

MACHINE LEARNING ASSIGNMENT 1

Birla Institute of Technology and Science, Hyderabad

Kush Desai(2021B4A73158H)
Manav Sahni(2021A8PS2199H)
Om Narayan(2021A7PS1324H)

March 27, 2023

Introduction

In this report, our group has trained machine learning models using several supervised learning techniques in order to predict the outputs for the problem statement. These machine-learning approaches are providing good results for decision-making in academic fields. this report consists of several machine learning algorithms trained using some given data in order to predict if a tumor is benign or malignant based on the values of some of its given features. The report is divided into four parts- the first part uses the perceptron learning algorithm, the second part uses Fisher's linear discriminant analysis, the third part contains logistic regression, and the fourth part is a comparative study of all three learning algorithms based on their accuracy score.

Problem Statement

The problem solved by this report is to make a framework to analyse and predict the kind of tumor using AI based techniques that use some given data without the need to manually evaluate it.

The Dataset

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430
...
564	926424	M	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890
565	926682	M	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791
566	926954	M	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302
567	927241	M	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200
568	92751	B	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000

569 rows × 32 columns

...	radius_worst	texture_worst	perimeter_worst	area_worst	smoothness_worst	compactness_worst	concavity_worst	concave points_worst	symmetry_worst
...	25.380	17.33	184.60	2019.0	0.16220	0.66560	0.7119	0.2654	0.4601
...	24.990	23.41	158.80	1956.0	0.12380	0.18660	0.2416	0.1860	0.2750
...	23.570	25.53	152.50	1709.0	0.14440	0.42450	0.4504	0.2430	0.3613
...	14.910	26.50	98.87	567.7	0.20980	0.86630	0.6869	0.2575	0.6638
...	22.540	16.67	152.20	1575.0	0.13740	0.20500	0.4000	0.1625	0.2364
...
...	25.450	26.40	166.10	2027.0	0.14100	0.21130	0.4107	0.2216	0.2060
...	23.690	38.25	155.00	1731.0	0.11660	0.19220	0.3215	0.1628	0.2572
...	18.980	34.12	126.70	1124.0	0.11390	0.30940	0.3403	0.1418	0.2218
...	25.740	39.42	184.60	1821.0	0.16500	0.86810	0.9387	0.2650	0.4087
...	9.456	30.37	59.16	268.6	0.08996	0.06444	0.0000	0.0000	0.2871

General Procedure

Although every algorithm is different from one another, there are a few steps that are common while performing any machine-learning algorithm, which are as follows:

Feature engineering (optional): Feature engineering refers to manipulation — addition, deletion, combination, mutation — of your data set to improve machine learning model training, leading to better performance and greater accuracy.

Train test split: we are a set of data, a portion of which is used to train our machine learning model, and once the model is finalized, the remaining data called the testing data, is used to test how accurate our trained machine learning model actually is.

A model with a fit and predict functions: this includes the implementation of the machine learning algorithm. The fit function is used to train the classifier, while the predict function is used to predict the output for new inputs once the model is trained.

Classifier: A classifier in machine learning is an algorithm that automatically orders or categorizes data into one or more of a set of “classes.”

Calculation of accuracy: the final step in our techniques is to calculate how accurate our model is for a new set of training data.

Part A: Perceptron Learning Algorithm

This technique is used to classify the data if the features of the dataset are linearly separable. The perceptron model can theoretically give us 100% accuracy, provided the data is linearly separable.

We have trained 4 models using this algorithm, which resulted in the following observations:

Model Name	Epochs	Accuracy (Train/Test)	Confusion Matrix	Precision	Recall
PM1	1000	(0.9259, 0.9351)	[[38, 7] [5, 135]]	0.8444	0.8837
PM2	1000	(0.9233, 0.9135)	[[58, 2] [14, 111]]	0.9666	0.8055
PM3	2158 (Terminated)	(1.0, 0.9251)	[[40, 11] [3, 133]]	0.7843	0.9302
PM4	1000	(0.9259, 0.9351)	[[38, 7] [5, 135]]	0.8444	0.8837

- PM1 and PM2 have different accuracies during test as well as training as their rows are shuffled, leading to a different random state. This will cause weights to change differently, hence leading to the same.
- PM1 is not normalized whereas PM3 is normalized (both feature engineering tasks 1 and 2 are applied). This allows PM3 to have less outliers and become centred around a Gaussian Distribution.
- In PM3, Training Accuracy = 100%. Thus, we can conclude that the data is **Linearly Separable**.
- PM4 has its attributes shuffled, but since the order of plotting the points in the same random state does not matter, it will not show any deviation from PM1. Thus, PM1 is same as PM4.
- Perceptron Learning Algorithm does not depend on Learning Rate

Part B: Fisher's Linear Discriminant Analysis

We have seen how the perceptron learning algorithm can help us achieve high accuracy. However, the drawbacks of using this algorithm are that it requires the points to be linearly separable and is computationally very expensive hence developing a model gets costlier. To overcome these problems, we try to reduce the number of dimensions of our training features to a single dimension so that the originality of the features is retained as much as possible. Hence, we try to project our D-dimensional data features onto a one-dimensional vector, 'w.'

Here's how we achieve it:

Say x_1, x_2, \dots, x_N are 'N' points each one having 'D' dimensions, to be classified into class c_1 or c_2 .

Any general $x_n =$ (a D-dimensional vector)

$$\begin{bmatrix} x_{1n} \\ x_{2n} \\ \vdots \\ x_{Dn} \end{bmatrix}$$

Mean of all points in class $c_1 = m_1' =$

$$\frac{1}{N_1} \sum_{x_n \in c_1} x_n$$

Mean of all points in class $c_2 = m_2' =$

$$\frac{1}{N_2} \sum_{x_n \in c_2} x_n$$

Now, we can choose the w for which $m_2' - m_1'$ is maximized.

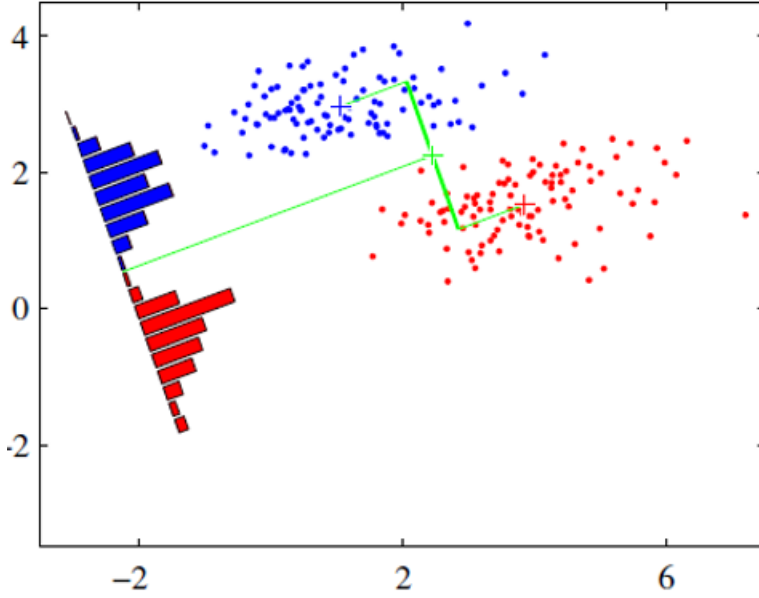
Fisher's condition: $|m_2 - m_1|$ should be maximized AND the total variance should be minimized.

