In [2]:
```
import numpy as np
from scipy.io import loadmat
from sklearn.datasets import make_regression
from sklearn.linear_model import LinearRegression

in_data = loadmat('face_emotion_data.mat')
print([key for key in in_data])

X = in_data['X']
y = in_data['y']
```

```
['__header__', '__version__', '__globals__', 'y', 'X']
```

a

Use the training data X and y and a least squares problem to train your classier weights.

In [3]:
```
w = np.linalg.lstsq(X, y, rcond=None)[0]
print("w = \n", w)
```

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

b

Suppose there is a new face with feature vectors xi = [x1i x2i ... x9i]^T Compute for label yi = xi^T @ w. If yi > 0, then it will be classified as a happy face; it yi < 0, then it will be classfied as an angry face.

c

Feature realted to the first column of X seems to be most important, since its corresponding weight, w1, has the largest absolute value among all weights, meaning that the feautre will have the largest effect on the value of its predicted label.

d

To design a classifier based on three of the nine features, choosing column 1, 4, and 3 can best represent the original matrix, since their respective weights have the highest 3 absolute values.

e

In [4]:

```
# assessing performance using all 9 features.
y_head = X @ w
counter = 128;
for i in range(0, 128):
    if (y[i] > 0 and y_head[i] > 0) or (y[i] <= 0 and y_head[i] <= 0):
        counter-=1
print("error rate using 9 features: ", counter/128)

# assessing performance using only 3 features
X2 = np.hstack([X[:, 0:1], X[:, 2:3], X[:, 3:4]])
w2 = np.linalg.lstsq(X2, y, rcond=None)[0]
y2_head = X2 @ w2
counter = 128
for i in range(0, 128):
    if (y[i] > 0 and y2_head[i] > 0) or (y[i] <= 0 and y2_head[i] <= 0
):
        counter-=1
print("error rate using 3 features: ", counter/128)
```

```
error rate using 9 features:  0.0234375
error rate using 3 features:  0.0625
```

In [5]:
```
SETS = 8
SAMPLE_SIZE = 16
# [t*16 :(t+1)*16 - 1]
miss_rate = np.array(np.ones(SETS))
for t in range(0, SETS):
    X3 = np.delete(X, slice(t*16, (t+1)*16), 0)
    y3 = np.delete(y, slice(t*16, (t+1)*16), 0)
    w3 = np.linalg.lstsq(X3, y3, rcond=None)[0]
    y3_head = X3 @ w3
    counter = 0
    for i in range(0, 127-SAMPLE_SIZE):
        if (y3[i] > 0 and y3_head[i] <= 0) or (y3[i] <= 0 and y3_head[
i] > 0):
            counter+=1
    miss_rate[t] = counter / SAMPLE_SIZE
print("error rate = ", np.average(miss_rate))
```

```
error rate =  0.171875
```

In [ ]:

In [ ]: