CS 189: Homework 3

Michael Stephen Chen
Kaggle Acct: michaelstchen
SID: 23567341

March 3, 2016

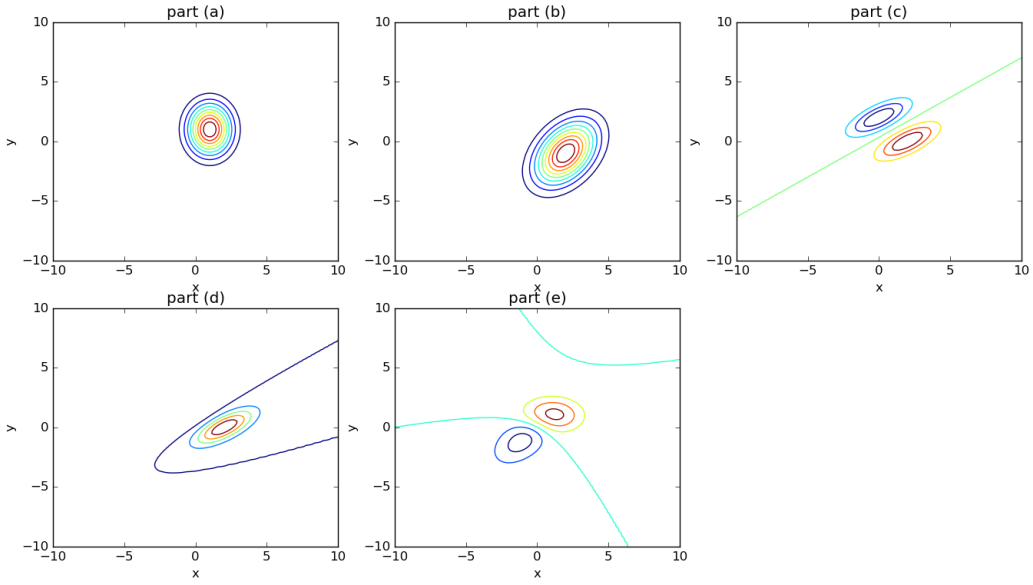
Problem 1

a)

b)

Problem 2

Below are my isocontour plots for part a-e



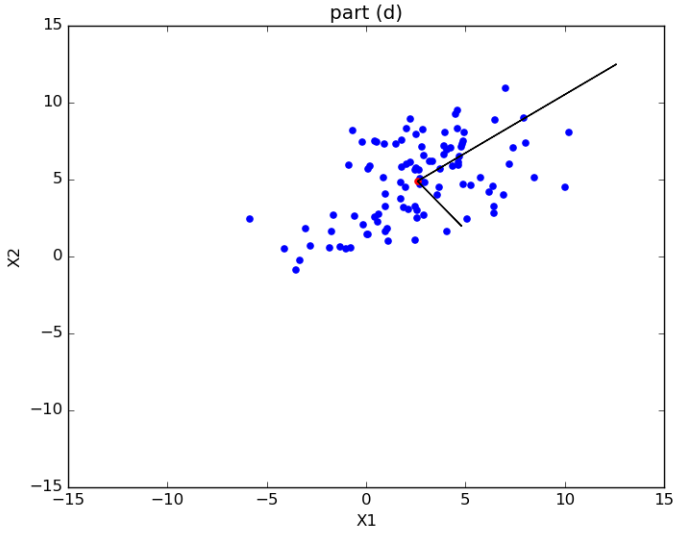
Problem 3

a) For one run I found the mean to be $mean^T = [2.926, 5.776]$, which is close to what we expect of $\mu^T = [3, 5.5]$ if we took an infinite number of samples. Please see the variable *mu* in the file *prob3.py*

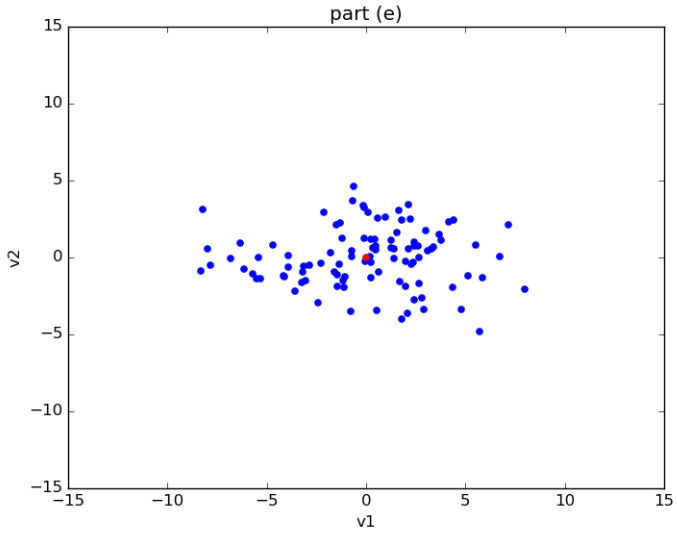
b) Please see the variable *covmat* in the file *prob3.py* for my covariance matrix.

c) Please see the variables *eigvals* and *eigvecs* in the file *prob3.py* for my eigenvalues and eigenvectors, respectively.

d) Below is my plot of the X_1 and X_2 samples in blue, the mean in red, and the eigenvectors of the covariance matrix as arrows



e) Below is my rotated plot of the X_1 and X_2 samples so that the eigenvectors of the covariance matrix correspond to the axes.



Problem 4

a)

b)

c)

d)

Problem 5

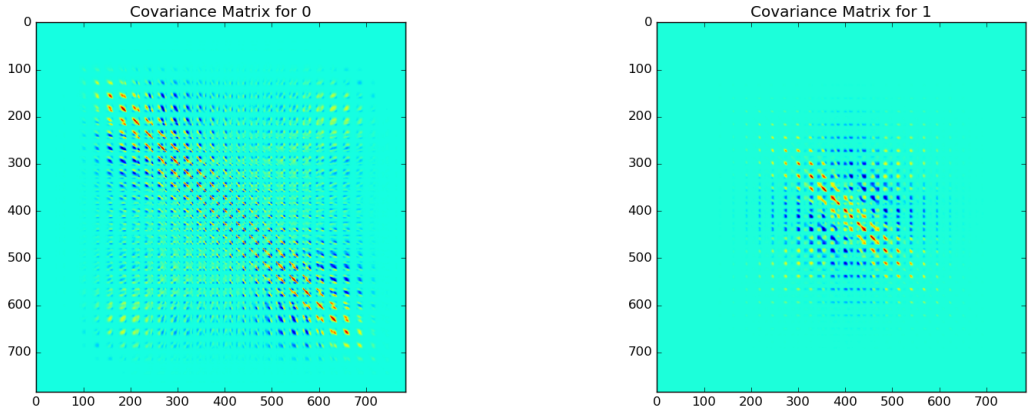
a) The MLE for the mean ($\hat{\mu}$) is simply the sample mean for our samples x_i

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

The MLE for the covariance matrices is simply a square matrix with the variances of the features along the diagonal and zeros on the off-diagonals. This is because we assume the features to be i.i.d, meaning that they are independent of one another. As a result the $Cov(X_i, X_j) = 0$ if $i \neq j$. These make up the off-diagonals. When $i = j$, $Cov(X_i, X_j) = Var(X_i)$, which will be the diagonals for our covariance matrix. See *prob5abc.py*.

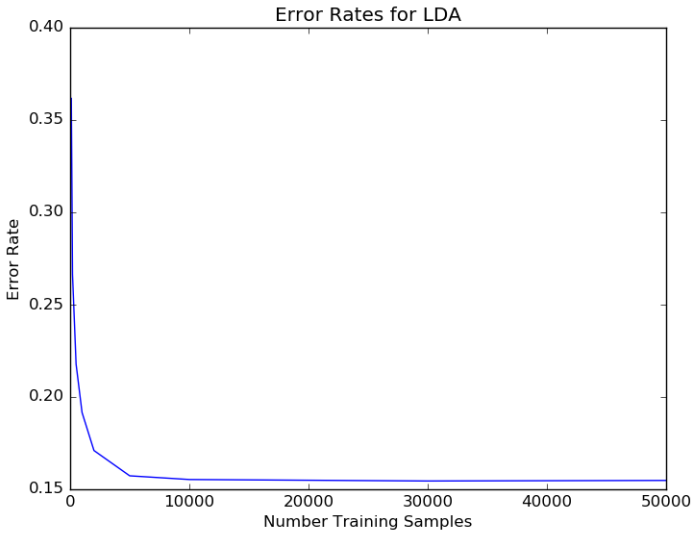
b) We can model the prior distribution for each class by tabulating the counts of each digit sample in our training set data and dividing by the total number of samples. So we essentially compute the percent composition of each class in the training set and use that as an estimate for our class priors. See *prob5abc.py*.

