CS189–FALL 2015 — Homework 4 Write up

ZUBO GU, SID 25500921, gu.zubo@berkeley.edu

Problem 1. solution

a . $J(\mathbf{w}, \omega_0) = (\mathbf{y} - \mathbf{Xw} - \omega_0 \mathbf{1})^T(\mathbf{y} - \mathbf{Xw} - \omega_0 \mathbf{1}) + \lambda \mathbf{w}^T\mathbf{w}$

$=\mathbf{y}^T\mathbf{y} - \mathbf{y}^T\mathbf{Xw} - \mathbf{y}^T\omega_0\mathbf{1} - \mathbf{w}^T\mathbf{X}^T\mathbf{y} + \mathbf{w}^T\mathbf{X}^T\mathbf{XW} + \mathbf{w}^T\mathbf{X}^T\omega_0\mathbf{1} + \mathbf{1}^T\omega_0 y + \mathbf{1}^T\omega_0\mathbf{XW} + \omega_0^2\mathbf{1}^T\mathbf{1} + \lambda\mathbf{w}^T\mathbf{w}$

derivative with respect to ω_0 and set equal to 0, we get

$-y^T1 + w^TX^T1 - 1^Ty + 1^TXw + 2\omega_0 1^T1 = 0$

$X^T1 = 0$ since $\overline{x} = 0, 1^T1 = n$

$-\sum_{i=1}^n y_i - \sum_{i=1}^n y_i + 2\omega_0 n = 0$

Thus, $\omega_o = \frac{1}{n}\sum_{i=1}^n y_i = \overline{y}$ which is optimal.

derivative with respect to $\mathbf{w}$ and set equal to 0, we get

$-x^Ty - x^Ty + 2X^TXw + X^T\omega_0 + X^T\omega_0 1 + 2\lambda I w = 0$

Simplify, $(X^TX + \lambda I)w = X^Ty$ Thus, $w = (X^TX + \lambda I)^{-1}X^Ty$ which is optimal.

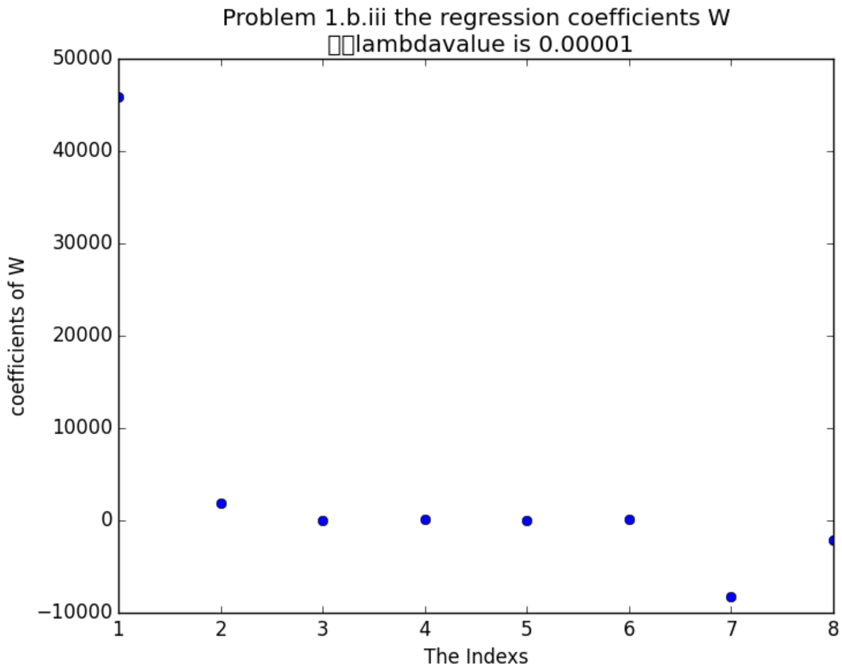
b .

i . code see attach q1.py

ii . The RSS is 6.03013493e+13

The residuals sum of squares increase compare to HW3 result.

iii . The 1st, 7th an 8th coefficients are still most significant. However, comparing to HW3 result, The 1st coefficient value increase in magnitude The 7th an 8th coefficients value decrease magnitude.



Problem 2. solution

a . $\frac{1}{36}$

b .

$1-(1-\frac{1}{36})^6 = \frac{3781}{46656}$

c .

d . No. the $\alpha = 0.05/5 = 0.001$. Thus, we can't assert "significant better" that with 5 percent wrong.

e . $p = 1-(1-p')^m$

$(1-p)^{\frac{1}{m}} = 1-p'$

$p' = 1-(1-p)^{\frac{1}{m}} = 1-(1-\frac{p}{m}) = \frac{p}{m}$ as p is very small

Thus, $p = m * p'$ the Bonferroni correction

f . P(at least one significant result) $= 1-(1-0.0001)^{50000} = 0.99326373738$. Thus it is a significant gene.

By computer the Bonferroni correction we get 0.0001 * 50000 = 5 which is greater than 1. Because we have large number test which make the result be somewhat conservative

Problem 3. solution

a . There are four possible cases $P(X,Y) \in \{(0,1),(0,-1),(1,0),(-1,0)\}$

$E(X) = E(Y) = \frac{1}{4} * -1 + 0 * \frac{1}{2} + \frac{1}{4} * 1 = 0$

$E(XY) = 0$ since there is one zero between X Y for all possible cases.

Thus, $COV(X,Y) = E(XY) - E(X)E(Y) = 0$, X and Y are uncorrelated.

$P(X=0, Y=1) = \frac{1}{4}$. But, $P(X=0) = \frac{1}{2}, P(Y=1) = \frac{1}{4}$

Thus, $P(X,Y) \neq P(X)P(Y)$, X and Y are not independent.

b .

B_1	0	0	0	1	0	1	1	1
B_2	0	0	1	0	1	1	0	1
B_3	0	1	0	0	1	0	1	1
X	0	0	1	1	1	0	1	0
Y	0	1	1	0	0	1	1	0
Z	0	1	0	1	1	1	0	0
P	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$

Thus,

X	0	1
P(X)	$\frac{1}{2}$	$\frac{1}{2}$

,

Y	0	1
P(Y)	$\frac{1}{2}$	$\frac{1}{2}$

Z	0	1
P(Z)	$\frac{1}{2}$	$\frac{1}{2}$

X \ Y	0	1
0	$\frac{1}{4}$	$\frac{1}{4}$
1	$\frac{1}{4}$	$\frac{1}{4}$

,

Y \ Z	0	1
0	$\frac{1}{4}$	$\frac{1}{4}$
1	$\frac{1}{4}$	$\frac{1}{4}$

,

Z \ X	0	1
0	$\frac{1}{4}$	$\frac{1}{4}$
1	$\frac{1}{4}$	$\frac{1}{4}$

By above we see $P(X,Y) = P(X)P(Y)$,

$P(Y,Z) = P(Y)P(Z)$

$P(Z,X) = P(Z)P(X)$

Thus, they are pairwise independent.

By above $P(X=1, Y=1, Z=1) = 0$,

$P(X=1)P(Y=1)P(Z=1) = \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{8}$

they are not equal.

Thus, they are not Mutually independent.

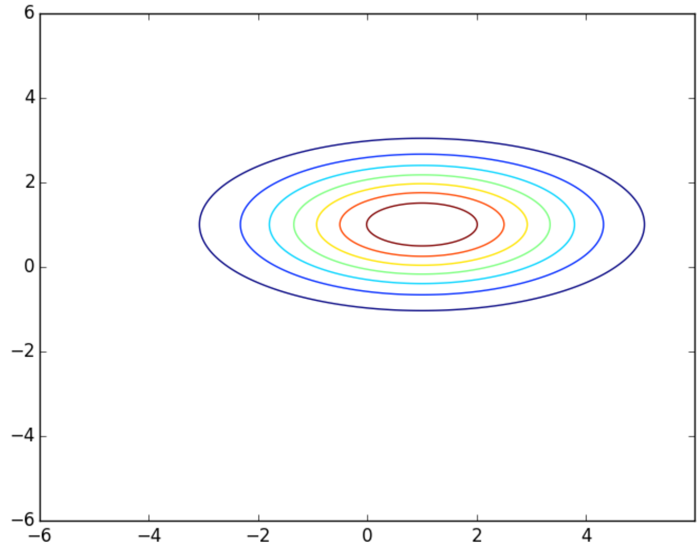
c .

The good features should be uncorrelated to each other. Thus, when we have redundant that are correlated to another feature, we can eliminate it and reduce the computation of data.

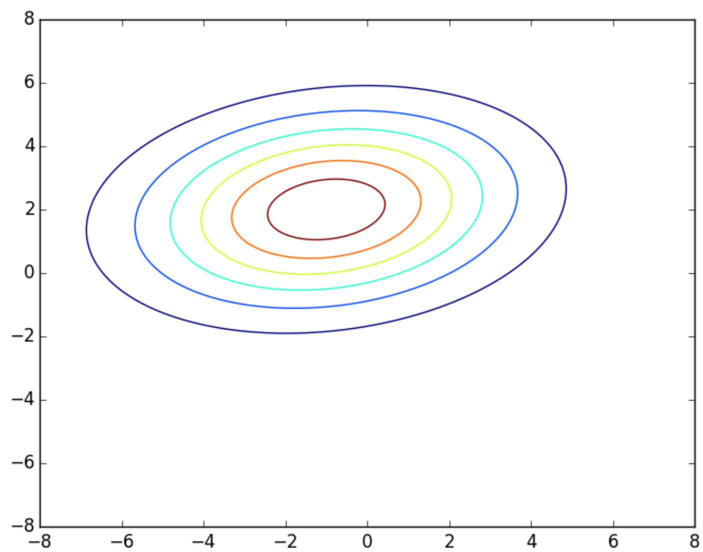
Thus, for data set that all features are independent. we can't eliminate feature

Problem 4. solution

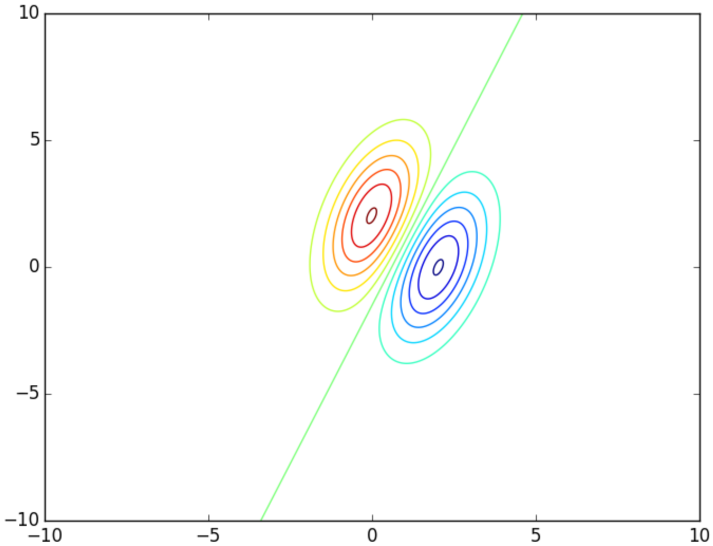
a .



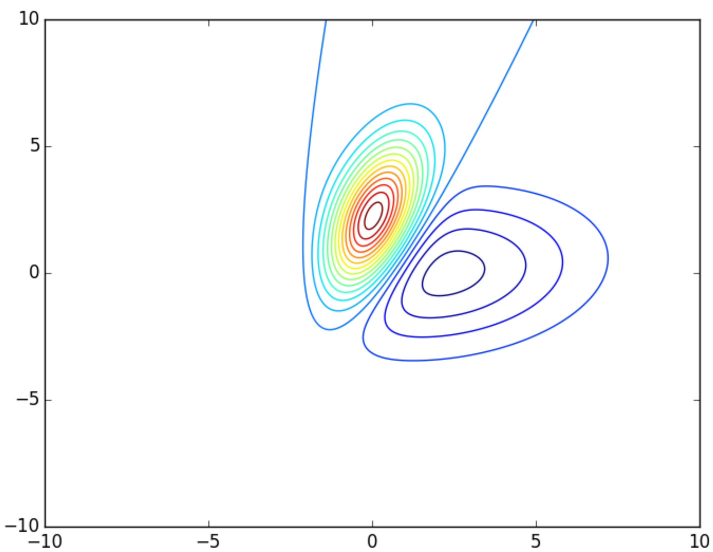
b .



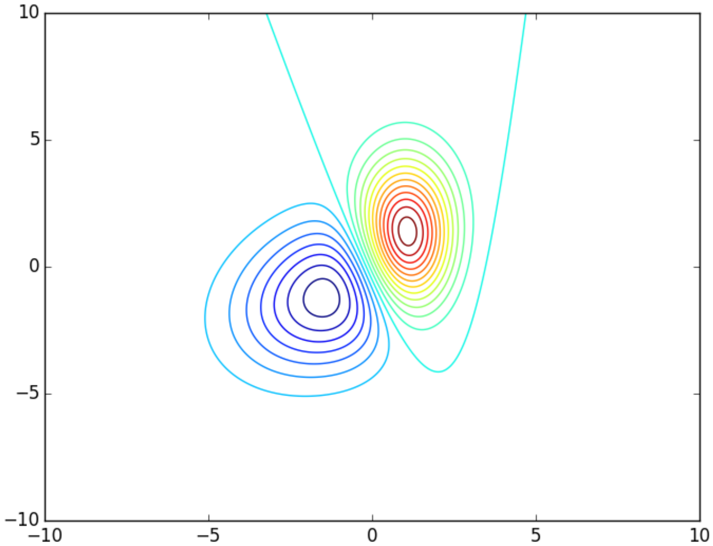
c .



d .



e .



Problem 5. solution

a .

part.a. mean of X1 is 2.67187421622

part.a. mean of X2 is 5.33879004655

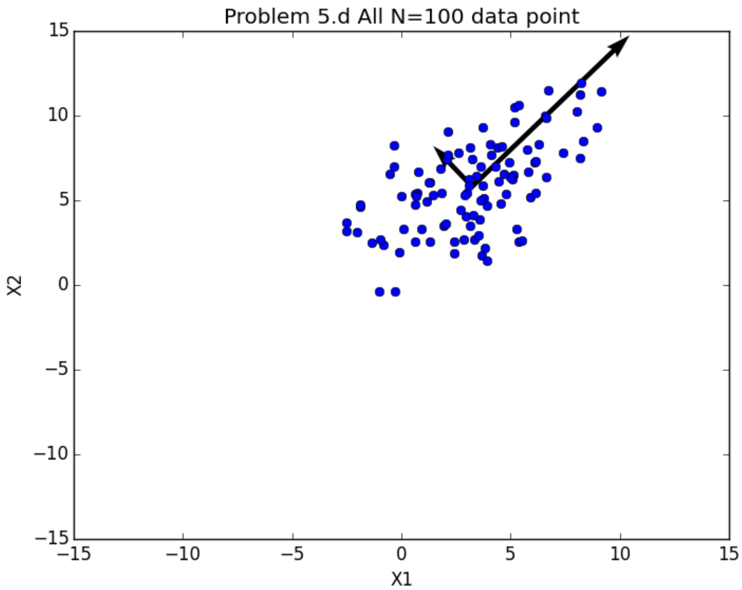
b .

convaranceMatrix is [[7.54570894 3.8874626] [3.8874626 5.90007162]]

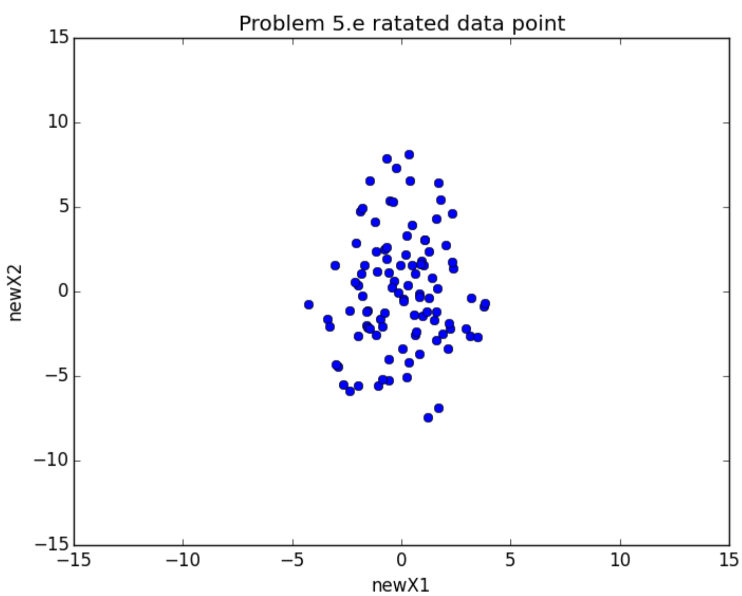
c .

eeigenvalue is 10.6964775788 eigenvector is [0.77687579 0.62965387] eigenvalue is 2.74930298331 eigenvector is [-0.62965387 0.77687579]

d .



e .



Problem 6. solution

a.). Σ_x corresponding to X is positive semidefinite, In order to be invertible, Σ_x must be positive definite. Thus, 0 can't be a eigenvalue for Σ_x. Since $\Sigma_x = US^2U^T$ where S^2 is a diag matrix with eigenvalue as entery, U is basis eigen vector. When Σ_x is invertible. If, some eigenvalue is 0. S^2 is not invertible. Also the basis eigenvectors need to be independent to each other to make U invertible.

Thus, if some X_i are deterministic. Then all $Cov(X_i, X_j) = 0$ $\forall j \in \{1, \cdots, n\}$. all that column and row will be zero.

Then, Σ has an eigenvalue of 0. Σ_x will be not invertible.

We can change X into X' by remove the deterministic item. Then, the new $\Sigma_{x'}$ convariance matrix will be invertible and. without loss information

b.). $\Sigma = US^2U^T$, where S^2 is diagnal matrix with eigen value as entries and U be the eigenvectors (normalized). U is orthogonal normal. $(U^{-1} = U^T)$

$$\Sigma^{-1} = U S^{-2} U^T$$

$$x^T \Sigma^{-1} x = x^T A^T \cdot A \cdot x \Rightarrow US^{-1} \cdot S^{-1}U^T = A^T \cdot A \Rightarrow A = S^{-1}U^T$$

Thus, exist matrix $A = S^{-1}U^T$, such that $x^T\Sigma^{-1}x = \|Ax\|_2^2$.

c.) With $x^T\Sigma^{-1}x$, it is not obvious see the meaning of it. However with $\|Ax\|$, we know $A = S^{-1}U^T$, where U is matrix with eigen vectors, S^{-1} is diagonal matrix with square root value of eigenvalue. Thus, it look like map X to new basis. And calulate the distance to O_N in the new bases. And square it for $\|Ax\|_2^2$.

d.) ① $\|X\|_2 = 1$, $\|U^T X\|_2 = 1$ as well U is normal matrix

let $U^T X = \langle U_1, \cdots, U_n \rangle$.

$\|AX\|_2^2 = \sum_{i=1}^{N} \frac{U_i^2}{\lambda_i}$, and we have $U_1^2 + U_2^2 + \cdots + U_N^2 = 1$

$0 \leq U_i^2 \leq 1$

Thus, in order to maximum $\|AX\|_2^2$, we need $|U_i^2| = 1$ with respect to maximum $\frac{1}{\lambda_i}$, for $i \in \{1, \cdots, n\}$

and in order to minimum $\|AX\|_2^2$, we need $|U_j^2| = 1$ with respect to the minimum value of $\frac{1}{\lambda_j}$ for $j \in \{1, \cdots, n\}$

Thus, the max value is $\frac{1}{\lambda_i}$ with minimum eigenvalue λ_i

the min value is $\frac{1}{\lambda_j}$ with maximum eigenvalue λ_j.

② when $X_i \perp X_j$ $\forall i, j$, $\Sigma_{ij}^{-1} = \begin{cases} \frac{1}{cov(X_i, X_j)} & \text{if } i = j \text{ for } i,j \text{ entry} \\ 0 & \text{otherwise.} \end{cases}$

similar as above. The maximum value will be $\frac{1}{cov(X_i, X_i)}$ for X_i such that give minimum $cov(X_i, X_i)$. and The minimum value will be $\frac{1}{cov(X_j, X_j)}$ with X_j which given the maximum value $cov(X_j, X_j)$

Sin X is a ~~whole~~ unit circle shape in original shape, the max and min value represent the max and min square distance to the new mean. Such that we can ~~d~~ projection in the new bases space.

③ To maximize $f(x)$, we need minimize $\|AX\|_2^2$.

we want choose X that corresponding to the eigenvector that has maximum eigenvalue λ. or to the vector has maximum covariance for itself.

Problem 7. solution

a .

means $\mu = \frac{1}{n}\sum_{i=1}^{n} y_i$

covariance matrices $\Sigma = \frac{1}{n}\sum_{i=1}^{n}(X_i - \mu)^T(X_i - \mu)$

Model see $q7 \cdot py$

b . Compute the size for each class in the train data the divide by the total size of the train data which is the prior probability for that class

The prior probability for 0 is 0.09871666666666666

The prior probability for 1 is 0.11236666666666667

The prior probability for 2 is 0.0993

The prior probability for 3 is 0.10218333333333333

The prior probability for 4 is 0.09736666666666667

The prior probability for 5 is 0.09035

The prior probability for 6 is 0.09863333333333334

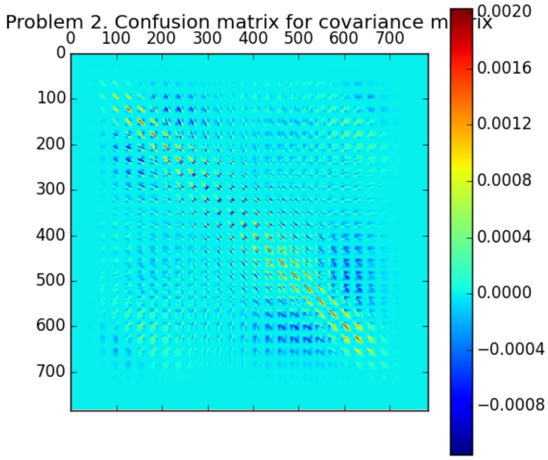
The prior probability for 7 is 0.10441666666666667

The prior probability for 8 is 0.09751666666666667

The prior probability for 9 is 0.09915

c . Below is visualize picture for digit 2.In the center area, some positive value in diagonal, left-bottom and right top, some negative value around diagonal. Zeros are in four edges.

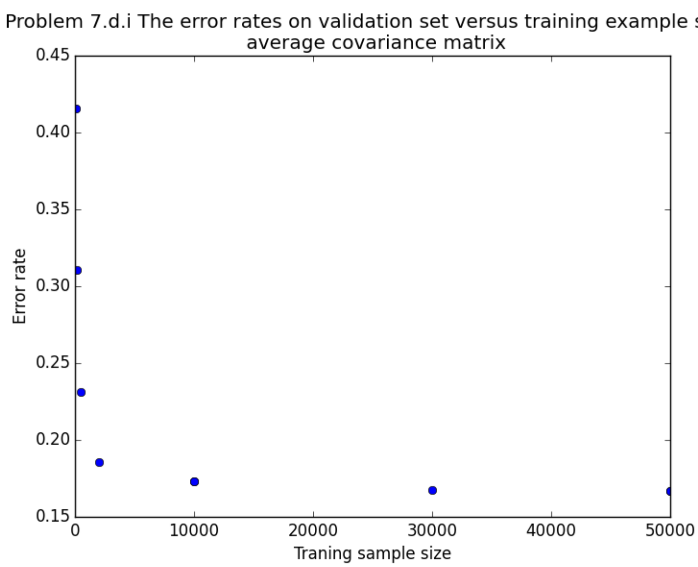
This is 784 x 784 matrix, as we concatenate 28 * 28 matrix to a 784 x 1 row vector. thus, most part of this graph is zero, since they are 0 in the row vector. Only original term not 0, we can get a covariance not zero. Thus,center area has same nonzero value since they are related and could be nonzero in the center of a 28 * 28 matrix. The edges in original 28 x 28 are most zeros, too.



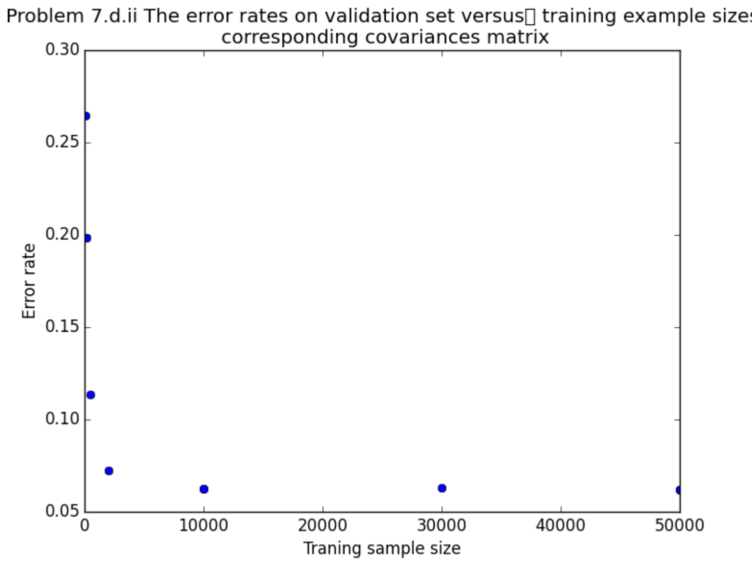
d .

i .

For here, the decision boundary is linear since we share the same covariance matrix.



ii . For here, the decision boundary is quadratic since each class has their own covariance matrix.



iii . The second method is lower in error rate which mean more accuracy in predict performance. Since second method does not using average covariance matrix and the decision boundary is quadratic, it will be more accuracy in making right predicts.

iv . Optimal prediction rate is 0.94600

e . Optimal prediction rate is 0.73506

q1 code

```
import scipy.io as sio
import numpy as np
from numpy.linalg import inv
import matplotlib.pyplot as plt
from sklearn.utils import shuffle

if __name__ == '__main__':
	#read input
	housingData = sio.loadmat('./data/housing_data.mat')

	#parta Train the model
	Xvalidate = housingData['Xvalidate']
	Yvalidate = housingData['Yvalidate']
	Xtrain = housingData['Xtrain']
	Ytrain = housingData['Ytrain'] #19440 *

	def ridgeregressionModel(Xtrain, Ytrain, Lambda):
		Xplus = Xtrain.T.dot(Xtrain) + Lambda * np.identity(len(Xtrain[0]))
		Xplus = inv(Xplus)
		W = Xplus.dot(Xtrain.T).dot(Ytrain)
		w0 = np.mean(Ytrain)
		return W, w0

	def ridgeregressionFit(Xvalidate, W, w0):
		return Xvalidate.dot(W) + w0

	def calculateRSS(expect, actual, W, lamb):
		return np.sum(np.square(expect - actual)) + lamb * W.T.dot(W)

	#partb.ii 10-fold cross validation training to find optimal lambda
	lambdavalue = 0.00001 # change lanbdavalue to test best lambda
	Xtrain, Ytrain = shuffle(Xtrain, Ytrain)
	RSS = 0
	print("10-fold cross-validation training on 10000 samples")
	for i in range(10):
		newXtrain = np.concatenate((Xtrain[:i * 1944], Xtrain[(i + 1) * 1944:]), axis = 0)
		newYtrain = np.concatenate((Ytrain[:i * 1944], Ytrain[(i + 1) * 1944:]), axis = 0)
		W, w0 = ridgeregressionModel(newXtrain, newYtrain, lambdavalue)
		expect = ridgeregressionFit(Xtrain[i * 1944:(i + 1) * 1944] , W, w0)
		RSS += calculateRSS(expect, Ytrain[i * 1944:(i + 1) * 1944], W, lambdavalue)

	#partb.ii the RSS
	W, w0 = ridgeregressionModel(Xtrain, Ytrain, lambdavalue)
	expect = ridgeregressionFit(Xvalidate, W, w0)
	RSS = calculateRSS(expect, Yvalidate, W, lambdavalue)
	print("The RSS is", RSS)

	# part2b.ii plot w
```

```
        x_label = [1, 2, 3, 4, 5, 6, 7, 8]
        plt.title('Problem 1.b.iii the regression coefficients W \n \
                lambdavalue is 0.00001')
        plt.xlabel('The Indexs')
        plt.ylabel('coefficients of W')
        plt.plot(x_label, W, 'bo')
        plt.show()
```

q4 code

```
import numpy as np
from matplotlib.patches import Ellipse
from matplotlib.mlab import bivariate_normal
import matplotlib.pyplot as plt

if __name__ == '__main__':
        def plot(eigen_value, cov_matrix):
                x = np.arange(-8.0, 8.0, 0.01)
                y = np.arange(-8.0, 8.0, 0.01)
                X, Y = np.meshgrid(x, y)

                Z = bivariate_normal(X, Y, cov_matrix[0][0], cov_matrix[1][1], eigen_value[0], eigen_value[1],
                plt.contour(X,Y,Z)
                plt.show()

        #part a
        cov_matrix = [[2,0],
                  [0,1]]
        eigen_value = [1, 1]
        plot(eigen_value, cov_matrix)

        #part b
        cov_matrix = [[3,1],
                  [1,2]]
        eigen_value = [-1, 2]
        plot(eigen_value, cov_matrix)


        def plot1(eigen_value, cov_matrix, eigen_value1, cov_matrix1):
                x = np.arange(-10.0, 10.0, 0.01)
                y = np.arange(-10.0, 10.0, 0.01)
                X, Y = np.meshgrid(x, y)

                Z = bivariate_normal(X, Y, cov_matrix[0][0], cov_matrix[1][1], eigen_value[0], eigen_value[1],
                Z1 = bivariate_normal(X, Y, cov_matrix1[0][0], cov_matrix1[1][1], eigen_value1[0], eigen_value1
                plt.contour(X,Y, Z - Z1, 20)
                plt.show()


        # part c
        cov_matrix = [[1,1],
                  [1,2]]
        eigen_value = [0, 2]

        cov_matrix1 = [[1,1],
                  [1,2]]
        eigen_value1 = [2, 0]
        plot1(eigen_value, cov_matrix, eigen_value1, cov_matrix1)
```

```
# part d
cov_matrix = [[1,1],
          [1,2]]
eigen_value = [0, 2]

cov_matrix1 = [[3,1],
          [1,2]]
eigen_value1 = [2, 0]
plot1(eigen_value, cov_matrix, eigen_value1, cov_matrix1)

#part e
cov_matrix = [[1,0],
          [0,2]]
eigen_value = [1, 1]

cov_matrix1 = [[2,1],
          [1,2]]
eigen_value1 = [-1, -1]
plot1(eigen_value, cov_matrix, eigen_value1, cov_matrix1)
```

q5 code

```
import numpy as np
import matplotlib.pyplot as plt

if __name__ == '__main__':

        #part a
        x1 = np.random.normal(3, 3, 100)
        N = np.random.normal(4, 2, 100)
        x2 = x1/2 + N

        print("part.a. mean of X1 is", np.mean(x1))
        print("part.a. mean of X2 is", np.mean(x2))

        print("part.a. variance of X1 is", np.var(x1))
        print("part.a. variance of X2 is", np.var(x2))

        #part b
        convaranceMatrix = np.cov(x1, x2)
        print("part.b. convaranceMatrix is \n", convaranceMatrix)

        #part c
        print("part.c.")
        eigenvalues, eigenvectors = np.linalg.eig(convaranceMatrix)
        eigenvectors = eigenvectors.T
        for i in range(len(eigenvalues)):
                print("eigenvalue is ", eigenvalues[i], "eigenvector is ", eigenvectors[i])

        #part d
        print("part.d.")
        plt.quiver(np.mean(x1), np.mean(x2), eigenvectors[0][0], eigenvectors[0][1], scale = eigenvalues[1])
        plt.quiver(np.mean(x1), np.mean(x2), eigenvectors[1][0], eigenvectors[1][1], scale = eigenvalues[0])
        plt.title('Problem 5.d All N=100 data point')
        plt.xlabel('X1')
        plt.ylabel('X2')
        plt.plot(x1, x2, 'bo')
        plt.xlim(-15, 15)
        plt.ylim(-15, 15)
        plt.show()

        #part e
        print("part.e.")
        UT = eigenvectors.T
        newX = UT.dot(np.array([x1 - np.mean(x1), x2 - np.mean(x2)]))
        newx1 = newX[0]
        newx2 = newX[1]
        plt.title('Problem 5.e ratated data point')
        plt.xlabel('newX1')
        plt.ylabel('newX2')
```

```
        plt.plot(newx1, newx2, 'bo')
        plt.xlim(-15, 15)
        plt.ylim(-15, 15)
        plt.show()
```

q7 code

```
import scipy.io as sio
import numpy as np
from numpy.linalg import inv
import matplotlib.pyplot as plt
from sklearn.utils import shuffle
from sklearn import preprocessing
from scipy.stats import multivariate_normal
from sklearn.utils import shuffle
import csv

if __name__ == '__main__':
        #read input
        trainData = sio.loadmat('./data/train.mat')
        trainImages = trainData['train_images']

        x_dim = len(trainImages)
        y_dim = len(trainImages[0])
        image_index = len(trainImages[0][0])
        trainImages = trainImages.transpose((2, 0, 1))
        trainImages = trainImages.reshape(image_index, x_dim * y_dim)
        actualLabels = np.transpose(trainData['train_labels'],(1, 0))[0]
        trainImages, actualLabels = shuffle(trainImages, actualLabels)
        trainImages, actualLabels = shuffle(trainImages, actualLabels)

        trainImages = preprocessing.normalize(trainImages.astype("float"))

        # part a
        data = {}
        for i in range(10):
                data[i] = []
        for i in range(len(trainImages)):
                key = actualLabels[i]
                value = trainImages[i]
                data[key].append(value)
        means = {}
        for i in range(10):
                means[i] = np.sum(data[i], axis=0) / len(data[i])

        print(means)

        covariances = {}
        for i in range(10):
                arrays = np.array(data[i])
                mu = np.array(means[i])
                matrix = (arrays - mu).T.dot((arrays - mu))
                matrix = matrix / len(arrays)
                covariances[i] = matrix
        print(covariances)
```

```
        # part b
        prior = []
        for i in range(10):
                prior += [len(data[i]) / len(trainImages)]
                print("The prior probability for ", i, "is", prior[i])

        # part c
        plt.matshow(covariances[2])
        plt.title('Problem 2. Confusion matrix for covariance matrix')
        plt.colorbar()
        plt.show()

        #part d.i

        def calculateAccuracy(expect, actual):
                same = [i for i in range(len(expect)) if expect[i] == actual[i]]
                return len(same) / len(actual)

        def errorrate(trainImages, actualLabels, validateX, validateY):
                data = {}
                for i in range(10):
                        data[i] = []
                for i in range(len(trainImages)):
                        key = actualLabels[i]
                        value = trainImages[i]
                        data[key].append(value)

                means = {}
                for i in range(10):
                        means[i] = np.sum(data[i], axis=0) / len(data[i])

                covariances = {}
                for i in range(10):
                        arrays = np.array(data[i])
                        mu = np.array(means[i])
                        matrix = (arrays - mu).T.dot((arrays - mu))
                        matrix = matrix / len(arrays)
                        covariances[i] = matrix

                prior = []
                for i in range(10):
                        prior += [len(data[i]) / len(trainImages)]

                avgCovariance = 0
                alpha = 0.0008

                for i in range(10):
                        avgCovariance += covariances[i] + alpha * np.identity(784)
                avgCovariance = avgCovariance / 10
```

```
            predict = []
            predict1 =[]
            var = [multivariate_normal(means[i], avgCovariance) for i in range(10)]
            var1 = [multivariate_normal(means[i], covariances[i] +\
                             alpha * np.identity(784)) for i in range(10)]
            for i in range(len(validateX)):
                    if i % 1000 == 0:
                            print(i)
                    pdfs = [var[j].logpdf(validateX[i]) + np.log(prior[j]) for j in range(10)]
                    predict += [np.argmax(pdfs)]

                    pdfs1 = [var1[j].logpdf(validateX[i]) + np.log(prior[j]) for j in range(10)]
                    predict1 += [np.argmax(pdfs1)]

            accuracy = calculateAccuracy(predict, validateY)
            accuracy1 = calculateAccuracy(predict1, validateY)
            return 1 - accuracy, 1 - accuracy1

    samplesSize = [100, 200, 500, 10000, 2000, 50000, 10000, 30000, 50000]
    errorrates1 = []
    errorrates2 = []
    for size in samplesSize:
            print("samplesSize is", size)
            error1, error2 = errorrate(trainImages[:size], actualLabels[:size], \
                    trainImages[50000:], actualLabels[50000:])
            errorrates1 += [error1]
            errorrates2 += [error2]

    plt.title('Problem 7.d.i The error rates on validation set \
            versus training example sizes\n average covariance matrix')
    plt.xlabel('Traning sample size')
    plt.ylabel('Error rate')
    plt.plot(samplesSize, errorrates1, 'bo')
    plt.show()

    plt.title('Problem 7.d.ii The error rates on validation set versus\
     training example sizes\n corresponding covariances matrix')
    plt.xlabel('Traning sample size')
    plt.ylabel('Error rate')
    plt.plot(samplesSize, errorrates2, 'bo')
    plt.show()
```

q7 for kaggle digit code

```
import scipy.io as sio
import numpy as np
from numpy.linalg import inv
import matplotlib.pyplot as plt
from sklearn.utils import shuffle
from sklearn import preprocessing
from scipy.stats import multivariate_normal
from sklearn.utils import shuffle
import csv

if __name__ == '__main__':
        #read input
        testData = sio.loadmat('./data/test.mat')
        trainData = sio.loadmat('./data/train.mat')

        trainImages = trainData['train_images']
        testImages = testData['test_images']
        testLabel = testData['test_labels']
        testLabels = np.transpose(testLabel,(1, 0))[0]

        x_dim = len(trainImages)
        y_dim = len(trainImages[0])
        image_index = len(trainImages[0][0])
        trainImages = trainImages.transpose((2, 0, 1))
        trainImages = trainImages.reshape(image_index, x_dim * y_dim)
        actualLabels = np.transpose(trainData['train_labels'],(1, 0))[0]
        trainImages, actualLabels = shuffle(trainImages, actualLabels)
        trainImages, actualLabels = shuffle(trainImages, actualLabels)

        #rotate test data to correct way
        for i in range(len(testImages)):
                testImages[i] = np.fliplr(np.rot90(np.rot90(np.rot90(testImages[i].reshape(28,28))))).reshape(2

        testImages = preprocessing.normalize(testImages.astype("float"))
        trainImages = preprocessing.normalize(trainImages.astype("float"))

        def calculateAccuracy(expect, actual):
                same = [i for i in range(len(expect)) if expect[i] == actual[i]]
                return len(same) / len(actual)

        def errorrate(trainImages, actualLabels, validateX, validateY):
                data = {}
                for i in range(10):
                        data[i] = []
                for i in range(len(trainImages)):
                        key = actualLabels[i]
                        value = trainImages[i]
                        data[key].append(value)
```

```

            means = {}
            for i in range(10):
                    means[i] = np.sum(data[i], axis=0) / len(data[i])

            covariances = {}
            for i in range(10):
                    arrays = np.array(data[i])
                    mu = np.array(means[i])
                    matrix = (arrays - mu).T.dot((arrays - mu))
                    matrix = matrix / len(arrays)
                    covariances[i] = matrix

            prior = []
            for i in range(10):
                    prior += [len(data[i]) / len(trainImages)]

            alpha = 0.0008

            predict =[]
            var = [multivariate_normal(means[i], covariances[i] + alpha * np.identity(784)) for i in range
            for i in range(len(validateX)):

                    pdfs = [var[j].logpdf(validateX[i]) + np.log(prior[j]) for j in range(10)]
                    predict += [np.argmax(pdfs)]
            accuracy = calculateAccuracy(predict, validateY)

            # creat Kaggle submission file
            predict_labels = predict
            indexs = [i for i in range(1, 10001)]
            data = []
            data += [indexs]
            data += [predict_labels]
            data = np.transpose(data, (1, 0)).tolist()
            first_row = [['Id', 'Category']]
            with open('digitpredict.csv', 'w') as f:
                a = csv.writer(f)
                a.writerows(first_row)
                a.writerows(data)

            return 1 - accuracy

    print(errorrate(trainImages[:60000], actualLabels[:60000], testImages, testLabels))

```

q7 for kaggle digit code

```
import scipy.io as sio
import numpy as np
from numpy.linalg import inv
import matplotlib.pyplot as plt
from sklearn.utils import shuffle
from sklearn import preprocessing
from scipy.stats import multivariate_normal
from sklearn.utils import shuffle
import csv

if __name__ == '__main__':
        #input data
        trainData = sio.loadmat('./data/spam_data.mat')
        trainImages = trainData['training_data']
        actualLabels = trainData['training_labels'][0]
        testImages = trainData['test_data']

        trainImages, actualLabels = shuffle(trainImages, actualLabels)
        trainImages, actualLabels = shuffle(trainImages, actualLabels)

        trainImages = preprocessing.normalize(trainImages.astype("float"))

        def predict(trainImages, actualLabels, testImages):
                data = {}
                for i in range(2):
                        data[i] = []
                for i in range(len(trainImages)):
                        key = actualLabels[i]
                        value = trainImages[i]
                        data[key].append(value)

                means = {}
                for i in range(2):
                        means[i] = np.sum(data[i], axis=0) / len(data[i])

                covariances = {}
                for i in range(2):
                        arrays = np.array(data[i])
                        mu = np.array(means[i])
                        matrix = (arrays - mu).T.dot((arrays - mu))
                        matrix = matrix / len(arrays)
                        covariances[i] = matrix

                prior = []
                for i in range(2):
                        prior += [len(data[i]) / len(trainImages)]

                alpha = 0.0008
```

```

        predict =[]
        var = [multivariate_normal(means[i], covariances[i] + alpha * np.identity(32)) for i in range(2
        for i in range(len(testImages)):
                pdfs = [var[j].logpdf(testImages[i]) + np.log(prior[j]) for j in range(2)]
                predict += [np.argmax(pdfs)]

        # creat Kaggle submission file
        predict_labels = predict
        indexs = [i for i in range(1, len(testImages) + 1)]
        data = []
        data += [indexs]
        data += [predict_labels]
        data = np.transpose(data, (1, 0)).tolist()
        first_row = [['Id', 'Category']]
        with open('spampredict.csv', 'w') as f:
            a = csv.writer(f)
            a.writerows(first_row)
            a.writerows(data)

predict(trainImages, actualLabels, testImages)
```