STATISTICAL MACHINE LEARNING

Stat 775 HW 4

Andrew Olson - Harrison Stanton

February 25, 2015

1 PROBLEM SET

1.1 EXERCISE 3.12

Show that the ridge regression estimates can be obtained by ordinary least squares regression on an augmented data set. We augment the centered matrix $\mathbf{X}$ with p additional rows $\sqrt{\lambda}\mathbf{I}$, and augment $\mathbf{y}$ with p zeros.

$$\hat{X} = \begin{bmatrix} X \\ \sqrt{\lambda} I_{pxp} \end{bmatrix} \hat{Y} = \begin{bmatrix} Y \\ \mathcal{O}_{px1} \end{bmatrix}$$

By introducing artificial data having response value zero, the fitting procedure is forced to shrink the coefficients toward zero. This is ... that satisfy them.

$$\hat{\beta}_{LS} = (\hat{X}^T \hat{X} + \lambda I_{pxp})^{-1} \hat{X}^T \hat{Y}$$

$$\hat{X}^T \hat{X} = \begin{bmatrix} X^T & \sqrt{\lambda} I_{pxp} \end{bmatrix} \begin{bmatrix} X \\ \sqrt{\lambda} I_{pxp} \end{bmatrix} = X^X + \lambda I_{pxp}$$

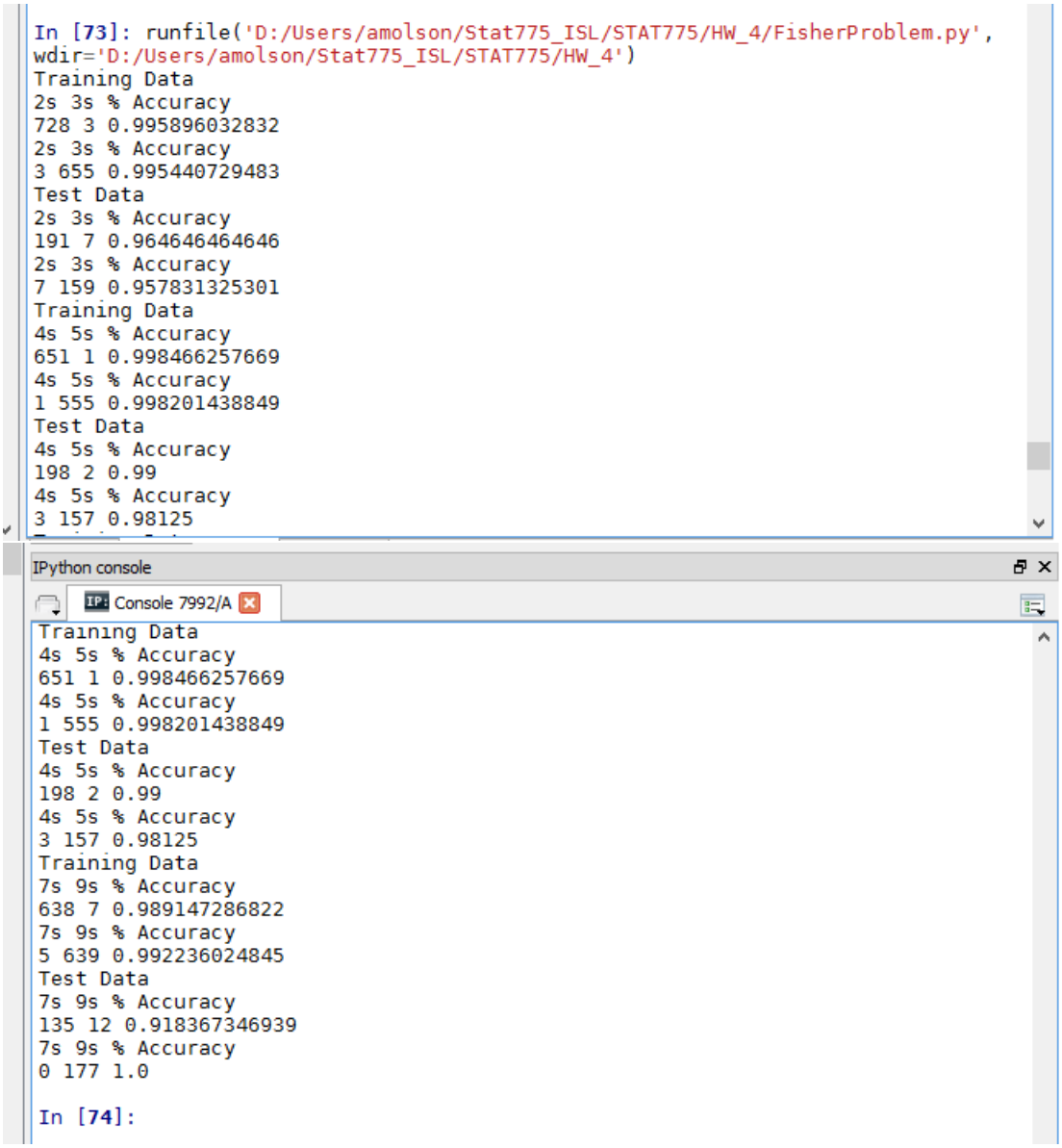
$$\hat{X}^T \hat{Y} = \begin{bmatrix} X^T & \sqrt{\lambda} \end{bmatrix} \begin{bmatrix} Y \\ \mathcal{O}_{px1} \end{bmatrix} = X^T Y$$

