In [20]:
```
import numpy as np
from scipy.io import loadmat
import matplotlib.pyplot as plt

in_data = loadmat('face_emotion_data.mat')
print([key for key in in_data]) # -- use this line to see the keys in the di

# m(faces) x n(features) = y
```

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

In [21]:
```
#1a)

# Using the formula: w = (X^T X)^(-1) X^T y
X_transpose = X.transpose()
w = np.linalg.inv(X.transpose()@ X) @ X.transpose() @ y
w
```

Out[21]:
```
array([[ 0.94366942],
       [ 0.21373778],
       [ 0.26641775],
       [-0.39221373],
       [-0.00538552],
       [-0.01764687],
       [-0.16632809],
       [-0.0822838 ],
       [-0.16644364]])
```

In [22]:
```
#1b)
# Each weight corresponds to one of the 9 features the model takes
# when we solve y=X^Tw, each weight will be applied to its associated measur
```

In [23]:
```
#1c)
#The features that seem to be most important are the ones who have the highe
#all the features have been normalized, they are on the same scale. Features
```

In [24]:
```
#1d)
selected_columns = [0, 2, 3]
x_slice = X[:, selected_columns]
w2 = np.linalg.inv(x_slice.transpose()@ x_slice) @ x_slice.transpose() @ y
#If we are minimizing the features we want to use we should include the ones
#the three expressed above
w
```

Out[24]:
```
array([[ 0.94366942],
       [ 0.21373778],
       [ 0.26641775],
       [-0.39221373],
       [-0.00538552],
       [-0.01764687],
       [-0.16632809],
       [-0.0822838 ],
       [-0.16644364]])
```

In [36]:
```
#1e)
y_hat1 = np.sign(X@w)
y_hat2 = np.sign(x_slice@w2)

error_vec1 = [0 if i[0]==i[1] else 1 for i in np.hstack((y_hat1, y))]
error_vec2 = [0 if i[0]==i[1] else 1 for i in np.hstack((y_hat2, y))]

print("Percent error for 9 features: {}%. Percent error for 3 features: {}%.
```

```
Percent error for 9 features: 2.34%. Percent error for 3 features: 6.25%.
```

In [42]:
```
#1d)
num_subsets = 8
subset_size = len(X) // num_subsets
error_rates = []

#8folds cross validation
for fold in range(num_subsets):
    start_index = fold * subset_size
    end_index = (fold + 1) * subset_size

    X_train = np.concatenate((X[:start_index], X[end_index:]), axis=0)
    y_train = np.concatenate((y[:start_index], y[end_index:]), axis=0)
    X_holdout = X[start_index:end_index]
    y_holdout = y[start_index:end_index]

    predictions = X_holdout @ w
    misclassifications = np.sum(np.sign(predictions) != y_holdout)
    error_rate = misclassifications / len(X_holdout)
    error_rates.append(error_rate)


average_error_rate = np.mean(error_rates)
print("Average Error Rate: {}%.".format(round(average_error_rate*100, 2)) )
```

```
Average Error Rate: 2.34%.
```

In [ ]: