HW 1- CMSC 25300

Bayard Walsh

March 2023

1

1.1 1 a)

First, we take the set of all columns in X to produce the following system of equations:
$a + b = 0$
$2a + b + c = 0$
$c + d = 0$
$-a + d = 0$
$0 = 0$
From here, we make the following substitutions:
$a = -b$
$2(-b) + b + c = 0$ or $-b + c = 0$, or $c = b$
$c = -d$
$d = a$
Then we have:
$a = -b$
$c = b$
$c = -d$, then $c = -a$, then $c = b$
$d = a$, then $d = -b$
Finally we have:
$a = -b$
$c = b$
$d = -b$
Of course, $b = b$. Therefore we can set b to any integer, and follow the system of equations above to form a 0 vector based on b. Therefore we cannot have all columns of X as the largest set of linearly independent columns.

Next we select columns 2,3,4 to form the following system of equations:
$a = 0$
$a + b = 0$
$b + c = 0$
$c = 0$
$0 = 0$

We have $a = c = 0$, and for b, we substitute so we have $(0) + b = 0$
$b + (0) = 0$
So $a = b = c = 0$, meaning the columns are linearly independent. As we know all 4 columns are not independent, and we have picked 3 columns which are, $\{2, 3, 4\}$ is a largest set of linearly independent columns.

1.2 1 b)

As we have rank(X)= the largest set of linear independent columns for X, as given, and we have the $|\{2, 3, 4\}| = 3$ is a largest set of linearly independent columns from **1 a)**, then we rank(X)=3

1.3 1 c)

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

$$X^T = \begin{bmatrix} 1 & 2 & 0 & -1 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

$$X^T X = \begin{bmatrix} 6 & 4 & 1 & -2 \\ 4 & 3 & 1 & 0 \\ 1 & 1 & 2 & 0 \\ -2 & 0 & 0 & 3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

We now have the following system of equations for $X^T X$:
$6a + 4b + c - 2d = 0$
$4a + 3b + c = 0$
$a + b + 2c = 0$
$-2a + 3d = 0$
From these equations, we will derive a series of equations below from the equations above:
$d = 2/3a$
$4(-2c - b) + 3b + c = 0$, so $-8c - 4b + 3b + c = 0$
$b = -7c$
$a + b - 2/7b = 0$
$a = -5/7b$
$4a + 3(-7c) + c = 0$

$a = 5c$
$6a + 4(-7c) + c - 2d = 0$
$6a + -27c - 2d = 0$
$6a + -27/5a - 2d = 0$
$3/5a - 2d = 0$
$3/10a = d$
At this point, we have $d = 2/3a = 3/10a$, and as $3/10 \neq 2/3$, and assuming that the system of equations above is true, we have $a = d = 0$
Now we solve for b, c. Using:
$a = -5/7b$
$b = -7c$
from above, we have $a = b = c = d = 0$, proving the matrix is linearly independent for all columns, meaning rank = 4 for matrix $X^T X$

2

2.1 2 a)

Yes, it is linearly independent. We have the following matrix

$$X = \begin{pmatrix} .63 & -.63 \\ -.63 & .63 \\ .63 & .63 \\ -.63 & -.63 \end{pmatrix}$$

To determine if the columns are linearly independent, we want to prove that the weight sum of each column vector is 0 only if every weight is 0. Therefore we will use the following to determine if $a_1 = a_2 = 0$ for all weighted sum possibilities for a 0 vector, which proves linear independence:

$$\begin{pmatrix} (a_1 \cdot .63) + (a_2 \cdot -.63) \\ (a_1 \cdot -.63) + (a_2 \cdot .63) \\ (a_1 \cdot .63) + (a_2 \cdot .63) \\ (a_1 \cdot -.63) + (a_2 \cdot -.63) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

We have the following system of equations
$.63a_1 - .63a_2 = 0$
$-.63a_1 + .63a_2 = 0$
$.63a_1 + .63a_2 = 0$
$-.63a_1 - .63a_2 = 0$
From the first equation we have $a_1 - a_2 = 0$, and from the third equation we have $a_1 + a_2 = 0$. Therefore we have $2a_1 = 0$, so $a_1 = 0$, and then $0 - a_2 = 0$, so $a_2 = 0$, so $a_1 = a_2 = 0$, so linear independence

2.2 2 b)

Yes. We have the following matrix

$$X = \begin{pmatrix} 1 & -1 & 1 \\ 1 & 1 & -1 \\ 1 & -1 & 0 \end{pmatrix}$$

To determine if the columns are linearly independent, we want to prove that the weight sum of each column vector is 0 only if every weight is 0. Therefore we will use the following to determine if $a_1 = a_2 = a_3 = 0$ for all weighted sum possibilities for a 0 vector, which proves linear independence:

$$\begin{pmatrix} (a_1) + (-a_2) + (a_3) \\ (a_1) + (a_2) + (-a_3) \\ (a_1) + (-a_2) + 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

Therefore from the top row, we have $a_1 - a_2 + a_3 = 0$, and from the bottom row we have $a_1 - a_2 = 0$
This implies that $a_1 - a_2 + a_3 - (a_1 - a_2) = 0$, meaning that $a_3 = 0$.
Next from the middle row we have $a_1 + a_2 - a_3 = 0$.
We substitute in $a_3 = 0$, which gives us $a_1 + a_2 = 0$.
Next from the bottom row we have $a_1 - a_2 = 0$, so we have $2a_1 = 0$, which also implies $a_1 = 0$, as we have $a_1 - a_2 = a_1 - 0 = 0$. So we have $a_1 = a_2 = a_3 = 0$ for a weighted sum to obtain the column vector 0, which implies the matrix is linearly independent.

2.3 2 c)

No. We can perform the following calculation, which shows linear dependence.

$$2x_1 = x_2 + x_3$$

$$2 \times \begin{bmatrix} 1 \\ 3 \\ 8 \end{bmatrix} = \begin{bmatrix} 2 \\ 5 \\ 13 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ 3 \end{bmatrix}$$

As we can form this relation from columns in X, we know that the matrix is not linearly independent

2.4 2 d)

We have the following system of equations:
$2a + 4b = 0$
$-8a + 12b = 0$
$4a + 8b = 0$

Therefore we have $a = -2b$, so we have $-8(-2b) + 12b = 0 = b$. So we have $a = -2(0)$, so $a = b = 0$, meaning we have rank 2. As rank is $\leq min(p, n)$, where $x^{p \cdot n}$, and we have $x^{3 \cdot 2}$, we know that the rank cannot be larger then 2. Therefore its rank is 2.

3

3.1 3 a)

$\nabla_w f = 3x$

3.2 3 b)

$\nabla_w f = 2w - x^T - x$

3.3 3 c)

$\nabla_w f = x^T \cdot \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$

3.4 3 d)

$\nabla_w f = \left(\begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix} + \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix} \right) w = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} w$

3.5 3 e)

$\nabla_w f = \left(\begin{bmatrix} 1 & 3 \\ 3 & 9 \end{bmatrix} + \begin{bmatrix} 1 & 3 \\ 3 & 9 \end{bmatrix} \right) w = \begin{bmatrix} 2 & 6 \\ 6 & 18 \end{bmatrix} w$

4

4.1 4 a)

```python
import scipy.io as sio
import numpy as np
import matplotlib.pyplot as plt

##### Part a #####
# load the training data X and the training labels y
matlab_data_file = sio.loadmat('face_emotion_data.mat')
X = matlab_data_file['X']
y = matlab_data_file['y']
# n = number of data points # p = number of features
n, p = np.shape(X)
# Solve the least-squares solution. w is the list of # weight coefficients
w=np.dot((np.linalg.inv(np.dot(X.T,X))), np.dot( X.T,y))


spl_x=np.array_split(X, 8) # split X into 8 equal subsets, with ordering 1-16,
# 17-32 etc.
spl_y=np.array_split(y, 8) # split Y into 8 equal subsets, with ordering 1-16,
# 17-32 etc.

overallerror=0 # global error val

# helper function, takes predicted classification vector, "true" classification
# vector, and the var as the classification point (fw >= var)
# designed for a subset of size 16, implimented below
def test_accuraccy (pred,truth,var):
    accur_count=0 # correctly classified label
    for i in range(16):
        # compute classification based on correct smiling or not smiling
        if ((pred[i] >= var) and (truth[i]==1)) or ((pred[i] < var) and (truth[i]==-1)):
            accur_count+=1
    return (16-accur_count)/16 # return error rate for given subset size 16

# testing accuracy for all 9 features
for i in range(8):
    test_x=spl_x[i] # select ith index of split to test
    test_y=spl_y[i] # select ith index of split to test
    train_x=np.concatenate(spl_x[:i]+spl_x[(i+1):]) # combined other 7 sets for x
    train_y=np.concatenate(spl_y[:i]+spl_y[(i+1):]) # combined other 7 sets for y
    # calculate wp based on the 7 other sets, least squares
    wp=np.dot((np.linalg.inv(np.dot(train_x.T,train_x))), np.dot( train_x.T,train_y))
    pred=np.dot(test_x,wp) # use wp to predict labels for non test sets
    overallerror+=test_accuraccy(pred,test_y,-0.3) # increment error rate for subset
print(overallerror/8) # average overall error rate
```

4.2 4 b)

In order to classify a new face as smiling or non smiling, first use the feature-extraction method to compute F, where F is a single feature vector for the new face (given by problem). Therefore, we would have column vector F with degree 9, based on the 9 features given by the training data. From there, apply predicted weight vector W found in **4 a)**, so that we have Fw, which would be a single value. From there, we would apply the following formula:

$$Fw = \begin{cases} \text{smiling,} & \text{if } Fw \geq 0 \\ \text{not smiling,} & \text{otherwise} \end{cases}$$

Note: In the rare case of $FW = 0$, could classify not smiling, our predictor formula is somewhat arbitrary for deciding $\geq$ over $>$. Also choosing negative/ positive as classifier somewhat arbitrarily but simple solution

4.3 4 c)

We have calculated weight vector w as follows

$$w = \begin{bmatrix} 0.94366942 \\ 0.21373778 \\ 0.26641775 \\ -0.39221373 \\ -0.00538552 \\ -0.01764687 \\ -0.16632809 \\ -0.0822838 \\ -0.16644364 \end{bmatrix}$$

As w is a scalar, the most important features will be those which cause the estimation to skew the most. Therefore, we can rank based on highest $|w|$ to lowest. Therefore, the most important features are ranked as follows, and we can chose the most important ones as the highest ranking features:
$w_1, w_4, w_3, w_2, w_9, w_7, w_8, w_6, w_5$

4.4 4 d)

Yes, I could design a classifier, though with less features the accuracy would likely decrease. Using the ranking from before, I would chose a classifier with the weights w_1, w_4, w_3 from the original w as those had the highest magnitude and would impact the classification the most. In terms of applying a formula, I would pick the same one above (in terms of checking if Fw is positive). Another strategy could be considering the overall feature weights lost and adjusting from there. Because w is negative for 6 of its 9 features and I the picked the 3 with the largest magnitude, 2 of which are positive, maybe I would use $Fw \geq -.2$ for classifying as smiling instead of $Fw \geq 0$ to account for potential skew towards smiling classification with less features. This is implemented in my model with the variable "var" in my testaccuracy function, as I wanted to see if slightly adjusting the constant from 0 leads to an improvement in accuracy. Another note is that **classification** of features is done based on those 3 features as well, which is implemented in my code. That is, for the second section, I only considered the columns with the greatest magnitude and computed w' based on that, meaning that the weights in the second function are different then just picking the 3 highest weights from w with 9 features. This is done by narrowing down the data in X to X_p through the rows function.

4.5 4 e)

```python

# question 4 f)
# testing accuracy for 3 features- pick row 1 3 and 4 (note -1 for index)
rows = [0, 2, 3]

# Make X_p, by selecting the specific rows from the data array
X_p = X[:, rows]

spl_xp=np.array_split(X_p, 8) # split X into 8 equal subsets, with ordering 1-16
seconderror=0 # new global error val

for i in range(8):
    test_x=spl_xp[i] # select ith index of split to test
    test_y=spl_y[i] # select ith index of split to test
    train_x=np.concatenate(spl_xp[:i]+spl_xp[(i+1):]) # combined other 7 sets for x
    train_y=np.concatenate(spl_y[:i]+spl_y[(i+1):]) # combined other 7 sets for y
    # calculate wp based on the 7 other sets
    wp=np.dot((np.linalg.inv(np.dot(train_x.T,train_x))), np.dot( train_x.T,train_y))
    pred=np.dot(test_x,wp)
    seconderror+=test_accuraccy(pred,test_y,-0.25) # increment error rate for subset
    # this function call is identical to the one in 4a
print(seconderror/8) # average overall error rate
```

4.6 4 f)

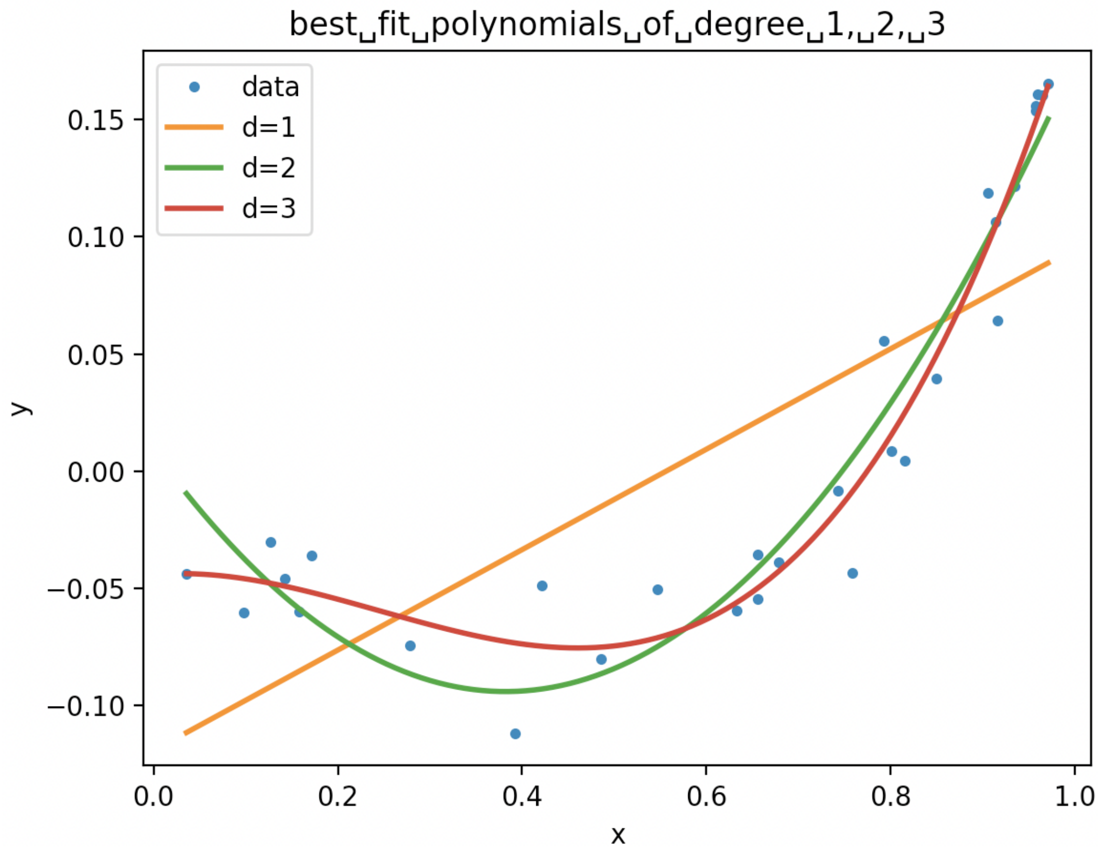
Error rate for 9 features, where $Fw \geq -.3$ **is classifier** : 0.0234375

Error rate for 3 features $Fw \geq -.25$ **is classifier**: 0.046875

note: in $Fw \geq x$ *above,* x *is picked through running the error rate calculation with enough trial and error, looking for smallest error rate. Smaller error rate could be possible with better* x*, but these rough* x *give solid estimate of error rate.*

5

```python
#5 polynomial fitting
# NOTE: libraries (numpy etc.) are loaded earlier in file

# load x and y vectors
data = sio.loadmat('polydata.mat')
x = data['x']
y = data['y']
# n = number of data points

# N = number of points to use for interpolation
# z_test = points where interpolant is evaluated
# p = array to store the values of the interpolated polynomials
n = x.size
N = 100
z_test = np.linspace(np.min(x), np.max(x), N)
p = np.zeros((3, N))

X=np.array(np.full((n,1),1)) # create array of 1s

for d in [1, 2, 3]:
    # generate X-matrix for this choice of d
    a=np.power(x,d)
    X = np.concatenate([X, a], axis = 1)
    # solve least-squares problem. w is the list
    # of polynomial coefficients
    w=np.dot((np.linalg.inv(np.dot(np.transpose(X),X))), np.dot( X.T,y))

    # solve least-squares problem. w is the list
    # of polynomial coefficients
    # evaluate best -fit polynomial at all points z_test ,
    # and store the result in p
    # NOTE (optional): this can be done in one line
    # with the polyval command!
    p[d-1]=np.polyval(w[::-1],z_test)
# plot the datapoints and the best-fit polynomials
plt.plot(x, y, '.', z_test , p[0, :], z_test , p[1, :], z_test , p [2, :], linewidth=2)
plt.legend(['data', 'd=1', 'd=2', 'd=3'], loc='upper left')
plt.title('best␣fit␣polynomials␣of␣degree␣1,␣2,␣3')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```


best␣fit␣polynomials␣of␣degree␣1,␣2,␣3
data
d=1
d=2
d=3
0.15
0.10
0.05
0.00
−0.05
−0.10
y
0.0
0.2
0.4
0.6
0.8
1.0
x