least-square

February 27, 2024

1 1. Happy/ Angry Classifier

[1]:
```
import numpy as np
from scipy.io import loadmat
import matplotlib.pyplot as plt
import sympy
from sympy.abc import x

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

X = in_data['X']
y = in_data['y']
# print(np.shape(X))
# print(np.shape(y))
n_y = np.size(y)
print(n_y)
```

```
['__header__', '__version__', '__globals__', 'y', 'X']
128
```

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

[2]:
```
# least squares as a loss function
# w = (X^T X)^(-1)X^T y
w = np.linalg.inv(X.transpose()@X)@X.transpose()@y
print(w)
```

```
[[ 0.94366942]
 [ 0.21373778]
 [ 0.26641775]
 [-0.39221373]
 [-0.00538552]
 [-0.01764687]
 [-0.16632809]
```

```
[-0.0822838 ]
[-0.16644364]]
```

1.0.2 b) Explain how to use the weights you found to classify a new face image as happy or angry?

Utilize the weights to perform a dot product with the feature matrix of the face images (X). If the result is greater than 0, classify the image as +1 (happy); otherwise, classify it as -1 (angry).

[3]:
```
y_hat = np.sign(X@w)
# print(y_hat)
```

1.0.3 c) Which features seem to be most important? Justify your answer. Note that the nine columns of the training data feature matrix X have been normalized to have the same two-norm.

The most crucial feature for determining whether an image is happy or not appears to be the 1st feature, as it possesses the highest weight value (0.94). Conversely, when discerning an angry face, the 4th feature becomes most significant, given its weight of (-0.39).

1.0.4 d) Design a classifier based on three of the nine features. Which three should you choose? Describe the procedure for designing your classifier.

I created two classifier options:

1. Selected features with the highest positive weights, along with the column with the greatest negative weight (indices: 0, 2, 3).
2. Chose the column with the highest positive weight along with the first two columns featuring the greatest negative weights (indices: 0, 3, 8).

I trained new classifiers for both options and compared their performance against each other and the original classifier. –> Features indexed 0, 3, and 8 were identified as the optimal choice based on the evaluation results.

[4]:
```
# 9 feat performance
error_vec = [0 if i[0]==i[1] else 1 for i in np.hstack((y_hat, y))]
print('Error Rate:' + str(100*sum(error_vec)/ n_y) + '%')
```

```
Error Rate:2.34375%
```

[5]:
```
print(w)
```

```
[[ 0.94366942]
 [ 0.21373778]
 [ 0.26641775]
 [-0.39221373]
 [-0.00538552]
 [-0.01764687]
 [-0.16632809]
```

```
[-0.0822838 ]
[-0.16644364]]
```

[6]:
```
# print(X)
# option1: [0, 2, 3]
X_1 = X[:, [0, 2, 3]]
# print(X_1)
# option2: [0, 3, 8]
X_2 = X[:, [0, 3, 8]]
# print(X_2)
```

[7]:
```
# option1: [0, 2, 3]
w = np.linalg.inv(X_1.transpose()@X_1)@X_1.transpose()@y
y_hat = np.sign(X_1@w)
error_vec = [0 if i[0]==i[1] else 1 for i in np.hstack((y_hat, y))]
print('Error Rate:' + str(100*sum(error_vec)/ n_y) + '%')
```

```
Error Rate:6.25%
```

[8]:
```
# option2: [0, 3, 8]
w = np.linalg.inv(X_2.transpose()@X_2)@X_2.transpose()@y
y_hat = np.sign(X_2@w)
error_vec = [0 if i[0]==i[1] else 1 for i in np.hstack((y_hat, y))]
print('Error Rate:' + str(100*sum(error_vec)/ n_y) + '%')
```

```
Error Rate:1.5625%
```

1.0.5 e) What percent of the training labels are incorrectly classified using all nine features? What percent of the training labels are incorrectly classified using your reduced set of three features?

Using all nine features, I observed a 2.34% misclassification rate. However, with the reduced set of three features, the misclassification rate improved to 1.56%, indicating better performance.

1.0.6 f) Now use cross validation to assess your classifier performance. Divide the available data in to eight subsets of sixteen samples (e.g., examples 1—16, 17—32, . . . , 113—128). Use seven sets to design your classifier weights, then use the remaining holdout set to evaluate the classifier performance. Compute the number of misclassifications made on this hold-out set and divide that number by 16 (the size of the set) to estimate the error rate for that hold-out set. Repeat this process eight times using the eight different possible divisions between training and holdout sets and average the error rates to obtain a final performance estimate.

The final performance estimate is 4.69%.

[9]:
```
from sklearn.model_selection import KFold
# KFold: Provides train/test indices to split data in train/test sets. Split␣
 ↪dataset into k consecutive folds
```

```
# print(X[:16,:], y[:16])
kf = KFold(n_splits=8)
# for X_train, y_train, X_test, y_test in kf.split(X, y):
total_error_rate = 0
i = 0
for train, test in kf.split(X):
    # print(train, '\n', test)
    X_train = X[train, :]
    y_train = y[train, :]
    X_test = X[test, :]
    y_test = y[test, :]
    # print(X[test, :])
    # print(y[test])
    n_y_test = np.size(y_test)
    # print(n_y_test)
    w = np.linalg.inv(X_train.transpose()@X_train)@X_train.transpose()@y_train
    y_hat = np.sign(X_test@w)
    error_vec = [0 if i[0]==i[1] else 1 for i in np.hstack((y_hat, y_test))]
    error_rate = 100*sum(error_vec)/ n_y_test
    # print('Error Rate:' + str(error_rate) + '%')
    i += 1
    total_error_rate += error_rate
    # print('Average Error Rate:' + str(total_error_rate/i) + '%')

print('Average Error Rate:' + str(total_error_rate/i) + '%')
```

```
Average Error Rate:4.6875%
```