Machine Learning Exam

Kangbo Chen(kanch152)

2021/8/24

solemnly swear that I wrote the exam honestly, I did not use any unpermitted aids, nor did I communicate with anybody except of the course examiners.

Kangbo Chen

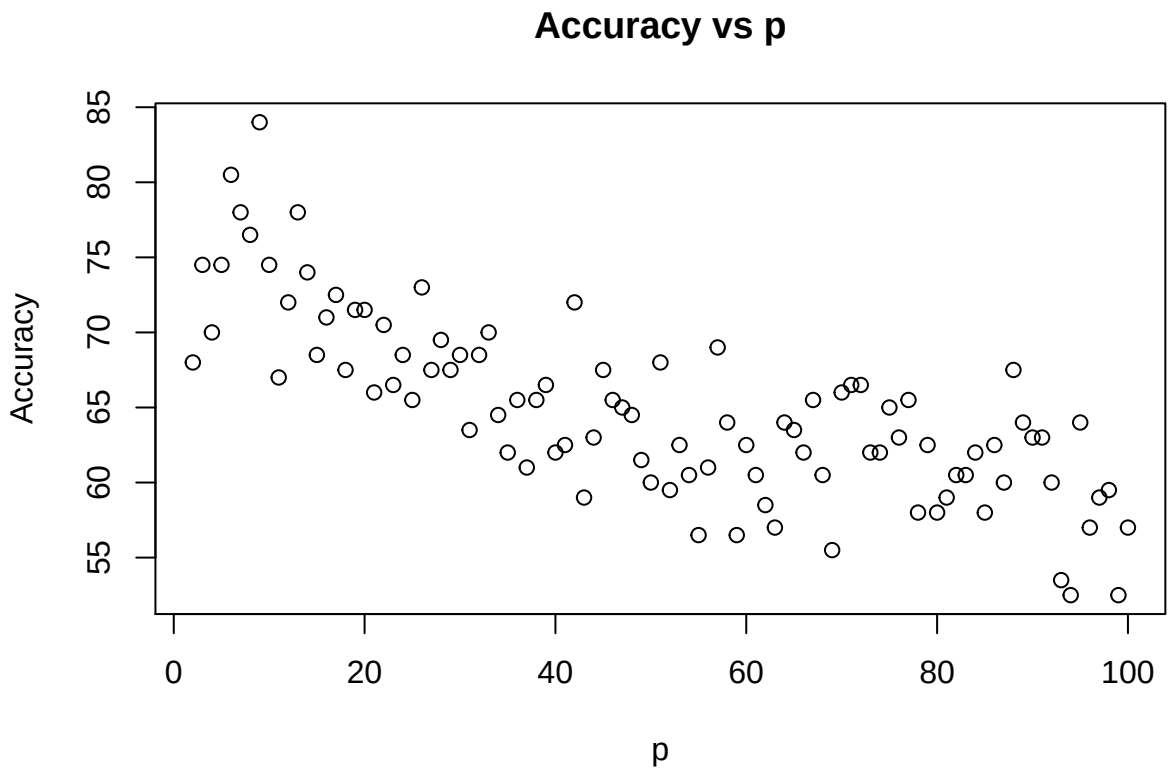
Assignment 1

part1

1

```
generate_f <- function(n,p){

  y <- c()
  x <- matrix(0,n,p)
  for (i in 1:n) {
    x1 <- runif(p,0,1)
    x[i,] <- x1
    ep <- rnorm(1,0,sd = sqrt(0.1))
    if(sum(x[i,])<0.5*p+ep){
      y[i] <- 1
    }else{
      y[i] <- 0
    }
  }
  res <- data.frame(x,y)
  return(res)
}
# generate training and test data
set.seed(12345)
train <- lapply(2:100,function(p){generate_f(100,p)})
test <- lapply(2:100,function(p){generate_f(200,p)})

library(kknn)
kknn_test <- list()
accuracy <- c()

accur_rate <- function(X,X1){
  t <- table(X,X1)
```

```
  return(sum(diag(t))/sum(t))
}
num_mod <- 2:100
#get accuracy rate for each p
for (i in 1:length(num_mod)) {

  kknn_test[[i]] <- kknn(as.factor(y) ~., train[[i]], test[[i]], k = 3, kernel = "rectangular")

  accuracy[i] <- accur_rate(test[[i]]$y,kknn_test[[i]]$fitted.values)

}

plot(num_mod,accuracy*100,xlab = 'p',ylab = 'Accuracy',main = "Accuracy vs p")
```

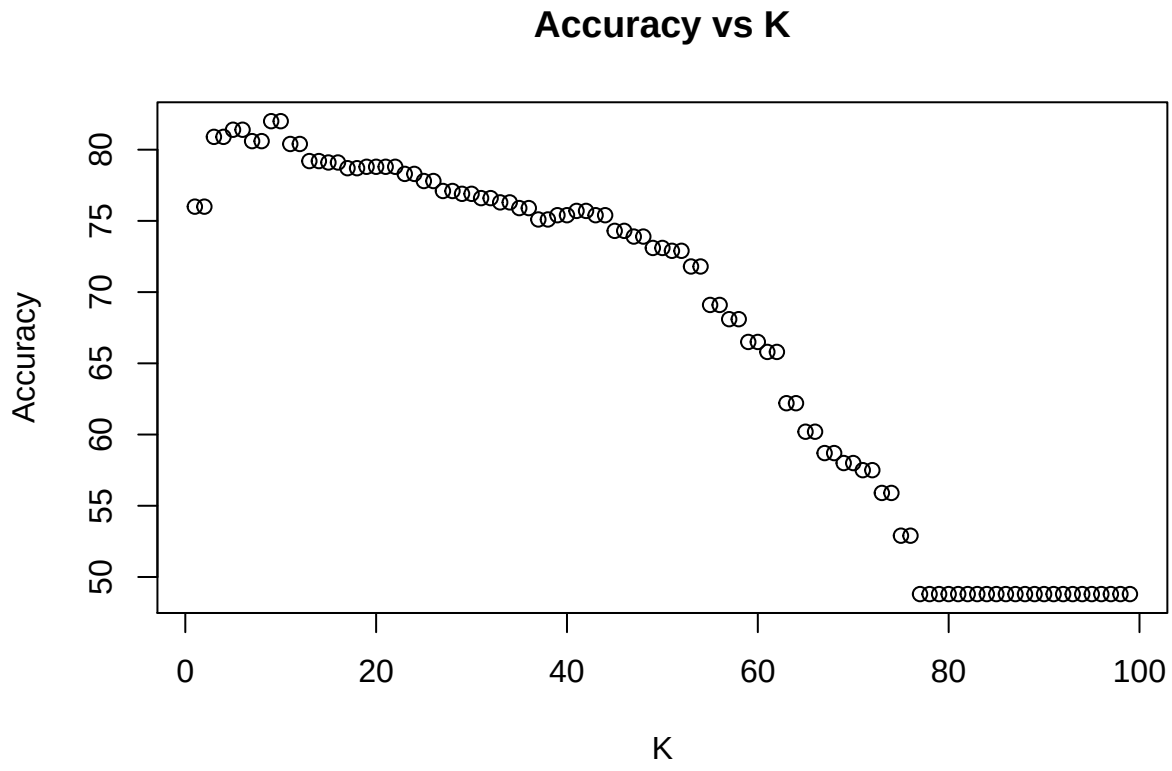


comment: As we can see from the figure there is a decreasing trend for accuracy as p increases.p increases meaning that the data is more complex, therefore,this trend shows that test accuracy for knn model decrease as the data becoming more complex.

2

```
set.seed(12345)
train_2 <- generate_f(100,3)
test_2 <- generate_f(1000,3)
k_v <- 1:99
```

```r
kknn_test2 <- list()
accuracy2 <- c()
for (i in k_v) {

  kknn_test2[[i]] <- kknn(as.factor(y) ~., train_2, test_2, k = i, kernel = "rectangular")

  accuracy2[i] <- accur_rate(test_2$y,kknn_test2[[i]]$fitted.values)

}
plot(k_v,accuracy2*100,xlab = 'K',ylab = 'Accuracy',main = "Accuracy vs K")
```



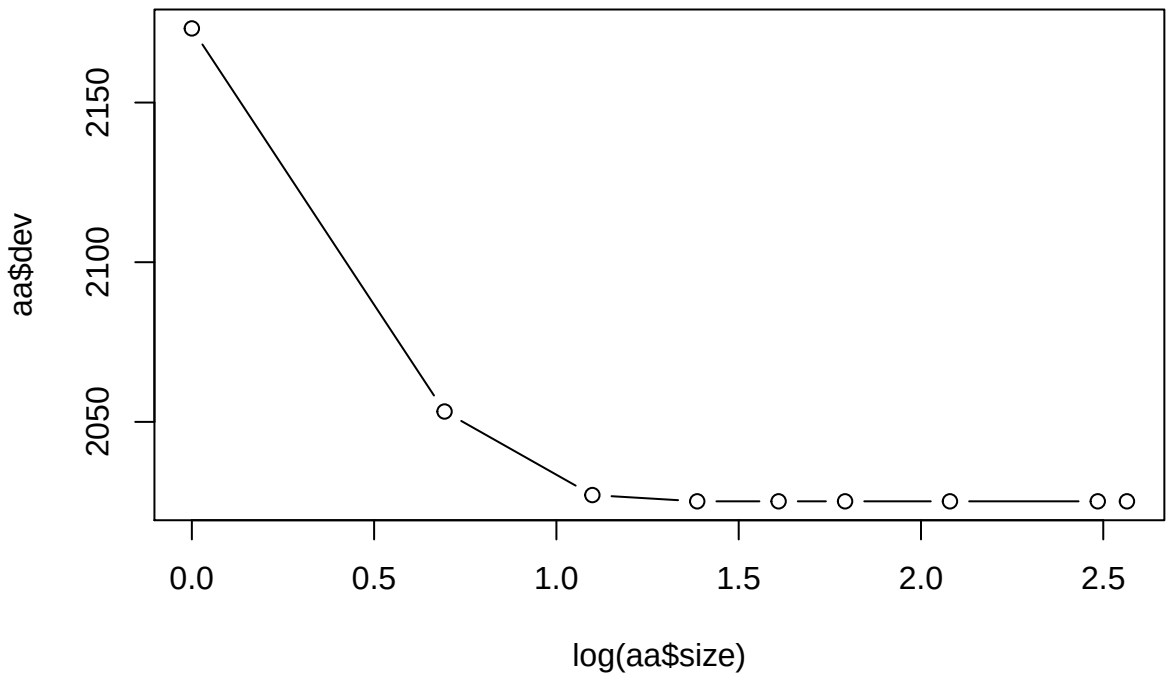
comment: As we can see from the figure there is a decreasing trend for accuracy as K increases.K increases meaning that the model is more complex, Thus,this trend shows that test accuracy for knn model decrease as the model become more complex.

part2

1

```r
library(tree)
data <- read.csv('women.csv')

n=dim(data)[1]
```

```
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train_wo=data[id,]
test_wo=data[-id,]

tree_full <- tree(as.factor(Death)~.,data = train_wo,mindev=0.003)
aa <- cv.tree(tree_full)
plot(log(aa$size),aa$dev,type = 'b')
```




```
aa$dev
```

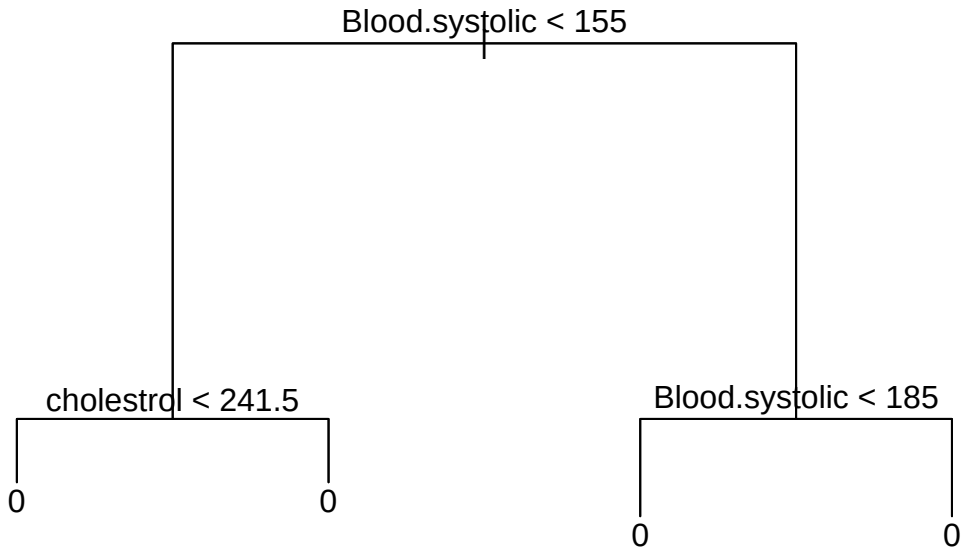
```
## [1] 2025.122 2025.122 2025.122 2025.122 2025.122 2025.122 2027.079 2053.246
## [9] 2173.208
```

```
aa$size
```

```
## [1] 13 12  8  6  5  4  3  2  1
```

