Midterm Exam

Kaitlyn Mulligan

3/13/19

1 Instructions

This test could be written in LaTeX, just as all the homework assignments. Write in understandable, easy to follow English. Make sure you provide good illustrations and figures. Remember to include your Python programs in your assignment.

Your assignment should be submitted in two ways: through GitHub, and in hardcopy (in class). Use the **same** repository you have been using and submit your work in a folder named "`lastname-midterm`", where lastname is your last name.

2 Problem Set

The following is a list of problems you will work on. When providing your solutions (hopefully using LaTeX), do not simply give the final answer, show how you arrived to the solution, justify your assumptions, and explain your results clearly.

2.1 1

Compare two algorithms on a classification task: the Pocket algorithm (designed for classification), and linear regression (not designed for classification). For linear regression, after learning the weights $\mathbf{w}$; we use $h(\mathbf{x}) = \text{sign}\ (\mathbf{w}^T\mathbf{x})$ to classify $\mathbf{x}$. For the dataset, start by using the following Python code:

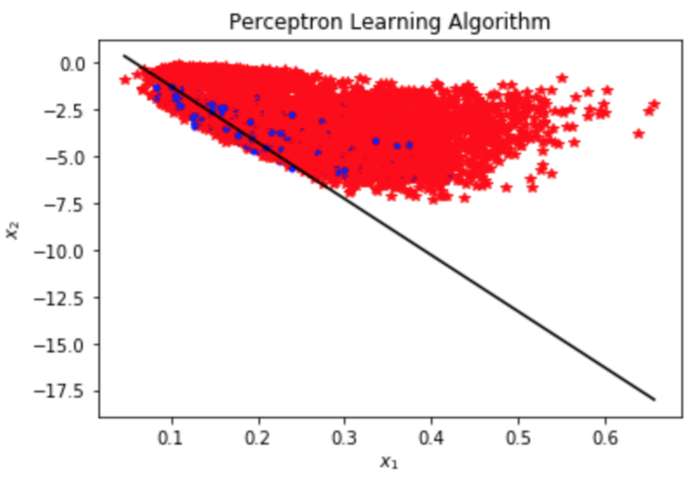
```
import numpy as np
import matplotlib.pyplot as plt
from numpy import genfromtxt
# read digits data & split it into X (training input) and y (target output)
dataset = genfromtxt('features.csv', delimiter=' ')
y = dataset[:, 0]
X = dataset[:, 1:]
y[y!=0] = -1     #rest of numbers are negative class
y[y==0] = +1     #number zero is the positive class
# plots data
c0 = plt.scatter(X[y==-1,0], X[y==-1,1], s=20, color='r', marker='x')
c1 = plt.scatter(X[y==1,0], X[y==1,1], s=20, color='b', marker='o')
# displays legend
plt.legend((c0, c1), ('All Other Numbers -1', 'Number Zero +1'),
        loc='upper right', scatterpoints=1, fontsize=11)
# displays axis legends and title
plt.xlabel(r'$x_1$')
plt.ylabel(r'$x_2$')
plt.title(r'Intensity and Symmetry of Digits')
# saves the figure into a .pdf file (desired!)
plt.savefig('midterm.plot.pdf', bbox_inches='tight')
plt.show()
```

Create another dataset using the same method as above but now you will classify between 4 and the rest of the numbers. To do this you should change lines 8 and 9 as follows:
`y[y!=4] = -1`
`y[y==4] = +1`
Try the following three approaches using the dataset with the new changes, plot the final (best) hypothesis on each and then explain which works best in terms both E_{out} and the amount of computation required. E.g., what is the final classification error? after how many iterations did you stop the Pocket algorithm?

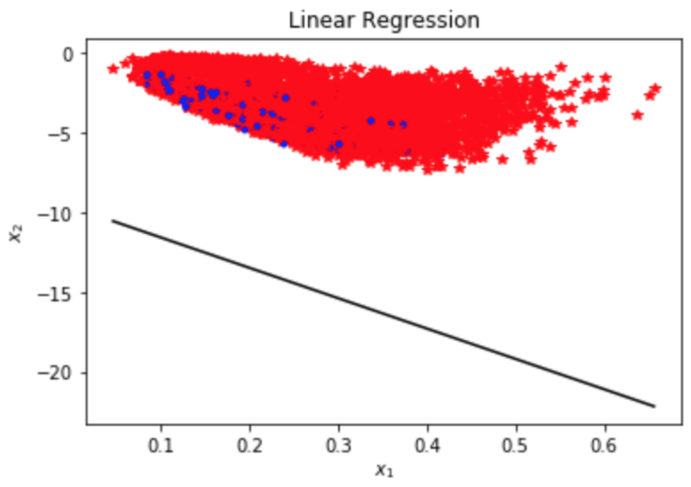
First I created another data set using the same method as above and utilized it in my Perceptron Learning Algorithm. The code I used for the next three parts can be seen below. You can find the data given in the question in the data section of the main method. For each part, the only changes that were made were to comment/uncomment specific lines of the code which I will metion below.

```
from sklearn.manifold import TSNE
from sklearn.decomposition import PCA
from sklearn.manifold import Isomap
from sklearn.manifold import SpectralEmbedding

import random
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets.samples_generator import make_blobs

from numpy import genfromtxt

def pltPer(X, y, W):
    f = plt.figure()
    if X.shape[1] > 3:
        # reduce dimensions for display purposes
        Xw = np.append(X, np.reshape(W, (1, len(W))), 0)    # add a column of ones
        Xs = SpectralEmbedding(n_components=2).fit_transform(Xw)

        Xs = np.append(np.ones((Xs.shape[0],1)), Xs, 1)    # add a column of ones
        X = Xs[:-1,:]
        W = Xs[-1,:]

    for n in range(len(y)):
        if y[n] == -1:
            plt.plot(X[n,1],X[n,2],'r*')
        else:
            plt.plot(X[n,1],X[n,2],'b.')
    m, b = -W[1]/W[2], -W[0]/W[2]
    l = np.linspace(min(X[:,1]),max(X[:,1]))
    plt.plot(l, m*l+b, 'k-')
    plt.xlabel("$x_1$")
    plt.ylabel("$x_2$")
    plt.title("Perceptron Learning Algorithm")
    #plt.title("Linear Regression")

def classification_error(w, X, y):
    err_cnt = 0
    N = len(X)
    for n in range(N):
        s = np.sign(w.T.dot(X[n])) # if this is zero, then :(
        if y[n] != s:
            err_cnt += 1
    print(err_cnt)
    return err_cnt

def choose_miscl_point(w, X, y):
    mispts = []
    # Choose a random point among the misclassified
    for n in range(len(X)):
        if np.sign(w.T.dot(X[n])) != y[n]:
            mispts.append((X[n], y[n]))
    #print(len(mispts))
    return mispts[random.randrange(0,len(mispts))]

def main():
    itlst = []
    for x in range(1):

        # data

        dataset = genfromtxt('features.csv', delimiter=' ')
        y = dataset[:, 0]
        X = dataset[:, 1:]
        N = len(y)
        y[y!=4] = -1    #rest of numbers are negative class
        y[y==4] = +1    #number zero is the positive class
        X = np.append(np.ones((N,1)), X, 1)   # add a column of ones

        # Linear Regression
        #Xs = np.linalg.pinv(X.T.dot(X)).dot(X.T)
        #wlr = Xs.dot(y)

        #pltPer(X,y,wlr)
        #return

        # initialize the weigths to zeros
        w = np.zeros(3)
        #w = wlr
        it = 0
        pltPer(X,y,w)  # initial solution (bad!)

        stopIt = 10000
        currIt = 0
        bestW = w
        bestE = N
        # Iterate until all points are correctly classified
        nerr = classification_error(w, X, y)
        while nerr!= 0:
            it += 1
            currIt += 1
            if currIt > stopIt:
                print("Early stop! No progress!")
                break
            # Pick random misclassified point
            x, s = choose_miscl_point(w, X, y)
            # Update weights
            w += s*x
            nerr = classification_error(w, X, y)
            if nerr < bestE:
                currIt = 0
                bestE = nerr
                bestW = w

        w = bestW
        f = plt.figure()
        pltPer(X,y,w)
        print("Total iterations: " + str(it))
        itlst.append(it)
        #print(itlst)
    plt.title("Perceptron Learning Algorithm")
    #plt.title("Linear Regression")

main()
```

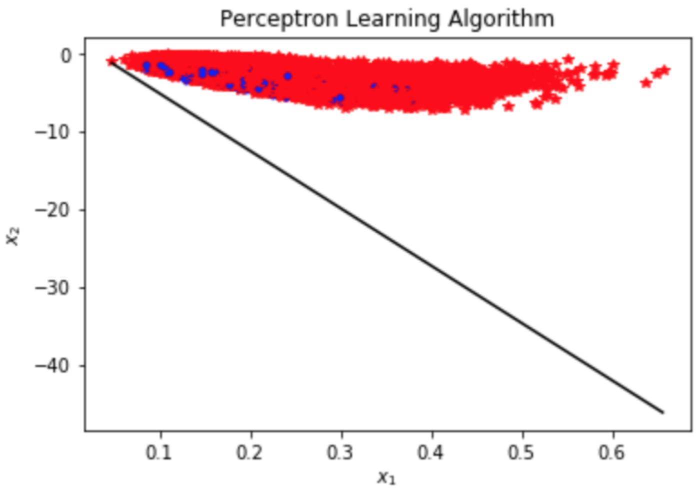
(a) The Pocket algorithm, starting from **w**=0.
To implement the Pocket algorithm, starting from **w**=0, I commented out the lines in the linear regression section, initialized the weights to zero with the line `w = np.zeros(3)`, and commented out the line `w=wlr`. I stopped the Pocket Algorithm after 10,000 iterations. Below is a plot of the final hypothesis. The final classification error was 1063.



(b) Linear regression (applied as a classification method).
To implement Linear Regression, I uncommented the lines in the linear regression section. After the return statement the algorithm will stop running so I do not need to comment or uncomment any lines in that section. Below is a plot of the final hypothesis.



(c) The Pocket algorithm, starting from the solution given by linear regression.
To implement the Pocket algorithm, starting from the solution given by linear regression, I uncommented the first two lines of the linear regression section and commented the two lines: `pltPer(X,y,wlr)` and `return`. I then commented the line `w=np.zeros(3)` and uncommented the line `w=wlr`. I stopped the Pocket Algorithm after 10,000 iterations. Below is a plot of the final hypothesis. The final classification error was 652.



Analyzing these results we can see that the Pocket algorithm, starting from **w**=0, has a classification error of 1063. Then we used linear regression and did not recieve great results. Going back to the Pocket Algorithm starting from the solution given by linear regression, we obtained a smaller classification error than in part a. The classification error for part c was 652. For both part a and c, the Pocket algorithm was stopped after 10,000 iterations.

2.2 2

Write a Python program that solves **Problem 2.12** in an iterative manner. If you are feeling adventurous, plot the values of N as the program converges to a steady value of N.

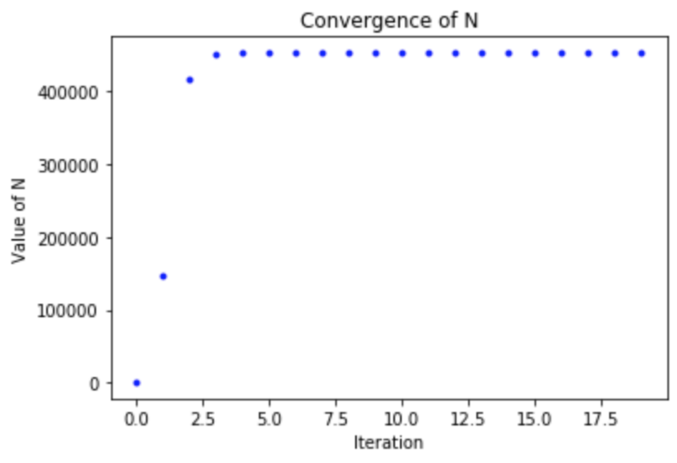
Problem 2.12 states:

For an $\mathcal{H}$ with $d_{VC} = 10$, what sample size do you need (as prescribed by the generalization bound) to have 95% confidence that your generalization error is at most 0.05?

Having already answered this question in a previous homework assignment, we know that the value of N should be converging to approximately 452956. To solve this in an iterative manner, we can use the following Python program which utilizes Equation (2.13) and the given values.

```
import numpy as np
import matplotlib.pyplot as plt

def main():
    dVC = 10 # value of d_{VC} given
    epsilon = 0.05 # value of epsilon given
    delta = 0.05 # value of delta given
    Nlst = []
    N = 1000 # initial value of N
    Nlst.append(N)
    for i in range(1,20):
        N = (8 / (np.power(epsilon, 2))) * (np.log((4 * (np.power((2 * N),
            (dVC)) + 1)) / delta))
        Nlst.append(N)
    plt.xlabel("Iteration")
    plt.ylabel("Value of N")
    plt.title("Convergence of N")
    iter = np.arange(20)
    plt.plot(iter,Nlst,'b.')
    print(N)

main()
```

After 20 iterations, this Python program returns a value of $N = 452956.8647230992$. Feeling adventurous, I decided to plot the values of N as the program converges to a steady value of N. I decided to write the code for the plot in the code above. The resulting plot of N is below.



2.3 3 (extra credit)

Albert Einstein said "Creativity is intelligence having fun". You are very smart and talented. That is why you are in this class. With that in mind... I recently acquired the domain name "`marist.ai`" and I am looking to create site for students and faculty to share their AI-related projects; so, for a chance to be displayed our future website, draw some type of logo for the new site here:

Here is my proposed logo for the new `marist.ai` domain name.