STAT 542: Homework 5

Spring 2020, by Yifan Shi (yifans16)

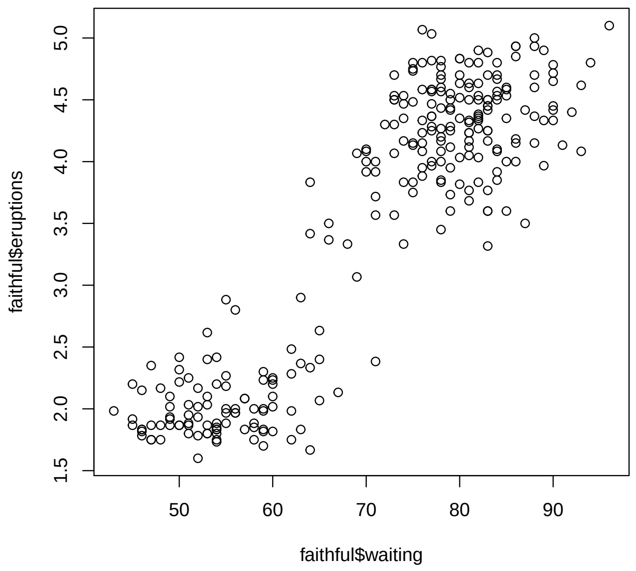
Due: Monday, Apr 13 by 11:59 PM

Contents

Question 1 [40 Points] Two-dimensional Gaussian Mixture Model

If you do not use latex to type your answer, you will lose 2 points. We consider another example of the EM algorithm, which fits a Gaussian mixture model to the Old Faithful eruption data. The data is used in HW4. For a demonstration of this problem, see the figure provided on Wikipedia. As a result, we will use the formula to implement the EM algorithm and obtain the distribution parameters of the two underlying Gaussian distributions. Here is a visualization of the data:

```r
# load the data
load("faithful.Rda")
plot(faithful$waiting, faithful$eruptions)
```



We use both variables `eruptions` and `waiting`. The plot above shows that there are two eruption patterns (clusters). Hence, we use a hidden Bernoulli variable $Z_i \sim \text{Bern}(\pi)$, that indicates the pattern when we wait for the next eruption. The corresponding distribution of `eruptions` and `waiting` is a two-dimensional Gaussian — either $N(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)$ or $N(\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2)$ — depending on the outcome of Z_i. Here, of course, we do not know the parameters $\boldsymbol{\theta} = \{\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1, \boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2, \pi\}$, and we want to use the observed data to estimate them.

- [5 points] Based on the above assumption of the description, write down the full log-likelihood $\ell(\mathbf{x}, \mathbf{z}|\boldsymbol{\theta})$. Then, following the strategy of the EM algorithm, in the E-step, we need the conditional expectation

$$g(\boldsymbol{\theta}|\boldsymbol{\theta}^{(k)}) = E_{\mathbf{Z}|\mathbf{x},\boldsymbol{\theta}^{(k)}}[\ell(\mathbf{x}, \mathbf{z}|\boldsymbol{\theta})].$$

The answer is already provided on Wikipedia page. Write down the conditional distribution of $\mathbf{Z}$ given $\mathbf{x}$ and $\boldsymbol{\theta}^{(k)}$ with notations used in our lectures.

Let $\pi_1 = \pi$ and $\pi_2 = 1 - \pi$

$$\begin{aligned}
\ell(x, z|\theta) &= log(\prod_{i=1}^{n}\prod_{j=1}^{2}[f(x_i;\mu_j,\Sigma_j)\pi_j]^{\mathbb{I}(z_i=j)}) \\
&= log(\prod_{i=1}^{n}\prod_{j=1}^{2}[(\pi_j)(2\pi)^{-\frac{2}{2}}|\Sigma_j|^{-\frac{1}{2}}e^{-\frac{1}{2}(x_i-\mu_j)^T\Sigma_j^{-1}(x_i-\mu_j)}]^{\mathbb{I}(z_i=j)}) \\
&= \sum_{i=1}^{n}\sum_{j=1}^{2}\mathbb{I}(z_i=j)[log\pi_j - log(2\pi) - \frac{1}{2}log(|\Sigma_j|) - \frac{1}{2}(x_i-\mu_j)^T\Sigma_j^{-1}(x_i-\mu_j)]
\end{aligned}$$

$$\begin{aligned}
g(\theta|\theta^{(k)}) &= E_{Z|x,\theta^{(k)}}[\ell(x,z|\theta)] \\
&= \sum_{i=1}^{n}\sum_{j=1}^{2}\frac{P(z_i=j,x_i|\theta^{(k)})}{P(z_i=1,x_i|\theta^{(k)})+P(z_i=2,x_i|\theta^{(k)})}\ell(x_i,z_i=j|\theta) \\
&= \sum_{i=1}^{n}\sum_{j=1}^{2}\frac{\pi_j^{(k)}f(x_i;\mu_j^{(k)},\Sigma_j^{(k)})}{\pi_1^{(k)}f(x_i;\mu_1^{(k)},\Sigma_1^{(k)})+\pi_2^{(k)}f(x_i;\mu_2^{(k)},\Sigma_2^{(k)})}[log\pi_j - log(2\pi) - \frac{1}{2}log(|\Sigma_j|) - \frac{1}{2}(x_i-\mu_j)^T\Sigma_j^{-1}(x_i-\mu_j)]
\end{aligned}$$

- [10 points] Once we have the $g(\boldsymbol{\theta}|\boldsymbol{\theta}^{(k)})$, the M-step is to re-calculate the maximum likelihood estimators of $\boldsymbol{\mu}_1$, $\boldsymbol{\Sigma}_1$, $\boldsymbol{\mu}_2$, $\boldsymbol{\Sigma}_2$ and π. Again the answer was already provided. However, you need to provide a derivation of these estimators. **Hint**: by taking the derivative of the objective function, the proof involves three tricks:
  - $\text{Trace}(\beta^T\Sigma^{-1}\beta) = \text{Trace}(\Sigma^{-1}\beta\beta^T)$
  - $\frac{\partial}{\partial A}\log|A| = A^{-1}$
  - $\frac{\partial}{\partial A}\text{Trace}(BA) = B^T$

Let

$$\begin{aligned}
\hat{p_i} &= P(z_i=1|x=x_i,\theta^{(k)}) \\
&= \frac{P(z_i=1,x_i|\theta^{(k)})}{P(z_i=1,x_i|\theta^{(k)})+P(z_i=2,x_i|\theta^{(k)})} \\
&= \frac{\pi_1^{(k)}f(x_i;\mu_1^{(k)},\Sigma_1^{(k)})}{\pi_1^{(k)}f(x_i;\mu_1^{(k)},\Sigma_1^{(k)})+\pi_2^{(k)}f(x_i;\mu_2^{(k)},\Sigma_2^{(k)})}
\end{aligned}$$

Then

$$\begin{aligned}
\pi^{(k+1)} &= \underset{\pi}{argmax}\ g(\theta|\theta^{(k)}) \\
&= \underset{\pi}{argmax}\ \sum_{i=1}^{n}[\hat{p_i}log(\pi_1) + (1-\hat{p_i})log(\pi_2)] \\
\frac{\partial g}{\partial \pi_1} &= \frac{\sum_{i=1}^{n}\hat{p_i}}{\pi_1} - \frac{\sum_{i=1}^{n}(1-\hat{p_i})}{1-\pi_1} \\
&= \sum_{i=1}^{n}\hat{p_i}\cdot\frac{1-\pi_1}{\pi_1} - \sum_{i=1}^{n}(1-\hat{p_i}) \\
&= \sum_{i=1}^{n}\hat{p_i}\cdot\frac{1}{\pi_1} - \sum_{i=1}^{n}1 \\
&= \frac{\sum_{i=1}^{n}\hat{p_i}}{\pi_1} - n \\
&= 0
\end{aligned}$$

Thus $\hat{\pi}^{(k+1)} = \hat{\pi_1}^{(k+1)} = \frac{1}{n}\sum_{i=1}^{n}\hat{p_i}$

Also

$$\begin{aligned}
(\mu_1^{(k+1)}, \Sigma_1^{(k+1)}) &= \underset{\mu_1,\Sigma_1}{argmax}\ g(\theta|\theta^{(k)}) \\
&= \underset{\mu_1,\Sigma_1}{argmax}\ \sum_{i=1}^{n}\hat{p_i}[-\frac{1}{2}log(|\Sigma_1|) - \frac{1}{2}(x_i-\mu_1)^T\Sigma_1^{-1}(x_i-\mu_1)] \\
\frac{\partial g}{\partial \mu_1} &= \sum_{i=1}^{n}\hat{p_i}\Sigma_1^{-1}(x_i-\mu_1) \\
&= 0
\end{aligned}$$

Σ_1 is postive definite, its inverse is also postive definite

So $\sum_{i=1}^{n}\hat{p_i}(x_i-\mu_1) = 0$

Thus $\hat{\mu_1}^{(k+1)} = \frac{\sum_{i=1}^{n}\hat{p_i}x_i}{\sum_{i=1}^{n}\hat{p_i}}$

$$\begin{aligned}
\frac{\partial g}{\partial \Sigma_1^{-1}} &= \frac{\partial}{\partial \Sigma_1^{-1}}(\sum_{i=1}^{n}[\frac{1}{2}\hat{p_i}log(|\Sigma_1^{-1}|) - \frac{1}{2}\hat{p_i}\ trace((x_i-\mu_1)^T(x_i-\mu_1)\Sigma_1^{-1})]) \\
&= \sum_{i=1}^{n}\hat{p_i}\Sigma_1 - \sum_{i=1}^{n}\hat{p_i}(x_i-\mu_1)(x_i-\mu_1)^T \\
&= 0
\end{aligned}$$

Thus $\hat{\Sigma_1}^{(k+1)} = \frac{\sum_{i=1}^{n}\hat{p_i}(x_i-\hat{\mu_1}^{(k+1)})(x_i-\hat{\mu_1}^{(k+1)})^T}{\sum_{i=1}^{n}\hat{p_i}}$

And

$$\begin{aligned}
(\mu_2^{(k+1)}, \Sigma_2^{(k+1)}) &= \underset{\mu_2,\Sigma_2}{argmax}\ g(\theta|\theta^{(k)}) \\
&= \underset{\mu_2,\Sigma_2}{argmax}\ \sum_{i=1}^{n}(1-\hat{p_i})[-\frac{1}{2}log(|\Sigma_2|) - \frac{1}{2}(x_i-\mu_2)^T\Sigma_2^{-1}(x_i-\mu_2)]
\end{aligned}$$

Thus $\hat{\mu_2}^{(k+1)} = \frac{\sum_{i=1}^{n}(1-\hat{p_i})x_i}{\sum_{i=1}^{n}(1-\hat{p_i})}$ and $\hat{\Sigma_2}^{(k+1)} = \frac{\sum_{i=1}^{n}(1-\hat{p_i})(x_i-\hat{\mu_2}^{(k+1)})(x_i-\hat{\mu_2}^{(k+1)})^T}{\sum_{i=1}^{n}(1-\hat{p_i})}$

- [15 points] Implement the EM algorithm using the formulas you have. You need to give a reasonable initial value such that the algorithm will converge. Make sure that you properly document each step and report the final results (all parameter estimates). For this question, you may use other packages to calculate the Gaussian densities.

```
library(mvtnorm)
x = as.matrix(faithful)
# initialization of parameters
hat_pi = .5
hat_mu1 =c(4,80)
hat_mu2 = c(2, 50)
hat_sig1 = matrix(c(0.1, 0, 0, 30), 2, 2)
hat_sig2 = matrix(c(0.1, 0, 0, 30), 2, 2)
for (em in 1:10){
  # e step
  d1 = hat_pi*dmvnorm(x,hat_mu1,hat_sig1) #prior times prob density of 1st class
  d2 = (1-hat_pi)*dmvnorm(x,hat_mu2,hat_sig2) #prior times prob density of 2nd class
  hat_p = d1/(d1+d2) #conditional density of z
  # m step
  hat_pi = mean(hat_p) #update pi
  new_mu1 = colSums(hat_p*x) / sum(hat_p) #calculate new mu1
  new_mu2 = colSums((1-hat_p)*x) / sum(1-hat_p) #calculate new mu2
  new_sig1 = matrix(rep(0,4),2,2)
  new_sig2 = matrix(rep(0,4),2,2)
  mu1 = as.matrix(new_mu1)
  mu2 = as.matrix(new_mu2)
  # calculate new sig1 and sig2
  for (idx in 1:nrow(x)){
    xrow = as.matrix(x[idx,])
    new_sig1 = new_sig1 + hat_p[idx]*(xrow-mu1)%*%t(xrow-mu1) / sum(hat_p)
    new_sig2 = new_sig2 + (1-hat_p)[idx]*(xrow-mu2)%*%t(xrow-mu2) /sum(1-hat_p)
  }
  # check the convergence
  if (mean(abs(new_mu1-hat_mu1))<1e-5 & mean(abs(new_mu2-hat_mu2))<1e-5) {break}
  # update mu1 mu2 sig1 sig2
  hat_mu1 = new_mu1
  hat_mu2 = new_mu2
  hat_sig1 = new_sig1
  hat_sig2 = new_sig2
}
```

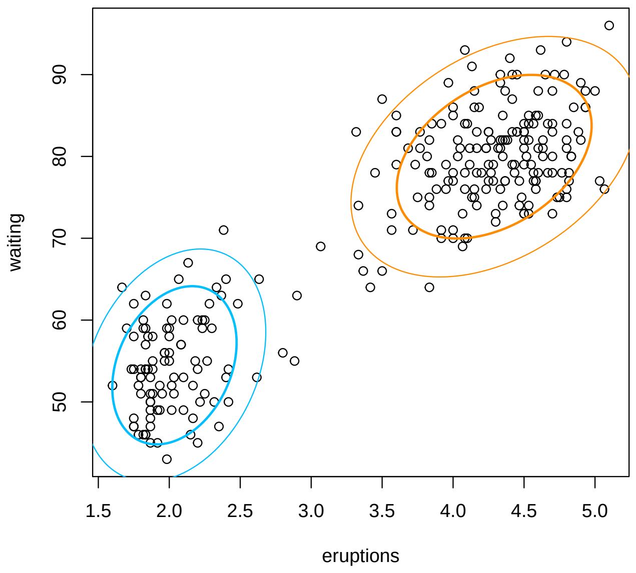
- [10 points] Plot your final results intuitively. You may borrow the idea at the Wikipedia page, or use the following code, which plots the contours of a Gaussian distribution.

```
# plot the current fit
library(mixtools)
plot(faithful)

addellipse <- function(mu, Sigma, ...)
{
  ellipse(mu, Sigma, alpha = .05, lwd = 1, ...)
  ellipse(mu, Sigma, alpha = .25, lwd = 2, ...)
}

addellipse(new_mu1, new_sig1, col = "darkorange")
```

```r
addellipse(new_mu2, new_sig2, col = "deepskyblue")
```



Question 2 [45 Points] Discriminant Analysis

For this question, you need to write your own code. We will use the handwritten digit recognition data again from the `ElemStatLearn` package. We only consider the train-test split, with the pre-defined `zip.train` and `zip.test`, and no cross-validation is needed. However, **use `zip.test` as the training data, and `zip.train` as the testing data!**

- [15 points] Write your own linear discriminate analysis (LDA) code following our lecture note. Use the training data to estimate all parameters and apply them to the testing data to evaluate the performance. Report the model fitting results (such as a confusion table and misclassification rates). Which digit seems to get misclassified the most?

```r
# Handwritten Digit Recognition Data
library(ElemStatLearn)

# this is the training data!
dim(zip.test)
```

```
## [1] 2007  257
```

```r
train = zip.test

# this is the testing data!
dim(zip.train)
```

```
## [1] 7291  257
```

```r
test = zip.train
```

```r
pi_k <- rep(0,10)
mu_k <- matrix(0,10,256)
sig1 <- 0
for (k in c(0:9)){
  idx_k1 = which(train[,1]==k) # find indexes for each class
  nk1 = sum(as.numeric(train[,1]==k)) # num of obs in each class
  trn_k1 = train[idx_k1,] # training data for each class
  pi_k[k+1] = nk1/nrow(train)
  mu_k[k+1,] = apply(trn_k1[,-1],2,mean)
  s1 = 0
  for (i in 1:nk1){
    s1 = s1 + (trn_k1[i,-1] - mu_k[k+1,]) %*% t(trn_k1[i,-1] - mu_k[k+1,])
    # find the variances within each class
  }
  sig1 = sig1 + s1
}
sig1 = sig1 / (nrow(train) - 10) # the estimated sigma
```

```
temp_pred2a = matrix(0,nrow(test),10)
for (j in 1:10){
  temp_mu1 = as.matrix(mu_k[j,])
  temp1 = matrix(t(temp_mu1) %*% solve(sig1) %*% temp_mu1 / (-2) + log(pi_k[j]),nrow(test),1)
  temp_pred2a[,j] = test[,-1] %*% solve(sig1) %*% temp_mu1 + temp1
}
rm(temp1, temp_mu1)
```

```
pred2a = apply(temp_pred2a,1,which.max)-1
mean(pred2a!=test[,1])
```

```
## [1] 0.09916335
```

```
table(pred_value=pred2a,true_value=test[,1])
```

```
##           true_value
## pred_value    0    1    2    3    4    5    6    7    8    9
##          0 1158    0    4    9    3    9   11    1    8    0
##          1    0 1001    4    0   19    1    8    1    3    2
##          2    2    1  615   14   17   11    9    0    5    1
##          3    5    0   23  562    2   32    0    4   19    1
##          4    6    1   26    3  554   15   10   13   19   40
##          5    6    0    6   28    2  469    7    1   12    0
##          6    5    0   17    0    5    7  611    0    6    0
##          7    0    0    3   11    0    0    0  569    1   23
##          8   12    1   30   24    9    9    8    3  460    8
##          9    0    1    3    7   41    3    0   53    9  569
```

```
mis2a = matrix(0,1,10)
colnames(mis2a) = seq(0,9)
for (m in c(0:9)){
  mis2a[1,m+1] = mean(pred2a[which(test[,1]==m)]!=test[which(test[,1]==m),1])
}
mis2a
```

```
##               0         1         2         3         4         5          6         7
## [1,] 0.03015075 0.0039801 0.1586867 0.1458967 0.1503067 0.1564748 0.07981928 0.1178295
##              8         9
## [1,] 0.1512915 0.1164596
```

The digit 5 is most misclassified.

- [15 points] QDA does not work directly in this example because we do not have enough samples to estimate the inverse covariance matrix. An alternative idea to fix this issue is to consider a regularized

QDA method, which uses

$$\widehat{\Sigma}_k(\alpha) = \alpha\widehat{\Sigma}_k + (1-\alpha)\widehat{\Sigma}$$

for some $\alpha \in (0,1)$. Here $\widehat{\Sigma}$ is the estimation from the LDA method. Implement this method and select the best tuning parameter (on a grid) based on the testing error. You should again report the model fitting results similar to the previous part. What is your best tuning parameter, and what does that imply in terms of the underlying data and the performence of the model?

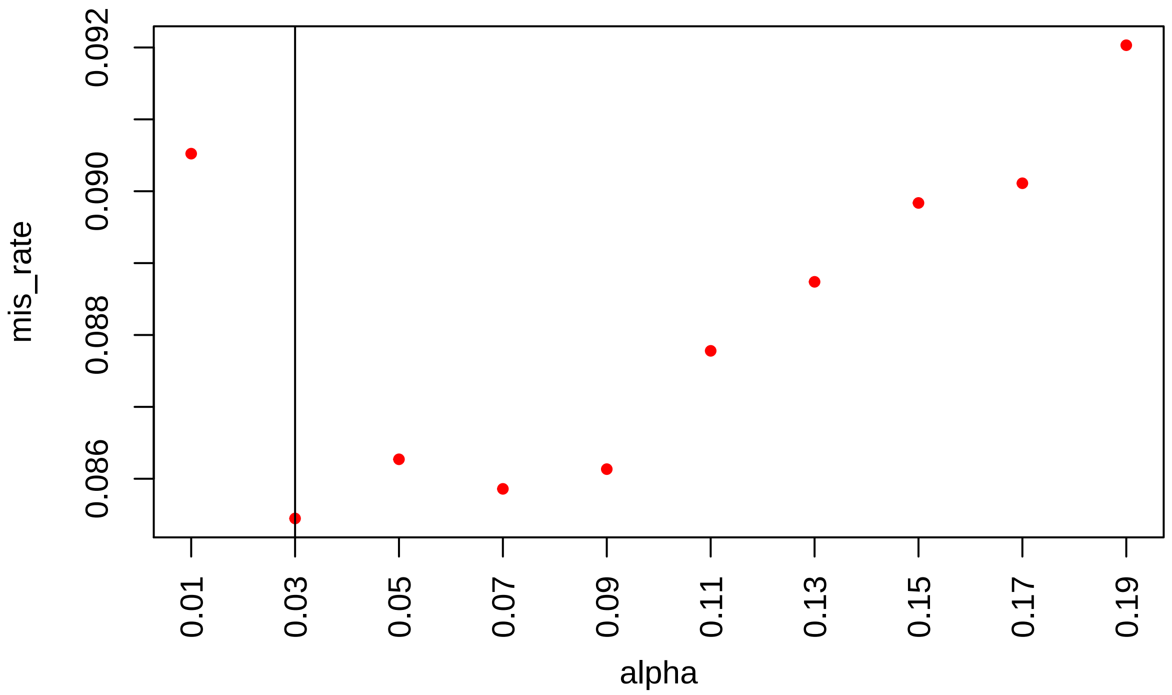
```r
sig2 <- list()
for (k in c(0:9)){
  idx_k2 = which(train[,1]==k) # find indexes for each class
  nk2 = sum(as.numeric(train[,1]==k)) # num of obs in each class
  trn_k2 = train[idx_k2,] # training data for each class
  s2 = 0
  for (i in 1:nk2){
    s2 = s2 + as.matrix(trn_k2[i,-1] - mu_k[k+1,]) %*% t(as.matrix(trn_k2[i,-1] - mu_k[k+1,]))
    # find the variances within each class
  }
  sig2[[k+1]] = s2 / (nk2 - 1)
  # the sigma for each class
}
```

I chose range of alpha to be 0.01 to 0.2 since I found that the misclassification rate monotonely increases when alpha is larger than 0.2

```r
alpha = seq(0.01,0.2,by=0.02)
temp_pred2b = matrix(0,nrow(test),10)
pred2b = matrix(NA,nrow(test),length(alpha))
mis_rate = rep(0,length(alpha))
for (l in 1:length(alpha)){
  a = alpha[l]
  for (j in 1:10){
    temp_sig2 = a*sig2[[j]] + (1-a)*sig1
    temp2a = matrix(determinant(temp_sig2)$modulus[1]/(-2) + log(pi_k[j]),nrow(test),1)
    temp2x = test[,-1]-matrix(mu_k[j,],nrow(test),256,byrow=T)
    temp2b = as.matrix(diag(temp2x%*%solve(temp_sig2)%*%t(temp2x)),nrow(test),1)
    #temp2a & 2b are components in discriminant function
    temp_pred2b[,j] = temp2b /(-2) + temp2a
  }
  pred2b[,l] = apply(temp_pred2b,1,which.max)-1
  mis_rate[l] = mean(pred2b[,l]!=test[,1])
}
```

The misclassification rate vs. each alpha is shown below:

```r
plot(alpha,mis_rate,col="red",pch=20,xaxt="n")
axis(1,at=seq(0.01,0.2,by=0.02),las=2)
abline(v=alpha[which.min(mis_rate)])
```



The best tuned alpha is 0.03

The alpha is very small which means only changing a little bit from LDA gave us the best performance of QDA.

```
pred2b_m = pred2b[,which.min(mis_rate)]
mean(pred2b_m!=test[,1])
```

```
## [1] 0.08544781
```

```
table(pred_value=pred2b_m,true_value=test[,1])
```

```
##           true_value
## pred_value    0    1    2    3    4    5    6    7    8    9
##          0 1163    0    1    7    2    7   11    1    5    0
##          1    0 1000    4    0   13    1    8    1    3    2
##          2    4    2  666   18   15   13   10    1    2    0
##          3    1    0   20  563    0   31    0    4   13    1
##          4    4    1   11    2  563   10    8   10   11   32
##          5    5    0    2   31    2  471    3    1   11    2
##          6    5    0    3    0    8    9  615    0    5    0
##          7    2    0    2   11    0    0    0  568    3   22
##          8   10    1   19   20    5   12    9    1  480    6
##          9    0    1    3    6   44    2    0   58    9  579
```

```
mis2b = matrix(0,1,10)
colnames(mis2b) = seq(0,9)
for (m in c(0:9)){
  mis2b[1,m+1] = mean(pred2b_m[which(test[,1]==m)]!=test[which(test[,1]==m),1])
}
mis2b
```

```
##               0           1          2         3         4         5          6         7
## [1,] 0.02596315 0.004975124 0.08891929 0.1443769 0.1365031 0.1528777 0.07379518 0.1193798
##              8          9
## [1,] 0.1143911 0.1009317
```

- [15 points] Naive Bayes is another approach that can be used for discriminant analysis. Instead of jointly modeling the density of pixels as multivariate Gaussian, we treat them independently.

  $$f_k(x) = \prod_j f_{kj}(x_j).$$

  However, this brings up other difficulties, such as the dimensionality problem. Hence, we consider first to reduce the dimension of the data and then use independent Gaussian distributions to model the density. It proceeds with the following:
  - Perform PCA on the training data and extract the first ten principal components.
  - model the density based on the principal components using the Naive Bayes approach. Assume that each $f_{kj}(x_j)$ Gaussian density and estimate their parameters.
  - Apply the model to the testing data and evaluate the performance. Report the results similarly to the previous two parts.

```
post3 = matrix(NA,nrow(test),10)
pca_temp = prcomp(train[,-1])
pca_tst = scale(test[,-1],pca_temp$center,pca_temp$scale) %*% pca_temp$rotation[,1:10]
for (k in c(0:9)){
  idx_k3 = which(train[,1]==k)
  trn_k3 = train[idx_k3,]
  pca_trn = scale(trn_k3[,-1],pca_temp$center,pca_temp$scale) %*% pca_temp$rotation[,1:10]
  pca_mean = matrix(apply(pca_trn,2,mean),nrow(test),10,byrow=T)
  pca_sd = matrix(apply(pca_trn,2,sd),nrow(test),10,byrow=T)
  prob3 = matrix(dnorm(pca_tst,pca_mean,pca_sd),nrow(test),10)
  post3[,k+1] = apply(prob3,1,prod) * pi_k[k+1]
}
```

```
pred2c = apply(post3,1,which.max)-1
mean(pred2c!=test[,1])
```

```
## [1] 0.1603347
```

```
table(pred_value=pred2c,true_value=test[,1])
```

```
##           true_value
## pred_value    0    1    2    3    4    5    6    7    8    9
##          0 1056    0    9    3    3   24   99    0    3    0
##          1    0  993    1    0    6    0    1    0    2    1
##          2    7    0  604   13   14    7   17    7    9    1
##          3   12    0   17  553    0   45    0    0   13    1
##          4    7    1   53    2  515   14    4    5   13   89
##          5    8    2    8   33    2  384    6    5   14    3
##          6  103    1    8    3   15   23  535    1    8    0
##          7    0    0    9    5    3    2    0  533    2   43
##          8    1    6   19   40    6   46    2    4  449    6
##          9    0    2    3    6   88   11    0   90   29  500
```

```
mis2c = matrix(0,1,10)
colnames(mis2c) = seq(0,9)
for (m in c(0:9)){
  mis2c[1,m+1] = mean(pred2c[which(test[,1]==m)]!=test[which(test[,1]==m),1])
}
mis2c
```

```
##              0         1         2         3         4         5         6         7
## [1,] 0.1155779 0.0119403 0.1737346 0.1595745 0.2101227 0.3093525 0.1942771 0.1736434
##               8         9
## [1,] 0.1715867 0.2236025
```

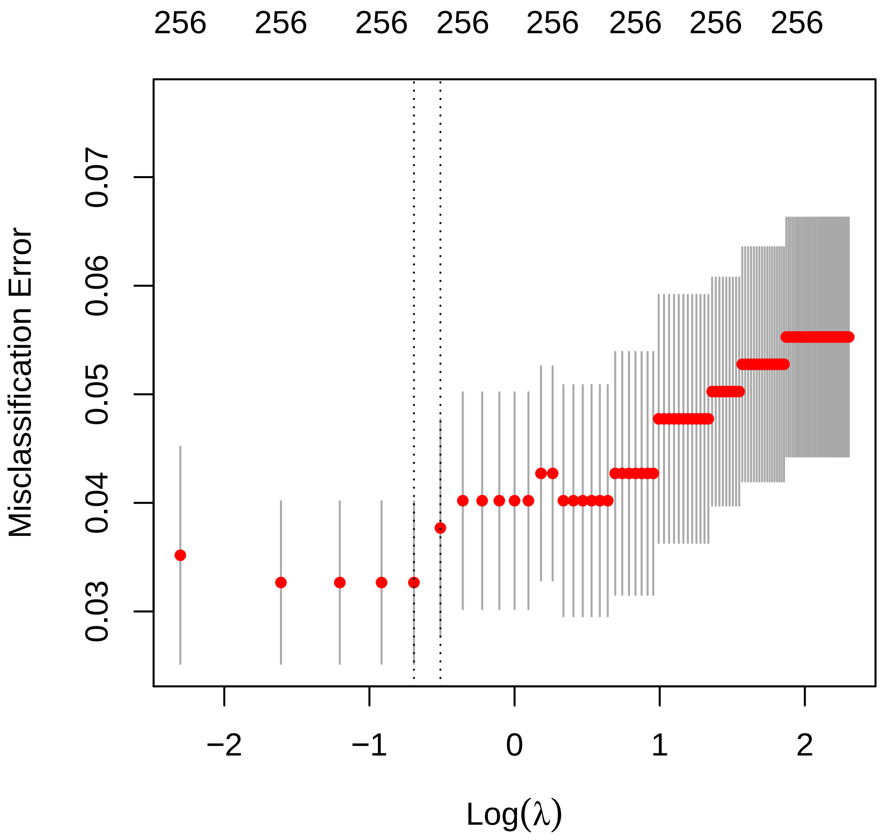
The digit 5 is most misclassified.

Question 3 [15 Points] Penalized Logistic Regression

Take digits 2 and 4 from the handwritten digit recognition data and perform logistic regression using penalized linear regression (with Ridge penalty). Use the same train-test setting we had in Question 2. However, for this question, you need to perform cross-validation on the training data to select the best tuning parameter. This can be done using the `glmnet` package. Evaluate and report the performance on the testing data, and summarize your fitting result. A researcher is interested in which regions of the pixel are most relevant in differentiating the two digits. Use an intuitive way to present your findings.

```r
library(glmnet)
trn3 = train[which(train[,1]==2|train[,1]==4),]
tst3 = test[which(test[,1]==2|test[,1]==4),]
trn3_x = scale(trn3[,-1])
trn3_y = trn3[,1]
tst3_x = scale(tst3[,-1])
tst3_y = tst3[,1]
```

```r
m3 = cv.glmnet(trn3_x, trn3_y, family="binomial", type.measure="class",
               alpha=0, nfolds=10, lambda=seq(0,10,by=0.1))
plot(m3)
```



```r
pred3 = predict(m3, newx=tst3_x, s="lambda.min", type="class")
mean(pred3!=tst3_y)
```

```
## [1] 0.0253073
```

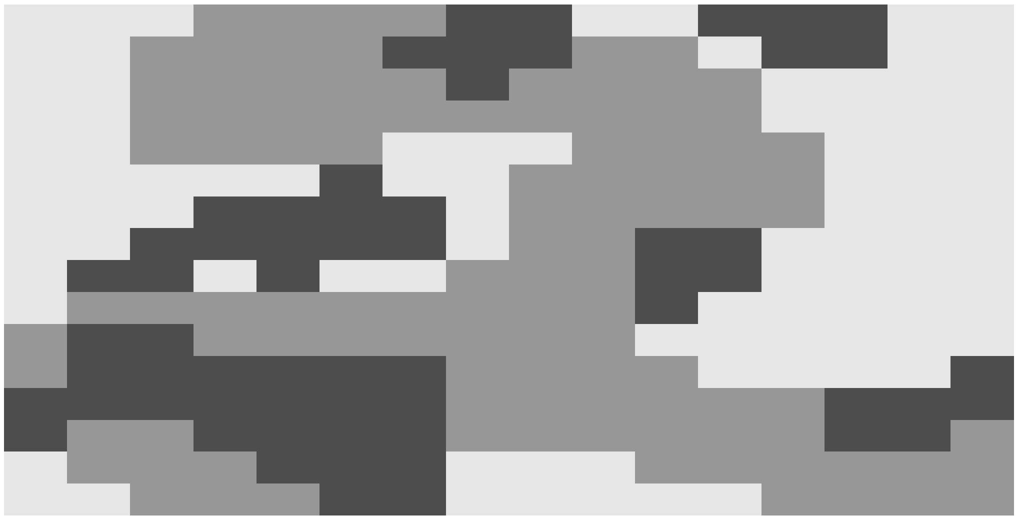
```r
table(pred_value=pred3,true_value=tst3_y)
```

```
##             true_value
## pred_value   2   4
##            2 706  10
##            4  25 642
```

```r
top60 = function(x, n=60) {
  nx = length(x)
  p = nx-n
  xp = sort(x, partial=p)[p]
  which(x > xp)
}
region = top60(abs(coef(m3)))
```

I selected 60 pixels with highest absolute values of coefficients in the model. Then I chose one example for each digit and changed the color of the digit to be light grey, the color of the selected pixels to be black, and the rest of the region to be white.

```r
d2 = trn3_x[5,]
d2[which(d2>=0)] = 87
d2[which(d2<0)] = 255
d2[region] = 0
digit2 = t(apply(matrix(d2,16,16,byrow=T),2,rev))
image(digit2, col = gray.colors(255), xaxs = "i", yaxs = "i", axes = F)
```



```r
d4 = trn3_x[6,]
d4[which(d4>=0)] = 87
d4[which(d4<0)] = 255
d4[region] = 0
digit4 = t(apply(matrix(d4,16,16,byrow=T),2,rev))
image(digit4, col = gray.colors(255), xaxs = "i", yaxs = "i", axes = F)
```