Lewis Arnsten

Homework 2

1. $X=\begin{bmatrix}1&2&1\\2&1&0\\2&0&0\end{bmatrix}$

a) All columns are linearly independent so the matrix is full rank. The rank is 3.

b) $y = Xw \rightarrow$ multiply by X^{-1}

$X^{-1}y = XX^{-1}w$

$X^{-1}y = Iw = w$

$w = X^{-1}y = \begin{bmatrix}0&0&1/2\\0&1&-1\\1&-2&3/2\end{bmatrix} y$

2. $X = \begin{bmatrix}10&1&3\\20&0&2\\30&0&1\end{bmatrix}$ $y = \begin{bmatrix}110\\110\\210\end{bmatrix}$

a) $Xw=y$, $X^{-1}y=w$ $w = \begin{bmatrix}31/4\\100\\-45/2\end{bmatrix}$

See code

b) $Xw=y$ $\begin{bmatrix}10&x_1&3\\20&x_2&2\\30&x_3&1\end{bmatrix}\begin{bmatrix}5\\10\\5\end{bmatrix} = \begin{bmatrix}110\\110\\210\end{bmatrix} = \begin{bmatrix}50+10x_1+15\\100+10x_2+10\\150+10x_3+5\end{bmatrix}$

$x_1=4.5$ $x_2=0$ $x_3=5.5$ fat $= \begin{bmatrix}4.5\\0\\5.5\end{bmatrix}$

i

```python
import numpy as np
import pandas as pd
from scipy.io import loadmat
import matplotlib.pyplot as plt

#2a)
x = [[10,1,3],[20,0,2],[30,0,1]]
y = [[110],[110],[210]]

mx = np.matrix(x)
my = np.matrix(y)
inverse = np.linalg.inv(mx)

xy = inverse @ my

print(xy)
```

```
[[  7.75]
 [100.  ]
 [-22.5 ]]
```

column 2 + column 3 = column 1

c) $X = \begin{array}{c} \text{column} \\ \begin{bmatrix} 10 & 5 & 5 & 1 & 3 \\ 20 & 12 & 8 & 0 & 2 \\ 30 & 20 & 10 & 0 & 1 \\ 30 & 15 & 15 & 0 & 3 \\ 35 & 20 & 15 & 2 & 4 \end{bmatrix} \end{array}$ $\quad y = \begin{bmatrix} 75 \\ 110 \\ 155 \\ 165 \\ 215 \end{bmatrix}$

Since columns 1-3 of X are not linearly independent, X doesn't have an inverse.

Thus you cannot solve $Xw = y$

$$X \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \end{bmatrix} = \begin{bmatrix} 75 \\ 110 \\ 155 \\ 165 \\ 215 \end{bmatrix}$$

$3(10w_1 + 5w_2 + 5w_3 + w_4 + 3w_5 = 75)3$

$30w_1 + 15w_2 + 15w_3 + 0w_4 + 3w_5 = 165$

$3w_4 + 6w_5 = 60$

$20w_1 + \underbrace{10w_2 + 10w_3}_{} - w_4 = 90$

$$w = \begin{bmatrix} 5 \\ 0 \\ 0 \\ 10 \\ 5 \end{bmatrix} \rightarrow \begin{bmatrix} 5 - w_2 \\ w_2 \\ w_2 \\ 10 \\ 5 \end{bmatrix}$$

