**Neel Zadafiya (1115533)**

**Machine Learning - Home Assignment 1**

**Goal:**

To learn how to use Perceptron for simple classification problem

**Task 1:**

(5 points) Repeat the computer experiment mentioned in the class, this time, however, positioning the two moons Figure to be on the edge of separability, that is, d=0. Determine the classification error rate produced by the algorithm over 2,000 test data points.

**Answer:** HalfMoon.py

################################################################################

# #

# File name : HalfMoon.py #

# Version : Python 3.8.3rc1 64bit #

# Author : Neel Zadafiya #

# StudentId : 1115533 #

# Purpose : To implement perceptron using halfmoon dataset #

# #

################################################################################

#Import Libraries

**import** random

**import** math

**import** matplotlib**.**pyplot **as** plt

**import** numpy **as** np

#Control variables

num\_of\_inputs **=** 2

epoches **=** 10

#Function to generate half moon

**def** halfmoon**(**rad**,**width**,**d**,**n\_samp**):**

data **=** **list()**

#For upper half moon

**for** i **in** **range(int(**n\_samp**/**2**)):**

theta **=** random**.**uniform**(**0**,** math**.**pi**)**

x **=** random**.**uniform**(**rad **-** width **/** 2**,**rad **+** width **/**2**)** **\*** math**.**cos**(**theta**)**

y **=** random**.**uniform**(**rad **-** width **/** 2**,**rad **+** width **/**2**)** **\*** math**.**sin**(**theta**)**

data**.**append**([**x**,**y**,**1**])**

#For lower half moon

**for** i **in** **range(int(**n\_samp**/**2**)):**

theta **=** random**.**uniform**(-**math**.**pi**,**0**)**

x **=** random**.**uniform**(**rad **-** width **/** 2**,**rad **+** width **/**2**)** **\*** math**.**cos**(**theta**)** **+** rad

y **=** random**.**uniform**(**rad **-** width **/** 2**,**rad **+** width **/**2**)** **\*** math**.**sin**(**theta**)** **-** d

data**.**append**([**x**,**y**,-**1**])**

#Shuffle data

random**.**shuffle**(**data**)**

**return** data

#Activation function

**def** activation\_function**(**x**):**

**if** x **>=** 0**:**

result **=** 1

**else:**

result **=** **-**1

**return** result

#============================== Data Preprocess ================================

data **=** halfmoon**(**10**,**4**,**0**,**3000**)**

#Train test split

train\_x **=** **[]**

train\_y **=** **[]**

test\_x **=** **[]**

test\_y **=** **[]**

**for** i **in** data**[:**1000**]:**

train\_x**.**append**(**i**[:-**1**])**

train\_y**.**append**(**i**[-**1**])**

**for** i **in** data**[**1000**:]:**

test\_x**.**append**(**i**[:-**1**])**

test\_y**.**append**(**i**[-**1**])**

#=================================== Training ==================================

#Weight vector and learning rate

weight\_vector **=** np**.**zeros**(**num\_of\_inputs**)**

n **=** 0.1

#Train model

**for** e **in** **range(**epoches**):**

**for** i **in** **range(len(**train\_x**)):**

#Construct input vector

input\_vector **=** **[]**

**for** k **in** **range(**num\_of\_inputs**):**

input\_vector**.**append**(**train\_x**[**i**][**k**])**

input\_vector **=** np**.**array**(**input\_vector**)**

#Desired output

d **=** train\_y**[**i**]**

#Generated output

y **=** np**.**matmul**(**input\_vector**,**weight\_vector**)**

y **=** activation\_function**(**y**)**

#Update weights

weight\_vector **=** weight\_vector **+** n **\*** **(**d **-** y**)** **\*** input\_vector

