TRIBHUVAN UNIVERSITY

INSTITUTE OF SCIENCE AND TECHNOLOGY

Central Department of Computer Science and Information Technology

Kirtipur, Kathmandu

Classroom Assignment 1

Dataset Visualization, Linear and Logistic Regression and Perceptron Learning Algorithm

Submitted by:
Name: Brihat Ratna Bajracharya
Roll No.: 19/075

Submitted to:
Mr. Tej Bahadur Shahi
Central Department of Computer
Science and Information Technology

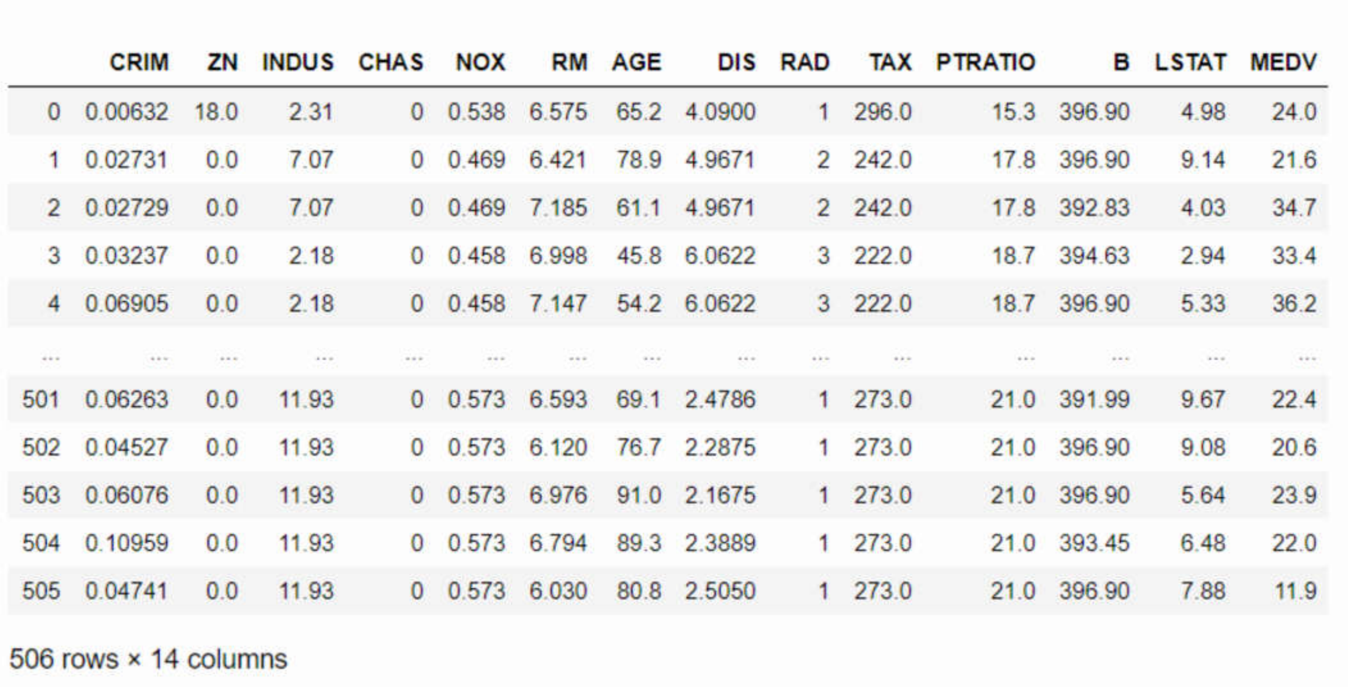
Date of submission: *2076 Mangsir 29*

Activities and Solutions

```
# Libraries imported
import pandas as pd

%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns

import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
```

1. Download the HousePrice.csv (https://www.kaggle.com/vikrishnan/boston-house-prices) and visualize it using Matplotlib or any other plotting library. Hint: Choose any one or two attribute with respect to price to visualize and further processing of data for easiness)

```
# reading csv
data = pd.read_csv('housing.csv', delim_whitespace=True)
data
```



	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90	5.33	36.2
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
501	0.06263	0.0	11.93	0	0.573	6.593	69.1	2.4786	1	273.0	21.0	391.99	9.67	22.4
502	0.04527	0.0	11.93	0	0.573	6.120	76.7	2.2875	1	273.0	21.0	396.90	9.08	20.6
503	0.06076	0.0	11.93	0	0.573	6.976	91.0	2.1675	1	273.0	21.0	396.90	5.64	23.9
504	0.10959	0.0	11.93	0	0.573	6.794	89.3	2.3889	1	273.0	21.0	393.45	6.48	22.0
505	0.04741	0.0	11.93	0	0.573	6.030	80.8	2.5050	1	273.0	21.0	396.90	7.88	11.9

506 rows × 14 columns

Table 1: HousePrice Dataset

```
# obtain data from certain columns (taken most correlated columns with MEDV)
lstat = data['LSTAT']
room = data['RM']
pt_ratio = data['PTRATIO']
price = data['MEDV']
```

```
# lSTAT vs MEDV plot
plt.plot(lstat, price, 'g+')
plt.xlabel('% Lower Status of Population')
plt.ylabel('Price of house (in 1000s)')
plt.show()
```

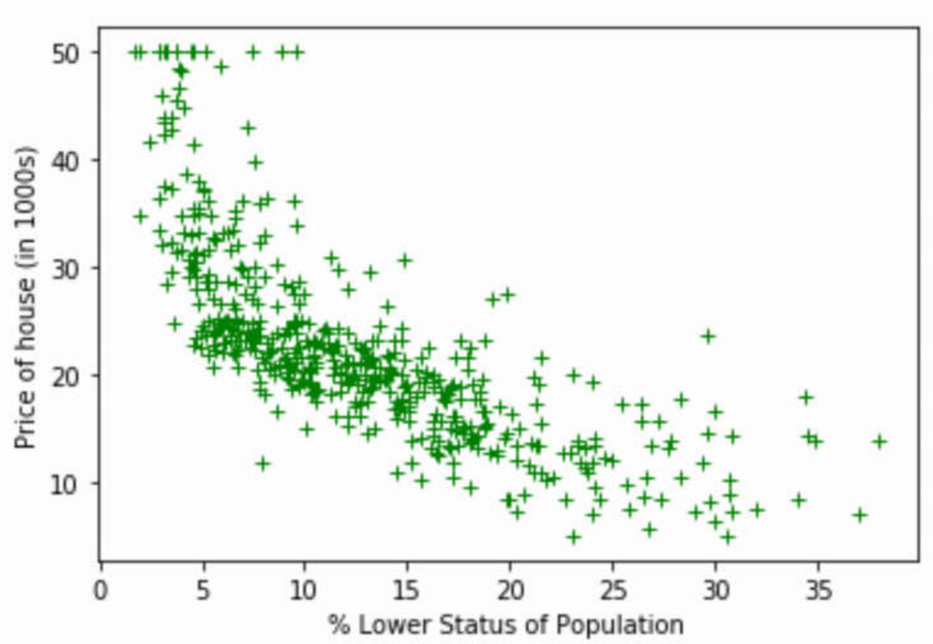


Figure 1: LSTAT vs MEDV Plot

```
# RM vs MEDV plot
plt.plot(room, price, 'b+')
plt.xlabel('Avg number of rooms (per house)')
plt.ylabel('Price of house (in 1000s)')
plt.show()
```