any solution in this form agrees with the true calories/gram

```
#3) a)
data = loadmat('face_emotion_data.mat')
Face_X = data['X']
Face_y = data['y']
Face_w = np.linalg.inv(Face_X.T@Face_X)@Face_X.T@Face_y

print(Face_w)
```

```
[[ 0.94366942]
 [ 0.21373778]
 [ 0.26641775]
 [-0.39221373]
 [-0.00538552]
 [-0.01764687]
 [-0.16632809]
 [-0.0822838 ]
 [-0.16644364]]
```

```
#b)
#Given a new array of features, you would multiply by the weights
#and then take the sign of the computed y_hat to classify it as happy or mad, {-1, 1}
yhat_real = Face_X@Face_w
yhat_sign = np.sign(yhat_real)
```

```
#c)
'''
Features 1-4 seem most important as they have the largest weights.
This indicates that they have the largest impact in making a clasification.
[[ 0.94366942]
 [ 0.21373778]
 [ 0.26641775]
 [-0.39221373]
 [-0.00538552]
 [-0.01764687]
 [-0.16632809]
 [-0.0822838 ]
 [-0.16644364]]
'''
```

```
#d)
# I would design a classifier based on features 1, 3, and 4 as their weights indicate they are most important
# in determining label. To build this you would cut the features matrix to be 128 x 3 instead of 128 x 9 and
# use a weights vector of 3 x 1.
```

```python
#e)
ys1 = Face_y[0:16]
ys2 = Face_y[16:32]
ys3 = Face_y[32:48]
ys4 = Face_y[48:64]
ys5 = Face_y[64:80]
ys6 = Face_y[80:96]
ys7 = Face_y[96:112]
ys8 = Face_y[112:128]
subset1 = Face_X[0:16]
subset2 = Face_X[16:32]
subset3 = Face_X[32:48]
subset4 = Face_X[48:64]
subset5 = Face_X[64:80]
subset6 = Face_X[80:96]
subset7 = Face_X[96:112]
subset8 = Face_X[112:128]

def CV(ys1,ys2,ys3,ys4,ys5,ys6,ys7,ys8,subset1,subset2,subset3,subset4,subset5,subset6,subset7,subset8):
    ys1_6 = np.vstack((ys1,ys2,ys3,ys4,ys5,ys6,ys8))
    ys1_5 = np.vstack((ys1,ys2,ys3,ys4,ys5,ys7,ys8))
    ys1_4 = np.vstack((ys1,ys2,ys3,ys4,ys6,ys7,ys8))
    ys1_3 = np.vstack((ys1,ys2,ys3,ys5,ys6,ys7,ys8))
    ys1_2 = np.vstack((ys1,ys2,ys4,ys5,ys6,ys7,ys8))
    ys1_1 = np.vstack((ys1,ys3,ys4,ys5,ys6,ys7,ys8))
    ys1_0 = np.vstack((ys2,ys3,ys4,ys5,ys6,ys7,ys8))
    ys1_7 = np.vstack((ys1,ys2,ys3,ys4,ys5,ys6,ys7))
    #ys1_7 = Face_y[0:112]
    y1_7m = np.matrix(ys1_7)
    y1_6m = np.matrix(ys1_6)
    y1_5m = np.matrix(ys1_5)
    y1_4m = np.matrix(ys1_4)
    y1_3m = np.matrix(ys1_3)
    y1_2m = np.matrix(ys1_2)
    y1_1m = np.matrix(ys1_1)
    y1_0m = np.matrix(ys1_0)
```

```python
    #subset1_7 = Face_X[0:112]
    subset1_7 = np.vstack((subset1,subset2,subset3,subset4,subset5,subset6,subset7))
    subset1_6 = np.vstack((subset1,subset2,subset3,subset4,subset5,subset6,subset8))
    subset1_5 = np.vstack((subset1,subset2,subset3,subset4,subset5,subset7,subset8))
    subset1_4 = np.vstack((subset1,subset2,subset3,subset4,subset6,subset7,subset8))
    subset1_3 = np.vstack((subset1,subset2,subset3,subset5,subset6,subset7,subset8))
    subset1_2 = np.vstack((subset1,subset2,subset4,subset5,subset6,subset7,subset8))
    subset1_1 = np.vstack((subset1,subset3,subset4,subset5,subset6,subset7,subset8))
    subset1_0 = np.vstack((subset2,subset3,subset4,subset5,subset6,subset7,subset8))
    x1_7m = np.matrix(subset1_7)
    x1_6m = np.matrix(subset1_6)
    x1_5m = np.matrix(subset1_5)
    x1_4m = np.matrix(subset1_4)
    x1_3m = np.matrix(subset1_3)
    x1_2m = np.matrix(subset1_2)
    x1_1m = np.matrix(subset1_1)
    x1_0m = np.matrix(subset1_0)

    x8m = np.matrix(subset8)
    x7m = np.matrix(subset7)
    x6m = np.matrix(subset6)
    x5m = np.matrix(subset5)
    x4m = np.matrix(subset4)
    x3m = np.matrix(subset3)
    x2m = np.matrix(subset2)
    x1m = np.matrix(subset1)

    w1 = np.linalg.inv(x1_7m.T@x1_7m)@x1_7m.T@y1_7m
    w2 = np.linalg.inv(x1_6m.T@x1_6m)@x1_6m.T@y1_6m
    w3 = np.linalg.inv(x1_5m.T@x1_5m)@x1_5m.T@y1_5m
    w4 = np.linalg.inv(x1_4m.T@x1_4m)@x1_4m.T@y1_4m
    w5 = np.linalg.inv(x1_3m.T@x1_3m)@x1_3m.T@y1_3m
    w6 = np.linalg.inv(x1_2m.T@x1_2m)@x1_2m.T@y1_2m
    w7 = np.linalg.inv(x1_1m.T@x1_1m)@x1_1m.T@y1_1m
    w8 = np.linalg.inv(x1_0m.T@x1_0m)@x1_0m.T@y1_0m
```

```
    y_hat1 = x8m @ w1
    y_hat1_sign = np.sign(y_hat1)
    avg1 = np.sum(y_hat1_sign != ys8) / 16

    y_hat2 = x7m @ w2
    y_hat2_sign = np.sign(y_hat2)
    avg2 = np.sum(y_hat2_sign != ys7) / 16

    y_hat3 = x6m @ w3
    y_hat3_sign = np.sign(y_hat3)
    avg3 = np.sum(y_hat3_sign != ys6) / 16

    y_hat4 = x5m @ w4
    y_hat4_sign = np.sign(y_hat4)
    avg4 = np.sum(y_hat4_sign != ys5) / 16

    y_hat5 = x4m @ w5
    y_hat5_sign = np.sign(y_hat5)
    avg5 = np.sum(y_hat5_sign != ys4) / 16

    y_hat6 = x3m @ w6
    y_hat6_sign = np.sign(y_hat6)
    avg6 = np.sum(y_hat6_sign != ys3) / 16

    y_hat7 = x2m @ w7
    y_hat7_sign = np.sign(y_hat7)
    avg7 = np.sum(y_hat7_sign != ys2) / 16

    y_hat8 = x1m @ w8
    y_hat8_sign = np.sign(y_hat8)
    avg8 = np.sum(y_hat8_sign != ys1) / 16

    CV = (avg1 + avg2 + avg3 + avg4 + avg5 + avg6 + avg7 + avg8) / 8

    return CV

print(CV(ys1,ys2,ys3,ys4,ys5,ys6,ys7,ys8,subset1,subset2,subset3,subset4,subset5,subset6,subset7,subset8))
```

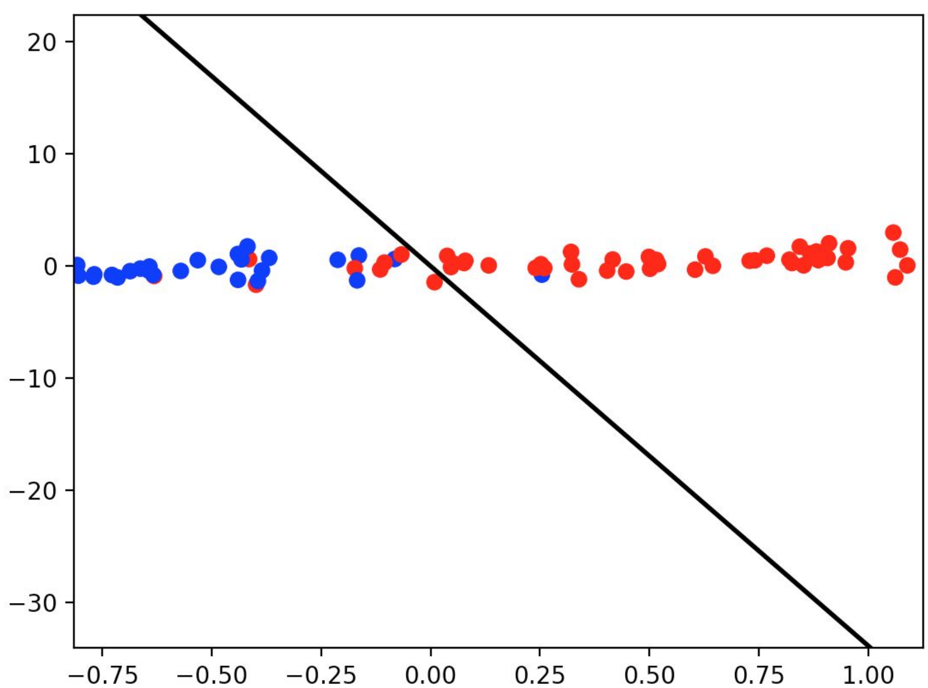
0.046875

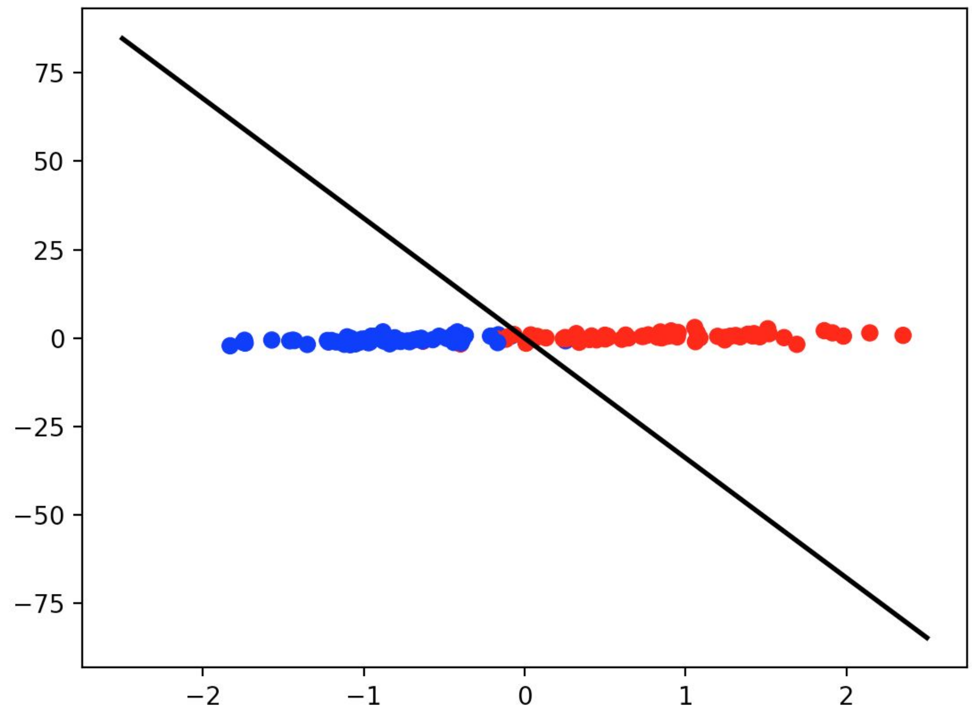
```
#f
ss1 = []
ss2 = []
ss3 = []
ss4 = []
ss5 = []
ss6 = []
ss7 = []
ss8 = []
for i in range(len(subset1)):
    ss1.append([subset1[i][0],subset1[i][2],subset1[i][3]])
    ss2.append([subset2[i][0],subset2[i][2],subset2[i][3]])
    ss3.append([subset3[i][0],subset3[i][2],subset3[i][3]])
    ss4.append([subset4[i][0],subset4[i][2],subset4[i][3]])
    ss5.append([subset5[i][0],subset5[i][2],subset5[i][3]])
    ss6.append([subset6[i][0],subset6[i][2],subset6[i][3]])
    ss7.append([subset7[i][0],subset7[i][2],subset7[i][3]])
    ss8.append([subset8[i][0],subset8[i][2],subset8[i][3]])

print(CV(ys1,ys2,ys3,ys4,ys5,ys6,ys7,ys8,ss1,ss2,ss3,ss4,ss5,ss6,ss7,ss8))
```

0.078125

The error rate using all 9 features (0.047) is slightly lower than the error rate using only 3 (0.078)

```
#g
data2points = []
for i in range(128):
    data2points.append([data['X'][i][0],data['X'][i][3]])
data2matrix = np.matrix(data2points)
data2w = np.linalg.inv(data2matrix.T@data2matrix)@data2matrix.T@Face_y

data2x = []
data2y = []
for n in data2points:
    data2x.append(n[0])
    data2y.append(n[1])
c = []
for i in Face_y:
    if i > 0:
        c.append('red')
    else:
        c.append('blue')
plt.scatter(data2x,data2y, c = c)
slope = np.float64(data2w[0] / data2w[1])
xpoints = np.linspace(-2.5,2.5,100)
y_comp = slope * xpoints
plt.plot(xpoints, y_comp, linewidth=2, color='black')
plt.show()
```





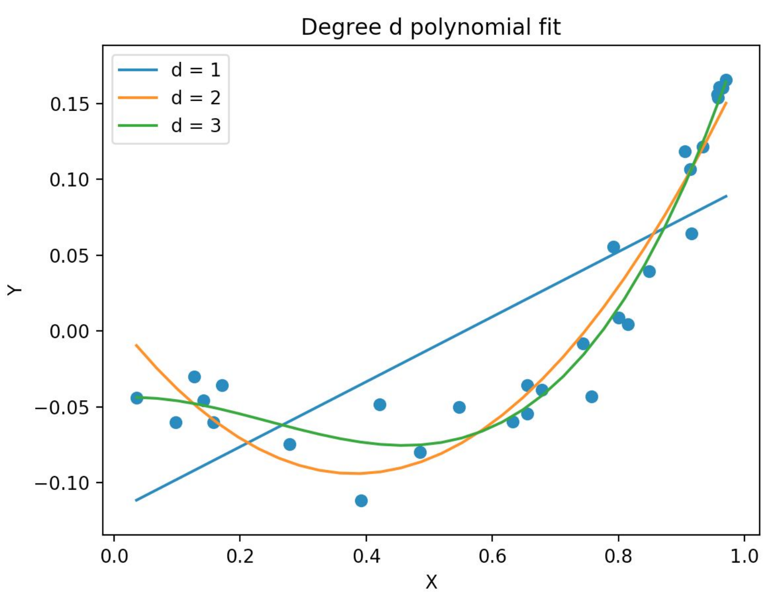
The plot looks fairly accurate, however there are some points which are classified incorrectly.

4\. a) $p(z) = y = \omega_0 1 + \omega_1 z_1 + \omega_2 z_2^2 + \omega_3 z_3^3 + \dots + \omega_d z_n^d$

b) $X_i = \begin{bmatrix} z_i^0 \\ z_i^1 \\ z_i^2 \\ z_i^3 \\ \vdots \\ z_i^d \end{bmatrix}$ $\quad X = \begin{bmatrix} 1 & z_1 & z_1^2 & \cdots & z_1^d \\ 1 & z_2 & z_2^2 & \cdots & \vdots \\ 1 & z_3 & z_3^2 & \cdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & z_n & z_n^2 & \cdots & z_n^d \end{bmatrix}$

$$X = \begin{bmatrix} \text{---} & X_1^T & \text{---} \\ \text{---} & X_2^T & \text{---} \\ & \vdots & \\ \text{---} & X_n^T & \text{---} \end{bmatrix} \qquad X \begin{bmatrix} \omega_0 \\ \omega_1 \\ \omega_2 \\ \vdots \\ \omega_d \end{bmatrix} = \begin{bmatrix} y_0 & y_1 & \dots & y_n \end{bmatrix}$$

```
#4) c)
labels = ['d = 1', 'd = 2', 'd = 3']
poly_data = loadmat('polydata.mat')
poly_X = poly_data['x']
poly_y = poly_data['y']
xvalues = np.linspace(min(poly_X[:,0]), max(poly_X[:,0]), 30)
plt.scatter(poly_X,poly_y)
for d in [1,2,3]:
    newA = []
    for i in range(d + 1):
        newA.append(poly_X ** i)
    newA = np.hstack(newA)
    Xwd = np.array(newA)
    poly_w = np.linalg.inv(Xwd.T@Xwd)@Xwd.T@poly_y
    temp = [xvalues ** i for i in range(d + 1)]
    mapped = [np.array([x]).T for x in temp]
    yvalues = np.hstack(mapped)@poly_w
    plt.plot(xvalues,yvalues, label = labels[d-1])

plt.title('Degree d polynomial fit')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.show()
```



d) Consider money continually gaining interest, Since it will grow exponentially the curve will not be linear, thus a polynomial fit will perform better than least squares.

To choose d simply test curves of different degrees. Choose the one that fits best.