Tracy Sun
COEN 129
Sprint 2017

HW #2 Report

Source Code

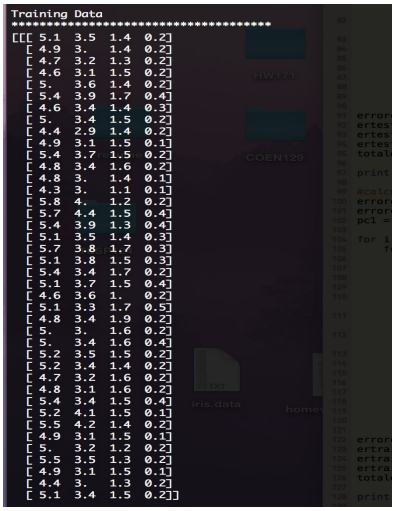
```
'''
    File: LDA.py
    Desciption: LDA analysis for 150 data points calculating mean and covariance, then output error rate
    Author: Tracy Sun
'''

import math
import numpy as np
import re

file_name = "iris.data.txt"
N = 40 # num of test data

data =[[[0.0]*4 for row in range(50)] for num in range (3)]
# an empty list of 3 matrices for each category
cnt= j= k=0
i = -1
with open(file_name,'r') as f:
    for line in f:
        if (cnt%50 == 0):
            i+=1
        j = j%50
        line = re.findall(r'\d+.\d',line)
        for number in line:
            k = k%4
            data[i][j][k] = float(number)
            k+=1
        j+=1
        cnt+=1
f.close()
data = np.array(data)


# extract 80% test data
train = data[:,:N,:]
print("\nTraining Data\n**************************************")
print (train)


# test data
test = data[:,-10:,:]
print("\nTest Data\n**************************************")
print(test)


#calculate mean vector
mean_vectors = []
```

```
for i in range(3):
    mean_vectors.append(np.mean(train[i],axis=0))
mean_vectors = np.array(mean_vectors)
print("\nMean mu\n**************************************")
print(mean_vectors)

#calculate covariance matrix
sigma_vec = []
subsum = np.zeros((1,4))
for i in range(3):
    sig = np.zeros((4,4))
    for j in range(N):
        subsum = train[i][j]- mean_vectors[i]
        subsum = np.reshape(subsum,(4,1))
        sig = sig + subsum * subsum.transpose()
    sig/=N
    sigma_vec.append(sig)
sigma_vec = np.array(sigma_vec)

#calculate average sigma vector
average_sigma = np.zeros((4,4))
for i in range(3):
    average_sigma += sigma_vec[i]
average_sigma/=3

print("\nLDA Average Sigma matrix\n**************************************")
print(average_sigma)


#calculate error rate for test data
errorcount = 0
errorc1 = errorc2 = errorc3 = 0
pc1 = pc2 = pc3 = 0.0

for i in range(3):
    for j in range(10):
        xm1 = np.reshape((test[i][j] - mean_vectors[0]),(-1,1))
        xm2 = np.reshape((test[i][j] - mean_vectors[1]),(-1,1))
        xm3 = np.reshape((test[i][j] - mean_vectors[2]),(-1,1))

        pc1 = 1/(math.pow(2.0*math.pi, 2))*1/math.sqrt(np.linalg.det(average_sigma)) *
math.exp(-(np.dot(np.dot(xm1.transpose(), np.linalg.inv(average_sigma)), xm1))/2)
        pc2 = 1/(math.pow(2.0*math.pi, 2))*1/math.sqrt(np.linalg.det(average_sigma)) *
math.exp(-(np.dot(np.dot(xm2.transpose(), np.linalg.inv(average_sigma)), xm2))/2)
        pc3 = 1/(math.pow(2.0*math.pi, 2))*1/math.sqrt(np.linalg.det(average_sigma)) *
math.exp(-(np.dot(np.dot(xm3.transpose(), np.linalg.inv(average_sigma)), xm3))/2)

        if (i==0):
            if pc1<pc2 or pc1<pc3: errorc1+=1
        elif (i == 1):
            if pc2<pc1 or pc2<pc3: errorc2+=1
        else:
            if pc3<pc1 or pc3<pc2: errorc3+=1

errorcount = errorc1+errorc2+errorc3
test_pred = 3*10
```

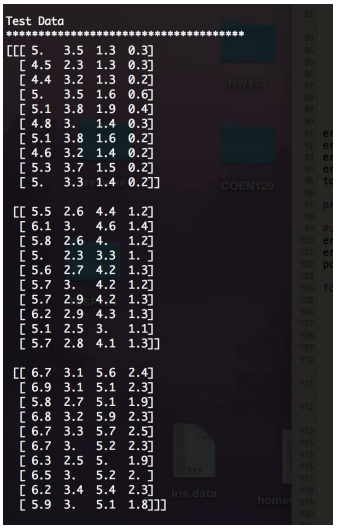
```
ertest1 = errorc1/float(test_pred)
ertest2 = errorc2/float(test_pred)
ertest3 = errorc3/float(test_pred)
totalerrorrate = errorcount/float(test_pred)
print('C1 test error rate {}'.format(ertest1))
print('C2 test error rate {}'.format(ertest2))
print('C3 test error rate {}'.format(ertest3))
print('Total error rate for test data: {}\n'.format(totalerrorrate))

# calculate error rate for training data
for i in range(3):
    for j in range(N):
        xm1 = np.reshape((train[i][j] - mean_vectors[0]),(-1,1))
        xm2 = np.reshape((train[i][j] - mean_vectors[1]),(-1,1))
        xm3 = np.reshape((train[i][j] - mean_vectors[2]),(-1,1))

        pc1 = 1/(math.pow(2.0*math.pi, 2))*1/math.sqrt(np.linalg.det(average_sigma)) *
math.exp(-(np.dot(np.dot(xm1.transpose(), np.linalg.inv(average_sigma)), xm1))/2)
        pc2 = 1/(math.pow(2.0*math.pi, 2))*1/math.sqrt(np.linalg.det(average_sigma)) *
math.exp(-(np.dot(np.dot(xm2.transpose(), np.linalg.inv(average_sigma)), xm2))/2)
        pc3 = 1/(math.pow(2.0*math.pi, 2))*1/math.sqrt(np.linalg.det(average_sigma)) *
math.exp(-(np.dot(np.dot(xm3.transpose(), np.linalg.inv(average_sigma)), xm3))/2)

        if (i==0):
            if pc1<pc2 or pc1<pc3: errorc1+=1
        elif (i == 1):
            if pc2<pc1 or pc2<pc3: errorc2+=1
        else:
            if pc3<pc1 or pc3<pc2: errorc3+=1

errorcount = errorc1+errorc2+errorc3
train_pred = 3*N

ertrain1 = errorc1 / float(train_pred)
ertrain2 = errorc2 / float(train_pred)
ertrain3 = errorc3 / float(train_pred)
totalerrorrate = errorcount/float(train_pred)

print('C1 training error rate {}'.format(ertrain1))
print('C2 training error rate {}'.format(ertrain2))
print('C3 training error rate {}'.format(ertrain3))
print('Total error rate for training data: {}\n' .format(totalerrorrate))

# linearly separatable
if (ertest1 == 0.0 and ertrain1 == 0.0):
    print("C1 is linearly seperateable")
elif (ertest2 == 0.0 and ertrain2 == 0.0):
    print("C2 is linearly seperateable")
elif (ertest3 == 0.0 and ertrain3 == 0.0):
    print("C3 is linearly seperateable")
```

LDA Analysis:

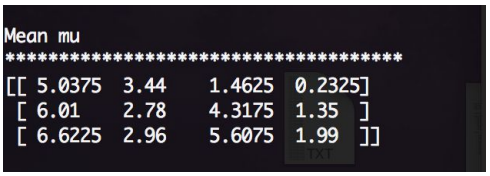
Sample of 80% training data



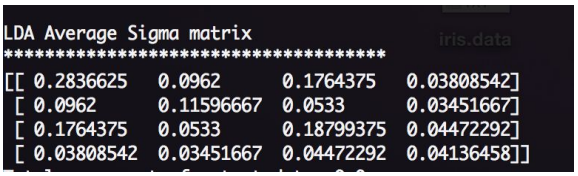
test data



Miu



Sigma



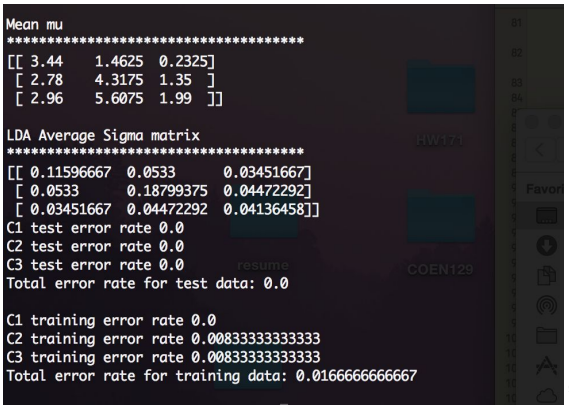
3&4

Conclusion:
One of the classes (Iris Setosa) is linearly separable from the other two because both of its test data error rate and training data error rate are 0%. However, the other two classes are not linearly separable.

#5

Remove first feature

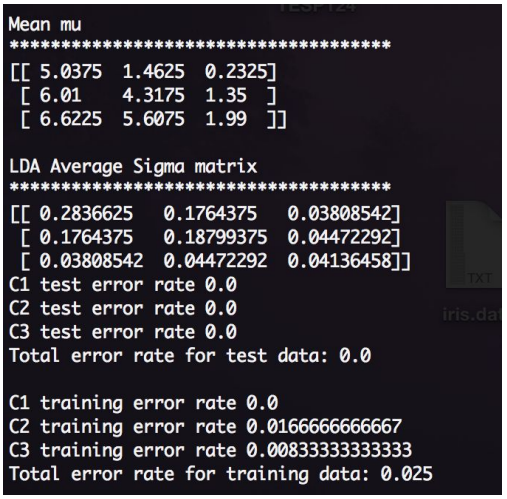
```
data = data[:,:,1:]
```



,

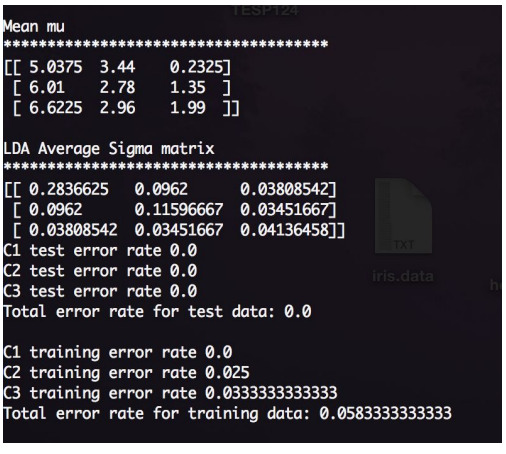
Remove 2nd

```
data = data[:,:,[0,2,3]]
```



Remove 3rd

```
data = data[:,:,[0,1,3]]
```



Remove last

```
data = data[:,:,:-1]
```

```
Mean mu
************************************
[[ 5.0375  3.44    1.4625]
 [ 6.01    2.78    4.3175]
 [ 6.6225  2.96    5.6075]]

LDA Average Sigma matrix
************************************
[[ 0.2836625   0.0962      0.1764375 ]
 [ 0.0962      0.11596667  0.0533    ]
 [ 0.1764375   0.0533      0.18799375]]
C1 test error rate 0.0
C2 test error rate 0.0
C3 test error rate 0.0333333333333
Total error rate for test data: 0.0333333333333

C1 training error rate 0.0
C2 training error rate 0.0166666666667
C3 training error rate 0.0333333333333
Total error rate for training data: 0.05
```