Hence, our aim is to maximise $\frac{(m_2 - m_1)^2}{(s_1^2 + s_2^2)}$, where,

s_1^2 and s_2^2 are variances of projected points of classes c_1 and c_2

m_1 = mean of projected points belonging to class c_1

m_2 = mean of projected points belonging to class c_2



Hence,

$$m1 = \frac{1}{N1} \sum_{xn \in c1} w^T xn = w^T m1'$$

$$m2 = \frac{1}{N2} \sum_{xn \in c2} w^T xn = w^T m2'$$

$$s1^2 = \frac{1}{N1} \sum_{xn \in c1} (w^T xn - m1)^2 = \frac{1}{N1} \sum_{xn \in c1} (w^T xn - w^T m1')^2 = \frac{1}{N1} \sum_{xn \in c1} (w^T (xn - m1'))^2$$

$$= \frac{1}{N1} \sum_{xn \in c1} [(w^T (xn - m1'))(w^T (xn - m1'))^T]$$

$$= w^T (\frac{1}{N1} \sum_{xn \in c1} [(xn - m1')(xn - m1')^T]) w$$

$$\text{Similarly, } s2^2 = w^T (\frac{1}{N2} \sum_{xn \in c2} [(xn - m2')(xn - m2')^T]) w$$

Thus, $s1^2 + s2^2 = w^T s_w w$, where,

$$s_w = (\frac{1}{N1} \sum_{xn \in c1} [(xn - m1')(xn - m1')^T]) + (\frac{1}{N2} \sum_{xn \in c2} [(xn - m2')(xn - m2')^T])$$

$$\text{Now, } (m2 - m1)^2 = (w^T m2' - w^T m1')^2 = w^T (m2' - m1')(m2' - m1')^T w = w^T s_B w$$

$$\text{Where, } s_B = (m2' - m1')(m2' - m1')^T$$

$$\text{Max}[\frac{(m2-m1)^2}{(s1^2+s2^2)}] = \text{max}(\frac{w^T s_B w}{w^T s_w w})$$

$$\text{Therefore, } \frac{d}{dw} (\frac{w^T s_B w}{w^T s_w w}) = 0$$

$$\text{This gives: } s_B w (w^T s_w w) = s_w w (w^T s_B w) \text{ or } s_w w \propto s_B w$$

$$\text{Now, } s_B w = (m2' - m1')(m2' - m1')^T w$$

$$\text{Hence, } s_B w \propto (m2' - m1')$$

Or, $w \propto S_w^{-1}(m_2' - m_1')$

This is the result we have used in the program.

After this, we have to project the points using $w^T \cdot x$, and then find if x belongs to c_1 or c_2 . For this, we will use the generative approach using Bayes' Theorem (assuming gaussian distribution for the projected points).

$$p(x \in c_1 | w^T x) = p(w^T x | x \in c_1) p(x \in c_1) / p(w^T x)$$

$$p(x \in c_2 | w^T x) = p(w^T x | x \in c_2) p(x \in c_2) / p(w^T x)$$

Hence, $x \in c_1$ if $p(x \in c_1 | w^T x) > p(x \in c_2 | w^T x)$ else $x \in c_2$

Here, $p(x \in c_1)$ and $p(x \in c_2)$ can be found out from the training data.

$p(w^T x | x \in c_1)$ and $p(w^T x | x \in c_2)$ are the prior probabilities of $w^T x$ according to the gaussian probability distributions.

We developed two models using this algorithm- FLDM1 and FLDM2

FLDM1: In this model, we apply the fisher's linear discriminant analysis on the raw data and thus reduce the 32-dimensional problem to univariate dimensional problem.

Confusion Matrix = $\begin{bmatrix} 57 & 2 \\ 2 & 125 \end{bmatrix}$

Accuracy: 0.9785

Precision: 0.9661

Recall: 0.9661

FLDM2: In this model, we randomly change the order of the features in the dataset and apply the fisher's linear discriminant analysis on the raw data and thus reduce the 32-dimensional problem to univariate dimensional problem. **We get an accuracy of 97.849%, which is the same as the accuracy we got for FLDM1.** This is because no matter the order of the features, ultimately all the points will be the same in the 30-Dimensional space, and hence, the projections of these points will also be the same, giving us the same results.

