COEN 240 MACHINE LEARNING

HOMEWROK ONE

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

PROBLEM ONE:

First of all, since $N >> M$, there's no need to include a penalty term in the error function, which deals with the overfitting problem.

$$\vec{w} = \arg\min_{\vec{w} \in R^M} E(\vec{w}) = \arg\min_{\vec{w} \in R^M} \frac{1}{2} \sum_{n=1}^{N} \{ \vec{w}^T \vec{x}_n - t_n \}^2$$

Suppose $\underset{N \times (M+1)}{X} = \begin{bmatrix} — \vec{x}_1^T — \\ — \vec{x}_2^T — \\ — — — \\ — \vec{x}_n^T — \end{bmatrix}$, $\underset{(M+1) \times N}{X^T} = [\vec{x}_1 \ \vec{x}_2 \cdots \vec{x}_n]$

$$E(\vec{w}) = \frac{1}{2} \sum_{n=1}^{N} (\vec{x}_n^T \cdot \vec{w} - t_n)^2$$

$$= \frac{1}{2} \| X \cdot \vec{w} - \vec{t} \|_2^2$$

$$= \frac{1}{2} (\vec{w}^T \cdot X^T - \vec{t}^T)(X \cdot \vec{w} - \vec{t})$$

$$= \frac{1}{2} (\vec{w}^T \cdot X^T \cdot X \cdot \vec{w} - \vec{w}^T \cdot X^T \cdot \vec{t} - \vec{t}^T \cdot X \cdot \vec{w} + \vec{t}^T \cdot \vec{t})$$

$$\frac{\partial E(\vec{w})}{\partial \vec{w}} = \frac{1}{2} (2 \cdot X^T \cdot X \cdot \vec{w} - 2 \cdot \vec{t}^T \cdot X) = X^T \cdot X \cdot \vec{w} - \vec{t}^T \cdot X$$

We ~~sta~~ set $\frac{\partial E(\vec{w})}{\partial \vec{w}} = 0$ and then it requires $X^T \cdot X \cdot \vec{w} = \vec{t}^T \cdot X$

$$= X \cdot \vec{t}$$

$$\boxed{\text{Therefore, } \vec{w} = (X^T \cdot X)^{-1} \cdot X \cdot \vec{t} \text{ is the closed form formula for } \vec{w} = \arg\min_{\vec{w} \in R^M} E(\vec{w}).}$$

PROBLEM TWO:

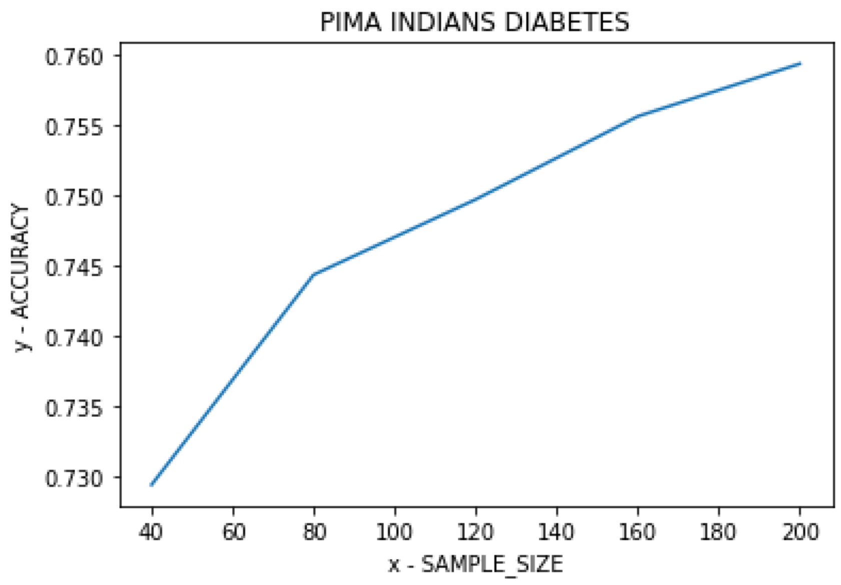
```
The prediction accuracy rate on 1000 independent experiments is 0.7294
TRAINING SIZE: 80

The prediction accuracy rate on 1000 independent experiments is 0.7444
TRAINING SIZE: 160

The prediction accuracy rate on 1000 independent experiments is 0.7497
TRAINING SIZE: 240

The prediction accuracy rate on 1000 independent experiments is 0.7556
TRAINING SIZE: 320

The prediction accuracy rate on 1000 independent experiments is 0.7594
TRAINING SIZE: 400
```



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()
```