CS 789 – Intelligent Systems and Machine Learning

Assignment #7

1. **Problem Statement**:

The main purpose of this assignment is to implement Multi-layer Perceptron (MLP) in python/matlab to solve

a) Classification Problem or Regression Problem

2. **Languages and Libraries Used:**

Python being an open source is used for this assignment rather than MATLAB. The following libraries are used for this assignment:

a) ***NumPy***- This library is used because of its extensive features and matrix functions and ease of conversion from list to NumPy array and vice vera.

b) ***Pandas***- Pandas is used to load the data, column insertion and data selection.

c) ***train_test_split*** from ***sklearn .model_selection***- This function has been used for splitting the data into train and test subsets that makes random partitions of the two subsets.

d) ***from sklearn.metrics import confusion_matrix .*** This is used to evaluate the quality of theoutput of a classifier

3. **Implementation of Binary Classification Problem:**

For this assignment, *Iris* dataset is used to solve Classification Problem. This dataset is obtained from UCI Machine Learning Repository.

The following steps were done for implementing the assignment using this dataset.

a. Dataset

The data set is a multivariate real valued dataset which has 4 Attributes with 150 instances classifying 3 categories/classes. The Attributes of the classes are:

1) Sepal length (cm) 2) Sepal width (cm) 3) Petal length (cm) 4) Petal width (cm)

These attributes decide which of the following classes each of the input belongs to:

1) Iris Setosa 2) Iris Versicolor 3) Iris Virginica

This data is obtained as Iris. Data which was converted to Iris.data.csv for better readability in Excel.

b. Data Preprocessing

The *Iris Data* is read using pandas. The classes used for classification are **Iris-Virginica** and **Iris-Versicolor** which is represented as **class 0** and **class 1** respectively. The ***train_test_split*** function from *sklearn model selection* is used to split the dataset randomly into train data and test data in the ratio 80% and 20 %

respectively with Shuffle equals True and Random_state as 1. The **bias** term 1 is added with the data. The data is normalized using the equation:

$$z_i = \frac{(x_i - \overline{x})}{S} \quad \text{where } \overline{x} \textit{ is mean and } S \textit{ is standard deviation}$$

c. Train the binary classification model

- Initialise the Parameters

The following are the *specifications* given for the neural network model.

Number of layers	2(hidden + output)
Number of hidden units	5
Number of output units	1
Learning rate (alpha)	0.01

The experiments to choose the number of hidden units are given in the Experiments table in the next page 3

V: Intialise the random matrix of weights between -1 to +1 from input layer to hidden layer.

Dimension: (H × (D+1))

H is the number of hidden units and D is number of features

W: Initialise the random matrix of weights from hidden layer to output layer between -1 to +1.

Dimension: (K × (H+1)), K is the number of output units

The weights in hidden layer and output layer is updated using ***Backpropagation algorithm*** given in step 1 to step 4 below:

Step 1. Forward Pass to compute

a) pre-synaptic hidden layer(an):

$\mathrm{an} = \mathbf{V} \,.\, \mathrm{Xn}$

b) post-synaptic hidden layer (Zn):

$\mathrm{Zn} = \mathrm{sigmoid(an)}$, sigmoid is the *Activation function*

c) pre-synaptic output layer(bn):

$\mathrm{bn} = \mathrm{W} \,.\, \mathrm{Zn}$

d) post -synaptic output layer(Yn) :

Yn=sigmoid (bn), sigmoid is the *Activation function*

Step 2: Compute the error signals at the output layer:

$$\varsigma_{\mathrm{n}}^{\mathrm{w}} = \mathrm{Yn} - Y$$

Yn is post synaptic output and Y is true output

Step 3: Pass ς_n^w backwards to compute error signals for the hidden layer ς_n^v

$$\delta_n^v = (W \,.\, \delta_n^w) * Zn * (1 - Zn)$$

Step 4 : Compute all the gradients (ΔW,ΔV)

$$\Delta W = \delta_n^w \,.\, Zn$$

$$\Delta V = \delta_n^v \,.\, Xn$$

Step 5 : Update the Weights :

$W = W - \frac{\alpha}{N} \Delta W$
$V = V - \frac{\alpha}{N} \Delta V$

d) Evaluation Metrics

Confusion matrix from sklearn metrics is used to find the quality of the classifier. Sensitivity, specificity and accuracy are found using the confusion matrix in both train and test data. The formulas are given in the Appendix.

Experiments

The following table shows the different results obtained with the change in values of different number of hidden units.

Number of hidden units	Cost of train data	Cost of test data	Number of epochs	Accuracy in train data (%)	Accuracy in test data (%)
1	0.00250410	0.002748985	2566	100	100
2	0.00194322	0.002306850	2009	100	100
3	0.00172141	0.002186123	1801	100	100
4	0.00122291	0.001670495	1318	100	100
5	0.00116861	0.001685418	1284	100	100
10	0.00090034	0.001482250	1039	100	100
15	0.000807262	0.0014656754	943	100	100
40	0.694375905	0.6866137559	2	52.5	60

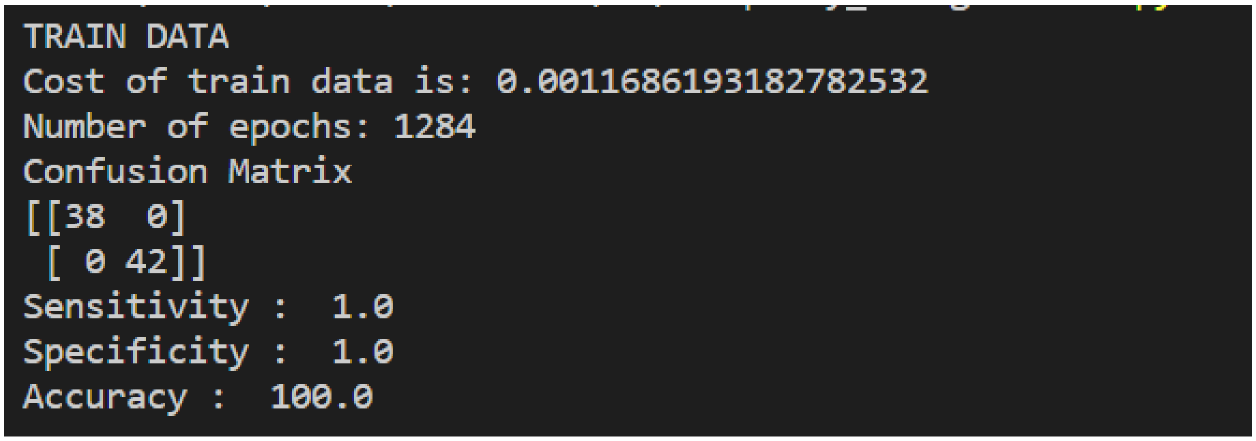
The neural network model performed well and much **faster** to train with 5 hidden units in binary classification problem with **1284** epochs when compared with other hidden units given in the table above. The number of hidden units were chosen by taking into consideration of accuracy of test data, cost of train data and cost of test data. For more than 5 hidden units in the hidden layer, the difference between cost of test data and cost of train data is high which tells that the train data is likely to be overfit.

Conclusion

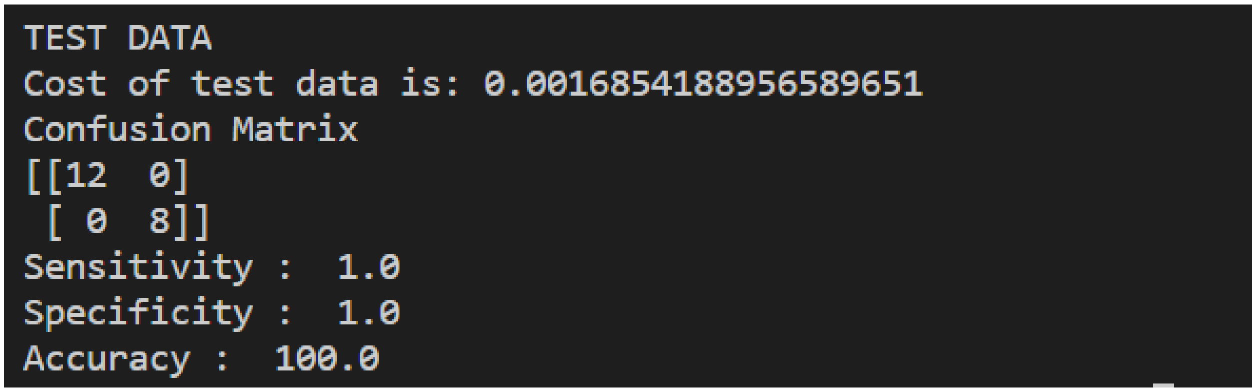
Neural network performed **same** when compared to linear discriminant analysis and logistic regression implemented in assignment 3 and assignment 5 respectively.

Note: This model is performed in batch wise. That is all training data is taken into consideration to take a single step.

RESULTS



```
TRAIN DATA
Cost of train data is: 0.0011686193182782532
Number of epochs: 1284
Confusion Matrix
[[38  0]
 [ 0 42]]
Sensitivity :  1.0
Specificity :  1.0
Accuracy :  100.0
```



```
TEST DATA
Cost of test data is: 0.0016854188956589651
Confusion Matrix
[[12  0]
 [ 0  8]]
Sensitivity :  1.0
Specificity :  1.0
Accuracy :  100.0
```

APPENDIX

Confusion Matrix:

T.N	F.P
F.N	T.P

T.N: True Negative F.P: False Positive

F.N: False Negative T.P: True Positive

Sensitivity: $\frac{TP}{TP+FN}$

Specificity: $\frac{TN}{TN+FP}$

Accuracy : $\frac{(TP+TN)}{(TP+TN+FP+FN)}$

References:

https://archive.ics.uci.edu/ml/datasets/Iris

https://medium.com/towards-artificial-intelligence/nothing-but-numpy-understanding-creating-neural-networks-with-computational-graphs-from-scratch-6299901091b0

Neural Network Class Notes CS 789, Spring 2020 at University of Nevada Las, Vegas