c) Below are the visualizations for the "0" and "1" covariance matrices



The axes are the features. The light blue regions represent relatively independent features while the redder regions represent regions that are highly dependent.

d) Below is a plot of error rate vs sample size for our LDA digits classifier



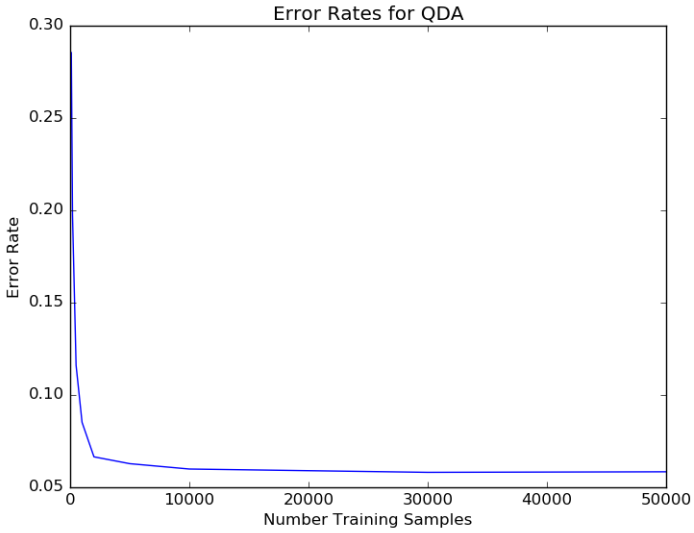
The error rates are as follows:

$SampleSize$	$ErrorRate$
100	0.3617
200	0.2664
500	0.2178
1000	0.1916
2000	0.1710
5000	0.1573
10000	0.1553
30000	0.1545
50000	0.1548

When we use just one overall covariance matrix, the discriminant function (our decision boundaries) turn out to be linear:

$$\mu_c^T \Sigma^{-1} x - \frac{1}{2} \mu_c^T \Sigma^{-1} \mu_c + ln(\pi_c)$$

e) Below is a plot of error rate v sample size for our QDA digits classifier



The error rates are as follows:

$SampleSize$	$ErrorRate$
100	0.2854
200	0.1976
500	0.1161
1000	0.0852
2000	0.0665
5000	0.0628
10000	0.0599
30000	0.0581
50000	0.0584

When we use a covariance matrix for each class, the disscriminant function (our decision boundaries) turn out to be quadratic:

$$-\frac{1}{2}(x-\mu)^T\Sigma_c^{-1}(x-\mu)-\frac{1}{2}ln|\Sigma_c|+ln(\pi_c)$$

f) The LDA classifier's performances were lower than the QDA's probably because the LDA is underfitting the data because it only uses an average, overall covariance matrix.

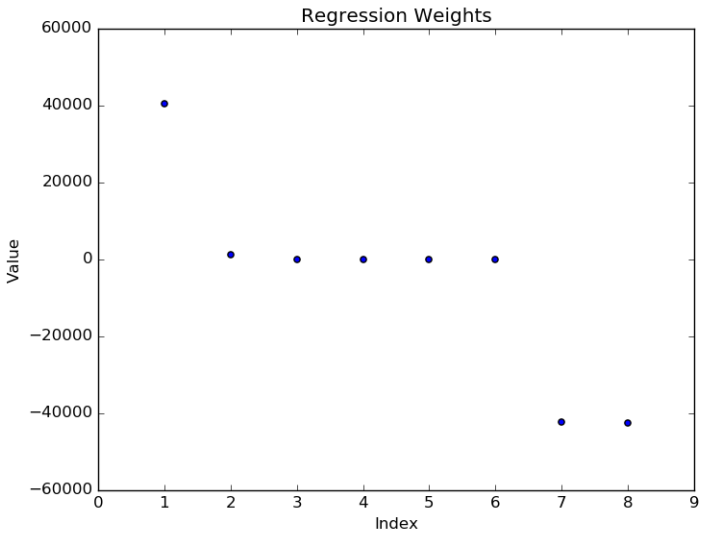
g) When I train all all 60000 training samples and classify the 10000 test samples using the QDA procedure, I get a accuracy of 0.93620. I did not use any additional features.

h) Using the QDA method, this time with the multivariate_normal.logpdf() function to determine the discriminant, I got an accuracy of 0.79834. For some reason I was trending around 0.59 when I sued the same QDA implementation from the previous part but I get a much higher score with this. I was reading that multivariate_normal.logpdf uses the pseudoinverse so maybe that's why. I added some additional feature words in *featurize.py* (e.g. "penis").

