CS 189 Homework3

Xu Zhihao

July 15, 2019

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.

Signature: Zhihao Xu

Contents

1 Gaussian Classification

(a) Since

$$P(C_i|x) = \frac{P(x|C_i) \cdot P(C_i)}{p(x)}$$

and $P(C_1) = P(C_2) = \frac{1}{2}$. So, if we want to compare $P(C_1|x)$ and $P(C_2|x)$, we only need to compare $P(x|C_1)$ and $P(x|C_2)$. Here we are going to solve

$$P(x|C_1) \geq P(x|C_2)$$

which is equivalent to

$$(x - \mu_1)^2 \leq (x - \mu_2)^2$$
$$\Rightarrow (2x - \mu_1 - \mu_2)(\mu_2 - \mu_1) \leq 0$$

Since $\mu_2 \geq \mu_1$, we can get

$$x \leq \frac{\mu_1 + \mu_2}{2}$$

Here the decision boundary is

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

The decision rule is

$$\begin{cases} C_1 & x < \frac{\mu_1+\mu_2}{2} \\ C_2 & x > \frac{\mu_1+\mu_2}{2} \end{cases}$$

(b) First we compute $P\left((\text{misclassified as } C_1)|C_2\right)$

$$\begin{aligned} P\left((\text{misclassified as } C_1)|C_2\right) &= P(x < \frac{\mu_1+\mu_2}{2}|C_2) \\ &= \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\frac{\mu_1+\mu_2}{2}} \frac{1}{\sigma}\exp\left\{-\frac{(x-\mu_2)^2}{2\sigma^2}\right\}dx \end{aligned}$$

Substitute x by $y = \frac{x-\mu_2}{\sigma}$

$$\begin{aligned} P\left((\text{misclassified as } C_1)|C_2\right) &= \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\frac{\mu_1-\mu_2}{2\sigma}} \exp\left\{-\frac{z^2}{2}\right\}dz \\ &= \frac{1}{\sqrt{2\pi}}\int_{\frac{\mu_2-\mu_1}{2\sigma}}^{+\infty} \exp\left\{-\frac{z^2}{2}\right\}dz \end{aligned}$$

Similarly we can get

$$P\left((\text{misclassified as } C_2)|C_1\right) = \frac{1}{\sqrt{2\pi}}\int_{\frac{\mu_2-\mu_1}{2\sigma}}^{+\infty} \exp\left\{-\frac{z^2}{2}\right\}dz$$

Substitute it back, we can get

$$\begin{aligned} P_e &= \frac{1}{\sqrt{2\pi}}\int_{\frac{\mu_2-\mu_1}{2\sigma}}^{+\infty} \exp\left\{-\frac{z^2}{2}\right\}dz \times \frac{1}{2} + \frac{1}{\sqrt{2\pi}}\int_{\frac{\mu_2-\mu_1}{2\sigma}}^{+\infty} \exp\left\{-\frac{z^2}{2}\right\}dz \times \frac{1}{2} \\ &= \frac{1}{\sqrt{2\pi}}\int_{\frac{\mu_2-\mu_1}{2\sigma}}^{+\infty} \exp\left\{-\frac{z^2}{2}\right\}dz \end{aligned}$$

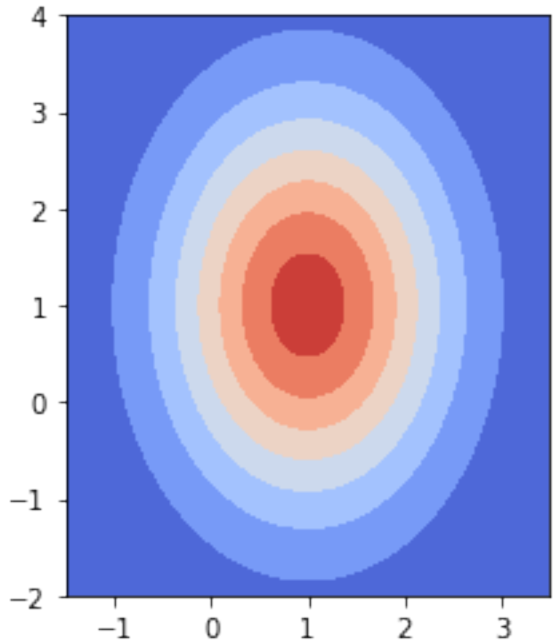
2 Isocontours of Normal Distributions

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        import pandas as pd
        import scipy.io
        import random
        from scipy.stats import multivariate_normal
        import seaborn as sns
```

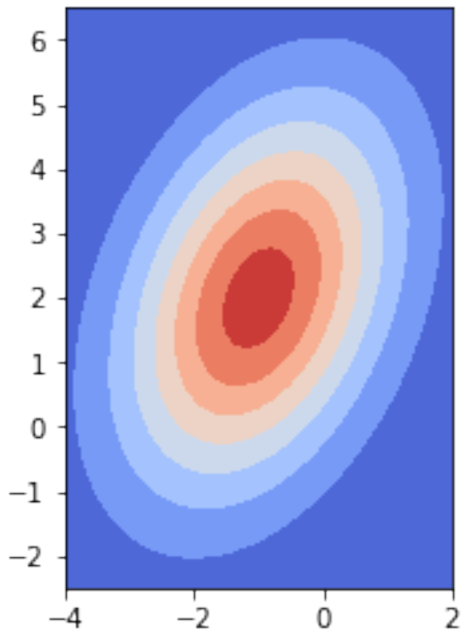
```
In [2]: def multiNormalPdf(mu,Sigma,pos):
            d = mu.size
            fac = np.einsum('...k,kl,...l->...', \
                            pos-mu, np.linalg.inv(Sigma), pos-mu)
            Z = 1/(np.sqrt(2*np.pi)**d * \
                   np.sqrt(np.linalg.det(Sigma))) *np.exp(-fac / 2)
            return Z
```

```
In [3]: # (a)
        def ContourPlot1(mu,Sigma,xmin,xmax,ymin,ymax):
            N = 60
            X = np.linspace(xmin, xmax, N)
            Y = np.linspace(ymin, ymax, N)
            X, Y = np.meshgrid(X, Y)

            pos = np.empty(X.shape + (2,))
            pos[:, :, 0] = X
            pos[:, :, 1] = Y

            Z = multiNormalPdf(mu,Sigma,pos)
            fig = plt.figure()
            ax = fig.gca()
            ax.set_aspect(1)
            cset = ax.contourf(X, Y, Z, cmap='coolwarm')
```

```
In [4]: Sigma = np.array([[1,0], [0,2]])
        mu = np.array([1,1])
        ContourPlot1(mu,Sigma,-1.5,3.5,-2,4)
```

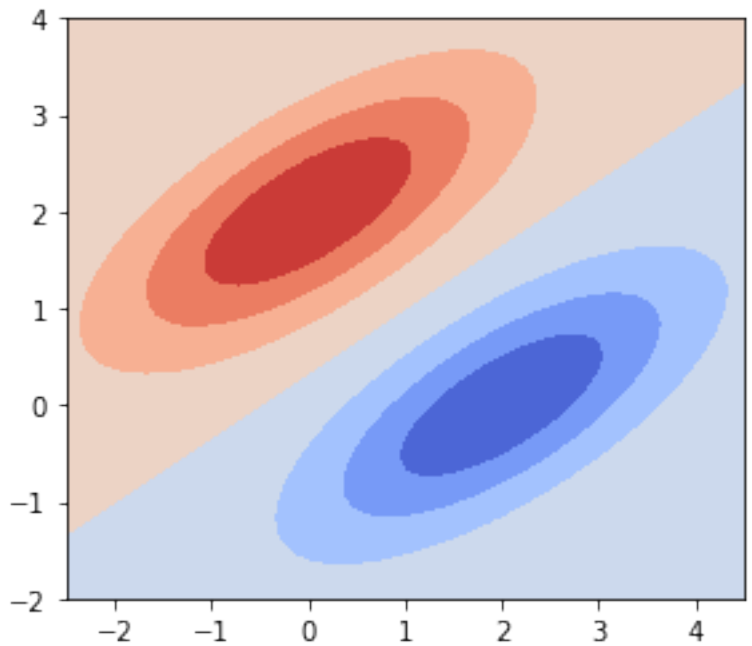



```
In [5]: # (b)
        Sigma = np.array([[2,1], [1,4]])
        mu = np.array([-1,2])
        ContourPlot1(mu,Sigma,-4,2,-2.5,6.5)
```



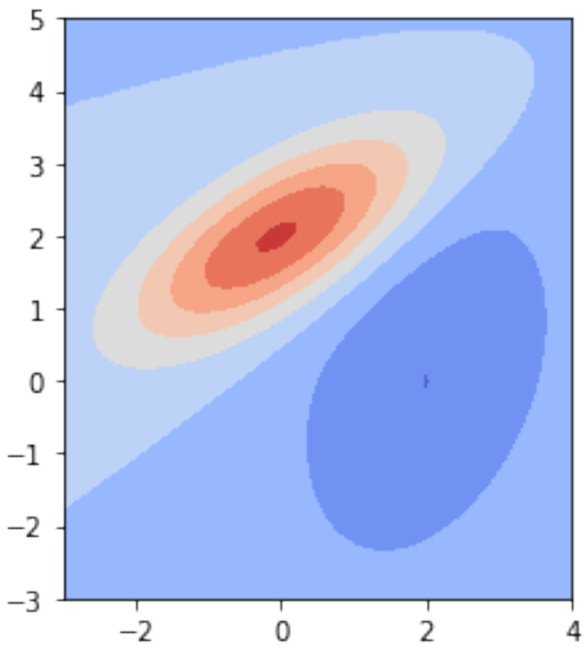
```
In [6]: def ContourPlot2(mu_1,Sigma_1,mu_2,Sigma_2, \
                         xmin,xmax,ymin,ymax):
            N = 60
            X = np.linspace(xmin, xmax, N)
            Y = np.linspace(ymin, ymax, N)
            X, Y = np.meshgrid(X, Y)

            pos = np.empty(X.shape + (2,))
            pos[:, :, 0] = X
            pos[:, :, 1] = Y

            Z = multiNormalPdf(mu_1,Sigma_1,pos) - \
                        multiNormalPdf(mu_2,Sigma_2,pos)
            fig = plt.figure()
            ax = fig.gca()
            ax.set_aspect(1)
            cset = ax.contourf(X, Y, Z, cmap='coolwarm')
```

