Hetsvi Navnitlal
Programming Assignment

1. Training error:

	Error
1	0.04128440366972477
2	0.04036697247706422
3	0.02110091743119266
4	0.01834862385321101
5	0.026605504587155965

Test error:

	Error
1	0.5013262599469496
2	0.4588859416445623
3	0.4986737400530504
4	0.4960212201591512
5	0.4535809018567639

2. The words with highest coordinate:
   First – File
   Second – Program
   Third – Line

   The words with lowest values:
   First – He
   Second – Team
   Third – Game

   Yes, these match because the highest one matches with a class which is a computer group so terms like file, program and line are popular in computer. The one in the

lowest ones are with the baseball group so terms like he, team and game go with baseball.

3. Exponential function:

Training error:

	Error
1	0.14862385321100918
2	0.015596330275229359
3	0.01651376146788991
4	0.003669724770642202
5	0.0

Test error:

	Error
1	0.19363395225464192
2	0.050397877984084884
3	0.058355437665782495
4	0.04509283819628647
5	0.03713527851458886

Polynomial function:

Training error:

	Error
1	0.044954128440366975
2	0.027522935779816515
3	0.01834862385321101
4	0.013761467889908258
5	0.006422018348623854

Test error:

	Error
1	0.0636604774535809
2	0.050397877984084884
3	0.03183023872679045
4	0.03978779840848806
5	0.042440318302387266

Code:

```
# imports
import pandas as pd
import numpy as np
from collections import Counter
import random

# loading texts
text = np.loadtxt('./pa3train.txt')
other = np.loadtxt('./pa3test.txt')
words = open('./pa3dictionary.txt')
word = words.readlines()

# only features of files
training = np.delete(text,len(text[0])-1, 1)
others = np.delete(other,len(other[0])-1, 1)

#imports
import copy
import sys

# part 1 class
class Perceptron():
   def __init__(self):
      self.weights = np.zeros(len(training[0]))

   def perceptron_train(self,train_x, train_y):   # training
      labels = np.where(train_y[:,-1]==1,1,-1)    # if 1 return 1 else -1
      for i in range(len(text)):
```

```
            if labels[i]* np.dot(self.weights, train_x[i]) <= 0:
                self.weights = self.weights + labels[i]*train_x[i]
            else:
                self.weights = self.weights

    def p(self, inputs):                # predicting
        output = []
        for i in range(len(inputs)):
            if np.dot(self.weights, inputs[i]) <= 0:
                output.append(-1)
            else:
                output.append(1)
        return output

    def words(self, word):              # finding the words from weights
        weight = copy.deepcopy(self.weights)

        first_max = np.argmax(weight)
        weight[first_max] = -sys.maxsize
        second_max = np.argmax(weight)
        weight[second_max] = -sys.maxsize
        third_max = np.argmax(weight)

        w= copy.deepcopy(self.weights)
        first_min = np.argmin(w)
        w[first_min] = sys.maxsize
        second_min = np.argmin(w)
        w[second_min] = sys.maxsize
        third_min = np.argmin(w)

        print(word[first_max], word[second_max], word[third_max], word[first_min],
word[second_min], word[third_min])

y=np.where(other[:,-1]==1,1,-1)     # converting

def e(train, other):        # error function
    count = 0

    for i in range(len(other)):
        if other[i] != train[i]:
            count = count + 1
    return count/ len(other)

perceptron = Perceptron()   # calling class
```

```
for i in range(5):
   perceptron.perceptron_train(training, text)
   a = perceptron.p(training)
   print(e(a, y))
perceptron.words(word)

# importing
import math
#fucntion = math.exp(-(np.linalg.norm(x-z))/20)
#poly = math.pow(((np.dot(x, z)) + 10), 2)

# Class for polynomial function
class Perceptronn():
   def __init__(self):
      self.M = []


   def perceptron_train(self,train_x, train_y, other_x):
      labels = np.where(train_y[:,-1]==1,1,-1) # if 1 return 1 else -1
      output = []
      for t in range(len(train_x)):      # training
         a = 0
         for i in self.M:
            a += labels[i]*(math.pow(((np.dot(train_x[i], train_x[t])) + 10), 2))
         if labels[t]*a <=0:
            self.M.append(t)


      for i in range(len(other_x)):        # predict
         b = 0
         for j in self.M:
            b += labels[j]*(math.pow(((np.dot(train_x[j], other_x[i])) +10), 2))

         output.append(np.sign(b))
      return output


y=np.where(other[:,-1]==1,1,-1)
others = np.delete(other,len(other[0])-1, 1)
perceptronn = Perceptronn()     #calling class
for i in range(5):
   a = perceptronn.perceptron_train(training, text, others)
   print(e(a, y))
```

```
# class for exponential function
class Perceptron_s():
    def __init__(self):
        self.M = []
        #self.x = []

    def perceptron_train(self,train_x, train_y, other_x):
        labels = np.where(train_y[:,-1]==1,1,-1) # if 1 return 1 else -1
        output = []
        for t in range(len(train_x)):      # training
            a = 0
            for i in self.M:
                a += labels[i]*(math.exp(-(np.linalg.norm(train_x[i]-train_x[t]))/20))
            if labels[t]*a <=0:
                self.M.append(t)
                #self.x.append([labels[i], train_x[i]])

        for t in range(len(other_x)):      # predict
            b = 0
            for i in self.M:
                #print(len(labels))
                #print(len(other_x))
                b += labels[i]*(math.exp(-(np.linalg.norm(train_x[i]-other_x[t]))/20))


            output.append(np.sign(b))
        #print(output)

        return output


y=np.where(other[:,-1]==1,1,-1)
others = np.delete(other,len(other[0])-1, 1)
perceptron_s = Perceptron_s()     # calling class
for i in range(5):
    a= perceptron_s.perceptron_train(training, text, others)
    print(e(a, y))
```