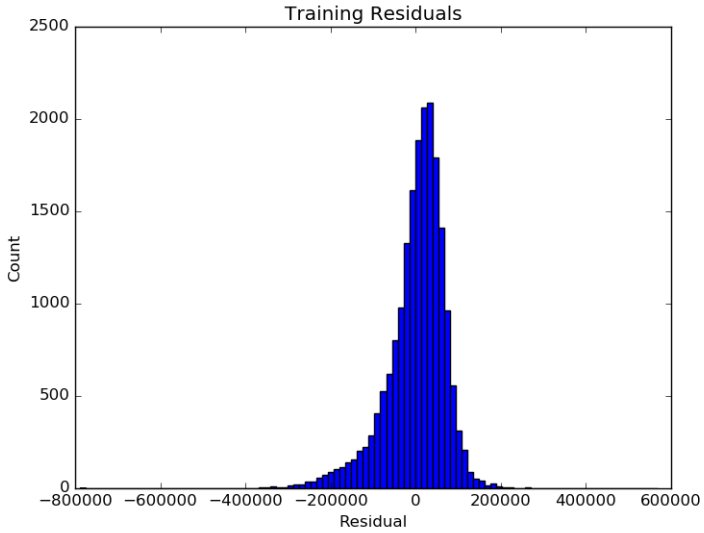
Problem 6

1. Please see *prob6.py*
2. My residual sum of squares for my 1200 training sample data was 5794953797667.3555. The predicted range of home values is [-56562.83, 710798.84]. The minimum definitely doesn't make sense given that you can't have a negative home value (debt?). The max seems reasonable, especially here in the bay area, but is significantly higher that the actual max of the validation set (5000001)

3. Below is a plot of my regression weights for all terms except the constant term



4. Below is a histogram of my training data residuals. It resembles a normal distribution, albeit a little skewed.



Appendix: prob2.py

```
import numpy as np
import matplotlib.pyplot as plt
from math import sqrt, pi

def multgaussprob(x, y, mu, covmat):
    prob_a = np.zeros((len(x), len(y)))
    norm = 1 / (sqrt(2*pi)*np.linalg.det(covmat))

    for ix in range(0, len(x)):
        for iy in range(0, len(y)):
            x_right = np.array([[x[ix]],[y[iy]]]) - mu
            x_left = np.transpose(x_right)
            x_exp = np.dot(x_left, np.dot(np.linalg.inv(covmat),
                                          x_right))
            prob_a[ix, iy] = norm * np.exp(x_exp/-2)

    return prob_a


x = np.linspace(-10, 10, 100)
y = np.linspace(-10, 10, 100)

#part (a)
mu_a = np.array([[1], [1]])
covmat_a = np.array([[2, 0], [0, 1]])
prob_a = multgaussprob(x, y, mu_a, covmat_a)
ax_a = plt.subplot(2, 3, 1)
ax_a.set_title("part (a)")
ax_a.set_xlabel("x")
ax_a.set_ylabel("y")
ax_a.contour(x, y, prob_a, 10)

#part (b)
mu_b = np.array([[-1], [2]])
covmat_b = np.array([[3, 1], [1, 2]])
prob_b = multgaussprob(x, y, mu_b, covmat_b)
ax_b = plt.subplot(2, 3, 2)
ax_b.set_title("part (b)")
ax_b.set_xlabel("x")
ax_b.set_ylabel("y")
ax_b.contour(x, y, prob_b, 10)

#part (c)
mu_c1 = np.array([[0], [2]])
mu_c2 = np.array([[2], [0]])
covmat_c = np.array([[1, 1], [1, 2]])
prob_c1 = multgaussprob(x, y, mu_c1, covmat_c)
prob_c2 = multgaussprob(x, y, mu_c2, covmat_c)
prob_c = prob_c1 - prob_c2
ax_c = plt.subplot(2, 3, 3)
ax_c.set_title("part (c)")
ax_c.set_xlabel("x")
```

```
ax_c.set_ylabel("y")
ax_c.contour(x, y, prob_c)

#part (d)
mu_d1 = np.array([[0], [2]])
mu_d2 = np.array([[2], [0]])
covmat_d1 = np.array([[1, 1], [1, 2]])
covmat_d2 = np.array([[3, 1], [1, 2]])
prob_d1 = multgaussprob(x, y, mu_d1, covmat_d1)
prob_d2 = multgaussprob(x, y, mu_d2, covmat_d2)
prob_d = prob_d1 - prob_d2
ax_d = plt.subplot(2, 3, 4)
ax_d.set_title("part (d)")
ax_d.set_xlabel("x")
ax_d.set_ylabel("y")
ax_d.contour(x, y, prob_d)

#part (e)
mu_e1 = np.array([[1], [1]])
mu_e2 = np.array([[-1], [-1]])
covmat_e1 = np.array([[1, 0], [0, 2]])
covmat_e2 = np.array([[2, 1], [1, 2]])
prob_e1 = multgaussprob(x, y, mu_e1, covmat_e1)
prob_e2 = multgaussprob(x, y, mu_e2, covmat_e2)
prob_e = prob_e1 - prob_e2
ax_e = plt.subplot(2, 3, 5)
ax_e.set_title("part (e)")
ax_e.set_xlabel("x")
ax_e.set_ylabel("y")
ax_e.contour(x, y, prob_e)

plt.show()
```

Appendix: prob3.py

```
import numpy as np
import matplotlib.pyplot as plt
import math
from math import sqrt, pi

N = 100
samples = np.zeros((N, 2))
for i in range(0, N):
    x1 = np.random.normal(3, 3)
    x2 = x1/2.0 + np.random.normal(4, 2)
    samples[i, :] = np.array([x1, x2])

#part(a)
mu = np.mean(samples, axis=0)

#part(b)
covmat = np.cov(np.transpose(samples))

#part(c)
```

```
eigvals, eigvecs = np.linalg.eig(covmat)

#part(d)
plt.figure()
plt.scatter(samples[:,0], samples[:,1], color='blue')
plt.scatter(mu[0], mu[1], color='red')
ang0 = math.atan(eigvecs[1,0]/eigvecs[0,0])
ang1 = math.atan(eigvecs[1,1]/eigvecs[0,1])
plt.arrow(mu[0], mu[1],
          eigvals[0]*math.cos(ang0),eigvals[0]*math.sin(ang0))
plt.arrow(mu[0], mu[1],
          eigvals[1]*math.cos(ang1),eigvals[1]*math.sin(ang1))
plt.title('part (d)')
plt.xlabel('X1')
plt.ylabel('X2')
plt.xlim([-15, 15])
plt.ylim([-15, 15])


#part(e)
trans_samples = np.zeros((N, 2))
for i in range(0, N):
    xcent = np.transpose(samples[i, :]-mu)
    xrot = np.dot(np.transpose(eigvecs), np.transpose(xcent))
    trans_samples[i, :] = np.transpose(xrot)

plt.figure()
plt.scatter(trans_samples[:,0], trans_samples[:,1], color='blue')
plt.scatter(0, 0, color='red')
plt.title('part (e)')
plt.xlabel('v1')
plt.ylabel('v2')
plt.xlim([-15, 15])
plt.ylim([-15, 15])

plt.show()
```

Appendix: prob5di.py

```
import numpy as np
import math
import csv
import matplotlib.pyplot as plt
from scipy.io import loadmat
from sklearn.preprocessing import normalize

#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
```

```
train_set = loadmat(file_name="train.mat", mat_dtype=True)
test_set = loadmat(file_name="test.mat")

train_data = np.transpose(train_set["train_images"])
train_labels = train_set["train_labels"].reshape(1,-1)[0]
test_data = test_set["test_images"]

random_state = np.random.get_state()
np.random.shuffle(train_data)
np.random.set_state(random_state)
np.random.shuffle(train_labels)

num_train = train_data.shape[0]
num_test = test_data.shape[0]

for i in range(0, num_train):
    train_data[i] = train_data[i] / np.linalg.norm(train_data[i])

set_size = [100, 200, 500, 1000, 2000, 5000, 10000, 30000, 50000]
err_list = [0, 0, 0, 0, 0, 0, 0, 0, 0]
for si in range(0, len(set_size)):
    sz = set_size[si]
    train_slice = train_data[0:sz]
    label_slice = train_labels[0:sz]

    index = label_slice.argsort()
    t_sort = train_slice[index[::1]]
    l_sort = label_slice[index[::1]]
    t_flat = t_sort.reshape(len(t_sort), -1)
    t_flat = np.transpose(t_flat)
    for i in range(0, sz):
        t_flat[:, i] = t_flat[:, i] / np.linalg.norm(t_flat[:, i])

    priors = np.zeros(10)
    for i in l_sort:
        priors[i] = priors[i] + 1

    digit_sect = np.zeros(11)
    for i in range(0, 10):
        digit_sect[i+1] = digit_sect[i]+priors[i]

    priors = priors / len(l_sort)

    covmat = np.zeros((10,784,784))
    mu = np.zeros((10, 784))
    for i in range(0,10):
        ai = digit_sect[i]
        ae = digit_sect[i+1]
        covmat[i,:,:] = np.cov(t_flat[:, ai:ae])
        mu[i, :] = np.mean(t_flat[:, ai:ae], axis=1)

    n = 10000
    overall = np.mean(covmat, axis=0)
    train_valid = train_data[len(train_data)-n:len(train_data)]
```

```
    train_valid = train_valid.reshape(len(train_valid), -1)
    train_valid = np.transpose(train_valid)
    labels_valid = train_labels[len(train_data)-n:len(train_data)]
    preds = np.zeros(len(labels_valid))

    a = 0.001
    over_inv = np.linalg.inv(overall + a*np.identity(784))
    firsts = np.zeros((10, 784))
    seconds = np.zeros(10)
    thirds = np.zeros(10)
    for c in range(0,10):
        mu_t = np.transpose(mu[c])
        firsts[c] = np.dot(mu_t, over_inv)
        seconds[c] = 0.5 * np.dot(mu_t, np.dot(over_inv,mu[c]))
        thirds[c] = np.log(priors[c])

    for i in range(0, len(preds)):
        x = train_valid[:, i]
        max_c = 0
        max_disc = 0
        for c in range(0, 10):
            mu_t = np.transpose(mu[c])
            first = np.dot(firsts[c],x)
            curr = first - seconds[c] + thirds[c]
            if curr > max_disc:
                max_c = c
                max_disc = curr
        preds[i] = max_c
    err_list[si], ind = benchmark(preds, labels_valid)

plt.plot(set_size, err_list)
plt.title('Error Rates for LDA')
plt.xlabel('Number Training Samples')
plt.ylabel('Error Rate')
plt.show()
```

Appendix: prob5dii.py

```
import numpy as np
import math
import csv
import matplotlib.pyplot as plt
from scipy.io import loadmat
from sklearn.preprocessing import normalize

#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

print("Initializations")
train_set = loadmat(file_name="train.mat", mat_dtype=True)
```

```
test_set = loadmat(file_name="test.mat")

train_data = np.transpose(train_set["train_images"])
train_labels = train_set["train_labels"].reshape(1,-1)[0]
test_data = test_set["test_images"]

random_state = np.random.get_state()
np.random.shuffle(train_data)
np.random.set_state(random_state)
np.random.shuffle(train_labels)

num_train = train_data.shape[0]
num_test = test_data.shape[0]

for i in range(0, num_train):
    train_data[i] = train_data[i] / np.linalg.norm(train_data[i])

set_size = [100, 200, 500, 1000, 2000, 5000, 10000, 30000, 50000]
err_list = [0, 0, 0, 0, 0, 0, 0, 0, 0]
for si in range(0, len(set_size)):
    print("Slicing, Sorting, Transposing")
    sz = set_size[si]
    train_slice = train_data[0:sz]
    label_slice = train_labels[0:sz]

    index = label_slice.argsort()
    t_sort = train_slice[index[::1]]
    l_sort = label_slice[index[::1]]
    t_flat = t_sort.reshape(len(t_sort), -1)
    t_flat = np.transpose(t_flat)
    for i in range(0, sz):
        t_flat[:, i] = t_flat[:, i] / np.linalg.norm(t_flat[:, i])

    priors = np.zeros(10)
    for i in l_sort:
        priors[i] = priors[i] + 1

    digit_sect = np.zeros(11)
    for i in range(0, 10):
        digit_sect[i+1] = digit_sect[i]+priors[i]

    priors = priors / len(l_sort)
    print("Covariance Matrix")
    covmat = np.zeros((10,784,784))
    mu = np.zeros((10, 784))
    for i in range(0,10):
        ai = digit_sect[i]
        ae = digit_sect[i+1]
        covmat[i,:,:] = np.cov(t_flat[:, ai:ae])
        mu[i, :] = np.mean(t_flat[:, ai:ae], axis=1)

    n = 10000
    train_valid = train_data[len(train_data)-n:len(train_data)]
    train_valid = train_valid.reshape(len(train_valid), -1)
```

```
    train_valid = np.transpose(train_valid)
    labels_valid = train_labels[len(train_data)-n:len(train_data)]
    preds = np.zeros(len(labels_valid))

    a = 0.001
    covmat_inv = np.zeros((10,784,784))
    seconds = np.zeros(10)
    thirds = np.zeros(10)
    for c in range(0,10):
        covmat_id = covmat[c]+a*np.identity(784)
        covmat_inv[c] = np.linalg.inv(covmat_id)
        sign, val = np.linalg.slogdet(covmat_id)
        seconds[c] = sign * val
        thirds[c] = np.log(priors[c])

    print("Predictions")
    for i in range(0, len(preds)):
        if i % 1000 == 0: print("i: " + str(i))
        x = train_valid[:, i]
        max_c = 0
        max_disc = 0
        for c in range(0, 10):
            x_mu = x - mu[c]
            qc = np.dot(x_mu, np.dot(covmat_inv[c], x_mu))
            curr = -0.5*qc - 0.5*seconds[c] + thirds[c]
            if curr > max_disc:
                max_c = c
                max_disc = curr
        preds[i] = max_c
    err_list[si], ind = benchmark(preds, labels_valid[0:len(preds)])


plt.plot(set_size, err_list)
plt.title('Error Rates for QDA')
plt.xlabel('Number Training Samples')
plt.ylabel('Error Rate')
plt.show()
```

Appendix: prob5div.py

```
import numpy as np
import math
import csv
import matplotlib.pyplot as plt
from scipy.io import loadmat

print("Initializations")
train_set = loadmat(file_name="train.mat", mat_dtype=True)
test_set = loadmat(file_name="test.mat", mat_dtype=True)

train_data = np.transpose(train_set["train_images"])
train_labels = train_set["train_labels"].reshape(1,-1)[0]
test_data = test_set["test_images"]
```

```
random_state = np.random.get_state()
np.random.shuffle(train_data)
np.random.set_state(random_state)
np.random.shuffle(train_labels)

num_train = train_data.shape[0]
num_test = test_data.shape[0]

for i in range(0, num_train):
    train_data[i] = train_data[i] / np.linalg.norm(train_data[i])

for i in range(0, num_test):
    test_data[i] = test_data[i] / np.linalg.norm(test_data[i])
test_data = np.transpose(test_data)

pred_labels = np.zeros(num_test)
set_size = [num_train]
for si in range(0, len(set_size)):
    sz = set_size[si]
    print("Slicing, Sorting, Reshaping")
    train_slice = train_data[0:sz]
    label_slice = train_labels[0:sz]

    index = label_slice.argsort()
    t_sort = train_slice[index[::1]]
    l_sort = label_slice[index[::1]]
    t_flat = t_sort.reshape(len(t_sort), -1)
    t_flat = np.transpose(t_flat)
    for i in range(0, sz):
        t_flat[:, i] = t_flat[:, i] / np.linalg.norm(t_flat[:, i])

    priors = np.zeros(10)
    for i in l_sort:
        priors[i] = priors[i] + 1

    digit_sect = np.zeros(11)
    for i in range(0, 10):
        digit_sect[i+1] = digit_sect[i]+priors[i]

    priors = priors / len(l_sort)

    print("Covariance Matrix")
    covmat = np.zeros((10,784,784))
    mu = np.zeros((10, 784))
    for i in range(0,10):
        ai = digit_sect[i]
        ae = digit_sect[i+1]
        covmat[i,:,:] = np.cov(t_flat[:, ai:ae])
        mu[i, :] = np.mean(t_flat[:, ai:ae], axis=1)

    a = 0.001
    covmat_inv = np.zeros((10,784,784))
    seconds = np.zeros(10)
    thirds = np.zeros(10)
```

```
    for c in range(0,10):
        covmat_id = covmat[c]+a*np.identity(784)
        covmat_inv[c] = np.linalg.inv(covmat_id)
        sign, val = np.linalg.slogdet(covmat_id)
        seconds[c] = sign * val
        thirds[c] = np.log(priors[c])

    print("Predictions")
    for i in range(0, len(pred_labels)):
        if i % 1000 == 0: print("i: " + str(i))
        x = test_data[:, i]
        max_c = 0
        max_disc = -99999999
        for c in range(0, 10):
            x_mu = x - mu[c]
            qc = np.dot(x_mu, np.dot(covmat_inv[c], x_mu))
            curr = -0.5*qc - 0.5*seconds[c] + thirds[c]
            if curr > max_disc:
                max_c = c
                max_disc = curr
        pred_labels[i] = max_c

with open('test_labels.csv', 'wb') as f:
    writer = csv.writer(f)
    writer.writerow(['Id'] + ['Category'])
    for i in range(0, len(pred_labels)):
        writer.writerow([i+1] + [int(pred_labels[i])])
```

Appendix: prob5e.py

```
import numpy as np
import math
import csv
import matplotlib.pyplot as plt
from scipy.io import loadmat
from sklearn.preprocessing import normalize
from scipy.stats import multivariate_normal

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

print("Initializations")
execfile('featurize.py')
train_data = np.array(file_dict['training_data'])
train_labels = np.array(file_dict['training_labels'])
test_data = np.array(file_dict['test_data'])

feat_num = train_data.shape[1]

random_state = np.random.get_state()
np.random.shuffle(train_data)
```

```
np.random.set_state(random_state)
np.random.shuffle(train_labels)

num_train = train_data.shape[0]
num_test = test_data.shape[0]

set_size = [num_train]
pred_labels = np.zeros(num_test)
for si in range(0, len(set_size)):
    sz = set_size[si]
    print("Slicing, Sorting, Reshaping")
    train_slice = train_data[0:sz]
    label_slice = train_labels[0:sz]

    index = label_slice.argsort()
    t_sort = train_slice[index[::1]]
    t_sort = np.transpose(t_sort)
    l_sort = label_slice[index[::1]]

    priors = np.zeros(2)
    for i in l_sort:
        priors[i] = priors[i] + 1

    digit_sect = np.zeros(3)
    for i in range(0, 2):
        digit_sect[i+1] = digit_sect[i]+priors[i]

    priors = priors / len(l_sort)

    print("Covariance Matrix")
    covmat = np.zeros((2,feat_num,feat_num))
    mu = np.zeros((2, feat_num))
    for i in range(0,2):
        ai = digit_sect[i]
        ae = digit_sect[i+1]
        covmat[i,:,:] = np.cov(t_sort[:, ai:ae])
        mu[i, :] = np.mean(t_sort[:, ai:ae], axis=1)

    a = 0
    overall = np.mean(covmat, axis=0)

    print("Predicting...")
    test_data = np.transpose(test_data)
    for i in range(0, len(pred_labels)):
        x = test_data[:, i]
        max_c = 0
        max_disc = -9999999
        for c in range(0, 2):
            curr = multivariate_normal.logpdf(x,
                                              mean=mu[c],
                                              cov=overall)
            if curr > max_disc:
                max_c = c
                max_disc = curr
```

```
        pred_labels[i] = max_c

with open('test_labels.csv', 'wb') as f:
    writer = csv.writer(f)
    writer.writerow(['Id'] + ['Category'])
    for i in range(0, len(pred_labels)):
        writer.writerow([i+1] + [int(pred_labels[i])])
```

Appendix: prob6.py

```
import numpy as np
import math
import csv
import matplotlib.pyplot as plt
from scipy.io import loadmat

housing_data = loadmat(file_name="housing_data.mat", mat_dtype=True)

Xtrain = housing_data["Xtrain"]
Ytrain = housing_data["Ytrain"]
Xvalid = housing_data["Xvalidate"]
Yvalid = housing_data["Yvalidate"]

num_train = Xtrain.shape[0]
num_valid = Xvalid.shape[0]
Xtrain = np.append(Xtrain, (np.ones(num_train)).reshape(num_train, 1), 1)
Xvalid = np.append(Xvalid, (np.ones(num_valid)).reshape(num_valid, 1), 1)

X_t = np.transpose(Xtrain)
X_pinv = np.linalg.inv(np.dot(X_t, Xtrain))
X_pinv = np.dot(X_pinv, X_t)
w = np.dot(X_pinv, Ytrain)

Ypred = np.dot(Xvalid, w)
res_train = np.dot(Xtrain, w) - Ytrain
res_valid = Yvalid - Ypred
RSS = np.sum(res_valid**2)

minYpred = np.amin(Ypred)
maxYpred = np.amax(Ypred)

plt.figure()
plt.scatter(range(1,9), w[0:8])
plt.title('Regression Weights')
plt.xlabel('Index')
plt.ylabel('Value')
plt.show()


plt.figure()
plt.hist(res_train, 100)
plt.title('Training Residuals')
plt.xlabel('Residual')
plt.ylabel('Count')
```

```
plt.show()
```