lab08

Yixuan Li, UPI:yil845

3 October 2021

The Data: The data source for this lab is from UCI Machine Learning Repository,the Vertebral Column data set.

Import data

```
library(MASS)
library(class)
library(gam)
```

```
## Loading required package: splines
```

```
## Loading required package: foreach
```

```
## Loaded gam 1.20
```

```
library(nnet)
library(e1071)
library(parallel)
vc = read.csv("vertebral-column.csv", stringsAsFactors=TRUE)
head(vc)
```

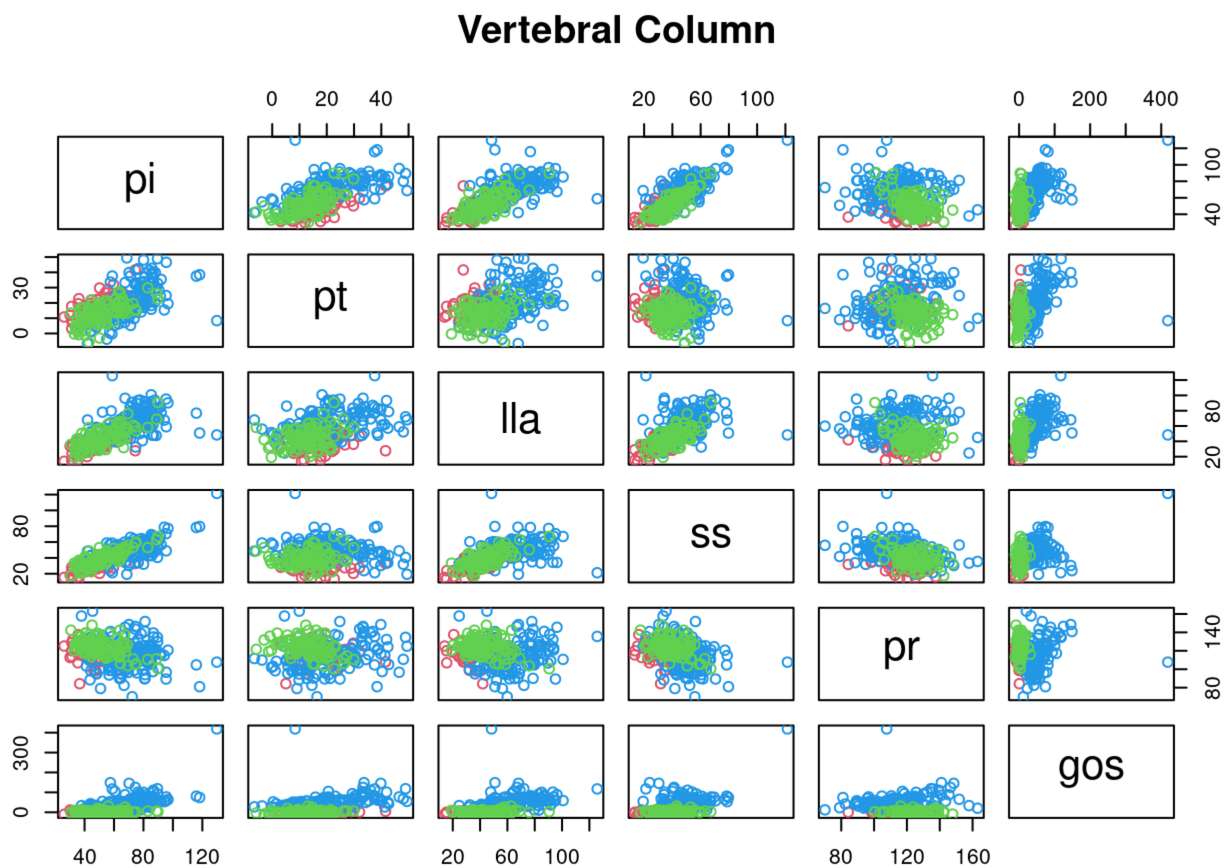
```
table(vc$class)
```

```
##
##  DH  NO  SL
##  60 100 150
```

Exploration

1,These codes create a pairwise scatterplot, with observations of different classes shown in different colors.

```
plot(vc[,-7], col=as.numeric(vc[,7])+1, pch=1, main="Vertebral Column" )
```



Classification

Five classification methods are done with underlying codes. Predictions and models are developed based on orginal datasets. The confusion matrix and (resubstitution) classification accuracy are computed as following:

2,Linear discriminant analysis.

```
r.ld = lda(class ~ ., data=vc)
yhat.ld = predict(r.ld, newdata=vc)$class
table(vc$class, yhat.ld)          # confusion table
```

```
##     yhat.ld
##        DH  NO  SL
##    DH  39  20   1
##    NO  12  79   9
##    SL   4  11 135
```

```
(A.ld = mean(vc$class == yhat.ld))     # Classfication accuracy
```

```
## [1] 0.816129
```

