Machine Learning Assignment - 1 report

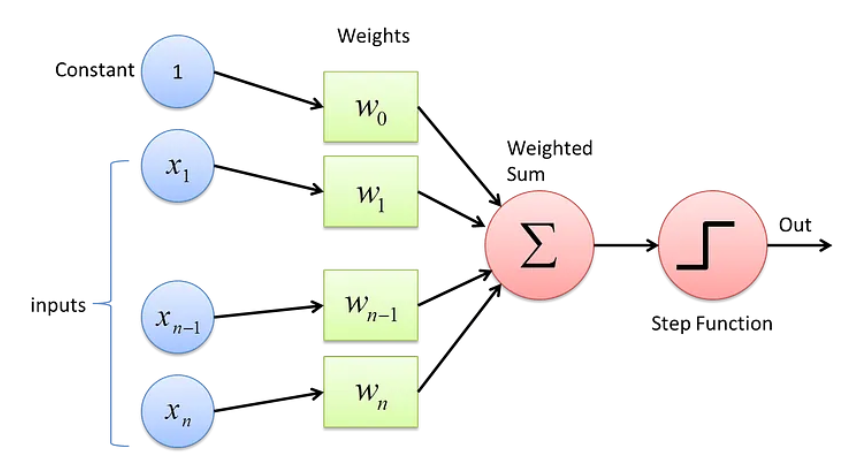
*By:*

*Tushar Dey Biswas (ID: 2020A4PS1987H)*

*Anushtup Nandy (ID: 2020A4PS1981H)*

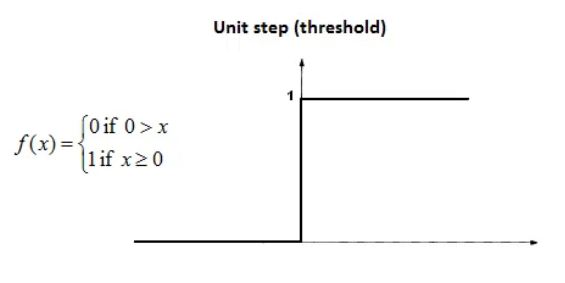
*Sarvaswa Mohata (ID: 2020B1AA2358H)*

## Perceptron Algorithm :

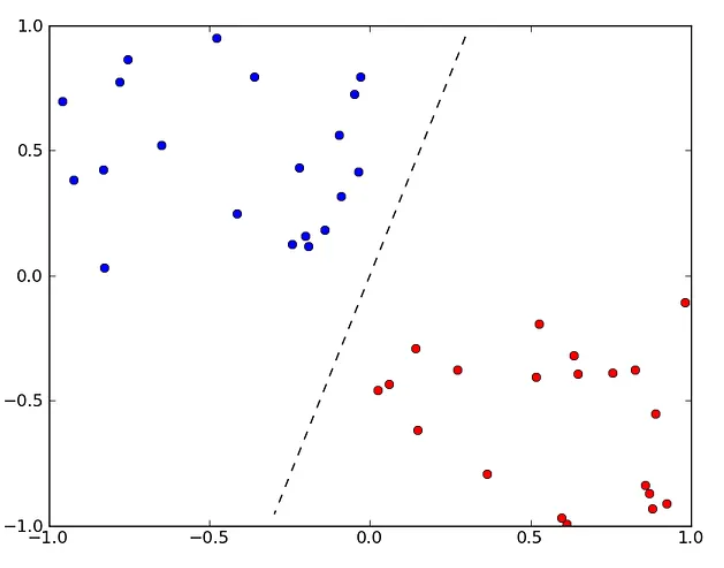


The perceptron algorithm executes with the help of following steps :

1. First, all the inputs are multiplied with their weights ‘w’.
2. Then we add all the multiplied values and call them weighted sum.
3. After this step, we apply the weighted sum to an activation function.



In this assignment, we have used the Perceptron Algorithm to check whether the data corresponding to the two types of tumors : Benign and Malignant are linearly separable.



## Implementation:

We have defined a class perceptron and initialized some instance variables including, **weights, bias, learning rate, and epochs(iteration).** The activation function has been defined as follows :

## 

The weights and bias have been initialized with 0 and get updated with each subsequent iteration according to the following mathematical equation :

where, denotes the learning rate.

