Machine Learning - CS 675
Fall 15 – Prof. Roshan Mohammad Esfandiari

Assignment 2

1) Gradient Descent Perceptron

a) Gradient Descent
To find the local extrema of a function we can use the gradient of the function. After computing the gradient we move proportional to the negative or positive of the gradient.

b) Implement Gradient Descent Perceptron. Use the slides as your reference. The format of the data is like previous assignment. Your code needs to have the prediction functionality. **ONLY PRINT OUT W, DISTANCE TO ORIGIN.**

c) **Run on the following data :**

0 0
0 1
1 0
1 1
10 10
10 11
11 10
11 11

and labels :
0 0
0 1
0 2
0 3
1 4
1 5
1 6
1 7

2) Coordinate Descent Perceptron

a) As you probably heard in the class Coordinate Descent is another approach in optimizing functions. Please follow these steps to implement the coordinate descent version of Perceptron.
You can look at this paper :
http://www.work.caltech.edu/~ling/pub/tr05perceptron.pdf

b) There are three components in Coordinate Descent. The first one is the Error. In the original Perceptron the error term was

$$\sum_i \left(Y_i - (W^T x_i)\right)^2$$

Where in this version of perceptron the error is 0/1 loss which is the number of misclassifications given a W.
Second, in the original Perceptron we move in the direction of gradient but in this version we will have **d** and **alpha** which will guide us to the answer.
Our update statement would be like this :

$$W^{New} = W^{Old} + alpha * d$$

where **alpha** is a number which will determine our step size and **d** is a vector which will show us which component of W is changing. In our version of Perceptron we will look at **d** with the following definition :

$$d_i = (0,0,\dots,1,\dots,0,0)\ 1\ is\ in\ the\ i-th\ position$$

In order to determine **alpha** we need to introduce **delta**. The following is the definition of delta :

$$delta_i = dot\_product(x_i, d)$$

delta is a vector.

c) Read the data and labels and store them in **data, labels** array in Python**. Change all the zeros to -1 in labels array. This is very important. Add a column to each data row with the value 1 in it just like you did in the original Perceptron.**

d) **Initialize d to have 0 in all the columns. "d" is a vector of 1 x number of columns.**

e) Do an iteration on the columns. Start with the first column and do the steps **(f) to (h)** below for each column.

f) Create a new dataset like this :

$$\forall x_i \in data\ and\ \forall y_i \in labels(x_i, y_i)\ and\ delta_i \neq 0$$
$$\longmapsto (delta_i^{-1}\langle W, x_i\rangle, y_i sign(delta_i))$$

which basically means that for every data point in **data** we calculate **delta** value based on the definition in **(c)** and then if it was not zero we map that data point with its corresponding label to a new 1-dimensional data point with a new label. Store these values in lists called **data_prime and label_prime respectively.**

g) To find the best **alpha** you need to use this method. First multiply each **data_prime** with its corresponding **label_prime**. Then sort the result. Scan the result from left to right and whenever you have a change in the sign set alpha to be the middle point. If there is no change in the sign set return 0. This is a simple example for you to better understand this :

if **`data_prime`** is :

```
([1.5862809419631958, 2.3768045902252197,
1.5592775344848633, 1.6383299827575684,
1.4791539907455444, 1.5510196685791016])
```

and **`label_prime`** is **:**

```
[-1, -1, 1, 1, 1, 1]
```

then the multiplication would be this :

```
[-2.3768045902252197, -1.5862809419631958,
1.4791539907455444, 1.5510196685791016,
1.5592775344848633, 1.6383299827575684]
```

which by scanning we can see there is a sign change between the second element and the third element so we set alpha to be :

`alpha = (-1.5863+1.4792)/2 = -0.0535`

Create a **FUNCTION** named **`find_alpha(data_prime,label_prime)`** to do this.

h) Update the W with the statement in **(b)**. remember because we are doing coordinate descent in each iteration we only update one column of W and that column is where **d** has 1. Compute the error for the new **W** and if it was better than before keep the change. If not revert the change to the W before the update.

Do all the steps from **(d)** to **(h)** **until either you have zero error** or your **counter** hit **100.** Print out the **Error and W.**

The following is a starting point for your code:

```
from sys import argv
from array import array
import random
from math import copysign as sign
import pdb

def dot(a,b):
    res = float(0)
    for i in range(0,len(a)):
        res += (a[i]*b[i])

    return res

def find_alpha(data,label):
    x = sorted([a*b for a,b in zip(data,label)])
    #### FIND WHERE THE SIGN CHANGES IN x
    return alpha


data_path = argv[1]
label_path = argv[2]

##### READ THE DATA IN AN ARRAY
data_file = open(data_path)
data = []
### READ DATA AND ADD 1 TO THE END OF EACH ROW.
rows = len(data)
cols = len(data[0])
##### READ THE LABELS IN AN ARRAY
label_file = open(label_path)
labels = []
### READ LABELS
##### Starting The Coordinate Descent
W = array('f')
d = array('f')
```

```
### INITIALIZE W TO A RANDOM PLANE.
delta = float(0)
data_prim = array('f')
label_prim = array('i')
alpha = 0
prev_error = 0
error = 10000000
stop_condition = 100
while(stop_condition != 0 or error != 0):
    prev_error = error
    for j in range(0,cols):
        d[j] = 1
        for i in range(0,rows):
            delta = dot(d,data[i])
            if(delta != 0):
                ###CREAT THE NEW DATA AND APPEND IT
TO DATA_PRIME AND LABEL_PRIME
        alpha = find_alpha(data_prim,label_prim)
        error = 0
        ### CHANGE W AND COMPUTE ERROR
        if (error < prev_error):
            ### UPDATE W
        d[j]= 0
        del data_prim[:]
        del label_prim[:]
    stop_condition -= 1

print(W)
print(error)
##### NOW READ THE TEST DATA FROM THE INPUT
##### ASSIGN THE CORRECT CLASS TO THE TEST DATA
BASEDD ON THE SIGNS
```