3, Quadratic discriminant analysis.

```
(r.qd = qda(class ~ ., data=vc))
```

```
## Call:
## qda(class ~ ., data = vc)
##
## Prior probabilities of groups:
##        DH        NO        SL
## 0.1935484 0.3225806 0.4838710
##
## Group means:
##          pi       pt      lla       ss       pr       gos
## DH 47.63833 17.39867 35.46350 30.24000 116.4750  2.479333
## NO 51.68560 12.82180 43.54230 38.86380 123.8912  2.187000
## SL 71.51367 20.74800 64.10987 50.76613 114.5183 51.896867
```

```
(yhat.qd = predict(r.qd, newdata=vc)$class)
```

```
##   [1] DH DH NO DH DH NO NO NO NO SL DH DH DH DH DH NO NO NO NO DH NO DH SL DH DH
##  [26] DH DH DH DH NO DH DH DH NO DH DH NO DH DH DH DH NO DH DH NO DH DH DH DH DH
##  [51] DH DH DH DH DH DH DH DH DH NO SL SL SL SL SL SL SL SL NO SL SL SL SL SL SL
##  [76] SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL
## [101] SL SL SL SL SL DH SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL
## [126] SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL
## [151] SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL
## [176] SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL
## [201] SL SL SL SL SL SL SL SL SL SL DH NO NO NO NO NO NO NO NO DH DH NO SL SL SL
## [226] NO NO NO NO NO NO NO NO NO NO NO NO NO NO NO NO NO NO NO DH NO DH DH NO DH
## [251] NO NO NO NO NO SL NO NO NO NO NO NO NO NO NO NO NO NO NO DH NO DH DH NO NO
## [276] NO NO NO NO NO NO NO NO NO NO NO NO NO NO NO DH DH NO NO NO NO NO DH NO NO
## [301] NO NO DH NO NO NO DH NO NO NO
## Levels: DH NO SL
```

```
table(vc$class, yhat.qd)     # confusion table
```

```
##     yhat.qd
##        DH  NO  SL
##   DH   42  16   2
##   NO   15  81   4
##   SL    1   1 148
```

```
(A.qd = mean(vc$class == yhat.qd))    # Classfication accuracy
```

```
## [1] 0.8741935
```

4,Naive Bayes.

```
(r.nb = naiveBayes(class ~ ., data=vc))
```

```
##
## Naive Bayes Classifier for Discrete Predictors
##
## Call:
## naiveBayes.default(x = X, y = Y, laplace = laplace)
##
## A-priori probabilities:
## Y
##        DH        NO        SL
## 0.1935484 0.3225806 0.4838710
##
## Conditional probabilities:
##     pi
## Y        [,1]     [,2]
##   DH 47.63833 10.69694
##   NO 51.68560 12.36790
##   SL 71.51367 15.10908
##
##     pt
## Y        [,1]      [,2]
##   DH 17.39867  7.016809
##   NO 12.82180  6.778658
##   SL 20.74800 11.505857
##
##     lla
## Y        [,1]      [,2]
##   DH 35.46350  9.767804
##   NO 43.54230 12.361581
##   SL 64.10987 16.396336
##
##     ss
## Y        [,1]      [,2]
##   DH 30.24000  7.555102
##   NO 38.86380  9.623776
##   SL 50.76613 12.318389
##
##     pr
## Y        [,1]      [,2]
##   DH 116.4750  9.355709
##   NO 123.8912  9.013755
##   SL 114.5183 15.580436
##
##     gos
## Y         [,1]      [,2]
##   DH  2.479333  5.531449
##   NO  2.187000  6.307020
##   SL 51.896867 40.107622
```

```
(yhat.nb = predict(r.nb, newdata=vc))   # predicted classes
```

```
##   [1] DH DH NO SL DH DH NO NO NO DH DH DH DH DH DH DH NO NO DH DH NO NO SL DH DH
##  [26] DH DH DH DH NO DH DH DH DH DH DH NO DH DH DH DH DH DH NO NO DH DH DH DH DH
##  [51] DH SL DH DH DH DH DH DH DH DH SL SL SL SL SL SL SL SL NO SL SL SL SL SL SL
##  [76] SL SL SL SL NO SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL
## [101] SL SL SL SL SL NO SL SL SL SL SL SL SL SL SL NO SL SL NO SL SL SL SL SL SL
## [126] SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL
## [151] SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL
## [176] SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL SL
## [201] SL SL SL SL SL SL SL SL SL SL DH NO NO NO NO NO NO NO NO DH DH NO SL SL SL
## [226] NO NO NO DH NO NO NO NO NO NO NO NO NO NO NO DH NO NO NO NO NO DH DH NO DH
## [251] NO NO NO SL SL SL NO NO NO NO NO SL DH NO NO NO NO NO NO DH NO DH DH NO DH
## [276] NO NO NO NO NO DH NO NO NO NO NO NO NO SL NO DH DH NO NO NO NO DH DH NO SL
## [301] NO SL DH NO NO DH DH NO NO NO
## Levels: DH NO SL
```

```
table(vc$class, yhat.nb)     # confusion table
```

```
##     yhat.nb
##       DH  NO  SL
##   DH  45  12   3
##   NO  21  69  10
##   SL   0   5 145
```

```
(A.nb = mean(vc$class == yhat.nb))    # Classfication accuracy
```

```
## [1] 0.8354839
```

```
## result is hidden due to long list
(yhat.nb = predict(r.nb, newdata=vc, type="raw"))   # posterior probabilities
```

5, Multinomial logistic regression.

```
(r.mlr = multinom(class ~ ., data=vc))
```

```
## # weights:  24 (14 variable)
## initial  value 340.569809
## iter  10 value 182.252739
## iter  20 value 91.500126
## iter  30 value 89.599802
## iter  40 value 89.590489
## iter  50 value 89.584980
## iter  60 value 89.544492
## iter  70 value 89.544275
## iter  70 value 89.544275
## final  value 89.544264
## converged
```

```
## Call:
## multinom(formula = class ~ ., data = vc)
##
## Coefficients:
##    (Intercept)        pi          pt        lla         ss          pr
## NO   -20.23244 -4.584673   4.485069 0.03527065   4.737024 0.13049227
## SL   -21.71458 16.597946 -16.609482 0.02184513 -16.390098 0.07798643
##             gos
## NO -0.005418782
## SL  0.309521250
##
## Residual Deviance: 179.0885
## AIC: 207.0885
```

```
table(vc$class, predict(r.mlr, vc))    # Confusion table
```

```
##
##       DH  NO  SL
##   DH  40  19   1
##   NO  13  85   2
##   SL   1   3 146
```

```
A.mlr<- mean(vc$class == predict(r.mlr, vc))   # classification accuracy
A.mlr
```

```
## [1] 0.8741935
```

6, K-nearest neighbours (with K=10).

```
yhat.knn = knn(train=vc[,1:6], test=vc[,1:6], cl=vc[,7], k=10)   # K = 10
t<-table(vc[,7], yhat.knn)
t  # Confusion table
```

```
##     yhat.knn
##       DH  NO  SL
##   DH  43  16   1
##   NO  12  86   2
##   SL   1   3 146
```

```
A.knn<-sum(diag(t))/sum(t) # classification accuracy
A.knn
```

```
## [1] 0.8870968
```

Primary Performance Evaluation

7,Present the resulting classification accuracy for all five classification methods in a table.

From this table, what can we say about the relative performance of these methods for the data set? The result shows that “Quadratic discriminant analysis” “Multinomial logistic regression” and “K-nearest neighbours” have the better classification accuracy. KNN gives highest accuracy.

```
m <- matrix(0,5,2)
colnames(m)<-c("Method","Classification accuracy")
m[,1] = c("Linear discriminant analysis","Quadratic discriminant analysis","Naive Bayes","Multin
omial logistic regression","K-nearest neighbours")
m[,2] = c(A.ld,A.qd,A.nb,A.mlr,A.knn)
m
```

```
##      Method                            Classification accuracy
## [1,] "Linear discriminant analysis"    "0.816129032258065"
## [2,] "Quadratic discriminant analysis" "0.874193548387097"
## [3,] "Naive Bayes"                     "0.835483870967742"
## [4,] "Multinomial logistic regression" "0.874193548387097"
## [5,] "K-nearest neighbours"            "0.887096774193548"
```

Multiple Logistic Regression and Generalised Additive Models

8,Create a response variable from variable class so that both classes DH and SL are relabelled as AB (Abnormal).

Use glm() to build a multiple logistic regression model for this new class variable, and compute the confusion matrix and resubstitution classification accuracy.

```
# create a new column called newclass
vc$newclass <- vc$class
# change levels of "DH" and "SL" in column newclass into "AB"
levels(vc$newclass)[levels(vc$newclass)=="DH" | levels(vc$newclass)=="SL" ] <- "AB"

# glm.fit: algorithm did not converge glm.fit: fitted probabilities numerically 0 or 1 occurred
(r.glm = glm(newclass ~ pi+pt+lla+ss+pr+gos, data=vc, family=binomial))
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
##
## Call:  glm(formula = newclass ~ pi + pt + lla + ss + pr + gos, family = binomial,
##     data = vc)
##
## Coefficients:
## (Intercept)           pi           pt          lla           ss           pr
##   -15.14270     12.88787    -12.96465      0.01939    -12.79191      0.10684
##         gos
##    -0.16794
##
## Degrees of Freedom: 309 Total (i.e. Null);  303 Residual
## Null Deviance:       389.9
## Residual Deviance: 178.1     AIC: 192.1
```

```
yhat = as.numeric(predict(r.glm, vc, type="response") > 0.5)+1       # Predicted classes
table(vc$newclass, levels(vc$newclass)[yhat])                          # Confusion table
```

```
##
##       AB  NO
##   AB 188  22
##   NO  22  78
```

```
mean((vc$newclass) == levels(vc$newclass)[yhat])                       # classification accuracy
```

```
## [1] 0.8580645
```

