In [52]:
```python
import numpy as np
from scipy.io import loadmat
import matplotlib.pyplot as plt

in_data = loadmat('face_emotion_data.mat')
#loadmat() loads a matlab workspace into a python dictionary, where the names of the var
#in the dictionary.  To see what variables are loaded, uncomment the line below:
#print([key for key in in_data])

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

In [43]:
```python
### ECE 532 Assignment #4 - Devin Bresser ###

# Problem 1a #
# least squares solution to X w = y:
# w_min = X (X^T X)^-1 X^T y
# (the projection of y onto the space spanned by X)

# X is m x n, m=128, n=9
# (128 face images, 9 features)

X_proj = np.linalg.inv(X.T @ X) @ X.T
w_min = X_proj @ y
w_min
```

Out[43]:
```
array([[ 0.94366942],
       [ 0.21373778],
       [ 0.26641775],
       [-0.39221373],
       [-0.00538552],
       [-0.01764687],
       [-0.16632809],
       [-0.0822838 ],
       [-0.16644364]])
```

In [23]:
```python
# Problem 1b #
# To classify a new image as happy or angry, we must first
# extract the n=9 relevant features from the new image.
# Call this x_new ^ T
# Then, we can multiply the image x_new ^ T * w_min = y_pred
# This dot product will give us a scalar value,
# and the sign of it will tell us if the face is happy (+) or sad (-)
# more specifically, if the sign is positive we assign y = 1 (happy label)
# and if the sign is negative we assign y = -1 (sad label)
```

In [47]:
```python
# Problem 1c #
w_min

# Because the columns of X have been normalized to have the same l2 norm,
# the magnitudes of the weights can be compared directly
# and that will tell us some useful information about the feature importance.
# In this case, looking at the absolute values of the w_min vector,
# the most important features seem to be:
# |w_min[0]| = 0.944
# |w_min[3]| = 0.392
# |w_min[2]| = 0.266

# w_min[4] gets a (dis)honorable mention for having almost no impact
# on the classifier.
```

Out[47]:
```
array([[ 0.94366942],
       [ 0.21373778],
```

```
       [ 0.26641775],
       [-0.39221373],
       [-0.00538552],
       [-0.01764687],
       [-0.16632809],
       [-0.0822838 ],
       [-0.16644364]])
```

In [ ]:

```
# Problem 1d #
# I would choose the three features associated with the weights described
# above, namely:
# 1st column of x (weight: w_min[0] = 0.944)
# 4th column of x (weight: w_min[3] = -0.392)
# 3rd column of x (weight: w_min[2] = 0.266)

# 1.) Isolate only these three features from the training matrix X.
# Our X-matrix goes from nine columns to three.

# 2.) Re-compute w_min for the new trimmed X-matrix

# 3.) When a new data image, is to be classified,
# Measure the three features and store them in x^T.

# 4.) Compute x^T . w = y_pred for the new image x^T.

# 5.) If y_pred is positive, assign y_pred_binary = +1 to the image
# This means we classified the face as "happy".
# If y_pred is negative, assign y_pred_binary = -1 to the image.
# This means we classified the face as "sad".

# See below cell for the execution of this design:
```

In [155…

```
# Problem 1d continued #

# Isolate column index 0, 2 and 3 of x (1st, 3rd, and 4th columns)
X_trim = np.hstack((X[:, 0:1], X[:, 2:3], X[:, 3:4]))

# Re-compute w_min parameters with X_trim
X_trim_proj = np.linalg.inv(X_trim.T @ X_trim) @ X_trim.T
w_min_trim = X_trim_proj @ y

# Write a function that outputs the label for a new data point
def classify_happy_sad(_x, _w):
    """
    input: _x, a stack of row vectors of features of a new image
    output: y_pred, a predicted label, -1 (sad) or +1 (happy)

    """
    y_stack = np.zeros((len(_x), 1))

    for a in range(len(_x)):
        if(_x[a] @ _w >= 0):
            y_stack[a] += 1
        if(_x[a] @ _w < 0):
            y_stack[a] -= 1
    return y_stack
```

In [158…

```
# Problem 1e #
# Run all training data through both models:
y_pred9 = classify_happy_sad(X, w_min)
y_pred3 = classify_happy_sad(X_trim, w_min_trim)

misses9 = 0
for a in range(len(y)):
    if(y_pred9[a] != y[a]):
```

```
        misses9+=1

misses3 = 0
for a in range(len(y)):
    if(y_pred3[a] != y[a]):
        misses3+=1


print(f"Classification error with 9 features: {misses9/len(y)}")
print(f"Classification error with 3 features: {misses3/len(y)}")
```

```
Classification error with 9 features: 0.0234375
Classification error with 3 features: 0.0625
```

In [227…

```
# Problem 1f #

# create the slices using some beautiful hard coding
X_slice0 = X[0:16, :]
X_slice1 = X[16:32, :]
X_slice2 = X[32:48, :]
X_slice3 = X[48:64, :]
X_slice4 = X[64:80, :]
X_slice5 = X[80:96, :]
X_slice6 = X[96:112, :]
X_slice7 = X[112:128, :]

y_slice0 = y[0:16, :]
y_slice1 = y[16:32, :]
y_slice2 = y[32:48, :]
y_slice3 = y[48:64, :]
y_slice4 = y[64:80, :]
y_slice5 = y[80:96, :]
y_slice6 = y[96:112, :]
y_slice7 = y[112:128, :]

# store the slices in a list
X_slices = [X_slice0, X_slice1, X_slice2, X_slice3, X_slice4, X_slice5, X_slice6, X_slic
y_slices = [y_slice0, y_slice1, y_slice2, y_slice3, y_slice4, y_slice5, y_slice6, y_slic

# create the possible 7-slice combos X
X_combinations = []
for i in range(8):
    combination = X_slices[:i] + X_slices[i+1:]
    stacked_combination = np.vstack(combination)
    X_combinations.append(stacked_combination)

# # create the possible 7-slice combos y
# y_combinations = []
# for i in range(8):
#     combination = y_slices[:i] + y_slices[i+1:]
#     stacked_combination = np.vstack(combination)
#     y_combinations.append(stacked_combination)
```

In [247…

```
# Problem 1f continued #

# compute w_min parameters for each 7-length slice stack
w_min_combinations = [(np.linalg.inv(X_slice.T @ X_slice) @ X_slice.T) @ y_slice for X_s

# test each w_min_combinations on the missing slice
# for the first X_combinations[0], the missing slice is X_slice0
# the parameters are w_min_combinations[0]
# and the outputs to be computed are y_pred_slice1

# for the second X_combinations[1], the missing slice is X_slice1, etc.
```

```python
# classify each missing slice based upon the other 7 slices
y_pred_slice0 = classify_happy_sad(X_slice0, w_min_combinations[0])
y_pred_slice1 = classify_happy_sad(X_slice1, w_min_combinations[1])
y_pred_slice2 = classify_happy_sad(X_slice2, w_min_combinations[2])
y_pred_slice3 = classify_happy_sad(X_slice3, w_min_combinations[3])
y_pred_slice4 = classify_happy_sad(X_slice4, w_min_combinations[4])
y_pred_slice5 = classify_happy_sad(X_slice5, w_min_combinations[5])
y_pred_slice6 = classify_happy_sad(X_slice6, w_min_combinations[6])
y_pred_slice7 = classify_happy_sad(X_slice7, w_min_combinations[7])
y_pred_slices = [y_pred_slice0, y_pred_slice1, y_pred_slice2, y_pred_slice3, y_pred_slic


errors = np.zeros(8)
for a in range(8):
    for b in range(len(y_pred_slice1)):
        if(y_pred_slices[a][b] != y_slices[a][b]): errors[a]+=1
print(f"test error for each 16-length slice: \n{errors/16}")
print(f"Average test error: {np.sum(errors/16) / 8}")
```

```
test error for each 16-length slice: 
[0.0625 0.0625 0.125  0.0625 0.     0.0625 0.     0.    ]
Average test error: 0.046875
```

In [ ]: