STAT4001 Homework 3

Chan Sze Yuen Syngiene 1155127616

1/12/2021

Q1

(a)

```
library(MASS)
```

```
## Warning: package 'MASS' was built under R version 3.6.3
```

```
data(Boston)
fit.cubic=lm(dis~poly(nox,degree=3),data=Boston)
summary(fit.cubic)
```

```
##
## Call:
## lm(formula = dis ~ poly(nox, degree = 3), data = Boston)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.2094 -0.6112 -0.1121  0.4798  4.9914
##
## Coefficients:
##                        Estimate Std. Error t value Pr(>|t|)
## (Intercept)              3.7950     0.0467  81.264  < 2e-16 ***
## poly(nox, degree = 3)1 -36.3999     1.0505 -34.650  < 2e-16 ***
## poly(nox, degree = 3)2  17.9570     1.0505  17.094  < 2e-16 ***
## poly(nox, degree = 3)3  -6.1479     1.0505  -5.852 8.75e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.05 on 502 degrees of freedom
## Multiple R-squared:  0.7526, Adjusted R-squared:  0.7511
## F-statistic:   509 on 3 and 502 DF,  p-value: < 2.2e-16
```

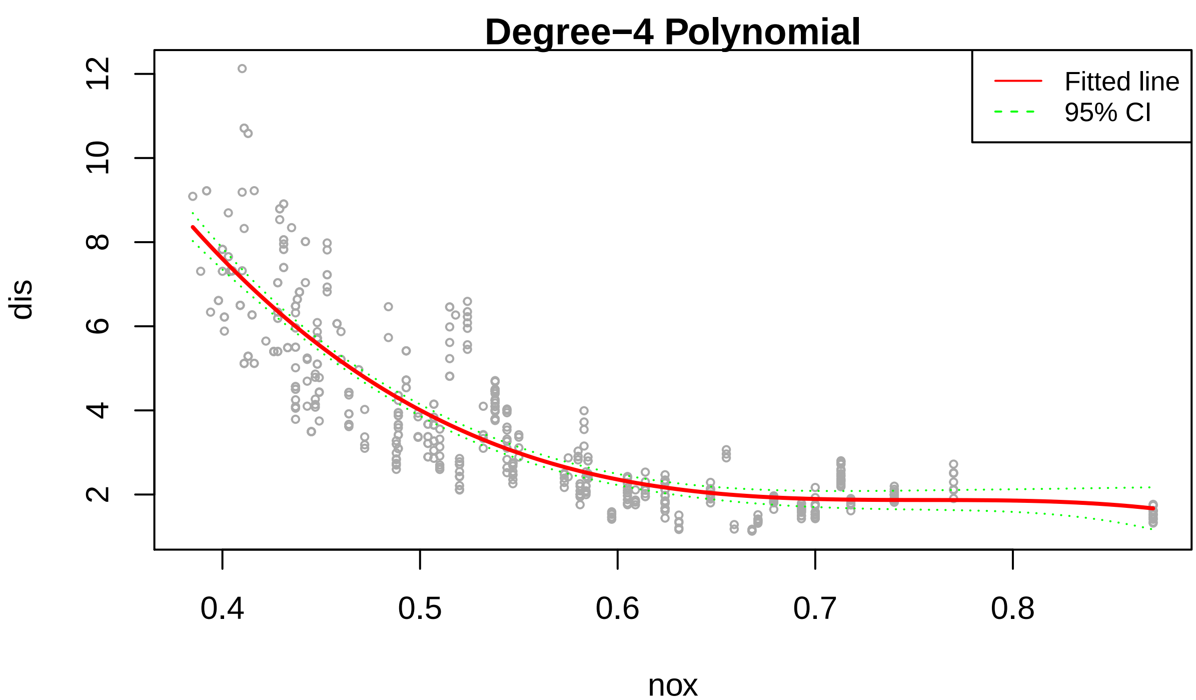
```
yhat=as.vector(fit.cubic$fitted.values) #fitted value
print(yhat)
```

```
##   [1] 3.192096 4.878326 4.878326 5.236743 5.236743 5.236743 3.462278 3.462278
##   [9] 3.462278 3.462278 3.462278 3.462278 3.462278 3.192096 3.192096 3.192096
##  [17] 3.192096 3.192096 3.192096 3.192096 3.192096 3.192096 3.192096 3.192096
```

```
##  [25] 3.192096 3.192096 3.192096 3.192096 3.192096 3.192096 3.192096 3.192096
##  [33] 3.192096 3.192096 3.192096 4.032143 4.032143 4.032143 4.032143 6.356647
##  [41] 6.356647 5.586357 5.586357 5.586357 5.586357 5.586357 5.586357 5.586357
##  [49] 5.586357 5.586357 5.921024 5.921024 5.921024 5.921024 7.135084 7.460601
##  [57] 7.135084 7.089641 5.408669 5.408669 5.408669 5.408669 5.408669 5.408669
##  [65] 6.861947 7.701156 7.701156 7.180791 7.180791 7.180791 6.999541 6.999541
##  [73] 6.999541 6.999541 5.998027 5.998027 5.998027 5.998027 5.998027 5.998027
##  [81] 6.439066 6.439066 6.439066 6.439066 5.550354 5.550354 5.550354 5.550354
##  [89] 4.293638 4.293638 4.293638 4.293638 5.037907 5.037907 5.037907 5.695776
##  [97] 5.695776 5.695776 5.695776 5.695776 3.545735 3.545735 3.545735 3.545735
## [105] 3.545735 3.545735 3.545735 3.545735 3.545735 3.545735 3.545735 3.035701
## [113] 3.035701 3.035701 3.035701 3.035701 3.035701 3.035701 3.035701 3.035701
## [121] 2.556447 2.556447 2.556447 2.556447 2.556447 2.556447 2.556447 2.166882
## [129] 2.166882 2.166882 2.166882 2.166882 2.166882 2.166882 2.166882 2.166882
## [137] 2.166882 2.166882 2.166882 2.166882 2.166882 2.166882 1.669400 1.669400
## [145] 1.669400 1.669400 1.669400 1.669400 1.669400 1.669400 1.669400 1.669400
## [153] 1.669400 1.669400 1.669400 1.669400 1.669400 2.312547 2.312547 1.669400
## [161] 2.312547 2.312547 2.312547 2.312547 2.312547 2.312547 2.312547 2.312547
## [169] 2.312547 2.312547 2.312547 2.312547 3.767052 3.767052 3.767052 3.767052
## [177] 3.767052 3.767052 3.767052 4.320892 4.320892 4.320892 4.320892 4.320892
## [185] 4.320892 4.320892 4.320892 5.998027 5.998027 5.998027 5.998027 5.998027
## [193] 5.998027 7.556012 7.556012 6.606923 7.413299 7.413299 7.413299 7.460601
## [201] 7.460601 6.910485 6.910485 6.861947 6.861947 4.293638 4.293638 4.293638
## [209] 4.293638 4.293638 4.293638 4.293638 4.293638 4.293638 4.293638 4.293638
## [217] 2.986458 2.986458 2.986458 2.986458 3.837061 3.837061 3.837061 3.837061
## [225] 3.908775 3.908775 3.908775 3.908775 3.908775 3.908775 3.908775 3.908775
## [233] 3.837061 3.837061 3.837061 3.837061 3.837061 3.837061 6.356647 6.356647
## [241] 6.356647 6.356647 6.356647 6.356647 6.234888 6.234888 6.234888 6.234888
## [249] 6.234888 6.234888 6.234888 6.234888 6.234888 6.234888 7.998820 7.998820
## [257] 7.898499 2.038531 2.038531 2.038531 2.038531 2.038531 2.038531 2.038531
## [265] 2.038531 2.038531 2.038531 2.628989 2.628989 5.037907 5.037907 5.037907
## [273] 5.037907 5.037907 5.622595 5.622595 5.622595 5.622595 5.622595 5.773635
## [281] 5.773635 5.773635 5.773635 7.556012 7.604123 8.151380 8.358707 7.366265
## [289] 7.366265 7.366265 7.089641 7.089641 7.089641 5.998027 5.998027 5.998027
## [297] 5.998027 5.998027 7.604123 7.604123 7.604123 6.154954 6.154954 6.154954
## [305] 4.785203 4.785203 4.785203 4.785203 3.086374 3.086374 3.086374 3.086374
## [313] 3.086374 3.086374 3.086374 3.086374 3.086374 3.086374 3.086374 3.086374
## [321] 4.186645 4.186645 4.186645 4.186645 4.186645 4.186645 4.186645 4.186645
## [329] 5.169566 5.169566 5.169566 5.963256 5.963256 3.654102 3.654102 3.654102
## [337] 3.654102 3.654102 3.654102 3.654102 3.654102 5.807325 3.588538 4.431955
## [345] 4.431955 5.807325 5.807325 6.315812 6.076002 6.315812 6.315812 7.089641
## [353] 7.089641 7.135084 6.999541 6.999541 1.868172 1.868172 1.868172 1.868172
## [361] 1.868172 1.868172 1.868172 1.868172 1.878133 1.878133 1.878133 2.122625
## [369] 2.122625 2.122625 2.122625 2.122625 1.959851 1.959851 1.959851 1.951183
## [377] 1.951183 1.951183 1.951183 1.951183 1.951183 1.951183 1.894289 1.894289
## [385] 1.894289 1.894289 1.894289 1.894289 1.894289 1.894289 1.894289 1.894289
## [393] 1.894289 1.904046 1.904046 1.904046 1.904046 1.904046 1.904046 1.904046
## [401] 1.904046 1.904046 1.904046 1.904046 1.904046 1.904046 1.989550 1.989550
## [409] 2.385979 2.385979 2.385979 2.385979 2.385979 2.385979 1.904046 1.930852
## [417] 1.930852 1.930852 1.930852 1.878133 1.878133 1.878133 2.238700 2.238700
## [425] 2.521983 1.930852 2.521983 1.930852 1.930852 1.930852 2.521983 2.521983
## [433] 2.521983 1.881495 1.881495 1.870612 1.870612 1.870612 1.870612 1.870612
## [441] 1.870612 1.870612 1.870612 1.870612 1.870612 1.870612 1.870612 1.870612
## [449] 1.881495 1.881495 1.881495 1.881495 1.881495 1.881495 1.881495 1.881495
```

```
## [457] 1.881495 1.881495 1.881495 1.881495 1.881495 1.881495 1.881495 1.881495
## [465] 2.004632 2.004632 2.004632 2.521983 2.568200 2.568200 2.568200 3.303787
## [473] 2.568200 2.238700 2.521983 2.521983 2.238700 2.238700 2.238700 2.238700
## [481] 3.303787 3.303787 3.303787 3.303787 2.533339 2.533339 2.533339 2.533339
## [489] 2.278611 2.278611 2.278611 2.278611 2.278611 2.510758 2.510758 2.510758
## [497] 2.510758 2.510758 2.510758 2.510758 2.510758 2.654263 2.654263 2.654263
## [505] 2.654263 2.654263
```

```
nox.grid=seq(from=range(Boston$nox)[1],to=range(Boston$nox)[2],0.001)
preds=predict(fit.cubic,newdata=list(nox=nox.grid),se=T)
se.bands=cbind(preds$fit-1.96*preds$se.fit,preds$fit+1.96*preds$se.fit) #95% CI of the predicted value
par(mar=c(4.5,4.5,1,1),oma=c(0,0,4,0))
plot(Boston$nox,Boston$dis,cex=.5,col="darkgrey",main="Degree-4 Polynomial"
     ,xlab="nox",ylab="dis")
lines(nox.grid,preds$fit,lwd=2,col='red',type='l')
matlines(nox.grid,se.bands,lwd=1,col="green",lty=3)
legend("topright",legend=c("Fitted line","95% CI"),col=c("red","green"),lty=1:2, cex=0.8)
```



(b) In this part i will use LOOCV and 6-fold CV to see if these two method reach out same optimal degree for polynomial regression model. LOOCV:

```
library(boot)
```

```
## Warning: package 'boot' was built under R version 3.6.3
```

```r
cv_error=c()
df=cbind(Boston$dis,Boston$nox)
df=as.data.frame(df)
colnames(df)=c("dis","nox")
#LOOCV
for (i in (1:10)){
  fit=glm(dis~poly(nox,degree=i),data=Boston)
  cv_error=append(cv_error,cv.glm(df,fit,K=length(Boston$nox))$delta[1])
}
cv_error
```

```
##  [1] 1.824880 1.183421 1.109149 1.110468 1.092519 1.095313 1.070930 1.074281
##  [9] 1.066487 1.046577
```

```r
which(cv_error==min(cv_error))
```

```
## [1] 10
```

Hence based on the LOOCV result, degree 10 will be optimal for our model

6-fold CV:

```r
cv_error=c()
df=cbind(Boston$dis,Boston$nox)
df=as.data.frame(df)
colnames(df)=c("dis","nox")
#LOOCV
for (i in (1:10)){
  fit=glm(dis~poly(nox,degree=i),data=Boston)
  cv_error=append(cv_error,cv.glm(df,fit,K=6)$delta[1])
}
cv_error
```

```
##  [1] 1.824920 1.184805 1.108730 1.120592 1.090632 1.083757 1.082262 1.077524
##  [9] 1.058979 1.062552
```

```r
which(cv_error==min(cv_error))
```

```
## [1] 9
```

Hence based on the 6-fold CV result, degree 10 will be optimal for our model

These two different CV give us same result that the optimal degree will be 10, which is expected as higher degree the fitting will be closer to the data but with higher bias (refer to the Bias-Variance trade off), and CV error is calculated by the distance between the fitted value and real value.

Q2

(a)

```r
set.seed(1155127616)
library(rpart)
library(ISLR)
```

```
## Warning: package 'ISLR' was built under R version 3.6.3
```

```r
data(Carseats)
n=length(Carseats[,1]) #sample size
test_label=sample(1:n,n/2,replace = F)
head(test_label)
```

```
## [1] 151 242 255 159 371   2
```

```r
Carseats_test=Carseats[test_label,]
Carseats_train=Carseats[-test_label,]
head(Carseats_test) ; head(Carseats_train)
```

```
##     Sales CompPrice Income Advertising Population Price ShelveLoc Age Education
## 151 10.49       122     84           8        176   114      Good  57        10
## 242 12.01       136     63           0        160    94    Medium  38        12
## 255  9.58       108    104          23        353   129      Good  37        17
## 159 12.53       142     90           1        189   112      Good  39        10
## 371  7.68       126     41          22        403   119       Bad  42        12
## 2   11.22       111     48          16        260    83      Good  65        10
##     Urban  US
## 151    No Yes
## 242   Yes  No
## 255   Yes Yes
## 159    No Yes
## 371   Yes Yes
## 2     Yes Yes
```

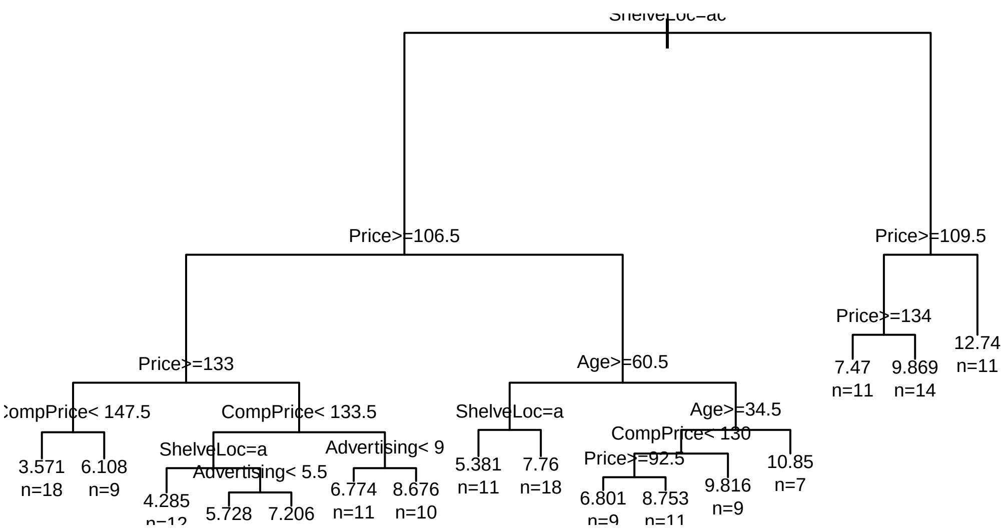
```
##    Sales CompPrice Income Advertising Population Price ShelveLoc Age Education
## 1   9.50       138     73          11        276   120       Bad  42        17
## 3  10.06       113     35          10        269    80    Medium  59        12
## 4   7.40       117    100           4        466    97    Medium  55        14
## 5   4.15       141     64           3        340   128       Bad  38        13
## 7   6.63       115    105           0         45   108    Medium  71        15
## 10  4.69       132    113           0        131   124    Medium  76        17
##    Urban  US
## 1    Yes Yes
## 3    Yes Yes
## 4    Yes Yes
## 5    Yes  No
## 7    Yes  No
## 10    No Yes
```

(b)

```r
tree.carseats = rpart(Sales ~ .,data=Carseats_train)
tree.carseats
```

```
## n= 200
##
## node), split, n, deviance, yval
##       * denotes terminal node
##
##  1) root 200 1569.059000  7.285850
##    2) ShelveLoc=Bad,Medium 164  962.964900  6.686951
##      4) Price>=106.5 99  443.187000  5.818283
##        8) Price>=133 27  101.617400  4.416296
##         16) CompPrice< 147.5 18   52.977690  3.570556 *
##         17) CompPrice>=147.5 9   10.014760  6.107778 *
##        9) Price< 133 72  268.597900  6.344028
##         18) CompPrice< 133.5 51  160.728600  5.794118
##           36) ShelveLoc=Bad 12   35.635100  4.285000 *
##           37) ShelveLoc=Medium 39   89.355310  6.258462
##             74) Advertising< 5.5 25   49.046260  5.727600 *
##             75) Advertising>=5.5 14   20.682720  7.206429 *
##         19) CompPrice>=133.5 21   54.992300  7.679524
##           38) Advertising< 9 11   23.741850  6.773636 *
##           39) Advertising>=9 10   12.293840  8.676000 *
##      5) Price< 106.5 65  331.294200  8.010000
##       10) Age>=60.5 29  124.616900  6.857586
##         20) ShelveLoc=Bad 11   34.073890  5.380909 *
##         21) ShelveLoc=Medium 18   51.898400  7.760000 *
##       11) Age< 60.5 36  137.138700  8.938333
##         22) Age>=34.5 29   88.875820  8.476897
##           44) CompPrice< 130 20   56.396690  7.874500
##             88) Price>=92.5 9   21.133490  6.801111 *
##             89) Price< 92.5 11   16.409620  8.752727 *
##           45) CompPrice>=130 9    9.093422  9.815556 *
##         23) Age< 34.5 7   16.506800 10.850000 *
##    3) ShelveLoc=Good 36  279.296900 10.014170
##      6) Price>=109.5 25  122.987600  8.813600
##       12) Price>=134 11   42.961400  7.470000 *
##       13) Price< 134 14   44.565690  9.869286 *
##      7) Price< 109.5 11   38.379820 12.742730 *
```

```r
plot(tree.carseats)
text(tree.carseats,use.n=T,cex=0.6) #Add text to the tree
```




```r
yhat=predict(tree.carseats,newdata=Carseats_test)
mean((yhat-Carseats$Sales[test_label])^2) #Test MSE
```

```
## [1] 4.586003
```

(c)

```r
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 3.6.3
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```r
rf.carseats=randomForest(Sales~.,data=Carseats_train,importance=TRUE)
yhat.rf=predict(rf.carseats,newdata=Carseats_test)
mean((yhat.rf-Carseats$Sales[test_label])^2) #Test MSE
```

```
## [1] 3.007347
```

```r
importance(rf.carseats)
```

```
##                 %IncMSE IncNodePurity
## CompPrice    8.1628402     127.71289
## Income       4.7991691     133.60770
## Advertising 11.9158899     142.50473
## Population   0.3393804     101.05029
## Price       33.7347652     384.07064
## ShelveLoc   34.3397801     294.26291
## Age         15.3451286     187.59639
## Education    0.4784678      62.90997
## Urban       -0.2835804      15.31623
## US           3.1356219      20.18347
```

Price is the most important variable as it has highest purity.

(d)

```r
dim(Carseats_train) #11 variable, p/3 approximatly 3.66667
```

```
## [1] 200  11
```

```r
#set mtry=2
rf.carseats=randomForest(Sales~.,data=Carseats_train,mtry=3,importance=TRUE)
yhat.rf=predict(rf.carseats,newdata=Carseats_test)
mean((yhat.rf-Carseats$Sales[test_label])^2) #Test MSE
```

```
## [1] 2.903658
```

```r
#set mtry=5
rf.carseats=randomForest(Sales~.,data=Carseats_train,mtry=4,importance=TRUE)
yhat.rf=predict(rf.carseats,newdata=Carseats_test)
mean((yhat.rf-Carseats$Sales[test_label])^2) #Test MSE
```

```
## [1] 2.719888
```

We can see that the test error are both reduced if we use the smaller or larger mtry in the randomForest function.

Q3

(a)

```r
library(ISLR)
data(Hitters)
head(Hitters$Salary) #salary data with na
```

```
## [1]    NA 475.0 480.0 500.0  91.5 750.0
```

```r
Hitters=na.omit(Hitters)
head(Hitters$Salary) #omited na value
```

```
## [1] 475.0 480.0 500.0  91.5 750.0  70.0
```

```r
log_salary=log(Hitters$Salary)
head(log_salary)
```

```
## [1] 6.163315 6.173786 6.214608 4.516339 6.620073 4.248495
```

(b)

```r
Hitters_train=Hitters[1:200,]
Hitters_test=Hitters[-(1:200),]
head(Hitters_train) ; head(Hitters_test)
```

```
##                    AtBat Hits HmRun Runs RBI Walks Years CAtBat CHits CHmRun
## -Alan Ashby          315   81     7   24  38    39    14   3449   835     69
## -Alvin Davis         479  130    18   66  72    76     3   1624   457     63
## -Andre Dawson        496  141    20   65  78    37    11   5628  1575    225
## -Andres Galarraga    321   87    10   39  42    30     2    396   101     12
## -Alfredo Griffin     594  169     4   74  51    35    11   4408  1133     19
## -Al Newman           185   37     1   23   8    21     2    214    42      1
##                    CRuns CRBI CWalks League Division PutOuts Assists Errors
## -Alan Ashby          321  414    375      N        W     632      43     10
## -Alvin Davis         224  266    263      A        W     880      82     14
## -Andre Dawson        828  838    354      N        E     200      11      3
## -Andres Galarraga     48   46     33      N        E     805      40      4
## -Alfredo Griffin     501  336    194      A        W     282     421     25
## -Al Newman            30    9     24      N        E      76     127      7
##                    Salary NewLeague
## -Alan Ashby         475.0         N
## -Alvin Davis        480.0         A
## -Andre Dawson       500.0         N
## -Andres Galarraga    91.5         N
## -Alfredo Griffin    750.0         A
## -Al Newman           70.0         A
```

```
##                  AtBat Hits HmRun Runs RBI Walks Years CAtBat CHits CHmRun
## -Reggie Jackson    419  101    18   65  58    92    20   9528  2510    548
## -Ron Kittle        376   82    21   42  60    35     5   1770   408    115
## -Ray Knight        486  145    11   51  76    40    11   3967  1102     67
## -Rick Leach        246   76     5   35  39    13     6    912   234     12
## -Rick Manning      205   52     8   31  27    17    12   5134  1323     56
## -Rance Mulliniks   348   90    11   50  45    43    10   2288   614     43
##                  CRuns CRBI CWalks League Division PutOuts Assists Errors
## -Reggie Jackson   1509 1659   1342      A        W       0       0      0
## -Ron Kittle        238  299    157      A        W       0       0      0
## -Ray Knight        410  497    284      N        E      88     204     16
## -Rick Leach        102   96     80      A        E      44       0      1
## -Rick Manning      643  445    459      A        E     155       3      2
```

```
## -Rance Mulliniks   295  273    269       A         E       60      176       6
##                  Salary NewLeague
## -Reggie Jackson   487.5         A
## -Ron Kittle       425.0         A
## -Ray Knight       500.0         A
## -Rick Leach       250.0         A
## -Rick Manning     400.0         A
## -Rance Mulliniks  450.0         A
```

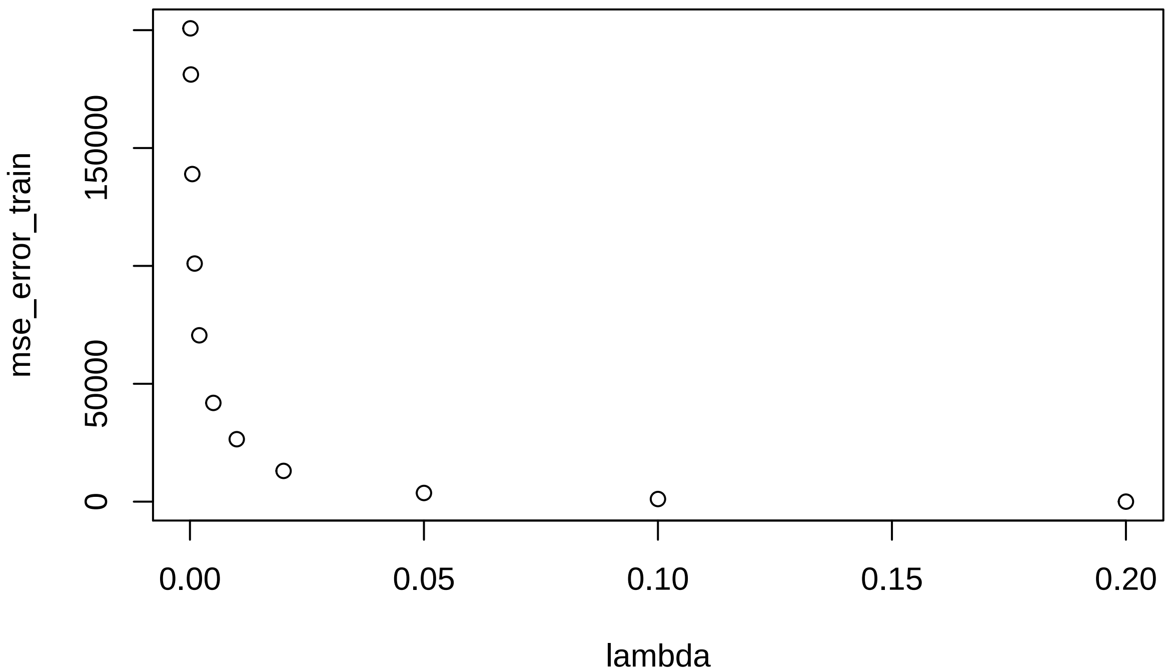
(c)

```r
library(MASS)
library(gbm)
```

```
## Warning: package 'gbm' was built under R version 3.6.3
```

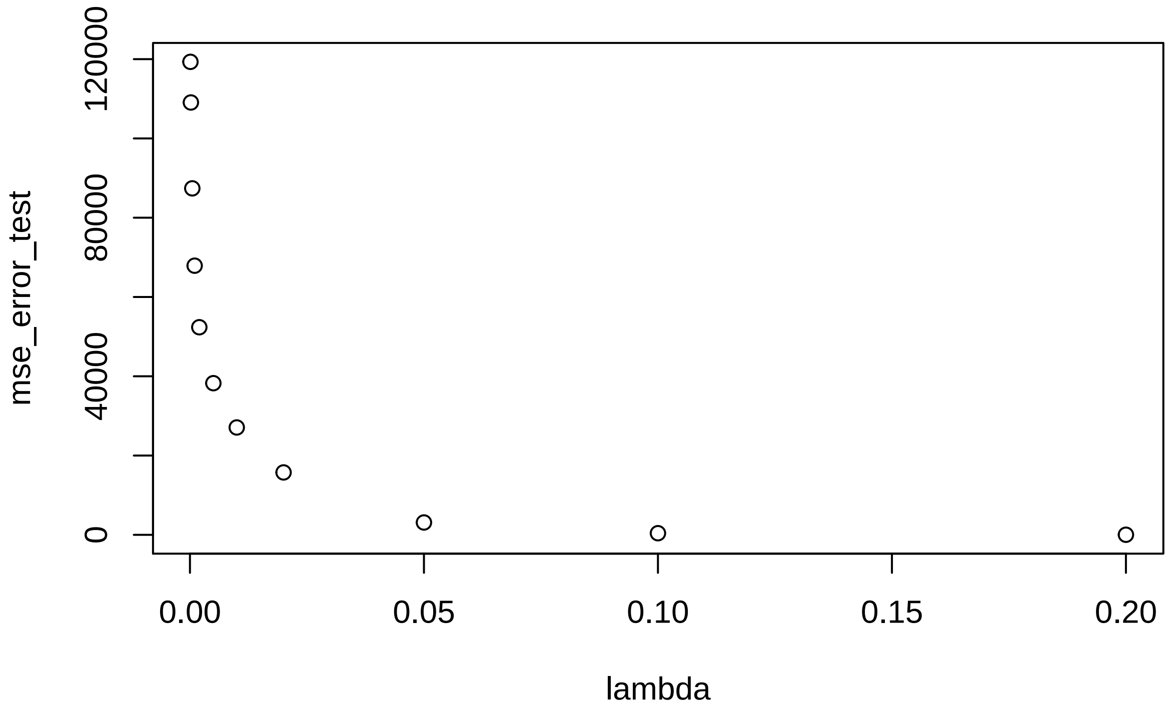
```
## Loaded gbm 2.1.8
```

```r
set.seed(1155127616)
lambda=c(0.0001, 0.0002, 0.0005, 0.001, 0.002, 0.005, 0.01, 0.02, 0.05, 0.1, 0.2)
mse_error_train=c()
for (i in 1:length(lambda)){
boost_train=gbm(Salary~.,data=Hitters_train,distribution="gaussian",n.trees=1000,interaction.depth=4,sh
yhat=predict(boost_train,newdata=Hitters_train,n.trees=1000)
mse_error_train=append(mse_error_train,mean((yhat-Hitters_train$Salary)^2))
}
plot(lambda,mse_error_train,xlab="lambda",ylab="mse_error_train")
```



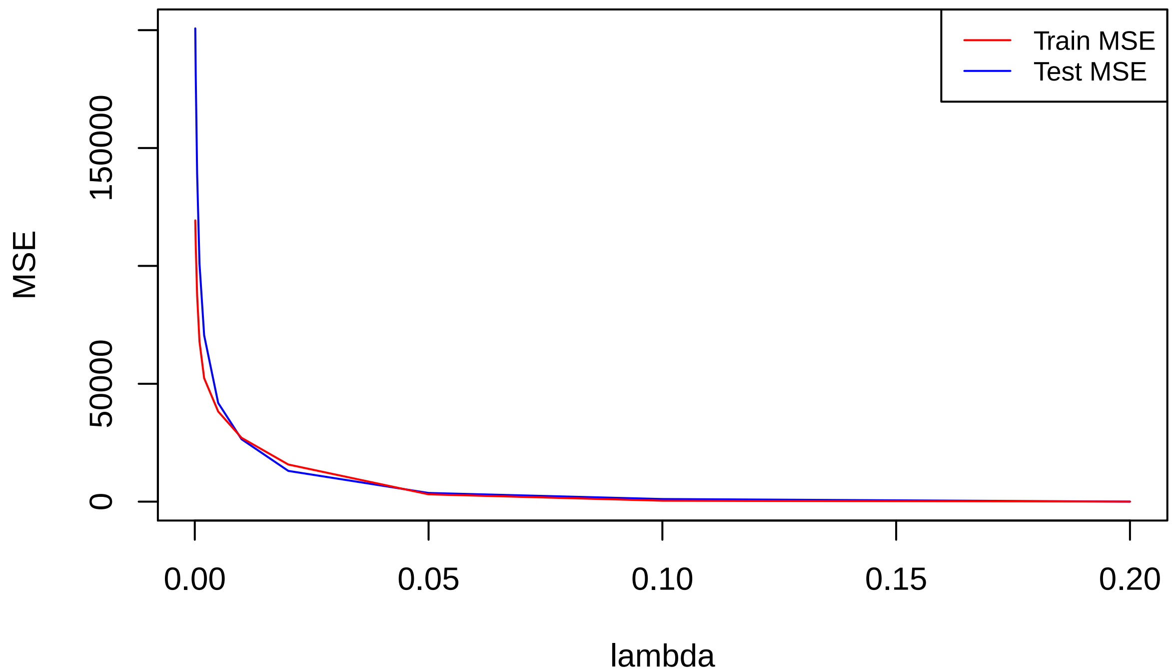
(d)

```
library(MASS)
library(gbm)
set.seed(1155127616)
lambda=c(0.0001, 0.0002, 0.0005, 0.001, 0.002, 0.005, 0.01, 0.02, 0.05, 0.1, 0.2)
mse_error_test=c()
for (i in 1:length(lambda)){
boost_test=gbm(Salary~.,data=Hitters_test,distribution="gaussian",n.trees=1000,interaction.depth=4,shri
yhat=predict(boost_test,newdata=Hitters_test,n.trees=1000)
mse_error_test=append(mse_error_test,mean((yhat-Hitters_test$Salary)^2))
}
plot(lambda,mse_error_test,xlab="lambda",ylab="mse_error_test")
```



#Comparsion

```
plot(lambda,mse_error_train,xlab="lambda",ylab="MSE",type="l",col='blue')
lines(lambda,mse_error_test,col='red')
legend("topright",legend=c("Train MSE","Test MSE"),col=c("red","blue"),lty=1:1, cex=0.8)
```



(e)

```
library(randomForest)
rf.Hitters=randomForest(Salary~.,data=Hitters_train,importance=TRUE)
yhat.rf=predict(rf.Hitters,newdata=Hitters_test)
mean((yhat.rf-Hitters$Salary[-(1:200)])^2) #Test MSE
```

```
## [1] 48998.78
```

```
mse_error_test #MSE in part d
```

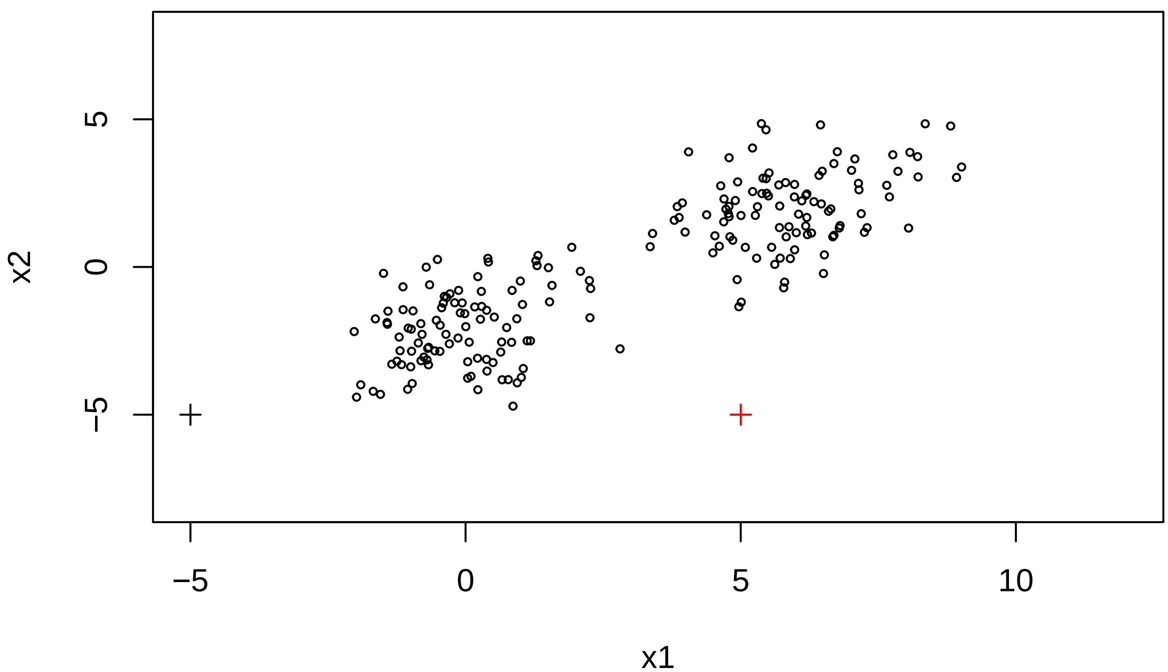
```
##  [1] 119317.19356 109069.01549  87404.08452  67893.56433  52369.17447
##  [6]  38252.66423  27079.43722  15739.18698   3114.27134    407.77844
## [11]     19.90016
```

We can see that the mse error of the random forest is around 50000, which is simuliar to the mse when the shrinkage value=0.002 (mse=52369.17447).

4

(a)

```r
library(MASS)
Sigma <- matrix(c(1.5, 0.4, 0.4, 1.5),nrow=2)
mu1 <- c(0, -2)
mu2 <- c(6, 2)
n <- 200
d1 <- mvrnorm(n = n/2, mu=mu1, Sigma=Sigma)
d2 <- mvrnorm(n = n/2, mu=mu2, Sigma=Sigma)
x <- rbind(d1, d2)
plot(x, pch=1, cex=0.5, col=1, xlab="x1", ylab="x2", xlim=c(-5, 12), ylim=c(-8, 8))
m1=c(-5, -5)
m2=c(5, -5)
points(x=m1[1], y=m1[2], pch=3, cex=1, col='black')
points(x=m2[1], y=m2[2], pch=3, cex=1, col='red')
```

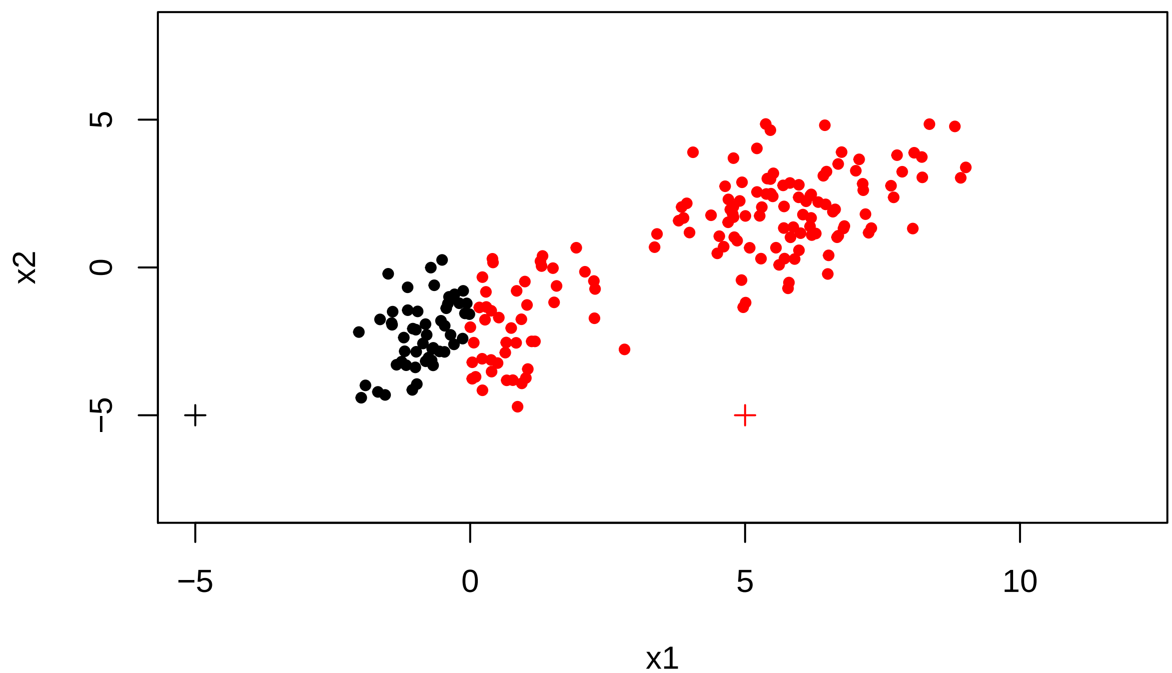


The two plus signs are the initialized cluster centers.

(b)

```r
#first step
m1=c(-5, -5)
m2=c(5, -5)
c1=c(m1[1],m1[2])
c2=c(m2[1],m2[2])
plot(x, pch=1, cex=0.5, col=1, xlab="x1", ylab="x2", xlim=c(-5, 12),
     ylim=c(-8, 8))
points(x=m1[1], y=m1[2], pch=3, cex=1, col='black')
```

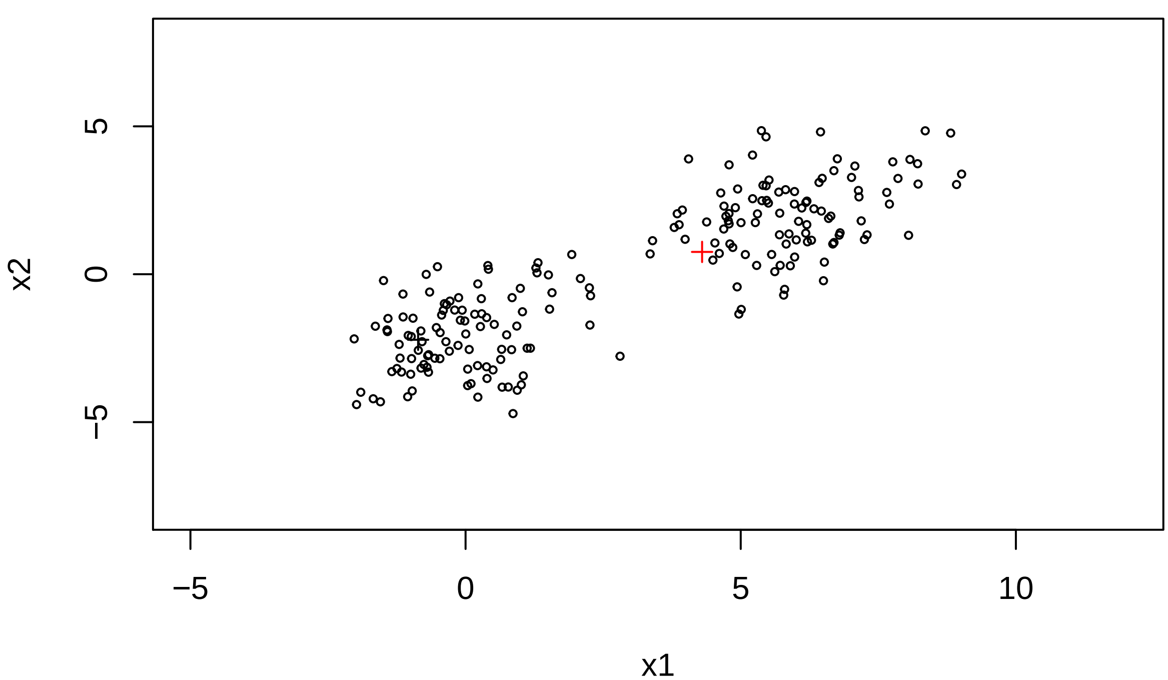
```
points(x=m2[1], y=m2[2], pch=3, cex=1, col='red')
for (i in 1:length(x[,1])){
  if (sqrt(((x[i,1]-c1[1])^2+(x[i,2]-c1[2])^2))<(sqrt((x[i,1]-c2[1])^2+(x[i,2]-c2[2])^2))){
  points(x=x[i,1], y=x[i,2], pch=20, cex=1, col='black')
  }else{
  points(x=x[i,1], y=x[i,2], pch=20, cex=1, col='red')
}
}
```



(c) Now we take the mean of the assigned point to determine the new cluster:

```
x_c1=c()
y_c1=c()
x_c2=c()
y_c2=c()
for (i in 1:length(x[,1])){
  if (sqrt(((x[i,1]-c1[1])^2+(x[i,2]-c1[2])^2))<(sqrt((x[i,1]-c2[1])^2+(x[i,2]-c2[2])^2))){
  x_c1[i]=x[i,1]
  y_c1[i]=x[i,2]
  }else{
  x_c2[i]=x[i,1]
  y_c2[i]=x[i,2]
}
}
xy_c1=cbind(x_c1,y_c1) ; xy_c2=cbind(x_c2,y_c2)
```

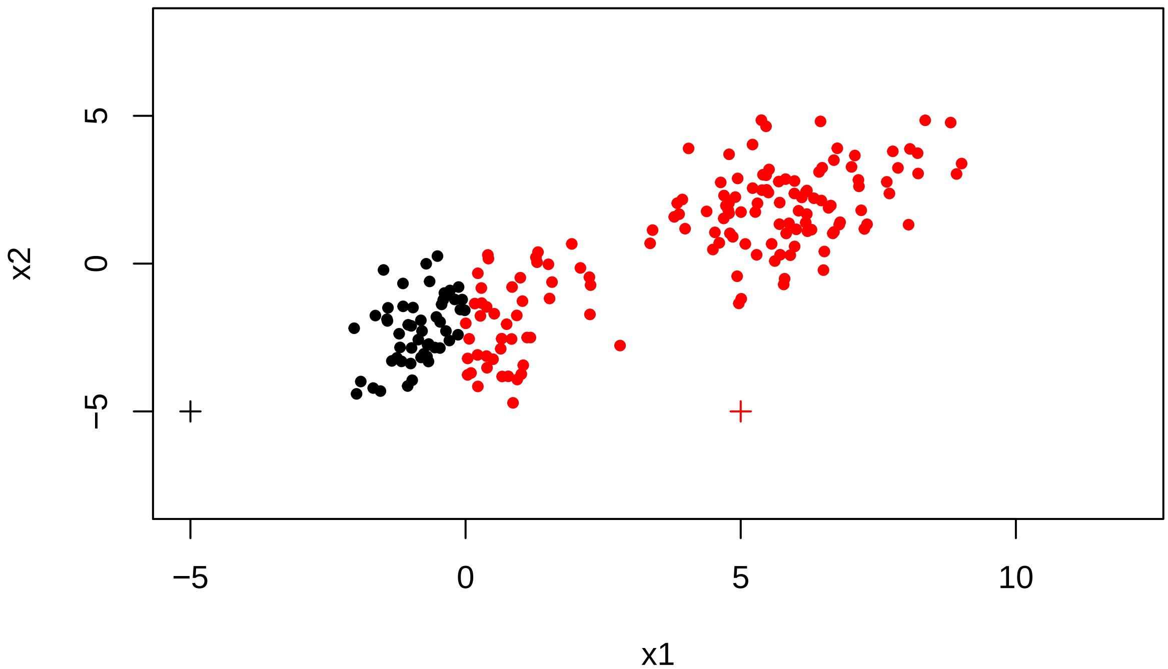
```
xy_c1=na.omit(xy_c1) ; xy_c2=na.omit(xy_c2)
c1=colMeans(xy_c1) ; c2=colMeans(xy_c2) #two new updated clusters
plot(x, pch=1, cex=0.5, col=1, xlab="x1", ylab="x2", xlim=c(-5, 12),
     ylim=c(-8, 8))
points(x=c1[1], y=c1[2], pch=3, cex=1, col='black')
points(x=c2[1], y=c2[2], pch=3, cex=1, col='red')
```



(d) In this question i will stop the iteration if the distance between the updated cluster 1 and the older cluster 1 is less than tolerance AND the distance between the updated cluster 2 and the older cluster 2 is less than tolerance

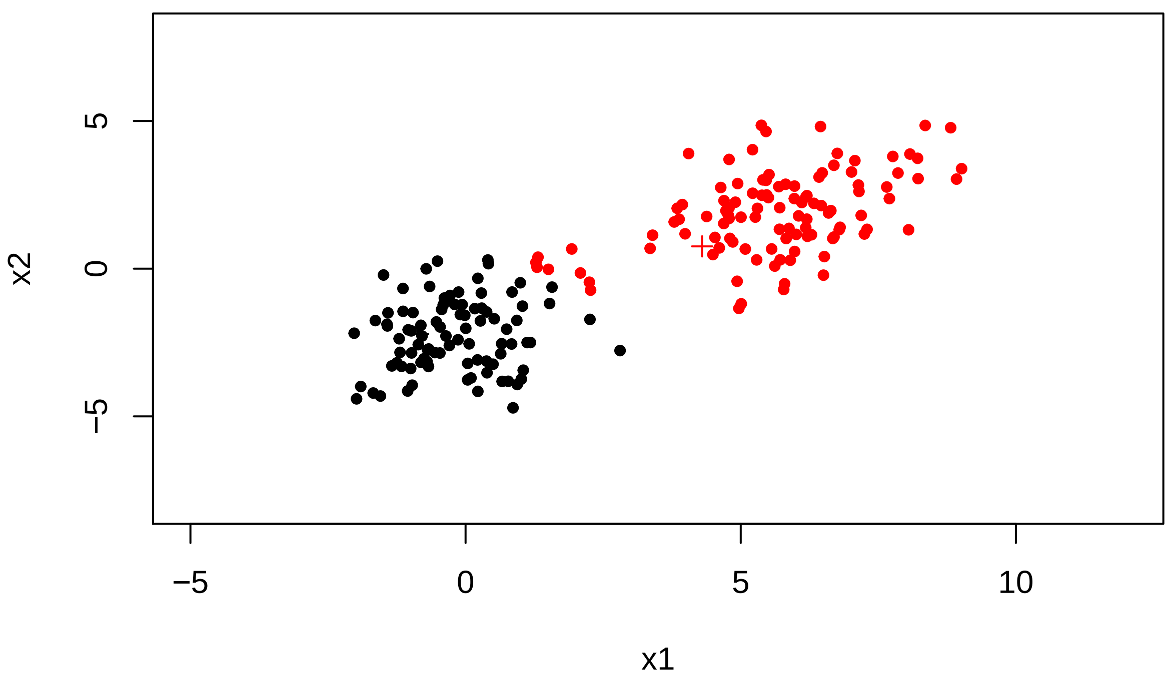
```
#first step
m1=c(-5, -5)
m2=c(5, -5)
c1=c(m1[1],m1[2])
c2=c(m2[1],m2[2])
plot(x, pch=1, cex=0.5, col=1, xlab="x1", ylab="x2", xlim=c(-5, 12),
     ylim=c(-8, 8))
points(x=m1[1], y=m1[2], pch=3, cex=1, col='black')
points(x=m2[1], y=m2[2], pch=3, cex=1, col='red')
for (i in 1:length(x[,1])){
  if (sqrt(((x[i,1]-c1[1])^2+(x[i,2]-c1[2])^2))<(sqrt((x[i,1]-c2[1])^2+(x[i,2]-c2[2])^2))){
  points(x=x[i,1], y=x[i,2], pch=20, cex=1, col='black')
  }else{
  points(x=x[i,1], y=x[i,2], pch=20, cex=1, col='red')
```

```
}
}
```

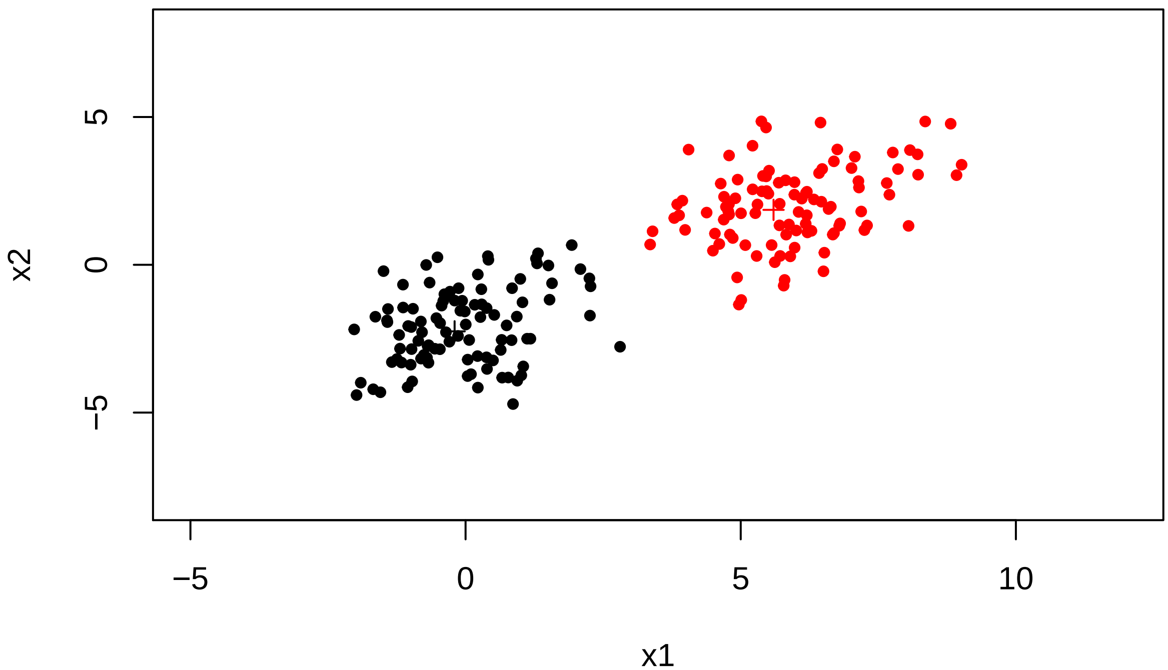



```
tol=10^-4 #for stopping the looping
repeat{
#Updating cluser
x_c1=c()
y_c1=c()
x_c2=c()
y_c2=c()
for (i in 1:length(x[,1])){
  if (sqrt(((x[i,1]-c1[1])^2+(x[i,2]-c1[2])^2))<(sqrt((x[i,1]-c2[1])^2+(x[i,2]-c2[2])^2))){
  x_c1[i]=x[i,1]
  y_c1[i]=x[i,2]
  }else{
  x_c2[i]=x[i,1]
  y_c2[i]=x[i,2]
}
}
xy_c1=cbind(x_c1,y_c1) ; xy_c2=cbind(x_c2,y_c2)
xy_c1=na.omit(xy_c1) ; xy_c2=na.omit(xy_c2)
c1old=c1 ; c2old=c2
c1=colMeans(xy_c1) ; c2=colMeans(xy_c2) #two new updated clusters

#Updating points assignment
```

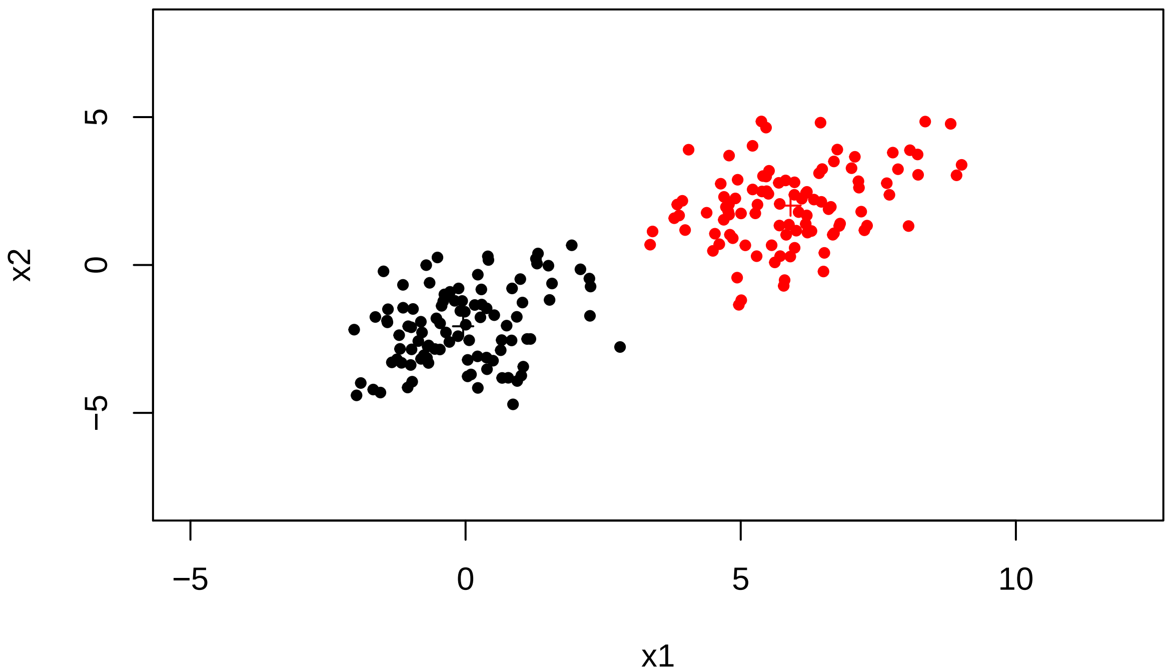
```r
plot(x, pch=1, cex=0.5, col=1, xlab="x1", ylab="x2", xlim=c(-5, 12),
     ylim=c(-8, 8))
points(x=c1[1], y=c1[2], pch=3, cex=1, col='black')
points(x=c2[1], y=c2[2], pch=3, cex=1, col='red')

for (i in 1:length(x[,1])){
  if (sqrt(((x[i,1]-c1[1])^2+(x[i,2]-c1[2])^2))<(sqrt((x[i,1]-c2[1])^2+(x[i,2]-c2[2])^2))){
  points(x=x[i,1], y=x[i,2], pch=20, cex=1, col='black')
  }else{
  points(x=x[i,1], y=x[i,2], pch=20, cex=1, col='red')
}
}
d1_updated=sqrt((c1old[1]-c1[1])^2+(c1old[2]-c1[2])^2) #distance between old cluster 1 and updated clus
d2_updated=sqrt((c2old[1]-c2[1])^2+(c2old[2]-c2[2])^2) #distance between old cluster 2 and updated clus
names(d1_updated)=c('diff_c1')
names(d2_updated)=c('diff_c2')

print(c(d1_updated,d2_updated))
  if(d1_updated<tol && d2_updated<tol){
    break
  }
}
```




```
##  diff_c1  diff_c2
## 4.986791 5.797375
```




```
##   diff_c1   diff_c2
## 0.6641728 1.7041671
```




```
##    diff_c1   diff_c2
## 0.2375535 0.3424976
```

```
## diff_c1 diff_c2
##       0       0
```

compute and print the objective function in each iteration. -5