$$\hat{\beta}_{LS} = (X^T X + \lambda_{pxp})^{-1} X^T Y$$

1.2 CODE

The assignment was to use Fisher Discriminators to distinguish between 2 and 3, 4 and 5, and 7 and 9 from the Zip code digit data given by the book.

The only real issues were related to trying to get Python and Numpy to get the dimensionality of arrays and matrices correct. Beyond that, dealing with the occasional singular matrix was the only other excitement. The following give the output for both the training and the test data.



Here is the actual code.

```
#23, 45, 79

import numpy as np

import matplotlib.pyplot as plt
import numpy.linalg as la

import math

sigma = np.load("sigmazip.npy")
mu = np.load("muzip.npy")

training = np.load("trainingzip.npy")
testing = np.load("testingzip.npy")

#Check any pairs you like.
Numbers = [(2, 3), (4, 5), (7, 9)]

in_array = testing

distancearray = []

tracking = []

def probofx(x, mu, sigma):
    inter = 1.0/(math.sqrt(2 * math.pi * sigma))
    expon = (-0.5 * (x - mu) ** 2)/(sigma ** 2)
    return inter * math.exp(expon)

def buildu(sigma, mu, first, second):
    """Builds a based on sigma, mu, first and second values under
    investigation"""
    #Prebuild arrays and catch singular matrix and try epsilon I matrix
    try:
        sigmasub = np.matrix(la.inv(sigma[second] + sigma[first]))
    except np.linalg.LinAlgError:
        sigmasub = np.matrix(la.inv(sigma[second] + sigma[first] +
            np.identity(len(sigma[second]))*np.finfo(float).eps))

    musub = np.matrix(mu[second] - mu[first])
```

```
    #Put them together and then normalize to unit vector. a \equiv u
    a = sigmasub * np.transpose(musub)
    a = np.divide(a, la.norm(a))

    return a

#Change this value to test the pairs in Numbers. 0 index
for problem in range(3):

    first = Numbers[problem][0]
    second = Numbers[problem][1]

    points = [first, second]

    a = buildu(sigma, mu, first, second)

    #Python was being a butt so I had to go through this little song and
    #dance to get floats in the array, not an array of array points. Don't ask.
    aprime = []
    for i in a:
        aprime.append(float(i))

    train = []
    test = []

    mean = [[], []]
    sd = [[], []]
    count = [0, 0]

    #Build empty array
    for i in range(10):
        train.append([])
    #Add training data to matrix it belongs in
    for i in training:
        train[i[1]].append(i[0])

    #Build empty array
    for i in range(10):
        test.append([])
    #Add training data to matrix it belongs in
    for i in testing:
        test[i[1]].append(i[0])

    #This was getting silly since note they use different mu.
```

```
mean[0] = np.dot(mu[first], aprime)
sd[0] = math.sqrt(float(np.transpose(a) * sigma[first] * a))

mean[1] = np.dot(mu[second], aprime)
sd[1] = math.sqrt(float(np.transpose(a) * sigma[second] * a))

print "Training Data"
#And this is it
for order in points:
    count = [0, 0]
    for i in range(len(train[order])):

        ifis = [0, 0]
        whichvalue = np.dot(train[order][i], aprime)
        ifis[0] = probofx(whichvalue, mean[0], sd[0])

        ifis[1] = probofx(whichvalue, mean[1], sd[1])

        if ifis[0] > ifis[1]:
            count[0] += 1
        else:
            count[1] += 1

    if order == first:
        print str(first) + 's', str(second) + 's', '% Accuracy'
        print count[0], count[1], float(count[0])/float(count[0]+count[1])
    else:
        print str(first) + 's', str(second) + 's', '% Accuracy'
        print count[0], count[1], float(count[1])/float(count[0]+count[1])

print "Test Data"
#And this is it
for order in points:
    count = [0, 0]
    for i in range(len(test[order])):

        ifis = [0, 0]
        whichvalue = np.dot(test[order][i], aprime)
        ifis[0] = probofx(whichvalue, mean[0], sd[0])

        ifis[1] = probofx(whichvalue, mean[1], sd[1])

        if ifis[0] > ifis[1]:
            count[0] += 1
```

```
            else:
                count[1] += 1

        if order == first:
            print str(first) + 's', str(second) + 's', '% Accuracy'
            print count[0], count[1], float(count[0])/float(count[0]+count[1])
        else:
            print str(first) + 's', str(second) + 's', '% Accuracy'
            print count[0], count[1], float(count[1])/float(count[0]+count[1])
```