Assignment-4-Rossi

March 30, 2022

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

[ ]:
```python
# a)
data = loadmat('face_emotion_data.mat')
X = data['X']
y = data['y']

w = np.linalg.inv(X.T@X)@X.T@y
print("Classifier weights: \n",w.round(4))

# b)
# Use the following steps:
# i. Extract the same nine features from the new face image. Place them in a␣
 ↪row vector VT
# ii. Compute the product y_hat = VT@w , where w is the weigth vector␣
 ↪calculated before.
# iii. Compute s = sign(y_hat). If s = 1, classify as happy. Otherwise if s =␣
 ↪-1, classify as angry.
```

```
Classifier weights:
 [[ 0.9437]
 [ 0.2137]
 [ 0.2664]
 [-0.3922]
 [-0.0054]
 [-0.0176]
 [-0.1663]
 [-0.0823]
 [-0.1664]]
```

[ ]:
```python
# a) Using SVD
U,s,VT = np.linalg.svd(X, full_matrices= False)
S = np.arange(81).reshape(9,9)
S_matrix = np.zeros_like(S)
np.fill_diagonal(S_matrix, s)
```

```
w = np.transpose(VT)@np.linalg.inv(S_matrix)@np.transpose(U)@y

# b) to classify a new face as happy or angry we can use y_hat => -1 as a angry␣
↪and 1 as happy
y_hat = np.sign(X@w)
```

[ ]:
```
# c)
w_hat = np.transpose(U)@y
w_hat

# c) Solution:
# The classifier computes a weighted sum of nine features. If the features are␣
↪v_1, ... , v_9 then the predicted
# label (before taking the sign) is:
#   y_hat = w_1@v_1 + ... + w_9@v_9
# Since the features are normalized - each column of X has a squared norm of␣
↪128 - the size of the weigth w_i
# determines the amount a feature v_i contributes to th epredicted label.␣
↪Therefore, if a weigth is small, then
# the relative contribution of that feature to the predicted label will be␣
↪commensurately small. Thus, the
# magnitude of the weights indicates the degree of importance of the␣
↪corresponding feature.

# Then,
# To see which feature is most important we have to take the biggest abstract␣
↪on w_hat. That is w_hat[3] = w4
```

[ ]:
```
array([[ 5.6455856 ],
       [-4.16572338],
       [-1.63547687],
       [ 6.24709159],
       [-3.28039126],
       [ 0.39327948],
       [-0.37757024],
       [-0.32271646],
       [-0.567622  ]])
```

[ ]:
```
# d)
# To design the classifier I'll pick the three most important features from␣
↪w_hat. Those will be w4, w1 and w2
# Then,
w_hat_2 = np.array((w_hat[3],w_hat[0],w_hat[1]))
y_hat_2 = np.sign(X@w_hat)
print(w_hat_2)

# d) Solution:
```

```
# Choose the three features with the largest associated weight magnitudes.␣
↪These turn our to be features 1, 3
# and 4. We may design the classifier based on these three features by first␣
↪removing the columns of X
# corresponding to features we are no longer using. Call this new matriz X_r,␣
↪the three feature classifier
# is designed by solving the least-squares problem X_r@w_r = y.

X_r = X[:,[0,2,3]]
w_r = np.linalg.inv(X_r.T@X_r)@X_r.T@y
print("new classifier weights:\n", w_r.round(4))
```

```
[[ 6.24709159]
 [ 5.6455856 ]
 [-4.16572338]]
new classifier weights:
 [[ 0.7055]
 [ 0.8738]
 [-0.7881]]
```

[ ]:
```
# e)
# 1)
print(len([j for k in (y_hat-y) for j in k if j != 0]))

# 2)
print(len([j for k in (y_hat_2-y) for j in k if j != 0]))

# e) Solution:
# 1)
y_predict = np.sign(X@w)
err1 = np.mean( y != y_predict)
print("Error rate is %.2f using all the features" %(err1*100))

# 2)
y_r_predict = np.sign(X_r@w_r)
err2 = np.mean( y != y_r_predict)
print("Error rate is %.2f using only features 1, 3, 4" %(err2*100))
```

```
3
13
Error rate is 2.34 using all the features
Error rate is 6.25 using only features 1, 3, 4
```