prod_{i=1}^{m} P(W, X_i || y_i == 1)$
$\prod_{i=1}^{m} P(W, X_i || y_i == 1) = \prod_{i=1}^{m} y_i^{z_i} \times (1 - y_i)^{1-z_i}$ where $z_i = \frac{1}{1+e^{-(W^T X_i)}}$

To maximize $\sum_{i=1}^{m} y_i \times log\,(z_i) + (1 - y_i) \times log\,(1 - z_i) = Q$

We minimize $-Q = Loss \;=\; E\,(W)$

By differentiating and using $\delta^{|}\,(z)$ **=** $\delta\,(z) \times (1 - \delta\,(z))$

We calculate loss and minimize to get,

$\frac{dE}{dW_i} = \sum_{n=1}^{N} (y_n - z_n) \times x_i$ and using gradient descent with α as learning rate and β as stopping criteria we optimize W values as

$W = W - \alpha \times \frac{dE}{dW_i}$

To get the best possible W that maximizes the probability.

Now , we test the data by $0 \le \delta\left(W^T X_{test}\right) \le 0.5$ or $0.5 \le \delta\left(W^T X_{test}\right) \le 1$ to belong to positive or negative class respectively

Design Decisions :

We have not used any scikit library or advanced library for implementation, used numpy,pandas and matplotlib in python for model construction

The Given Dataset consists of 4 features let’s say X = (x1,x2,x3,x4) and with output value ‘y’.

Given Dataset of $(x1_1, x2_1, x3_1, x4_1, y_1)$, $(x1_2, x2_2, x3_2, x4_2, y_2)$, , $(x1_3, x2_3, x3_3, x4_3, y_3)$, ……. $(x1_N, x2_N, x3_N, x4_N, y_N)$

N data points where N=1372

We split the dataset for training and testing in 80:20 ratio and preprocess the data using many scaling techniques like min-max normalization, mean normalization, standardization techniques to make mean to be zero.

We initialized weights using different distributions some of them include standard_normal, uniform distribution, beta, normal and also random allocation.

Using Gradient Descent method of minimizing loss , we found the best optimal weights vector, Also Implemented L1 regularization which is adding additional $\lambda \sum_{i=1}^{m} |W|$ so that overfitting does not take place and similar reason different approach of L2 regularization $\lambda \sum_{i=1}^{m} |W|^2$, used both L1 and L2 regularization.

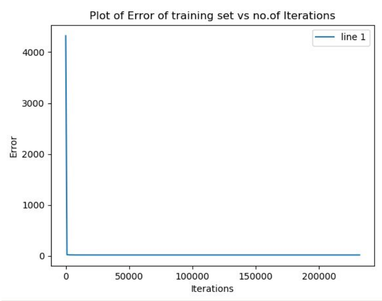
We also found out the features which are important in determining the class in which points belong to, by comparing the features of which the magnitude is higher.

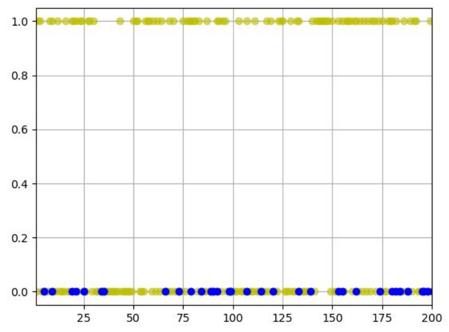
After Calculating optimal W at which point loss is not decreasing, reached global minimum , we used the testing data to test the implementation and compare it with the test data outputs to get true_positives,true_negatives,false_positives,false_negatives

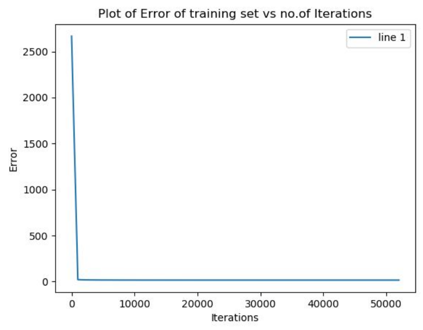
$$Precision = \frac{true-positives}{true-positives+true-negatives} \qquad Recall = \frac{true-positives}{true-positives+false-positives}$$

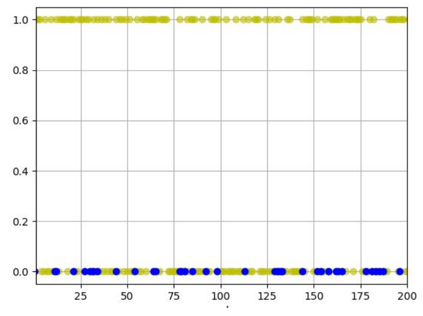
$$F-Score = 2 \times \frac{Precision \times Recall}{Precision+Recall}$$

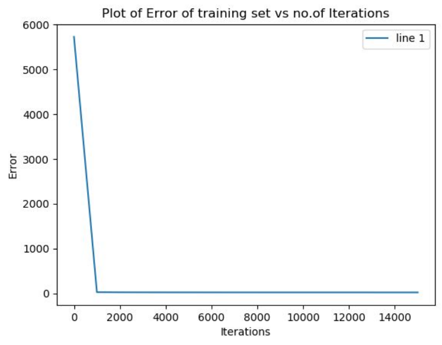
Results :

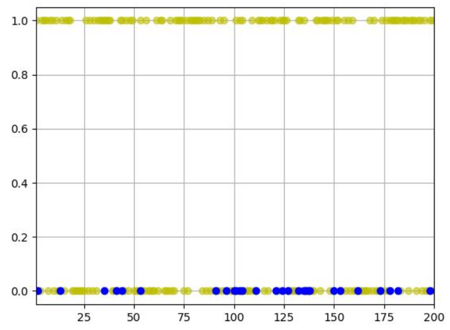


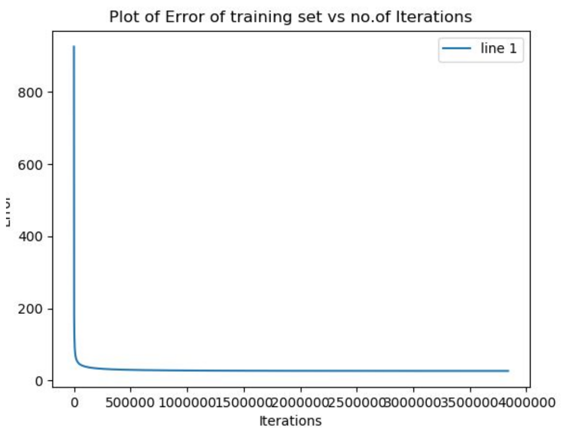


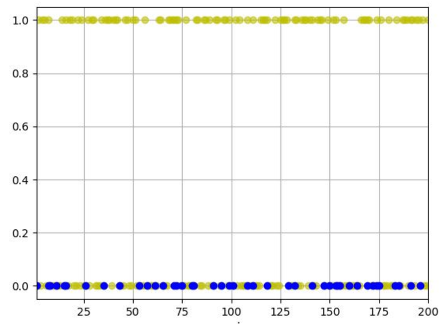


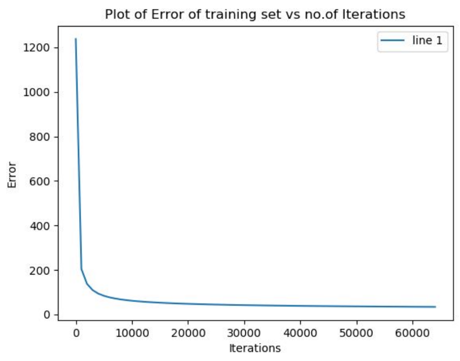


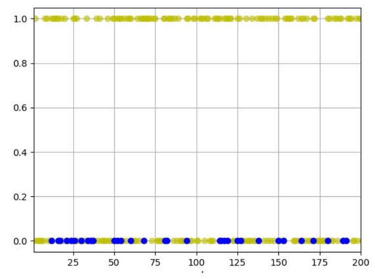


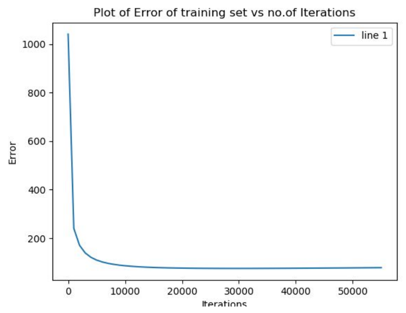


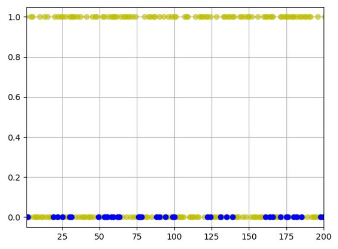


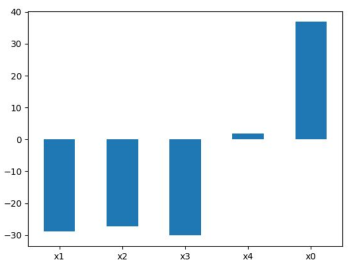


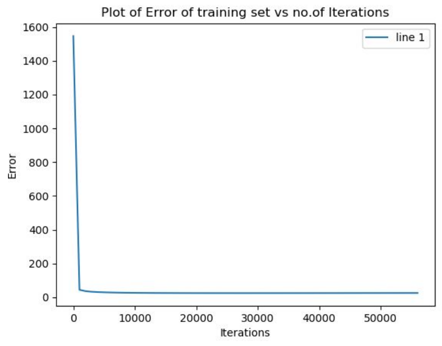


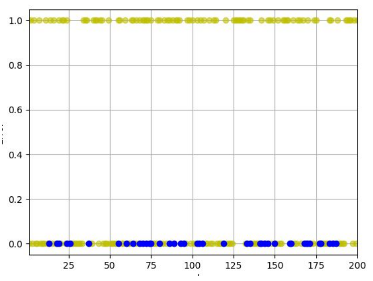


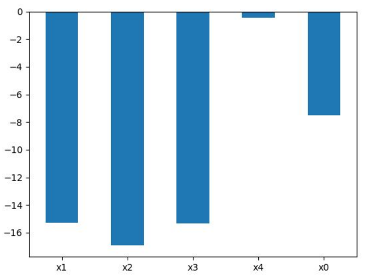




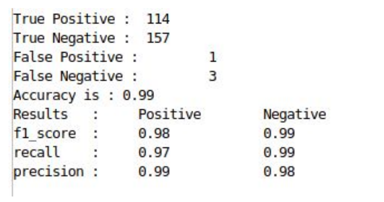


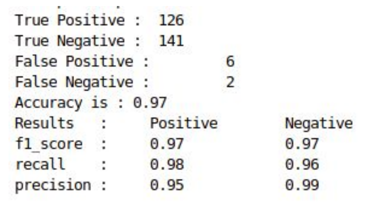


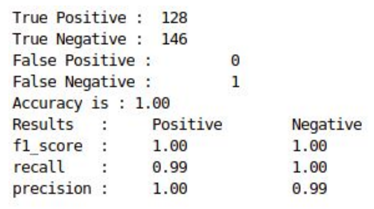


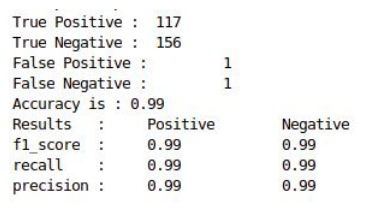


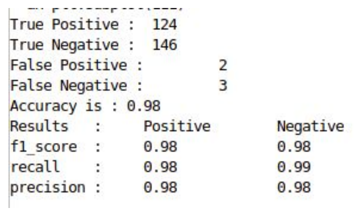
Variables	Losss	Accuracy	F-Score

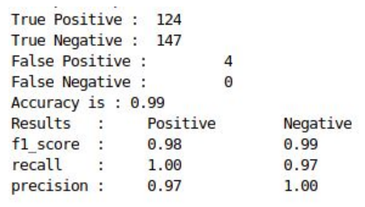
Scale - None
Reg - withoutReg
Dist - stdNormal
Alpha - 1e-3
	Loss = 17.32
	Iterations = 200000+
		Accuracy : 99%
		Time taken : 332 sec
			
			True Positive : 114
			True Negative : 157
			False Positive : 1
			False Negative : 3
			Accuracy is : 0.99
			Results : Positive Negative
			f1_score : 0.98 0.99
			recall : 0.97 0.99
			precision : 0.99 0.98

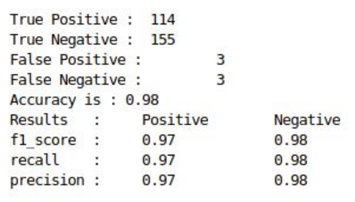
Scale - None
Reg - L1
Dist - stdNormal
Alpha - 1e-3
	Loss = 16.03
	Iterations = 50000+
		Accuracy : 97%
		Time Taken : 117 sec
			
			True Positive : 126
			True Negative : 141
			False Positive : 6
			False Negative : 2
			Accuracy is : 0.97
			Results : Positive Negative
			f1_score : 0.97 0.97
			recall : 0.98 0.96
			precision : 0.95 0.99

Scale - None
Reg - L2
Dist - stdNormal
Alpha - 1e-3
	Loss = 23.8
	Iterations = 14000+
		Accuracy = 100%
		Time Taken : 45 sec
			
			True Positive : 128
			True Negative : 146
			False Positive : 0
			False Negative : 1
			Accuracy is : 1.00
			Results : Positive Negative
			f1_score : 1.00 1.00
			recall : 0.99 1.00
			precision : 1.00 0.99

Scale - min-max
Reg - L1
Dist - uniform
Alpha - 1e-3
	Loss = 26.7
	Iterations = 4000000+
		Accuracy = 99%
		Time Taken = 7920 sec
			
			True Positive : 117
			True Negative : 156
			False Positive : 1
			False Negative : 1
			Accuracy is : 0.99
			Results : Positive Negative
			f1_score : 0.99 0.99
			recall : 0.99 0.99
			precision : 0.99 0.99

Scale - min-max
Reg - wR
Dist - stdNormal
Alpha - 1e-3
	Loss = 16.8
	Iterations = 50000+
		Accuracy = 98%
		Time Taken = 67 sec
			
			True Positive : 124
			True Negative : 146
			False Positive : 2
			False Negative : 3
			Accuracy is : 0.98
			Results : Positive Negative
			f1_score : 0.98 0.98
			recall : 0.98 0.99
			precision : 0.98 0.98

Scale - min-max
Reg - L2
Dist - randn
Alpha - 1e-2
	Loss = 23
	Iterations = 10000+
		Accuracy = 99%
		Time Taken = 83 sec
			
			True Positive : 124
			True Negative : 147
			False Positive : 4
			False Negative : 0
			Accuracy is : 0.99
			Results : Positive Negative
			f1_score : 0.98 0.99
			recall : 1.00 0.97
			precision : 0.97 1.00

Scale - std
Reg - L2
Dist - normal
Alpha - 1e-3
	Loss = 25.7
	Iterations = 10000+
		Accuracy = 98%
		Time Taken = 73 sec
			
			True Positive : 114
			True Negative : 155
			False Positive : 3
			False Negative : 3
			Accuracy is : 0.98
			Results : Positive Negative
			f1_score : 0.97 0.98
			recall : 0.97 0.98
			precision : 0.97 0.98

Conclusion:

Logistic regression is easier to implement, interpret and very efficient to train. It is the appropriate regression analysis to conduct when the dependent variable is binary. Logistic regression is used to describe data and to explain the relationship between one dependent binary variable. From the results, we can conclude that Regularization is an important part for regression analysis, with it we can get accurate results as testing error is decreased and loss is reaching global optimum in less number of iterations. The F-score on average is 0.98 and the accuracy is 98 to 99.

Task 2 : Neural Networks :

Aim:

The aim is to predict whether a house's price will be above or below the market's median price (binary classification).

Background Reading:

The neurons within the network interact with the neurons in the next layer, with every output acting as an input for a future function. Every function, including the initial neuron receives a numeric input, and produces a numeric output, based on an internalized function, which includes the addition of a bias term, which is unique for every neuron. That output is then converted to the numeric input for the function in the next layer, by being multiplied with an appropriate weight. This continues until one final output for the network is produced.

Procedure:

In neural networks, First normalization of the data has been done. Further, we used stochastic gradient descent to arrive at the local minima. The activation functions used for the hidden layer are tanh and sigmoid. For the output layer we have used sigmoid function for the classification problem. The error function generally used for classification using neural networks is *Binary cross Entropy function.*

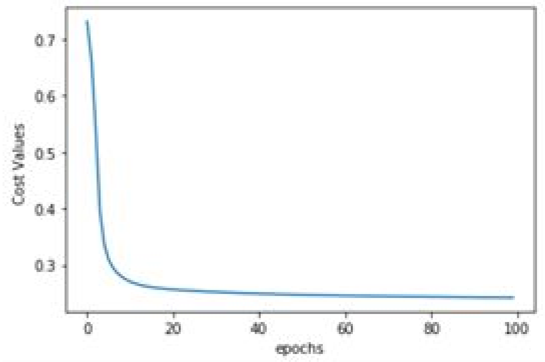
$$L(y,\hat{y}) = -\frac{1}{N}\sum_{i=0}^{N}(y * log(\hat{y}_i) + (1-y) * log(1-\hat{y}_i))$$

Observations & Results:

i) For 100 epochs and learning rate=0.05 and using tanh as the activation function for hidden layer,

a) For M=15 hidden nodes:
accuracy_score 88.6986301369863
F-score 0.896551724137931

b) For M=8 hidden nodes:
accuracy_score 89.04109589041096
F-score 0.9



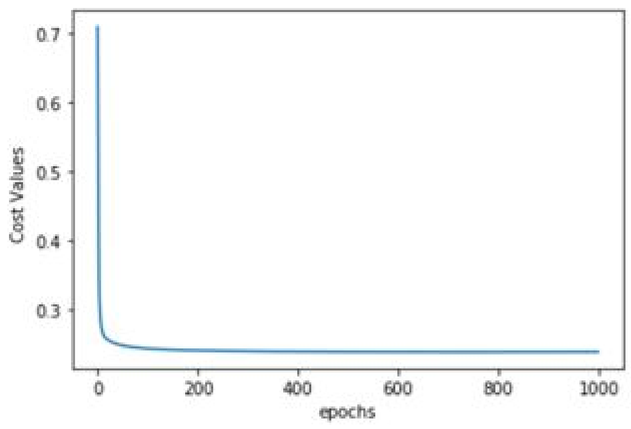
Final weights w to the hidden layer:

```
[[ 1.66730698e+00  1.23868788e+00  7.84511349e-01  8.51797297e-01
  -8.85709988e-01  3.41036881e-01  2.39416380e-01 -5.23059756e-02
   6.20448792e-01 -8.02266650e-02]
 [ 1.52765306e+00  1.39173576e+00  2.79380655e-01  9.58333004e-01
  -2.68967975e-01  3.90231225e-01  1.01193064e+00  2.00261949e-03
  2.72793660e-01 -5.04369097e-01]
 [ 3.49911098e+00  2.80945848e+00  1.21987272e+00  3.00167685e+00
   1.10296226e+00  1.85804857e+00  3.89533886e-01 -1.41379850e+00
   3.68346803e-01  1.28776251e+00]
 [ 2.97405944e-01  7.93413101e-01  5.45167189e-01  9.64405349e-01
   1.84838355e-01  1.10391638e+00  2.98797640e-01  7.29401412e-01
   8.69280893e-01  3.09019915e-01]
 [-2.48064400e-01  8.94581899e-02  5.33987987e-01 -3.44889472e-01
   9.28218489e-01  2.33141774e-01  4.63236892e-01  1.11050049e+00
   5.28035909e-01  5.72060189e-01]
 [ 7.23012027e-01  6.25855892e-01  2.35919441e-01  2.54684878e-01
   7.50999105e-01  4.35777188e-01  6.91627340e-01  7.65439878e-01
   9.81685591e-01  7.60059968e-01]
 [ 4.62706377e-01  6.28578489e-01 -3.30610097e-03  1.76781402e-01
   9.34065944e-01  1.79401060e-02  6.90315973e-01  5.46798715e-01
   6.52309236e-01  5.27509342e-01]
 [ 2.56503855e-01  2.46189599e-01  1.53436141e-01  6.90361523e-01
   5.23588735e-01  5.60849019e-01  2.64008251e-01  6.78486852e-01
   3.48421975e-01  3.40270366e-01]]
```
Final bias weights to each node of the hidden layer:
```
 [[-0.89217077]
 [-1.26124356]
 [-3.78224894]
 [ 0.21193218]
 [ 1.21336563]
 [ 0.65836194]
 [ 0.734335  ]
 [ 0.72072241]]
```
Final weights from hidden layer to output:
```
 [[ 1.72699895  1.67119091  4.79406238  0.42168987 -0.82768392 -0.49013076
  -0.41374219 -0.18840978]]
```
Final bias weight from hidden layer to output:
```
 [[-0.79382328]]
```

c) For M=7 hidden nodes:
accuracy_score 88.6986301369863
F-score 0.896551724137931

ii) For 1000 epochs and learning rate=0.05 and using tanh as the activation function for hidden layer,

For M=8 hidden nodes:
accuracy_score 89.04109589041096
F-score 0.89937106918239

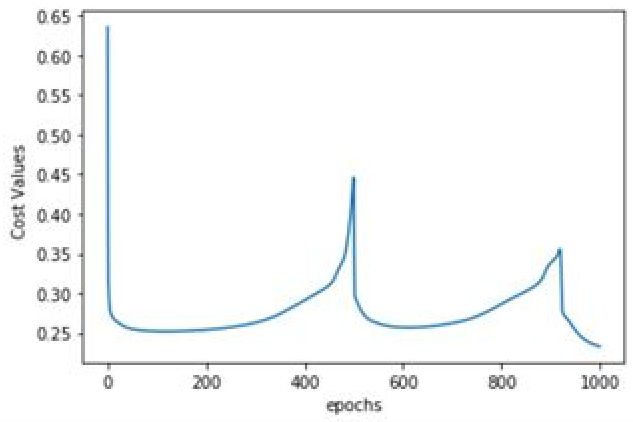


iii) For 1000 epochs and learning rate=0.1 and using tanh as the activation function for hidden layer,

For M=8 hidden nodes:

accuracy_score 88.01369863013699

F-score 0.8867313915857605



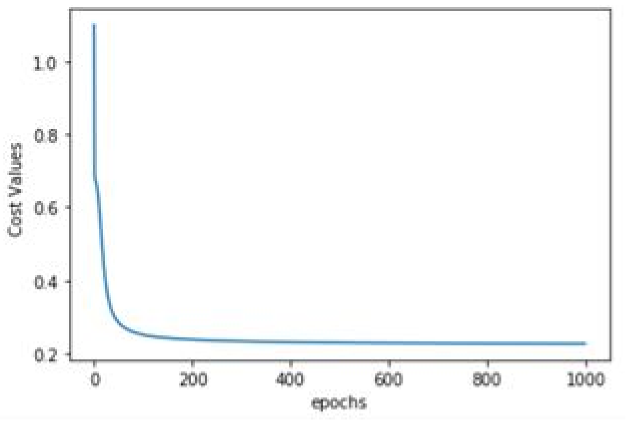
Due to higher learning rate, the error may spike sometimes.

iv) For 1000 epochs and learning rate=0.005 and using tanh as the activation function for hidden layer,

For M=8 hidden nodes:

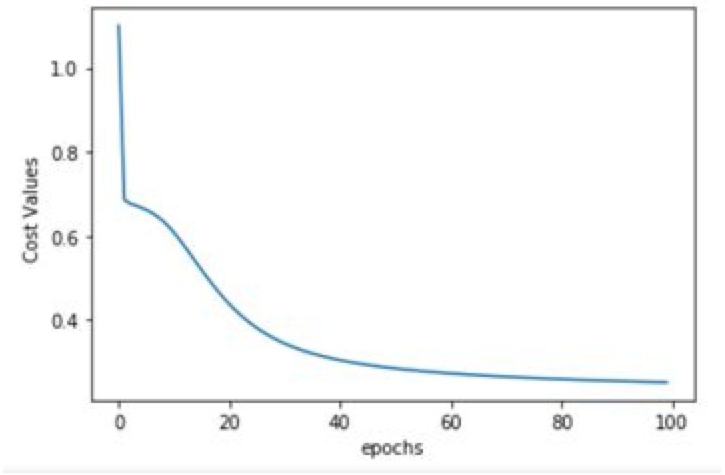
accuracy_score 89.04109589041096

F-score 0.9



v) For 100 epochs and learning rate=0.005 and using tanh as the activation function for hidden layer,

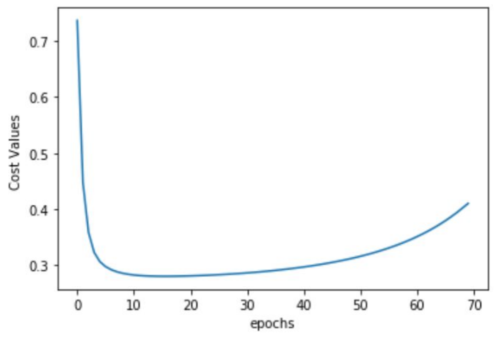
For M=8 hidden nodes:
accuracy_score 88.35616438356165
F-score 0.89375



If ReLU is taken as an activation function,

i) For 100 epochs and learning rate=0.05 and using tanh as the activation function for hidden layer,

For M=8 hidden nodes:
accuracy_score 56.5068493150685
F-score 0.7221006564551423

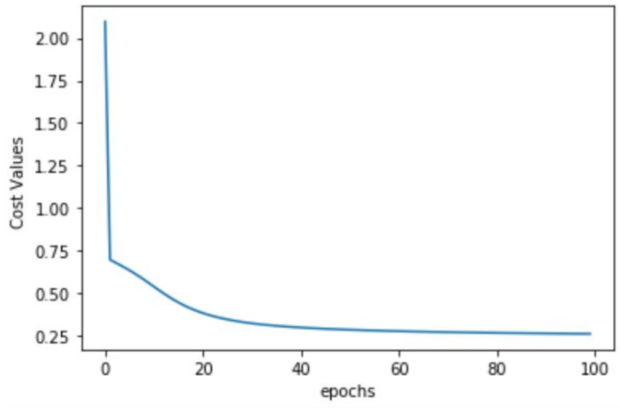


ii) For 1000 epochs and learning rate=0.005 and using tanh as the activation function for hidden layer,

For M=8 hidden nodes:

accuracy_score 80.82191780821918

F-score 0.8541666666666666

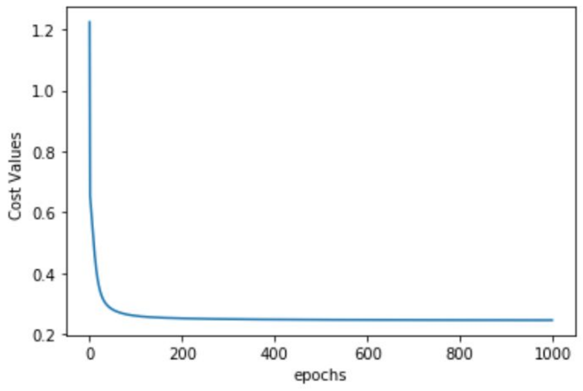


iii)For 1000 epochs and learning rate=0.005 and using tanh as the activation function for hidden layer,

For M=8 hidden nodes:

accuracy_score 74.65753424657534

F-score 0.8168316831683168



We have also used different activation functions like ReLU and sigmoid for the hidden layer but the results were not satisfactory for sigmoid even after changing the learning rate, no. of hidden nodes. The best activation function for this problem was ***tanh().***

Conclusion:

Therefore, from the above results we can say that the maximum accuracy that was achievable was around 86-89% for a particular random initialization. (random.seed=1111).
Even if we do not use random seed after 1000 epochs (i.e 1epoch= complete training data set), the error gets so low that the accuracy and F-score do not change significantly.