A predict function has also been used governed by the following equation :

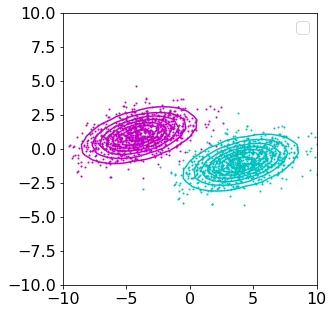
## Inferences :

The given dataset is **linearly separable**.

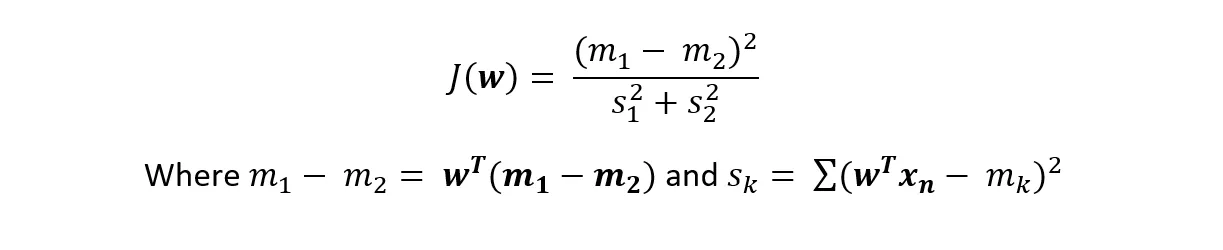
One possible way for checking whether the data is linearly separable could have been to plot the various features taken two at a time and check whether we could obtain a linear decision boundary separating the two features for each and every plot. But given 32 features, that was pretty much tedious to apply. So, we ran the perceptron algorithm on the dataset for 10000 epochs and the accuracy came out to be 94.68% for normalized data, and for the unnormalised one, the accuracy was close to 67%.

## Fischer LDA:

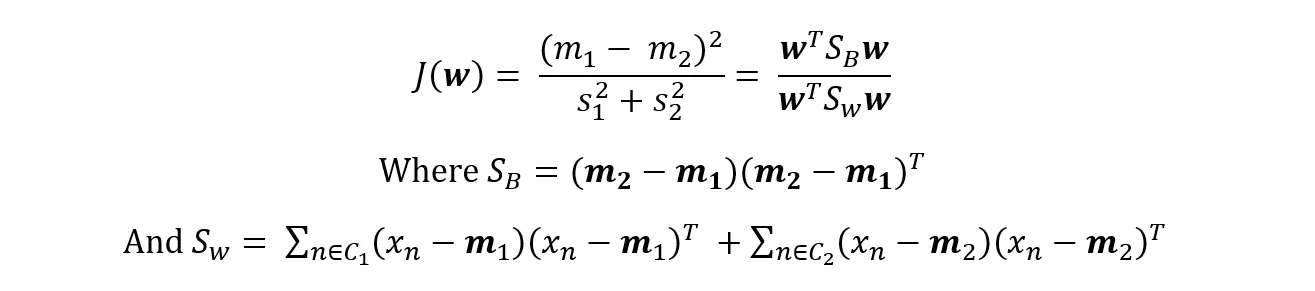
Fisher's linear discriminant can be used to make a classifier that learns with help. If the classifier is given data that has been labeled, it can find a set of weights that can be used to draw a decision limit and sort the data. Fisher's linear discriminant tries to find the vector that separates the classes of projected data as much as possible. "Separation" can be hard to define. Fisher's linear discriminant does this by making sure that the gap between the projected means is as big as possible and that the projected within-class variance is as small as possible.

