

1. (a)

Bayes optimal decision boundary:

$$P(x_0|C_1)P(C_1) = P(x_0|C_2)P(C_2)$$

$$\frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\left(\frac{x_0 - \mu_1}{\sigma^2}\right)^2\right) \cdot \frac{1}{2} = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\left(\frac{x_0 - \mu_2}{\sigma^2}\right)^2\right) \cdot \frac{1}{2}$$

$$\exp\left(-\left(\frac{x_0 - \mu_1}{\sigma^2}\right)^2\right) = \exp\left(-\left(\frac{x_0 - \mu_2}{\sigma^2}\right)^2\right)$$

$$(x_0 - \mu_1)^2 = (x_0 - \mu_2)^2$$

Since  $\mu_1 < \mu_2$ ,

$$x_0 - \mu_1 \neq x_0 - \mu_2$$

$$x_0 - \mu_1 = \mu_2 - x_0$$

$$2x_0 = \mu_2 + \mu_1$$

$$x_0 = \frac{\mu_2 + \mu_1}{2}$$

Bayes decision rule:

if  $p(x|C_1)P(C_1) > p(x|C_2)P(C_2)$ , decide  $C_1$

otherwise decide  $C_2$

where decision boundary is at  $x_0 = \frac{\mu_2 + \mu_1}{2}$

1. (b)

$$P_e = P(\text{misclassified as } C_1 | C_2) P(C_2) +$$

$$P(\text{misclassified as } C_2 | C_1) P(C_1)$$

$$= P((X < x_0) | C_2) P(C_2) +$$

$$P((X > x_0) | C_1) P(C_1)$$

$$= \frac{1}{2} \left( P\left(\frac{X - \mu_2}{\sigma} < \frac{x_0 - \mu_2}{\sigma} | C_2\right) + \right.$$

$$\left. P\left(\frac{X - \mu_1}{\sigma} < \frac{x_0 - \mu_1}{\sigma} | C_1\right) \right)$$

$$= \frac{1}{2} \left( P\left(Z < \frac{\mu_1 - \mu_2}{2\sigma}\right) + \right.$$

$$\left. P\left(Z > \frac{\mu_1 - \mu_2}{2\sigma}\right) \right)$$

$$= \frac{1}{2} (P(Z < -a) + P(Z > a))$$

$$= P(Z > a)$$

$$= \frac{1}{\sqrt{2\pi}} \int_a^{\infty} e^{-z^2/2} dz$$

where  $Z \sim N(0, 1)$

$$f(Z | 0, 1) = \frac{1}{\sqrt{2\pi}} e^{-\frac{z^2}{2}}$$

$$Z = \frac{X - \mu}{\sigma} = X$$

4.

(a) (1) If choose class  $i$ , we need:

$$R(r(x)=i|x) \leq R(r(x)=j|x)$$

$$\lambda_S (1 - P(Y=i|x)) \leq \lambda_S (1 - P(Y=j|x))$$

$$P(Y=i|x) \geq P(Y=j|x)$$

we also need

$$R(r(x)=i|x) \leq R(r(x)=c+1|x)$$

$$\lambda_S (1 - P(Y=i|x)) \leq \lambda r \cdot 1$$

$$P(Y=i|x) \geq 1 - \frac{\lambda r}{\lambda_S}$$

likewise  $P(Y=j|x) \geq 1 - \frac{\lambda r}{\lambda_S}$

(2) Otherwise

$$P(Y=i|x) < 1 - \frac{\lambda r}{\lambda_S}$$

$$P(Y=j|x) < 1 - \frac{\lambda r}{\lambda_S}$$

e.i.

$$R(r(x)=i|x) \geq R(r(x)=c+1|x)$$

$$R(r(x)=j|x) \geq R(r(x)=c+1|x)$$

Choose class  $i$ .

27. The log likelihood function is:

$$l(\mu, \Sigma; x_1, \dots, x_n) = \sum_{i=1}^n \left( -\frac{1}{2} (x_i - \mu)^T \Sigma^{-1} (x_i - \mu) - \frac{d}{2} \log 2\pi - \frac{1}{2} \log |\Sigma| \right)$$

Take the derivative with respect to  $\mu$  and  $\Sigma^{-1}$

$$\frac{\partial l}{\partial \mu} = \sum_{i=1}^n (x_i - \mu)^T \Sigma^{-1} = 0$$

$$\frac{\partial l}{\partial \Sigma^{-1}} = \frac{n}{2} \Sigma - \frac{1}{2} \sum_{i=1}^n (x_i - \mu)(x_i - \mu)^T = 0$$

Set them to zero and we get

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

$$\hat{\Sigma} = \frac{1}{n} \sum_{i=1}^n (x_i - \hat{\mu})(x_i - \hat{\mu})^T$$

$$\hat{\sigma}_j = \sqrt{\sum_{i=1}^n (x_{ij} - \mu_j)^2}$$

(b)

The log likelihood function is:

$$l(\mu, \Sigma; x_1, \dots, x_n) = \sum_{i=1}^n \left( -\frac{1}{2} (x_i - A\mu)^T \Sigma^{-1} (x_i - A\mu) - \frac{d}{2} \log 2\pi - \frac{1}{2} \log |\Sigma| \right)$$

Take the derivative with respect to  $\mu$  and set it to zero:

$$\frac{\partial l}{\partial \mu} = A^T \sum_{i=1}^n (x_i - A\mu)^T \Sigma^{-1} = 0$$

we get

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

(a)  $\hat{\Sigma}$  would be invertible in two circumstances:

①  $n < d$

② The linearly independent vector is less than  $d$ .