We can find from the figure above and the deviance and tree size results, that the deviance does not change when tree size takes between 4 and 13. Here, we choose the smallest tree size,tree size equals to4, as it is better for storage.

```
finalTree=prune.tree(tree_full, best=4)
plot(finalTree)
text(finalTree)
```




```
yfit <- predict(finalTree,newdata = test_wo,type = 'class')

misclass <- function(X,X1){
  t <- table(X,X1)
  return(1-sum(diag(t))/sum(t))
}

mismatch <- misclass(test_wo[['Death']],yfit)
cat('The misclassification rate for chosen tree size is ',mismatch )
```

```
## The misclassification rate for chosen tree size is  0.15
```

```
print('The confusion matrix for test set is:')
```

```
## [1] "The confusion matrix for test set is:"
```

```
print(table(test_wo[['Death']],yfit))
```

```
##    yfit
```

```
##         0    1
##   0 2125    0
##   1  375    0
```

(b) We can know from the optimal tree that 2 features are chosen

###2

```
library(mgcv)
```

```
## Warning: package 'mgcv' was built under R version 4.0.5
```

```
## Loading required package: nlme
```

```
## This is mgcv 1.8-36. For overview type 'help("mgcv-package")'.
```

```
library(akima)
```

```
## Warning: package 'akima' was built under R version 4.0.5
```

```
library(plotly)
```

```
## Warning: package 'plotly' was built under R version 4.0.5
```

```
## Loading required package: ggplot2
```

```
##
## Attaching package: 'plotly'
```

```
## The following object is masked from 'package:ggplot2':
##
##     last_plot
```

```
## The following object is masked from 'package:stats':
##
##     filter
```

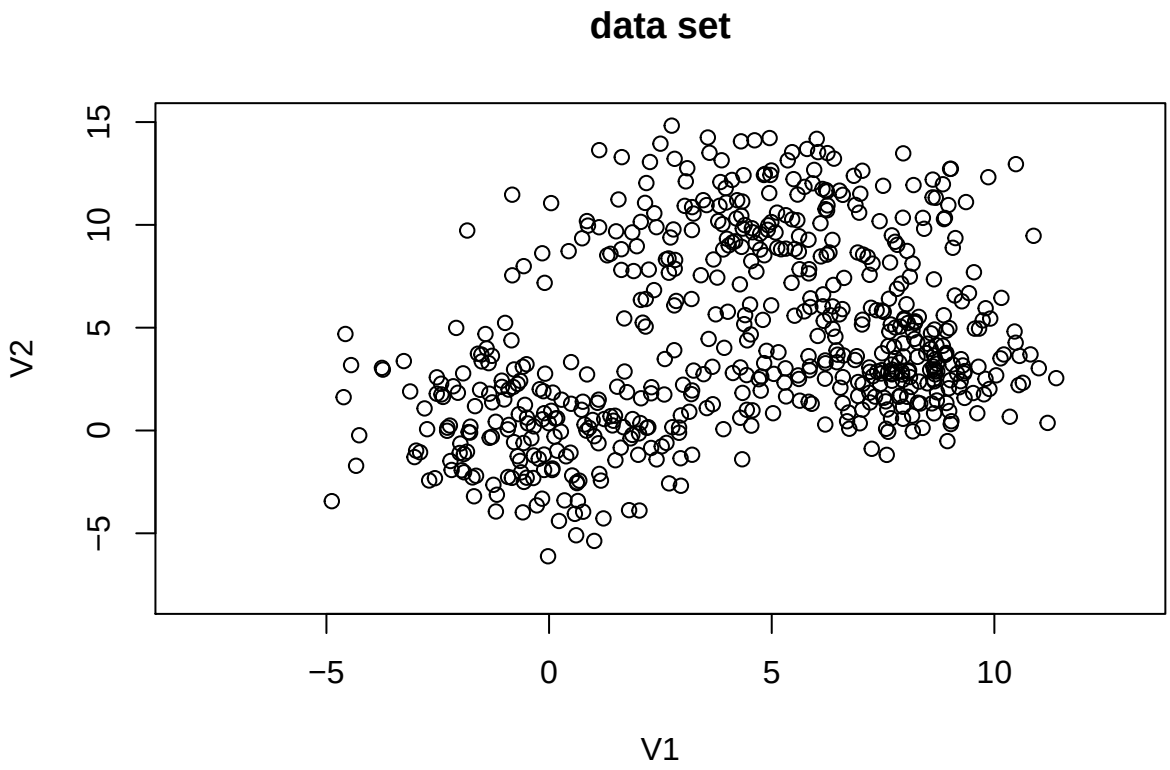
```
## The following object is masked from 'package:graphics':
##
##     layout
```

```
new_tr <- train_wo[,c(4,9,13)]
new_va <- test_wo[,c(4,9,13)]
```

Assignment 2

EM

```r
## EM
library(mvtnorm)
set.seed(12345)

max_it <- 100 # max number of EM iterations
min_change <- 0.1 # min change in log likelihood between two consecutive EM iterations
N=600 # number of training points
D=2 # number of dimensions
x <- matrix(nrow=N, ncol=D) # training data

data <- read.table('dataEM.txt')
plot(data,xlim=c(-8,13),ylim=c(-8,15),main="data set")
```




```r
x <- data

K=3 # number of guessed components
z <- matrix(nrow=N, ncol=K) # fractional component assignments
pi <- vector(length = K) # mixing coefficients
mu <- matrix(nrow=K, ncol=D) # conditional means
Sigma <- array(dim=c(D,D,K)) # conditional covariances
llik <- vector(length = max_it) # log likelihood of the EM iterations

# Random initialization of the parameters
pi <- runif(K,0,1)
pi <- pi / sum(pi)
```

```
for(k in 1:K) {
  mu[k,] <- runif(D,0,5)
  Sigma[,,k]<-c(1,0,0,1)
}
pi
```

```
## [1] 0.3057710 0.3714587 0.3227703
```

```
mu
```

```
##           [,1]     [,2]
## [1,] 4.4306228 2.282405
## [2,] 0.8318589 1.625477
## [3,] 2.5461217 3.638526
```

```
Sigma
```

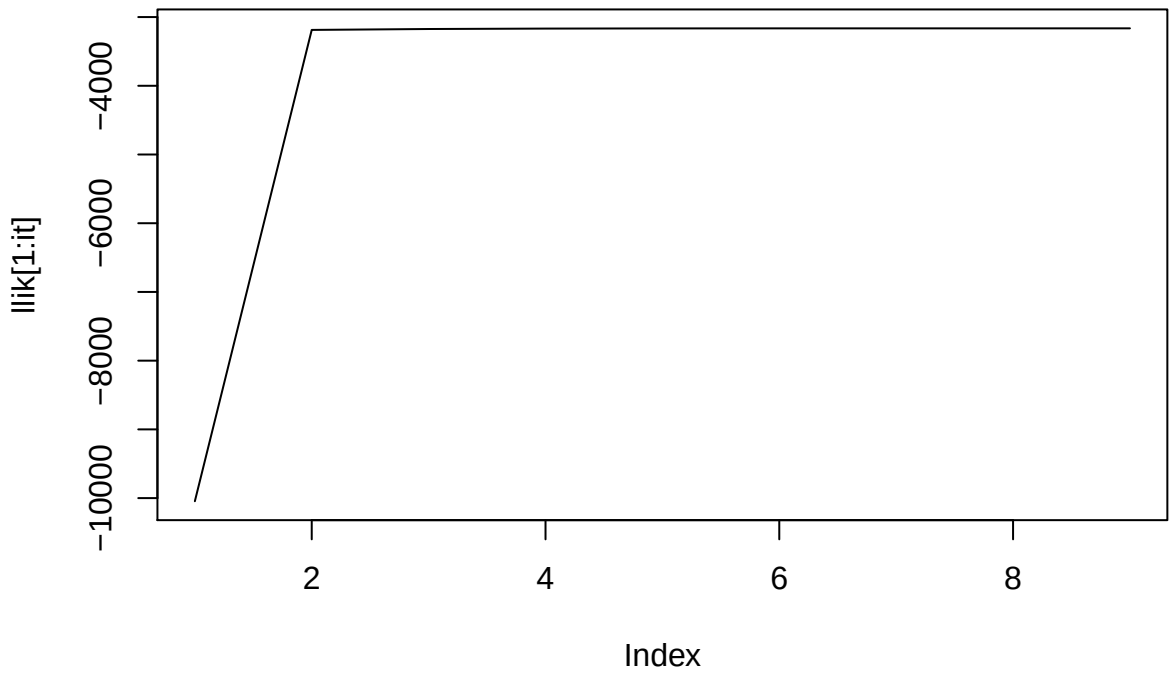
```
## , , 1
##
##      [,1] [,2]
## [1,]    1    0
## [2,]    0    1
##
## , , 2
##
##      [,1] [,2]
## [1,]    1    0
## [2,]    0    1
##
## , , 3
##
##      [,1] [,2]
## [1,]    1    0
## [2,]    0    1
```

```
for(it in 1:max_it) {
  # E-step: Computation of the fractional component assignments
  llik[it] <- 0
  for(n in 1:N) {
    for(k in 1:K) {
      z[n,k] <- pi[k]*dmvnorm(x[n,],mu[k,],Sigma[,,k])
    }

    #Log likelihood computation.
    llik[it] <- llik[it] + log(sum(z[n,]))

    z[n,] <- z[n,]/sum(z[n,])
  }

  cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
  flush.console()
```

```
  # Stop if the lok likelihood has not changed significantly
  if (it > 1) {
    if(abs(llik[it] - llik[it-1]) < min_change) {
      break
    }
  }

  #M-step: ML parameter estimation from the data and fractional component assignments
  for(k in 1:K) {
    pi[k] <- sum(z[,k]) / N
    for(d in 1:D) {
      mu[k, d] <- sum(x[, d] * z[, k]) / sum(z[,k])
    }
    for(d in 1:D) {#(*unrestrict*)
      for(d2 in 1:D)
      Sigma[d,d2,k]<-sum((x[, d]-mu[k,d])*(x[, d2]-mu[k,d2]) * z[, k]) / sum(z[,k])
    }
  }
}
```

```
## iteration:  1 log likelihood:  -10045.72
## iteration:  2 log likelihood:  -3187.292
## iteration:  3 log likelihood:  -3173.888
## iteration:  4 log likelihood:  -3167.725
## iteration:  5 log likelihood:  -3165.553
## iteration:  6 log likelihood:  -3164.872
## iteration:  7 log likelihood:  -3164.618
## iteration:  8 log likelihood:  -3164.493
## iteration:  9 log likelihood:  -3164.42
```

```
plot(llik[1:it], type="l")
```




```
mu
```

```
##             [,1]     [,2]
## [1,]  7.62336625 3.131895
## [2,] -0.06694873 0.130733
## [3,]  4.83866448 9.898083
```

```
Sigma
```

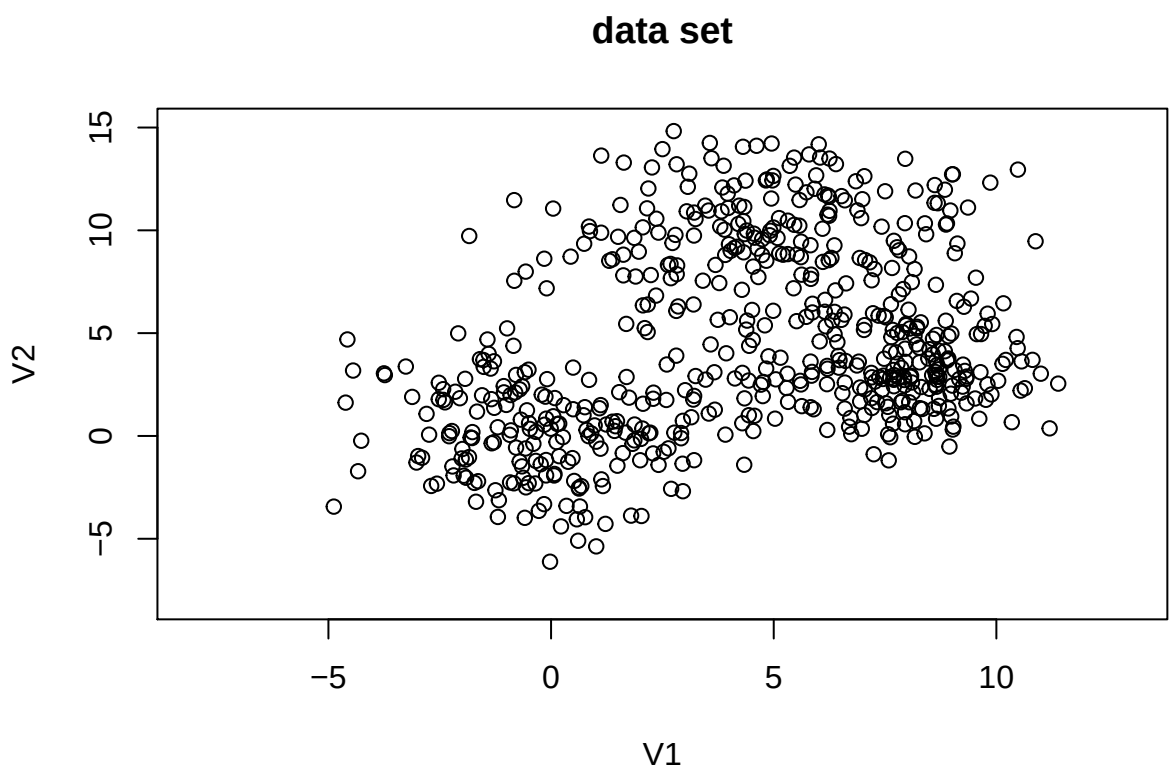
```
## , , 1
##
##            [,1]       [,2]
## [1,] 2.88179961 0.05058612
## [2,] 0.05058612 3.17694472
##
## , , 2
##
##            [,1]       [,2]
## [1,]  3.9196891 -0.3030387
## [2,] -0.3030387  5.2440557
##
## , , 3
##
##          [,1]     [,2]
## [1,] 6.057763 0.661176
## [2,] 0.661176 5.235981
```

```
BIC<-llik[it] - log(N) * 0.5 * ((K-1)+K*D+K*D)
BIC
```

```
## [1] -3209.198
```

comment: we can see that the liklihood value is increasing as iteration number grows.

```
############################## restricted MM
set.seed(12345)

max_it <- 100 # max number of EM iterations
min_change <- 0.1 # min change in log likelihood between two consecutive EM iterations
N=600 # number of training points
D=2 # number of dimensions
x <- matrix(nrow=N, ncol=D) # training data

data <- read.table('dataEM.txt')
plot(data,xlim=c(-8,13),ylim=c(-8,15),main="data set")
```




```
x <- data

K=3 # number of guessed components
z <- matrix(nrow=N, ncol=K) # fractional component assignments
pi <- vector(length = K) # mixing coefficients
```

```
mu <- matrix(nrow=K, ncol=D) # conditional means
Sigma <- array(dim=c(D,D,K)) # conditional covariances
llik <- vector(length = max_it) # log likelihood of the EM iterations

# Random initialization of the parameters
pi <- runif(K,0,1)
pi <- pi / sum(pi)
for(k in 1:K) {
  mu[k,] <- runif(D,0,5)
  Sigma[,,k]<-c(1,0,0,1)
}
pi
```

```
## [1] 0.3057710 0.3714587 0.3227703
```

```
mu
```

```
##           [,1]     [,2]
## [1,] 4.4306228 2.282405
## [2,] 0.8318589 1.625477
## [3,] 2.5461217 3.638526
```

```
Sigma
```

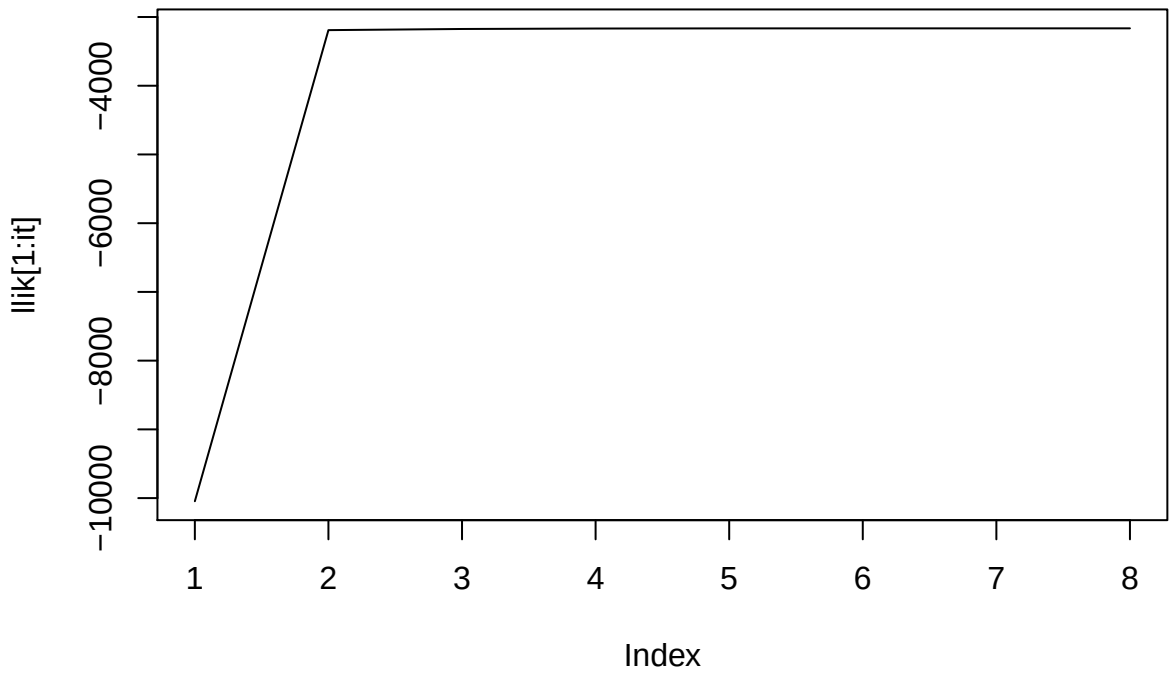
```
## , , 1
##
##      [,1] [,2]
## [1,]    1    0
## [2,]    0    1
##
## , , 2
##
##      [,1] [,2]
## [1,]    1    0
## [2,]    0    1
##
## , , 3
##
##      [,1] [,2]
## [1,]    1    0
## [2,]    0    1
```

```
for(it in 1:max_it) {
  # E-step: Computation of the fractional component assignments
  llik[it] <- 0
  for(n in 1:N) {
    for(k in 1:K) {
      z[n,k] <- pi[k]*dmvnorm(x[n,],mu[k,],Sigma[,,k])
    }

    #Log likelihood computation.
```

```
    llik[it] <- llik[it] + log(sum(z[n,]))

    z[n,] <- z[n,]/sum(z[n,])
  }

  cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
  flush.console()
  # Stop if the lok likelihood has not changed significantly
  if (it > 1) {
    if(abs(llik[it] - llik[it-1]) < min_change) {
      break
    }
  }

  #M-step: ML parameter estimation from the data and fractional component assignments
  for(k in 1:K) {
    pi[k] <- sum(z[,k]) / N
    for(d in 1:D) {
      mu[k, d] <- sum(x[, d] * z[, k]) / sum(z[,k])
    }
    for(d in 1:D) {#(*restrict*)
      Sigma[d,d,k]<-sum((x[, d]-mu[k,d])^2 * z[, k]) / sum(z[,k])
    }
    Sigma[1,2,k] <- 0
    Sigma[2,1,k] <- 0
  }
}
```

```
## iteration:  1 log likelihood:  -10045.72
## iteration:  2 log likelihood:  -3191.181
## iteration:  3 log likelihood:  -3174.338
## iteration:  4 log likelihood:  -3167.669
## iteration:  5 log likelihood:  -3165.78
## iteration:  6 log likelihood:  -3165.335
## iteration:  7 log likelihood:  -3165.193
## iteration:  8 log likelihood:  -3165.124
```

```
plot(llik[1:it], type="l")
```




```
mu
```

```
##              [,1]      [,2]
## [1,]  7.672315100 3.0938791
## [2,] -0.006717986 0.1865146
## [3,]  4.900586586 9.8548078
```

```
Sigma
```

```
## , , 1
##
##         [,1]     [,2]
## [1,] 2.74009 0.000000
## [2,] 0.00000 3.058321
##
## , , 2
##
##          [,1]     [,2]
## [1,] 4.090076 0.000000
## [2,] 0.000000 5.395413
##
## , , 3
##
##          [,1]     [,2]
## [1,] 6.070953 0.000000
## [2,] 0.000000 5.343097
```

```
BIC_restrict<-llik[it] - log(N) * 0.5 * ((K-1)+K*D+K*D)
BIC_restrict
```

```
## [1] -3209.902
```

comment: we can find that the BIC for unrestricted MM and restricted MM are -3209.929 and -3209.902 respectively.The BIC for unrestricted MM is lower, therefore, we can conclude that unrestricted MM is better.

Kernel

1

```
dak <- read.table('dataKernel.txt')

n <- dim(dak)[1]
set.seed(12345)
id <- sample(1:n, floor(n*0.5))
D2 <- dak[id,]
id1 <- setdiff(1:n, id)
D1 <- dak[id1,]
set.seed(12345)
id2 <- sample(id1, floor(n*0.25))
D11 <- dak[id2,]
id3 <- setdiff(id1,id2)
D12 <- dak[id3,]

# miscalssification rate function
misclass <- function(X,X1){
  t <- table(X,X1)
  return(1-sum(diag(t))/sum(t))
}

# kernel function
G_kernel <- function(x,h){
  res <- dnorm(x,sd = h)
  return(res)
}

#classify function
classify <- function(t,h,tra){

  c1 <- tra[which(tra[,2]==1),]
  c2 <- tra[which(tra[,2]==2),]

  c1_ker <- sapply(c1[,1], function(x){G_kernel(x-t,h)})
  c2_ker <- sapply(c2[,1], function(x){G_kernel(x-t,h)})

  p_t_c1 <- sum(c1_ker)/length(c1_ker)
  p_t_c2 <- sum(c2_ker)/length(c2_ker)
```

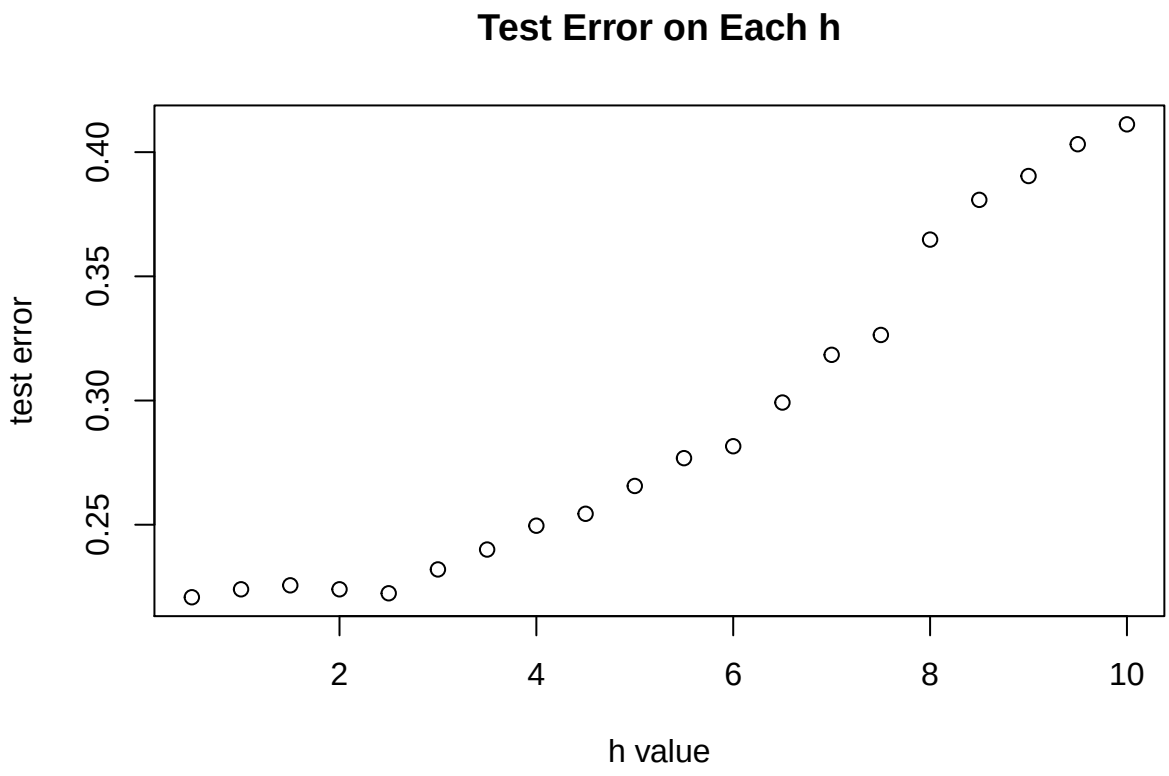
```
  c <- 1
  if(p_t_c1*length(c1_ker)/dim(tra)[1]<p_t_c2*length(c2_ker)/dim(tra)[1]){
    c <- 2
  }
  return(c)
}

# inner cross validation function to choose the h
cv_inner <- function(tr,va,h){

  mismatch <- c()
  # loop each h value
  for(i in 1:length(h)){
    c_v <- sapply(va[,1], function(x){classify(x,h[i],tr)})
    mismatch[i] <- misclass(as.numeric(va[,2]),c_v)
  }
  # return miscalssification rate
  return(mismatch)
}
#outter cross validation to summarize final result
cv_outer <- function(D1,D2,h){

  id1 <- as.numeric(rownames(D1))

  set.seed(12345)
  id2 <- sample(id1, floor(n*0.25))
  D11 <- dak[id2,]
  id3 <- setdiff(id1,id2)
  D12 <- dak[id3,]

  #cross
  mis1 <- cv_inner(D11,D12,h)
  mis2 <- cv_inner(D12,D11,h)
  # average error
  avg_err <- (mis1+mis2)/2
  h_opt1 <- h[which.min(avg_err)]

  test_err <- c()
  aa <- sapply(D2[,1], function(x){classify(x,h_opt1,D1)})
  test_err[1] <- misclass(as.numeric(D2[,2]),aa)

  # D1 D2 switch
  id1 <- as.numeric(rownames(D2))

  set.seed(12345)
  id2 <- sample(id1, floor(n*0.25))
  D11 <- dak[id2,]
  id3 <- setdiff(id1,id2)
  D12 <- dak[id3,]

  #cross
  mis1 <- cv_inner(D11,D12,h)
  mis2 <- cv_inner(D12,D11,h)
```

```
  #average error
  avg_err <- (mis1+mis2)/2
  h_opt2 <- h[which.min(avg_err)]

  aa <- sapply(D1[,1], function(x){classify(x,h_opt2,D2)})
  test_err[2] <- misclass(as.numeric(D1[,2]),aa)

  res <- list('h_opt1'=h_opt1,'h_opt2'=h_opt2,'test_error'=test_err)

  return(res)
}

h <- c(0.5,1,5,10)
res <- cv_outer(D1,D2,h)

cat('Average error on tests is ',mean(res$test_error))
```

```
## Average error on tests is  0.2352
```

###2

```
h2 <- seq(0.5,10,0.5)
test <- cv_inner(D11,D12,h2)

plot(h2,test,xlab = 'h value',ylab = 'test error',main = 'Test Error on Each h')
```



As the figure shown above, the test error for each corresponding h value grows gradually with h.We can roughly conclude that test error and h have positive correlation. Therefore, we can choose the average h values of those 2 folds cross validation as the optimal h value.

```
h_opt <- (res$h_opt1+res$h_opt2)/2
h_opt
```

```
## [1] 0.5
```

Appendix: All code for this report

```
knitr::opts_chunk$set(echo = TRUE)
generate_f <- function(n,p){

  y <- c()
  x <- matrix(0,n,p)
  for (i in 1:n) {
    x1 <- runif(p,0,1)
    x[i,] <- x1
    ep <- rnorm(1,0,sd = sqrt(0.1))
    if(sum(x[i,])<0.5*p+ep){
      y[i] <- 1
    }else{
      y[i] <- 0
    }
  }
  res <- data.frame(x,y)
  return(res)
}
# generate training and test data
set.seed(12345)
train <- lapply(2:100,function(p){generate_f(100,p)})
test <- lapply(2:100,function(p){generate_f(200,p)})

library(kknn)
kknn_test <- list()
accuracy <- c()

accur_rate <- function(X,X1){
  t <- table(X,X1)
  return(sum(diag(t))/sum(t))
}
num_mod <- 2:100
#get accuracy rate for each p
for (i in 1:length(num_mod)) {

  kknn_test[[i]] <- kknn(as.factor(y) ~., train[[i]], test[[i]], k = 3, kernel = "rectangular")

  accuracy[i] <- accur_rate(test[[i]]$y,kknn_test[[i]]$fitted.values)

}
```

```r
plot(num_mod,accuracy*100,xlab = 'p',ylab = 'Accuracy',main = "Accuracy vs p")


set.seed(12345)
train_2 <- generate_f(100,3)
test_2 <- generate_f(1000,3)
k_v <- 1:99
kknn_test2 <- list()
accuracy2 <- c()
for (i in k_v) {

  kknn_test2[[i]] <- kknn(as.factor(y) ~., train_2, test_2, k = i, kernel = "rectangular")

  accuracy2[i] <- accur_rate(test_2$y,kknn_test2[[i]]$fitted.values)

}
plot(k_v,accuracy2*100,xlab = 'K',ylab = 'Accuracy',main = "Accuracy vs K")

library(tree)
data <- read.csv('women.csv')

n=dim(data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train_wo=data[id,]
test_wo=data[-id,]

tree_full <- tree(as.factor(Death)~.,data = train_wo,mindev=0.003)
aa <- cv.tree(tree_full)
plot(log(aa$size),aa$dev,type = 'b')
aa$dev
aa$size
finalTree=prune.tree(tree_full, best=4)
plot(finalTree)
text(finalTree)

yfit <- predict(finalTree,newdata = test_wo,type = 'class')

misclass <- function(X,X1){
  t <- table(X,X1)
  return(1-sum(diag(t))/sum(t))
}

mismatch <- misclass(test_wo[['Death']],yfit)
cat('The misclassification rate for chosen tree size is ',mismatch )

print('The confusion matrix for test set is:')
print(table(test_wo[['Death']],yfit))
library(mgcv)
library(akima)
library(plotly)

new_tr <- train_wo[,c(4,9,13)]
```

```r
new_va <- test_wo[,c(4,9,13)]

## EM
library(mvtnorm)
set.seed(12345)

max_it <- 100 # max number of EM iterations
min_change <- 0.1 # min change in log likelihood between two consecutive EM iterations
N=600 # number of training points
D=2 # number of dimensions
x <- matrix(nrow=N, ncol=D) # training data

data <- read.table('dataEM.txt')
plot(data,xlim=c(-8,13),ylim=c(-8,15),main="data set")
x <- data

K=3 # number of guessed components
z <- matrix(nrow=N, ncol=K) # fractional component assignments
pi <- vector(length = K) # mixing coefficients
mu <- matrix(nrow=K, ncol=D) # conditional means
Sigma <- array(dim=c(D,D,K)) # conditional covariances
llik <- vector(length = max_it) # log likelihood of the EM iterations

# Random initialization of the parameters
pi <- runif(K,0,1)
pi <- pi / sum(pi)
for(k in 1:K) {
  mu[k,] <- runif(D,0,5)
  Sigma[,,k]<-c(1,0,0,1)
}
pi
mu
Sigma

for(it in 1:max_it) {
  # E-step: Computation of the fractional component assignments
  llik[it] <- 0
  for(n in 1:N) {
    for(k in 1:K) {
      z[n,k] <- pi[k]*dmvnorm(x[n,],mu[k,],Sigma[,,k])
    }

    #Log likelihood computation.
    llik[it] <- llik[it] + log(sum(z[n,]))

    z[n,] <- z[n,]/sum(z[n,])
  }

  cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
  flush.console()
  # Stop if the lok likelihood has not changed significantly
  if (it > 1) {
    if(abs(llik[it] - llik[it-1]) < min_change) {
```

```
      break
    }
  }

  #M-step: ML parameter estimation from the data and fractional component assignments
  for(k in 1:K) {
    pi[k] <- sum(z[,k]) / N
    for(d in 1:D) {
      mu[k, d] <- sum(x[, d] * z[, k]) / sum(z[,k])
    }
    for(d in 1:D) {#(*unrestrict*)
      for(d2 in 1:D)
      Sigma[d,d2,k]<-sum((x[, d]-mu[k,d])*(x[, d2]-mu[k,d2]) * z[, k]) / sum(z[,k])
    }
  }
}

plot(llik[1:it], type="l")
mu
Sigma

BIC<-llik[it] - log(N) * 0.5 * ((K-1)+K*D+K*D)
BIC
############################## restricted MM
set.seed(12345)

max_it <- 100 # max number of EM iterations
min_change <- 0.1 # min change in log likelihood between two consecutive EM iterations
N=600 # number of training points
D=2 # number of dimensions
x <- matrix(nrow=N, ncol=D) # training data

data <- read.table('dataEM.txt')
plot(data,xlim=c(-8,13),ylim=c(-8,15),main="data set")

x <- data

K=3 # number of guessed components
z <- matrix(nrow=N, ncol=K) # fractional component assignments
pi <- vector(length = K) # mixing coefficients
mu <- matrix(nrow=K, ncol=D) # conditional means
Sigma <- array(dim=c(D,D,K)) # conditional covariances
llik <- vector(length = max_it) # log likelihood of the EM iterations

# Random initialization of the parameters
pi <- runif(K,0,1)
pi <- pi / sum(pi)
for(k in 1:K) {
  mu[k,] <- runif(D,0,5)
  Sigma[,,k]<-c(1,0,0,1)
}
pi
mu
```

```
Sigma

for(it in 1:max_it) {
  # E-step: Computation of the fractional component assignments
  llik[it] <- 0
  for(n in 1:N) {
    for(k in 1:K) {
      z[n,k] <- pi[k]*dmvnorm(x[n,],mu[k,],Sigma[,,k])
    }

    #Log likelihood computation.
    llik[it] <- llik[it] + log(sum(z[n,]))

    z[n,] <- z[n,]/sum(z[n,])
  }

  cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
  flush.console()
  # Stop if the lok likelihood has not changed significantly
  if (it > 1) {
    if(abs(llik[it] - llik[it-1]) < min_change) {
      break
    }
  }

  #M-step: ML parameter estimation from the data and fractional component assignments
  for(k in 1:K) {
    pi[k] <- sum(z[,k]) / N
    for(d in 1:D) {
      mu[k, d] <- sum(x[, d] * z[, k]) / sum(z[,k])
    }
    for(d in 1:D) {#(*restrict*)
      Sigma[d,d,k]<-sum((x[, d]-mu[k,d])^2 * z[, k]) / sum(z[,k])
    }
    Sigma[1,2,k] <- 0
    Sigma[2,1,k] <- 0
  }
}

plot(llik[1:it], type="l")
mu
Sigma

BIC_restrict<-llik[it] - log(N) * 0.5 * ((K-1)+K*D+K*D)
BIC_restrict
dak <- read.table('dataKernel.txt')

n <- dim(dak)[1]
set.seed(12345)
id <- sample(1:n, floor(n*0.5))
D2 <- dak[id,]
id1 <- setdiff(1:n, id)
D1 <- dak[id1,]
```

```r
set.seed(12345)
id2 <- sample(id1, floor(n*0.25))
D11 <- dak[id2,]
id3 <- setdiff(id1,id2)
D12 <- dak[id3,]

# miscalssification rate function
misclass <- function(X,X1){
  t <- table(X,X1)
  return(1-sum(diag(t))/sum(t))
}

# kernel function
G_kernel <- function(x,h){
  res <- dnorm(x,sd = h)
  return(res)
}

#classify function
classify <- function(t,h,tra){

  c1 <- tra[which(tra[,2]==1),]
  c2 <- tra[which(tra[,2]==2),]

  c1_ker <- sapply(c1[,1], function(x){G_kernel(x-t,h)})
  c2_ker <- sapply(c2[,1], function(x){G_kernel(x-t,h)})

  p_t_c1 <- sum(c1_ker)/length(c1_ker)
  p_t_c2 <- sum(c2_ker)/length(c2_ker)

  c <- 1
  if(p_t_c1*length(c1_ker)/dim(tra)[1]<p_t_c2*length(c2_ker)/dim(tra)[1]){
    c <- 2
  }
  return(c)
}

# inner cross validation function to choose the h
cv_inner <- function(tr,va,h){

  mismatch <- c()
  # loop each h value
  for(i in 1:length(h)){
    c_v <- sapply(va[,1], function(x){classify(x,h[i],tr)})
    mismatch[i] <- misclass(as.numeric(va[,2]),c_v)
  }
  # return miscalssification rate
  return(mismatch)
}
#outter cross validation to summarize final result
cv_outer <- function(D1,D2,h){

  id1 <- as.numeric(rownames(D1))
```

```
  set.seed(12345)
  id2 <- sample(id1, floor(n*0.25))
  D11 <- dak[id2,]
  id3 <- setdiff(id1,id2)
  D12 <- dak[id3,]

  #cross
  mis1 <- cv_inner(D11,D12,h)
  mis2 <- cv_inner(D12,D11,h)
  # average error
  avg_err <- (mis1+mis2)/2
  h_opt1 <- h[which.min(avg_err)]

  test_err <- c()
  aa <- sapply(D2[,1], function(x){classify(x,h_opt1,D1)})
  test_err[1] <- misclass(as.numeric(D2[,2]),aa)

  # D1 D2 switch
  id1 <- as.numeric(rownames(D2))

  set.seed(12345)
  id2 <- sample(id1, floor(n*0.25))
  D11 <- dak[id2,]
  id3 <- setdiff(id1,id2)
  D12 <- dak[id3,]

  #cross
  mis1 <- cv_inner(D11,D12,h)
  mis2 <- cv_inner(D12,D11,h)
  #average error
  avg_err <- (mis1+mis2)/2
  h_opt2 <- h[which.min(avg_err)]

  aa <- sapply(D1[,1], function(x){classify(x,h_opt2,D2)})
  test_err[2] <- misclass(as.numeric(D1[,2]),aa)

  res <- list('h_opt1'=h_opt1,'h_opt2'=h_opt2,'test_error'=test_err)

  return(res)
}

h <- c(0.5,1,5,10)
res <- cv_outer(D1,D2,h)

cat('Average error on tests is ',mean(res$test_error))

h2 <- seq(0.5,10,0.5)
test <- cv_inner(D11,D12,h2)

plot(h2,test,xlab = 'h value',ylab = 'test error',main = 'Test Error on Each h')
h_opt <- (res$h_opt1+res$h_opt2)/2
h_opt
```