Conclusion:

There is some redundancy in the four input variables, so it is possible to achieve a good solution with only three of them. From the experiment, removing feature either one or two (sepal length or sepal width) gives a smaller or the same error rate for the training data that are less or equal to 2.5%. But removing either feature three or four (petal length or petal width) gives a bigger error rate that is about 5%. Therefore, removing either variable sepal length or sepal width will give a better result.

 # 6

```
diagonal = sigma_vec
for i in range(3):
    diagonal[i] = np.diag(np.diag(sigma_vec[i]))
print(diagonal)
```

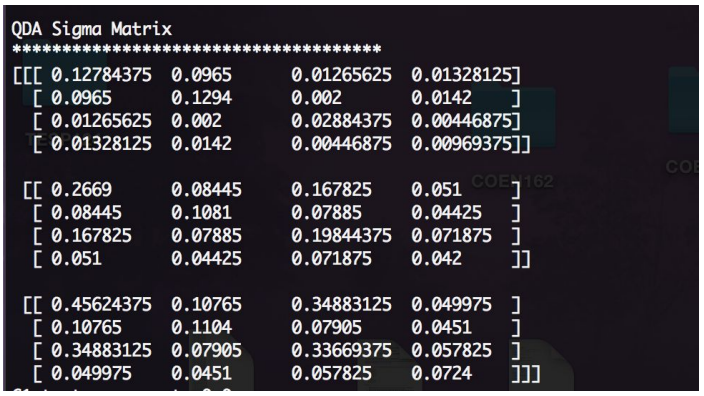
```
LDA Diagnal Sigma matrix
************************************
[[[ 0.12784375  0.          0.          0.        ]
  [ 0.          0.1294      0.          0.        ]
  [ 0.          0.          0.02884375  0.        ]
  [ 0.          0.          0.          0.00969375]]

 [[ 0.2669      0.          0.          0.        ]
  [ 0.          0.1081      0.          0.        ]
  [ 0.          0.          0.19844375  0.        ]
  [ 0.          0.          0.          0.042     ]]

 [[ 0.45624375  0.          0.          0.        ]
  [ 0.          0.1104      0.          0.        ]
  [ 0.          0.          0.33669375  0.        ]
  [ 0.          0.          0.          0.0724    ]]]

LDA Average Sigma matrix
************************************
[[ 0.2836625   0.          0.          0.        ]
 [ 0.          0.11596667  0.          0.        ]
 [ 0.          0.          0.18799375  0.        ]
 [ 0.          0.          0.          0.04136458]]
C1 test error rate 0.0
C2 test error rate 0.0
C3 test error rate 0.0
Total error rate for test data: 0.0

C1 training error rate 0.0
C2 training error rate 0.0166666666667
C3 training error rate 0.0333333333333
Total error rate for training data: 0.05
```

Diagonal matrix gives a bigger error rate because we made an assumption to use only diagonal entries of the covariance matrix. So the calculation for the average sigma matrix will be less accurate than using the calculated one.

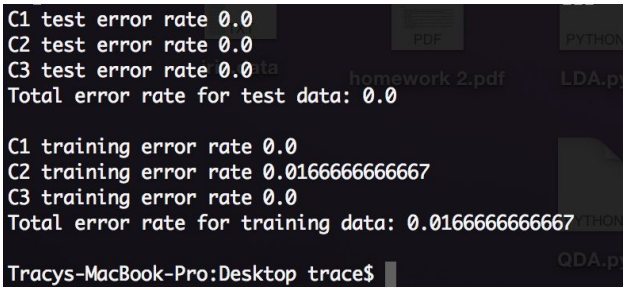
QDA Analysis

Source code reference in LDA
Output the same test & training data and the same mean vector as in LDA analysis

QDA uses sigma matrix instead of the average sigma

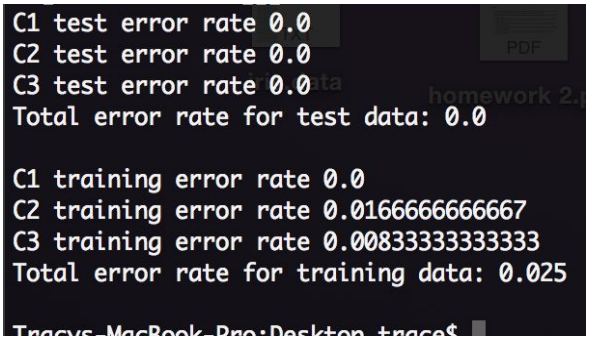


Error rate:

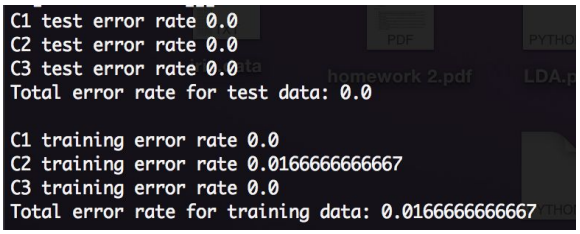


 # 5

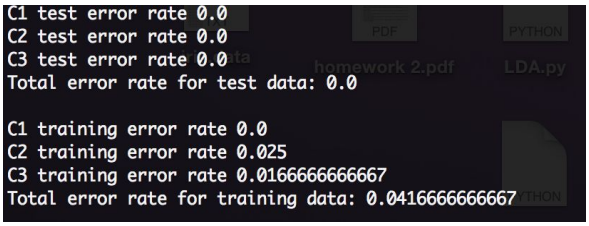
Remove 1st variable:



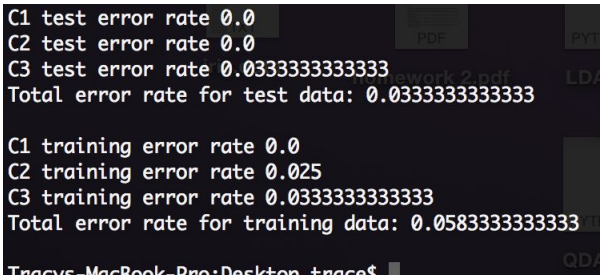
Remove 2nd variable:



Remove 3rd variable:



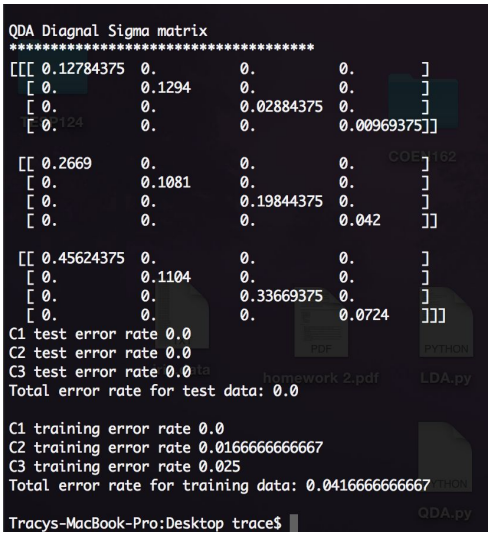
Remove last variable:



Conclusion:

From the experiment, removing 2nd variable (sepal width) gives the same error rate for the training data. But removing either variable one, three or four (sepal length, petal length or petal width) gives a bigger error rate. Therefore, variable sepal width is non relevant.

 # 6



Diagonal matrix gives a bigger error rate because again we made an assumption to use only diagonal entries of the covariance matrix.