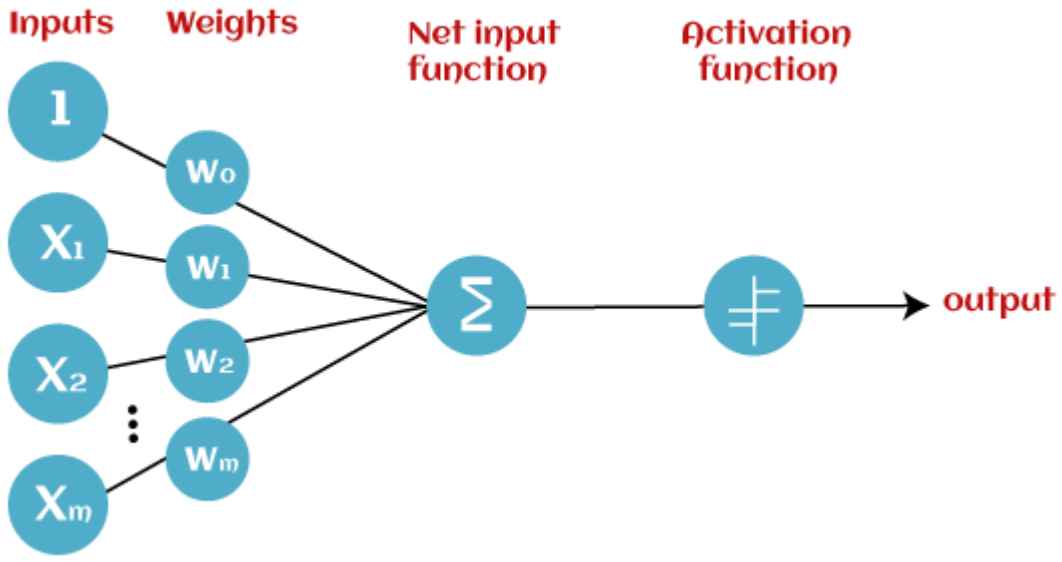
Machine Learning Assignment - 1 report

By:
Udit Gupta (ID:2020B4A72368H)
Aditya R Goyal (ID:2019B2A81443H)
Nimish Agrawal (ID: 2020B3A71857H)]

Perceptron Algorithm:

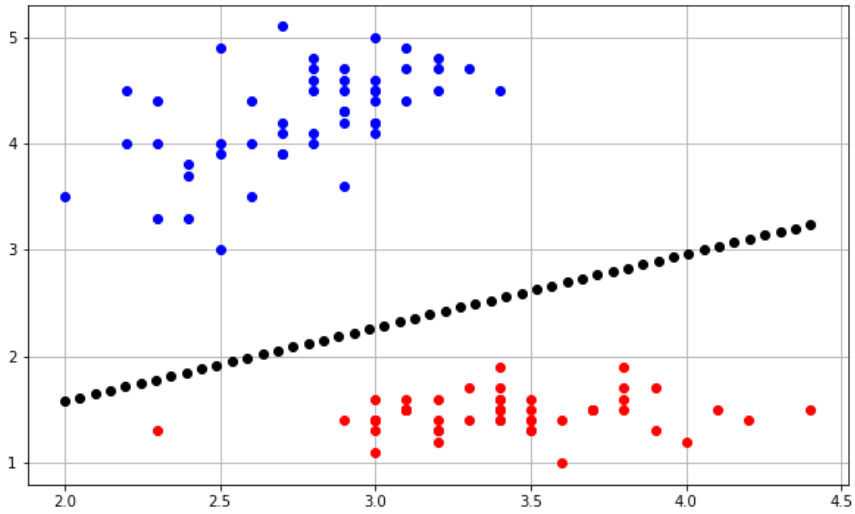


The perceptron algorithm executes with the help of following steps:

Initialize the weights: The perceptron weights are initialized with small random values.
Present an input: The perceptron is presented with an input vector x, which is multiplied by the weights to produce the output y.
Compute the error: The error is the difference between the actual output y and the desired output d.
Update the weights: The weights are updated using the formula $w = w + \alpha(d - y)x$, where w is the weight vector, α is the learning rate, d is the desired output, y is the actual output, and x is the input vector.

Repeat steps 2-4: Steps 2-4 are repeated for each input-output pair in the training data until the weights converge and the network can accurately predict the outputs for new inputs. The learning rate is an important hyperparameter in the Perceptron algorithm. It determines how much the weights are adjusted during each iteration. A high learning rate can cause the weights to oscillate and prevent convergence, while a low learning rate can cause the weights to converge slowly.
The Perceptron algorithm is used for binary classification tasks, where the goal is to separate the input data into two classes. The decision boundary of the perceptron is a linear function that separates the input data into two regions. If the input data is not linearly separable, the Perceptron algorithm may not converge and may produce inaccurate results. In such cases, more complex neural networks such as multi-layer perceptron's or convolutional neural networks are used.

