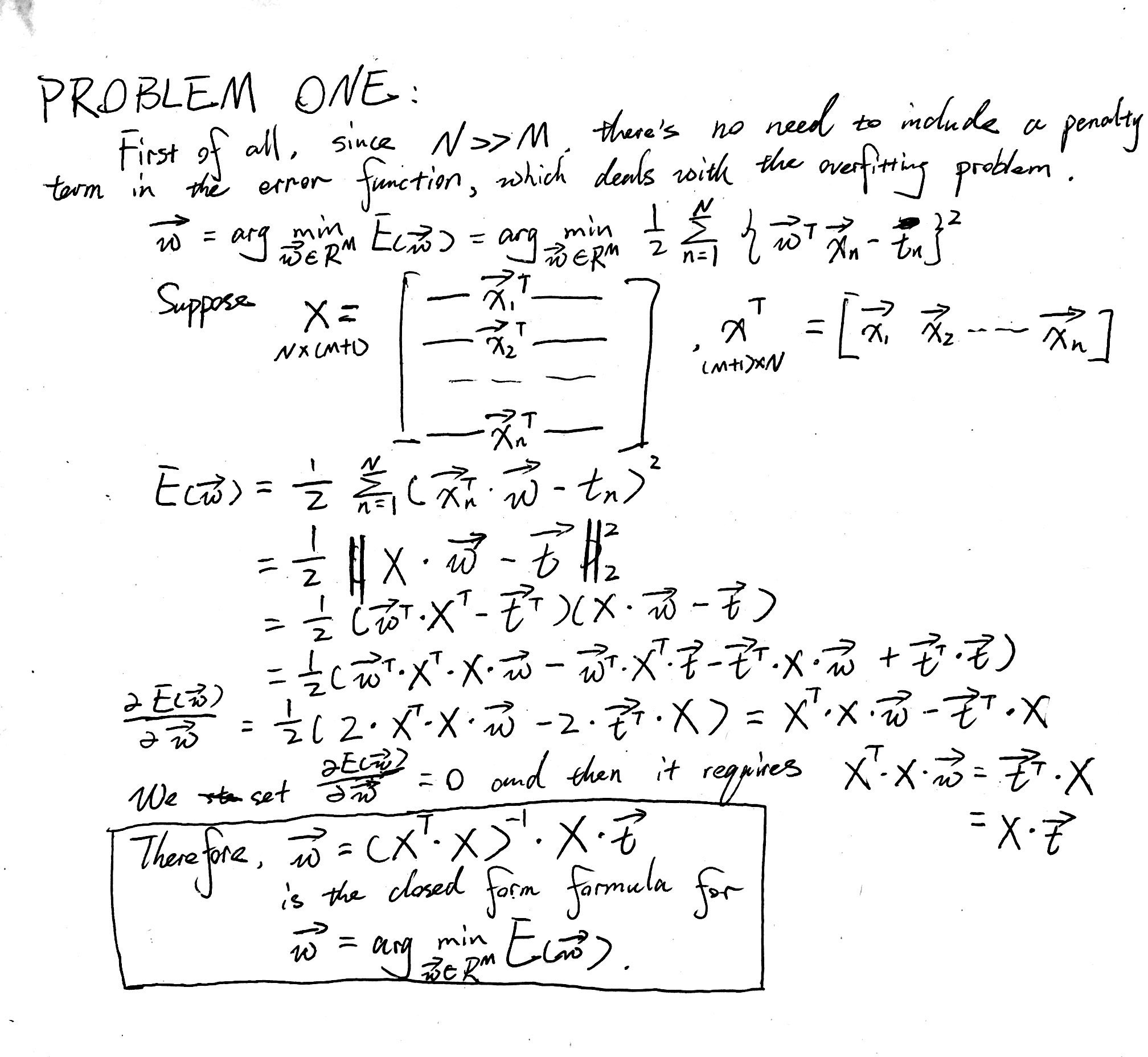
