1\. a) $\begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \Rightarrow \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \Rightarrow \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$

$\therefore$ largest set $\begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$

b) $\text{rank}(X) = 4$

c) X is non-singular matrix

$\text{rank}(X^T X) = 4$

2\. a) $X = 0.63\ X \begin{pmatrix} 1 & -1 \\ -1 & 1 \\ 1 & 1 \\ -1 & -1 \end{pmatrix}$ $\begin{pmatrix} 1 & -1 \\ -1 & 1 \\ 1 & 1 \\ -1 & -1 \end{pmatrix} \Rightarrow \begin{pmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{pmatrix}$

$\therefore$ Yes they're linearly independent

b) $\begin{pmatrix} 1 & -1 & 1 \\ 1 & 1 & -1 \\ 1 & -1 & 0 \end{pmatrix} \Rightarrow \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & -2 \\ 1 & 0 & -1 \end{pmatrix} \Rightarrow \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$

$\therefore$ Yes

c) $\begin{pmatrix} 1 & 2 & 1 \\ 3 & 5 & 4 \\ 8 & 13 & 11 \end{pmatrix} \Rightarrow \begin{pmatrix} 1 & 0 & 0 \\ 3 & -1 & 1 \\ 8 & -3 & 3 \end{pmatrix} \Rightarrow \begin{pmatrix} 1 & 0 & 0 \\ 3 & -1 & 0 \\ 8 & -3 & 0 \end{pmatrix}$

$\therefore$ No.

d) $\begin{pmatrix} 2 & 3 \\ -8 & 12 \\ 4 & -6 \end{pmatrix} \Rightarrow \begin{pmatrix} 2 & 3 \\ 0 & 1 \\ 0 & 0 \end{pmatrix} \Rightarrow \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix}$

$\therefore$ $\text{rank}(X) = 2$

3. a) $f(w) = w^T(3x)$

$\nabla_w f = 3x$

b) $f(w) = (w-x)^T(w-x)$

$$\nabla_w f = \frac{\partial f(w)}{\partial(w-x)} \cdot \frac{\partial(w-x)}{\partial(w)}$$

$$= 2(w-x)$$

c) $f(w) = x^T \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} w$

$$\nabla_w f = \left(x^T \left(\begin{matrix} 1 & 2 \\ 3 & 4 \end{matrix}\right)\right)^T = \begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix} \cdot x$$

d) $f(w) = w^T \begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix} w$

$$\nabla_w f = \left[\begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix} + \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix}\right] = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix} w = 2w$$

e) $f(w) = w^T \begin{pmatrix} 1 & 3 \\ 3 & 9 \end{pmatrix} w$

$$\nabla_w f = \left[\begin{pmatrix} 1 & 3 \\ 3 & 9 \end{pmatrix} + \begin{pmatrix} 1 & 3 \\ 3 & 9 \end{pmatrix}\right] w = 2 \cdot \begin{pmatrix} 1 & 3 \\ 3 & 9 \end{pmatrix} w$$

4. b) step1. use feature-extraction methods to create test data X

step2. for each X:

calculate Xw

if $Xw \geq 0$ then classify it as smiling face

else classify it as a non-smiling face.

c). I think the feature with the largest absolute weight value seems to be the most important. For it would contribute the most in the classification. Empirically speaking, the weight value of all the features are randomly initialized in the beginning. And then, in the gradient decent the feature which affects classifcation the most would get the largest gradient for its weight. After all the epochs, this feature's weight value is changed to be the largest.

d) If I could do experiment, I would choose the features by randomly picking up 3 features and masking all the others, then selecting the group with the least derivation $|Wx-y|_2$.

If I could not, I would simply choose 3 features with top 3 absolute weight value.

To build a classifier:

step1. randomly divide the training set and test set (80% : 20%)

Step2. get the weight for training set with least square.

Step3. Test the error rate:

for each (x, y) test set:

calculate Xw

if $Xw \cdot y \geq 0$: right

else: wrong

$$\text{error rate} = \frac{\text{wrong samples}}{\text{total samples}} \times 100\%$$

In [1]:

```
import scipy . io as sio
import numpy as np
# #### 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


X_=np.hstack((X,[[1.]for i in range(n)]))

np.random.seed(1234)
w_init=np.random.normal(0,1,size=(p+1,1))

lr=0.0001
sigma=1e-8

def dw(x,y,w):
    return 2*x.T.dot(x.dot(w)-y)

def grad_least_square(x,y,lr,w_init,sigma):
    w=w_init
    i=0
    while 1:
        dw_cur=dw(x,y,w)
        w_hat=w-2*lr*dw_cur
        sigma_hat=np.sum((w_hat-w)**2)
        #print(np.sum((y-X_.dot(w))**2))
        #print(sigma_hat)
        #print()
        if sigma_hat<sigma:
            w=w_hat
            break
        w=w_hat
    return w

w=grad_least_square(X_,y,lr,w_init,sigma)
print(w)
```

```
[[ 0.94128699]
 [ 0.1292234 ]
 [ 0.29656433]
 [-0.33543203]
 [-0.00827229]
 [-0.02106024]
 [-0.16532586]
 [-0.0785805 ]
 [-0.16675397]
 [ 0.078125  ]]
```

In [2]:

```
#4e
def estimate_error(x,y,w):
```

```
    y_hat=x.dot(w)
    y_hat=np.where(y_hat <= 0, -1, 1)
    #print(y_hat)
    #print(y)
    return 1.-np.sum(y_hat==y)/y.shape[0]

lr=0.00001
sigma=1e-8

error=0
for i in range(8):
    hold_out_X=X_[16*i:16*(i+1)]
    hold_out_y=y[16*i:16*(i+1)]
    train_X=np.concatenate((X_[0:16*i],X_[16*(i+1):]))
    train_y=np.concatenate((y[0:16*i],y[16*(i+1):]))

    np.random.seed(1234)
    w_init=np.random.normal(0,1,size=(p+1,1))

    w=grad_least_square(train_X,train_y,lr,w_init,sigma)
    #print(w)
    error+=estimate_error(hold_out_X,hold_out_y,w)

print(error/8.)
```

```
0.0390625
```

In [3]:

```
#4f
#9features

fature_index=[i for i in range(9)]
X_new=X_[:,fature_index+[9]]
#X_new=X_
lr=0.00001
sigma=1e-9


hold_out_X=X_new[:32]
hold_out_y=y[:32]
train_X=X_new[32:]
train_y=y[32:]
error=0

np.random.seed(1234)
w_init=np.random.normal(0,1,size=(10,1))

w=grad_least_square(train_X,train_y,lr,w_init,sigma)
cur_err=estimate_error(hold_out_X,hold_out_y,w)
#print(cur_err)
error+=cur_err

print(error)
```

```
0.0625
```

In [4]:

```
#4f
#3features

fature_index=[0,2,3]
```

```
X_new=X_[:,fature_index+[9]]

lr=0.00001
sigma=1e-8


hold_out_X=X_new[:32]
hold_out_y=y[:32]
train_X=X_new[32:]
train_y=y[32:]
error=0

np.random.seed(1234)
w_init=np.random.normal(0,1,size=(4,1))

w=grad_least_square(train_X,train_y,lr,w_init,sigma)
cur_err=estimate_error(hold_out_X,hold_out_y,w)
#print(cur_err)
error+=cur_err

print(error)
```

```
0.25
```

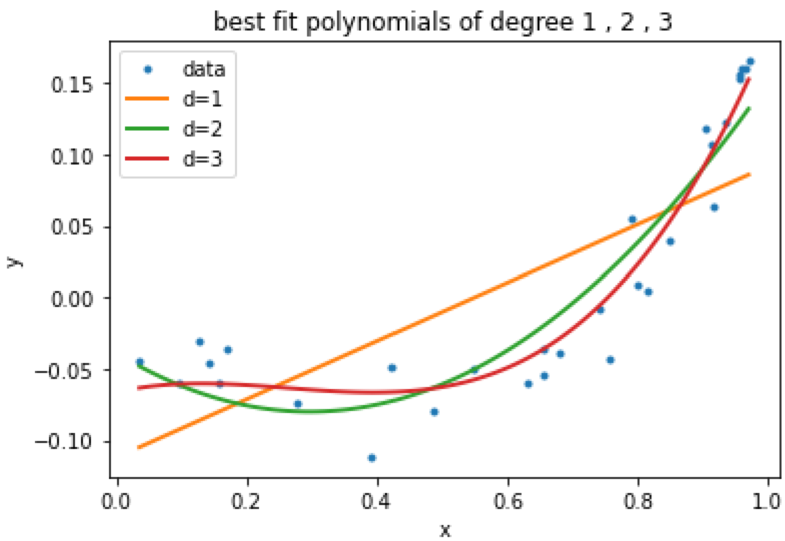
In [5]:

```
import numpy as np
import scipy . io as sio
import matplotlib . pyplot as plt
# Load x and y vectors
d = sio.loadmat ('polydata.mat')
z = d ['x']
y = d ['y']
# n = number of data points
# N = number of points to use for interpolation
# z = points where interpolant is evaluated
# p = array to store the values of the interpolated polynomials
n = z.size
N = 100
z_test= np.linspace(np.min(z),np.max(z),N)
p = np.zeros ((3,N))
np.random.seed(1234)
lr=0.001
sigma=1e-8

i=0
for d in [1,2,3]:
    x_train=np.array([[z_[0]**i for i in range(d+1)] for z_ in z])
    x_test=np.array([[z_**i for i in range(d+1)] for z_ in z_test])
    w_init=np.random.normal(0,1,size=(d+1,1))
    w=grad_least_square(x_train,y,lr,w_init,sigma)
    p[i]+=x_test.dot(w).reshape(-1)
    i=i+1


# generate X- matrix for this choice of d
# 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 !
# plot the datapoints and the best -fit polynomials
plt.plot(z,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 ()
```




```
In [ ]:
```