Visvesvaraya National Institute of Technology (VNIT), Nagpur

Machine Learning with Python (ECL 443)

Lab Report

Submitted by :
Pranay Dumbhare (BT19ECE089)
Semester 7

Submitted to :
Dr.Saugata Sinha
(Course Instructor)
Department of Electronics and Communication Engineering
VNIT Nagpur

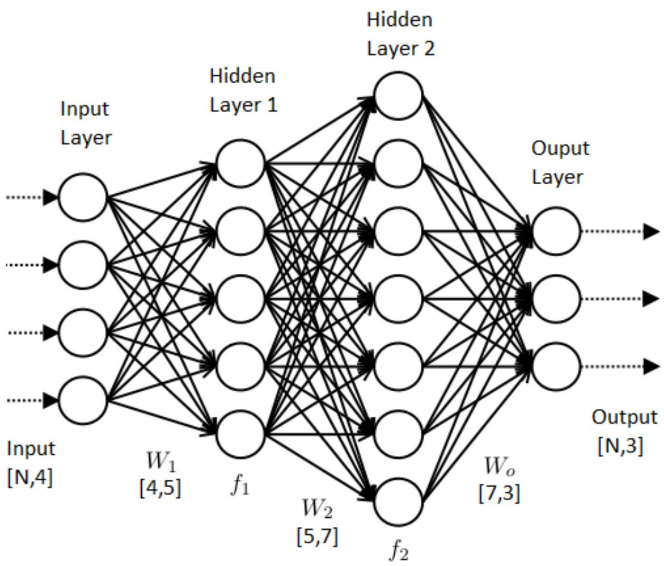
Contents

ANN

1.1 <u>Abstract</u>: Machine Learning is a computer program which is used to train our machine to predict some outputs on the basis of previous inputs. With advancements in machine learning their are many algorithms available to find patterns in the given data so that the machine can learn from this patterns to predict future output. The accuracy of the model is how correctly it predicts the output.

Human brains see the world and things in a way that computers are not able to. ANN is a type of machine learning model which is used to simulate the working of the human brain in the way that the machine is able to learn like the human brain and make decisions.

1.2 <u>Introduction</u>: An **Artificial Neural Network (ANN)** is a machine learning model that functions like neural networks inside our brain. ANNs are used widely to train machine to learn specific things by giving examples. For example, ANNs are used in applications like pattern detection or data classification. Training an ANN involves tuning the hyper parameters such as weights and bias of each neuron associated in the neural network.



ANN consists of three layers:

- Input Layer
- Hidden Layer/s

- Output Layer

Each layer consists of neurons which are interconnected. This neurons have some weights and bias associated with them and are governed by certain activation function and some learning algorithms.

1.3 Problem Statement: Using the data set (Matlab_Cancer.mat), build a classifier that can distinguish between cancer and normal patients.

1.4 Method/Procedure:

1. Import necessary packages.
2. Load the given data file is **Matlab_cancer.mat** using the user-defined function **BT19ECE089_train_test_split**.
3. After the data is loaded we spilt the dataset into training testing and validation.
4. Create a class Neural_Network which will contain all the necessary functions required.
5. We are using **sigmoid function** as the optimizer function and cost function is **Binary Cross Entropy Loss Function**.
6. We write functions for forward prop back prop , calculating confusion matrix and traing function.
7. Create object of this class Neural Network.
8. Train model for this object and the values provided.
9. Calculate accuracy with and without validation.
10. Calculate confusion matrix for range 0 to 1.
11. Calculate Specificity and sensitivity.
12. Calculate the accuracy and plot the accuracy curve.

1.5 Results/Discussion: Following are the results:

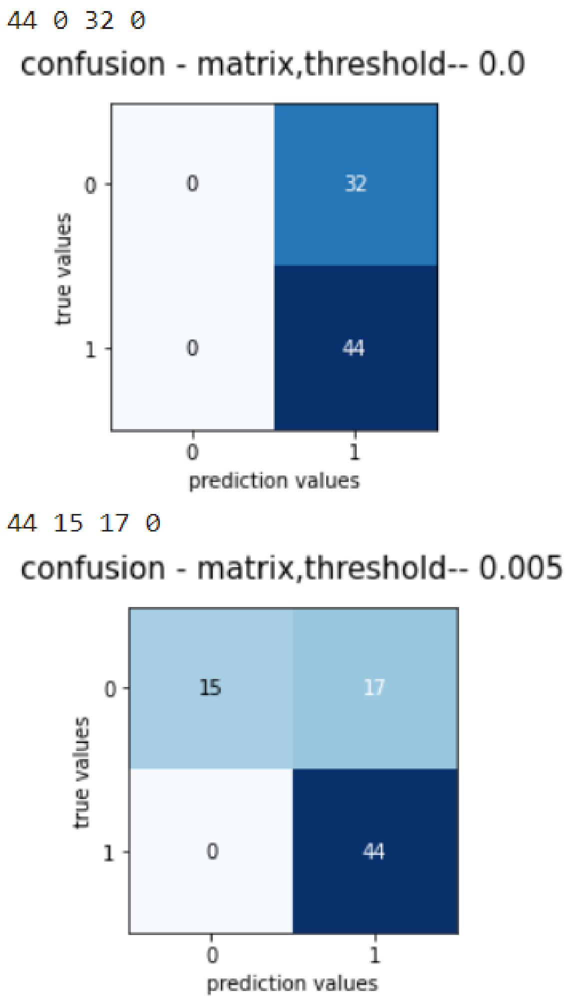


Figure 1: Confusion Matrix

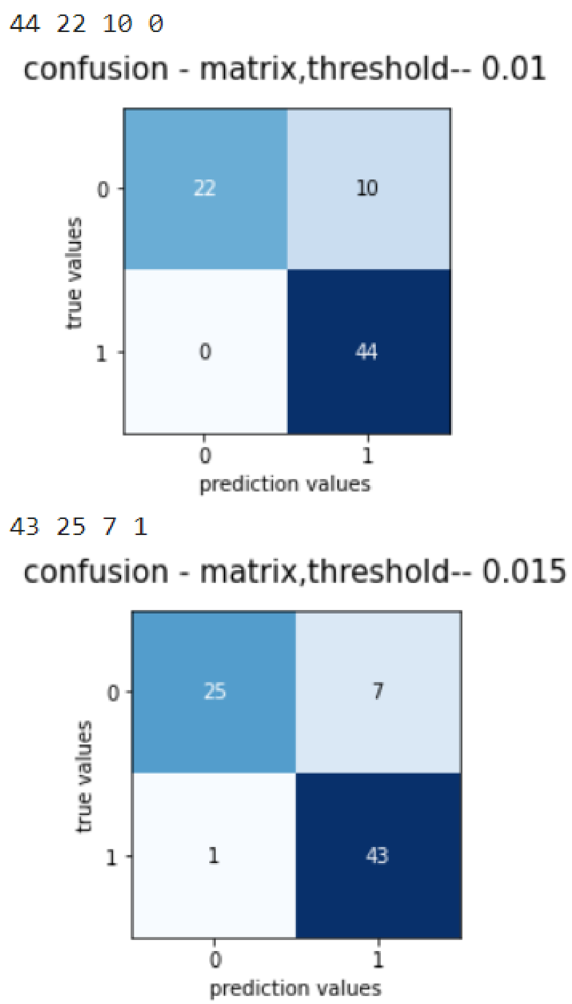


Figure 2: Confusion Matrix

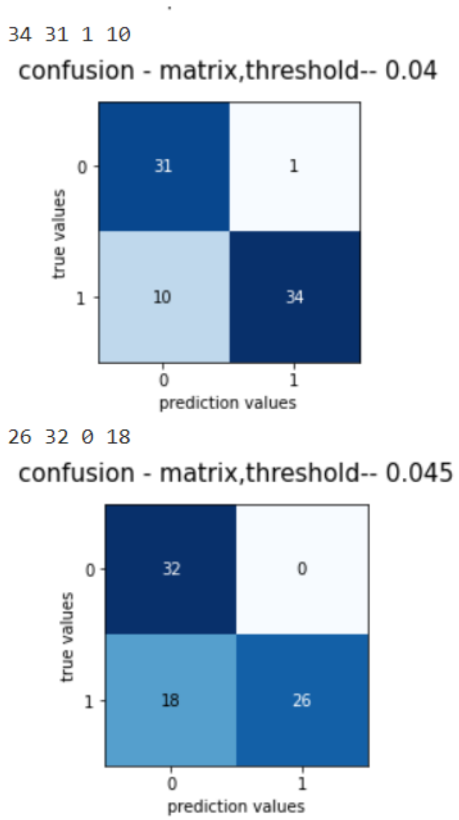


Figure 3: Confusion Matrix

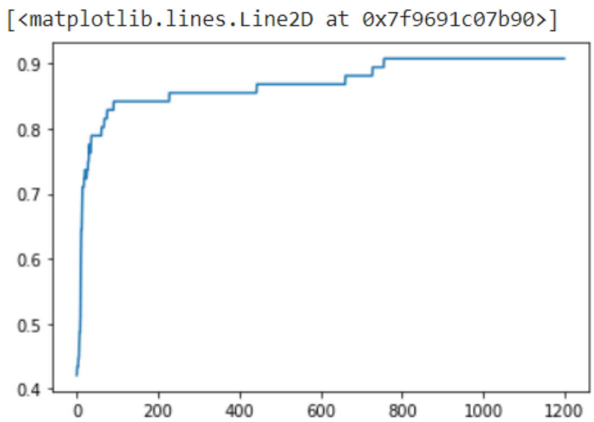


Figure 4: Accuracy Plot

```
print(specificity)
print(sensitivity)
```

```
[0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1.]
[1.         0.54545455 0.5        0.34090909 0.29545455 0.22727273
 0.22727273 0.20454545 0.15909091 0.15909091 0.09090909 0.09090909
 0.09090909 0.09090909 0.09090909 0.09090909 0.09090909 0.09090909
 0.09090909 0.09090909 0.09090909 0.09090909 0.09090909 0.09090909
 0.06818182 0.04545455 0.04545455 0.04545455 0.04545455 0.04545455
 0.04545455 0.04545455 0.04545455 0.02272727 0.02272727 0.02272727
 0.02272727 0.02272727 0.02272727 0.02272727 0.02272727 0.02272727
 0.02272727 0.02272727 0.02272727 0.02272727 0.02272727 0.02272727
 0.02272727 0.02272727 0.02272727 0.02272727 0.02272727 0.02272727
 0.02272727 0.02272727 0.02272727 0.02272727 0.02272727 0.02272727
 0.02272727 0.02272727 0.02272727 0.02272727 0.02272727 0.02272727
 0.02272727 0.02272727 0.02272727 0.02272727 0.02272727 0.02272727
 0.02272727 0.02272727 0.02272727 0.02272727 0.02272727 0.02272727
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.        ]
```

Figure 5: Specificity and Sensitivity

1.6 <u>Conclusion</u>: ANN is a machine learning algorithm used for classification, regression, and clustering problems. This is the building block for deep neural networks. It is mainly used for learning complex nonlinear hypotheses when the data set is too large and has too many features.

1.7 Appendices: The code for linear regression is given below:

```
"""BT19ECE089_L3"""
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import scipy.io
from sklearn.model_selection import train_test_split
from scipy.io import loadmat
import matplotlib.pyplot as plt
import random
import math

data= loadmat('/content/Matlab_cancer.mat')

data_x = data['x']
data_y = data['t']
trnspose = np.transpose(np.vstack([data_y, data_x]))
np.random.shuffle(trnspose)
trnspose = np.transpose(trnspose)
data_y = trnspose[0, :]
data_x = trnspose[2:, :]
split_ratio = 0.7
p = math.ceil(data_x.shape[1]*split_ratio)
print("training dataset:-",p)
train_x = data_x[:, :p//2]
train_y = data_y[:p//2]
test_x = data_x[:, p:]
test_y = data_y[p:]
val_x = data_x[:, p//2:p]
val_y = data_y[p//2:p]
np.count_nonzero(train_x)

class Neural_Network():

    def __init__(self, model, learning_rate):
        self.architecture = model
        self.neural_layers = len(model)
        self.learning_rate = learning_rate
        self.dw = []
        self.db = []
        self.biases = []
        self.weights = []
        self.cost = []
        self.test_accuracy = []
        self.train_accuracy = []
        self.validation_accuracy = []

```

```
        for i,j in zip(model[1:], model[:-1]):
            w = np.random.randn(i, j)
            b = np.random.randn(i, 1)
            dw = np.zeros([i, j])
            db = np.zeros([i, 1])
            self.dw.append(dw)
            self.db.append(db)
            self.weights.append(w)
            self.biases.append(b)
        self.activation = []
        for i in model:
            a = np.zeros(i)
            self.activation.append(a)

    def sigmoid(self, z):
        activation = 1/(1 + np.exp(-z))
        return activation
    def cost_function(self, Y):
        L = (Y * np.log(self.activation[-1]) + ...
            (1-Y)*np.log(1-self.activation[-1]))
        L = -L
        J = np.sum(L)/Y.shape[0]
        self.cost.append(J)
    def forward_propagation(self, ip):


        activation = ip
        self.activation[0] = activation
        save_num = list(range(1,self.neural_layers))

        for i,w,b in zip(save_num, self.weights, self.biases):
            z = np.matmul(w, activation) + b
            activation = self.sigmoid(z)
            self.activation[i] = activation

    def backward_propagation(self, batch_size, Y):

        dz = self.activation[-1] - Y
        dw = np.matmul(dz, self.activation[-2].T) / batch_size
        db = np.sum(dz, axis=1)/batch_size
        self.dw[-1] = dw
        self.db[-1] = db.reshape([-1, 1])

        for i in range(2, self.neural_layers):

            sis = self.activation[-i] * (1 - self.activation[-i])
            dz = np.matmul(self.weights[-i+1].T, dz)
            dz = dz * sis

```

```
            dw = np.matmul(dz, self.activation[-i-1].T) / batch_size
            db = np.sum(dz, axis=1) / batch_size

            self.dw[-i] = dw
            self.db[-i] = db.reshape([-1, 1])
            self.cost_function(Y)


    def gradient_descent(self):
        for i in range(self.neural_layers-1):
            self.weights[i] = self.weights[i] - ...
                self.learning_rate * self.dw[i]
            self.biases[i] = self.biases[i] - self.learning_rate ...
                * self.db[i]

    def accuracy(self, ip, op, threshold=0.5, confusion=False):
        activation = ip
        save_n = list(range(1,self.neural_layers))
        accuracy = 0

        for i,w,b in zip(save_n, self.weights, self.biases):
            z = np.matmul(w, activation) + b
            activation = self.sigmoid(z)

        activation = activation.reshape(-1,)
        activation[activation>threshold] = 1
        activation[activation<=threshold] = 0

        if confusion==True:
            return activation

        for i,j in enumerate(activation):
            if j==op[i]:
                accuracy = accuracy+1
        return accuracy/ip.shape[1]


    def confusion_matrix(self, ip, op, threshold_list):
        T_p = []
        T_n = []
        F_p = []
        F_n = []
        for i in threshold_list:
            activation = self.accuracy(ip, op, i, True)
            c = activation - 2 * op
            T_p.append(np.count_nonzero(c==-1))
            T_n.append(np.count_nonzero(c==0))
            F_p.append(np.count_nonzero(c==1))
            F_n.append(np.count_nonzero(c==-2))
```

```
        return T_p,T_n,F_p,F_n
    def train_function(self, ip, op, epochs, Validation_Set=None):
        batch_size = ip.shape[1]
        for i in range(epochs):
            self.forward_propagation(ip)
            self.backward_propagation(batch_size, op)
            self.gradient_descent()
            train_acc1 = self.accuracy(ip, op)
            print('accuracy-\t', train_acc1*100, '%')
            self.train_accuracy.append(train_acc1)
            if Validation_Set!=None:
                test_acc1 = self.accuracy(Validation_Set[0], ...
                    Validation_Set[1])
                self.test_accuracy.append(test_acc1)

ANN_obj = Neural_Network([100, 32, 1], 3e-2)

threshold_list = np.arange(0, 1, 0.005)
T_p,T_n,F_p,F_n = ANN_obj.confusion_matrix(train_x, train_y, ...
    threshold_list)
T_p = np.array(T_p)
T_n = np.array(T_n)
F_p = np.array(F_p)
F_n = np.array(F_n)

#without validation
ANN_obj.neural_layers
ANN_obj.train_function(train_x, train_y, 600)

#with validation
ANN_obj.train_function(train_x, train_y, 600, [val_x, val_y])

specificity = T_n/(F_p+T_n)
sensitivity = T_p/(T_p+F_n)
f_r = 1-specificity

print(specificity)
print(sensitivity)

from mlxtend.plotting import plot_confusion_matrix
threshold_a = np.arange(0, 1, 0.1)
Tp,Tn,Fp,Fn = ANN_obj.confusion_matrix(train_x, train_y, ...
    threshold_a)
Tp = np.array(Tp)
Tn = np.array(Tn)
Fp = np.array(Fp)
Fn = np.array(Fn)
for i,j,k,l,threshold_value in zip(Tp, Tn, Fp, Fn, threshold_list):
    print(i,j,k,l)
```

```
    confusion_matrix = np.array([[j, k], [l, i]])
    fig, ax = plot_confusion_matrix(conf_mat=confusion_matrix, ...
        figsize=(3, 3), cmap=plt.cm.Blues)
    plt.xlabel('prediction values')
    plt.ylabel('true values')
    plt.title('confusion - matrix,threshold-- ...
        {}'.format(round(threshold_value,3)), fontsize=15)
    plt.show()

plt.plot(ANN_obj.train_accuracy)

ANN_obj.accuracy(test_x, test_y)
```