CS289–Spring 2017 — Homework 3 Solutions

Shuhui Huang, SID 3032129712

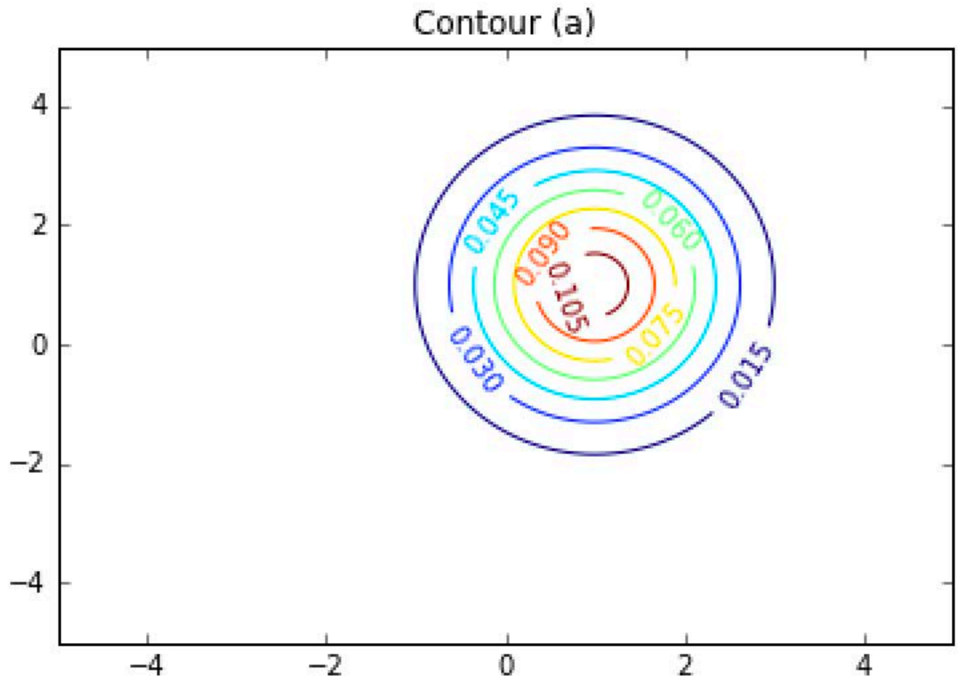
1. Independence vs. Correlation

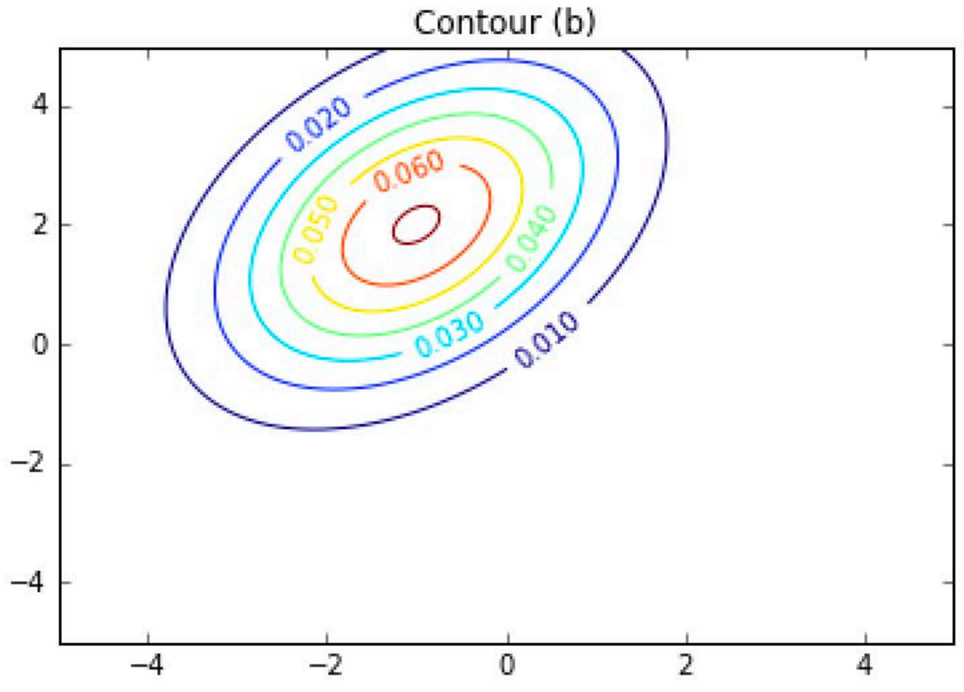
(a) There are 4 possible points (X, Y) can be, all with equal probability ($\frac{1}{1}$): (0,1); (0,-1); (1, 0); (-1,0). To prove they are uncorrelated, we just need to prove their covariance is 0:
E(X)=E(Y)=$\frac{1}{2} * 0 + \frac{1}{2} * (\frac{1}{2} + \frac{-1}{2}) = 0$
E(XY)=0
$\therefore$ cov(X,Y)=$E(XY) - E(X)E(Y) = 0$, which means they are uncorrelated.
But they are not independent. Because P(X=0)=$\frac{1}{2}$ while $P(X = 0|Y = 1) = 0$

(b) X,Y are independent.
$\because P(X|Y) = P(X) \therefore P(X)P(Y) = P(X, Y)$.
And we can make the same conclusion for all pairs of random variables. Therefore, X,Y,Z are pairwise independent.
However, $P(X|Y, Z) \neq P(X)$:
$Y \bigoplus Z = (C \bigoplus D) \bigoplus (B \bigoplus D) = C \bigoplus B = X$
Therefore, X,Y,Z are not mutually independent.

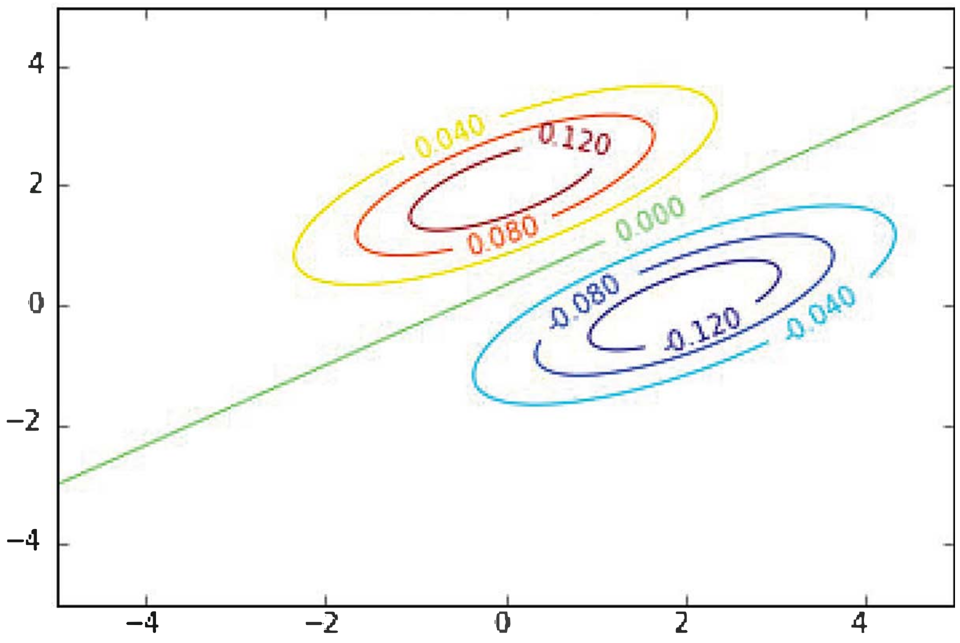
2.Isocontours of Normal Distributions

The pictures are:

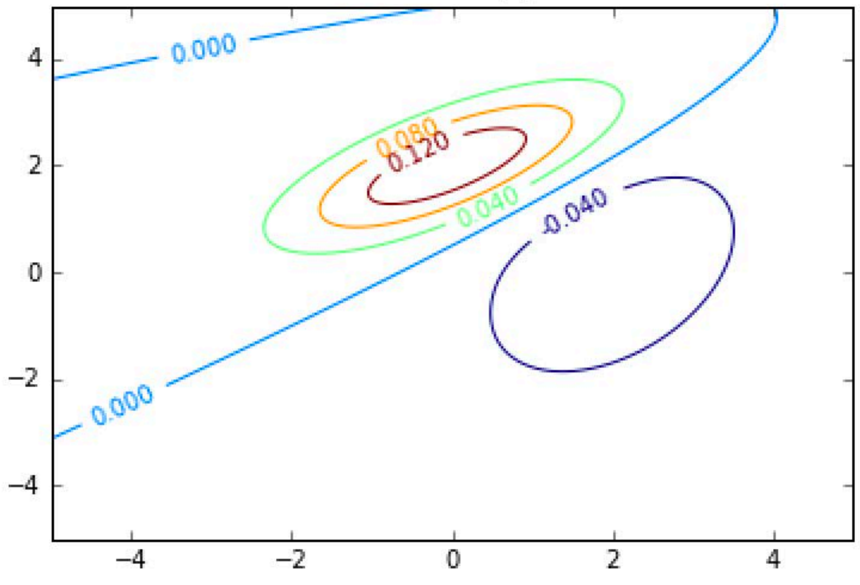


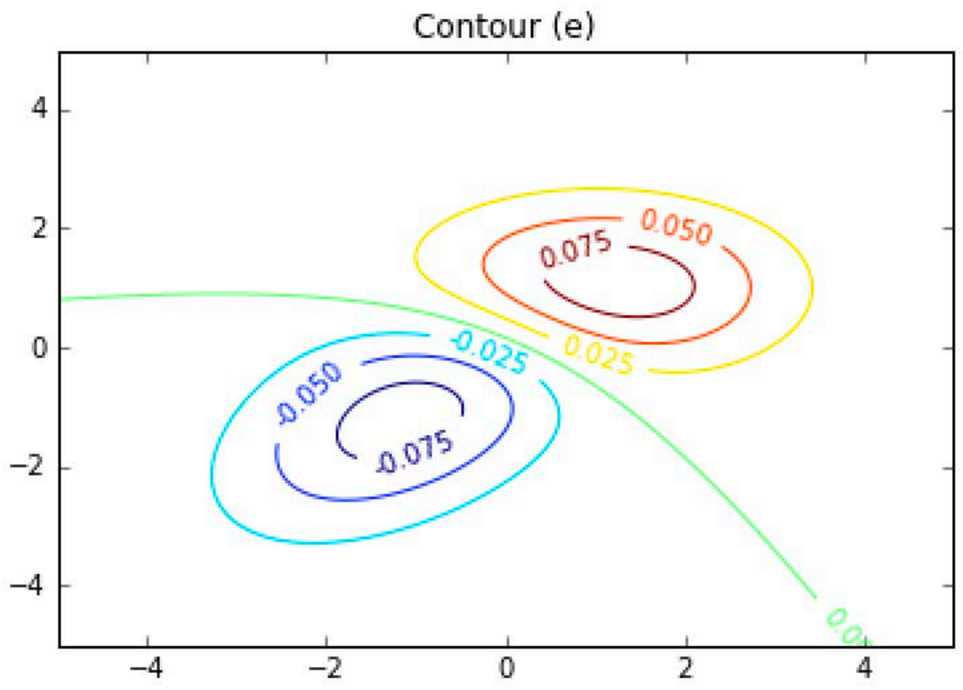


Contour (c)



Contour (d)




Contour (e)
4
2
0
−2
−4
−4
−2
0
2
4
0.075
0.050
0.025
-0.025
-0.050
-0.075
0.0

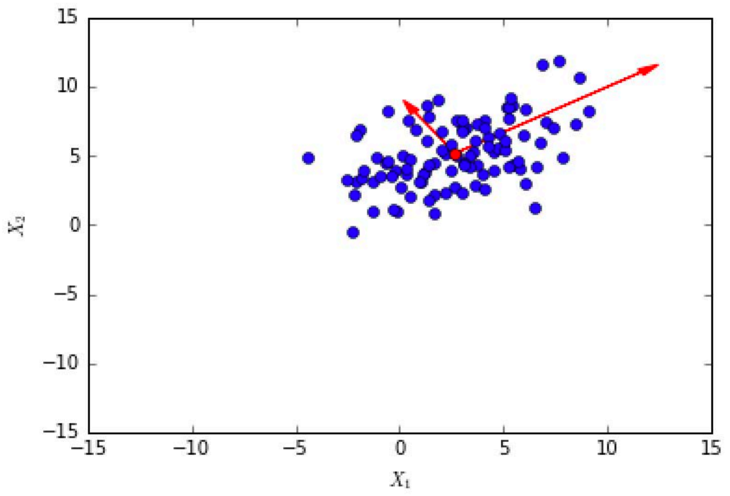
3.Eigenvectors of the Gaussian Covariance Matrix

(a) I set the random seed to be 100 in my code, so the results below can be replicated. The mean of the sampled data is: 2.68750243847, 5.2032588915.

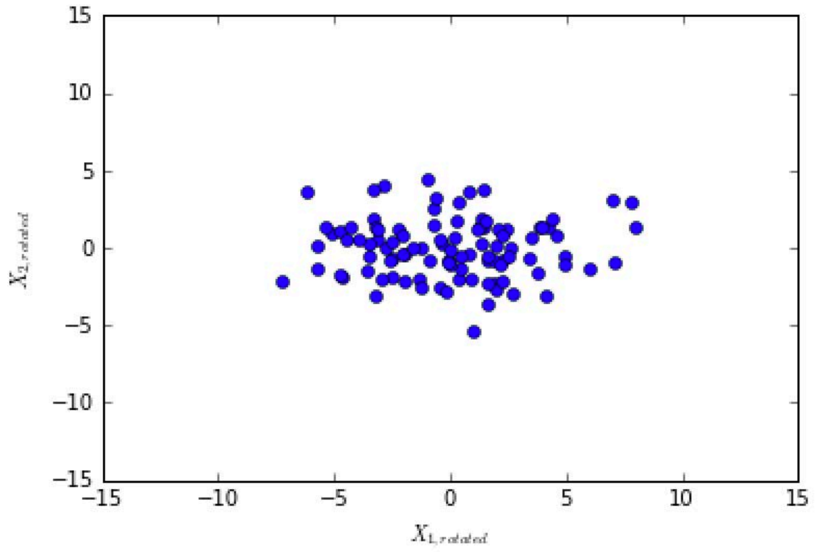
(b) The covariance matrix is: $\begin{bmatrix} 8.54927186 & 3.26257895 \\ 3.26257895 & 5.69991138 \end{bmatrix}$

(c) The eigenvalues are: 10.68466679 3.56451645.
The eigenvectors are: $\begin{bmatrix} 0.8367146 & 0.54763919 \end{bmatrix}$ and $\begin{bmatrix} -0.54763919 & 0.8367146 \end{bmatrix}$

(d) The plot is:



(e) The rotated points are plotted as follows:



4.

(a) First of all, notice that X_i are independent, so the log of density of this distribution is :
$L(X_1, \cdots, X_n) = lnf(X_1, X_2, \cdots, X_n) = constant - \frac{n}{2} ln|\Sigma| - \frac{1}{2}\sum_{i=1}^{n}(X_i - \mu)'\Sigma^{-1}(X_i - \mu)$
Because $\frac{\partial(X'AX)}{\partial X} = 2AX$, $\frac{\partial(X'AX)}{\partial A} = XX'$, $\frac{\partial ln|A|}{\partial A} = A^{-1}$
Now let $\frac{\partial lnL(\mu,\Sigma)}{\partial \mu} = \sum_{i=1}^{n} \Sigma^{-1}(X_i - \mu) = 0$
So $\hat{\mu} = \frac{1}{n}\sum X_i = \bar{X}$
And let $\frac{\partial lnL(\mu,\Sigma)}{\partial \Sigma} = -\frac{n}{2}\Sigma^{-1} + \frac{1}{2}\sum_{i=1}^{n}(X_i - \mu)(X_i - \mu)'(\Sigma^{-1})^2 = 0$
We have $\hat{\Sigma} = \frac{1}{n}\sum_{i=1}^{n}(X_i - \bar{X})(X_i - \bar{X})' = \frac{1}{n}S$
So $\hat{\sigma_i^2} = \frac{1}{n}\sum_{i=1}^{n}(X_{i1} - \bar{X_1}^2)$

(b) Suppose the normal distribution has a known covariance matrix and an unknown mean
$\frac{1}{n}\sum_{i=1}^{n}(X_i - \hat{A\mu})(X_i - A\mu)' = \Sigma$, Then we have:
$\hat{\mu} = A^{-1}(X - |\Sigma|^{\frac{1}{2}})$

5.Covariance Matrices and Decompositions

(a) Σ should be positive denite ($\forall x \in R^n$, $X'\Sigma X > 0$) in order for it to be invertible, because invertible matrices cannot have any eigenvalues be 0.

However, there are two possibilities that eigenvalues can be 0. One is that when one or more of the random variable R_i are dependent on the others. In this case, eigenvectors should be independent in order for the covariance matrix to have no eigenvalues of 0. Another case would be when one or more of the random variable R_i are deterministic.

For instance, let's consider a n-length random variable R that the first n-1 elements are standard normal RV's, but the last one element is deterministic(for example, always be 1). Then, we can easily convert random variable X into a new random variable X' that has n-1 different eigenvalues without losing any of information.

(b) Use the Spectral Decomposition Theorem to convert into the following:
Let U be a unitary matrix and D be a diagonal matrix with eigenvalues as Σ,
Σ=UDU', so we have $\Sigma^{-1} = (UDU')^{-1} = UD^{-1}U'$
Because all eigenvalues $d_{i,i} > 0$ since Σ is positive definite, so the eigenvalues of D^{-1} is $\frac{1}{d_{i,i}}$ exists, so does $\frac{1}{\sqrt{d_{i,i}}}$

Therefore, we can decompose D^{-1} into its square-root by defining Q as a diagonal matrix with eigenvalues $\frac{1}{\sqrt{d_{i,i}}}$. So in this way, we can represent as:
$\Sigma^{-1} = UD^{-1}U' = UQQ'U'$
Let A=Q'U', we have $\Sigma^{-1} = A'A$

(c) In (b), we already have $\Sigma^{-1} = A'A$ To maximize f(X), we want the superscript above the exponent to be minimal since there is a negative sign. So we want the the minimum of x'A'Ax, which is $||Ax||_2^2$
$\therefore\ ||Ax||_2^2 = x'A'Ax = X'UD^{-1}U'x = q'D^{-1}q$
Notice $|x| = 1$, so we have $|q| = |Ux| = 1$, let q be any Euclidean Basis Vector e_i such that the ith element is 1 and all other elements are 0.

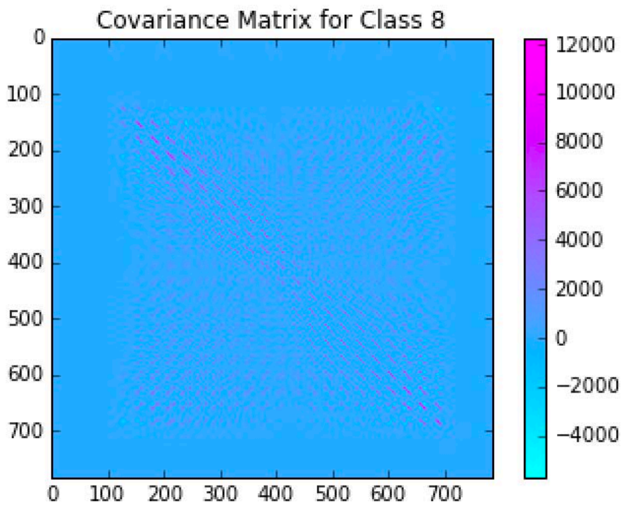
Therefore, to maximizes the PDF, the minimum value of $||Ax||_2^2$ is $\frac{1}{\lambda_i}$, where λ_i is the maximum eigenvalue of Σ, and we can get the x to be the vector corresponding to the eigenvector corresponding to the maximal eigenvector λ_i.

Similarly, to minimizes the PDF, the maximum value of $||Ax||_2^2$ is $\frac{1}{\lambda_j}$, where λ_j is the minimum eigenvalue of Σ, and we can get the x to be the vector corresponding to the eigenvector corresponding to the maximal eigenvector λ_j.

6.Gaussian Classifiers for Digits and Spam

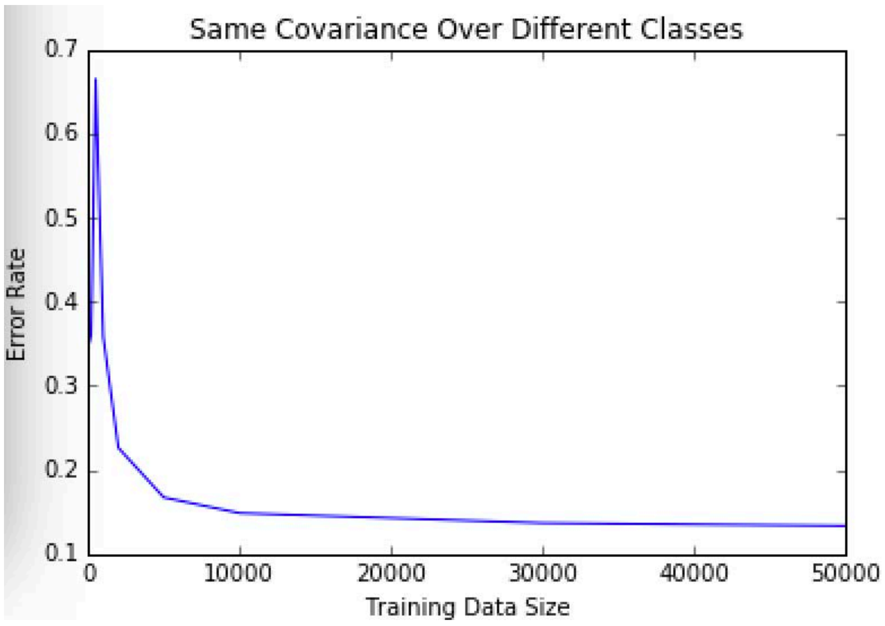
(a) the MLE of estimation is:
$\hat{\mu} = \bar{X}$, $\hat{\Sigma} = \frac{1}{n}\sum_{n=1}^{n}(X_i - \bar{X})(X_i - \bar{X})'$

(b) Below is the plot for the covariance matrix of class 8.

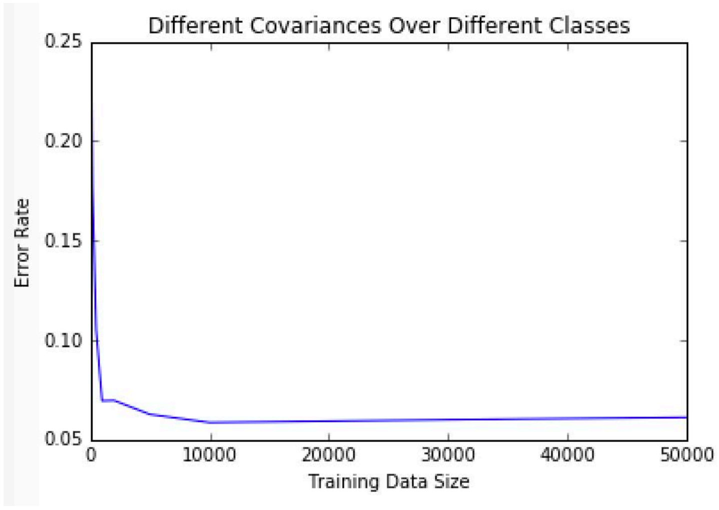


As can be seen, the entries in the diagonal of the matrix are much higher than other entries. This means that the features in the image are highly correlated with the features that are close to them.

(c) (i).Because the multivariate Gaussians share the same covariance matrix Σ, the decision boundary is linear. $log(P(Y = i|X)) \propto log(P(X|Y)P(Y)) = log(P(y = i)) - \frac{1}{2}(X - \mu_i)\Sigma^{-1}(X - \mu_i)' - \frac{1}{2}log|\Sigma|$
And we use the proportion as prior probability. The plot of the error rate over the training data size is as follows:
The error rate is $[0.3521, 0.3611, 0.6648, 0.3584, 0.2262, 0.1672, 0.1486, 0.1368, 0.1335]$



(ii). When we assume the covariance matrix is different over different classes, the posterior probability of different classes are:
$log(P(Y = i|X)) \propto log(P(X|Y)P(Y)) = log(P(y = i)) - \frac{1}{2}(X - \mu_i)\Sigma_i^{-1}(X - \mu_i)' - \frac{1}{2}log|\Sigma_i|$
The plot of the error rate over the training data size is as follows:
The error rate is $[0.2493, 0.1768, 0.1054, 0.0695, 0.0697, 0.0627, 0.0586, 0.06, 0.0612]$



(iii).QDA performs better. Because it does not make the assumption that all classes share the same covariance matrix, and the correlations between different features in different classes always be different, so the information will lose less in QDA compared with LDA.

(iv).First I do a cross validation to find the optimal (which is 0.0009), then I use this op-timal to get the predicted labels for the test data in Kaggle. My final score is: 0.95680 , name:Shuihui Huang.

(d) Similar to (c), I trained my model on the spam dataset and my final score on Kaggle is: 0.80620

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue Feb 14 10:02:30 2017

@author: huangshuhui
"""



###problem 2: Isocontours of Normal Distributions
import numpy as np
import matplotlib.pyplot as plt
import math
import scipy.io
import random
from scipy.stats import multivariate_normal
from sklearn import preprocessing

### (a) ###
x, y = np.mgrid[-5:5:.01, -5:5:.01]
pos = np.dstack((x, y))
rv = multivariate_normal([1,1], [[1,0], [0,2]])
CS=plt.contour(x, y, rv.pdf(pos))
plt.clabel(CS,inline=1, fontsize=10)
plt.title("Contour (a)")
plt.show()

### (b) ###
x, y = np.mgrid[-5:5:.01, -5:5:.01]
pos = np.dstack((x, y))
rv = multivariate_normal([-1,2], [[2,1], [1,3]])
CS=plt.contour(x, y, rv.pdf(pos))
plt.clabel(CS,inline=1, fontsize=10)
plt.title("Contour (b)")
plt.show()

### (c) ###
x, y = np.mgrid[-5:5:.01, -5:5:.01]
pos = np.dstack((x, y))
rv = multivariate_normal([0,2], [[2,1], [1,1]])
rv2 = multivariate_normal([2,0], [[2,1], [1,1]])
CS=plt.contour(x, y, rv.pdf(pos)-rv2.pdf(pos))
plt.clabel(CS,inline=1, fontsize=10)
plt.title("Contour (c)")
plt.show()

### (d) ###
x, y = np.mgrid[-5:5:.01, -5:5:.01]
pos = np.dstack((x, y))
rv = multivariate_normal([0,2], [[2,1], [1,1]])
```

```
rv2 = multivariate_normal([2,0], [[2,1], [1,3]])
CS=plt.contour(x, y, rv.pdf(pos)-rv2.pdf(pos))
plt.clabel(CS,inline=1, fontsize=10)
plt.title("Contour (d)")
plt.show()

### (e) ###
x, y = np.mgrid[-5:5:.01, -5:5:.01]
pos = np.dstack((x, y))
rv = multivariate_normal([1,1], [[2,0], [0,1]])
rv2 = multivariate_normal([-1,-1], [[2,1], [1,2]])
CS=plt.contour(x, y, rv.pdf(pos)-rv2.pdf(pos))
plt.clabel(CS,inline=1, fontsize=10)
plt.title("Contour (e)")
plt.show()

###problem3: Eigenvectors of the Gaussian Covariance Matrix

np.random.seed(100)
X1 = np.random.normal(3,3,100)
X0 = np.random.normal(4,2,100)
X2 = 0.5*X1+X0

### (a) ###
print("(a) The mean is:")
mean_1 = sum(X1)/len(X1)
mean_2 = sum(X2)/len(X2)
print(mean_1,mean_2)

### (b) ###
print("(b) The covariance matrix is: ")
cov_mat = np.cov(X1,X2)
print(cov_mat)

### (c) ###
print("(c) The eigenvalues and eigenvectors of this covariance matrix: ")
w, v = np.linalg.eig(cov_mat)
print(w)
print(v.transpose())

### (d) ###
plt.plot(X1,X2,'bo')
plt.plot(mean_1,mean_2,'ro')
dist0=math.sqrt(v[0][0]**2+v[1][0]**2)
dist1=math.sqrt(v[0][1]**2+v[1][1]**2)
plt.arrow(mean_1,mean_2,w[0]*v[0][0]/(dist0),w[0]*v[1][0]/dist0,color="r",head_width=
0.5,head_length=1)
plt.arrow(mean_1,mean_2,w[1]*v[0][1]/(dist1),w[1]*v[1][1]/dist1,color="r",head_width=
0.5,head_length=1)
plt.axis((-15,15,-15,15))
plt.xlabel("$X_1$")
plt.ylabel("$X_2$")
plt.show()


### (e) ###
value_max_index = list(w).index(max(w))
v1 = v[:,value_max_index]
v2 = v[:,1-value_max_index]
U = np.column_stack((v1,v2))
new_X1=[]
new_X2=[]
for i in range(len(X1)):
    x = np.array((X1[i],X2[i]))
    u = np.array((mean_1,mean_2))
    x_rotated = U.transpose().dot(x-u)
    new_X1.append(x_rotated[0])
    new_X2.append(x_rotated[1])
plt.plot(new_X1,new_X2,'bo')
plt.axis((-15,15,-15,15))
plt.xlabel("$X_{1,rotated}$")
plt.ylabel("$X_{2,rotated}$")
plt.show()

###problem 6: Gaussian Classifiers for Digits and Spam

##for mnist dataset
train_mdata = scipy.io.loadmat('/Users/huangshuhui/Desktop/study/CS289/hw2017/hw3/mni
st_dist/train.mat')
train_my=train_mdata['trainX'][...,784]
train_mx=train_mdata['trainX'][...,0:784]
test_mdata=scipy.io.loadmat('/Users/huangshuhui/Desktop/study/CS289/hw2017/hw3/mnist_
dist/test.mat')
test_mx=test_mdata['testtestX']
train_mx=preprocessing.normalize(train_mx,norm='l2')
test_mx=preprocessing.normalize(test_mx,norm='l2')

##for spam dataset
train_sdata = scipy.io.loadmat('/Users/huangshuhui/Desktop/study/CS289/hw2017/hw3/spa
m_dist/spam_data.mat')
train_sy=train_sdata['training_labels']
train_sy=train_sy[0]
train_sx=train_sdata['training_data']
test_sx=train_sdata['test_data']

### (a) ###
mean_mest=[]
cov_mest=[]
pct_mest=[]
for i in range(10):
    class_i = train_mx[np.where(train_my==i)[0]]
    mean_mest.append(np.mean(class_i,axis=0))
    cov_mest.append(np.cov(class_i.T,bias=1))
    pct_mest.append(len(class_i)/len(train_mx))

mean_sest=[]
cov_sest=[]
pct_sest=[]
for i in range(2):
    class_i = train_sx[[np.where(train_sy==i)[1]]]
    mean_sest.append(np.mean(class_i,axis=0))
    cov_sest.append(np.cov(class_i.T,bias=1))
    pct_sest.append(len(class_i)/len(train_sx))

### (b) ###
plt.imshow(cov_mest[8],cmap=plt.get_cmap('cool'))
plt.colorbar()
plt.title("Covariance Matrix for Class 8")
plt.show()

## (d_i) ###
mrange=range(0,60000)
msample=random.sample(mrange,10000)
valid_mx=np.array(train_mx[msample])
train_amx=np.delete(train_mx,msample,0)
valid_my=np.array(train_my[msample])
train_amy=np.delete(train_my,msample,0)

def Gaussian_classifier_overall(lamda,train_mx,train_my,test_mx):
    mean_mest=[]
    cov_mest=[]
    pct_mest=[]
    num_labels = np.unique(train_my)
    for i in range(len(num_labels)):
        class_i = train_mx[np.where(train_my==num_labels[i])[0]]
        mean_mest.append(np.mean(class_i,axis=0))
        cov_mest.append(np.cov(class_i.T,bias=1))
        pct_mest.append(len(class_i)/len(train_mx))
    cov_overall = cov_mest[0]
    for i in range(1, len(num_labels)):
        cov_overall = cov_overall + cov_mest[i]
    cov_overall = cov_overall / len(num_labels)
    prob_mat=np.zeros((len(test_mx),len(num_labels)))
    for i in range(len(num_labels)):
        X_U = np.mat(test_mx-mean_mest[i])
        M=np.linalg.inv(cov_overall+lamda*np.eye(len(cov_overall)))
        pro_i = -0.5*(np.diagonal(X_U*M*(X_U.T)))+math.log(pct_mest[i])
        prob_mat[:,i] = pro_i
    return prob_mat.argmax(axis=1).astype(float)
```