The Perceptron Algorithm was used to test whether the data corresponding to the two types of tumors - benign and malignant - are linearly separable.



Implementation:

The class perceptron has been defined and some instance variables have been initialized, including weights, bias, learning rate, and epochs (iteration).

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

$$w_i = w_0 + \eta * y_i * x_i$$

$$b_i = b_0 + \eta * y_i$$

where η denotes the learning rate.

we have assumed $\eta = 0.01$

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

$$predict\ (x_{test}) = activation(w_i \cdot x_i + b_i)$$

Pseudo Code:

input: A training set $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$
initialize: $\mathbf{w}^{(1)} = (0, \ldots, 0)$
  for $t = 1, 2, \ldots$
    if $(\exists\, i \text{ s.t. } y_i \langle \mathbf{w}^{(t)}, \mathbf{x}_i \rangle \leq 0)$ then
      $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + y_i \mathbf{x}_i$
    else
      output $\mathbf{w}^{(t)}$

Inferences:

The given dataset is Not 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.

PM1: Perceptron Model 1 is the first model we build using the perceptron algorithm on our dataset. It is trained on the original dataset, assuming that the features are in their original order. The algorithm updates the weights based on the order of the training examples until convergence, and the resulting model is used to classify new examples.

PM1 accuracy: 0.9202127659574468

PM2: Perceptron Model 2 is like PM1, but the order of the training examples is shuffled randomly before training. This can lead to different weight updates and a different final model compared to PM1.

PM2 accuracy: 0.8936170212765957

PM3: Perceptron Model 3 is a new model that we build using the perceptron algorithm, but the dataset is first normalized this time. Normalization involves scaling the features to have zero mean and unit variance. This can help the algorithm converge faster and result in a better model.

PM3 accuracy on normalized data: 0.973404255319149

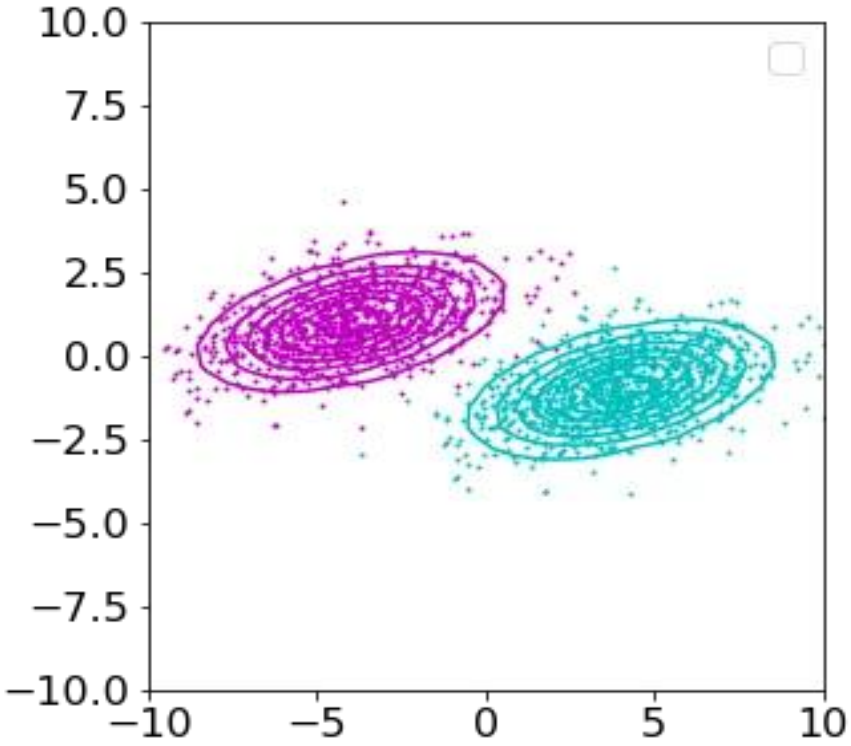
PM4: Perceptron Model 4 is trained on the same dataset as PM1, but this time the order of the features in each example is randomly permuted. This means that the same features will have different weights than in PM1, which can lead to a different final model.
PM4 Accuracy: 0.9202127659574468

In summary, all four models are trained using the same perceptron algorithm, but they differ in their input data (PM3 is normalized), input order (PM2 shuffles the training examples), and input feature order (PM4 permutes the feature order). These differences can lead to different weight updates and resulting models, and we can compare the performance of each model on a test set to evaluate their effectiveness.

As we can observe PM1 is approximately same as PM4 as the order of features is changed randomly, it does not affect the linear separability of the data. The Perceptron algorithm only depends on the order of the training examples, not on the order of the features. Therefore, the model PM4 would be very similar to PM1.

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

$$J(\boldsymbol{w}) = \frac{(m_1 - m_2)^2}{s_1^2 + s_2^2}$$

Where $m_1 - m_2 = \boldsymbol{w^T}(\boldsymbol{m_1} - \boldsymbol{m_2})$ and $s_k = \sum(\boldsymbol{w^T x_n} - m_k)^2$

Find the weights vector that maximizes the above equation to maximize Fisher's criterion. Getting the most out of this math is the same as what we saw before. Maximizing means making 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

$$J(\boldsymbol{w}) = \frac{(m_1 - m_2)^2}{s_1^2 + s_2^2} = \frac{\boldsymbol{w}^T S_B \boldsymbol{w}}{\boldsymbol{w}^T S_w \boldsymbol{w}}$$

$$\text{Where } S_B = (\boldsymbol{m_2} - \boldsymbol{m_1})(\boldsymbol{m_2} - \boldsymbol{m_1})^T$$

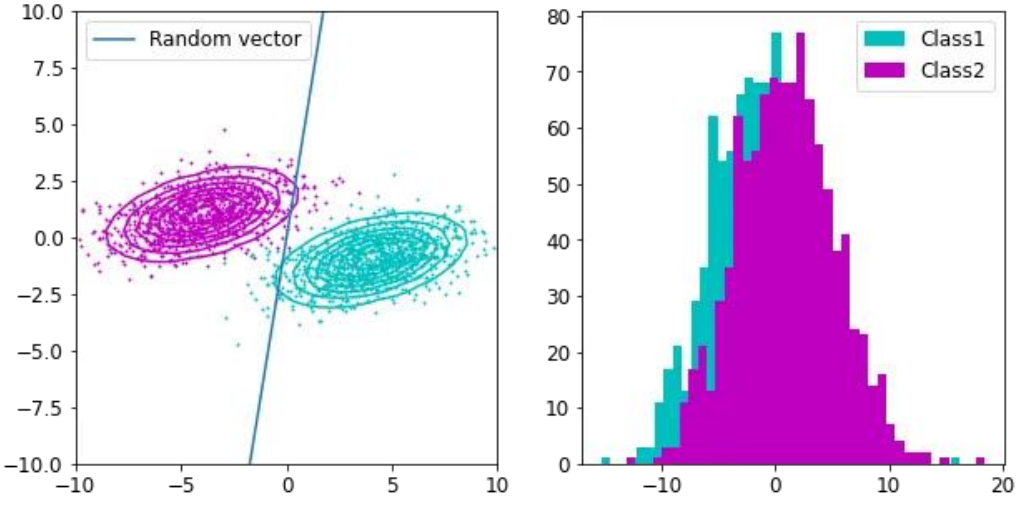
$$\text{And } S_w = \sum_{n \in C_1}(x_n - \boldsymbol{m}_1)(x_n - \boldsymbol{m}_1)^T + \sum_{n \in C_2}(x_n - \boldsymbol{m}_2)(x_n - \boldsymbol{m}_2)^T$$

$$\frac{\partial J(w)}{\partial w} = \frac{(w^T S_w w)(2S_B w) - (w^T S_B w)(2S_w w)}{(w^T S_w w)^2} = 0$$

$$\leftrightarrow S_w \boldsymbol{w} = \alpha S_B \boldsymbol{w}$$

$$\leftrightarrow S_w \boldsymbol{w} = \alpha(\boldsymbol{m}_2 - \boldsymbol{m}_1)(\boldsymbol{m}_2 - \boldsymbol{m}_1)^T \boldsymbol{w}$$