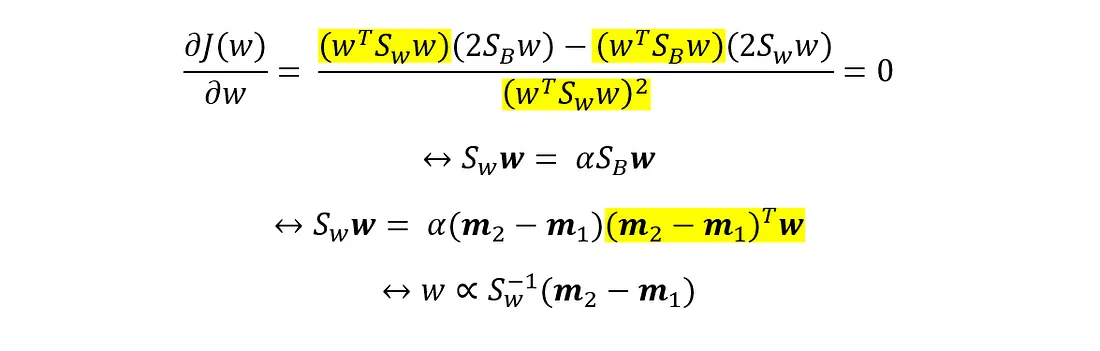


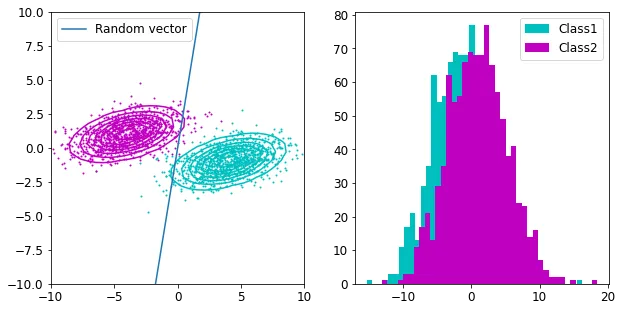
**Fisher’s criterion**



Find the weights vector that maximises the above equation to maximise Fisher's criterion. Getting the most out of this math is the same as what we saw before. Maximizing means to make the numerator (the distance between predicted means) as big as possible and to make the denominator as small as possible. (the within-class variance). We can rewrite Fisher's criterion by putting the formulae for the means and covariance in place of the means and covariance.







**Methodology**

**FLDA -1**

1. First we imported the dataset and checked for any NaNs and filled all such cells with 0s
2. Diagnosis column of the dataset was *one-hot encoded* and columns of diagnosis and ID were dropped.
3. The dataset which would become the training set was normalized and the then the one-hot encoded diagnosis-2 was added to it.
4. We then wrote functions for the following:
   1. Train-test splitting
   2. Divided the X and y into training and testing halves
   3. Then we segregated the training data into positive class and negative class
   4. Calculated Sw\_0 and Sw\_1
   5. Calculated mean and variance and mean difference
   6. Solved the equations for roots
   7. Plotted the roots of the equation
   8. The actual threshold value is the root between two means
   9. That x is now our threshold value
   10. Values lying left of the threshold values will be classified as Malignant and values lying toward the right will be classified as Benign
   11. Then to test the data we converted the target variable data into list , then we checked the value with corresponding y predicted then for each corresponding same value we are updating the count variable , then finally calculating the acuurary scores.
   12. We also calculated the f1 score

**FLDA -2**

Similar approach has been implemented in FLDA-2, only difference being shuffling the dataset column in a random permutation

**Results**

## 

Mean value of malignant = -0.47640398561250813

Threshold value(th\_val) = -0.07214067953779951

Mean value of benign = 0.2736370000005723

The threshold value we have obtained, th\_val is our separating hyperplane. for x < th\_val, the value is classified as Malignant and for x > th\_val its classified as Benign.

Accuracy: 0.9627659574468085

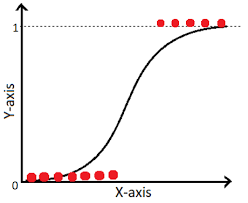
Recall: 0.9285714285714286

Precision: 0.9701492537313433

F1 Score: 0.948905109489051

## Logistic Regression:

It is a classification technique used when the target variable is *categorical* (example 0 or 1).

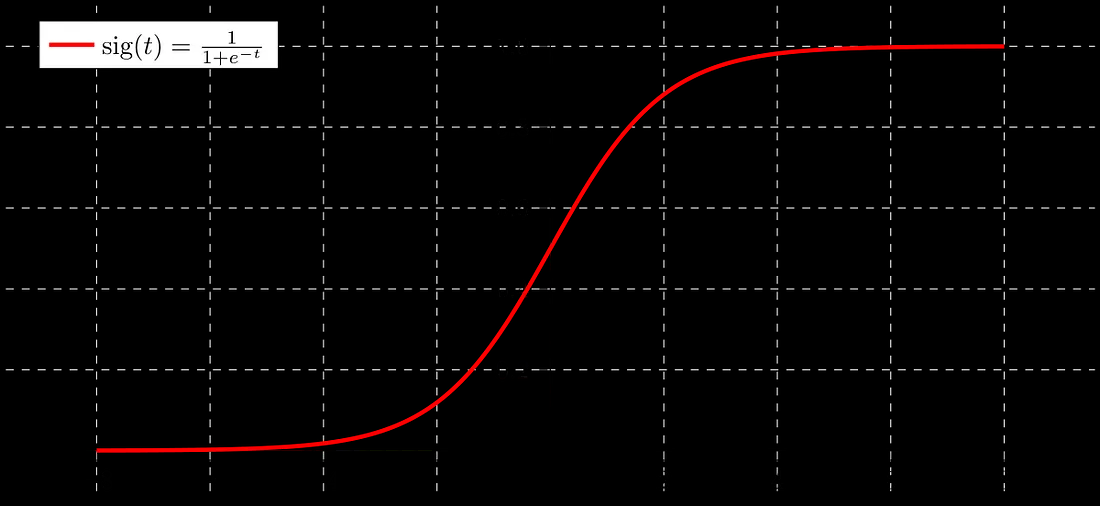


The math:

The hypothesis is ⇒ Z= Wx+B

The output is supposed to be one of 0 or 1

The probability is defined by ⇒ Sigmoid function:



If ‘Z’ goes to infinity, Y(predicted) will become 1 and if ‘Z’ goes to negative infinity, Y(predicted) will become 0.

The hypothesis’ output is considered as the estimated probability.This is used to infer how confident can predicted value be actual value when given an input X.

*- In mathematical format:*

h(x) = P(Y=1 | x) ⇒ probability of Y given x

and P(Y=0 | x) = 1- P(Y=1 | x)

Data is fit into linear regression model, which then be acted upon by a logistic function predicting the target categorical dependent variable.

*- Decision boundary:*

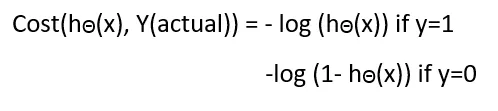
To predict which class a data belongs, a **threshold** can be set. Based upon this threshold, the obtained estimated probability is classified into classes.

If (y\_pred ≥ 0.5) ⇒ then classify 1 else 0.

Decision boundary can be linear or non-linear. Polynomial order can be increased to get complex decision boundary.

*- Cost Function:*

The cost function is decided by the NLL (or the negative log likelihood).



To converge this **gradient descent** is used ⇒ which will only converge if the function is a convex one!.

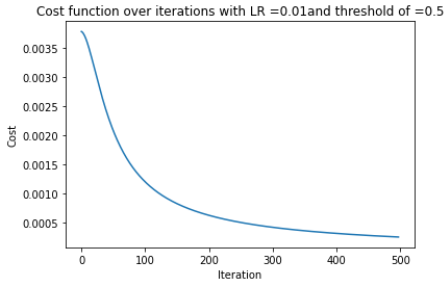
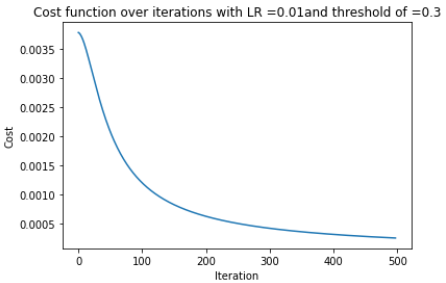
Negative function is because when we train, we need to maximize the probability by minimizing loss function. Decreasing the cost will increase the maximum likelihood assuming that samples are drawn from an identically independent distribution.

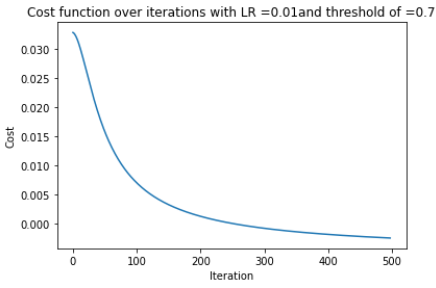
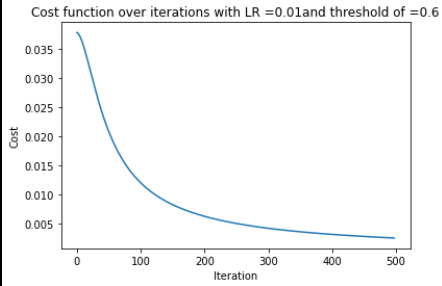
### ***LR-1:***

Methodology:

1. First we imported the dataset and checked for any NaNs and filled all such cells with 0s
2. Diagnosis column of the dataset was *one-hot encoded* and columns of diagnosis and ID were dropped.
3. The dataset which would become the training set was normalized and the then the one-hot encoded diagnosis-2 was added to it.
4. We then wrote functions for the following:
   1. Train-test splitting
   2. Divided the X and y into training and testing halves
   3. Then we converted them to numpy arrays
5. Lastly we wrote a class for the logistic regression model which contains the following functions:
   1. Initialization
   2. Sigmoid
   3. Compute\_cost
   4. Batch\_gradient\_descent
   5. Stochastic\_gradient\_descent
   6. Mini\_batch\_gradient\_descent
   7. Predict
   8. Plot\_cost

* **Plots and accuracy scores for threshold values of 0.5, 0.3, 0.4, 0.6 and 0.7 with a learning rate of 0.01 and batch-gradient descent:**





Threshold of 0.5:

Accuracy: 0.9468085106382979

Recall: 0.9821428571428571

Precision: 0.9322033898305084

Threshold of 0.3:

Accuracy: 0.9414893617021277

Recall: 0.9517241379310345

Precision: 0.93830985915493

Threshold of 0.4:

Accuracy: 0.9468085106382979

Recall: 0.9642857142857143

Precision: 0.9473684210526315

Threshold of 0.6:

Accuracy: 0.9414893617021277

Recall: 0.9517241379310345

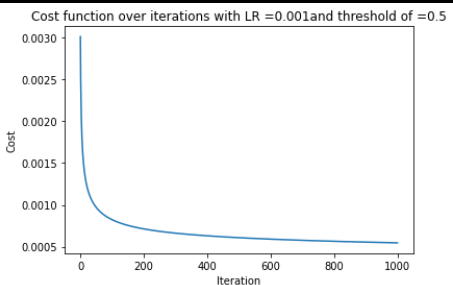
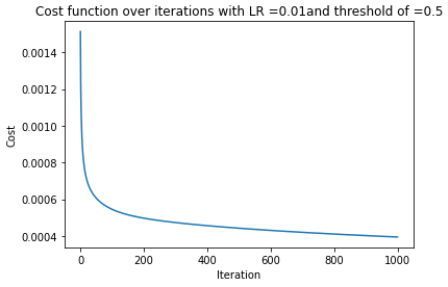
Precision: 0.971830985915493

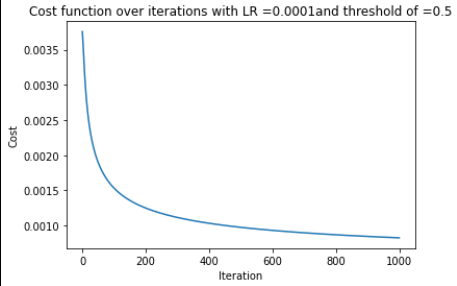
Threshold of 0.7:

Accuracy: 0.9468085106382979

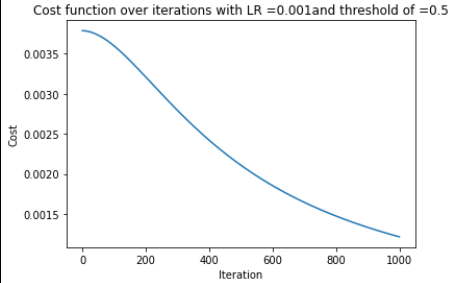
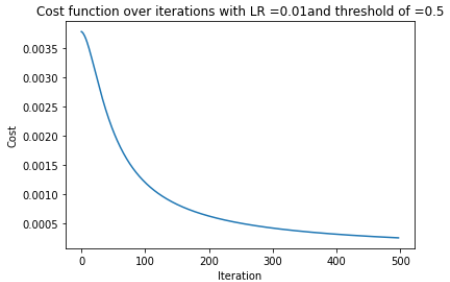
Recall: 0.9553571428571429

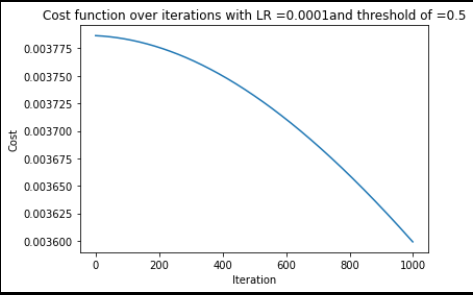
Precision: 0.9553571428571429

* **Plots for learning rates of 0.01, 0.001 and 0.0001 with stochastic gradient.**

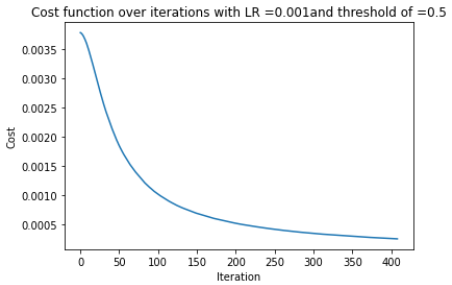


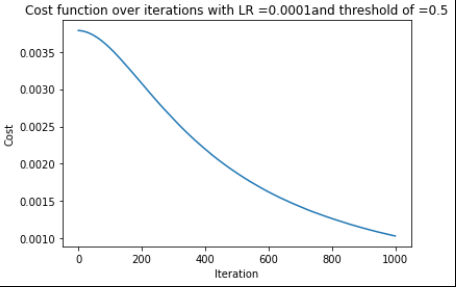
* **Plots for LR=0.01,0.001,0.0001 for batch gradient descent.**





* **Plots for mini-batch gradient descent on Learning rates of 0.01, 0.001, 0.0001.**





### ***LR-2:***

Methodology:

Same procedure as LR-1 was followed, instead, all the empty values were replaced with their **means** for continuous data and **mode** for discrete and the **X set was normalized**.

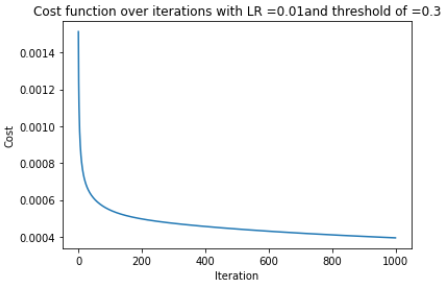
* **Plots and accuracy scores for 0.3, 0.5, 0.6 and 0.7 thresholds with stochastic gradient descent.**

Threshold values= 0.3

Accuracy: 0.9468085106382979

Recall: 0.9910714285714286

Precision: 0.925

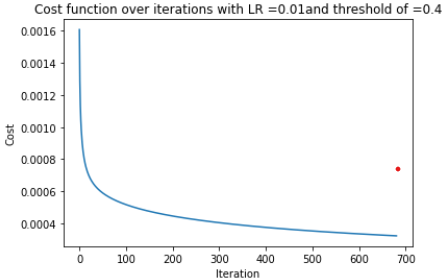
****

Threshold values= 0.4

Accuracy: 0.9414893617021277

Recall: 0.9821428571428571

Precision: 0.9243697478991597

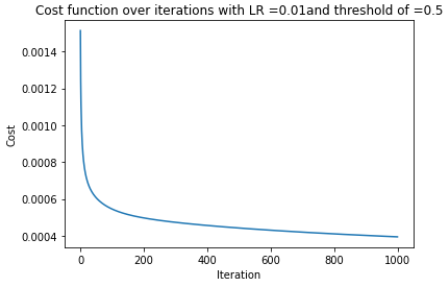
****

Threshold values= 0.5

Accuracy: 0.9468085106382979

Recall: 0.9821428571428571

Precision: 0.9322033898305084

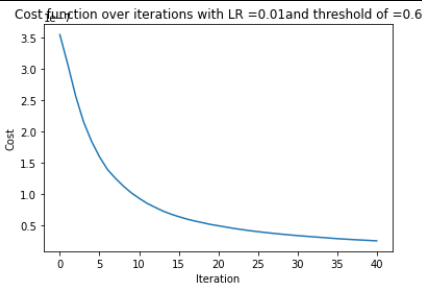


Threshold values= 0.6

Accuracy: 0.9414893617021277

Recall: 0.9732142857142857

Precision: 0.9316239316239316

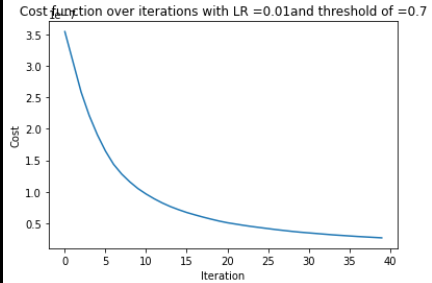


Threshold values= 0.7

Accuracy: 0.9414893617021277

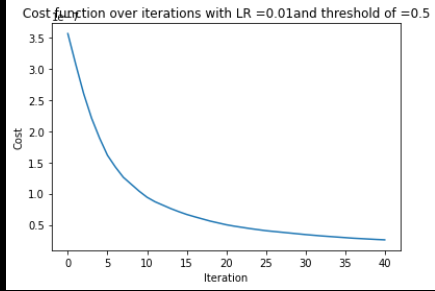
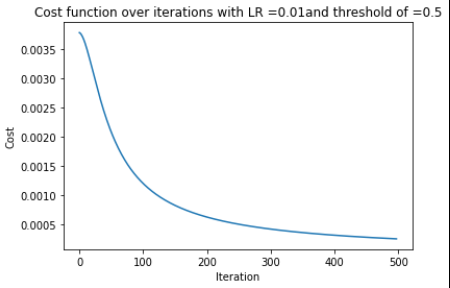
Recall: 0.9642857142857143

Precision: 0.93913043478260876737588

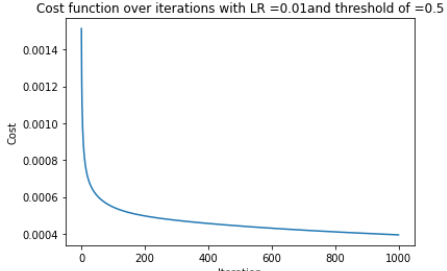


* **Plots for LR = 0.01 for the 3 gradient descents**

Mini-batch Batch

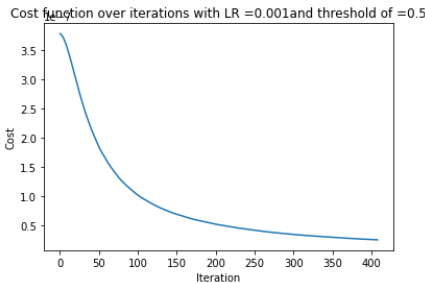
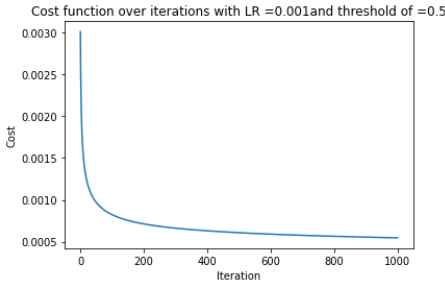
** **

