Jesse Li

SID: 23822462

HW 3 Writeup

1a) X and Y are not independent, knowing the value of X or Y affects the probability of the other variable. For example, if we know that X has a value of 1, then the variable Y has to have a value of zero. If they were independent, they would satisfy the relationship:

P(x|y) = P(x)   and P(y|x) = P(y)

Since P(y = 0|x = 1) = 1 and P(y = 0) = ½, we can see that the two variables are not independent. We can, however, show that X and Y are uncorrelated. Two random variables are uncorrelated if cov(X,Y) = 0.

cov(X,Y) = E(XY) - E(X)E(Y)

P(x=1) = ¼

P(x=0) = ½

P(x = -1) = ¼

P(y=1) = ¼

P(y=0) = ½

P(y = -1) = ¼

P(x = 1, y = 0) = ¼

P(x = -1, y = 0) = ¼

P(x = 0, y = 1) = ¼

P(x = 0, y = -1) = ¼

E(X) = P(x = 1)\*1 + P(x = 0)\*0 + P(x = -1)\*-1 = 0

Similarly, E(Y) = 0

E(XY) =  0, since either X or Y will be 0

Therefore, cov(X,Y) = E(XY) - E(X)E(Y) = 0

1b)

B1,B2,B3 = 0,0,0  -> X,Y,Z = 0,0,0

B1,B2,B3 = 1,0,0  -> X,Y,Z = 1,0,1

B1,B2,B3 = 0,1,0  -> X,Y,Z = 1,1,0

B1,B2,B3 = 0,0,1  -> X,Y,Z = 0,1,1

B1,B2,B3 = 1,1,0  -> X,Y,Z = 0,1,1

B1,B2,B3 = 1,0,1  -> X,Y,Z = 1,1,0

B1,B2,B3 = 0,1,1  -> X,Y,Z = 1,0,1

B1,B2,B3 = 1,1,1  -> X,Y,Z = 0,0,0

P(X = 1) = ½ P(X = 0) = ½

P(Y = 1) = ½ P(Y = 0) = ½

P(Z = 1) = ½  P(Z = 0) = ½

Consider the pair X,Y:

P(X = 1, Y = 1) = ¼ = P(X=1)P(Y=1)

P(X = 0, Y = 1) = ¼ = P(X=0)P(Y=1)

P(X = 0, Y = 0) = ¼ = P(X=0)P(Y=0)

P(X = 1, Y = 0) = ¼  = P(X=1)P(Y=0)

So P(X,Y) = P(X)P(Y), meaning X and Y are independent.

From symmetry of the problem, we can then see that this is true for the pairs Y,Z and X,Z as well, so X,Y,Z are pairwise independent, i.e. :

P(X,Y) =  P(X)P(Y)

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

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

X,Y,Z are not mutually independent since P(X,Y,Z) =/= P(X)P(Y)P(Z).

For example:

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

P(X = 1)P(Y=1) P(Z=1) = 1/8

P(X=1,Y=1,Z=1) =/= P(X=1)P(Y=1)P(Z=1)

2)