```python
error_rate=[]
num_range = [100, 200, 500, 1000, 2000, 5000,10000, 30000, 50000]

for num in num_range:
    index = random.sample(range(len(train_amx)),num)
    tra_mx = train_amx[index]
    tra_my = train_amy[index]
    predict_my = Gaussian_classifier_overall(0.0001,tra_mx,tra_my,valid_mx)
    error = 1-np.sum(predict_my==valid_my)/len(valid_my)
    error_rate.append(error)


print(error_rate)
plt.plot(num_range,error_rate)
plt.xlabel("Training Data Size")
plt.ylabel("Error Rate")
plt.title("Same Covariance Over Different Classes")



### (d_ii) ###
def Gaussian_classifier_seperately(lamda,train_x,train_y,test_x):
    mean_est=[]
    cov_est=[]
    pct_est=[]
    num_labels = np.unique(train_y)
    for i in range(len(num_labels)):
        class_i = train_x[np.where(train_y==num_labels[i])[0]]
        mean_est.append(np.mean(class_i,axis=0))
        cov_est.append(np.cov(class_i.T))
        pct_est.append(len(class_i)/len(train_x))
    prob_mat=np.zeros((len(test_x),len(num_labels)))
    for i in range(len(num_labels)):
        X_U = np.mat(test_x-mean_est[i])
        M=np.linalg.inv(cov_est[i]+lamda*np.eye(len(cov_est[i])))
        sign, logdet = np.linalg.slogdet(cov_est[i]+lamda*np.eye(len(cov_est[i])))
        pro_i = -0.5*(np.diagonal(X_U*M*(X_U.T)))+math.log(pct_est[i])-0.5*logdet
        prob_mat[:,i] = pro_i
    return prob_mat.argmax(axis=1).astype(float)

error_rate_seperate=[]
num_range = [100, 200, 500, 1000, 2000, 5000, 10000, 30000, 50000]
for num in num_range:
    index = random.sample(range(len(train_amx)),num)
    tra_x = train_amx[index]
    tra_y = train_amy[index]
    predict_y = Gaussian_classifier_seperately(0.0001,tra_x,tra_y,valid_mx)
    error = 1-np.sum(predict_y==valid_my)/len(valid_my)
    print(error)
    error_rate_seperate.append(error)


print(error_rate_seperate)
plt.plot(num_range,error_rate_seperate)
plt.xlabel("Training Data Size")
plt.ylabel("Error Rate")
plt.title("Different Covariances Over Different Classes")

### (d_iv) ###
def k_fold_CV_index(length, K):
    for k in range(K):
        training_index = [x for x in range(length) if x % K != k]
        validation_index = [x for x in range(length) if x % K == k]
        yield training_index, validation_index

alpha_range = [0.0005, 0.0007, 0.0009, 0.0011, 0.0013]

cv_rate =[]
for alpha  in alpha_range:
    err=[]
    for train_index,validate_index in k_fold_CV_index(10000,5):
        x_train = train_mx[train_index]
        y_train = train_my[train_index]
        x_validate = train_mx[validate_index]
        y_validate = train_my[validate_index]
        y_predict = Gaussian_classifier_seperately(alpha,x_train,y_train,x_validate)
        rate = 1-np.sum(y_predict==y_validate)/len(y_predict)
        err.append(rate)
        print(rate)
    cv_rate.append(sum(err)/len(err))
print("(d_iv) Range for lamda:")
print(alpha_range)
print("Range of error rate")
print(cv_rate)

kaggle_y = np.array(Gaussian_classifier_seperately(0.0009,train_mx,train_my,test_mx))
.astype(int)
np.savetxt('mnist_predict.csv', kaggle_y, delimiter = ',')



### (e) ###

train_sx=preprocessing.normalize(train_sx,norm='l2')
test_sx=preprocessing.normalize(test_sx,norm='l2')

spam_predict = np.array(Gaussian_classifier_seperately(0.0009,train_sx,train_sy,test_
sx)).astype(int)
np.savetxt('spam_predict.csv', spam_predict, delimiter = ',')
```