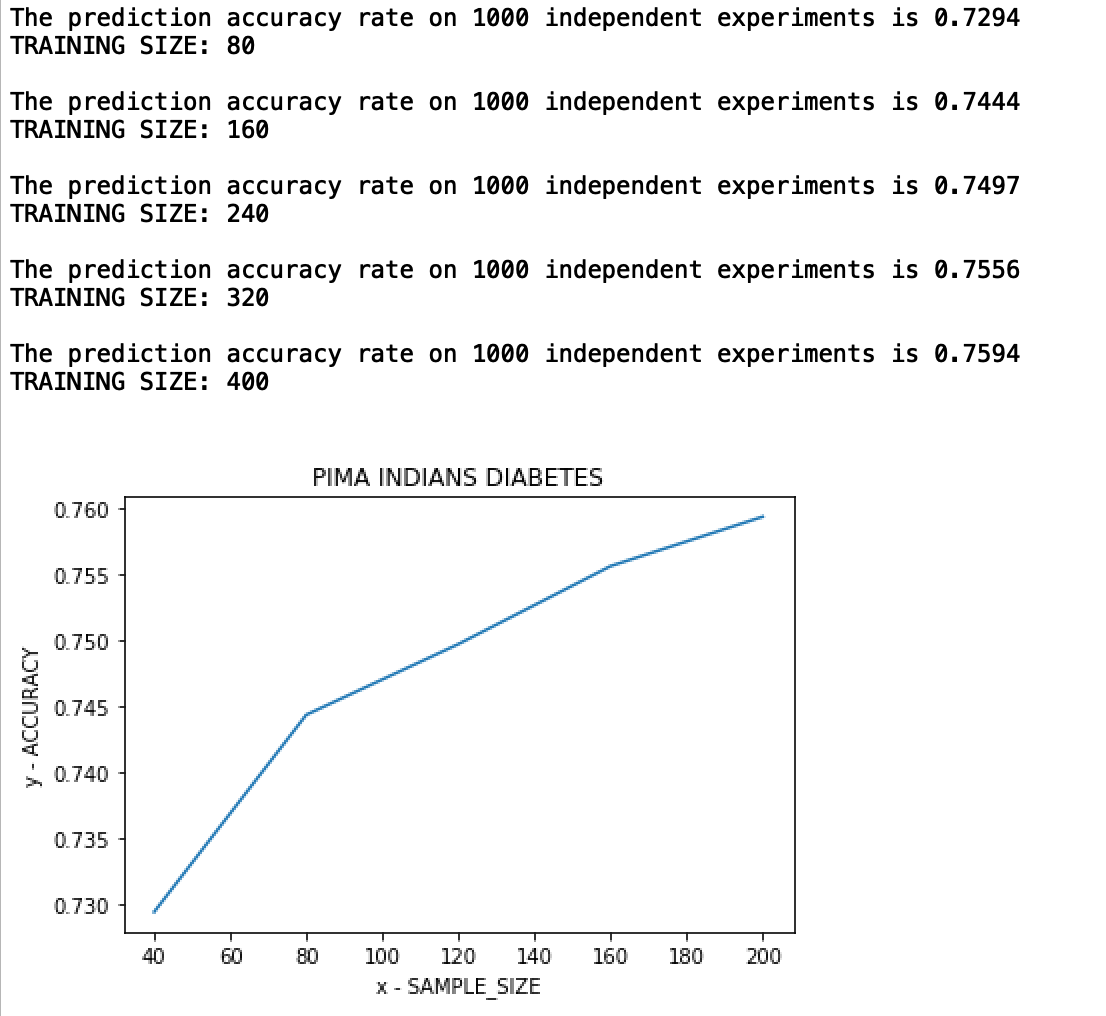
**COEN 240 MACHINE LEARNING**