```
In [7]: # (c)
        Sigma = np.array([[2,1], [1,1]])
        mu_1 = np.array([0,2])
        mu_2 = np.array([2,0])
        ContourPlot2(mu_1,Sigma,mu_2,Sigma,-2.5,4.5,-2,4)
```

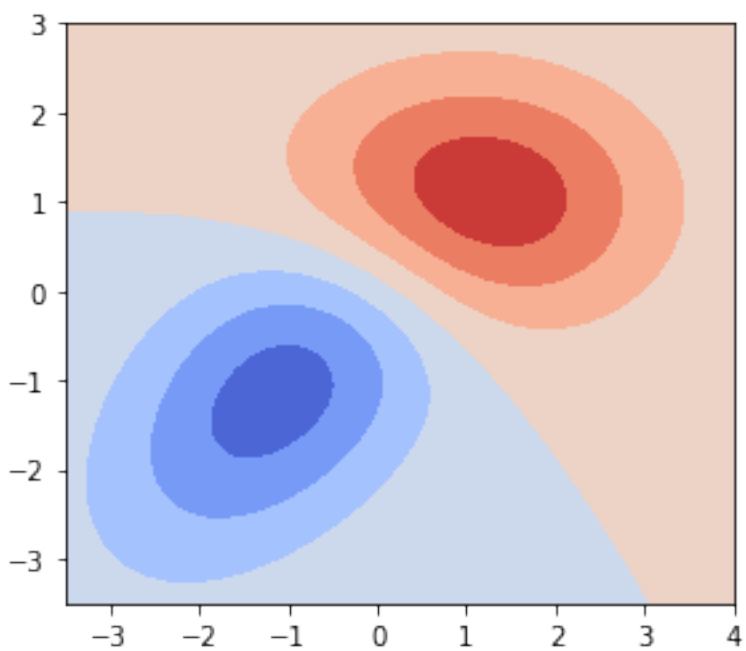



```
In [8]: # (d)
        Sigma_1 = np.array([[2,1], [1,1]])
        Sigma_2 = np.array([[2,1], [1,4]])
```

```
mu_1 = np.array([0,2])
mu_2 = np.array([2,0])
ContourPlot2(mu_1,Sigma_1,mu_2,Sigma_2,-3,4,-3,5)
```



```
In [9]: # (e)
        Sigma_1 = np.array([[2,0], [0,1]])
        Sigma_2 = np.array([[2,1], [1,2]])
        mu_1 = np.array([1,1])
        mu_2 = np.array([-1,-1])
        ContourPlot2(mu_1,Sigma_1,mu_2,Sigma_2,-3.5,4,-3.5,3)
```



3 Eigenvectors of the Gaussian Covariance Matrix

In [10]:
```
np.random.seed(10130)
x_1 = np.random.normal(3,3,size=100)
np.random.seed(189)
x_2 = x_1 / 2 + np.random.normal(4,2,size=100)
```

In [11]:
```
# (a)
mean = tuple([np.mean(x_1),np.mean(x_2)])
print("The mean of sample is",mean)
```

```
The mean of sample is (3.320627223330448, 5.394127429864527)
```

In [12]:
```
# (b)
X = np.stack((x_1,x_2),axis=0)
Sigma = np.cov(X)
print("The covariance matrix of the sample is \n",Sigma)
```

```
The covariance matrix of the sample is
 [[7.85220172 5.08973215]
 [5.08973215 7.61491086]]
```

In [13]:
```
# (c)
eigenVal, eigenVec = np.linalg.eig(Sigma)
idx = np.argsort(eigenVal)[::-1]
eigenVec = eigenVec[:,idx]
eigenVal = eigenVal[idx]
eigenVal
```

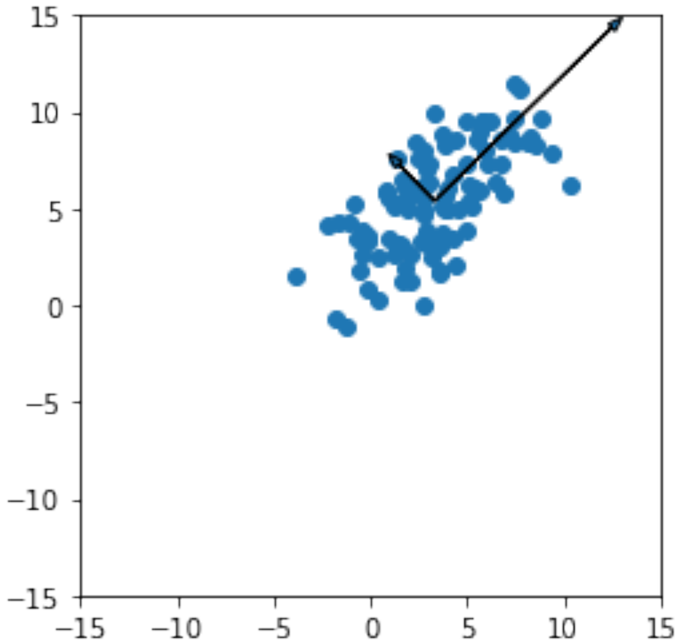
Out[13]:
```
array([12.82467112,  2.64244147])
```

In [14]:
```
eigenVec
```

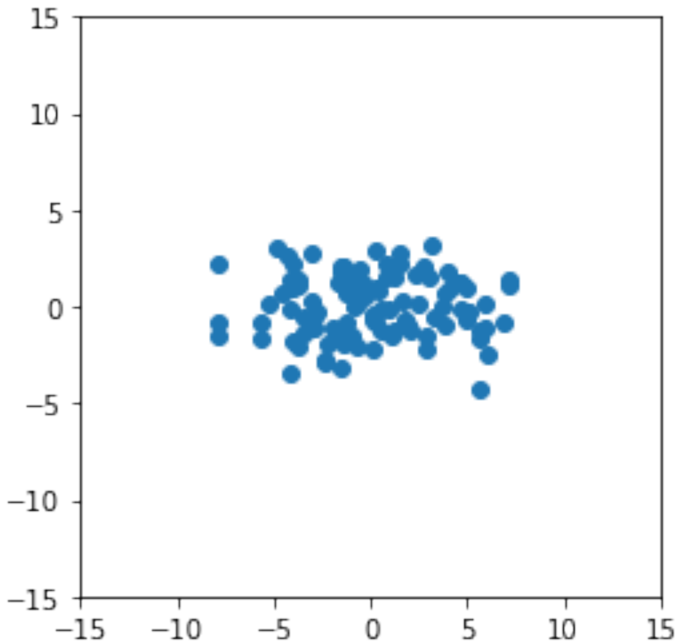
Out[14]:
```
array([[ 0.71529868, -0.69881886],
       [ 0.69881886,  0.71529868]])
```

In [15]:
```
# (d)
plt.scatter(x_1,x_2)
ax = plt.gca()
ax.set_aspect(1)
plt.xlim(-15,15)
plt.ylim(-15,15)
plt.arrow(mean[0],mean[1],eigenVec[0][0]*eigenVal[0], \
          eigenVec[1][0]*eigenVal[0],head_width=0.5)
plt.arrow(mean[0],mean[1],eigenVec[0][1]*eigenVal[1], \
          eigenVec[1][1]*eigenVal[1],head_width=0.5)
```

Out[15]:
```
<matplotlib.patches.FancyArrow at 0x1a21ff0cc0>
```



In [16]:
```
# (e)
ax = plt.gca()
ax.set_aspect(1)
X[0] = X[0] - mean[0]
X[1] = X[1] - mean[1]
X_rote = np.matmul(eigenVec.T,X)
plt.scatter(X_rote[0],X_rote[1])
plt.xlim(-15,15)
plt.ylim(-15,15)
plt.show()
```



4 Classification

(a) Using the given decision rule, when $i = 1, 2, 3, \cdots, c$

$$\begin{aligned} R(r(x) = i|x) &= \sum_{j \neq i}^{c} \lambda_s P(Y = j|x) + 0 \times P(Y = i|x) \\ &= \lambda_s \sum_{j=1}^{c} P(Y = j|x) - \lambda_s \times P(Y = i|x) \end{aligned}$$

when $i = c + 1$

$$R(r(x) = c + 1|x) = \lambda_r \sum_{j=1}^{c} P(Y = j|x)$$

Since we want to minimize the risk, when choose class i we need to solve

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

The solution to this inequality is

$$\begin{cases} P(Y = i|x) \geq P(Y = j|x) \\ P(Y = i|x) \geq \frac{\lambda_s - \lambda_r}{\lambda_s} = 1 - \frac{\lambda_r}{\lambda_s} \end{cases}$$

which is equivalent to our decision rule.

(b) When $\lambda_r = 0$,

$$R(r(x) = c + 1|x) = 0 \leq R(r(x) = i|x)$$

So, for all x, we just choose class doubt. Intuitively, since choose class doubt is no cost, we prefer to choose this class.

When $\lambda_r > \lambda_s$,

$$R(r(x) = i|x) \leq \lambda_s \sum_{j=1}^{c} P(Y = j|x) \leq \lambda_r \sum_{j=1}^{c} P(Y = j|x) = R(r(x) = c + 1|x)$$

So, here for all x, we will never choose class doubt. Intuitively, since the cost of predict doubt is even higher than the cost of predicting wrong, we will never choose it.

5 Maximum Likelihood Estimation

(a) The pdf of multivariate normal distribution is

$$f(x;\mu,\Sigma) = \frac{1}{\sqrt{(2\pi)^d|\Sigma|}} \exp\left[-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right]$$

The likelihood function is

$$L(\mu,\Sigma) = \left(\frac{1}{\sqrt{(2\pi)^d|\Sigma|}}\right)^n \exp\left\{\sum_{i=1}^n \left[-\frac{1}{2}(x_i-\mu)^T\Sigma^{-1}(x_i-\mu)\right]\right\}$$

The log-likelihood is

$$\log L(\mu,\Sigma) = -\frac{dn}{2}\log 2\pi - \frac{n}{2}\log|\Sigma| - \frac{1}{2}\sum_{i=1}^n (x_i-\mu)^T\Sigma^{-1}(x_i-\mu)$$

First we compute $\hat{\mu}$

$$\begin{aligned}
\frac{\partial \log L(\mu,\Sigma)}{\partial\mu} &= -\frac{1}{2}\sum_{i=1}^n \left\{(-1)\cdot\Sigma^{-1}(x_i-\mu) + (-1)\Sigma^{-1}(x_i-\mu)\right\} \\
&= \sum_{i=1}^n \Sigma^{-1}(x_i-\mu) \\
&= \Sigma^{-1}\sum_{i=1}^n (x_i-\mu) = 0 \\
\Rightarrow \hat{\mu} &= \frac{1}{n}\sum_{i=1}^n x_i
\end{aligned}$$

To compute $\hat{\Sigma}$, first we need to do some transformation on the log-likelihood function

$$\begin{aligned}
\log L(\mu,\Sigma) &= -\frac{dn}{2}\log 2\pi - \frac{n}{2}\log|\Sigma| - \frac{1}{2}\sum_{i=1}^n (x_i-\mu)^T\Sigma^{-1}(x_i-\mu) \\
&= -\frac{dn}{2}\log 2\pi - \frac{n}{2}\log|\Sigma| - \frac{1}{2}\sum_{i=1}^n Tr\left[(x_i-\mu)^T\Sigma^{-1}(x_i-\mu)\right] \\
&= -\frac{dn}{2}\log 2\pi - \frac{n}{2}\log|\Sigma| - \frac{1}{2}\sum_{i=1}^n Tr\left[\Sigma^{-1}(x_i-\mu)^T(x_i-\mu)\right] \\
&= -\frac{dn}{2}\log 2\pi - \frac{n}{2}\log|\Sigma| - \frac{1}{2}Tr\left[\sum_{i=1}^n \Sigma^{-1}(x_i-\mu)^T(x_i-\mu)\right]
\end{aligned}$$

Here we take the partial derivative with respect to Σ^{-1}

$$
\begin{aligned}
\frac{\partial \log L(\mu, \Sigma)}{\partial \Sigma^{-1}} &= \frac{n}{2}\Sigma - \frac{1}{2}\sum_{i=1}^{n}(x_i - \mu)^T(x_i - \mu) = 0 \\
\Rightarrow \hat{\Sigma} &= \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^T(x_i - \mu) \\
\hat{\sigma}_i = \hat{\Sigma}_{ii} &= \frac{1}{n}\sum_{j=1}^{n}(x_j - \mu_i)^2
