STAT 24610 Project

Lijing Wang

LDA

(a) The Bayes' classifier minimizes the probability of misclassification by assigning x to class k, which corresponds to the largest posterior probability.

$$h(x) = \underset{y}{\operatorname{argmax}}\, p(y|x)$$

Since $p(y|x) = \frac{p(x,y)}{p(x)}$, we have:

$$\begin{aligned}
h(x) =& \underset{y}{\operatorname{argmax}}\, p(y|x) = \underset{y}{\operatorname{argmax}} \ln(p(y|x)) \\
p(y=k|x) =& \frac{p(x, y=k)}{p(x)} = \frac{p(x|y=k)p(y=k)}{\sum_y p(x|y)p(y)} = \frac{\pi_k N(\mu_k, \Sigma)}{\sum_l \pi_l N(\mu_l, \Sigma)} \\
\ln(p(y=k|x)) =& \ln(\pi_k N(\mu_k, \Sigma)) - \ln(\sum_l \pi_l N(\mu_l, \Sigma)) \\
=& \ln(\pi_k) - \frac{1}{2}(x-\mu_k)^\intercal \Sigma^{-1}(x-\mu_k) - \ln(\sum_l \pi_l N(\mu_l, \Sigma)) + const \\
=& \ln(\pi_k) - \frac{1}{2}x^\intercal \Sigma^{-1} x + \mu_k^\intercal \Sigma^{-1} x - \frac{1}{2}\mu_k^\intercal \Sigma^{-1}\mu_k + const \\
=& \ln(\pi_k) - \frac{1}{2}\mu_k^\intercal \Sigma^{-1}\mu_k + \mu_k^\intercal \Sigma^{-1} x + const \\
\Rightarrow \quad h(x) =& \underset{k}{\operatorname{argmax}}\{\ln(\pi_k) - \frac{1}{2}\mu_k^\intercal \Sigma^{-1}\mu_k + \mu_k^\intercal \Sigma^{-1} x\}
\end{aligned}$$

Thus, $w_k = \mu_k^\intercal \Sigma^{-1}$, $w_{0k} = \ln(\pi_k) - \frac{1}{2}\mu_k^\intercal \Sigma^{-1}\mu_k$.

(b) Denote the log-likelihood as l

$$\begin{aligned}
l =& \sum_{i=1}^{n}(-\frac{1}{2}\log|\Sigma| - \frac{1}{2}(x_i-\mu_{ik})^\intercal \Sigma^{-1}(x_i-\mu_{ik})) + const \\
=& \sum_{k=1}^{K}\sum_{i=1}^{n_k}(-\frac{1}{2}\log|\Sigma| - \frac{1}{2}(x_i-\mu_k)^\intercal \Sigma^{-1}(x_i-\mu_k)) + const \\
\frac{\partial l}{\partial \mu_k} =& \sum_{i=1}^{n_k}\Sigma^{-1}(x_i-\mu_k) = n_k\Sigma^{-1}(\bar{x}-\mu_k) \\
\Rightarrow \quad \mu_k =& \frac{1}{n_k}\sum_{i=1}^{n_k} x_i = \frac{1}{n_k}\sum_{y_i=k} x_i \\
\frac{\partial l}{\partial \Sigma} =& \frac{1}{2}\Sigma^{-1}(\sum_{i=1}^{n}(x_i-\hat{\mu}_k)(x_i-\hat{\mu}_k)^\intercal)\Sigma^{-1} - \frac{n}{2}\Sigma^{-1} \\
\Rightarrow \quad \Sigma =& \frac{1}{n}\sum_{i=1}^{n}(x_i-\hat{\mu}_k)(x_i-\hat{\mu}_k)^\intercal \\
=& \frac{1}{n}\sum_{k=1}^{K}\sum_{i=1}^{n_k}(x_i-\bar{x}_k)(x_i-\bar{x}_k)^\intercal
\end{aligned}$$

(c) $\hat{\Sigma} = \frac{1}{n}\sum_{k=1}^{K}\sum_{i=1}^{n_k}(x_i - \bar{x}_k)(x_i - \bar{x}_k)^\intercal$. $\hat{\Sigma}$ is a positive semi-definite matrix. I_p is a positive definite matrix. Therefore, for some small $\lambda \in (0,1)$, $(1-\lambda)\hat{\Sigma}+(\lambda/4)I_p$ is a postive definite matrix. Therefore, 0 is not an eigenvalue of $(1-\lambda)\hat{\Sigma} + (\lambda/4)I_p$. $((1-\lambda)\hat{\Sigma} + (\lambda/4)I_p)x = 0$ has no non-trivial solution.

$\therefore (1-\lambda)\hat{\Sigma} + (\lambda/4)I_p$ is invertible.

We can write $\hat{\Sigma}_\lambda$ in a more explicit way as $(1-\lambda)\hat{\Sigma}+\lambda(\frac{1}{4}I_p)$. If the 20×20-pixel image doesn't have any pattern, then each pixel can be viewed as independent and follow Bernoulli distribution with mean parameter $\mu = \frac{1}{2}$.

If x_i and x_j are independent, $x_i \sim B(\mu_i)$, $x_j \sim B(\mu_j)$, then $Cov(x_i, x_j) = 0$ for $i \neq j$, $Var(x_i) = \mu_i(1-\mu_i) \leq \frac{1}{4}$. Thereforem $\frac{1}{4}I_p$ can be viewed as the covariance matrix of $X_i = (x_{i1}, \cdots, x_{ip})$, where x_{ij} is of independent $B(\frac{1}{2})$ distribution.

If $\hat{\Sigma}$ is singular, consider introducing covariance structure from the invertible covariance matrix $\frac{1}{4}I_p$. Use a 'weighted' covariance matrix $\hat{\Sigma}_\lambda$ instead, where $\hat{\Sigma}$ is the covariance matrix of the original data and $\frac{1}{4}I_p$ is the covariance matrix of $B(\frac{1}{2})$. The weights sum up to 1

R Code:

The R code of LDA is given as below:

```
###initialize the sample covariance matrix
S<-array(0,dim=c(400,400))
###initialize vector
rtr<-array(NA,dim=c(10,500,400))
###convert training data from 20x20 matrix to 400x1 vector
for(d in 1:10)
{
        for(n in 1:500)
        {
                rtr[d,n,]<-as.vector(training.data[d,n,,])
        }
}
###calculate sample mean of each digit class
dm<-apply(training.data[,index,,],c(3,4,1),mean)
###calculate sample covariance matrix for the data
for(d in 1:10)
{
        mu<-as.vector(dm[,,d])
        for(n in 1:500)
        {
                S<-S+(rtr[d,n,]-mu)%*%t(rtr[d,n,]-mu)
        }
}
S<-S/5000
### set value for lambda
lambda<-0.12
### calculate the revised sample covariance matrix
Ss<-(1-lambda)*S+(lambda/4)*diag(400)
### calculate the inverse of the matrix
```

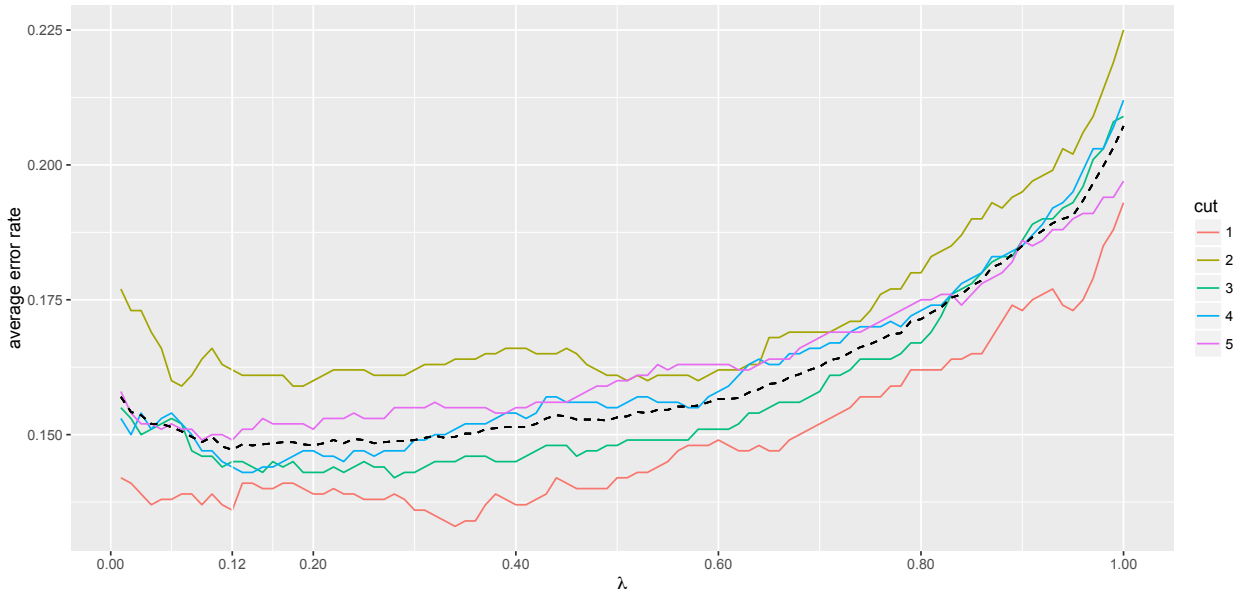
```
S_<-solve(Ss)


### test on testing data set
count<-0
for(d in 1:10)
{
        count<-0
        for(n in 1:1000)
        {
                x<-as.vector(test.data[d,n,,])
                y<-c()
                for(t in 1:10)
                {
                        m<-as.vector(dm[,,t])
                        ###calculate
                        y<-c(y,t(m)%*%S_%*%x-(1/2)*t(m)%*%S_%*%m)
                }
                ###assign x to k corresponding to the largest y
                ###if k doesn't match with its orginal class
                ###x is misclassified
                if(which.max(y)!=d)
                {
                        count<-count+1
                }
        }
        ### print the error rate
        print(count/1000)
}
```

Here I choose $\lambda = 0.12$ when inverting the sample covariance matrix.

Tuning Parameter Selection:

Use K-fold cross validation to select the appropriate λ. Here I cut the training data into 5 parts, each time select a λ and train the classifier using 4 parts of the training data and calculate the error rate on the remaining part. Choose the λ corresponding to the least average test error rate. The plot of cross validation error against λ is given as below: (dashed line representing average error rate over 5 parts)



When $\lambda = 0.01$, the error rate is 0.157. As λ increases, the average test error rate decreases. It reaches its bottom(0.1472) when $\lambda = 0.12$ and then starts to increase.

The table below shows the change of test error rate as λ increases.

λ	error rate 1	error rate 2	error rate 3	error rate 4	error rate 5	average
0.01	0.142	0.177	0.155	0.153	0.158	0.157
0.05	0.138	0.166	0.152	0.153	0.151	0.152
0.10	0.139	0.166	0.146	0.147	0.150	0.1496
0.12	0.136	0.162	0.145	0.144	0.149	0.1472
0.15	0.140	0.161	0.143	0.144	0.153	0.1482
0.20	0.139	0.160	0.143	0.147	0.151	0.148
0.50	0.142	0.161	0.148	0.155	0.160	0.153
0.75	0.157	0.173	0.164	0.170	0.170	0.1668
1.00	0.193	0.225	0.209	0.212	0.197	0.2072

Final Classifier:

Here we assume π_k's are the same for $k = 0, 1, \cdots, 9$. Therefore ignore the $\ln(\pi_k)$ in w_{0k}. The final classifier is $h(x) = \text{argmax}_k\{-\frac{1}{2}\hat{\mu}_k^\intercal \hat{\Sigma}_\lambda^{-1}\hat{\mu}_k + \hat{\mu}_k^\intercal \hat{\Sigma}_\lambda^{-1}x\}$, where $\hat{\Sigma}_\lambda = 0.88\hat{\Sigma} + 0.03I_p$

Error Rate on Test Set:

Test the classifier on 1000 samples of each digit.

The misclassification error rates of the 10 digits are given as below

0	1	2	3	4	5	6	7	8	9	average
0.101	0.044	0.196	0.168	0.125	0.170	0.103	0.138	0.235	0.172	0.1452

Mixture of Independent Bernoulli

(a) Consider maximizing $Q(\theta,\theta^{old})+\ln(p(\theta))$ instead of $Q(\theta,\theta^{old})$. Since $\mu_{mj}\sim Beta(2,2)$, $p(\mu_{mj})=c_1\mu_{mj}(1-\mu_{mj})$, $(\pi_1,\cdots,\pi_M)\sim Dirichlet(2,\cdots,2)$, $p(\pi_1,\cdots,\pi_M)=c_2\prod_{m=1}^{M}\pi_m$, where c_1,c_2 are constants.

$$\begin{aligned}\ln p(x,z)&=\sum_{i=1}^{n}\sum_{m=1}^{M}(z_{im}(\ln\pi_m+\sum_{j=1}^{D}(x_{ij}\ln\mu_{mj}+(1-x_{ij})\ln(1-\mu_{mj}))))\\ Q'(\theta,\theta^{old})&=\int p(z|x,\theta_{old})\ln(p(x,z|\theta))dz+\ln(p(\theta))+const\\ &=\sum_{i=1}^{n}\sum_{m=1}^{M}(\gamma(z_{im})(\ln\pi_m+\sum_{j=1}^{D}(x_{ij}\ln\mu_{mj}+(1-x_{ij})\ln(1-\mu_{mj}))))\\ &\quad+\sum_{m=1}^{M}\sum_{j=1}^{D}(\ln\mu_{mj}+\ln(1-\mu_{mj}))+\sum_{i=1}^{M}\ln\pi_m+const\end{aligned}$$

$$\gamma(z_{im})=p(z_i=m|x_i)=\frac{p(z_i=m,x_i)}{p(x_i)}=\frac{\pi_m p(x_i|z_{im}=1)}{\sum_{i=1}^{k}\pi_k p(x_i|z_{ik}=1)}$$

Need to maximize $Q'(\theta,\theta^{old})$ subject to $\sum_{m=1}^{M}\pi_m=1$.

Use Lagrange multiplier to find the θ's that maximize L.

$$\begin{cases}L=\sum_{m=1}^{M}(\sum_{i=1}^{n}\gamma(z_{im})+1)\ln\pi_m+\sum_{m=1}^{M}\sum_{j=1}^{D}((\sum_{i=1}^{n}\gamma(z_{im})x_{ij}+1)\ln\mu_{mj}+(\sum_{i=1}^{n}\gamma(z_{im})(1-x_{ij})+1)\ln(1-\mu_{mj}))\\ \qquad+\lambda(\sum_{m=1}^{M}\pi_m-1)\\ \dfrac{\partial L}{\pi_m}=0,\quad \dfrac{\partial L}{\mu_{mj}}=0,\quad \dfrac{\partial L}{\lambda}=0\end{cases}$$

$$\Rightarrow\hat{\pi}_m=\frac{\sum_{i=1}^{n}\gamma(z_{im})+1}{\sum_{m=1}^{M}(\sum_{i=1}^{n}\gamma(z_{im})+1)}=\frac{\sum_{i=1}^{n}\gamma(z_{im})+1}{M+n},\qquad\hat{\mu}_{mj}=\frac{\sum_{i=1}^{n}\gamma(z_{im})x_{ij}+1}{\sum_{i=1}^{n}\gamma(z_{im})+2}$$

(b) The log-likelihood for complete data set is given as below:

$$\ln p(x,z)=\sum_{i=1}^{n}\sum_{m=1}^{M}(z_{im}(\ln\pi_m+\sum_{j=1}^{D}(x_{ij}\ln\mu_{mj}+(1-x_{ij})\ln(1-\mu_{mj}))))$$

We need to maximize the log-likelihood function subject to $\sum_{m=1}^{M}\pi_m=1$.

$$\Rightarrow\hat{\pi}_m=\frac{\sum_{i=1}^{n}z_{im}}{n},\qquad\hat{\mu}_{mj}=\frac{\sum_{i=1}^{n}z_{im}x_{ij}}{\sum_{i=1}^{n}z_{im}}$$

To initialize the data, pick up M for the model. Assign each example at random to one of the M components with equal probability. After the assignment, use the formulae above to calculate $\hat{\pi}_k^{(0)}$ and $\hat{\mu}_{kj}^0$. In each round of iteration, update $\gamma(z_{im})$, $\hat{\pi}_k$ and $\hat{\mu}_{kj}$ using the formulae derived in part (a) and stop the updating process until $\hat{\pi}_k$ and $\hat{\mu}_{kj}$ converge.

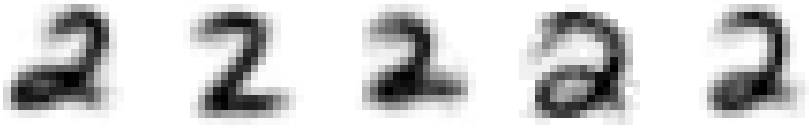
(c) Since the dimension of sample is relatively high, if we directly use $p(x_i|\mu_m)$ in the formulae, due to the relatively small absolute value of $p(x_i|\mu_m)$, the result may be inaccurate because of rounding error. Therefore it is necessary to rescale the probability to an appropriate scale. Therefore, instead of directly using $p(x_i|\mu_m)$ in the calculation, divide the numerator and denominator simultaneously by $\exp(\ell^*)$, then use the rescaled probability to update $\gamma(z_{im})$

(d) Choose digit 2 and set M=2,3,5. The M components are given as below:

M=2

M=3

M=5

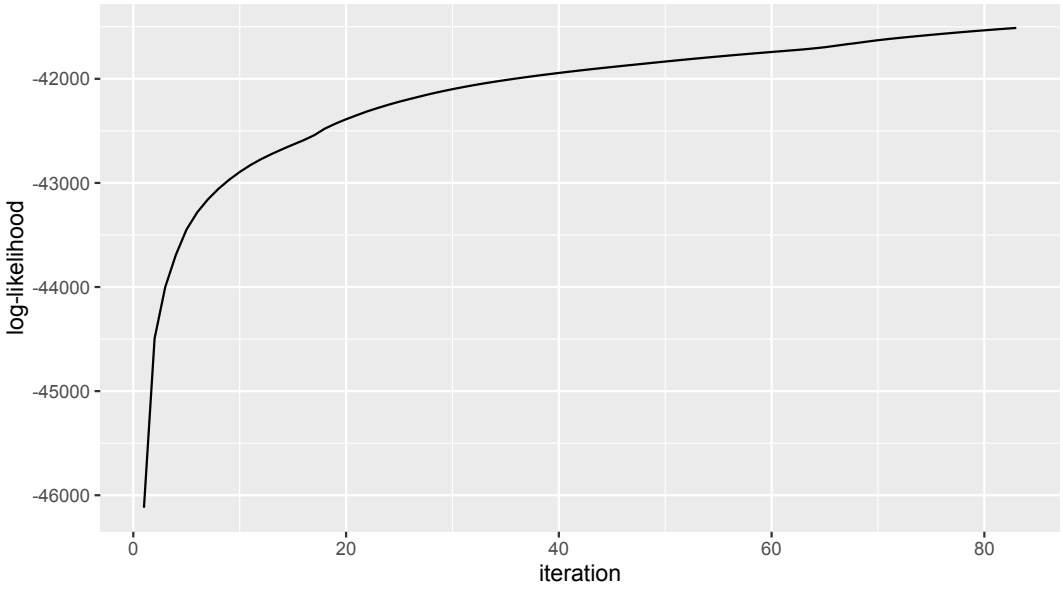


Within each mixture of bernoulli model, there're M components, each representing one kind of handwritten digit. Since we have different handwriting style, the shape of the handwritten number might vary within each digit class. Therefore, setting different components within each digit class can help better classify the data. As we can see from the images above, when M=2, the lower part of the first component is more curved than that of the second component.

As M increases, the classification within each digit class will be more and more sophisticated.

When implementing the EM algorithms, I use the change of coefficients between two iterations to set the terminating condition, namely $\left\|\theta^{old} - \theta^{new}\right\|^2$. As iteration time increases, the coefficients tend to converge. Stop the iteration when $\left\|\theta^{old} - \theta^{new}\right\|^2$ is small enough.

Here I set the terminating condition to $\left\|\pi^{new} - \pi^{old}\right\|^2 + \left\|\mu^{new} - \mu^{old}\right\|^2 \leq 10^{-6}$. For M=5, the iteration process will stop after 83 iterations. The log-likelihood and coefficient change are listed as below .



iteration	log-likelihood	$\left\|\theta^{new} - \theta^{old}\right\|^2$
1	-46120.21	7.612566
2	-44494.72	3.419348
5	-43447.98	0.179942
10	-42895.33	0.011184
20	-42388.57	0.011865
30	-42099.68	0.003086
50	-41834.12	0.000433
80	-41534.05	9.05×10^{-6}
81	-41526.20	2.91×10^{-6}
82	-41518.62	1.38×10^{-6}
83	-41511.29	7.90×10^{-7}

(e) I choose digit `3` and `5` for this part.

Here I set M= 3 for both classes. Fit a mixture of Bernoulli model and train the data to get the estimates of π_m and μ_m for each class.

Given a new data point, calculate the log-likelihood using both models and assign it to the class corresponding to the largest log-likelihood.

$$\ln p_k(x_i) = \log(\sum_{m=1}^{M}(\prod_{j=1}^{D}\mu_{km}^{x_{ij}}(1-\mu_{km})^{1-x_{ij}})\pi_{km})$$

$$h(x) = \underset{k}{\operatorname{argmax}} \ln p_k(x_i)$$

The classifier performs quite well. The test error of assigning `3` to `5` is 0.067, and assigning `3` to `5` 0.102.

I also tried some other combinations, the results are listed as below:

class		error rate		
A	B	classify A to B	classify B to A	average
3	5	0.067	0.102	0.0845
2	5	0.033	0.011	0.0220
1	7	0.009	0.026	0.0175
6	8	0.017	0.018	0.0175
6	9	0.003	0.012	0.0075
0	9	0.014	0.017	0.0155
4	5	0.014	0.033	0.0235

The R code for EM algorithm and classification is given as below:

```
##########EM ALGORITHM######
Class<-4
set.seed(123)
###set the number of components in each digit class
M<-3
n<-500
### assign each example at random to one of the M components
ini_index<-sample(1:M,n,replace=TRUE)
z<-diag(M)[ini_index,]
mu<-matrix(0,nrow=M,ncol=400)
### estimate the prior probability
pi_<-colMeans(z)
gamma<-matrix(0,nrow=n,ncol=M)
lnp<-matrix(0,nrow=n,ncol=M)
alnp<-matrix(0,nrow=n,ncol=1)
ite<-1
for(m in 1:M)
{
        mu[m,]<-as.vector(apply(training.data[Class,z[,m]==1,,],c(2,3),mean))
}
while(TRUE)
{
        print(ite)
        ### store the old value of mu
        mu_old<-mu
        ### store the old value of pi
        pi_old<-pi_
        ####UPDATING GAMMA####
        for(i in 1:n)
        {
                for(m in 1:M)
                {
x<-as.vector(training.data[Class,i,,])
### calculate l_m(X_i) for each m and i
lnp[i,m]<-sum((log(mu[m,])*x+log(1-mu[m,])*(1-x)),na.rm=TRUE)+log(pi_[m])
                }
        }
        ### find l^*
```

```
        l_star<-apply(lnp,1,max)
        for(i in 1:n)
        {
                for(m in 1:M)
                {
### update gamma(z_im) using the formular in part (c)
gamma[i,m]<-exp(lnp[i,m]-l_star[i])/sum(exp(lnp[i,]-l_star[i]))
                }
        }
####UPDATING MU and PI####
        for(m in 1:M)
        {
                u<-rep(1,400)
                g<-2
                for(i in 1:n)
                {
                        x<-as.vector(training.data[Class,i,,])
                        u<-u+t(gamma[i,m])*x
                        g<-g+gamma[i,m]
                }
                ### use updated gamma and
                ### the formular derived in part(a) to update mu
                mu[m,]<-u/g
        }
        ### use updated gamma and
        ### the formular derived in part(a) to update pi
        pi_<-(colSums(gamma)+1)/505

        for(i in 1:n)
        {
                x<-as.vector(training.data[Class,i,,])
                for(m in 1:M)
                {
### calculate the log-likelihood of each X_i
b<-sum(exp(sum(log(mu[m,])*x+log(1-mu[m,])*(1-x))+log(pi_[m])),na.rm=TRUE)
alnp[i]<-alnp[i]+b
                }
        }
        ### calculate the log-likelihood of training data
        L_new<-sum(log(alnp),na.rm=TRUE)
        ### calculate terminating criteria
        ctrl_<-sum((mu_old-mu)^2)+sum((pi_old-pi_)^2)
        print(ctrl_)
        ### add one to iteration time
        ite<-ite+1
        print(L_new)
        ### if terminating condition is satisfied
        ### stop the iteration process
        if(ctrl_<10^(-6))
                break
}
```

```
### store the coefficients estimates
mu3<-mu
pi3<-pi_

### plot the mean of each components
for(m in 1:M)
{
xx<-t(1-matrix(mu[m,],nrow=20))[,20:1]
image(xx,col=gray(seq(0,1,length.out=256)),axes=FALSE,asp=1)
}

### use the same code as above to calculate
### mu0, mu1,...,mu9 and pi0,pi1,...,pi9

######COMPARE LOG-LIKELIHOOD#####
###### assign x to the class with the largest log-likelihood
for(Class in c(4,6))
{
        ### count the examples that are misclassified
        count<-0
        for(i in 1:1000)
        {
                x<-as.vector(test.data[Class,i,,])
                lik3<-0
                ###calculate the log-likelihood of digit 3
                for(m in 1:M)
                {
lik<-sum(exp(sum(log(mu3[m,])*x+log(1-mu3[m,])*(1-x))+log(pi3[m])),na.rm=TRUE)
lik3<-lik3+lik
                }
                lik5<-0
                ###calculate the log-likelihood of digit 5
                for(m in 1:M)
                {
lik<-sum(exp(sum(log(mu5[m,])*x+log(1-mu5[m,])*(1-x))+log(pi5[m])),na.rm=TRUE)
lik5<-lik5+lik
                }
                ### assign x to the class with the largest log-likelihood
                if(lik5<lik3)
                        {y<-4}
                else
                        {y<-6}
                ### if \hat{y} doesn't match with its original class
                ### x is misclassified
                ### add 1 to the misclassification count
                if(y!=Class)
                        {count<-count+1}
        }
        print(count/1000)
}
```