#=================================== Testing ===================================

#Test model on training data

hit **=** 0

miss **=** 0

**for** i **in** **range(len(**train\_x**)):**

#Construct input vector

input\_vector **=** **[]**

**for** k **in** **range(**num\_of\_inputs**):**

input\_vector**.**append**(**train\_x**[**i**][**k**])**

input\_vector **=** np**.**array**(**input\_vector**)**

#Desired output

d **=** train\_y**[**i**]**

#Generated output

y **=** np**.**matmul**(**input\_vector**,**weight\_vector**)**

y **=** activation\_function**(**y**)**

#Compare the desired output with generated output

**if** y **==** d**:**

hit **=** hit **+** 1

**else:**

miss **=** miss **+** 1

#Print results

**print(**"Training accuracy :" **+** **str(**hit**/(**hit**+**miss**)))**

#Test model on testing data

hit **=** 0

miss **=** 0

**for** i **in** **range(len(**test\_x**)):**

#Construct input vector

input\_vector **=** **[]**

**for** k **in** **range(**num\_of\_inputs**):**

input\_vector**.**append**(**test\_x**[**i**][**k**])**

input\_vector **=** np**.**array**(**input\_vector**)**

#Desired output

d **=** test\_y**[**i**]**

#Generated output

y **=** np**.**matmul**(**input\_vector**,**weight\_vector**)**

y **=** activation\_function**(**y**)**

#Compare the desired output with generated output

**if** y **==** d**:**

hit **=** hit **+** 1

**else:**

miss **=** miss **+** 1

#Print results

**print(**"Testing accuracy :" **+** **str(**hit**/(**hit**+**miss**)))**

**print(**"Error rate :" **+** **str(**miss**/(**hit**+**miss**)))**

#Plot data points

x **=** **list()**

y **=** **list()**

**for** i **in** data**:**

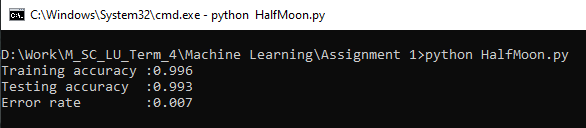
x**.**append**(**i**[**0**])**

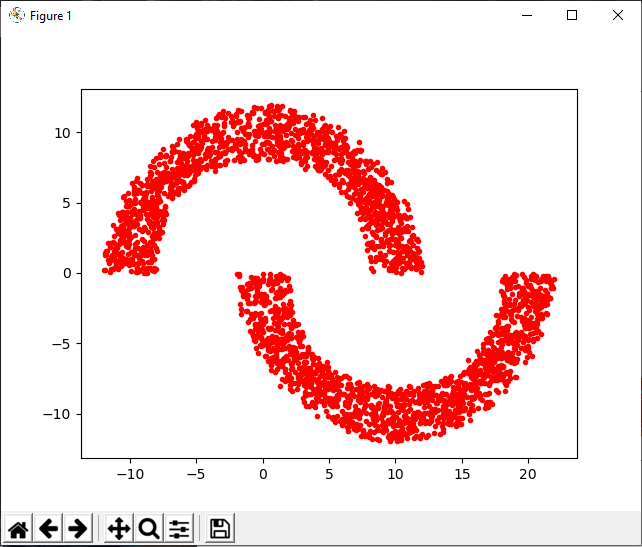
y**.**append**(**i**[**1**])**

plt**.**plot**(**x**,**y**,**'r.'**)**

plt**.**show**()**

**Output:**





**Task 2:**

(5 points) Download one of the UCI dataset, reuse your own perceptron codes to get the testing accuracy of the selected dataset.

**Answer:** UCI.py

################################################################################

# #

# File name : UCI.py #

# Version : Python 3.8.3rc1 64bit #

# Author : Neel Zadafiya #

# StudentId : 1115533 #

# Purpose : To implement perceptron using UCI (Connectionist Bench) dataset #

# #

################################################################################

#Dataset link: http://archive.ics.uci.edu/ml/datasets/connectionist+bench+(sonar,+mines+vs.+rocks)

#The label associated with each record contains the letter "R" if the object is a rock and "M" if it is a mine (metal cylinder).