(b) In order to change a singular covariance matrix to invertible one by adding an invertible matrix without changing the covariance matrix too much, we can add a small identity matrix:  $xI$  and get:

$$\hat{\Sigma}' = \hat{\Sigma} + xI$$

where  $x$  is small enough, may be two orders of magnitude smaller than the smallest non-zero eigenvalue of  $\hat{\Sigma}$ .

(c) As the PDF of  $N(0, \Sigma)$  is:

$$\frac{1}{\sqrt{2\pi|\Sigma|}} \exp\left(-\frac{x^2}{2|\Sigma|}\right)$$

the eigenvector with respect to the smallest eigenvalue of  $\Sigma$  will maximize PDF  $f(x)$ ; the eigenvector with respect to largest eigenvalue of  $\Sigma$  will minimize PDF  $f(x)$ .

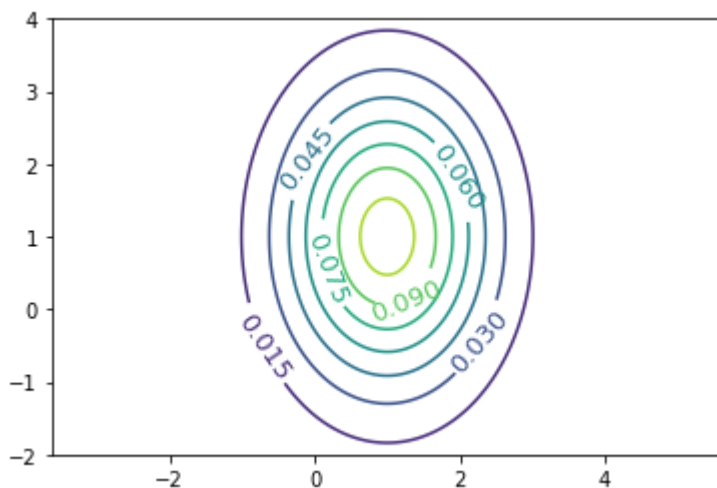
## HW3 Q2 Code

In [2]:

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

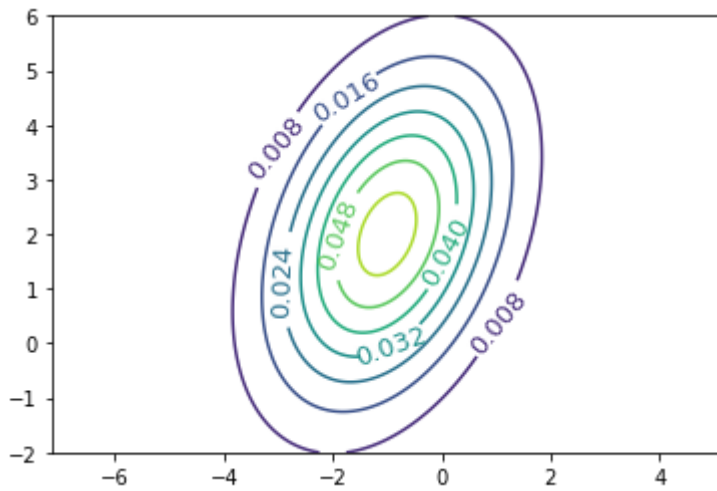
def normal_dist(mu, cov, x, y):
    Pos = np.array([x, y])
    Cov = np.linalg.inv(cov)
    dete = np.linalg.det(cov)
    Nor = 1/(2*np.pi*np.sqrt(dete)) * np.exp(-0.5*(Pos-mu).dot(Cov.dot(Pos-mu)))
    return Nor

# a
fig = plt.figure()
xi, yi = np.linspace(-2, 4, 150), np.linspace(-2, 4, 150)
xi, yi = np.meshgrid(xi, yi)
mu = np.array([1, 1])
cov = np.array([[1, 0], [0, 2]])
z = np.array([normal_dist(mu, cov, x, y) for (x,y) in zip(xi.ravel(), yi.ravel())]).reshape(xi.shape)
Con = plt.contour(z, extent=[xi.min(), xi.max(), yi.min(), yi.max()])
plt.clabel(Con, inline=1, fontsize=13)
plt.axis('equal')
plt.show()
fig.savefig('1a.png')
```



In [3]:

```
# b
fig = plt.figure()
xi, yi = np.linspace(-5, 3, 150), np.linspace(-2, 6, 150)
xi, yi = np.meshgrid(xi, yi)
mu = np.array([-1, 2])
cov = np.array([[2, 1], [1, 4]])
z = np.array([normal_dist(mu, cov, x, y) for (x,y) in zip(xi.ravel(), yi.ravel())]).reshape(xi.shape)
Con = plt.contour(z, extent=[xi.min(), xi.max(), yi.min(), yi.max()])
plt.clabel(Con, inline=1, fontsize=13)
plt.axis('equal')
plt.show()
fig.savefig('1b.png')
```

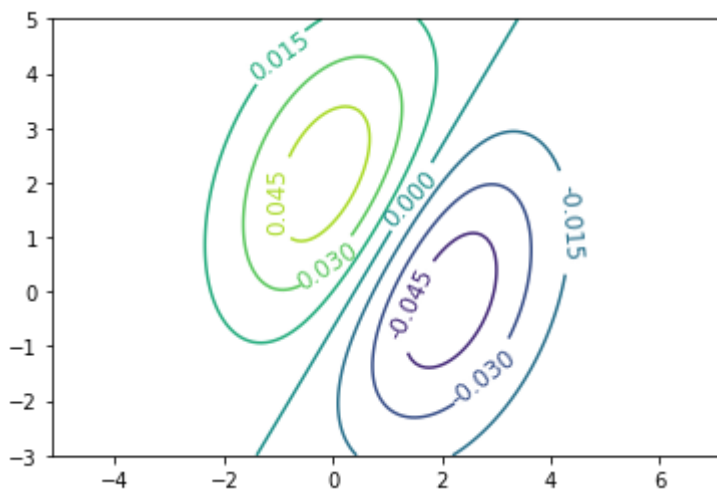


In [4]:

```

# c
fig = plt.figure()
xi, yi = np.linspace(-3, 5, 100), np.linspace(-3, 5, 100)
xi, yi = np.meshgrid(xi, yi)
mu_1 = np.array([0, 2])
cov_1 = np.array([[2, 1], [1, 1]])
z_1 = np.array([normal_dist(mu_1, cov, x, y) for (x,y) in zip(xi.ravel(), yi.ravel())])
.reshape(xi.shape)
mu_2 = np.array([2, 0])
z_2 = np.array([normal_dist(mu_2, cov, x, y) for (x,y) in zip(xi.ravel(), yi.ravel())])
.reshape(xi.shape)
Con = plt.contour(z_1-z_2, extent=[xi.min(), xi.max(), yi.min(), yi.max()])
plt.clabel(Con, inline=1, fontsize=12)
plt.axis('equal')
plt.show()
fig.savefig('1c.png')

```

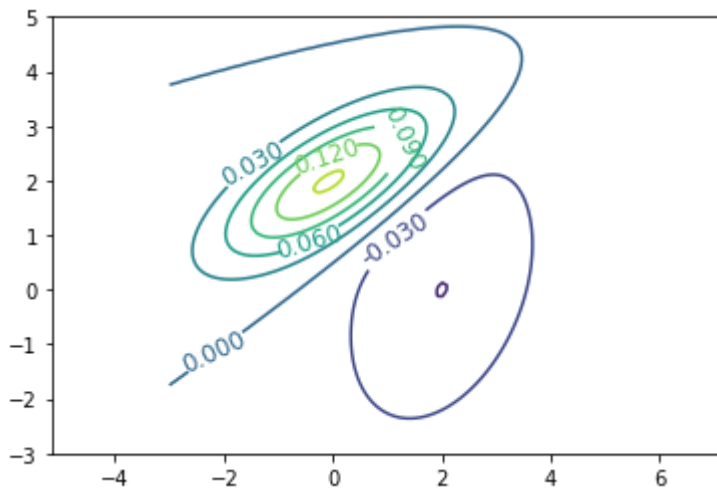


In [6]:

```

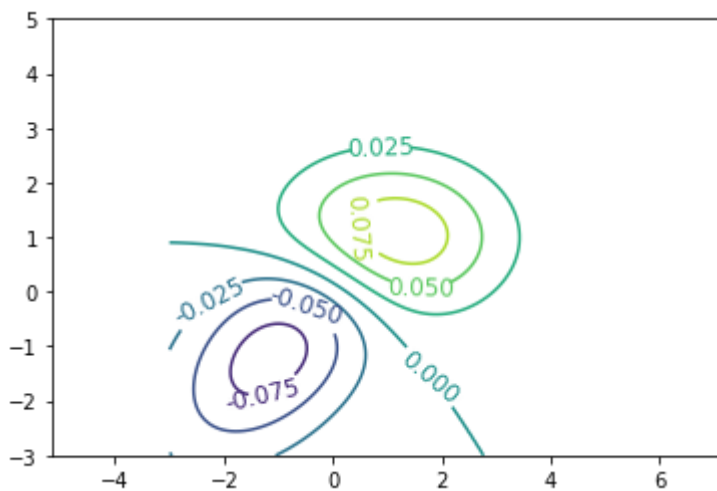
# d
fig = plt.figure()
xi, yi = np.linspace(-3, 5, 100), np.linspace(-3, 5, 100)
xi, yi = np.meshgrid(xi, yi)
mu_1 = np.array([0, 2])
cov_1 = np.array([[2, 1], [1, 1]])
z_1 = np.array([normal_dist(mu_1, cov_1, x, y) for (x,y) in zip(xi.ravel(), yi.ravel())]).reshape(xi.shape)
mu_2 = np.array([2, 0])
cov_2 = np.array([[2, 1], [1, 4]])
z_2 = np.array([normal_dist(mu_2, cov_2, x, y) for (x,y) in zip(xi.ravel(), yi.ravel())]).reshape(xi.shape)
Con = plt.contour(z_1-z_2, extent=[xi.min(), xi.max(), yi.min(), yi.max()])
plt.clabel(Con, inline=1, fontsize=12)
plt.axis('equal')
plt.show()
fig.savefig('1d.png')

```



In [5]:

```
# e
fig = plt.figure()
xi, yi = np.linspace(-3, 5, 100), np.linspace(-3, 5, 100)
xi, yi = np.meshgrid(xi, yi)
mu_1 = np.array([1, 1])
cov_1 = np.array([[2, 0], [0, 1]])
z_1 = np.array([normal_dist(mu_1, cov_1, x, y) for (x,y) in zip(xi.ravel(), yi.ravel())]).reshape(xi.shape)
mu_2 = np.array([-1, -1])
cov_2 = np.array([[2, 1], [1, 2]])
z_2 = np.array([normal_dist(mu_2, cov_2, x, y) for (x,y) in zip(xi.ravel(), yi.ravel())]).reshape(xi.shape)
Con = plt.contour(z_1-z_2, extent=[xi.min(), xi.max(), yi.min(), yi.max()])
plt.clabel(Con, inline=1, fontsize=12)
plt.axis('equal')
plt.show()
fig.savefig('1e.png')
```



## HW3 Q3 Code

In [2]:

```
import numpy as np
import random
import matplotlib.pyplot as plt
from numpy.random import normal
np.random.seed(42)
x1 = normal(3, 3, 100)
x2 = normal(4, 2, 100)
x2 = x2 + x1/2
```

In [3]:

```
# Compute the mean (in R2) of the sample
mu = np.array([x1.mean(), x2.mean()])
print(mu)
```

```
[2.68846045 5.3888394 ]
```

In [4]:

```
# Compute the 2*2 covariance matrix of the sample
x = np.vstack((x1, x2))
cov = np.cov(x)
print(cov)
```

```
[[7.42292904 3.00253936]
 [3.00253936 4.78474509]]
```

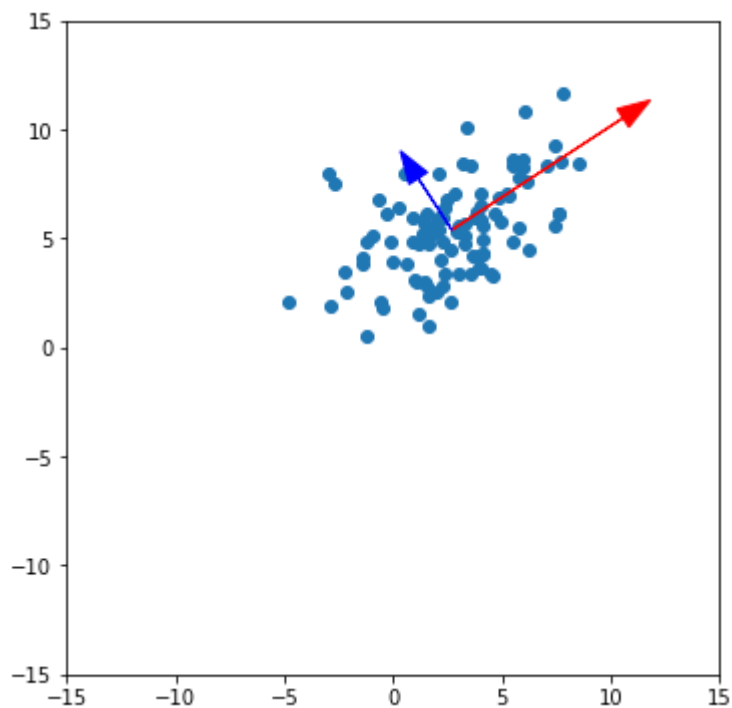
In [5]:

```
# Compute the eigenvectors and eigenvalues of this covariance matrix
evalues, evectors = np.linalg.eig(cov)
print("Eigenvalue: ", evalues)
print("Eigenvectors: ", evectors)
```

