LAB REPORT 2

November 10, 2020

Submitted by:

Kazi Amit Hasan
Roll: 1503089
Department of Computer Science & Engineering
Rajshahi University of Engineering & Technology

November 10, 2020

0.1 Title

Implementation of **Single Layer Perceptron Learning** algorithms.

0.2 Objectives

1. Learning the concept of perceptron learning algorithm.
2. Learning the one layer method.
3. Learning the adjustment of weights along with class identification.

0.3 Methodology

The Perceptron and its learning algorithm pioneered the research in neurocomputing. the perceptron is an algorithm for supervised learning of binary classifiers. A binary classifier is a function which can decide whether or not an input, represented by a vector of numbers, belongs to some specific class. It is a type of linear classifier, i.e. a classification algorithm that makes its predictions based on a linear predictor function combining a set of weights with the feature vector. Learning goes by calculating the prediction of the perceptron.

Basic Neuron equation:

$$\hat{y} = f\left(\vec{w} \cdot \vec{x} + b\right) = f\left(w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + b\right)$$

After that, we update the weights and the bias using as:

$$\hat{w_i} = w_i + \alpha(y - \hat{y})x_i\,,\ i = 1, \ldots, n;$$
$$\hat{b} = b + \alpha(y - \hat{y})\,.$$

0.4 Implementation

The steps of implementing the single layer perceptron learning algorithm is given below:

- Importing all the necessary libraries.
- Defining the main single layer perceptron function

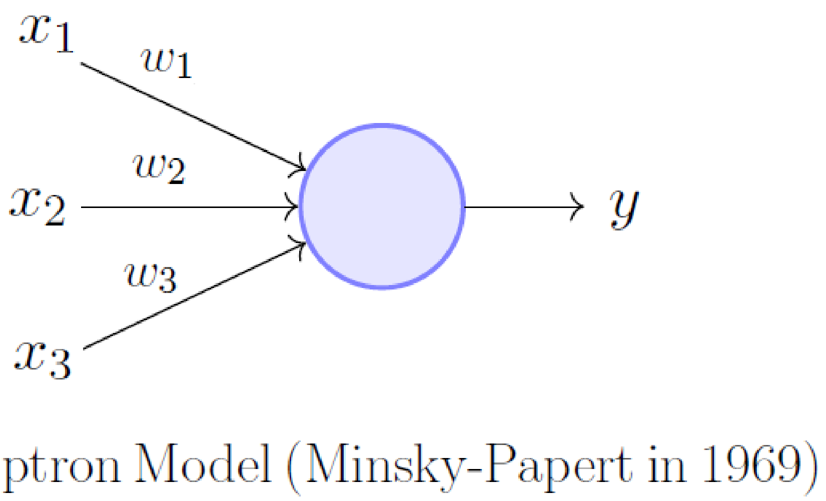

Figure 1: Perceptron model

- Creating dataset.
- Showing essential visuals of dataset.
- Tweaking the inputs and parameters.
- Train part.
- Final results.

0.5 Code

```
#importing all the libraries

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# The SLP function
class SingleLayerPerceptron:
    def __init__(self, my_weights, my_bias, learningRate=0.05):
        self.weights = my_weights
        self.bias = my_bias
        self.learningRate = learningRate

    def activation(self, net):
        answer = 1 if net > 0 else 0
        return answer

    def neuron(self, inputs):
        neuronArchitecture = np.dot(self.weights, inputs) + self.bias
        return neuronArchitecture

    def neuron_propagate(self, inputs):
        processing = self.neuron(inputs)
        return self.activation(processing)

    def training(self, inputs, output):
        output_prev = self.neuron_propagate(inputs)
        self.weights = [W + X * self.learningRate * (output - output_prev)
                       for (W, X) in zip(self.weights, inputs)]
        self.bias += self.learningRate * (output - output_prev)
        error_calculation = np.abs(output_prev - output)
        return error_calculations

#Creating dataset

data = pd.DataFrame(columns=('x1', 'x2'), data=np.random.uniform(size=(600,2)))
data.head()

#Showing the dataset

def show_dataset(data, ax):
    data[data.y==1].plot(kind='scatter', ax=ax, x='x1', y='x2', color='blue')
    data[data.y==0].plot(kind='scatter', ax=ax, x='x1', y='x2', color='red')
    plt.grid()
    plt.title(' My Dataset')
    ax.set_xlim(-0.1,1.1)
    ax.set_ylim(-0.1,1.1)

def testing(inputs):
    answer = int(np.sum(inputs) > 1)
    return answer

data['y'] = data.apply(testing, axis=1)

#Showing the dataset

fig = plt.figure(figsize=(10,10))
show_dataset(data, fig.gca())


# Giving the inputs

InitialWeights = [0.1, 0.1]
InitialBias = 0.01
LearningRate = 0.1
SLperceptron = SingleLayerPerceptron(InitialWeights,
                                     InitialBias,
                                     LearningRate)

# Train part

import random, itertools

def showAll(perceptron, data, threshold, ax=None):
    if ax==None:
        fig = plt.figure(figsize=(5,4))
        ax = fig.gca()

    show_dataset(data, ax)
    show_threshold(perceptron, ax)
    title = ’training={}’.format(threshold + 1)
    ax.set_title(title)

def trainingData(SinglePerceptron, inputs):
    count = 0
    for i, line in inputs.iterrows():
        count = count + SinglePerceptron.training(line[0:2],
                                                  line[2])

    return count

def limit(neuron, inputs):
    weights_0 = neuron.weights[0]
    weights_1 = neuron.weights[1]
    bias = neuron.bias
    threshold = -weights_0 * inputs - bias
    threshold = threshold / weights_1
    return threshold

def show_threshold(SinglePerceptron, ax):
    xlim = plt.gca().get_xlim()
    ylim = plt.gca().get_ylim()

    x2 = [limit(SinglePerceptron, x1) for x1 in xlim]

    ax.plot(xlim, x2, color="yellow")
    ax.set_xlim(-0.1,1.1)
    ax.set_ylim(-0.1,1.1)

f, axarr = plt.subplots(3, 4, sharex=True, sharey=True, figsize=(12,12))
axs = list(itertools.chain.from_iterable(axarr))
until = 12
for interaction in range(until):
    showAll(SLperceptron, data, interaction, ax=axs[interaction])
    trainingData(SLperceptron, data)
```

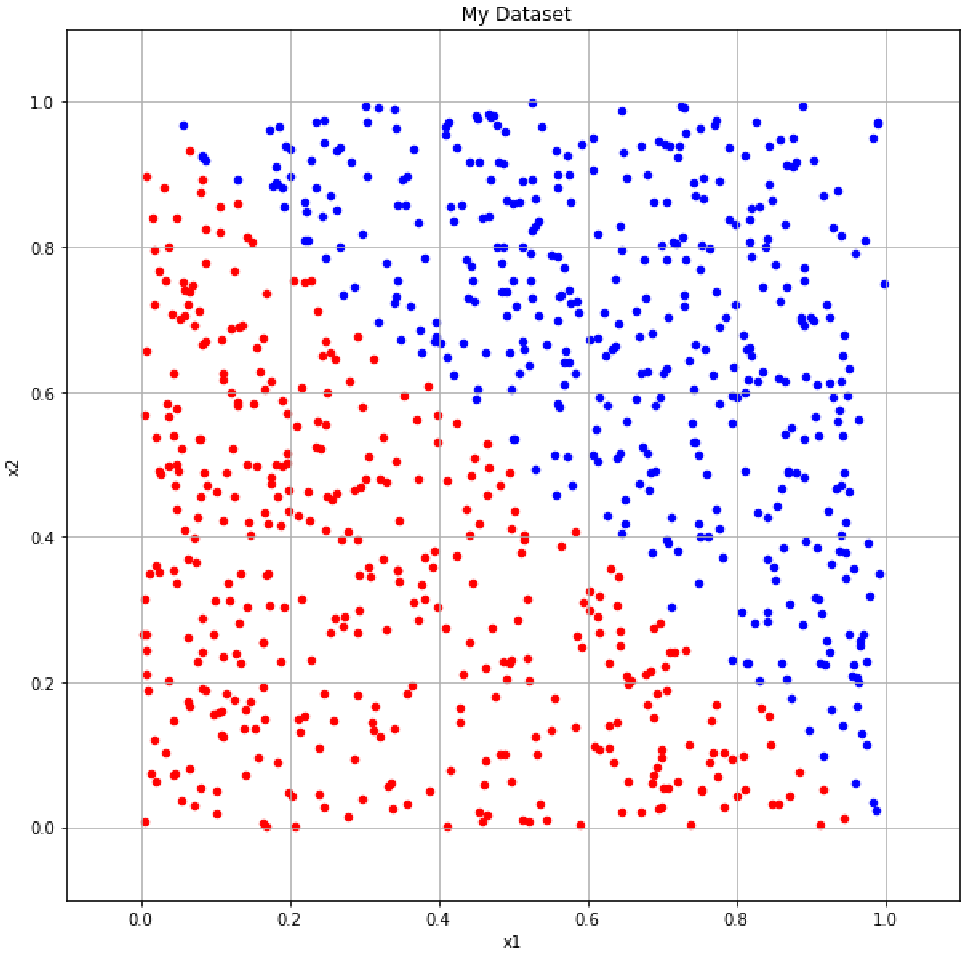

Figure 2: Generated data points

0.6 Results

The generated data points are shown in Fig. 2. In this lab, the dataset was generated randomly. For the classification part, the dataset has two classes. One is red and another is blue.

In the coding section, the initial weights, bias, learning rates were set to 0.1, 0.1, 0.01, 0.1 respectively. The results are shown in Fig. 3.

0.7 Performance Analysis

We trained the dataset with 12 epochs. In Fig. 3, the classification started with very poor outcomes. But the results improved significantly from second

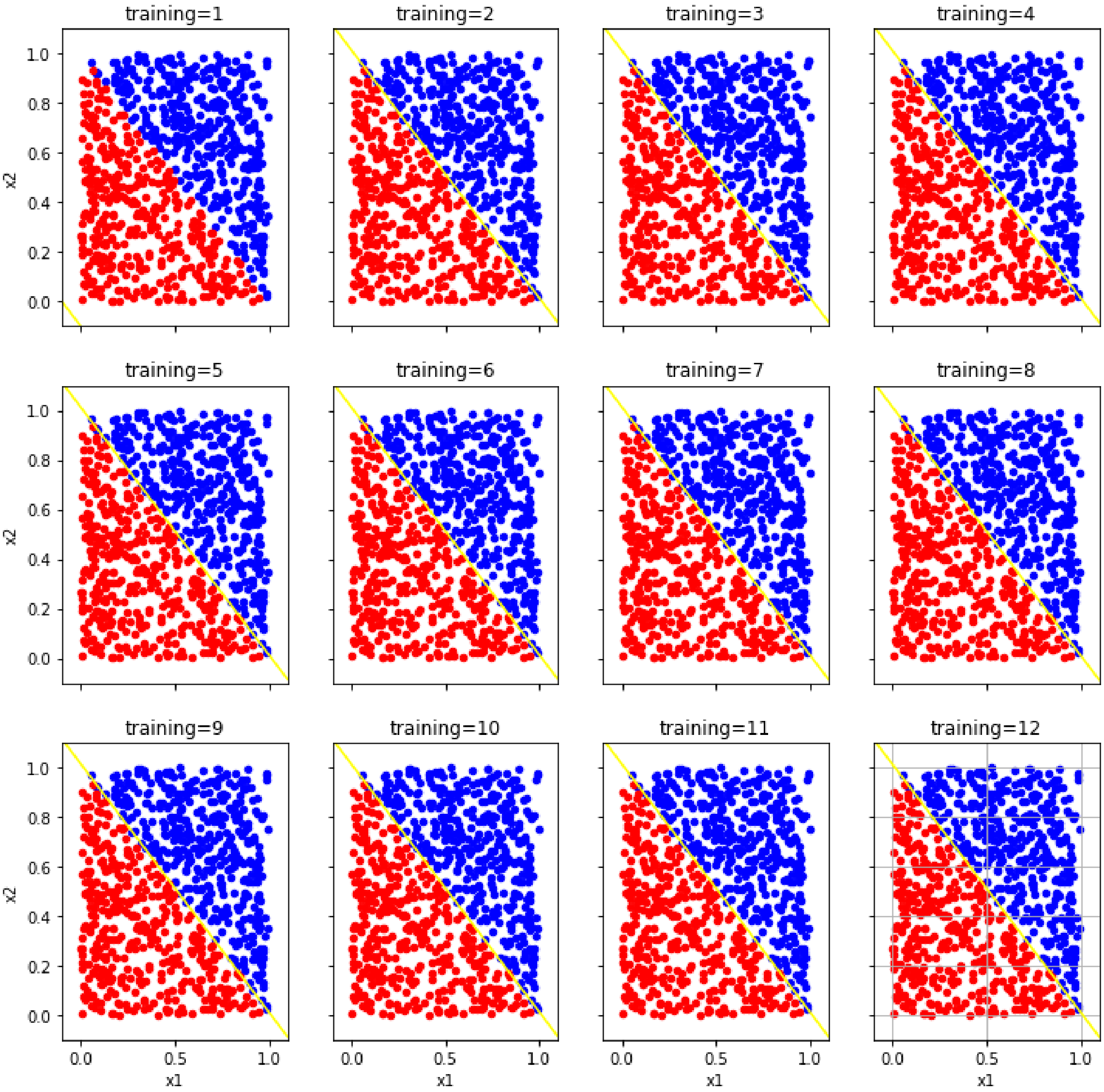

Figure 3: Output results

training. After finishing all 12 epochs, the data points are perfectly classified.

In order to analyze the performance, the generated dataset was trained with different weights, learning rate, bias, epochs.

The dataset was trained with 8, 10, 12 epochs respectively. The overall accurate performance was achieved with higher epochs.

0.8 Conclusion

Single layer Perceptrons can learn only linearly separable patterns. A Perceptron is a neural network unit that does certain computations to detect features or business intelligence in the input data. It is a function that maps its input “x,” which is multiplied by the learned weight coefficient, and generates an output value ”f(x). In this lab, I tried to implement the single layer perceptron learning algorithm without using any automated library functions All the codes and neccesary files are available in my github profile. The codes will be publicly available after my finals grades. My Github profile: https://github.com/AmitHasanShuvo/