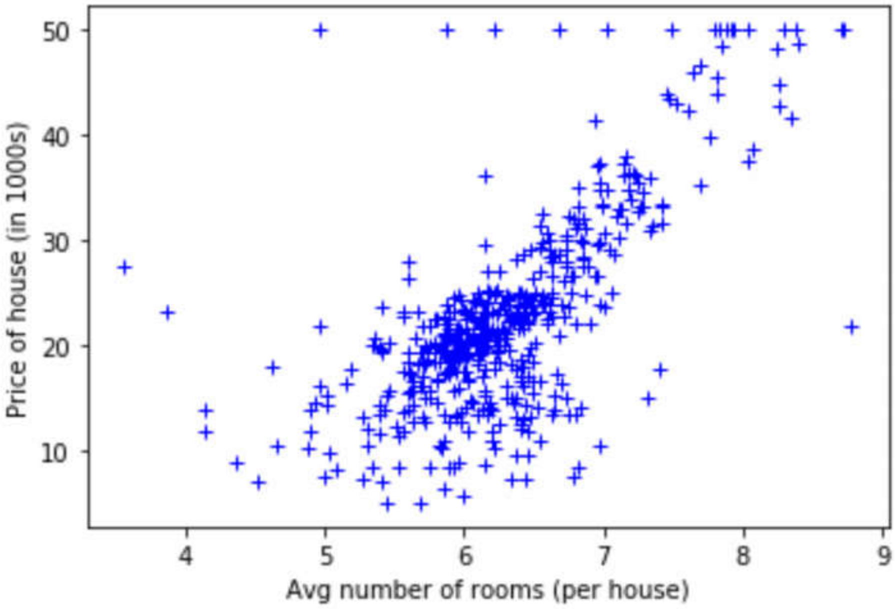


Figure 2: RM vs MEDV Plot

```
# PTRATIO vs MEDV plot
plt.plot(pt_ratio, price, 'r+')
plt.xlabel('Pupil-Teacher Ratio')
plt.ylabel('Price od house (in 1000s)')
plt.show()
```

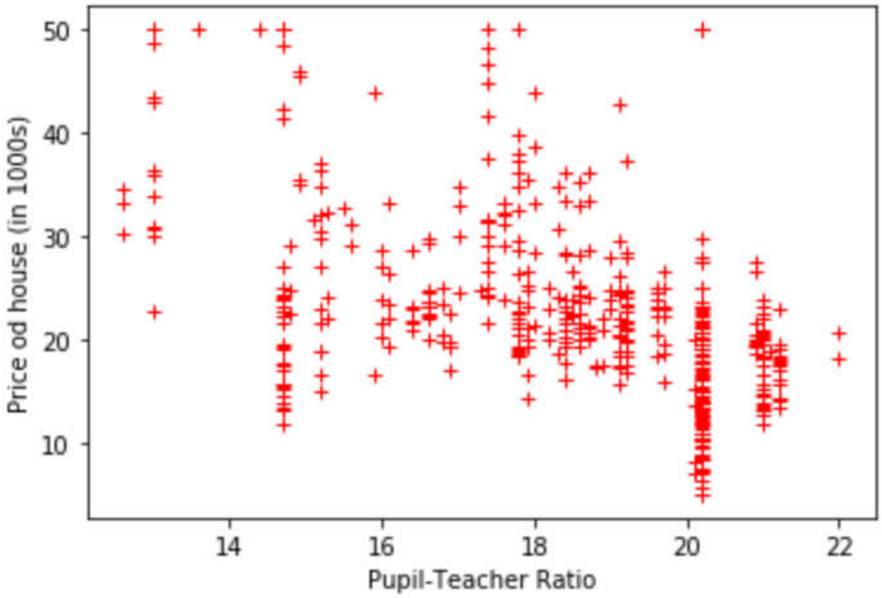


Figure 3: PTRATIO vs MEDV Plot

2. Implement the linear regression Algorithm:

```
# train test split using sklearn
' get X and Y '
Y = data[list(data.columns)[-1]]
selected_columns = ['LSTAT']
X = data[selected_columns]
X_train_lstat, X_test_lstat, Y_train_lstat, Y_test_lstat = train_test_split(X, Y,
test_size = 0.2)
X_train = X_train_room
X_test = X_test_room
Y_train = Y_train_room
Y_test = Y_test_room

# cost function
def costFunction(xVector, yVector, theta):
    inner = np.power(((xVector * theta.T) - yVector), 2)
    return np.sum(inner) / 2

# pre for linear regression
array_ones = np.ones(len(X_train))
xVector = np.column_stack((array_ones, X_train))
```

```
yVector = np.matrix(Y_train).T
theta = np.matrix(np.array([0.00, 0.00]))

# Linear Regression from scratch
learningRate = 0.0001
iterations = len(X_train)
costs = np.zeros(iterations)
m = np.size(theta,1)
newTheta = theta.T
for iter in range(iterations):
    costs[iter] = costFunction(xVector, yVector, theta)
    for i in range(len(xVector)):
        currentError = yVector[i,0] - (xVector[i,:] * newTheta)
        for j in range(m):
           term = np.multiply(np.multiply(currentError, xVector[i,j]), learningRate)
           newTheta[j,0] = newTheta[j,0] + term
           # print(i, j, newTheta)
print(newTheta)

# linear regression on actual data (plot of regression line)
t0 = float(newTheta[0])
t1 = float(newTheta[1])

plt.plot(X_train, Y_train, 'g+')
axes = plt.gca()
x_vals = np.array(axes.get_xlim())
y_vals = t0 + t1 * x_vals
plt.plot(x_vals, y_vals, 'r--')
plt.show()
```

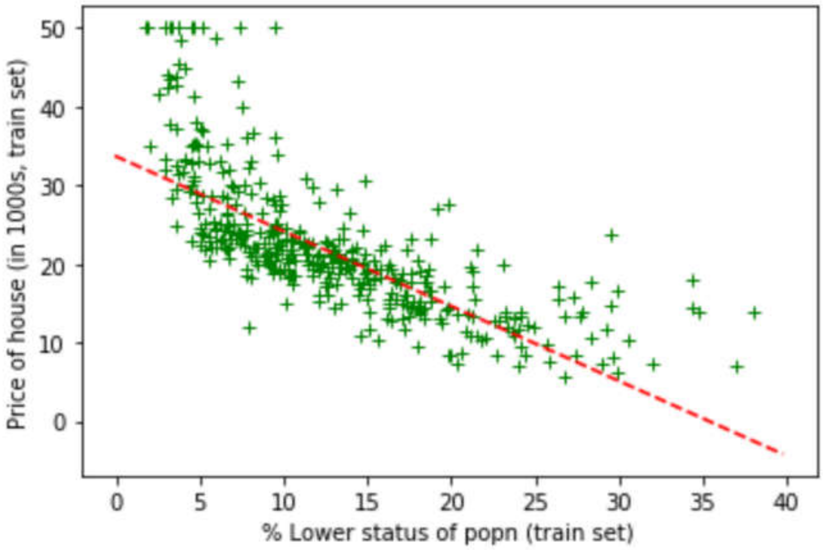


Figure 4: Regression Line for LSTAT vs MDEV Plot

3. Try to normalize the data in between [0-1] using min-max normalization and use this normalized data in above algorithm and analyze the output.

```
# normalize function
def normalize(X):
    mins = np.min(X, axis = 0)
    maxs = np.max(X, axis = 0)
    rng = maxs - mins
    norm_X = 1 - ((maxs - X)/rng)
    return norm_X

X_train_norm = normalize(X_train)
Y_train_norm = normalize(Y_train)
X_test_norm = normalize(X_test)
Y_test_norm = normalize(Y_test)
```

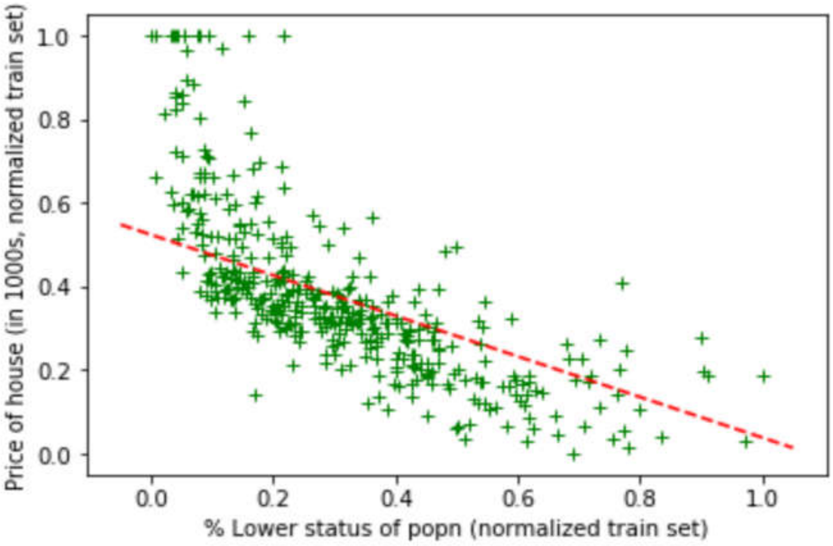
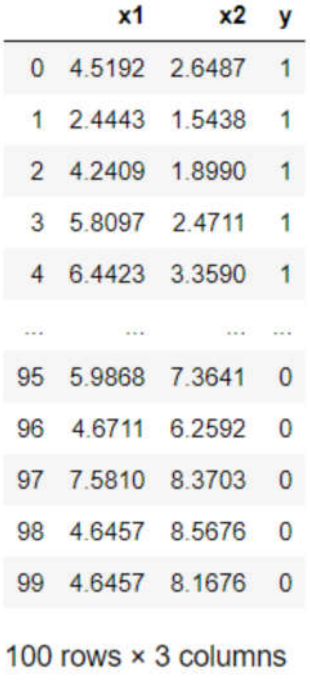


Figure 5: Regression Line for LSTAT vs MDEV Plot (Normalized)

4. Implement the Logistic Regression and list the coefficient ϴ0, ϴ1 and ϴ2 for the dataset-LogisticDataset.csv.

```
# read csv
data = pd.read_csv('Logisticdataset.csv', delimiter=',')
data

X = data[data.columns[:-1]].to_numpy()
# print(X)
y = data[data.columns[-1]].to_numpy()
# print(y)
# adds 1 to beginning of X vector
def add_ones(X):
    array_ones = np.ones((X.shape[0], 1))
    return np.concatenate((array_ones, X), axis=1)


XX = add_ones(X)


# plotting x1, x2 vs y
plt.figure(figsize=(10, 6))

plt.scatter(X[y == 0][:, 0], X[y == 0][:, 1], color='b',
label='0')
plt.scatter(X[y == 1][:, 0], X[y == 1][:, 1], color='r',
label='1')

plt.xlabel('x1')
plt.ylabel('x2')
plt.legend();
```



	x1	x2	y
0	4.5192	2.6487	1
1	2.4443	1.5438	1
2	4.2409	1.8990	1
3	5.8097	2.4711	1
4	6.4423	3.3590	1
...	...	...	...
95	5.9868	7.3641	0
96	4.6711	6.2592	0
97	7.5810	8.3703	0
98	4.6457	8.5676	0
99	4.6457	8.1676	0

100 rows × 3 columns

Table 2: Logistic Dataset

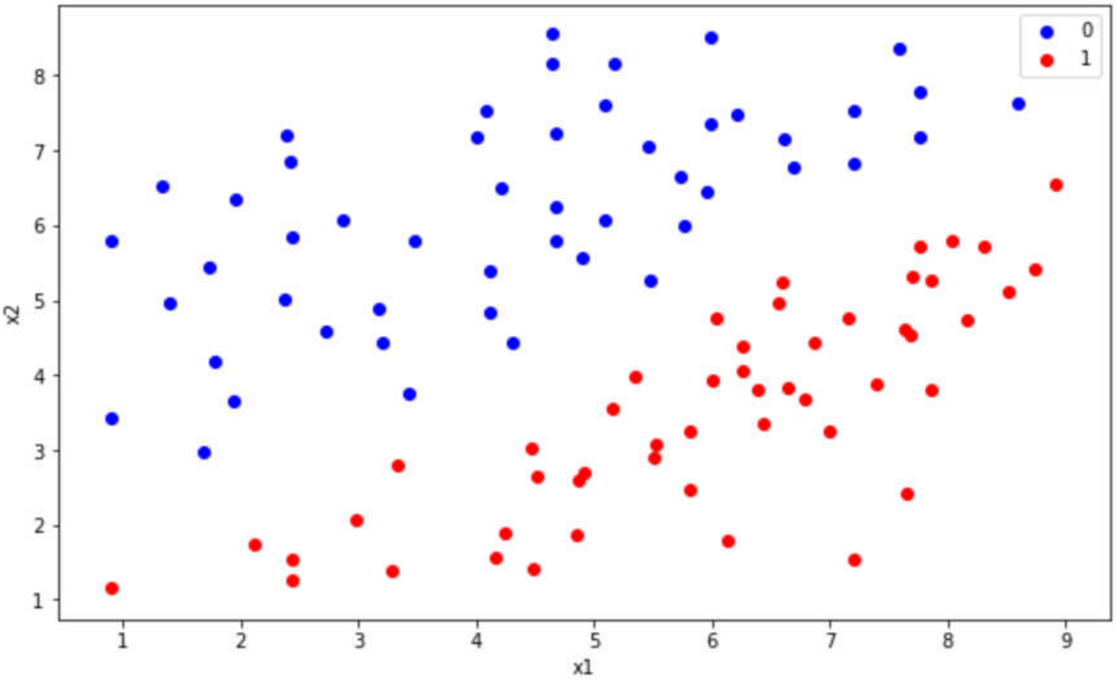


Figure 6: Scatter Plot for Logistic Dataset

```
# sigmoid function
def sigmoid(x):
    return 1 / (1 + np.exp(-x))
```

```
# loss function
def lossfunction(h, y):
    return (-y * np.log(h) - (1 - y) * np.log(1 - h)).mean()

# function that calculates logistic regression conefficients
def myLogisticRegression(X, verbose=False):
    theta = np.zeros(X.shape[1])
    learning_rate = 0.1
    iterations = len(X)

    for i in range(iterations):
        z = np.dot(X, theta)
        h = sigmoid(z)
        gradient = np.dot(X.T, (h - y)) / y.size
        theta -= learning_rate * gradient

        if(verbose==True and i % iterations == 0):
            print(f'loss: {lossfunction(h, y)} \t')
            print(theta)
    return theta

# obtain theta coefficients from logistic regression
theta_calc = myLogisticRegression(XX, False)
theta_calc
```

Output:

```
array([ 0.20254295,  1.24114545, -1.4518324 ])
```

```
# Finally solution for Question No 4
print("Coefficients calculated from logistic regression")
print("\u03B80 =", theta_calc[0])
print("\u03B81 =", theta_calc[1])
print("\u03B82 =", theta_calc[2])
```

Output:

```
Coefficients calculated from logistic regression
θ0 = 0.20254294513724383
θ1 = 1.241145447686426
θ2 = -1.4518323972814373
```

```
# Plot logistic dataset in scatter plot with boundary
plt.figure(figsize=(10, 6))
plt.scatter(X[y == 0][:, 0], X[y == 0][:, 1], color='b', label='0')
plt.scatter(X[y == 1][:, 0], X[y == 1][:, 1], color='r', label='1')
plt.xlabel('x1')
plt.ylabel('x2')
plt.legend()

# for contour line (boundary line plot)
x1_min, x1_max = X[:,0].min(), X[:,0].max(),
x2_min, x2_max = X[:,1].min(), X[:,1].max(),

xx1, xx2 = np.meshgrid(np.linspace(x1_min, x1_max), np.linspace(x2_min, x2_max))
grid = np.c_[xx1.ravel(), xx2.ravel()]
grid = add_ones(grid)
pro = sigmoid(np.dot(grid, theta_calc))
probs = pro.reshape(xx1.shape)
plt.contour(xx1, xx2, probs, [0.5], linewidths=2, colors='black');
```

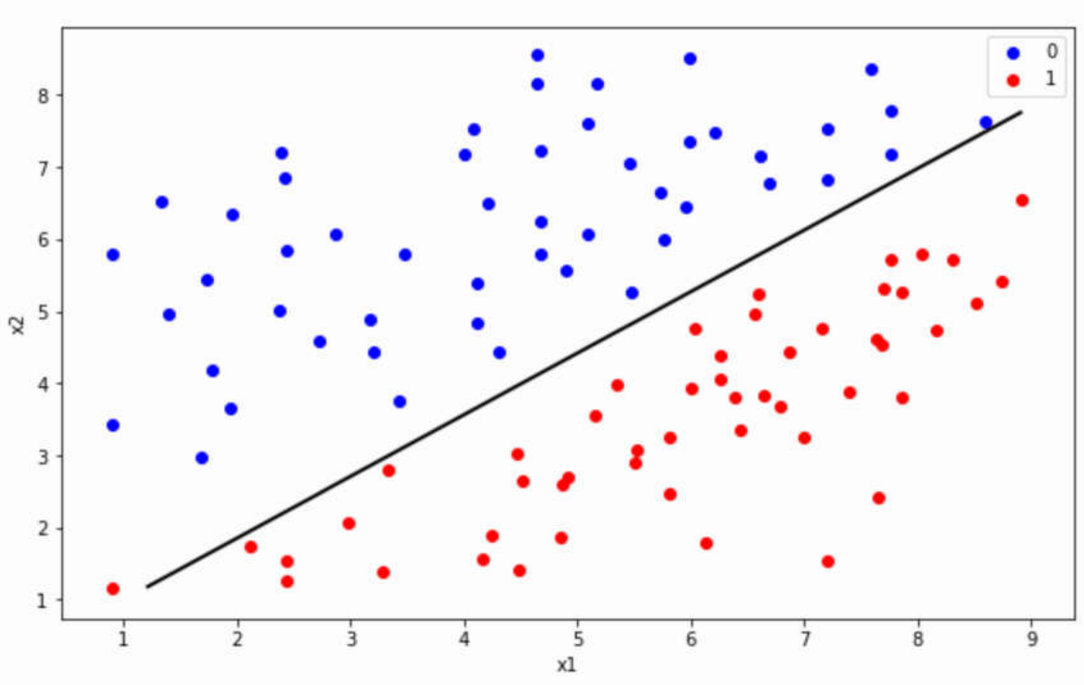


Figure 7: Scatter Plot with boundary for Logistic Dataset

5. Modify the code for question 4 so that it could behave as perceptron learning algorithm. Predict the value of y for X = (5.8097, 2.4711).

```
# modified Logistic Regression as Perceptron Learning Algorithm
def PerceptronLearning(X, verbose=False):
    theta = np.zeros(X.shape[1])
    learning_rate = 0.1
    iterations = len(X)
    h = [None] * iterations
    for i in range(iterations):
        z = np.dot(X, theta)

        for j in range(len(z)):
            h[j] = 1 if z[j] > 0 else 0

        gradient = np.dot(X.T, (h - y)) / y.size
        theta -= learning_rate * gradient

        if(verbose==True and i % iterations == 0):
            print(f'loss: {lossfunction(h, y)} \t')

    return theta

# calculate coefficients for perceptron learning algorithm
theta_calc_perc = PerceptronLearning(XX, False)

# Coefficients for Perceptron Learning Algorithm
print("Coefficients calculated from perceptron learning algorithm")
print("\u03B80 =", theta_calc_perc[0])
print("\u03B81 =", theta_calc_perc[1])
print("\u03B82 =", theta_calc_perc[2])
```

Output:

```
Coefficients calculated from perceptron learning algorithm
θ0 = 0.10500000000000007
θ1 = 0.3692555999999987
θ2 = -0.42804630000000204
```

```
# function that pre-processes input X for predicting Y using Perceptron
learning algorithm (used by predict function)
def preprocess(X):
    X.insert(0,1) # insert 1 at beginning
    # print(X)
    return np.array(X)

# prediction using perceptron learning algorithm
def threshold_function(z):
    return z > 0

# predicts output for given input X and learned theta
def predict(X, theta):
    preprocess(X)
    z = np.dot(X, theta)
    # print(z)
    h = threshold_function(z)
    # print(h)

    return 1 if h else 0

# for question 5
input_X = [5.8097,2.4711]
res = predict(input_X, theta_calc_perc)
print('Value of Y for', input_X, 'is:', res)
```

```
Output:
Value of Y for [1, 5.8097, 2.4711] is: 1
```

Source Code

Programmed in Jupyter Notebook (Python 3)

GitHub link:

https://github.com/Brihat9/ML/blob/master/Classroom_Assignment_1/brihat_ml_classroom_assignment_1.ipynb