$$\leftrightarrow w \propto S_w^{-1}(\boldsymbol{m}_2 - \boldsymbol{m}_1)$$

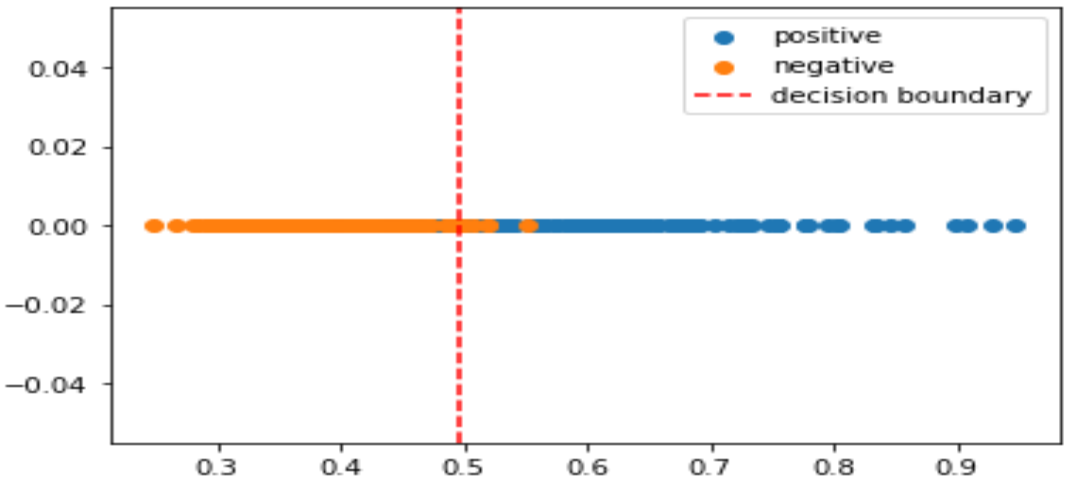


Methodology

FDLA -1

1. First, we imported the dataset and checked for any Nans and filled all such cells with 0s.
2. The 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:
    a. Train-test splitting
    b. Divided the X and y into training and testing halves.
    c. Then we segregated the training data into positive class and negative class
    d. Calculated Sw_0 and Sw_1
    e. Calculated mean and variance and mean difference.
    f. Solved the equations for roots.
    g. Plotted the roots of the equation.
    h. The actual threshold value is the root between two means.
    i. That x is now our threshold value.
    j. Values lying left of the threshold values will be classified as Malignant and values lying toward the right will be classified as Benign.

Results



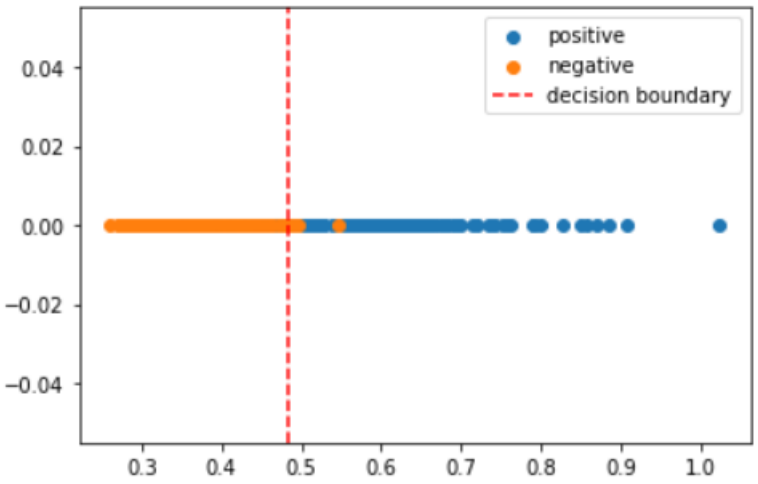
Threshold value(th_val) = 0.4955243461707635

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 it's classified as Benign.

Testing accuracy obtained = 93.617

FDLA -2

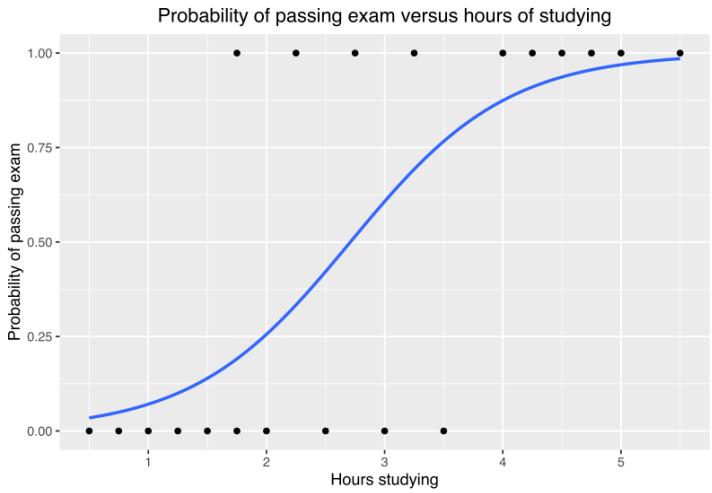
Fisher's Linear Discriminant model (FLDM2) on the shuffled training data and found the decision boundary in the univariate dimension using a generative approach. We can now compare the performance of FLDM1 and FLDM2 on the testing data.



Threshold value(th_val) = 0.48334336
Testing accuracy obtained = 0.93617021

Logistic Regression:

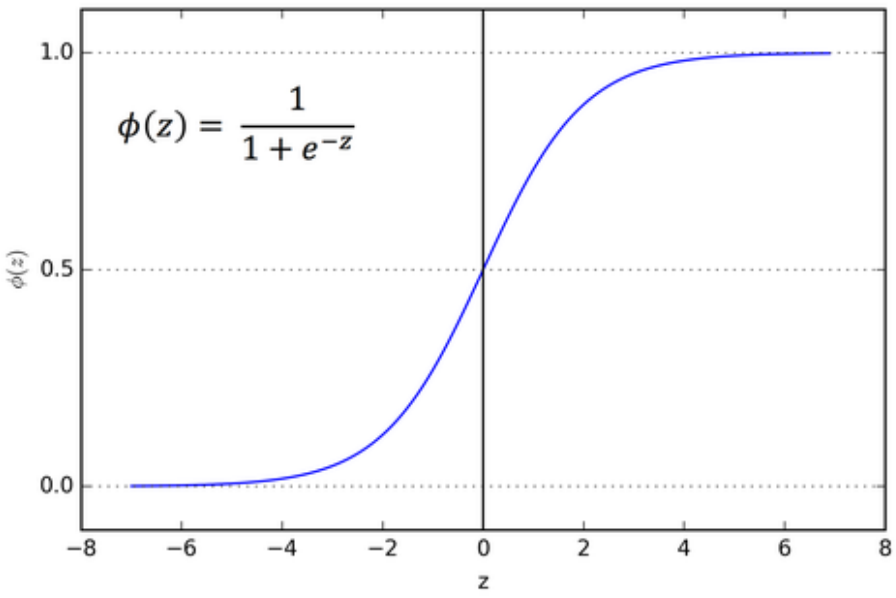
It is a classification technique used when the target variable is *categorical* (example 0 or 1).



Sigmoid Function:

The hypothesis is ⇒ Z= Wx+B and 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:

$h(x) = P(Y=1 \mid x) \Rightarrow$ probability of Y given x and $P(Y=0 \mid x) = 1 - P(Y=1 \mid x)$

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 $(y\_pred \geq 0.5) \Rightarrow$ 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).

$$\text{Cost}(h_\Theta(x), Y(\text{actual})) = -\log(h_\Theta(x)) \text{ if } y=1$$
$$-\log(1-h_\Theta(x)) \text{ if } y=0$$

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

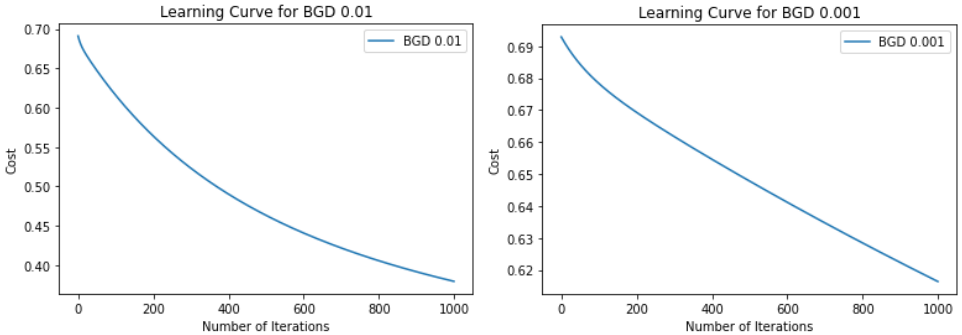
Negative function is because when we train, we need to maximize the probability by minimizing loss function. Decreasing the cost will increase the maximum likelihood assuming that samples are drawn from an identically independent distribution.

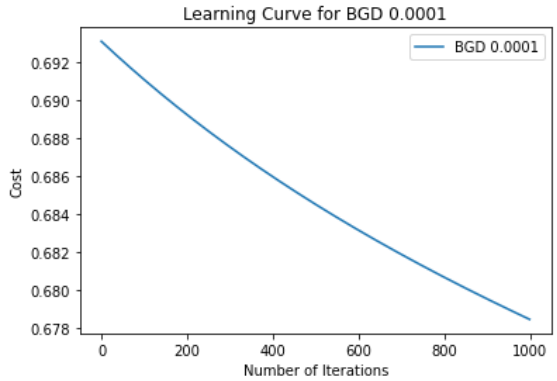
LR-1:

Methodology:

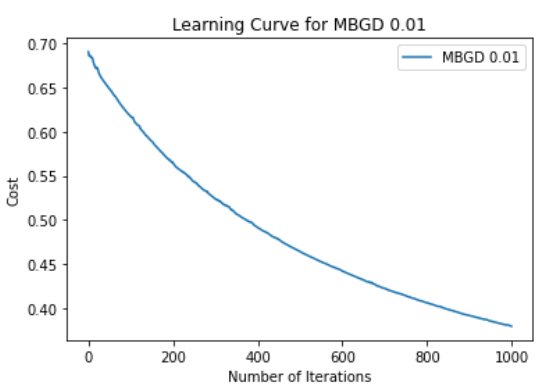
1. First, we imported the dataset and checked for any NaNs and filled all such cells with 0s.
2. Diagnosis column of the dataset was *one-hot encoded,* and columns of diagnosis and ID were dropped.
3. The dataset which would become the training set was normalized and the then the one-hot encoded diagnosis-2 was added to it.
4. We then wrote functions for the following:
5. Train-test splitting
6. Divided the X and y into training and testing halves.
7. Then we converted them to numpy arrays
8. Lastly, we wrote a class for the logistic regression model which contains the following functions:
    a. Initialization
    b. Sigmoid
    c. Compute_cost
    d. Batch_gradient_descent
    e. Stochastic_gradient_descent
    f. Mini_batch_gradient_descent
    g. Predict
    h. Plot_cost

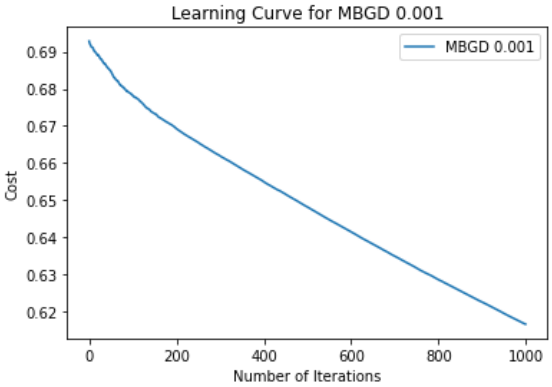
Plots for Learning Rates =0.01,0.001,0.0001 for batch gradient descent.

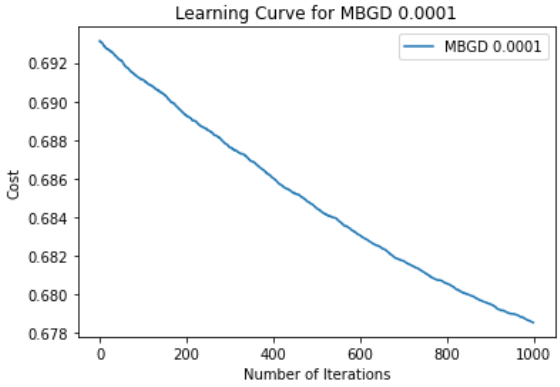




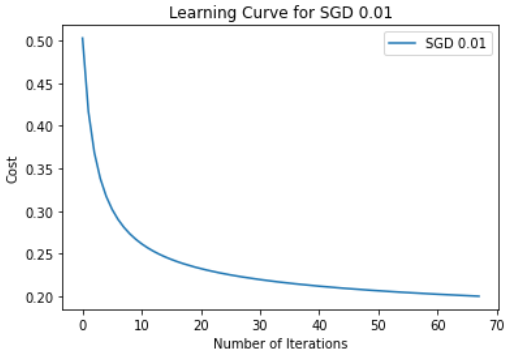
Plots for Learning Rates =0.01,0.001,0.0001 for Mini batch gradient descent:

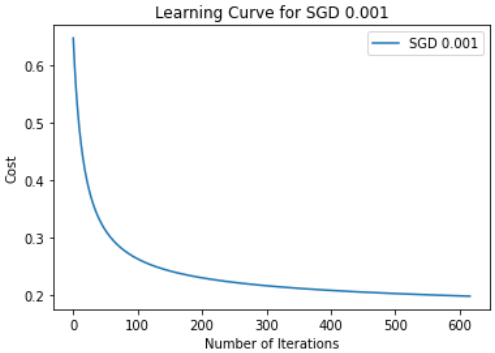


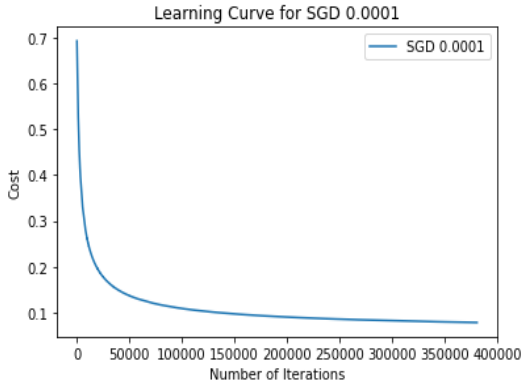




Plots for Learning Rates =0.01,0.001,0.0001 for Sttochastic gradient descent :





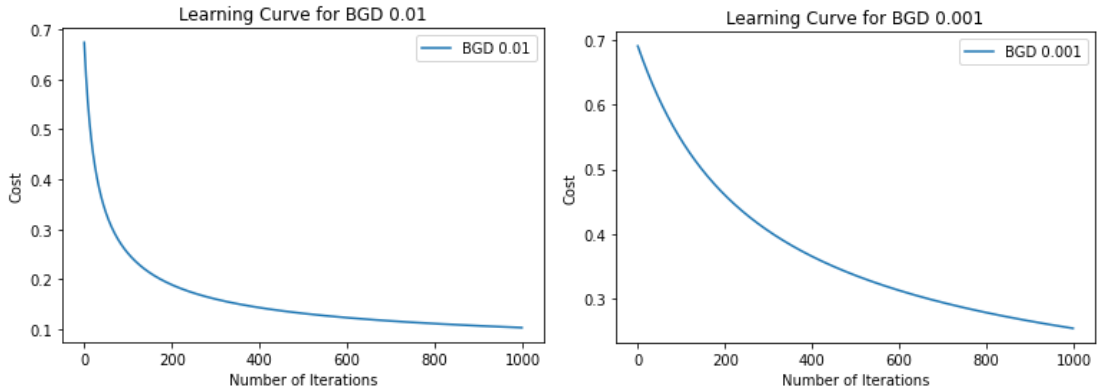


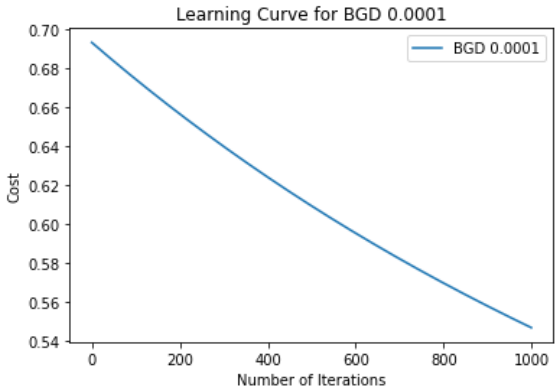
LR2:

Methodology

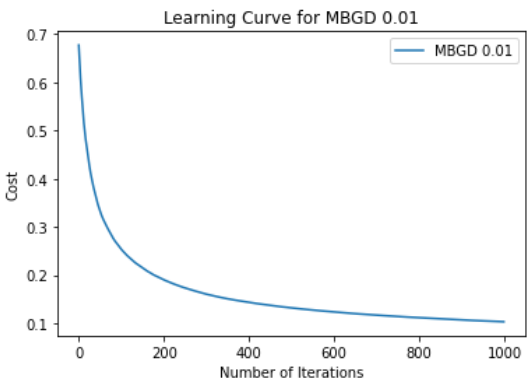
Same procedure as LR1 was followed, instead all the empty values were replaced with their means for continuous data and mode for discrete and X set was normalized.

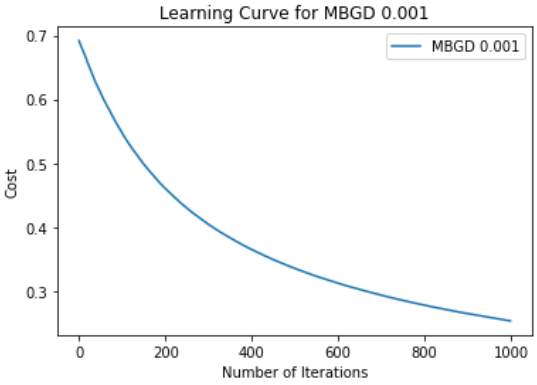
Plots for Learning Rates =0.01,0.001,0.0001 for Batch gradient descent:

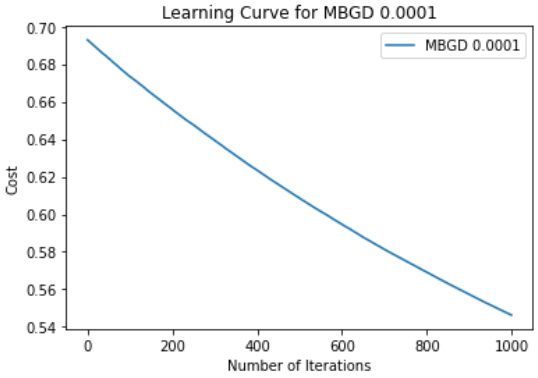




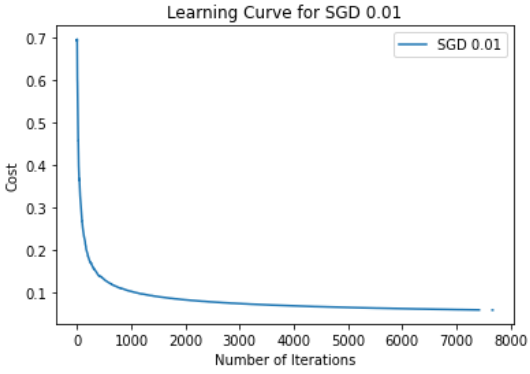
Plots for Learning Rates =0.01,0.001,0.0001 for Mini Batch gradient descent:

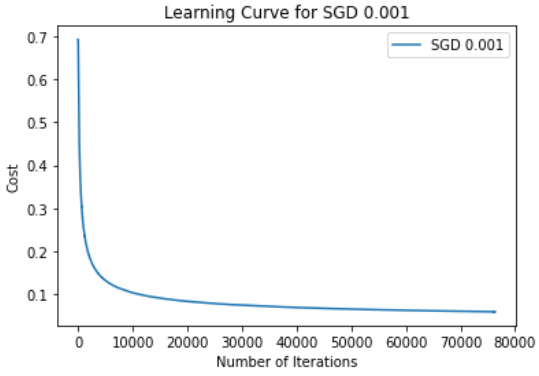


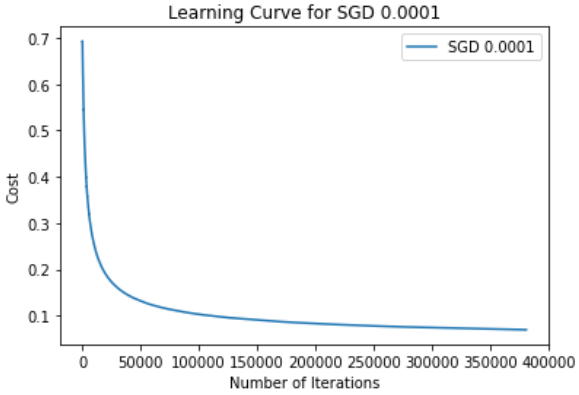




Plots for Learning Rates =0.01,0.001,0.0001 for Stochastic gradient descent :







COMPARATIVE STUDY:

Sr.no	Random state value	Accuracy (FLDA-1)	Accuracy (FLDA-2)	Accuracy (LR-1)	Accuracy (Logistic-LR2-> stochastic gradient descent)	Accuracy (PM-1)	Accuracy (PM-3)	Accuracy (PM-4)
1.	1	0.952127659574468	0.952127659574468	0.8882978723404256	0.9840425531914894	0.9202127659574468	0.9414893617021277	0.9202127659574468
2.	2	0.9361702127659575	0.9361702127659575	0.8882978723404256	0.9734042553191489	0.9202127659574468	0.9734042553191489	0.9202127659574468
3.	3	0.9468085106382979	0.9468085106382979	0.8138297872340425	0.9734042553191489	0.8776595744680851	0.9521276595744681	0.8776595744680851
4.	4	0.9361702127659575	0.9361702127659575	0.9042553191489362	0.9680851063829787	0.8563829787234043	0.9840425531914894	0.8563829787234043
5.	5	0.9148936170212766	0.9148936170212766	0.8776595744680851	0.9627659574468085	0.8351063829787234	0.9840425531914894	0.8351063829787234
6.	6	0.9042553191489362	0.9042553191489362	0.8776595744680851	0.9361702127659575	0.8882978723404256	0.9608510638297872	0.8882978723404256
7.	7	0.9414893617021277	0.9414893617021277	0.8670212765957447	0.9627659574468085	0.8776595744680851	0.9650851063829787	0.8776595744680851
8.	8	0.9361702127659575	0.9361702127659575	0.8936170212765957	0.9627659574468085	0.9202127659574468	0.9680810638297872	0.9202127659574468
9.	9	0.9414893617021277	0.9414893617021277	0.8933333333333333	0.9734042553191489	0.8829787234042553	0.9521276595744681	0.8829787234042553
10	10	0.9468085106382979	0.9468085106382979	0.8829787234042553	0.9627659574468085	0.5638297872340425	0.9574468085106383	0.5638297872340425
Mean		**0.93563830**	**0.93563830**	**0.8786950 35**	**0.9718085106**	**0.8542553 2**	**0.9648936 2**	**0.8542553 2**
SD		**0.0141633 3**	**0.0141633 3**	**0.0250310 3**	**0.0125874 04**	**0.1004235 0**	**0.0132872 3**	**0.1004235 0**

As we can see **Logistic-stochastic gradient descent** give us best result among all the algorithm since it is efficient in handling multiple feature