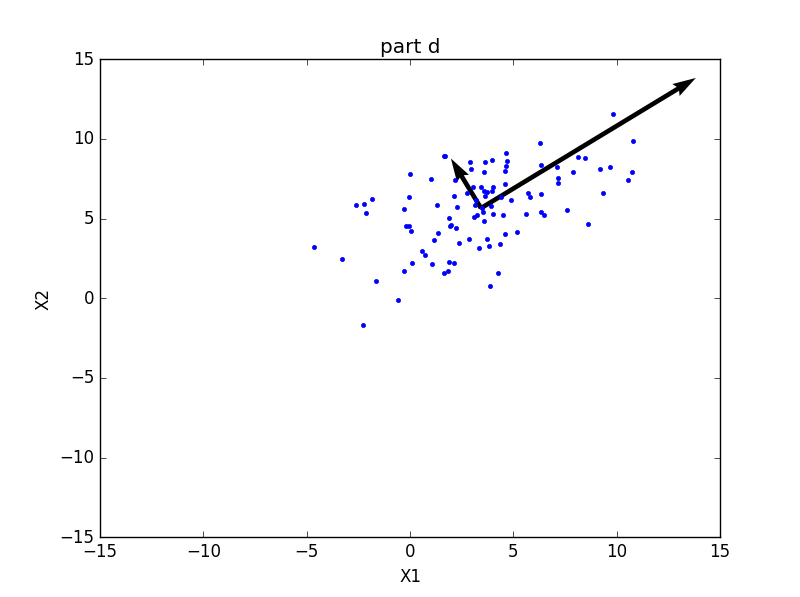


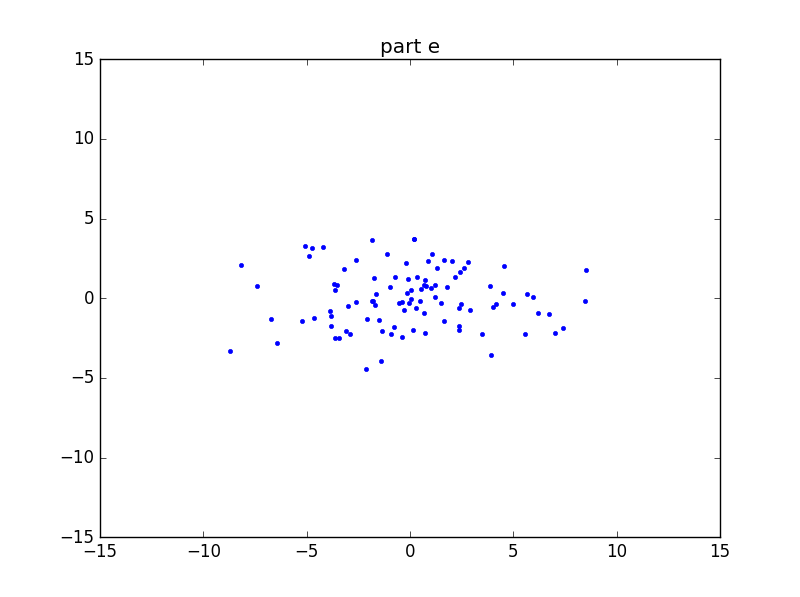


3) a) mean: [3.137, 5.698]

b) covariance matrix:

c) eigenvalues: [10.879 3.024]

eigenvectors: 



4) a)

For all *a* ∈ Rn, aTΣa = E**,** so aTΣa ≥ 0.

Since aTΣa ≥ 0, we know that Σ is positive semidefinite. In homework 2, we showed that all positive definite matrices are invertible. Therefore, we know that Σ-1 will not exist when Σ is positive semidefinite but not positive definite (one or more of the eigenvalues is 0 and one or more of the diagonal entries is 0). In order to ensure that Σ-1 exists, we must transform X into X’ such that the diagonals of Σ are always positive. Essentially, we need to find X’ such that cov(Xi’, Xi’) = E((xi-μi)2) > 0. We can linearly transform X by adding a positive constant γ to each Xi which will give us an X’ that ensures an invertible covariance matrix.

b) xTΣ-1x = ||Ax|| = xTATAx 🡪 ATA = Σ-1

From the spectral decomposition theorem, we know that we can express Σ-1 as UΛ-1UT, where U is an array with the eigenvectors of Σ-1 as columns and Λ is a diagonal matrix with the eigenvalues of Σ-1 as diagonal entries. Let B be a diagonal matrix with the square roots of the eigenvalues of Σ-1 as diagonal entries.

ATA = Σ-1 = UΛ-1UT = U(B-1)TB-1UT = (B-1 UT )T B-1UT

Therefore, A = B-1UT.

c) The intuitive meaning of xTΣ-1x is a measure of how far the point x is from the mean of the Gaussian.

d) The maximum value of ||Ax|| is the largest eigenvalue of Σ-1 and the minimum value is the smallest eigenvalue. If Xi and Xj are orthogonal, then they are independent and the intuitive meaning for the maximum and minimum of ||Ax|| is a measure of the higher variance and lower variance corresponding to Xi and Xj. From the equation for f(x), we can see that minimizing ||Ax|| will maximize f(x), so we will choose x corresponding to the smallest eigenvalue of Σ-1.

5) a) The maximum likelihood estimates for the mean and covariance matrix of a Gaussian distribution are simply the mean and covariance matrix of the sample data. For i.i.d observations X1, …, Xn from the same class, the mean will have the same dimensions as Xi and the elements will be the averages of each feature that is observed. The covariance matrix will have entries Σij which correspond to the covariance between feature i and feature j across all n samples. For class k, the mean and covariance matrix are given by:

b) The prior distribution for each class i is given by (number of training samples in class i)/(total samples). Using all 60,000 training samples, we find the priors to be:

class 0: 0.0987

class 1: 0.1124

class 2: 0.0993

class 3: 0.1022

class 4: 0.0974

class 5: 0.0904

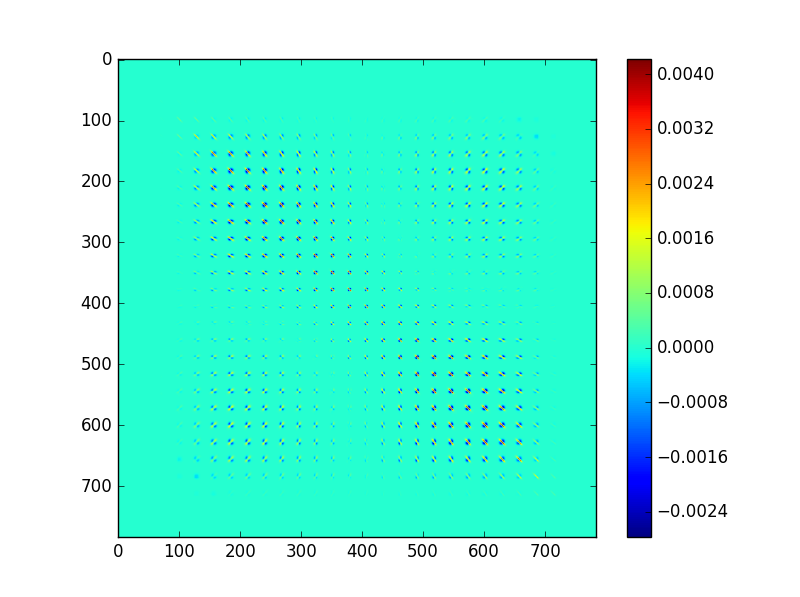
class 6: 0.0986

class 7: 0.1044

class 8: 0.0975

class 9: 0.0992

c) This is the covariance matrix for the digit 1.



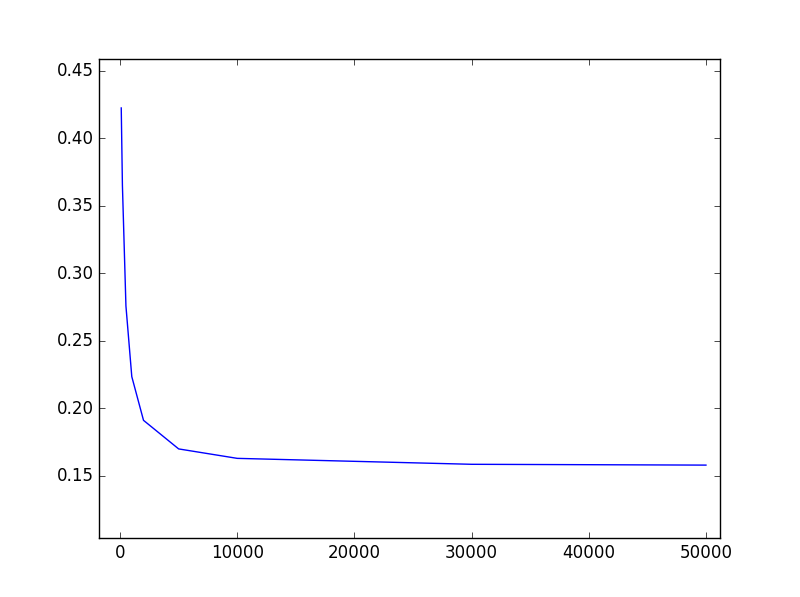
The diagonal entries of the matrix represent the variances of each feature. From this diagonal, we can see that the variance is highest where people would generally start writing and where they would finish writing. This intuitively makes sense, as people tend to draw their ones at different angles and different lengths. However, the midsection of a one usually passes through the center of the image regardless of the angle the one is drawn at. This can be seen in the structure of the covariance matrix, as the covariance matrix values decrease in the middle of the matrix.

d) i) Here we are using linear discriminant analysis to classify our images. We find the likelihood of each class for each sample and then classify the sample with the class of highest likelihood. The decision boundary in this case separates our 784-dimensional space into 10 sections (one for each digit). The implementation is a linear discriminant analysis classifier, so the decision boundary takes a linear form. The boundary takes on this form because the same covariance matrix was used for each of the Gaussians associated with each digit.

Error rates: [0.4386, 0.3777, 0.2911, 0.2301, 0.1936, 0.1731, 0.1664, 0.1615, 0.1604]

These error rates correspond to the training set sizes:

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



Part di

Training Set Size

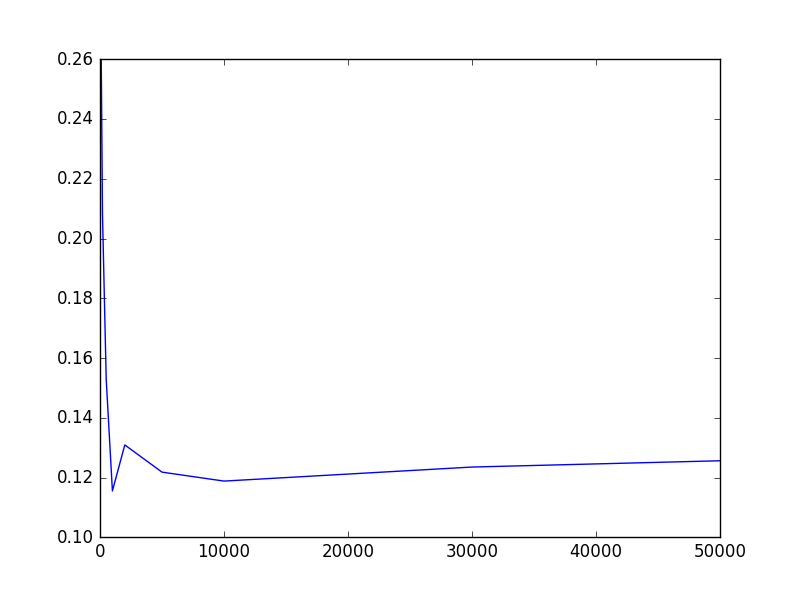
Error

ii) In this part, we have implemented quadratic discriminant analysis. Our algorithm for classifying is essentially the same as that of LDA, but by allowing each class to have its own covariance matrix, we create quadratic decision boundary surfaces.