```
Eigenvalue: [9.38335628 2.82431785]
Eigenvectors: [[ 0.83732346 -0.54670781]
 [ 0.54670781  0.83732346]]
```

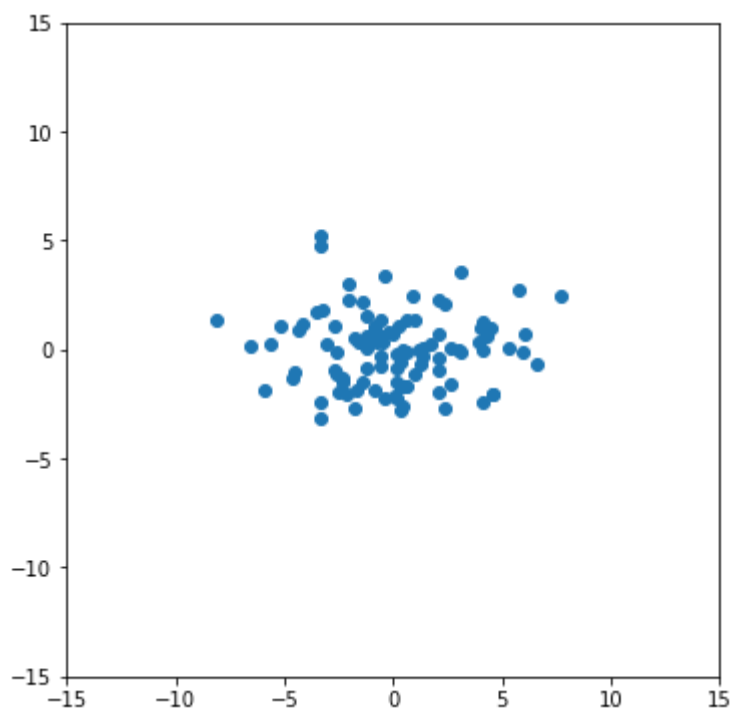
In [8]:

```
# plot all n = 100 data points, and arrows representing both covariance eigenvectors
v1 = evectors[:, 0].dot(evalues[0])
v2 = evectors[:, 1].dot(evalues[1])
fig = plt.figure(figsize=(6,6))
plt.scatter(x1, x2)
ax = plt.axes()
ax.set_aspect('equal')
ax.arrow(mu[0], mu[1], v1[0], v1[1], head_width=1, fc='r', ec='r')
ax.arrow(mu[0], mu[1], v2[0], v2[1], head_width=1, fc='b', ec='b')
plt.xlim([-15,15])
plt.ylim([-15,15])
plt.show()
fig.savefig('3d.png')
```



In [6]:

```
# Plot these rotated points
c_x1 = x1 - mu[0]
c_x2 = x2 - mu[1]
xr = evecs.T.dot(np.array([c_x1, c_x2]))
fig = plt.figure(figsize=(6,6))
plt.scatter(xr[0, :], xr[1, :])
plt.xlim([-15,15])
plt.ylim([-15,15])
ax = plt.axes()
ax.set_aspect('equal')
plt.show()
fig.savefig('3e.png')
```



## HW3.7.ab\_code

In [10]:

```
## fit a gaussian distribution
# Load data
import sys
if sys.version_info[0] < 3:
    raise Exception("Python 3 not detected.")
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm
from scipy import io
import random

mnist = io.loadmat("mnist-data/%s_data.mat" % "mnist")
print("\nloaded %s data!" % mnist)
fields = "test_data", "training_data", "training_labels"
for field in fields:
    print(field, mnist[field].shape)

mTraining = mnist['training_data']
mLabel = mnist['training_labels']
mTest = mnist['test_data']
```

```
loaded {'__header__': b'MATLAB 5.0 MAT-file Platform: posix, Created on: W
ed Feb 13 17:30:33 2019', '__version__': '1.0', '__globals__': [], 'traini
ng_data': array([[0, 0, 0, ..., 0, 0, 0],
                 [0, 0, 0, ..., 0, 0, 0],
                 [0, 0, 0, ..., 0, 0, 0],
                 ...,
                 [0, 0, 0, ..., 0, 0, 0],
                 [0, 0, 0, ..., 0, 0, 0],
                 [0, 0, 0, ..., 0, 0, 0]], dtype=uint8), 'training_labels': array
([[8],
  [2],
  [7],
  ...,
  [3],
  [6],
  [0]], dtype=uint8), 'test_data': array([[0, 0, 0, ..., 0, 0, 0],
                 [0, 0, 0, ..., 0, 0, 0],
                 [0, 0, 0, ..., 0, 0, 0],
                 ...,
                 [0, 0, 0, ..., 0, 0, 0],
                 [0, 0, 0, ..., 0, 0, 0],
                 [0, 0, 0, ..., 0, 0, 0]], dtype=uint8)} data!
test_data (10000, 784)
training_data (60000, 784)
training_labels (60000, 1)
```

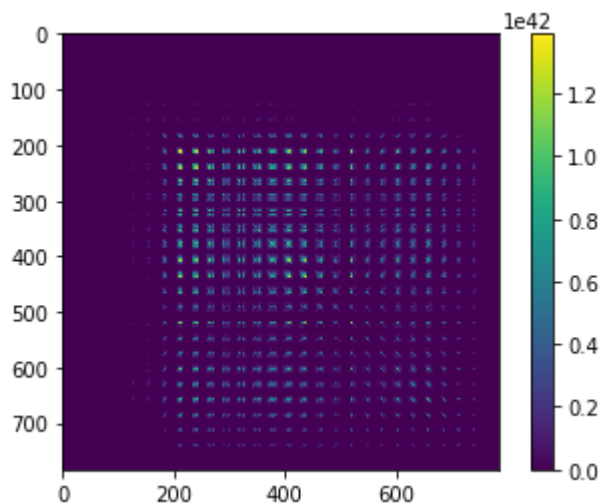
In [12]:

```
# Contrast-normalize the images
def normalize(X):
    X_n = np.zeros(X.shape)
    for i in range(X.shape[0]):
        x = X[i, :]
        X_n[i, :] = (x/(np.sqrt(x.dot(x))+1e-20))
    return X_n
X_train = normalize(mTraining)
```

In [13]:

```
# Computing a mean and a covariance matrix for each digit class
new = pd.DataFrame(mLabel)
for digit in range(0, len(new[0].unique())):
    X_train_d = X_train[new[0]==digit, :]
    mean = X_train_d.mean(axis=0)
    cov = np.cov(X_train_d.T)

# Visualize the covariance matrix for a particular class (digit)
plt.imshow(cov)
plt.colorbar()
plt.show()
```



```

1  # lda mnist code
2
3  import numpy as np
4
5  # LDA
6  def lda_mean(X,y):
7      Mean = np.zeros([len(set(y)), X.shape[1]])
8      for i, label in enumerate(set(y)):
9          X_class = X[y==label,:]
10         Nc = X_class.shape[0]
11         Mean[i, :] = X_class.mean(axis=0)
12     return Mean
13
14 def lda_covariance(X,y):
15     Cov = 0
16     N = len(y)
17     for i, label in enumerate(set(y)):
18         X_class = X[y==label,:]
19         Nc = X_class.shape[0]
20         Cov += np.cov(X_class.T) * Nc
21     return Cov / N
22
23 def lda_P(X,y):
24     N = len(y)
25     P = np.zeros([len(set(y))])
26     for i, label in enumerate(set(y)):
27         X_class = X[y==label,:]
28         Nc = X_class.shape[0]
29         P[i] = Nc / N
30     return P
31
32 def predict(X_test,Mean,Cov,P): # X_val or X_test
33     PreMat = np.linalg.pinv(Cov)
34     L = Mean.dot(PreMat).dot(X_test.T).T \
35         - 1/2*np.diag(Mean.dot(PreMat).dot(Mean.T)) \
36         + np.log(P)
37     yp = np.argmax(L.T, axis=0)
38     return yp
39
40 def accuracy(X_test,y_test,Mean,Cov,P):
41     yp = predict(X_test,Mean,Cov,P)
42     N = len(y_test)
43     error = (yp != y_test).sum()
44     return 1 - error/N
45
46 def normalize(X):
47     X_n = np.zeros(X.shape)
48     for i in range(X.shape[0]):
49         x = X[i, :]
50         X_n[i, :] = (x/(np.sqrt(x.dot(x))+1e-20))
51     return X_n
52
53
54 # load data
55 import sys
56 if sys.version_info[0] < 3:
57     raise Exception("Python 3 not detected.")
58 import pandas as pd
59 import numpy as np

```

```

60 import matplotlib.pyplot as plt
61 from sklearn import svm
62 from scipy import io
63 import random
64
65 mnist = io.loadmat("mnist-data/%s_data.mat" % "mnist")
66 print("\nloaded %s data!" % mnist)
67 fields = "test_data", "training_data", "training_labels"
68 for field in fields:
69     print(field, mnist[field].shape)
70
71 mTraining = mnist['training_data']
72 mLabel = mnist['training_labels']
73 mTest = mnist['test_data']
74 train_label = np.concatenate((mTraining, mLabel), axis=1)
75 Mcolname = np.array([str(i) for i in np.arange(mTraining.shape[1] +
76     1)])
77 train_label_df = pd.DataFrame(data = train_label, columns = Mcolname)
78 num_of_rows = int(train_label_df.shape[0] * 1/6)
79 train_label_df.sample(frac = 1)
80
81 train_label_df = np.array(train_label_df)
82 X_train, y_train =
83     normalize(train_label_df[10000:,-1]),train_label_df[10000:,-1]
84 X_val,y_val =
85     normalize(train_label_df[:10000,-1]),train_label_df[:10000,-1]
86
87 sample_size = np.array([100, 200, 500, 1000, 2000, 5000, 10000,
88     30000, 50000])
89 accuracy_matrix = np.zeros(len(sample_size))
90
91 for i, N in enumerate(sample_size):
92     idx = random.sample(range(50000), N)
93     Mean = lda_mean(X_train[idx, :], y_train[idx])
94     Cov = lda_covariance(X_train[idx, :], y_train[idx])
95     P = lda_P(X_train[idx, :], y_train[idx])
96     accuracy_matrix[i] = accuracy(X_val, y_val,Mean,Cov,P)
97
98 plt.plot(sample_size, accuracy_matrix * 100)
99 plt.xscale('log')
100 plt.xlabel('Sample size')
101 plt.ylabel('Accuracy(%)')
102 plt.show()
103
104 # best sample size = 250
105 N = 250
106 idx = random.sample(range(50000), N)
107 Mean = lda_mean(X_train[idx, :], y_train[idx])
108 Cov = lda_covariance(X_train[idx, :], y_train[idx])
109 P = lda_P(X_train[idx, :], y_train[idx])
110 yp_lda = predict(X_val,Mean,Cov,P)
111 accuracy(X_val,y_val,Mean,Cov,P)
112
113 yp_lda = predict(mTest,Mean,Cov,P)

```

```

1  import numpy as np
2
3  def qda_mean(X,y):
4      Mean = np.zeros([len(set(y)), X.shape[1]])
5      for i, label in enumerate(set(y)):
6          X_class = X[y==label,:]
7          Nc = X_class.shape[0]
8          Mean[i, :] = X_class.mean(axis=0)
9      return Mean
10
11 def qda_covariance(X,y,a):
12     Cov = np.zeros([X.shape[1], X.shape[1], len(set(y))])
13     N = len(y)
14     for i, label in enumerate(set(y)):
15         X_class = X[y==label,:]
16         Cov[:, :, i] = np.cov(X_class.T) + a*np.eye(X.shape[1])
17     return Cov
18
19 def qda_P(X,y):
20     N = len(y)
21     P = np.zeros([len(set(y))])
22     for i, label in enumerate(set(y)):
23         X_class = X[y==label,:]
24         Nc = X_class.shape[0]
25         P[i] = Nc / N
26     return P
27
28 def qda_Q(X_test,Mean,Cov,P):
29     i = np.eye(Cov.shape[0], Cov.shape[0])
30     PreMat = np.linalg.lstsq(Cov,i)[0]
31     Const = - 1/2*np.log(np.linalg.det(Cov)+1e-20) + np.log(P)
32     Qc = np.array([(-1/2*(x-Mean).dot(PreMat).dot(x-Mean)) + Const
33                     for x in X_test])
34     return Qc
35
36 def predict(X_test,Mean,Cov,P): # X_val or X_test
37     nC = len(P)
38     L = np.zeros([nC, len(X_test)])
39     for i in range(nC):
40         L[i, :] = qda_Q(X_test, Mean[i], Cov[:, :, i], P[i])
41     yp = np.argmax(L, axis=0)
42     return yp
43
44 def accuracy(X_test,y_test,Mean,Cov,P):
45     yp = predict(X_test,Mean,Cov,P)
46     N = len(y_test)
47     error = (yp != y_test).sum()
48     return 1 - error/N
49
50 def normalize(X):
51     X_n = np.zeros(X.shape)
52     for i in range(X.shape[0]):
53         x = X[i, :]
54         X_n[i, :] = (x/(np.sqrt(x.dot(x))+1e-20))
55     return X_n
56
57 import sys
58 if sys.version_info[0] < 3:
59     raise Exception("Python 3 not detected.")

```

```

59 import pandas as pd
60 import numpy as np
61 import matplotlib.pyplot as plt
62 from sklearn import svm
63 from scipy import io
64 import random
65
66 mnist = io.loadmat("mnist-data/%s_data.mat" % "mnist")
67 print("\nloaded %s data!" % mnist)
68 fields = "test_data", "training_data", "training_labels"
69 for field in fields:
70     print(field, mnist[field].shape)
71
72 mTraining = mnist['training_data']
73 mLabel = mnist['training_labels']
74 mTest = mnist['test_data']
75 train_label = np.concatenate((mTraining, mLabel), axis=1)
76 Mcolname = np.array([str(i) for i in np.arange(mTraining.shape[1] +
77     1)])
78 train_label_df = pd.DataFrame(data = train_label, columns = Mcolname)
79 num_of_rows = int(train_label_df.shape[0] * 1/6)
80 train_label_df.sample(frac = 1)
81
82 train_label_df = np.array(train_label_df)
83 X_train, y_train =
84     normalize(train_label_df[10000:,-1]),train_label_df[10000:,-1]
85 X_val,y_val =
86     normalize(train_label_df[:10000,-1]),train_label_df[:10000,-1]
87
88 sample_size = np.array([100, 200, 500, 1000, 2000, 5000, 10000,
89     30000, 50000])
90 accuracy_matrix = np.zeros(len(sample_size))
91
92 for i, N in enumerate(sample_size):
93     idx = random.sample(range(50000), N)
94     Mean = qda_mean(X_train[idx, :], y_train[idx])
95     Cov = qda_covariance(X_train[idx, :], y_train[idx],1e-8)
96     P = qda_P(X_train[idx, :], y_train[idx])
97     accuracy_matrix[i] = accuracy(X_val, y_val,Mean,Cov,P)
98
99 plt.plot(sample_size, accuracy_matrix * 100)
100 plt.xscale('log')
101 plt.xlabel('Sample size')
102 plt.ylabel('Accuracy(%)')
103 plt.show()
104
105 # best sample size = 200
106 N = 200
107 idx = random.sample(range(50000), N)
108 Mean = qda_mean(X_train[idx, :], y_train[idx])
109 Cov = qda_covariance(X_train[idx, :], y_train[idx],1e-8)
110 P = qda_P(X_train[idx, :], y_train[idx])
111 yp_lda = predict(X_val,Mean,Cov,P)
112 accuracy(X_val,y_val,Mean,Cov,P)
113
114 yp_lda_m = predict(mTest,Mean,Cov,P)
115
116 ##### QDA performs better than LDA

```

```

1  # QDA_spam
2  import numpy as np
3
4  def qda_mean(X,y):
5      Mean = np.zeros([len(set(y)), X.shape[1]])
6      for i, label in enumerate(set(y)):
7          X_class = X[y==label,:]
8          Nc = X_class.shape[0]
9          Mean[i, :] = X_class.mean(axis=0)
10     return Mean
11
12     def qda_covariance(X,y,a):
13         Cov = np.zeros([X.shape[1], X.shape[1], len(set(y))])
14         N = len(y)
15         for i, label in enumerate(set(y)):
16             X_class = X[y==label,:]
17             Cov[:, :, i] = np.cov(X_class.T) + a*np.eye(X.shape[1])
18         return Cov
19
20     def qda_P(X,y):
21         N = len(y)
22         P = np.zeros([len(set(y))])
23         for i, label in enumerate(set(y)):
24             X_class = X[y==label,:]
25             Nc = X_class.shape[0]
26             P[i] = Nc / N
27         return P
28
29     def qda_Q(X_test,Mean,Cov,P):
30         i = np.eye(Cov.shape[0], Cov.shape[0])
31         PreMat = np.linalg.lstsq(Cov,i)[0]
32         Const = - 1/2*np.log(np.linalg.det(Cov)+1e-20) + np.log(P)
33         Qc = np.array([(-1/2*(x-Mean).dot(PreMat).dot(x-Mean)) + Const
34             for x in X_test])
35         return Qc
36
37     def predict(X_test,Mean,Cov,P): # X_val or X_test
38         nC = len(P)
39         L = np.zeros([nC, len(X_test)])
40         for i in range(nC):
41             L[i, :] = qda_Q(X_test, Mean[i], Cov[:, :, i], P[i])
42         yp = np.argmax(L, axis=0)
43         return yp
44
45     def accuracy(X_test,y_test,Mean,Cov,P):
46         yp = predict(X_test,Mean,Cov,P)
47         N = len(y_test)
48         error = (yp != y_test).sum()
49         return 1 - error/N
50
51     import sys
52     if sys.version_info[0] < 3:
53         raise Exception("Python 3 not detected.")
54     import pandas as pd
55     import numpy as np
56     import matplotlib.pyplot as plt
57     from sklearn import svm
58     from scipy import io

```

```

59 import random
60
61 spam = io.loadmat("spam-data/%s_data.mat" % "spam")
62 print("\nloaded %s data!" % spam)
63 fields = "test_data", "training_data", "training_labels"
64 for field in fields:
65     print(field, spam[field].shape)
66
67 sTraining = spam['training_data']
68 sLabel = spam['training_labels']
69 sTest = spam['test_data']
70 train_label = np.concatenate((sTraining, sLabel), axis=1)
71 Scolname = np.array([str(i) for i in np.arange(sTraining.shape[1] +
72 1)])
73 train_label_df = pd.DataFrame(data = train_label, columns = Scolname)
74 num_of_rows = int(train_label_df.shape[0] * 1/6)
75 train_label_df.sample(frac = 1)
76
77 train_label_df = np.array(train_label_df)
78 X_train, y_train = train_label_df[2000:,-1],train_label_df[2000:,-1]
79 X_val,y_val = train_label_df[:2000,-1],train_label_df[:2000,-1]
80
81 idx = random.sample(range(3172), 2000)
82 Mean = qda_mean(X_train[idx, :], y_train[idx])
83 Cov = qda_covariance(X_train[idx, :], y_train[idx],1e-8)
84 P = qda_P(X_train[idx, :], y_train[idx])
85 yp_lda = predict(X_val,Mean,Cov,P)
86 accuracy_matrix[i] = accuracy(X_val, y_val,Mean,Cov,P)

```

I certify that all solutions are entirely in my own words and that I have not looked at another student's solutions. I have given credit to all external sources I consulted.

Di Zhen