Confusion Matrix = $\begin{bmatrix} 57 & 2 \\ 2 & 125 \end{bmatrix}$

Accuracy: 0.9785

Precision: 0.9661

Recall: 0.9661

Part C: Logistic Regression

In this model, we use the sigmoid function to find the posterior probability that our data point belongs to a certain class without going into the generative process. The sigmoid function is defined by

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

hence, the probability that our point x is a positive point is given by

$$p(\text{pos} | x) = \frac{1}{1+e^{-w \cdot x}}$$

hence, all we have to do is to find that value of w which makes our data most likely to happen. We achieve this by finding the negative log likelihood function (error function) of the data (a convex function) and minimizing it using the three gradient descent algorithms- batch, mini-batch and stochastic.

We use this approach with learning rates of 0.01, 0.001, and 0.0001 and vary the probability thresholds as 0.3, 0.4, 0.5, 0.6 and 0.7.

Also, the batch size used for mini-batch gradient descent = 20.

Number of Epochs for all the Models below = 1000

We have plotted the final values of accuracy, recall and precision in the following tables-

LR1: In this model, we apply the logistic regression on the raw data. Given below are the results of our model-

For **Batch Gradient Descent**

Learning Rate	Probability Threshold	Confusion Matrix	Accuracy	Accuracy from 0.5	Precision	Recall
0.01	0.3	[[67, 15] [5, 99]]	0.8924	No change	0.8171	0.9306
0.01	0.4	[[67, 15] [5, 99]]	0.8294	No change	0.8171	0.9306
0.01	0.5	[[67, 15] [5, 99]]	0.8294	-	0.8171	0.9306
0.01	0.6	[[67, 15] [5, 99]]	0.8294	No change	0.8171	0.9306
0.01	0.7	[[67, 15] [5, 99]]	0.8294	No change	0.8171	0.9306
0.001	0.3	[[67, 15] [5, 99]]	0.8294	No change	0.8171	0.9306
0.001	0.4	[[67, 15] [5, 99]]	0.8294	No change	0.8171	0.9306
0.001	0.5	[[67, 15] [5, 99]]	0.8294	-	0.8171	0.9306
0.001	0.6	[[67, 15] [5, 99]]	0.8294	No change	0.8171	0.9306
0.001	0.7	[[67, 15] [5, 99]]	0.8294	No change	0.8171	0.9306
0.0001	0.3	[[71, 25] [1, 89]]	0.8602	Decreases	0.7396	0.9861
0.0001	0.4	[[71, 23] [1, 91]]	0.8709	Decreases	0.7553	0.9861
0.0001	0.5	[[69, 20] [3, 94]]	0.8763	-	0.7753	0.9583
0.0001	0.6	[[69, 16] [3, 98]]	0.8978	Increases	0.8117	0.9583
0.0001	0.7	[[68, 15] [4, 99]]	0.8978	Increases	0.8193	0.9444

For Mini-Batch Gradient Descent

Learning Rate	Probability Threshold	Confusion Matrix	Accuracy	Accuracy from 0.5	Precision	Recall
0.01	0.3	[[65, 7] [7, 107]]	0.9247	No change	0.9027	0.9027
0.01	0.4	[[65, 7] [7, 107]]	0.9247	No change	0.9027	0.9027
0.01	0.5	[[65, 7] [7, 107]]	0.9247	-	0.9027	0.9027
0.01	0.6	[[65, 7] [7, 107]]	0.9247	No change	0.9027	0.9027
0.01	0.7	[[65, 7] [7, 107]]	0.9247	No change	0.9027	0.9027
0.001	0.3	[[66, 7] [6, 107]]	0.9301	No change	0.9041	0.9167
0.001	0.4	[[66, 7] [6, 107]]	0.9301	No change	0.9041	0.9167
0.001	0.5	[[66, 7] [6, 107]]	0.9301	-	0.9041	0.9167
0.001	0.6	[[66, 7] [6, 107]]	0.9301	No change	0.9041	0.9167
0.001	0.7	[[65, 7] [7, 107]]	0.9247	Decreases	0.9027	0.9027
0.0001	0.3	[[68, 10] [4, 104]]	0.9247	Increases	0.8718	0.9444
0.0001	0.4	[[68, 9] [4, 105]]	0.9301	Increases	0.8831	0.9444
0.0001	0.5	[[66, 9] [6, 105]]	0.9193	-	0.88	0.9167
0.0001	0.6	[[66, 9] [6, 105]]	0.9193	No change	0.88	0.9167
0.0001	0.7	[[66, 7] [6, 107]]	0.9301	Increases	0.9041	0.9167

For Stochastic Gradient Descent

Learning Rate	Probability Threshold	Confusion Matrix	Accuracy	Accuracy from 0.5	Precision	Recall
0.01	0.3	[[72, 50] [0, 64]]	0.7312	Decreases	0.5902	1.0
0.01	0.4	[[68, 9] [4, 105]]	0.9301	Decreases	0.8831	0.9444
0.01	0.5	[[63, 1] [9, 113]]	0.9462	-	0.9844	0.875
0.01	0.6	[[69, 13] [3, 101]]	0.9140	Decreases	0.8415	0.9583
0.01	0.7	[[71, 20] [1, 94]]	0.8871	Decreases	0.7802	0.9861
0.001	0.3	[[50, 0] [22, 114]]	0.8817	Decreases	1.0	0.6944
0.001	0.4	[[65, 2] [7, 112]]	0.9516	No change	0.9701	0.9028
0.001	0.5	[[65, 2] [7, 112]]	0.9516	-	0.9701	0.9028
0.001	0.6	[[63, 1] [9, 113]]	0.9462	Decreases	0.9843	0.875
0.001	0.7	[[66, 7] [6, 107]]	0.9301	Decreases	0.9041	0.9167
0.0001	0.3	[[63, 1] [9, 113]]	0.9462	Increases	0.9843	0.875
0.0001	0.4	[[69, 15] [3, 99]]	0.9032	Decreases	0.8214	0.9583
0.0001	0.5	[[69, 11] [3, 103]]	0.9247	-	0.8625	0.9583
0.0001	0.6	[[65, 2] [7, 112]]	0.9516	Increases	0.9701	0.9028
0.0001	0.7	[[65, 2] [7, 112]]	0.9516	Increases	0.9701	0.9028

LR2: In this model, we apply the logistic regression on the normalised dataset (after applying feature engineering tasks 1 and 2). Given below are the results of our model-

For **Batch Gradient Descent**

Learning Rate	Probability Threshold	Confusion Matrix	Accuracy	Accuracy from 0.5	Precision	Recall
0.01	0.3	[[69, 3] [2, 114]]	0.9734	Decreases	0.9583	0.9718
0.01	0.4	[[69, 0] [2, 117]]	0.9893	Increases	1.0	0.9718
0.01	0.5	[[68, 0] [3, 117]]	0.9840	-	1.0	0.9577
0.01	0.6	[[65, 0] [6, 117]]	0.9680	Decreases	1.0	0.9155
0.01	0.7	[[61, 0] [10, 117]]	0.9468	Decreases	1.0	0.8592
0.001	0.3	[[70, 16] [1, 101]]	0.9096	Decreases	0.8140	0.9859
0.001	0.4	[[69, 7] [2, 110]]	0.9521	Increases	0.9080	0.9718
0.001	0.5	[[63, 3] [8, 116]]	0.9512	-	0.9844	0.8873
0.001	0.6	[[61, 1] [10, 116]]	0.9145	Decreases	0.9839	0.8592
0.001	0.7	[[56, 1] [15, 116]]	0.9149	Decreases	0.9825	0.7887
0.0001	0.3	[[71, 117] [0, 0]]	0.3777	Decreases	0.3777	1.0
0.0001	0.4	[[71, 86] [0, 31]]	0.5426	Decreases	0.4522	1.0
0.0001	0.5	[[63, 3] [8, 114]]	0.9415	-	0.9545	0.8873
0.0001	0.6	[[37, 1] [34, 116]]	0.8138	Decreases	0.9736	0.5211
0.0001	0.7	[[8, 0] [63, 117]]	0.6649	Decreases	1.0	0.1127

For Mini-Batch Gradient Descent

Learning Rate	Probability Threshold	Confusion Matrix	Accuracy	Accuracy from 0.5	Precision	Recall
0.01	0.3	[[69, 1] [2, 116]]	0.9804	No Change	0.9857	0.9718
0.01	0.4	[[69, 0] [2, 117]]	0.9894	Increases	1.0	0.9718
0.01	0.5	[[68, 0] [3, 117]]	0.9804	-	1.0	0.9577
0.01	0.6	[[66, 0] [5, 117]]	0.9734	Decreases	1.0	0.9296
0.01	0.7	[[64, 0] [7, 117]]	0.9628	Decreases	1.0	0.9014
0.001	0.3	[[69, 2] [2, 115]]	0.9787	Decreases	0.9718	0.9718
0.001	0.4	[[69, 0] [2, 117]]	0.9894	Increases	1.0	0.9718
0.001	0.5	[[68, 0] [3, 117]]	0.9804	-	1.0	0.9577
0.001	0.6	[[65, 0] [6, 117]]	0.9680	Decreases	1.0	0.9155
0.001	0.7	[[63, 0] [8, 117]]	0.9574	Decreases	1.0	0.8873
0.0001	0.3	[[70, 9] [1, 108]]	0.9468	Decreases	0.8861	0.9859
0.0001	0.4	[[69, 4] [2, 113]]	0.9681	No change	0.9452	0.9718
0.0001	0.5	[[66, 1] [5, 116]]	0.9681	-	0.9501	0.9296
0.0001	0.6	[[62, 1] [9, 116]]	0.9468	Decreases	0.9841	0.8731
0.0001	0.7	[[59, 0] [12, 117]]	0.9361	Decreases	1.0	0.8310

For Stochastic Gradient Descent

Learning Rate	Probability Threshold	Confusion Matrix	Accuracy	Accuracy from 0.5	Precision	Recall
0.01	0.3	[[68, 1] [3, 116]]	0.9787	Increases	0.9855	0.9577
0.01	0.4	[[69, 1] [2, 116]]	0.9840	Increases	0.9857	0.9718
0.01	0.5	[[66, 1] [5, 116]]	0.9681	-	0.9851	0.9296
0.01	0.6	[[66, 1] [5, 116]]	0.9681	No change	0.9851	0.9296
0.01	0.7	[[65, 1] [6, 116]]	0.9628	Decreases	0.9848	0.9155
0.001	0.3	[[68, 1] [3, 116]]	0.9787	No change	0.9855	0.9577
0.001	0.4	[[68, 0] [3, 117]]	0.9840	Increases	1.0	0.9577
0.001	0.5	[[67, 0] [4, 117]]	0.9787	-	1.0	0.9437
0.001	0.6	[[67, 0] [4, 117]]	0.9787	No change	1.0	0.9437
0.001	0.7	[[65, 0] [6, 117]]	0.9681	Decreases	1.0	0.9155
0.0001	0.3	[[69, 2] [2, 115]]	0.9787	Decreases	0.9718	0.9718
0.0001	0.4	[[69, 0] [2, 117]]	0.9894	Increases	1.0	0.9718
0.0001	0.5	[[68, 0] [3, 117]]	0.9840	-	1.0	0.9577
0.0001	0.6	[[65, 0] [6, 117]]	0.9681	Decreases	1.0	0.9155
0.0001	0.7	[[65, 0] [6, 117]]	0.9681	Decreases	1.0	0.9155

Part D: Comparative Study

Model Name	Accuracy (Maximum in case of LR)
PM1	0.9351
PM3	0.9251
PM4	0.9351
FLDM1	0.9785
FLDM2	0.9785
LR1	0.9516
LR2	0.9894

As clear from the above table, LR2 is the best model out of all the given models. This is because:

1. LR2 is normalized, thus no outliers as in the case if PM1, PM4, FLDM1, FLDM2, LR1.
2. LR2 uses a sigmoid activation function instead of a unit step function, thus above a certain threshold, it gets better accuracies.
3. LR2 is also easier to realise than all the other models, such as FLDM1 or FLDM2.

Among the remaining models, FLDM1 or FLDM2, LR1, PM1 or PM4 and PM3 can be used in the given order respectively.