**HOMEWROK ONE**

**NAME: BOSEN YANG STUDENT ID: 1589880**



PROBLEM TWO:



It is obvious from the plot that the size of training set has a positive correlation with the overall accuracy. In other words, the more training data we have, the more accurate our model will be, and vice versa. Even our initial training set size, 80 as 2 times n=40, is substantially bigger than the number of attributes. It tells us having a bigger training set can be considered as an advantage in training our model, but of course within permitted computational power.

**ATTACHMENTS**

**PROBLEM TWO CODE**

**﻿**import numpy as np

from sklearn.model\_selection import train\_test\_split

import matplotlib.pyplot as plt

import time

# LOADING AND PROCESSING OF DATA

# READ FROM FILE AND ADD BIAS

attributes = {"pregnancies", "glucose", "blood\_pressure", "bmi", "insulin\_level", "age", "attribute7", "attribute8"}

file\_path = "/Users/bosen/Library/Mobile Documents/com~apple~CloudDocs/Portal/COEN 240/Assignment/Homework1/pima-indians-diabetes.csv"

# file\_path = ""

diabetes\_raw = np.genfromtxt(file\_path, delimiter=',')

N = diabetes\_raw.shape[0] # N = total number of samples

diabetes\_plus\_bias = np.c\_[np.ones((N,1)), diabetes\_raw]

# SPLIT DIABETES AND NO\_DIABETES GROUPS BASED ON TARGET VALUE

columnIndex = 9

target\_column = diabetes\_plus\_bias[:,columnIndex]

sorted\_diabetes = diabetes\_plus\_bias[target\_column.argsort()[::-1]]

split\_result = np.split(sorted\_diabetes, np.where(np.diff(sorted\_diabetes[:,9]))[0]+1)

class\_diabetes = split\_result[0]

class\_no\_diabetes = split\_result[1]

# SPLIT OUT TARGETS FROM ATTRIBUTES

target\_d = class\_diabetes[:, 9]

class\_diabetes = class\_diabetes[:, 0:9]

num\_d = class\_diabetes.shape[0]

target\_nd = class\_no\_diabetes[:, 9]

class\_no\_diabetes = class\_no\_diabetes[:, 0:9]

num\_nd = class\_no\_diabetes.shape[0]

# TRY DIFFERENT SAMPLE SIZE

samples = []

results = []

for SAMPLE\_SIZE in range(40, 240, 40):

# VARIABLES FOR STATISTICS

COUNT = 1000

result = 0

# RUN 1000 EXPERIMENTS

for i in range(COUNT):

# MERGE TWO SUBSETS INTO FINAL TRAINING SET

X\_train\_d, X\_test\_d, t\_train\_d, t\_test\_d = \

train\_test\_split(class\_diabetes, target\_d, test\_size=(num\_d-SAMPLE\_SIZE)/num\_d, random\_state=time.time\_ns()%(2\*\*32))

X\_train\_nd, X\_test\_nd, t\_train\_nd, t\_test\_nd = \

train\_test\_split(class\_no\_diabetes, target\_nd, test\_size=(num\_nd-SAMPLE\_SIZE)/num\_nd, random\_state=time.time\_ns()%(2\*\*32))

X\_train = np.concatenate((X\_train\_d, X\_train\_nd))

X\_test = np.concatenate((X\_test\_d, X\_test\_nd))

t\_train = np.concatenate((t\_train\_d, t\_train\_nd)).reshape(-1, 1)

t\_test = np.concatenate((t\_test\_d, t\_test\_nd)).reshape(-1, 1)

# CALCULATE, ACCELERATED BY REPLACING TENSORFLOW WITH NUMPY

temp = X\_train.transpose()

w\_val = np.linalg.inv(temp.dot(X\_train)).dot(temp).dot(t\_train)

y\_test\_val = np.rint(X\_test.dot(w\_val))

num\_test = X\_test.shape[0]

num\_match = np.count\_nonzero(np.equal(y\_test\_val, t\_test))

result = result + num\_match/num\_test

# RETURN THE AVERAGE RESULT OF THE 1000 EXPERIMENTS

result\_averaged = result/COUNT

results.append(result\_averaged)

samples.append(SAMPLE\_SIZE)

print("The prediction accuracy rate on %d independent experiments is %.4f" % (COUNT, result\_averaged))

print("TRAINING SIZE: %d\n" % (SAMPLE\_SIZE\*2))

# PLOTTING

plt.plot(samples, results)

plt.xlabel('x - SAMPLE\_SIZE')

plt.ylabel('y - ACCURACY')

plt.title('PIMA INDIANS DIABETES')

plt.show()