CS189/CS289A – Spring 2017 — Homework 3

Yaoyang Zhang, SID 3032114788

Problem 1

(a) X and Y are uncorrelated but they are dependent. The contingency table can be calculated easily as follows

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

Correlation between X and Y is

$$cor(X,Y) = \sum_x \sum_y (x-\mu_x)(y-\mu_y)p(x)p(y) = \sum_x \sum_y xyp(x)p(y) = 0$$

However, it is obvious that

$$P(X=0)P(Y=0) = \frac{1}{4} \neq P(X=0, Y=0) = 0$$

indicating that X and Y are not independent.

(b) The joint distribution of X, Y, Z is

$$P(X=0,Y=0,Z=0) = P(B=0)P(C=0)P(D=0) = \frac{1}{8}$$

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

$$P(X=1,Y=1,Z=0) = P(B=0)P(D=0)P(C=1) = \frac{1}{8}$$

$$P(X=1,Y=0,Z=1) = P(X=0,Y=1,Z=1) = P(X=1,Y=1,Z=0) = \frac{1}{8}$$

$$P(X=1,Y=1,Z=1) = 1 - \frac{1}{8} \times 4 = \frac{1}{2}$$

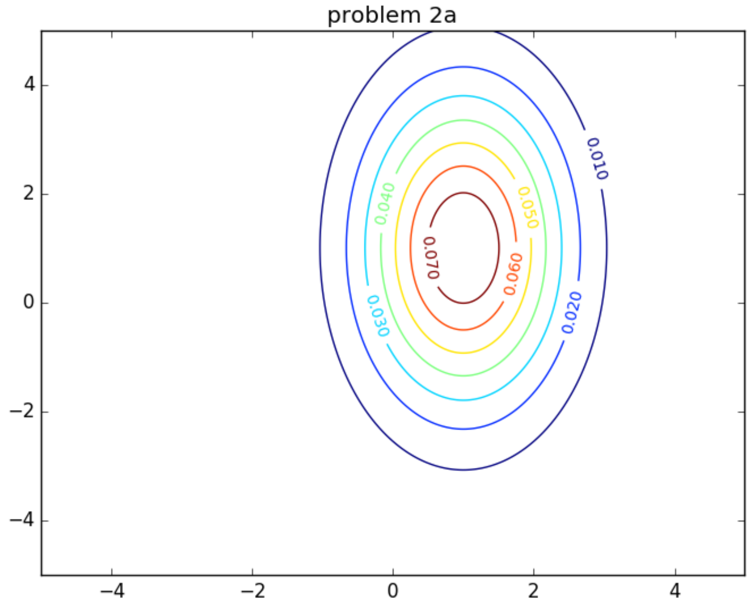
Marginal distribution can be calculated easily by summing out irrelevant variables. It can be shown that X, Y, Z are neither pairwise independent nor mutually independent.

$$P(X=1,Y=1) = \frac{1}{8} + \frac{1}{2} = \frac{5}{8} \neq P(X=1)P(Y=1) = \frac{3}{4} \times \frac{3}{4} = \frac{9}{16}$$

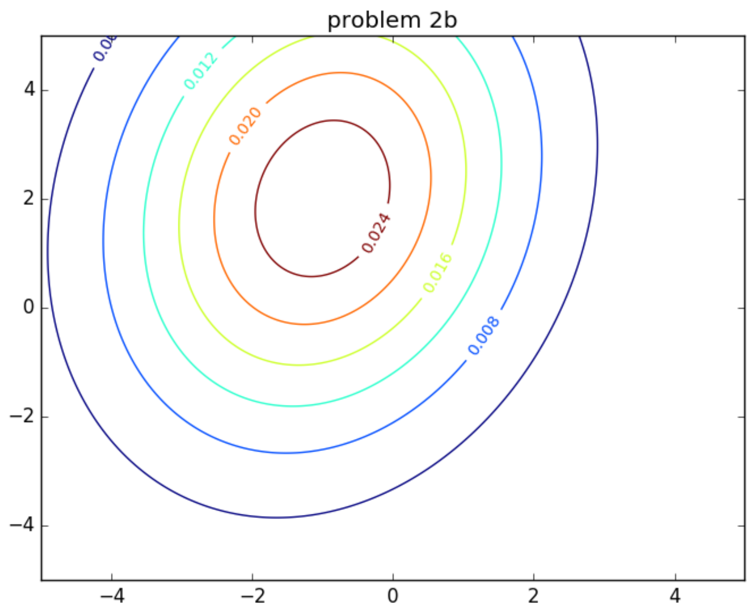
and

$$P(X=1,Y=1,Z=1) = \frac{1}{2} \neq P(X=1)P(Y=1)P(Z=1) = \frac{3}{4} \times \frac{3}{4} \times \frac{3}{4} = \frac{27}{64}$$

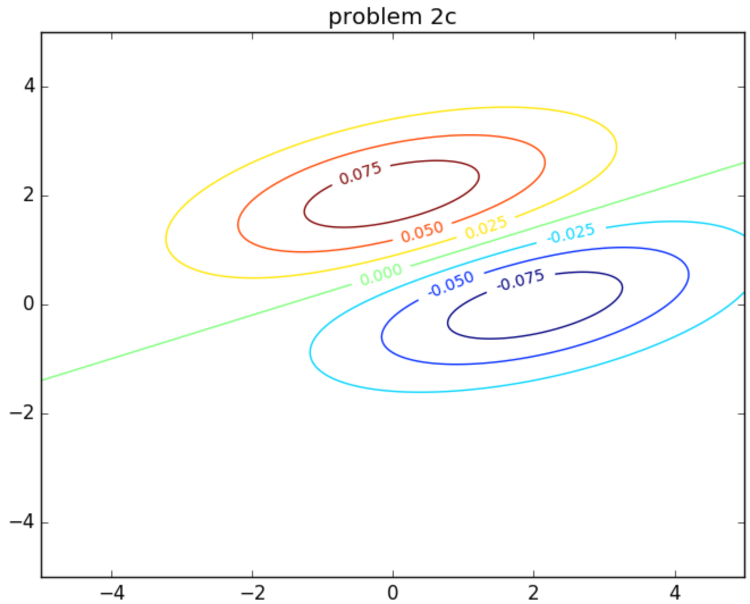
Problem 2


problem 2a
4
2
0
−2
−4
−4
−2
0
2
4
0.010
0.020
0.030
0.040
0.050
0.060
0.070

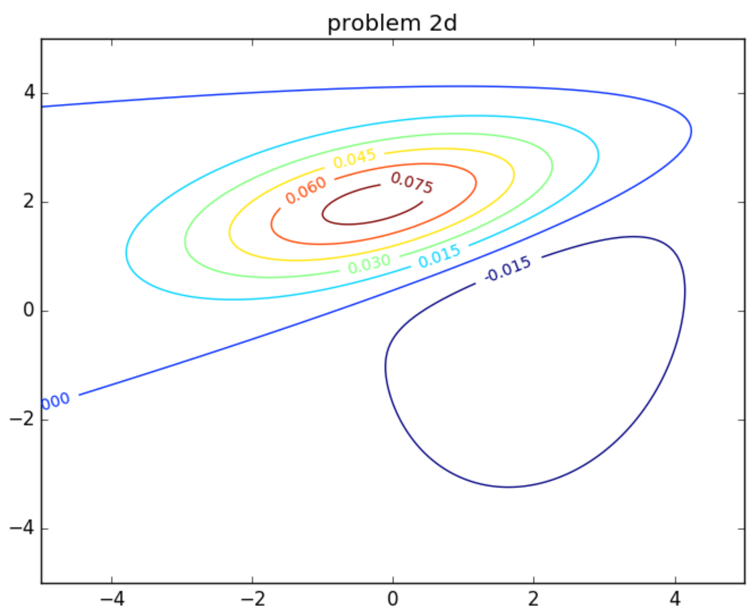
(a)


problem 2b
4
2
0
−2
−4
−4
−2
0
2
4
0.008
0.012
0.016
0.020
0.024

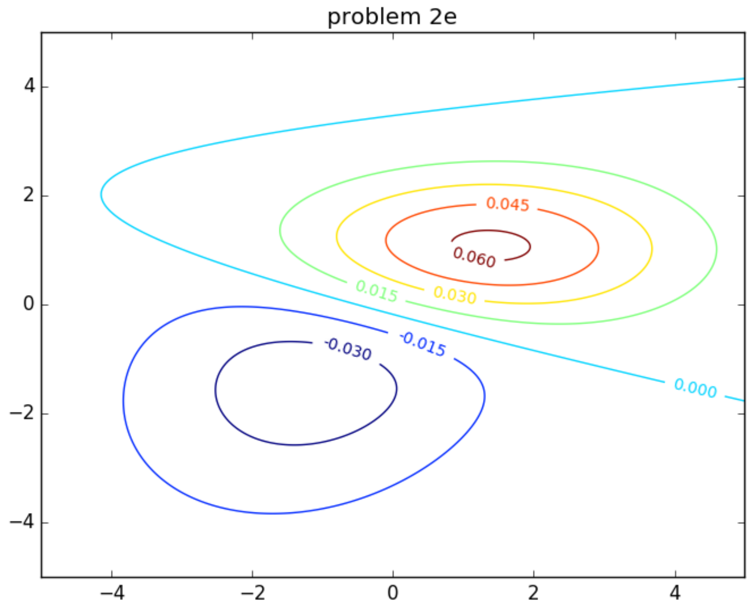
(b)


problem 2c
0.075
0.050
0.025
0.000
-0.025
-0.050
-0.075
4
2
0
−2
−4
−4
−2
0
2
4

(c)


problem 2d
0.045
0.060
0.075
0.030
0.015
-0.015
000
4
2
0
−2
−4
−4
−2
0
2
4

(d)



(e)

Problem 3

(a) The sample mean is $[2.81827412, 5.32312348]$

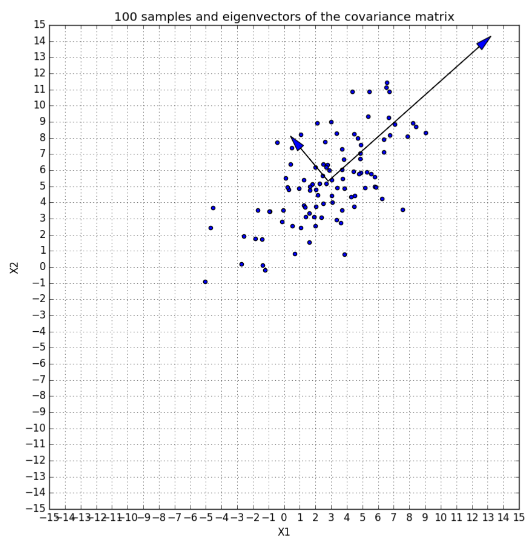
(b) The covariance matrix is

$$\begin{bmatrix} 8.45201673 & 4.97068676 \\ 4.97068676 & 6.9968489 \end{bmatrix}$$

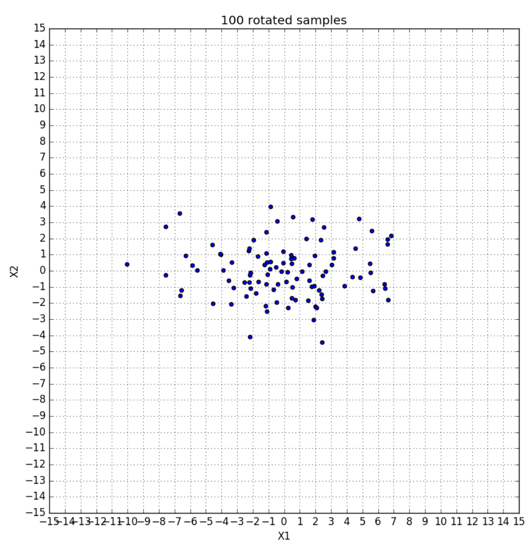
(c) The eigenvalues and eigenvectors are

$\lambda_1 = 12.74808738, \lambda_2 = 2.70077824, v_1 = [0.75658165, 0.65389923]^T, v_2 = [-0.65389923, 0.75658165]^T$

(d) 100 sample points and covariance eigenvectors



(e) 100 centered sample points



Problem 4

(a) Log-likelihood function

$$ll(\mu,\Sigma;x) = \ln\prod_{i=1}^{n} p(x;\mu,\Sigma) = \ln\prod_{i=1}^{n}\frac{1}{(\sqrt{2\pi})^d\sqrt{|\Sigma|}}exp\{-\frac{1}{2}(x_i-\mu)^T\Sigma^{-1}(x_i-\mu)\}$$

$$= \sum_{i=1}^{n}(-\frac{1}{2}(x_i-\mu)^T\Sigma^{-1}(x_i-\mu) - \frac{d}{2}\ln 2\pi - \frac{1}{2}\ln|\Sigma|)$$

$$= (\sum_{i=1}^{n} -\frac{1}{2}(x_i-\mu)^T\Sigma^{-1}(x_i-\mu)) - \frac{nd}{2}\ln 2\pi - \frac{n}{2}\ln|\Sigma|$$

$$= -\frac{1}{2}(\sum_{i=1}^{n}\sum_{j=1}^{d}(\frac{x_{ij}-\mu_j}{\sigma_j})^2) - \frac{nd}{2}\ln 2\pi - \frac{n}{2}\ln|\Sigma|$$

Set its gradient to 0

$$\frac{\partial}{\partial\mu_j}ll(\mu,\Sigma;x) = \sum_{i=1}^{n}(\frac{x_{ij}-\mu_j}{\sigma_j^2}) = 0, \forall j = 1...d$$

$$\frac{\partial}{\partial\sigma_j}ll(\mu,\Sigma;x) = \sum_{i=1}^{n}\frac{(x_{ij}-\mu_j)^2}{\sigma_j^3} - \frac{n}{\sigma_j} = 0, \forall j = 1...d$$

Solve for μ_j and σ_j we get

$$\hat{\mu} = [\hat{\mu}_1, ..., \hat{\mu}_d]^T = \frac{1}{n}\sum_{i=1}^{n} x_i$$

$$\hat{\Sigma} = \begin{bmatrix} \hat{\sigma}_1 & & & \\ & \hat{\sigma}_2 & & \\ & & \ddots & \\ & & & \hat{\sigma}_d \end{bmatrix} = \begin{bmatrix} \frac{1}{n}\sum_{i=1}^{n}(x_{i1}-\hat{\mu_1})^2 & & & \\ & \frac{1}{n}\sum_{i=1}^{n}(x_{i2}-\hat{\mu_2})^2 & & \\ & & \ddots & \\ & & & \frac{1}{n}\sum_{i=1}^{n}(x_{id}-\hat{\mu_d})^2 \end{bmatrix}$$

(b) Log-likelihood function

$$ll(\mu;x) = (\sum_{i=1}^{n} -\frac{1}{2}(x_i-A\mu)^T\Sigma^{-1}(x_i-A\mu)) - \frac{nd}{2}\ln 2\pi - \frac{n}{2}\ln|\Sigma|$$

Let $y_i = x_i - A\mu$, and apply chain rule

$$\nabla_\mu ll(\mu;x) = A^T\nabla_y ll(\mu;y) = A^T\nabla_y(\sum_{i=1}^{n} -\frac{1}{2}y_i^T\Sigma^{-1}y_i) = -A^T\sum_{i=1}^{n}\Sigma^{-1}y_i = -A^T\Sigma^{-1}(\sum_{i=1}^{n}x_i - nA\mu) = 0$$

Since Σ, A are invertible, we have

$$A\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$$

Problem 5

(a) If there is a certain direction along which the projection of $x_i - \mu$ for all the sample points x_i is 0, the covariance matrix is singular. That is, there is a vector in the feature space that is orthogonal to $x_i - \mu$ for all the sample points x_i.

(b) The covariance matrix is symmetric and semidefinite, thus it can be diagonalized as

$$\Sigma = V \Lambda V^T$$

where Λ is a diagonal matrix with non-negative elements and V is an orthonormal matrix. $V = [v_1, v_2, ..., v_n]$ where v_i is the eigenvector for eigenvalue λ_i. For all the non-zero eigenvalues $\lambda_{n_1}, \lambda_{n_2}, .., \lambda_{n_t}$, let $v_{n_1}, v_{n_2}, ..., v_{n_t}$ be the corresponding eigenvectors. For each sample point x_i, project x_i onto a t-dimensional space whose bases are $v_{n1}, v_{n2}, ..., v_{nt}$. That is,

$$y_i = \sum_{j=1}^{t} v_{n_j}^T x_i v_{n_j}, \forall i = 1, ..., N_s$$

Now in the t-dimensional feature space, the sample points y_i will have a $t \times t$ covariance matrix that is full rank.

(c) Maximizing the pdf $f(x)$ is equivalent to minimizing the following

$$\min_{||x||=1} x^T \Sigma^{-1} x$$

Since Σ is symmetric and positive definite, Σ^{-1} can be diagonalized as

$$\Sigma^{-1} = V \Lambda V^T$$

where V is an orthonormal matrix and Λ is a diagonal matrix whose elements are the reciprocals of the eigenvalues of Σ. The optimization problem is now

$$\min_{||x||=1} (V^T x)^T \Lambda V^T x$$

According to HW2, the solution is to set $v_j^T x = 1$ where j is the index of the largest eigenvalue of Σ^{-1} (or the index of the smallest eigenvalue of Σ), and $v_i^T x = 0$ where $i \neq j$. Since V is an orthonormal matrix, $x = v_j$.

Similarly, x that minimizes the pdf $f(x)$ is v_k where k is the index of the smallest eigenvalue of Σ^{-1} (or the index of the largest eigenvalue of Σ) and v_k is the corresponding eigenvector.

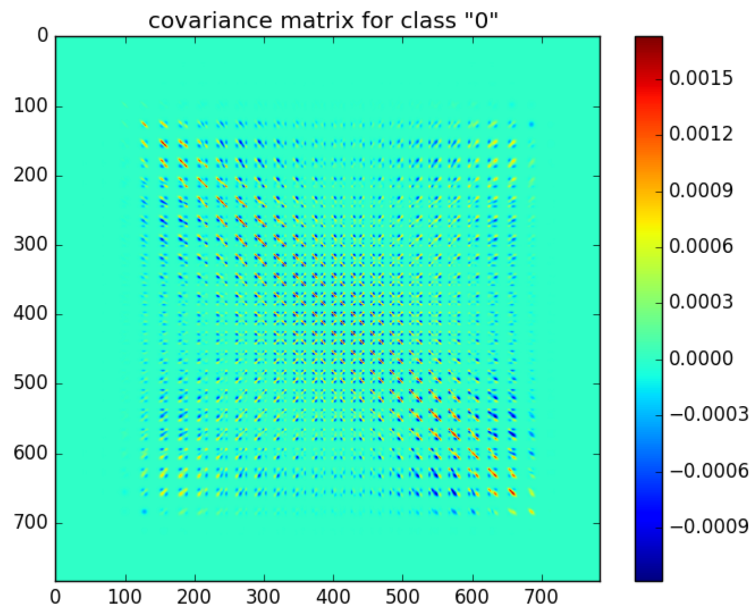
Problem 6

(a) The maximum likelihood estimate for μ_c and Σ_c is

$$\hat{\mu}_c = \frac{1}{n_c}\sum_{y_i=c} x_i, \hat{\Sigma}_c = \frac{1}{n_c}\sum_{y_i=c}(x_i - \hat{\mu}_c)(x_i - \hat{\mu}_c)^T$$

I normalized all the data points before calculating the mean and covariance matrix. See code for details

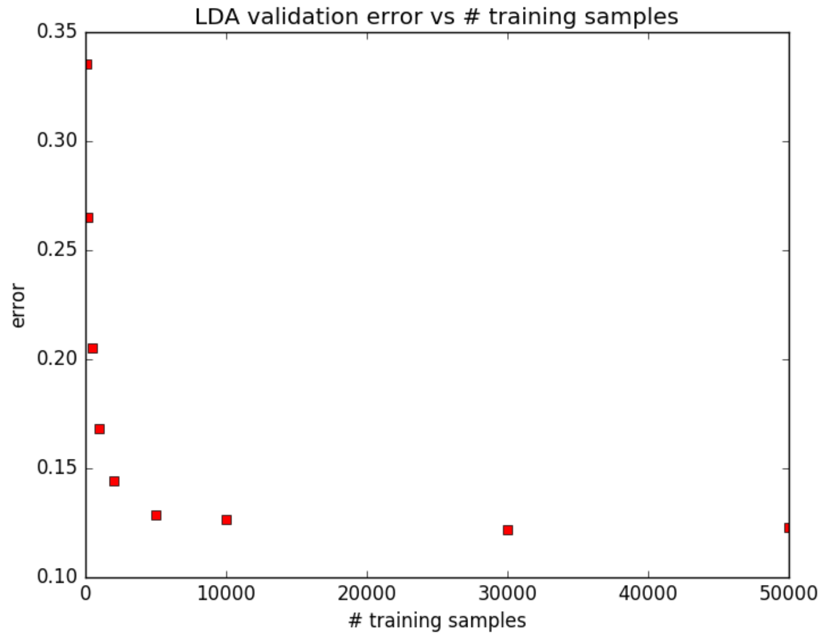
(b) A visualization of the covariance matrix for digit "0" is as follows



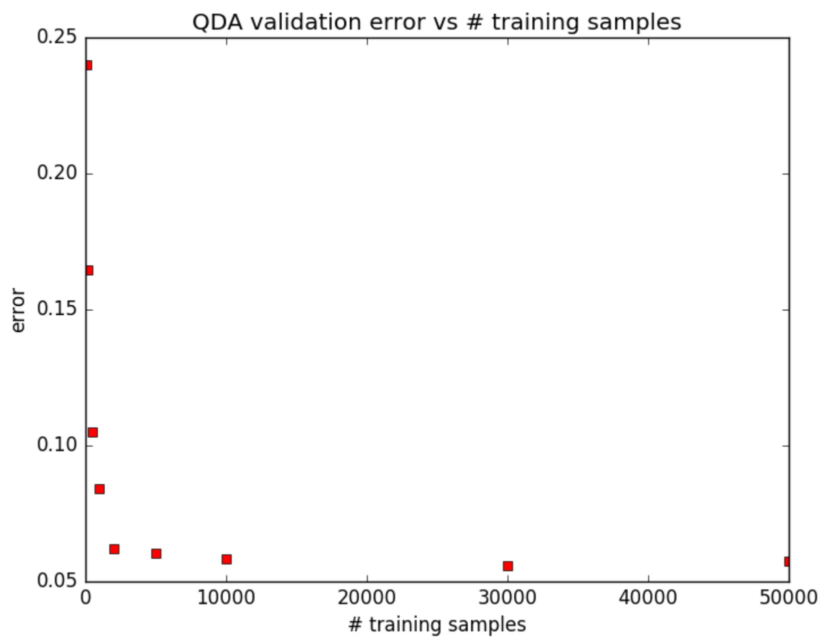
The diagonal terms are positive, but also contain zeros (because there is a lot of blank space in the same place for every image, i.e. with RGB values (0,0,0), which leads to a 0 in the diagonal). The off-diagonal terms are mostly negative, but the other diagonal elements are also positive. This might be due to the symmetry of shape of the image "0".

(c) In this part, I added a diagonal matrix γI to the covariance matrix $\hat{\Sigma}$ with a small value γ to avoid singularity of the covariance matrix. I tuned the parameter γ in order to get the lowest validation error, and the parameter I chose for this problem is $\gamma = 0.0001$

(i) The validation error of LDA versus the number of samples.



(ii) The validation error of QDA versus the number of samples.



(iii) QDA performs better than LDA. In this case, we should expect different covariance matrices for different classes since different digits would have different variations of the shades and blanks. LDA, however, assumes a same covariance matrix for all the classes and will under-fit in this case.

(iv) I used a QDA classifier for digits with $\gamma = 0.0001$ and no extra features. I scored 0.9422 on Kaggle and my username is YaoyangZhang.

(d) I used an LDA classifier for spam with $\gamma = 0.0001$ and extracted 104 most frequent words as the features from training samples. I scored 0.9254 on Kaggle and my username is YaoyangZhang. For feature extraction, see code 'featurize.py'.

Appendix

(a) python code for problem 2

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

# problem 2


delta = 0.01
x = np.arange(-5.0, 5.0, delta)
y = np.arange(-5.0, 5.0, delta)
X, Y = np.meshgrid(x, y)
Z1 = mlab.bivariate_normal(X, Y, 2.0, 1.0, 1.0, 1.0, 0.0)
Z2 = mlab.bivariate_normal(X, Y, 2.0, 2.0, -1.0, -1.0, 1.0)
Z = Z1 - Z2


plt.figure()
CS = plt.contour(X, Y, Z)
plt.clabel(CS, inline=1, fontsize=10)
plt.title('problem 2e')
plt.savefig('2e.png')
plt.show()
```

(b) python code for problem 3

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

# problem 3

X1 = []
X2 = []
for i in range(100):
    x1 = np.random.normal(3,3)
    x2 = x1/2 + np.random.normal(4,2)
    X1.append(x1)
    X2.append(x2)
sample = np.array([X1,X2])

mean = np.mean(sample, axis=1)
print(mean)
cov = np.cov(sample)
print(cov)
```

```
w,v = np.linalg.eig(cov)



v1 = v[:,0]
v1 = v1 / np.linalg.norm(v1) * w[0]

v2 = v[:,1]
v2 = v2 / np.linalg.norm(v2) * w[1]

print(w)
print(v)

fig = plt.figure(figsize=(10,10))
ax = fig.gca()
ax.set_xticks(np.arange(-15,16,1))
ax.set_yticks(np.arange(-15,16,1))
plt.grid()
plt.xlim(-15,15)
plt.ylim(-15,15)
plt.scatter(X1,X2)
ax =plt.axes()
ax.arrow(mean[0], mean[1], v1[0], v1[1],head_length=1,head_width=0.5)
ax.arrow(mean[0], mean[1], v2[0], v2[1],head_length=1,head_width=0.5)
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('100 samples and eigenvectors of the covariance matrix')
plt.savefig('p3_1.png')
plt.show()


for i in range(100):
    sample[:,i] = np.array(v.T * np.matrix(sample[:,i].reshape((2,1))- mean.r

fig = plt.figure(figsize=(10,10))
ax = fig.gca()
ax.set_xticks(np.arange(-15,16,1))
ax.set_yticks(np.arange(-15,16,1))
plt.grid()
plt.xlim(-15,15)
plt.ylim(-15,15)
plt.scatter(sample[0,:], sample[1,:])
ax =plt.axes()
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('100 rotated samples')
plt.savefig('p3_2.png')
plt.show()
```

(c) python code for "data_split.py"

```
import numpy as np
import scipy.io as sio


def preprocess_data(name, holdout = 0, sample=10000, dir='hw3_mnist_dist/trai
    data = sio.loadmat(dir)
    data_mnist = np.array(data['trainX'])
    np.random.seed(2)
    np.random.shuffle(data_mnist)
    y_validate = data_mnist[:holdout, -1]
    X_validate = data_mnist[:holdout, :-1]
    y_train = data_mnist[holdout:holdout+sample, -1]
    X_train = data_mnist[holdout:holdout+sample, :-1]

    res = {}
    res['X_train'] = X_train
    res['y_train'] = y_train
    res['X_validate'] = X_validate
    res['y_validate'] = y_validate
    sio.savemat('data_class_'+name+'.mat', res)
```

(d) python code for "LDA.py"

```
import numpy as np
from sklearn import preprocessing
import matplotlib.pyplot as plt

class LDA:
    sigma = None
    mu = None
    pi = None
    n_feature =0
    n_class = 0
    h = 0
    def __init__(self,c,f,h):
        self.n_class = c
        self.n_feature = f
        self.sigma = np.zeros((self.n_feature, self.n_feature))
        self.mu = np.zeros((self.n_class, self.n_feature))
        self.pi = np.zeros((self.n_class, 1))
        self.h = h


    def fit(self,X,y):
        X_norm = preprocessing.normalize(X)
        X_class = {}
        covariance = np.zeros((self.n_class, self.n_feature, self.n_feature))
```

```
        for i in range(y.shape[0]):
            if y[i] in X_class:
                X_class[y[i]].append(X_norm[i, :].tolist())
            else:
                X_class[y[i]] = []
                X_class[y[i]].append(X_norm[i, :].tolist())
        for key in X_class:
            X_class[key] = np.array(X_class[key])
        for i in range(self.n_class):
            self.pi[i] = X_class[i].shape[0] / X_norm.shape[0]
        for i in range(self.n_class):
            self.mu[i, :] = np.mean(X_class[i], axis=0)
            covariance[i, :, :] = np.cov(X_class[i].T)



        for i in range(self.n_class):
            self.sigma[:, :] += covariance[i, :, :] * self.pi[i]

        self.sigma[:, :] += np.identity(self.n_feature) * self.h

    def predict(self, X):
        X_norm = preprocessing.normalize(X)
        inv_cov = np.linalg.inv(np.matrix(self.sigma))
        const = np.zeros((self.n_class,1))
        for i in range(self.n_class):
            const[i] = 0.5 * np.matrix(self.mu[i]) * inv_cov * np.matrix(self

        const1 = np.zeros((self.n_class,self.n_feature))
        for i in range(self.n_class):
            const1[i] = np.matrix(self.mu[i]) * inv_cov

        predict_res = np.zeros((X.shape[0],1))
        for i in range(X_norm.shape[0]):
            index = 0
            val = const1[0] * np.matrix(X_norm[i]).T - const[0] + np.log(self
            for j in range(1,self.n_class):
                temp_val = const1[j] * np.matrix(X_norm[i]).T - const[j] +np.
                if temp_val > val:
                    val = temp_val
                    index = j
            predict_res[i] = index

        return predict_res
```

(e) python code for "QDA.py"

```
import numpy as np
```

```python
from sklearn import preprocessing
import matplotlib.pyplot as plt

class LDA:
    sigma = None
    mu = None
    pi = None
    n_feature =0
    n_class = 0
    h = 0
    def __init__(self,c,f,h):
        self.n_class = c
        self.n_feature = f
        self.sigma = np.zeros((self.n_feature, self.n_feature))
        self.mu = np.zeros((self.n_class, self.n_feature))
        self.pi = np.zeros((self.n_class, 1))
        self.h = h


    def fit(self,X,y):
        X_norm = preprocessing.normalize(X)
        X_class = {}
        covariance = np.zeros((self.n_class, self.n_feature, self.n_feature))
        for i in range(y.shape[0]):
            if y[i] in X_class:
                X_class[y[i]].append(X_norm[i, :].tolist())
            else:
                X_class[y[i]] = []
                X_class[y[i]].append(X_norm[i, :].tolist())
        for key in X_class:
            X_class[key] = np.array(X_class[key])
        for i in range(self.n_class):
            self.pi[i] = X_class[i].shape[0] / X_norm.shape[0]
        for i in range(self.n_class):
            self.mu[i, :] = np.mean(X_class[i], axis=0)
            covariance[i, :, :] = np.cov(X_class[i].T)



        for i in range(self.n_class):
            self.sigma[:, :] += covariance[i, :, :] * self.pi[i]

        self.sigma[:, :] += np.identity(self.n_feature) * self.h

    def predict(self, X):
        X_norm = preprocessing.normalize(X)
        inv_cov = np.linalg.inv(np.matrix(self.sigma))
        const = np.zeros((self.n_class,1))
```

```
        for i in range(self.n_class):
            const[i] = 0.5 * np.matrix(self.mu[i]) * inv_cov * np.matrix(self

        const1 = np.zeros((self.n_class,self.n_feature))
        for i in range(self.n_class):
            const1[i] = np.matrix(self.mu[i]) * inv_cov

        predict_res = np.zeros((X.shape[0],1))
        for i in range(X_norm.shape[0]):
            index = 0
            val = const1[0] * np.matrix(X_norm[i]).T - const[0] + np.log(self
            for j in range(1,self.n_class):
                temp_val = const1[j] * np.matrix(X_norm[i]).T - const[j] +np.
                if temp_val > val:
                    val = temp_val
                    index = j
            predict_res[i] = index

        return predict_res
```

(f) python code for problem 6c

```
import numpy as np
from data_split import preprocess_data
from LDA import LDA
from QDA import QDA
import scipy.io as sio
import matplotlib.pyplot as plt
from sklearn import discriminant_analysis
import csv

# problem 6ci, 6cii


num = [100,200,500,1000,2000,5000,10000,30000,50000]


error_LDA =[]
error_QDA =[]
for i in range(9):
    preprocess_data('1', 10, num[i])
    data = sio.loadmat('data_class_1.mat')
    X_train = data['X_train']
    y_train = data['y_train'][0]
    X_validate = data['X_validate']
    y_validate = data['y_validate'][0]

    sk_cls = discriminant_analysis.QuadraticDiscriminantAnalysis()
```

```
        sk_cls.fit(X_train, y_train)
        y_predict_1 = sk_cls.predict(X_validate)


        cls = LDA(10, 784, 0.0001)
        cls.fit(X_train, y_train)
        y_predict = cls.predict(X_validate)


        v_error = 0
        for j in range(10000):
            v_error += (int(y_validate[j]) != int(y_predict[j]))
        v_error /= 10000
        error_LDA.append(v_error)
        print(v_error)


        cls = QDA(10,784,0.0001)
        cls.fit(X_train, y_train)
        y_predict = cls.predict(X_validate)
        v_error = 0
        for j in range(10000):
            v_error += (int(y_validate[j]) != int(y_predict[j]))
        v_error /= 10000
        error_QDA.append(v_error)
        print(v_error)

    plt.plot(num, error_LDA,'rs')
    plt.xlabel('# training samples')
    plt.ylabel('error')
    plt.title('LDA validation error vs # training samples')
    plt.savefig('LDA_mnist_3.png')
    plt.show()

    plt.plot(num, error_QDA,'rs')
    plt.xlabel('# training samples')
    plt.ylabel('error')
    plt.title('QDA validation error vs # training samples')
    plt.savefig('QDA_mnist_3.png')
    plt.show()



    # problem 6civ, kaggle


    data = sio.loadmat('hw3_mnist_dist/train.mat')
    train_X = data['trainX']
```

```
train_y = train_X[:,-1]
train_X = train_X[:,:-1]
data = sio.loadmat('hw3_mnist_dist/test.mat')
test_X = data['testX']
cls = QDA(10,784,0.0001)
cls.fit(train_X,train_y)
predict = cls.predict(test_X)
with open('mnist_predict_QDA.csv','wt') as f:
    writer = csv.writer(f, delimiter=',')
    writer.writerow(['Id','Category'])
    for i in range(predict.shape[0]):
        writer.writerow([i, int(predict[i][0])])
```

(g) python code for problem 6d

```
import numpy as np
from data_split import preprocess_data
from LDA import LDA
from QDA import QDA
import scipy.io as sio
import csv


# problem 6d

data = sio.loadmat('spam/spam_data_2.mat')
train_X = data['training_data']
train_y = data['training_labels'][0]
test_X = data['test_data']
# validate_X = train_X[:5000,:]
# validate_y = train_y[:5000]
# train_X = train_X[5000:,:]
# train_y = train_y[5000:]



cls_lda = LDA(2, 104,0.0001)
cls_lda.fit(train_X,train_y)
y_predict = cls_lda.predict(test_X)


# error = 0
# for i in range(validate_X.shape[0]):
#     error += (int(validate_y[i]) != int(y_predict[i]))
# error /= validate_X.shape[0]
# print(error)

with open('spam_predict.csv','wt') as f:
```

```
        writer = csv.writer(f, delimiter=',')
        writer.writerow(['Id','Category'])
        for i in range(y_predict.shape[0]):
            writer.writerow([i, int(y_predict[i][0])])
```

(h) python code for ”featurize.py”

```
'''
**************** PLEASE READ ***************

Script that reads in spam and ham messages and converts each training example
into a feature vector

Code intended for UC Berkeley course CS 189/289A: Machine Learning

Requirements:
-scipy ('pip install scipy')

To add your own features, create a function that takes in the raw text and
word frequency dictionary and outputs a int or float. Then add your feature
in the function 'def generate_feature_vector'

The output of your file will be a .mat file. The data will be accessible using
the following keys:
    -'training_data'
    -'training_labels'
    -'test_data'

'''

from collections import defaultdict
import glob
import re
import scipy.io

NUM_TEST_EXAMPLES = 10000

BASE_DIR = './'
SPAM_DIR = 'spam/'
HAM_DIR = 'ham/'
TEST_DIR = 'test/'

# ************* Features *************

# Features that look for certain words
def freq_pain_feature(text, freq):
    return float(freq['pain'])
```

```
def freq_private_feature(text, freq):
    return float(freq['private'])

def freq_bank_feature(text, freq):
    return float(freq['bank'])

def freq_money_feature(text, freq):
    return float(freq['money'])

def freq_drug_feature(text, freq):
    return float(freq['drug'])

def freq_spam_feature(text, freq):
    return float(freq['spam'])

def freq_prescription_feature(text, freq):
    return float(freq['prescription'])

def freq_creative_feature(text, freq):
    return float(freq['creative'])

def freq_height_feature(text, freq):
    return float(freq['height'])

def freq_featured_feature(text, freq):
    return float(freq['featured'])

def freq_differ_feature(text, freq):
    return float(freq['differ'])

def freq_width_feature(text, freq):
    return float(freq['width'])

def freq_other_feature(text, freq):
    return float(freq['other'])

def freq_energy_feature(text, freq):
    return float(freq['energy'])

def freq_business_feature(text, freq):
    return float(freq['business'])

def freq_message_feature(text, freq):
    return float(freq['message'])

def freq_volumes_feature(text, freq):
    return float(freq['volumes'])
```

```python
def freq_revision_feature(text, freq):
    return float(freq['revision'])

def freq_path_feature(text, freq):
    return float(freq['path'])

def freq_meter_feature(text, freq):
    return float(freq['meter'])

def freq_memo_feature(text, freq):
    return float(freq['memo'])

def freq_planning_feature(text, freq):
    return float(freq['planning'])

def freq_pleased_feature(text, freq):
    return float(freq['pleased'])

def freq_record_feature(text, freq):
    return float(freq['record'])

def freq_out_feature(text, freq):
    return float(freq['out'])

# Features that look for certain characters
def freq_semicolon_feature(text, freq):
    return text.count(';')

def freq_dollar_feature(text, freq):
    return text.count('$')

def freq_sharp_feature(text, freq):
    return text.count('#')

def freq_exclamation_feature(text, freq):
    return text.count('!')

def freq_para_feature(text, freq):
    return text.count('(')

def freq_bracket_feature(text, freq):
    return text.count('[')

def freq_and_feature(text, freq):
    return text.count('&')

# --------- Add your own feature methods ----------
def example_feature(text, freq):
```

```
        return int('example' in text)

    # Generates a feature vector
    def generate_feature_vector(text, freq):
        words1 = []
        words2 = []
        with open('ham_dict.txt','rt') as f:
            for line in f:
                line = line.strip()
                words1.append(line)
        with open('spam_dict.txt','rt') as f:
            for line in f:
                line = line.strip()
                words2.append(line)
        words = set(words1) - set(words2)
        feature = []
        feature.append(freq_pain_feature(text, freq))
        feature.append(freq_private_feature(text, freq))
        feature.append(freq_bank_feature(text, freq))
        feature.append(freq_money_feature(text, freq))
        feature.append(freq_drug_feature(text, freq))
        feature.append(freq_spam_feature(text, freq))
        feature.append(freq_prescription_feature(text, freq))
        feature.append(freq_creative_feature(text, freq))
        feature.append(freq_height_feature(text, freq))
        feature.append(freq_featured_feature(text, freq))
        feature.append(freq_differ_feature(text, freq))
        feature.append(freq_width_feature(text, freq))
        feature.append(freq_other_feature(text, freq))
        feature.append(freq_energy_feature(text, freq))
        feature.append(freq_business_feature(text, freq))
        feature.append(freq_message_feature(text, freq))
        feature.append(freq_volumes_feature(text, freq))
        feature.append(freq_revision_feature(text, freq))
        feature.append(freq_path_feature(text, freq))
        feature.append(freq_meter_feature(text, freq))
        feature.append(freq_memo_feature(text, freq))
        feature.append(freq_planning_feature(text, freq))
        feature.append(freq_pleased_feature(text, freq))
        feature.append(freq_record_feature(text, freq))
        feature.append(freq_out_feature(text, freq))
        feature.append(freq_semicolon_feature(text, freq))
        feature.append(freq_dollar_feature(text, freq))
        feature.append(freq_sharp_feature(text, freq))
        feature.append(freq_exclamation_feature(text, freq))
        feature.append(freq_para_feature(text, freq))
        feature.append(freq_bracket_feature(text, freq))
        feature.append(freq_and_feature(text, freq))
```

```
    # ------------ Add your own features here ------------
    # Make sure type is int or float

    for word in words:
        feature.append(freq[word])

    return feature

# generate the most frequently used words in a document
def generate_most_freq(filenames, name):
    word_freq = defaultdict(int)
    res = []
    for filename in filenames:
        with open(filename, "r", encoding='utf-8', errors='ignore') as f:
            text = f.read() # Read in text from file
            text = text.replace('\r\n', ' ') # Remove newline character
            words = re.findall(r'\w+', text)
             # Frequency of all words
            for word in words:
                word_freq[word] += 1
    for w in sorted(word_freq, key=word_freq.get, reverse=True):
        res.append(w)
    with open(name+'_dict.txt','w') as f:
        for i in range(200):
            f.write(res[i]+'\n')


# This method generates a design matrix with a list of filenames
# Each file is a single training example
def generate_design_matrix(filenames):
    design_matrix = []
    for filename in filenames:
        with open(filename, "r", encoding='utf-8', errors='ignore') as f:
            text = f.read() # Read in text from file
            text = text.replace('\r\n', ' ') # Remove newline character
            words = re.findall(r'\w+', text)
            word_freq = defaultdict(int) # Frequency of all words
            for word in words:
                word_freq[word] += 1

            # Create a feature vector
            feature_vector = generate_feature_vector(text, word_freq)
            design_matrix.append(feature_vector)
    return design_matrix

# ************** Script starts here **************
# DO NOT MODIFY ANYTHING BELOW
```

```
spam_filenames = glob.glob(BASE_DIR + SPAM_DIR + '*.txt')
ham_filenames = glob.glob(BASE_DIR + HAM_DIR + '*.txt')
generate_most_freq(spam_filenames,'spam')
generate_most_freq(ham_filenames,'ham')

spam_design_matrix = generate_design_matrix(spam_filenames)
ham_design_matrix = generate_design_matrix(ham_filenames)


# Important: the test_filenames must be in numerical order as that is the
# order we will be evaluating your classifier
test_filenames = [BASE_DIR + TEST_DIR + str(x) + '.txt' for x in range(NUM_TE
test_design_matrix = generate_design_matrix(test_filenames)

X = spam_design_matrix + ham_design_matrix
Y = [1]*len(spam_design_matrix) + [0]*len(ham_design_matrix)

file_dict = {}
file_dict['training_data'] = X
file_dict['training_labels'] = Y
file_dict['test_data'] = test_design_matrix
scipy.io.savemat('spam_data_2.mat', file_dict, do_compression=True)
```