Part a)

```
import numpy as np
from scipy.io import loadmat

in_data = loadmat("face_emotion_data.mat")
X = in_data['X']
y = in_data['y']

# Part a
# using w = (XTX)^-1XTy
weights = np.linalg.inv(np.transpose(X)@X)@np.transpose(X)@y
print("Weights: \n", weights)
# predictor
y_hat = np.sign(X@weights)
```

```
Weights:
 [[ 0.94366942]
 [ 0.21373778]
 [ 0.26641775]
 [-0.39221373]
 [-0.00538552]
 [-0.01764687]
 [-0.16632809]
 [-0.0822838 ]
 [-0.16644364]]
```

Part b)

Use these weights to predict the emotion on a new face by measuring the 9 distances on the new face and arranging them into a feature vector in the same order as the training data feature vectors, and multiplying the feature vector by the vector of weights using the equation $y = x^T w$. Then if the resulting y is > 0 then the face is classified as happy and if $y < 0$, the face is classified as angry.
In summary, +1 (happy) or -1 (angry) is the resulting class based on $y = sign(x^T w)$ where w is the weights found above and x is the vector of 9 measured distances on the new face.

Part c)

Since these 9 features have been normalized, we can say that the features with larger weights are what is contributing the most to the result. Therefore larger weights means more important features. The largest weights are with features 1, 3, and 4 (indices 0, 2, 3), since we are looking at the magnitude of the weights, not the sign, so those are the most important features for classifying happy vs angry.

Part d)

```
X_2 = X[:, [0, 2, 3]]
weights_2 = np.linalg.inv(np.transpose(X_2)@X_2)@np.transpose(X_2)@y
print("Weights: \n", weights_2)
# predictor
y_hat_2 = np.sign(X_2@weights_2)
```

```
Weights:
 [[ 0.70546316]
 [ 0.8737872 ]
 [-0.78805643]]
```

The features I chose were the same three from part c) above because they are the ones that affect the decision the model makes the most. In this new classifier, I extracted those three columns (indices 0, 2, 3) out from the X matrix and then used the exact same equation as the first classifier, $w = (X^T X)^{-1} X^T y$, to calculate the weights for the predictor. The predictor is also the same, $y = sign(x^T w)$, however, the feature vector x now also only contains the three most important measured distances on the new face.

Part e)

```
# errors for all 9 features
errors = [0 if i[0]==i[1] else 1 for i in np.hstack((y, y_hat))]
print("Number of misclassifications with 9 features: ", sum(errors))
print("Training error for 9 features: {per}%\n".format(
    per=(sum(errors)/128)*100))
# errors for 3 most important features
errors_2 = [0 if i[0]==i[1] else 1 for i in np.hstack((y, y_hat_2))]
print("Number of misclassifications with 3 features: ", sum(errors_2))
print("Training error for 3 features: {per}%".format(
    per=(sum(errors_2)/128)*100))
```

```
Number of misclassifications with 9 features:  3
Training error for 9 features: 2.34375%

Number of misclassifications with 3 features:  8
Training error for 3 features: 6.25%
```

.

.

.

Part f)

```
# compute cross validation error with holdout set
def compute_error(X, y, holdout, y_holdout):
  weights = np.linalg.inv(np.transpose(X)@X)@np.transpose(X)@y
  y_hat = np.sign(holdout@weights)
  errors = [0 if i[0]==i[1] else 1 for i in np.hstack((y_holdout, y_hat))]
  return sum(errors)/16

# split into eight sets
eight_sets = np.split(X, 8)
eight_y = np.split(y, 8)
total_error = 0

# cross validation
for i in range(8):
  # remove one set from the eight
  training = np.delete(eight_sets, i, axis=0).reshape(-1, 9)
  training_y = np.delete(eight_y, i, axis=0).reshape(112, -1)
  # error for the holdout set
  total_error += compute_error(training, training_y, eight_sets[i], eight_y[i])

# for all features
print("Cross validation error for 9 features: {per}%".format(
    per=(total_error/8)*100))


eight_sets_2 = np.split(X_2, 8)
total_error = 0

# cross validation
for i in range(8):
  # remove one set from the eight
  training = np.delete(eight_sets_2, i, axis=0).reshape(-1, 3)
  training_y = np.delete(eight_y, i, axis=0).reshape(112, -1)
  # error for the holdout set
  total_error += compute_error(
      training, training_y, eight_sets_2[i], eight_y[i])

# for 3 features
print("Cross validation error for 3 features: {per}%".format(
    per=(total_error/8)*100))
```

```
Cross validation error for 9 features: 4.6875%
Cross validation error for 3 features: 7.8125%
```