Midterm LaTeX Sheet

Cameron Brenner

3/13/2019

1 Question 1

1.1 Part A.

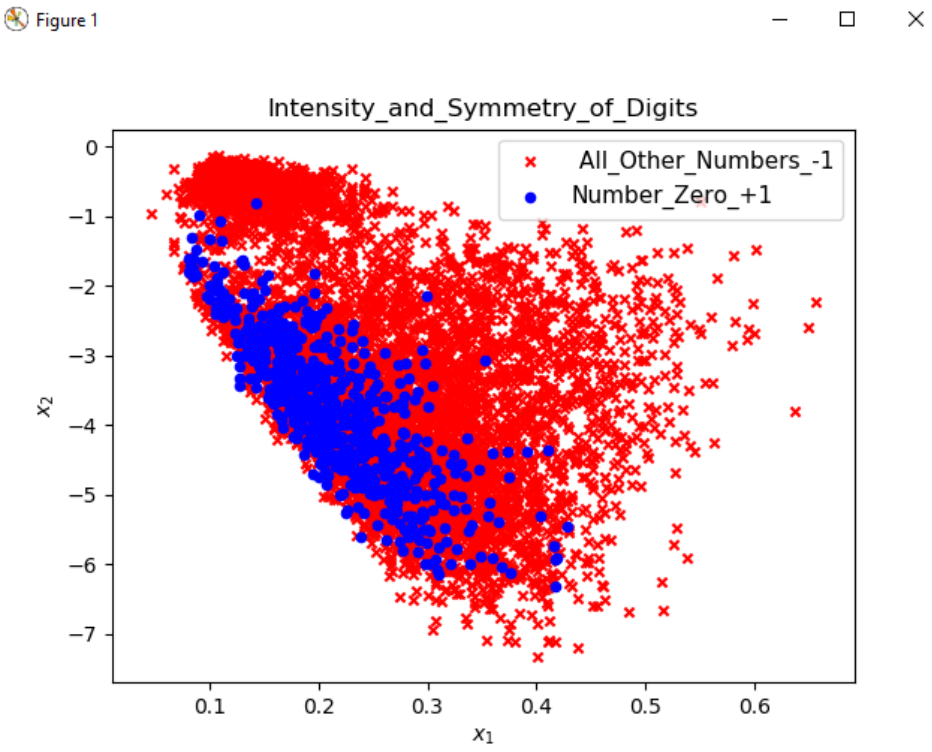
In part A, we are asked to change the data set such that we will be classifying four against all of the other numbers, and we do this by changing the lines where $y! = 0$ and $y == 0$ to $y! = 4$ and $y == 4$. We are asked to perform a Pocket Algorithm approach on this. A Pocket Algorithm picks the best error classification and stores it, and if the data is not linearly separable and the algorithm cannot find a line through the data, we use the best error classification. The code is as follows.

```
import numpy as np
import matplotlib.pyplot as plt
from numpy import genfromtxt
import random

def classification_error(w, X, y):
    bestErr = len(X)
    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
    if err_cnt < bestErr:
        bestErr = err_cnt
    print(err_cnt)
    return err_cnt
    return bestErr

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 plotAlg(X,y,w):
    #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' )

    #plot hypothesis
```

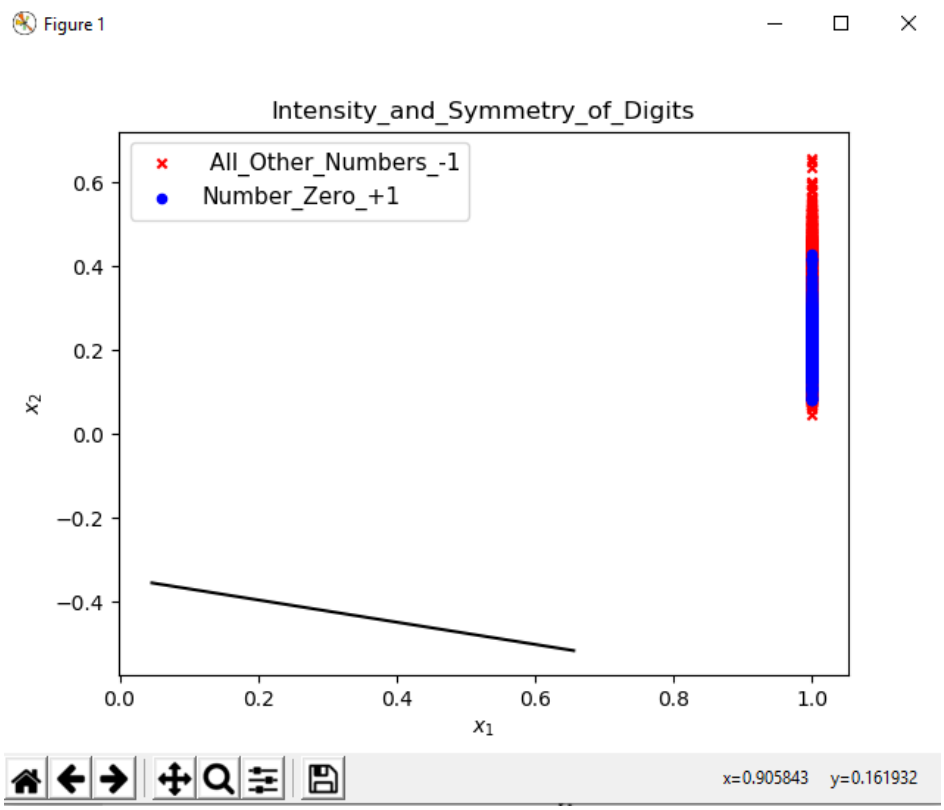
```
    m, b = -w[1]/w[2], -w[0]/w[2]
    l = np.linspace(min(X[:,1]),max(X[:,1]))
    plt.plot(l, m*l+b, 'k-')

    #displays legend
    plt.axis(option="auto")
    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()

#Pocket Algorithm

# read digits data & split it into X (training input) and y (target output)
dataset = genfromtxt('C:\\Users\\Cameron\\Documents\\DATA440\\midterm\\features.csv', delim
y = dataset[:, 0]
X = dataset[:, 1 : ]
y[y!=4] = -1 #rest of numbers are negative class
y[y==4] = +1 #number four is the positive class

Xs = np.append(np.ones((len(X),1)), X, 1)    # add a column of ones for compatability with
w = np.zeros(3)

plotAlg(X, y, w)#Initial Solution

it = 0

stopIt = 50
currIt = 0
bestW = w
bestE = len(Xs)
# Iterate until all points are correctly classified
nerr = classification_error(w, Xs, y)
while nerr != 0:
    nerr = classification_error(w, Xs, y)
    it += 1
    currIt += 1
    if currIt > stopIt:
        print("Early stop! no progress")
        break
    # Pick random misclassified point
    x, s = choose_miscl_point(w, Xs, y)
    # Update weights
    w += s*x
    nerr = classification_error(w, Xs, y)
    if nerr < bestE:
        currIt=0
        bestE = nerr
        bestW = w
```

```
w=bestW
plotAlg(Xs,y,bestW)
print("The number of iterations is: " + str(it))
```

Running this, we see that the data is not linearly separable.



And the final plot we get is:



Not that great, but the least error classification obtained is 652,

```
6639
6639
5609
5609
652
652
652
652
6639
6639
652
652
6639
6639
652
652
6639
6639
652
652
Early stop! no progress
The number of iterations is: 52
The best error classification is: 652
```

1.2 Part B.

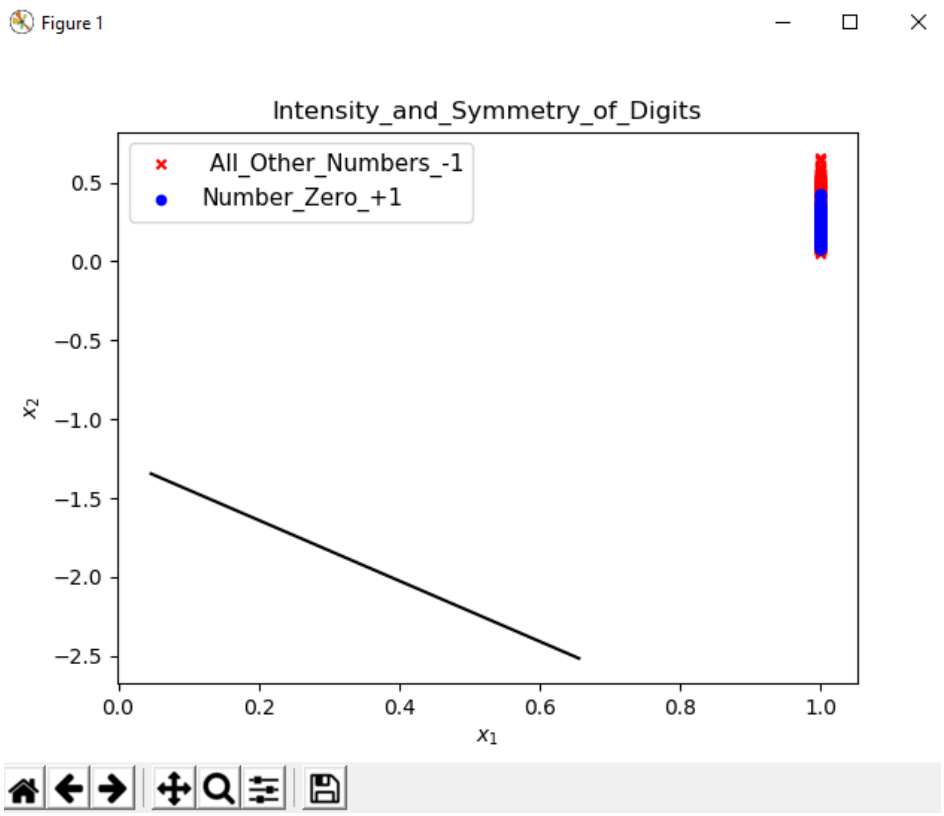
In Part B, we are asked to use linear regression on the data set. The code for linear regression with the data set is the following

```
import numpy as np
import matplotlib.pyplot as plt
from numpy import genfromtxt
import random

def classification_error(w, X, y):
    bestErr = len(X)
    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
    if err_cnt < bestErr:
        bestErr = err_cnt
    print(err_cnt)
    return err_cnt
    return bestErr

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 plotAlg(X,y,w):
    #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')

    #plot hypothesis
    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-')

    #displays legend
    plt.axis(option="auto")
    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()

#Linear Regression

# read digits data & split it into X (training input) and y (target output)
dataset = genfromtxt('C:\\Users\\Cameron\\Documents\\DATA440\\midterm\\features.csv', delir
y = dataset[:, 0]
X = dataset[:, 1 : ]
y[y!=4] = -1 #rest of numbers are negative class
y[y==4] = +1 #number four is the positive class

Xo = np.append(np.ones((len(X),1)), X, 1)    # add a column of ones
Xs = np.linalg.pinv(Xo.T.dot(Xo)).dot(Xo.T)
wlr = Xs.dot(y)

plotAlg(X, y, wlr)#Initial Solution
```

```
it = 0

stopIt = 50
currIt = 0
bestW = wlr
bestE = len(X)
# Iterate until all points are correctly classified
nerr = classification_error(wlr, Xo, y)
while nerr != 0:
    nerr = classification_error(wlr, Xo, y)
    it += 1
    currIt += 1
    if currIt > stopIt:
        print("Early stop! no progress")
        break
    # Pick random misclassified point
    x, s = choose_miscl_point(wlr, Xo, y)
    # Update weights
    wlr += s*x
    nerr = classification_error(wlr, Xo, y)
    if nerr < bestE:
        currIt=0
        bestE = nerr
        bestW = wlr

plotAlg(Xo,y,bestW)
print("The number of iterations is: " + str(it))
print("The best error classification is: " + str(bestE))
```

The result we get from using linear regression is



Still not ideal, but at least we are a little closer to the target data than when we were using the pocket algorithm. Because this is non linearly separable, it would run forever if it didn't have a function similar to the pocket algorithm, so they have similar run times/efficiencies. The best error classification is yet again 652.

```
5895
5895
652
652
6057
6057
652
652
6333
6333
652
652
6406
6406
652
652
6203
6203
Early stop! no progress
The number of iterations is: 54
The best error classification is: 652
```

1.3 Part C.

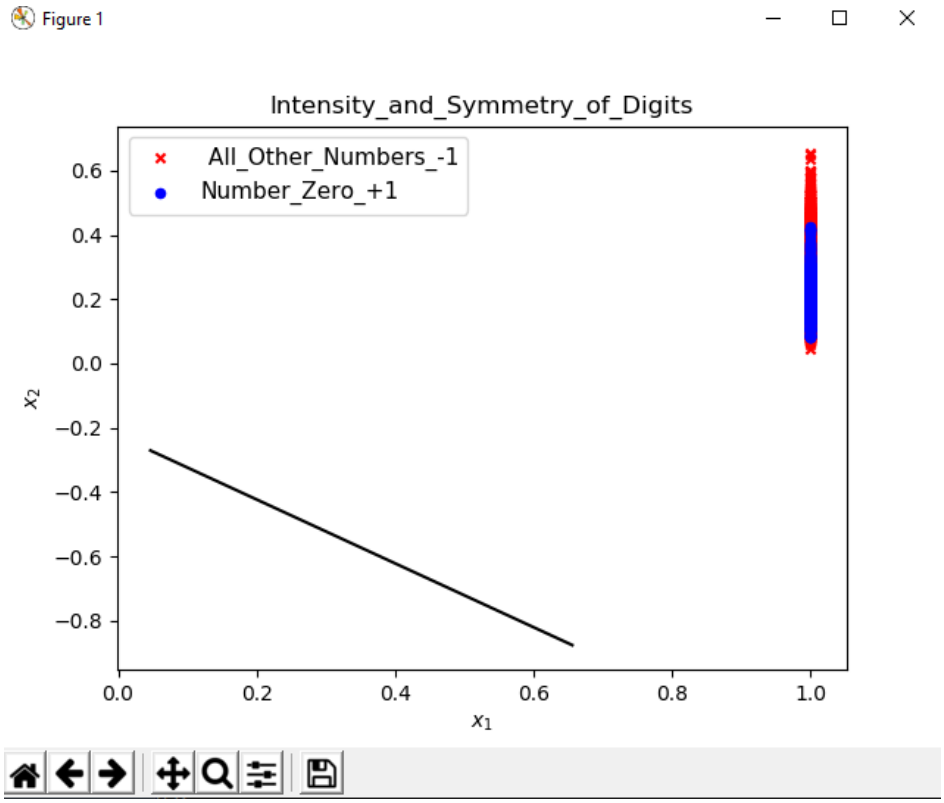
In Part C, we are asked to use linear regression to find a w, and then initialize the w in a pocket algorithm with the w we obtained in the pocket algorithm. The code is as follows.

```
import numpy as np
import matplotlib.pyplot as plt
from numpy import genfromtxt
import random

def classification_error(w, X, y):
    bestErr = len(X)
    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
    if err_cnt < bestErr:
        bestErr = err_cnt
```

```
    print(err_cnt)
    return err_cnt
    return bestErr

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 plotAlg(X,y,w):
    #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' )

    #plot hypothesis
    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-')

    #displays legend
    plt.axis(option="auto")
    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()

#Linear Regression

# read digits data & split it into X (training input) and y (target output)
dataset = genfromtxt('C:\\Users\\Cameron\\Documents\\DATA440\\midterm\\features.csv', delir
y = dataset[:, 0]
X = dataset[:, 1 : ]
y[y!=4] = -1 #rest of numbers are negative class
y[y==4] = +1 #number four is the positive class

#linear regression
Xo = np.append(np.ones((len(X),1)), X, 1)    # add a column of ones
Xs = np.linalg.pinv(Xo.T.dot(Xo)).dot(Xo.T)
wlr = Xs.dot(y)

plotAlg(X, y, wlr)#Initial Solution

it = 0

stopIt = 50
```

```
currIt = 0
bestW = wlr
bestE = len(X)
# Iterate until all points are correctly classified
nerr = classification_error(wlr, Xo, y)
while nerr != 0:
    nerr = classification_error(wlr, Xo, y)
    it += 1
    currIt += 1
    if currIt > stopIt:
        print("Early stop! no progress")
        break
    # Pick random misclassified point
    x, s = choose_miscl_point(wlr, Xo, y)
    # Update weights
    wlr += s*x
    nerr = classification_error(wlr, Xo, y)
    if nerr < bestE:
        currIt=0
        bestE = nerr
        bestW = wlr
print("Now for the pocket algorithm")
#starting pocket algorithm at bestW
w = bestW


stopIt = 50
currIt = 0
bestW = w
bestE = len(X)
# Iterate until all points are correctly classified
nerr = classification_error(w, Xo, y)
while nerr != 0:
    nerr = classification_error(w, Xo, y)
    it += 1
    currIt += 1
    if currIt > stopIt:
        print("Early stop! no progress")
        break
    # Pick random misclassified point
    x, s = choose_miscl_point(w, Xo, y)
    # Update weights
    w += s*x
    nerr = classification_error(w, Xo, y)
    if nerr < bestE:
        currIt=0
        bestE = nerr
        bestW = w
plotAlg(Xo, y, bestW)
print("The number of iterations is: " + str(it))
print("The best error classification is: " + str(bestE))
```

The result we get from this is



Not exactly helpful either, and less accurate than the solution we found from using linear regression on its own. Yet again the best error classification was 652.

```
652
652
5543
5543
652
652
5704
5704
652
652
5167
5167
652
652
652
652
5936
5936
652
652
5974
5974
Early stop! no progress
The number of iterations is: 106
The best error classification is: 652
```

2 Question 2

In Question 2, we are asked to find an iterative way of solving question 2.12 in the book, and if we are feeling adventurous, to plot the changes of N. I am not feeling adventurous. I am scared and tired. I am so sorry Dr. Rivas. The equation for finding the ideal sample size is equation 2.13 in the book and is,

$$N \geq \frac{8}{\epsilon^2} ln \frac{4((2N)^{d_{vc}} + 1)}{\delta} \tag{1}$$

We then substitute 0.05 for δ and ϵ. The code I used to solve this in an iterative manner is the following,

```
import numpy as np

delta = 0.05 #confidence error
```

```
epsilon = 0.05 #generalization error
d_vc = 10 #VC dimension
initGuess = d_vc * 1000 #initial guess for N
nextGuess = 0
count = 1
tempGuess = initGuess

for i in range(1,100):
    #print("tempguessin : " + str(tempGuess)) personal  book keeping
    print("The initial guess for iteration " + str(count) + " of the generalization is: " +
    count+=1
    nextGuess = 8 / epsilon**2 * np.log((4 * ((2 * tempGuess) ** d_vc + 1)) / delta)
    print("The output of this guess is: " + str(nextGuess) + ".")
    if(abs(tempGuess - nextGuess)>500):
        tempGuess = nextGuess
        #print("tempguessout : " + str(tempGuess))     personal book keeping
        print("now for the next iteration!")
    else:
        break


print("The Guess for sample size with a VC dimension of" + str(d_vc) + "  is: " + str(next
```

Running this, we get the result that does indeed solve for N, and the solution is similar to the one found for the previous homework question. The results are:

```
C:\Users\Cameron\Python\Python37-32>python midterm02.py
The initial guess for iteration 1 of the generalization is: 10000
The output of this guess is: 330934.0869121125.
now for the next iteration!
The initial guess for iteration 2 of the generalization is: 330934.0869121125
The output of this guess is: 442912.7790515471.
now for the next iteration!
The initial guess for iteration 3 of the generalization is: 442912.7790515471
The output of this guess is: 452239.2955728956.
now for the next iteration!
The initial guess for iteration 4 of the generalization is: 452239.2955728956
The output of this guess is: 452906.13048752997.
now for the next iteration!
The initial guess for iteration 5 of the generalization is: 452906.13048752997
The output of this guess is: 452953.28030570166.
The Guess for sample size with a VC dimension of 10  is: 452953.28030570166
```