\end{aligned}
$$

(b) Using the result in (a)

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

Since A is invertible

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

6 Covariance Matrices and Decompositions

(a) For $\forall y \in \mathbb{R}^d$,

$$
\begin{aligned}
y^T\hat{\Sigma}y &= y^T\left[\frac{1}{n}\sum_{i=1}^{n}(x_i-\mu)(x_i-\mu)^T\right]y \\
&= \frac{1}{n}\left[\sum_{i=1}^{n} y^T(x_i-\mu)(x_i-\mu)^T y\right] \\
&= \frac{1}{n}\sum_{i=1}^{n}\left\|(x_i-\mu)^T y\right\|_2 \geq -
\end{aligned}
$$

So, $\hat{\Sigma}$ is always PSD. If $y^T\hat{\Sigma}y = 0$, iff. $(x_i-\mu)y = 0$ for $\forall i \in [1, n]$. Then there exist a linear combination of $(x_i-\mu)$, $y = \alpha_1(x_1-\mu)+\alpha_2(x_2-\mu)+\cdots+\alpha_n(x_n-\mu)$, $y^Ty = \alpha_1(x_1-\mu)y+\alpha_2(x_2-\mu)y+\cdots+\alpha_n(x_n-\mu)y = 0$ when $\alpha_1 = \alpha_2 = \cdots = \alpha_n = 1$. So $y = 0$, and here Σ is not invertible. In conclusion, if $\text{span}\{(x_i-\mu)\} = \mathbb{R}^d$, $\hat{\Sigma}$ is invertible. If dimension of $\text{span}\{(x_i-\mu)\} \leq d$, $\hat{\Sigma}$ is not invertible.

Geometrically, if dim $\{(x_1-\mu),(x_2-\mu),\cdots,(x_i-\mu)\} = d$. All the data points are on the subspace of $\mathbb{R}^d$. If dim $\{(x_1-\mu),(x_2-\mu),\cdots,(x_i-\mu)\} = d-1$, there exist two points lying on the same hyperplane. The dimension of $\mathbb{R}^d$ can be deducted.

(b) Take $X = [X_1, X_2, \cdots, X_n] \in \mathbb{R}^{n \times d}$. We can define a centering matrix

$$C_n = I_n - \frac{1}{n}\mathbf{1}\mathbf{1}^T$$

where $\mathbf{1}$ is the all one column vector. Let

$$S = XC_nX = XC_nC_n^TX^T = (XC_n)(XC_n)^T$$

Take r equals the number of non-zero eigenvalues. v_i denote the eigenvector corresponding to the eigenvector λ_i. Take $H = [v_1, v_2, \cdots, v_n] \in \mathbb{R}^{n \times r}$ and $L = \text{diag}\{\lambda_1, \lambda_2, \cdots, \lambda_n\}$. The new maximum likelihood estimator of covariance matrix is

$$\hat{\Sigma} = \frac{1}{d}HLH^T$$

(c) The log-pdf of $N(0, \Sigma)$,

$$\log f(x) = -\frac{d}{2}\log 2\pi - \frac{1}{2}\log\|\Sigma\| - \frac{1}{2}x^T\Sigma^{-1}x$$

So the problem is equivalent to

$$\min_{\|x\|_2=1} x^T\Sigma^{-1}x \text{ and } \max_{\|x\|_2=1} x^T\Sigma^{-1}x$$

Using the Lagrange multiplier,

$$L(x,\lambda) = x^T\Sigma^{-1}x - \lambda(x^Tx - 1)$$

We need to solve

$$\begin{cases} \nabla_{x,\lambda}L(x,\lambda) = 0 \\ x^Tx - 1 = 0 \end{cases}$$

We can get that

$$\nabla_x L(x,\lambda) = 2\Sigma^{-1}x = 0 \text{ and } \nabla_\lambda L(x,\lambda) = 2\lambda x = 0$$

Since $\Sigma^{-1}x = \lambda x$, by definition, λ is the eigenvalue of Σ^{-1}, then

$$x^T\Sigma^{-1}x = x^T\lambda x = \lambda x^Tx = \lambda$$

So

$$\begin{cases} (x^T\Sigma^{-1}x)_{min} = \lambda_{min}(\Sigma^{-1}) = \lambda_{max}(\Sigma) \\ (x^T\Sigma^{-1}x)_{max} = \lambda_{max}(\Sigma^{-1}) = \lambda_{min}(\Sigma) \end{cases}$$

So, the unit eigenvector corresponding to $\lambda_{max}(\Sigma)$ maximize the PDF f(x) and the unit eigenvector corresponding to $\lambda_{min}(\Sigma)$ minimize the PDF f(x)

7 Gaussian Classifiers for Digits and Spam

(a) Fit the $\hat{\mu}_C$ and $\hat{\Sigma}_C$ for each class C, using maximum likelihood estimation. Here we plot the mean of some digit class and check whether it is correct or not. The result are shown in the code section.

(b) Visualize the covariance of a particular digit class. We can notice that the diagonal entries are much brighter than the off-diagonal one, which means the diagonal entries are larger than the off-diagonal entries. Since the diagonal entries actually are the variance of a pixel. Intuitively, the correlation between different pixel point is smaller than the variance itself.

(c) (1) (2) The result and plot of error rate are shown in the code section.

(3) The LDA performs better than QDA. I think the reason is there is no statistical difference between the covariance of each digit class. The QDA will cause extra variance.

(4) Notice that for LDA 2 is the easiest digit class to predict and for QDA 1 is the easiest digit class to predict.

(d) Kaggle username: Jack_xzh
Score of MNIST: 0.88360
Score of SPAM: 0.81953

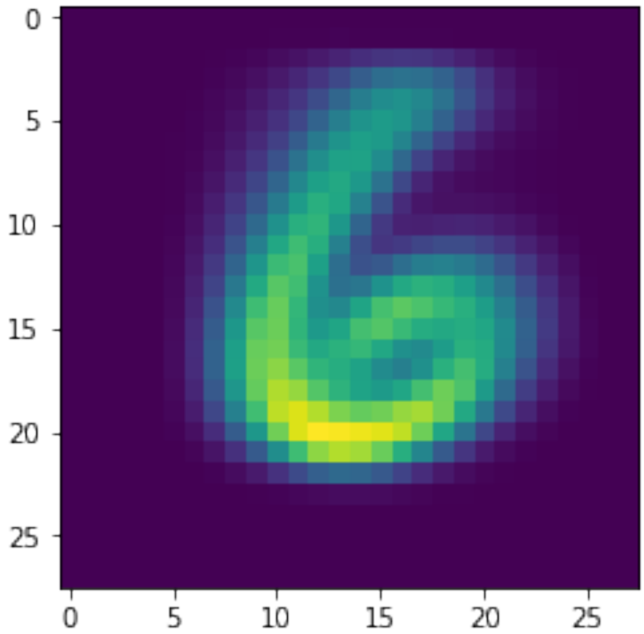
8 Appendix: Code for Gaussian Classifiers

```
In [17]: def processData(name,testing_size):
             folder = name + "-data"
             file = name + "_data"
             path = 'hw3-resources/' + folder + "/" + file + ".mat"
             data = scipy.io.loadmat(path)

             data_X = data["training_data"]
             data_X = data_X.astype(np.float64)
             data_y = data["training_labels"]
             data_t = data["test_data"]

             random.seed(189)
             index = random.sample(range(data_X.shape[0]), \
                                   data_X.shape[0]-testing_size)

             data_X_train = data_X[index]
             data_X_validate = np.delete(data_X, index, axis=0)
             data_y_train = data_y[index]
             data_y_validate = np.delete(data_y, index, axis=0)

             Data = dict()
             Data["X_train"] = data_X_train
             Data["X_validate"] = data_X_validate
             Data["y_train"] = data_y_train
             Data["y_validate"] = data_y_validate
             Data["test"] = data_t

             if name == "mnist":
                 Data["X_train"] = Data["X_train"]/255
                 Data["X_validate"] = Data["X_validate"]/255
                 Data["test"] = Data["test"]/255
             return Data

In [18]: mnistData = processData("mnist",0)
         spamData = processData("spam",0)

In [19]: Mu = []
         Index = []
         Sigma = []
         for i in range(10):
             index = np.where(mnistData["y_train"] == i)[0]
             mu = np.mean(mnistData["X_train"][index], axis=0)
             sigma = np.cov(mnistData["X_train"][index],rowvar=False, bias=True)

             Index.append(index)
             Mu.append(mu)
             Sigma.append(sigma)
```

```
        prior = [index.size/60000 for index in Index]
        SigmaLDA = sum(Sigma)/10.0
```

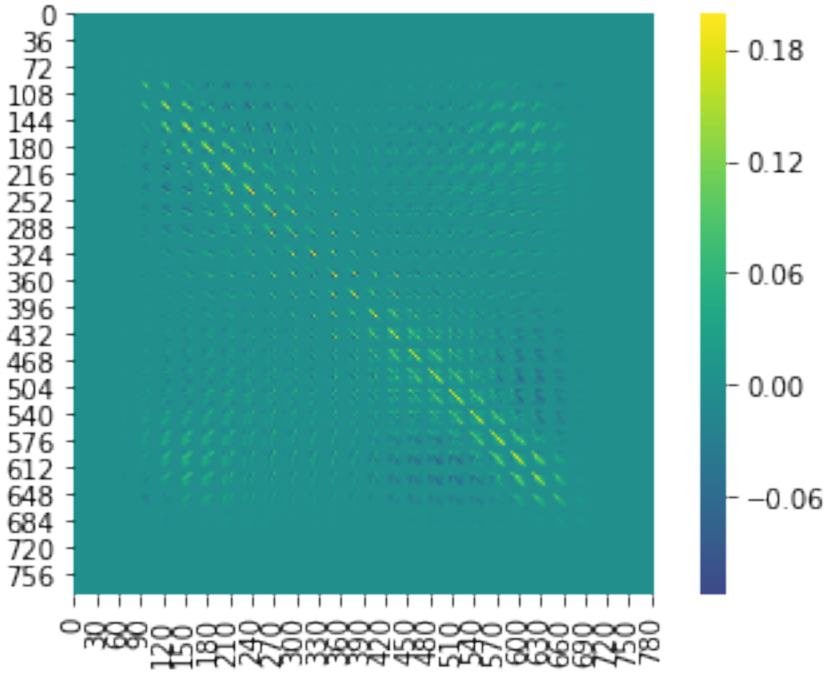
In [20]:
```
plt.imshow(Mu[6].reshape(28,28))
```

Out[20]: <matplotlib.image.AxesImage at 0x1a31119128>



In [21]:
```
# (b)
sns.heatmap(Sigma[2],square=True,center=0,cmap="viridis")
```

Out[21]: <matplotlib.axes._subplots.AxesSubplot at 0x1a31140a58>



```
In [32]: # (c)
         mnistData = processData("mnist",10000)
         def trainSplit(data,size):
             shuffle = np.random.permutation(data["X_train"].shape[0])
             data_x = data["X_train"][shuffle]
             data_y = data["y_train"][shuffle]
             train_X = data_x[:size,:]
             train_y = data_y[:size,:]
             return train_X, train_y

         def predictLDA(samples, priors, means, covar):
             multiGaussians = \
             [multivariate_normal(mean, covar, allow_singular=True) \
              for mean in means]
             pdfs = np.array([gauss.logpdf(samples) for gauss in multiGaussians]).T
             posteriors = pdfs + np.log(priors)
             predictions = posteriors.argmax(axis=1)
             return predictions

         def predictQDA(samples, priors, means, covars):
             multiGaussians = \
             [multivariate_normal(means[i], covars[i], allow_singular=True) \
              for i in range(len(means))]
             pdfs = np.array([gauss.logpdf(samples) for gauss in multiGaussians]).T
             posteriors = pdfs + np.log(priors)
             predictions = posteriors.argmax(axis=1)
             return predictions

         # Usage results_to_csv(clf.predict(X_test))
         def results_to_csv(y_test,dataname):
             y_test = y_test.astype(int)
             df = pd.DataFrame({'Category': y_test})
             df.index += 1  # Ensures that the index starts at 1.
             df.to_csv(dataname+'submission.csv', index_label='Id')

         def fitLDA(train_X,train_y,test,data="mnist"):
             Mu = []
             Index = []
             Sigma = []

             if data=="mnist":
                 c = 10
             else:
                 c = 2

             for i in range(c): #0-9 digits
```

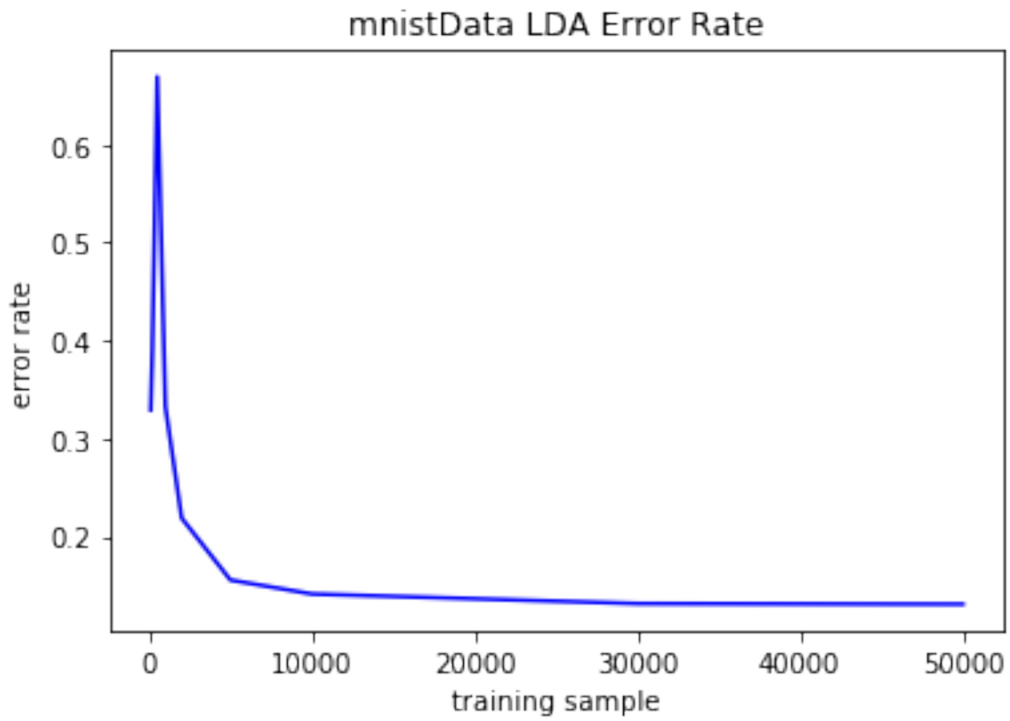
```
            index = np.where(train_y == i)[0]
            mu = np.mean(train_X[index], axis=0)
            sigma = np.cov(train_X[index],rowvar=False, bias=True)

            Index.append(index)
            Mu.append(mu)
            Sigma.append(sigma)

        prior = [index.size/sample for index in Index]

        SigmaLDA = sum(Sigma)/10.0

        pred = predictLDA(test,prior, Mu, SigmaLDA)
        return pred

    def fitQDA(train_X,train_y,test,data="mnist"):
        Mu = []
        Index = []
        Sigma = []

        if data=="mnist":
            c = 10
        else:
            c = 2

        for i in range(c): #0-9 digits
            index = np.where(train_y == i)[0]
            mu = np.mean(train_X[index], axis=0)
            sigma = np.cov(train_X[index],rowvar=False, bias=True)

            Index.append(index)
            Mu.append(mu)
            Sigma.append(sigma)


        prior = [index.size/sample for index in Index]

        pred = predictQDA(test,prior, Mu, Sigma)
        return pred
```

```
In [33]: errorLDA = []
         training_sample = [100, 200, 500, 1000, 2000, 5000, 10000, 30000, 50000]
         for sample in training_sample:
             train_X, train_y = trainSplit(mnistData,sample)

             pred = fitLDA(train_X,train_y,mnistData["X_validate"]) \
                     .reshape(*mnistData["y_validate"].shape)
             accuracy = sum(pred==mnistData["y_validate"])[0]/pred.size
             print(sample, ": ",1-accuracy)
```

```
        errorLDA.append(1-accuracy)
```

```
100 :  0.3275
200 :  0.3094
500 :  0.6796
1000 :  0.3517
2000 :  0.22119999999999995
5000 :  0.1603
10000 :  0.14629999999999999
30000 :  0.13280000000000003
50000 :  0.1311
```

In [24]:
```
plt.plot(training_sample,errorLDA,c="blue")

# plt.legend(loc="lower right")
plt.xlabel("training sample")
plt.ylabel("error rate")
plt.title("mnistData LDA Error Rate")

plt.show()
```

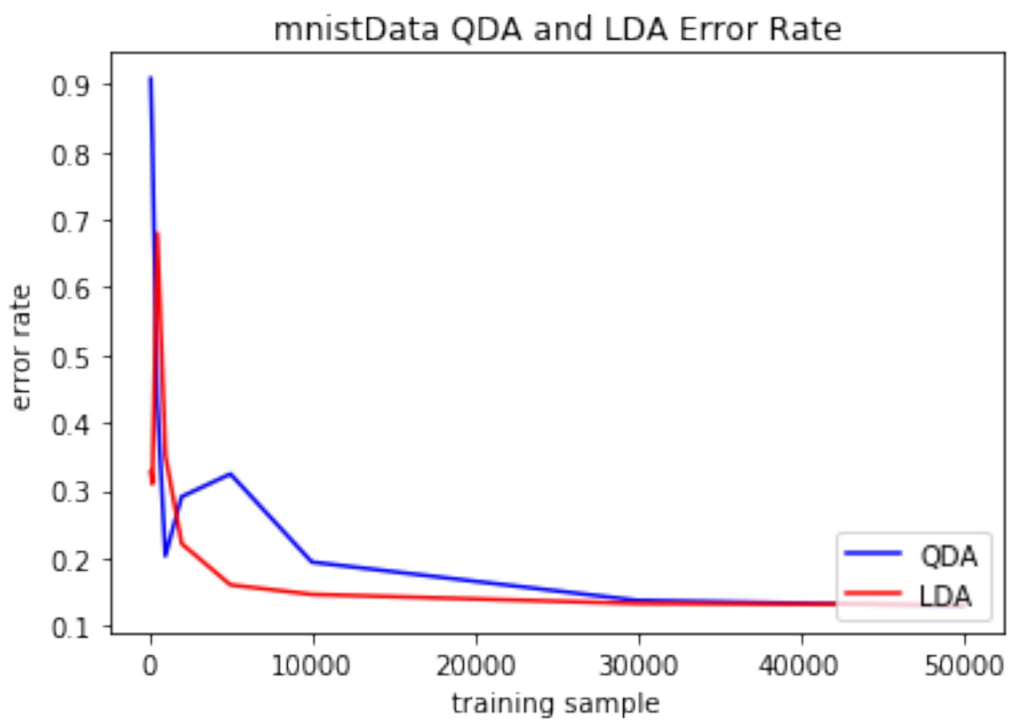


In [25]:
```
errorQDA = []
training_sample = [100, 200, 500, 1000, 2000, 5000, 10000, 30000, 50000]
for sample in training_sample:
    train_X, train_y = trainSplit(mnistData,sample)
```

```
        pred = fitQDA(train_X,train_y,mnistData["X_validate"]) \
                .reshape(*mnistData["y_validate"].shape)
        accuracy = sum(pred==mnistData["y_validate"])[0]/pred.size
        print(sample, ": ",1-accuracy)
        errorQDA.append(1-accuracy)
```

```
100 :  0.9082
200 :  0.8173
500 :  0.4324
1000 :  0.20309999999999995
2000 :  0.29059999999999997
5000 :  0.32420000000000004
10000 :  0.19399999999999995
30000 :  0.13739999999999997
50000 :  0.12890000000000001
```

In [66]:
```
plt.plot(training_sample,errorQDA,c="blue",label="QDA")
plt.plot(training_sample,errorLDA,c="red",label="LDA")
plt.legend(loc="lower right")
plt.xlabel("training sample")
plt.ylabel("error rate")
plt.title("mnistData QDA and LDA Error Rate")

plt.show()
```



```
In [61]: train_X, train_y = trainSplit(mnistData,50000)
         predLDA = fitLDA(train_X,train_y,mnistData["X_validate"])
         errLDA = []
         predQDA = fitQDA(train_X,train_y,mnistData["X_validate"])
         errQDA = []
         for i in range(10):
             index = np.where(mnistData["y_validate"] == i)[0]
             predLDA_index = predLDA[index] \
                             .reshape(*mnistData["y_validate"][index].shape)
             predQDA_index = predQDA[index] \
                             .reshape(*mnistData["y_validate"][index].shape)
             result = mnistData["y_validate"][index]

             errLDA.append(1-sum(predLDA_index==result)[0]/len(predLDA_index))
             errQDA.append(1-sum(predQDA_index==result)[0]/len(predQDA_index))
```

```
In [65]: plt.plot(errLDA,c="red",label="LDA")
         plt.plot(errQDA,c="blue",label="QDA")
         plt.legend(loc="lower right")
         plt.xlabel("digit class")
         plt.ylabel("error rate")
         plt.title("validation error for each digit class")
```

```
Out[65]: Text(0.5, 1.0, 'validation error for each digit class')
```

In [27]:
```
mnistData = processData("mnist",0)
train_X, train_y = trainSplit(mnistData,60000)


pred = fitLDA(train_X,train_y,mnistData["test"])
results_to_csv(pred,"mnistLDA")

pred = fitQDA(train_X,train_y,mnistData["test"])
results_to_csv(pred,"mnistQDA")
```

In [28]:
```
spamData = processData("spam",0)
train_X, train_y = trainSplit(spamData,5172)
pred = fitLDA(train_X,train_y,spamData["test"],data="spam")
results_to_csv(pred,"spamLDA")
pred = fitQDA(train_X,train_y,spamData["test"],data="spam")
results_to_csv(pred,"spamQDA")
```

In [ ]: