BITS F464 Machine Learning

Assignment-1 Report

Suraj Nair - 2020A7PS0051H

Piyush Kumar Sahu - 2020A7PS2042H

Pingale Sagar Subhash - 2020A7PS2062H

Description of the model:

With the help of provided code we are able to implement 3 different machine learning algorithms - Perceptron algorithm, Fisher Linear Discriminant Analysis and Logistic Regression - to predict whether a tumor is benign or malignant. The main idea behind these models is to understand which algorithm yields better results and draw inferences as to why such a result is obtained.

We implemented the algorithms in the following way:

1. Initially, we read the data from the CSV file provided using the pandas module present in python. Since the dataset has empty cells we perform feature engineering tasks to fill in the values as and when required. Then we normalize the dataset if required.
2. Next, we perform a 67-33 train - test split for training the models and testing later.

Fisher Linear Discriminant Analysis

- First of all we have defined a function called find_w where we calculate the means of the positive and negative classes and also find out the inclass variance.
- We take the inverse of the sum of inclass variance , i.e , $(s1^2 + s2^2)^{-1}$ as we have to maximize the difference of means and maximize the inclass variance
- Then we define a function called solve_pdfs that takes as input the projections of the data on a single dimensional vector and their respective standard deviations. This function calculates decision boundary. This function returns the threshold below which an example is classified as negative and above it is classified as positive.
- The function plot_pdfs which is used to plot the probability distributions of positive and negative classes (assumed to be gaussian).

- The function threshold is being used to calculate the threshold for below which the example will be classified as negative point.
- Apart from all these functions we have a plot_hist function that helps us to visualize the gaussian distribution obtained from above. Accuracy function is used to compute the accuracy of the model generated.

Test_Train split number	Training Accuracy	Testing Accuracy
1	0.9814323607427056	0.9623655913978495
2	0.986737400530504	0.9516129032258065
3	0.9893899204244032	0.9516129032258065
4	0.9734748010610079	0.9731182795698925
5	0.986737400530504	0.9354838709677419
6	0.9814323607427056	0.9623655913978495
7	0.9787798408488063	0.9731182795698925
8	0.9787798408488063	0.9623655913978495
9	0.9734748010610079	0.9623655913978495
10	0.9787798408488063	0.978494623655914

*Here the yellow highlighted text is the best accuracy obtained for the 10 random splits.

Perceptron Algorithm

- A function called step is used that returns 0 or -1 depending on if the input to the function is positive or negative. Another function fit is used to add a column of 1's in the input dataset.
- The function linear_fit is defined that keeps running till we are able to get linearly separable points. It also keeps track of the number of misclassified points
- Shuffle_split_data function is used to shuffle the rows and then split the data into training and testing dataset.
- Function predict is used to predict the outputs for all the inputs and store the result as a matrix. The output which is X.w (dot product of X and w) is then passed on to the step function defined above.
- Then we define a function predictOne which works similarly as predict function except that this function predicts for just one input i.e , input is a vector.
- A function called checkStatistics has been implemented to calculate the accuracy , recall and precision of the model.
- Lastly the function checkAccuracy is used to check the accuracy of our trained model based on how many points we classified correctly .
- Lastly we make four different models PM1 , PM2, PM3 and PM4 as specified in the question.

PM1 :

epochs	Misclassified	Accuracy	Recall	Prec
0	168	0.77265	0.95735	0.62928
5000	52	0.90941	0.75829	1
10000	45	0.91829	0.80095	0.97688
15000	45	0.90941	0.76303	0.99383
20000	41	0.90764	0.75829	0.99379
25000	37	0.92007	0.79147	0.99405
30000	35	0.92895	0.82464	0.98305
35000	38	0.90053	0.7346	1
40000	38	0.89876	0.7346	0.99359
45000	35	0.9325	0.83886	0.9779
50000	38	0.92007	0.79147	0.99405

NOTE:
For unnormalized data:
At 2,50,000 epochs -> 21 points are being misclassified.
At 60,40,000 epochs -> 23 still misclassified.

PM2 :

These are the results we obtained for PM2 -

- Accuracy: 0.9627659574468085
- Recall: 0.92

- Precision: 0.9857142857142858

PM3 :

These are the results we obtained for PM3 -

- Accuracy: 0.9840425531914894
- Recall: 0.9714285714285714
- Precision: 0.9855072463768116

PM4 :

These are the results we obtained for PM4 -

- Accuracy: 0.9627659574468085
- Recall: 0.92
- Precision: 0.9857142857142858

Cost Function :-

The cost function in logistic regression is given by:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}[-y^{(i)}log(h_\theta(x^{(i)})) - (1-y^{(i)})log(1-h_\theta(x^{(i)}))]$$

and the gradient of the cost is a vector of the same length as θ where the j^{th} element (for j = 0, 1, ..., n) is defined as follows:

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

Note that while this gradient looks identical to the linear regression gradient, the formula is actually different because linear and logistic regression have different definitions of $h_\theta(x)$.

Our aim in all the above algorithms is to minimize the cost function and find the optimum w (weight vector), as this w is the one which gives the least error or in other terms maximum accuracy. To find the minima of this function we make use of techniques such as Stochastic , batch, mini batch gradient descent algorithms

Logistic Regression

- A function called sigmoid_activation is used that applies the sigmoid function on the obtained final result
- The function model first of all calculates X_train**.**w (dot product of X_train and w) and then applies the activation function on the obtained result.
    - Error is defined as y_train - activated output
    - We then calculate the cost function and try to find the minima of this function using different forms of gradient descent algorithms with varying thresholds

- A function called accuracy is defined to calculate the accuracy of the model.
- Firstly we implement the batch gradient descent function followed by mini batch and stochastic gradient descent to obtain the minima of cost functions.

Comparative Analysis of the obtained models

- First we have created 10 different train-test splits and then run PM1, PM3 and PM4 on these train-test splits and observe the results

Accuracy obtained for PM1 : 0.9325
Accuracy obtained for PM4 : 0.9627659574468085
Accuracy obtained for PM3 : 0.9840425531914894

****

- Now we again split the dataset into 10 different train-test split and run FLDM1 and FLDM2
- **Accuracy** : 0.9515957446808512
- **Variance** : 0.00030811453146220005

  The accuracy obtained for both the models is the same . This is because changing the columns does not make any change as we get the same projections when we project these points on a unidimensional vector.

****

Here we first divide the dataset into 10 different train test splits and then for every **η** and threshold value (total 15 models for every pair), and then

repeat the process for LR2 and out of the obtained six models we perform a comparative study

LR2

****

Accuracy_bgd

Train_Test split no.	Accuracy
1	0.9677419354838710
2	0.9693548387096770
3	0.9591397849462370
4	0.9672043010752690
5	0.9731182795698920
6	0.9725806451612900
7	0.9623655913978500
8	0.9725806451612900
9	0.974731182795699
10	0.9688172043010750

Accuracy_mbgd

Train_Test split no.	Accuracy
1	0.9720430107526880
2	0.9623655913978490
3	0.946236559139785
4	0.971505376344086
5	0.967741935483871
6	0.9575268817204300
7	0.9693548387096770
8	0.9790322580645160
9	0.9666666666666670
10	0.9774193548387100

Accuracy_sgd

Train_Test split no.	Accuracy
1	0.9564516129032260
2	0.8618279569892470
3	0.3720430107526880
4	0.9612903225806450
5	0.9274193548387100
6	0.5112903225806450
7	0.975268817204301
8	0.9612903225806450
9	0.9274193548387100
10	0.9629032258064520

****

LR1

Accuracy table

sgd	bgd	mbgd
0.82096774193548	0.8032258064516130	0.8876344086021510
0.77043010752688	0.8258064516129030	0.7989247311827960
0.80752688172043	0.8881720430107530	0.8989247311827960
0.70483870967741	0.9107526881720430	0.8564516129032260

0.73602150537634	0.8951612903225810	0.832258064516129
0.68763440860215	0.9096774193548390	0.7908602150537630
0.68548387096774	0.9026881720430110	0.828494623655914
0.72204301075268	0.89247311827957	0.8758064516129030
0.80806451612903	0.893010752688172	0.8306451612903230
0.77419354838709	0.8580645161290320	0.8731182795698920
0.86075268817204	0.8543010752688170	0.878494623655914
0.74784946236559	0.8591397849462360	0.8704301075268820
0.67204301075268	0.896774193548387	0.8693548387096770
0.81612903225806	0.8569892473118280	0.8543010752688170
0.83333333333333	0.9016129032258070	0.8516129032258060

****

Final Comparative Table :

Method Used	Accuracy
LR2 SGD	0.975268817204301
LR2 BGD	0.9731182795698920
LR2 MBGD	0.9790322580645160

LR1 SGD	0.86075268817204
LR1 BGD	0.9107526881720430
LR1 MBGD	0.8989247311827960
PM1	0.93673
PM3	0.9740425531914894
PM4	0.9627659574468085
FLDM1	0.978494623655914
FLDM2	0.978494623655914

Conclusion :

From here we can observe that the best accuracy is obtained while using LR2 using mini batch gradient (batch size of 32) with **η** =0.01 and threshold = 0.5 descent as the model.