Stochastic

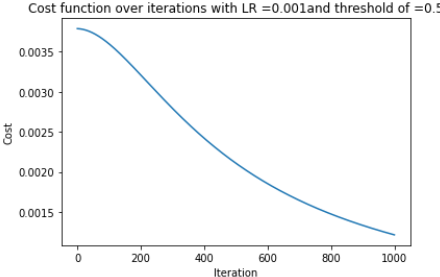


* **Plots for LR=0.001 for the 3 gradient descents**

Stochastic Mini-batch

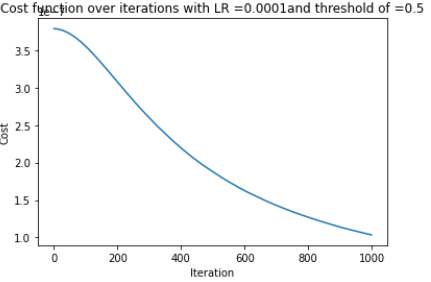
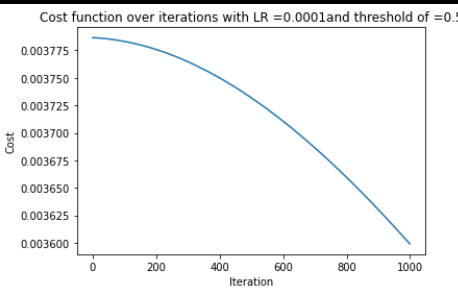


Batch

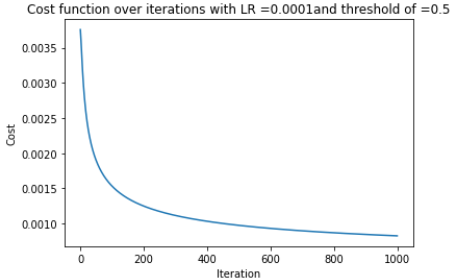


* **Plots for LR = 0.0001 for the 3 gradient descents**

Batch Mini -batch



Stochastic



## 

## COMPARATIVE STUDY:

| Sr. no | Random state value | Accuracy  (FLDA-1) | Accuracy (FLDA-2) | Accuracy (LR-1) | Accuracy  (Logistic-LR2-> stochastic gradient descent) | Accuracy  (PM-3) | Accuracy (PM-1) | Accuracy (PM-4) |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1. | 21 | 96.80851063829788 | 96.80851063829788 | 96.80851063829787 | 96.80851063829787 | 94.68085106382979 | 61.70212765957447 | 64.8936170212766 |
| 2. | 22 | 97.87234042553192 | 97.87234042553192 | 95.37659574468085 | 96.27659574468085 | 94.68085106382979 | 57.97872340425532 | 57.97872340425532 |
| 3. | 23 | 96.80851063829788 | 96.80851063829788 | 98.17234042553191 | 97.87234042553191 | 94.68085106382979 | 60.097263464794 | 66.48936170212766 |
| 4. | 24 | 97.87234042553192 | 97.87234042553192 | 98.6525531914894 | 98.40425531914894 | 94.14893617021277 | 61.17021276595744 | 60.63829787234043 |
| 5. | 25 | 98.40425531914893 | 98.40425531914893 | 97.98723 | 97.87234042553191 | 90.42553191489362 | 67.05147459571458 | 69.68085106382979 |
| 6. | 26 | 97.87234042553192 | 97.87234042553192 | 99.45659574468085 | 96.27659574468085 | 94.14893617021277 | 67.02127659574468 | 62.76595744680851 |
| 7. | 27 | 96.27659574468085 | 96.27659574468085 | 99.8904255319149 | 97.3404255319149 | 94.14893617021277 | 61.70212765957447 | 62.76595744680851 |
| 8. | 28 | 96.80851063829788 | 96.80851063829788 | 97.91851063829787 | 96.80851063829787 | 92.02127659574468 | 65.42553191489362 | 59.456457767 |
| 9. | 29 | 97.87234042553192 | 97.87234042553192 | 97.23455319149 | 97.3404255319149 | 92.5531914893617 | 59.57446808510638 | 63.29787234042553 |
| 10. | 30 | 96.80851063829788 | 96.80851063829788 | 99.918721063829787 | 96.80851063829787 | 92.02127659574468 | 61.70212765957447 | 57.97872340425532 |

#### ***F1 score for best random\_state (27):***

1. LR1-0.98766654489
2. LR2-0.97777777777
3. FLDA1 - 0.9466666666666667
4. FLDA2 - 0.9466666666666667

### ACCURACY VS RANDOM STATE for the 10 splits