Error rates: [0.2600, 0.2082, 0.1529, 0.1155, 0.1309, 0.1218, 0.1188, 0.1235, 0.1256]

These error rates correspond to the training set sizes:

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



Part dii

Training Set Size

Error

iii) The error rate in part (ii) is much lower than that of part (i) for small training set sizes. The decision boundaries are linear in part (i) and quadratic in part (ii). As a result, the decision boundary chosen in part (ii) is more complex and results in better performance, especially for small training set sizes where underfitting is more significant. This increased decision boundary can also result in more overfitting, which can be seen by the slightly increasing error rate in part (ii).

iv) QDA validation error: 0.1296.

QDA Kaggle score: 0.88320

LDA validation error: 0.1598

LDA Kaggle score: 0.85320

e) validation error: 0.2005

Kaggle score: 0.74240

6.1) Want to find w, α that minimizes: . We can insert a column of ones to the end of X to absorb the α term.

Find w that minimizes |Xw – y|2 🡪 Find w that minimizes RSS = (Xw−y)T(Xw−y)

(Xw−y)T(Xw−y) is minimized when XTXw = XTy 🡪 If XTX is invertible, w = (XTX)-1XTy

Using the training data, we find that w = [4.06e+04, 1.20e+03, -8.50, 1.18e+02, -3.79e+01, 4.31e+01, -4.22e+04, -4.25e+04, -3.57e+06]

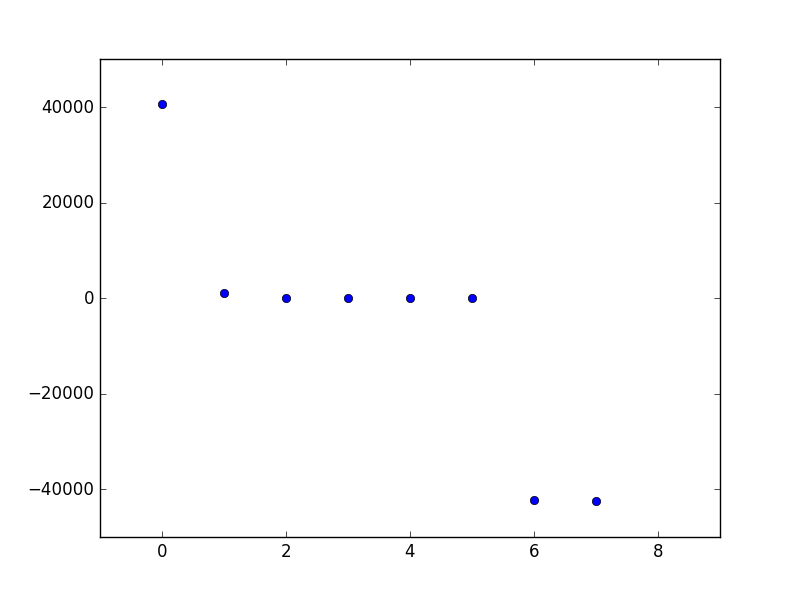
6.2) RSS: 5.795e+12

predicted min: -56562.827545

predicted max: 710798.838689

These numbers make sense. Dividing the RSS by the number of samples and then taking the square root gives us the average error per data point, which comes out to be $69,492. This is a reasonable amount of error for the value of a home. The actual range for the validation set is $28,300 to $500,001. The actual range differs significantly from the predicted range, which suggests that the median home value is not linear at the high and low ends of the spectrum.

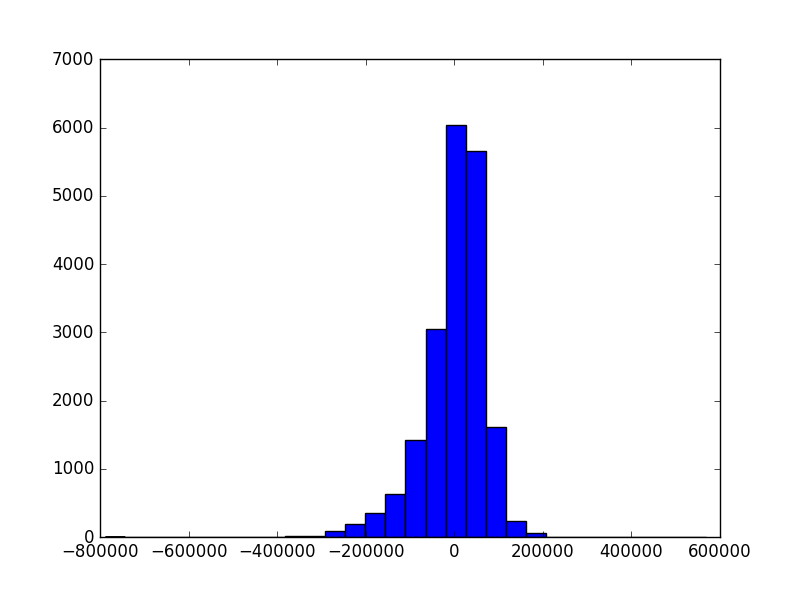
6.3)



Index

Coefficient value

6.4) Residual Histogram



The distribution resembles a Gaussian.

**Code for problem 2 (MATLAB)**

% Part (a)

mu = [1 1];

Sigma = [2 0;0 1];

x1 = -3:.2:5; x2 = -2:.2:4;

[X1,X2] = meshgrid(x1,x2);

F = mvnpdf([X1(:) X2(:)],mu,Sigma);

F = reshape(F,length(x2),length(x1));

figure()

contour(X1,X2,F)

set(gca,'FontSize', 14)

title('part a')

xlabel('X1')

ylabel('X2')

% Part (b)

mu = [-1 2];

Sigma = [3 1;1 2];

x1 = -5:.2:3; x2 = -2:.2:5;

[X1,X2] = meshgrid(x1,x2);

F = mvnpdf([X1(:) X2(:)],mu,Sigma);

F = reshape(F,length(x2),length(x1));

figure()

contour(X1,X2,F)

set(gca,'FontSize', 14)

title('part b')

xlabel('X1')

ylabel('X2')

% Part (c)

mu1 = [0 2];

Sigma1 = [1 1;1 2];

mu2 = [2 0];

Sigma2 = [1 1;1 2];

x1 = -6:.2:6; x2 = -6:.2:6;

[X1,X2] = meshgrid(x1,x2);

F1 = mvnpdf([X1(:) X2(:)],mu1,Sigma1);

F2 = mvnpdf([X1(:) X2(:)],mu2,Sigma2);

F3 = F1 - F2;

F3 = reshape(F3,length(x2),length(x1));

figure()

contour(X1,X2,F3,10)

set(gca,'FontSize', 14)

title('part c')

xlabel('X1')

ylabel('X2')

% Part (d)

mu1 = [0 2];

Sigma1 = [1 1;1 2];

mu2 = [2 0];

Sigma2 = [3 1;1 2];

x1 = -6:.2:6; x2 = -6:.2:6;

[X1,X2] = meshgrid(x1,x2);

F1 = mvnpdf([X1(:) X2(:)],mu1,Sigma1);

F2 = mvnpdf([X1(:) X2(:)],mu2,Sigma2);

F3 = F1 - F2;

F3 = reshape(F3,length(x2),length(x1));

figure()

contour(X1,X2,F3,10)

set(gca,'FontSize', 14)

title('part d')

xlabel('X1')

ylabel('X2')

% Part (e)

mu1 = [1 1];

Sigma1 = [1 0;0 2];

mu2 = [-1 -1];

Sigma2 = [2 1;1 2];

x1 = -7:.2:7; x2 = -7:.2:7;

[X1,X2] = meshgrid(x1,x2);

F1 = mvnpdf([X1(:) X2(:)],mu1,Sigma1);

F2 = mvnpdf([X1(:) X2(:)],mu2,Sigma2);

F3 = F1 - F2;

F3 = reshape(F3,length(x2),length(x1));

figure()

contour(X1,X2,F3,10)

set(gca,'FontSize', 14)

title('part e')

xlabel('X1')

ylabel('X2')

**Code for problems 3, 5, 6 (Python)**

import matplotlib.mlab as mlab

import numpy as np

import matplotlib.pyplot as plt

import math, random

import scipy.io

# Problem 3

'''

X1 = np.random.normal(3,3,100)

X2 = (X1/2) + np.random.normal(4,2,100)

# Part (a)

mu = [np.sum(X1)/100, np.sum(X2)/100]

print('mean:\n', mu)

# Part (b)

sigma = np.cov(np.vstack((X1,X2)))

print('covariance matrix:\n',sigma)

# Part (c)

eigen = np.linalg.eig(sigma)

eigvalues = eigen[0]

eigvectors = eigen[1].T

print('eigenvalues:\n',eigvalues)

print('eigenvectors:\n',eigvectors)

# Part (d)

plt.figure()

plt.plot(X1,X2,'.')

plt.xlim(-15,15)

plt.ylim(-15,15)

U,V = zip(\*eigvectors)

U = np.multiply(U,eigvalues)

V = np.multiply(V,eigvalues)

ax = plt.gca()

ax.quiver(mu[0], mu[1], U,V,scale\_units='xy',scale=1)

plt.title('part d')

plt.xlabel("X1")

plt.ylabel("X2")

# Part (e)

if eigvalues[0] > eigvalues[1]:

U = np.array(eigvectors).T

else:

U = np.array([eigvectors[1],eigvalues[0]]).T

rotated = np.dot(U.T,(np.vstack([X1,X2]) - np.tile(mu,[100,1]).T))

plt.figure()

plt.plot(rotated[0],rotated[1],'.')

plt.xlim(-15,15)

plt.ylim(-15,15)

plt.title('part e')

plt.show()

'''

# Problem 5

#benchmark.m, converted

def benchmark(pred\_labels, true\_labels):

errors = pred\_labels != true\_labels

err\_rate = sum(errors) / float(len(true\_labels))

indices = errors.nonzero()

return err\_rate, indices

#montage\_images.m, converted

def montage\_images(images):

num\_images=min(1000,np.size(images,2))

numrows=math.floor(math.sqrt(num\_images))

numcols=math.ceil(num\_images/numrows)

img=np.zeros((numrows\*28,numcols\*28));

for k in range(num\_images):

r = k % numrows

c = k // numrows

img[r\*28:(r+1)\*28,c\*28:(c+1)\*28]=images[:,:,k];

return img

digit\_train\_data = scipy.io.loadmat("data/digit\_dataset/train.mat")

digit\_train\_images= digit\_train\_data["train\_images"]

digit\_train\_labels= digit\_train\_data["train\_labels"]

train\_vectors=[]

for i in range(np.shape(digit\_train\_images)[2]):

vector = np.float64(digit\_train\_images[:,:,i].flatten())

train\_vectors.append( np.divide(vector,np.linalg.norm(vector, 2)))

normalized\_vectors= np.array(train\_vectors)

indices = list(range(np.shape(normalized\_vectors)[0]))

random.shuffle(indices)

train\_vectors = []

train\_labels = []

val\_vectors = []

val\_labels = []

for i in indices[:50000]:

train\_vectors.append(normalized\_vectors[i])

train\_labels.append(digit\_train\_labels[i])

for i in indices[50000:]:

val\_vectors.append(normalized\_vectors[i])

val\_labels.append(digit\_train\_labels[i])

val\_labels = np.array(val\_labels).ravel()

train\_labels = np.array(train\_labels).ravel()

train\_vectors = np.array(train\_vectors)

val\_vectors = np.array(val\_vectors)

# part (a)

'''

mean\_list = []

cov\_list = []

for i in range(10):

indices = np.where(digit\_train\_labels==i)[0]

subset = normalized\_vectors[indices]

mean\_list.append(np.sum(subset,axis=0)/subset.shape[0])

cov\_list.append(np.cov(subset.T))

# mean\_list and cov\_list are the mean and covariance matrices for part (a)

# mean\_list[i] and cov\_list[i] correspond to digit i

'''

# part (b)

'''

priors = []

for i in range(10):

indices = np.where(digit\_train\_labels==i)[0]

priors.append(indices.shape[0]/digit\_train\_labels.shape[0])

print('Priors:\n', priors)

'''

# part (c)

'''

plt.imshow(cov\_list[1])

plt.colorbar()

plt.show()

'''

# part (di)

'''

training\_size = [100,200,500,1000,2000,5000,10000,30000,50000]

errors = []

alpha = np.eye(784)\*0.0001

for N in training\_size:

subset\_vectors = train\_vectors[:N]

subset\_labels = train\_labels[:N]

mean\_list = []

cov\_list = []

for i in range(10):

indices = np.where(subset\_labels==i)[0]

subset = subset\_vectors[indices]

mean\_list.append(np.sum(subset,axis=0)/subset.shape[0])

cov\_list.append(np.cov(subset.T))

priors = []

for i in range(10):

indices = np.where(subset\_labels==i)[0]

priors.append(indices.shape[0]/subset\_labels.shape[0])

sigma\_overall = np.sum(cov\_list,axis=0)/len(cov\_list) + alpha

sig\_inv = np.linalg.inv(sigma\_overall)

predictions= []

for sample in val\_vectors:

guess = 0

best = -float('inf')

for i in range(10):

f = (np.dot(np.dot(mean\_list[i].T,sig\_inv),sample) - np.dot(np.dot(mean\_list[i].T,sig\_inv),mean\_list[i])/2 + math.log(priors[i]))

if f > best:

guess = i

best = f

predictions.append(guess)

errors.append(benchmark(np.array(predictions),val\_labels)[0])

plt.figure()

plt.plot(training\_size,errors,'-')

plt.show()

'''

# part (dii)

'''

training\_size = [100,200,500,1000,2000,5000,10000,30000,50000]

errors = []

alpha = np.eye(784)\*0.00001

for N in training\_size:

randomidx = random.randint(0,50000-N)

subset\_vectors = train\_vectors[randomidx:randomidx+N]

subset\_labels = train\_labels[randomidx:randomidx+N]

mean\_list = []

cov\_list = []

for i in range(10):

indices = np.where(subset\_labels==i)[0]

subset = subset\_vectors[indices]

mean\_list.append(np.sum(subset,axis=0)/subset.shape[0])

cov\_list.append(np.cov(subset.T) + alpha)

priors = []

for i in range(10):

indices = np.where(subset\_labels==i)[0]

priors.append(indices.shape[0]/subset\_labels.shape[0])

sig\_inv = [np.linalg.inv(sigma) for sigma in cov\_list]

logdet\_list = [np.linalg.slogdet(sigma)[1] for sigma in cov\_list]

predictions= []

for sample in val\_vectors:

guess = 0

best = -float('inf')

for i in range(10):

f = np.dot(np.dot((sample - mean\_list[i]).T,sig\_inv[i]),(sample - mean\_list[i]))/-2 + math.log(priors[i]) - logdet\_list[i]/2

if f > best:

guess = i

best = f

predictions.append(guess)

errors.append(benchmark(np.array(predictions),val\_labels)[0])

plt.figure()

plt.plot(training\_size,errors,'-')

plt.show()

'''

# part (div)

'''

# LDA for part (div)

digit\_test\_data = scipy.io.loadmat("data/digit\_dataset/test.mat")

digit\_test\_images= digit\_test\_data["test\_images"].transpose()

test\_vectors=[]

for i in range(np.shape(digit\_test\_images)[1]):

vector = np.float64(digit\_test\_images[:,i])

vector = vector.reshape((28,28)).T

vector = vector.flatten()

test\_vectors.append(np.divide(vector,np.linalg.norm(vector, 2)))

test\_vectors= np.array(test\_vectors)

alpha = np.eye(784)\*0.0001

subset\_vectors = train\_vectors

subset\_labels = train\_labels

mean\_list = []

cov\_list = []

for i in range(10):

indices = np.where(subset\_labels==i)[0]

subset = subset\_vectors[indices]

mean\_list.append(np.sum(subset,axis=0)/subset.shape[0])

cov\_list.append(np.cov(subset.T))

priors = []

for i in range(10):

indices = np.where(subset\_labels==i)[0]

priors.append(indices.shape[0]/subset\_labels.shape[0])

sigma\_overall = np.sum(cov\_list,axis=0)/len(cov\_list) + alpha

sig\_inv = np.linalg.inv(sigma\_overall)

predictions= []

for sample in val\_vectors:

guess = 0

best = -float('inf')

for i in range(10):

f = (np.dot(np.dot(mean\_list[i].T,sig\_inv),sample) - np.dot(np.dot(mean\_list[i].T,sig\_inv),mean\_list[i])/2 + math.log(priors[i]))

if f > best:

guess = i

best = f

predictions.append(guess)

error = benchmark(np.array(predictions),val\_labels)[0]

test\_predict = []

for sample in test\_vectors:

guess = 0

best = -float('inf')

for i in range(10):

f = (np.dot(np.dot(mean\_list[i].T,sig\_inv),sample) - np.dot(np.dot(mean\_list[i].T,sig\_inv),mean\_list[i])/2 + math.log(priors[i]))

if f > best:

guess = i

best = f

test\_predict.append(guess)

numbers = (np.arange(10000) + 1)

print(test\_predict[:25])

test\_predict = np.vstack((numbers,test\_predict))

print(error)

np.savetxt("digitsLDA.csv", test\_predict.transpose(), delimiter=",",fmt = '%u')

img = montage\_images(test\_vectors.T.reshape((28,28,test\_vectors.shape[0]))[:,:,:25])

plt.imshow(img)

plt.show()

'''

# QDA for part (div)

'''

digit\_test\_data = scipy.io.loadmat("data/digit\_dataset/test.mat")

digit\_test\_images= digit\_test\_data["test\_images"].transpose()

test\_vectors=[]

for i in range(np.shape(digit\_test\_images)[1]):

vector = np.float64(digit\_test\_images[:,i])

vector = vector.reshape((28,28)).T

vector = vector.flatten()

test\_vectors.append(np.divide(vector,np.linalg.norm(vector, 2)))

test\_vectors= np.array(test\_vectors)

alpha = np.eye(784)\*0.00001

mean\_list = []

cov\_list = []

for i in range(10):

indices = np.where(train\_labels==i)[0]

subset = train\_vectors[indices]

mean\_list.append(np.sum(subset,axis=0)/subset.shape[0])

cov\_list.append(np.cov(subset.T) + alpha)

priors = []

for i in range(10):

indices = np.where(train\_labels==i)[0]

priors.append(indices.shape[0]/train\_labels.shape[0])

sig\_inv = [np.linalg.inv(sigma) for sigma in cov\_list]

logdet\_list = [np.linalg.slogdet(sigma)[1] for sigma in cov\_list]

validation\_predict = []

for sample in val\_vectors:

guess = 0

best = -float('inf')

for i in range(10):

f = np.dot(np.dot((sample - mean\_list[i]).T,sig\_inv[i]),(sample - mean\_list[i]))/-2 + math.log(priors[i]) - logdet\_list[i]/2

if f > best:

guess = i

best = f

validation\_predict.append(guess)

error = benchmark(np.array(validation\_predict),val\_labels)[0]

test\_predict = []

for sample in test\_vectors:

guess = 0

best = -float('inf')

for i in range(10):

f = np.dot(np.dot((sample - mean\_list[i]).T,sig\_inv[i]),(sample - mean\_list[i]))/-2 + math.log(priors[i]) - logdet\_list[i]/2

if f > best:

guess = i

best = f

test\_predict.append(guess)

numbers = (np.arange(10000) + 1)

print(test\_predict[:25])

test\_predict = np.vstack((numbers,test\_predict))

print(benchmark(validation\_predict, val\_labels)[0])

np.savetxt("digitsQDA.csv", test\_predict.transpose(), delimiter=",",fmt = '%u')

img = montage\_images(test\_vectors.T.reshape((28,28,test\_vectors.shape[0]))[:,:,:25])

plt.imshow(img)

plt.show()

'''

# part (e)

'''

spam\_data = scipy.io.loadmat("data/spam\_dataset/spam\_data.mat")

spam\_train\_data= spam\_data["training\_data"]

spam\_train\_labels= np.ravel(spam\_data["training\_labels"])

spam\_test = spam\_data["test\_data"]

normalize = []

for vector in spam\_test:

if not np.linalg.norm(vector, 2):

normalize.append(vector)

else:

normalize.append(np.divide(vector,np.linalg.norm(vector, 2)))

spam\_test = np.array(normalize)

normalize = []

for vector in spam\_train\_data:

if not np.linalg.norm(vector, 2):

normalize.append(vector)

else:

normalize.append(np.divide(vector,np.linalg.norm(vector, 2)))

spam\_train\_data = np.array(normalize)

alpha = np.eye(32)\*0.0001

indices = list(range(np.shape(spam\_train\_data)[0]))

random.shuffle(indices)

subset\_vectors = []

subset\_labels = []

val\_vectors = []

val\_labels = []

for i in indices[:4000]:

subset\_vectors.append(spam\_train\_data[i])

subset\_labels.append(spam\_train\_labels[i])

for i in indices[4000:]:

val\_vectors.append(spam\_train\_data[i])

val\_labels.append(spam\_train\_labels[i])

val\_labels = np.array(val\_labels).ravel()

subset\_labels = np.array(subset\_labels).ravel()

subset\_vectors = np.array(subset\_vectors)

val\_vectors = np.array(val\_vectors)

mean\_list = []

cov\_list = []

priors = []

for i in range(2):

indices = np.where(subset\_labels==i)[0]

subset = subset\_vectors[indices]

mean\_list.append(np.sum(subset,axis=0)/subset.shape[0])

cov\_list.append(np.cov(subset.T) + alpha)

priors.append(indices.shape[0]/subset\_labels.shape[0])

sig\_inv = [np.linalg.inv(sigma) for sigma in cov\_list]

logdet\_list = [np.linalg.slogdet(sigma)[1] for sigma in cov\_list]

predictions= []

for sample in val\_vectors:

guess = 0

best = -float('inf')

for i in range(2):

f = np.dot(np.dot((sample - mean\_list[i]).T,sig\_inv[i]),(sample - mean\_list[i]))/-2 + math.log(priors[i]) - logdet\_list[i]/2

if f > best:

guess = i

best = f

predictions.append(guess)

error = benchmark(np.array(predictions),val\_labels)[0]

test\_predict = []

for sample in spam\_test:

guess = 0

best = -float('inf')

for i in range(2):

f = np.dot(np.dot((sample - mean\_list[i]).T,sig\_inv[i]),(sample - mean\_list[i]))/-2 + math.log(priors[i]) - logdet\_list[i]/2

if f > best:

guess = i

best = f

test\_predict.append(guess)

numbers = (np.arange(5857) + 1)

print('val error: ',error)

test\_predict = np.vstack((numbers,test\_predict))

np.savetxt("spam.csv", test\_predict.transpose(), delimiter=",",fmt = '%u')

'''

# problem 6

'''

housing\_data = scipy.io.loadmat("data/housing\_dataset/housing\_data.mat")

X\_train= housing\_data['Xtrain']

X = np.hstack((X\_train,np.ones((X\_train.shape[0],1))))

Y\_train= housing\_data['Ytrain']

y = np.float\_(Y\_train)[:,0]

w = np.dot(np.linalg.inv(np.dot(X.T,X)),np.dot(X.T,y))

X\_validate = housing\_data['Xvalidate']

X\_validate = np.hstack((X\_validate,np.ones((X\_validate.shape[0],1))))

Y\_validate= housing\_data['Yvalidate'][:,0]

validation\_predict = np.dot(X\_validate,w)

RSS = np.dot((validation\_predict - Y\_validate).T,(validation\_predict - Y\_validate))

print('RSS: ',RSS)

print('predicted min: ',min(validation\_predict))

print('predicted max: ',max(validation\_predict))

plt.figure()

plt.plot(np.arange(8),w[:8],'o')

plt.xlim((-1,9))

plt.figure()

plt.hist(np.dot(X,w) - y,30)

plt.show()

'''