#In the preprocessing part, M is converted to 1 and R is converted to -1

#Import Libraries

**import** random

**import** math

**import** matplotlib**.**pyplot **as** plt

**import** numpy **as** np

**import** csv

#Control variables

num\_of\_inputs **=** 60

epoches **=** 50

#Activation function

**def** activation\_function**(**x**):**

**if** x **>=** 0**:**

result **=** 1

**else:**

result **=** **-**1

**return** result

#============================== Data Preprocess ================================

results **=** **[]**

data **=** **[]**

#Read csv file to results

**with** **open(**"sonar.all-data"**)** **as** csvfile**:**

reader **=** csv**.**reader**(**csvfile**)**

**for** row **in** reader**:**

results**.**append**(**row**)**

#Convert data to float

**for** i **in** results**:**

temp **=** **[]**

**for** j **in** i**[:-**1**]:**

temp**.**append**(float(**j**))**

**if** i**[-**1**]** **==** 'R'**:**

temp**.**append**(-**1**)**

**else:**

temp**.**append**(**1**)**

data**.**append**(**temp**)**

#Shuffle data

random**.**shuffle**(**data**)**

#Train test split

train\_x **=** **[]**

train\_y **=** **[]**

test\_x **=** **[]**

test\_y **=** **[]**

**for** i **in** data**[:**69**]:**

train\_x**.**append**(**i**[:-**1**])**

train\_y**.**append**(**i**[-**1**])**

**for** i **in** data**[**69**:]:**

test\_x**.**append**(**i**[:-**1**])**

test\_y**.**append**(**i**[-**1**])**

#=================================== Training ==================================

#Weight vector and learning rate

weight\_vector **=** np**.**zeros**(**num\_of\_inputs**)**

n **=** 0.1

#Train model

**for** e **in** **range(**epoches**):**

**for** i **in** **range(len(**train\_x**)):**

#Construct input vector

input\_vector **=** **[]**

**for** k **in** **range(**num\_of\_inputs**):**

input\_vector**.**append**(**train\_x**[**i**][**k**])**

input\_vector **=** np**.**array**(**input\_vector**)**

#Desired output

d **=** train\_y**[**i**]**

#Generated output

y **=** np**.**matmul**(**input\_vector**,**weight\_vector**)**

y **=** activation\_function**(**y**)**

#Update weights

weight\_vector **=** weight\_vector **+** n **\*** **(**d **-** y**)** **\*** input\_vector

#=================================== Testing ===================================

#Test model on training data

hit **=** 0

miss **=** 0

**for** i **in** **range(len(**train\_x**)):**

#Construct input vector

input\_vector **=** **[]**

**for** k **in** **range(**num\_of\_inputs**):**

input\_vector**.**append**(**train\_x**[**i**][**k**])**

input\_vector **=** np**.**array**(**input\_vector**)**

#Desired output

d **=** train\_y**[**i**]**

#Generated output

y **=** np**.**matmul**(**input\_vector**,**weight\_vector**)**

y **=** activation\_function**(**y**)**

#Compare the desired output with generated output

**if** y **==** d**:**

hit **=** hit **+** 1

**else:**

miss **=** miss **+** 1

#Print results

**print(**"Training accuracy :" **+** **str(**hit**/(**hit**+**miss**)))**

#Test model on testing data

hit **=** 0

miss **=** 0

**for** i **in** **range(len(**test\_x**)):**

#Construct input vector

input\_vector **=** **[]**

**for** k **in** **range(**num\_of\_inputs**):**

input\_vector**.**append**(**test\_x**[**i**][**k**])**

input\_vector **=** np**.**array**(**input\_vector**)**

#Desired output

d **=** test\_y**[**i**]**

#Generated output

y **=** np**.**matmul**(**input\_vector**,**weight\_vector**)**

y **=** activation\_function**(**y**)**

#Compare the desired output with generated output

**if** y **==** d**:**

hit **=** hit **+** 1

**else:**

miss **=** miss **+** 1

#Print results

**print(**"Testing accuracy :" **+** **str(**hit**/(**hit**+**miss**)))**

**print(**"Error rate :" **+** **str(**miss**/(**hit**+**miss**)))**

**Output:**