9,Use step() to find the AIC-selected model, with backward selection.

Which variables are removed by the AIC? The variables lla & ss are removed by AIC backward selection. The variables “pi”, “pt”, “pr” and “gos” show very significiant P values with "***".

```
rstep = step(r.glm, type="backward")    # backward selection. Only AIC
```

```
## Start:  AIC=192.13
## newclass ~ pi + pt + lla + ss + pr + gos
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
##        Df Deviance    AIC
## - ss    1   178.27 190.27
## - pi    1   178.27 190.27
## - pt    1   178.27 190.27
## - lla   1   178.86 190.86
## <none>      178.13 192.13
## - pr    1   209.31 221.31
## - gos   1   302.31 314.31
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
##
## Step:  AIC=190.27
## newclass ~ pi + pt + lla + pr + gos
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
##        Df Deviance    AIC
## - lla   1   178.95 188.95
## <none>      178.27 190.27
## - pi    1   189.08 199.08
## - pt    1   202.23 212.23
## - pr    1   209.36 219.36
## - gos   1   303.45 313.45
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
##
## Step:  AIC=188.95
## newclass ~ pi + pt + pr + gos
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
##        Df Deviance    AIC
## <none>      178.95 188.95
## - pt    1   208.55 216.55
## - pr    1   212.18 220.18
## - pi    1   212.53 220.53
## - gos   1   310.54 318.53
```

```
summary(rstep)
```

```
##
## Call:
## glm(formula = newclass ~ pi + pt + pr + gos, family = binomial,
##     data = vc)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.21933  -0.37927  -0.03193   0.40184   2.79266
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -15.45723    3.27273  -4.723 2.32e-06 ***
## pi            0.11495    0.02266   5.072 3.94e-07 ***
## pt           -0.18140    0.03784  -4.794 1.63e-06 ***
## pr            0.10895    0.02279   4.780 1.75e-06 ***
## gos          -0.16538    0.02288  -7.227 4.92e-13 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 389.86  on 309  degrees of freedom
## Residual deviance: 178.95  on 305  degrees of freedom
## AIC: 188.95
##
## Number of Fisher Scoring iterations: 8
```

10,Extend the AIC-selected model with gam() so that each linear term is replaced with a smoothing spline of 5 degrees of freedom. Take a visual approach to reasonably lower the degrees of freedom in each term.

Visually speaking 2-3 degrees of freedom are the best, smoothly fitted. the computed confusion matrix and classification accuracy are based on 2 degrees of freedom.

```
(r1 = gam(newclass ~ s(pi, 1) + s(pt, 1) + s(pr, 1) + s(gos,1), data=vc, family=binomial()))
```

```
## Call:
## gam(formula = newclass ~ s(pi, 1) + s(pt, 1) + s(pr, 1) + s(gos,
##     1), family = binomial(), data = vc)
##
## Degrees of Freedom: 309 total; 304.9999 Residual
## Residual Deviance: 178.9455
```

```
(r2 = gam(newclass ~ s(pi, 2) + s(pt, 2) + s(pr, 2) + s(gos,2), data=vc, family=binomial()))
```

```
## Call:
## gam(formula = newclass ~ s(pi, 2) + s(pt, 2) + s(pr, 2) + s(gos,
##     2), family = binomial(), data = vc)
##
## Degrees of Freedom: 309 total; 300.9999 Residual
## Residual Deviance: 163.5696
```

```
(r3 = gam(newclass ~ s(pi, 3) + s(pt, 3) + s(pr, 3) + s(gos,3), data=vc, family=binomial()))
```

```
## Call:
## gam(formula = newclass ~ s(pi, 3) + s(pt, 3) + s(pr, 3) + s(gos,
##     3), family = binomial(), data = vc)
##
## Degrees of Freedom: 309 total; 297.0002 Residual
## Residual Deviance: 158.5306
```

```
(r4 = gam(newclass ~ s(pi, 4) + s(pt, 4) + s(pr, 4) + s(gos,4), data=vc, family=binomial()))
```

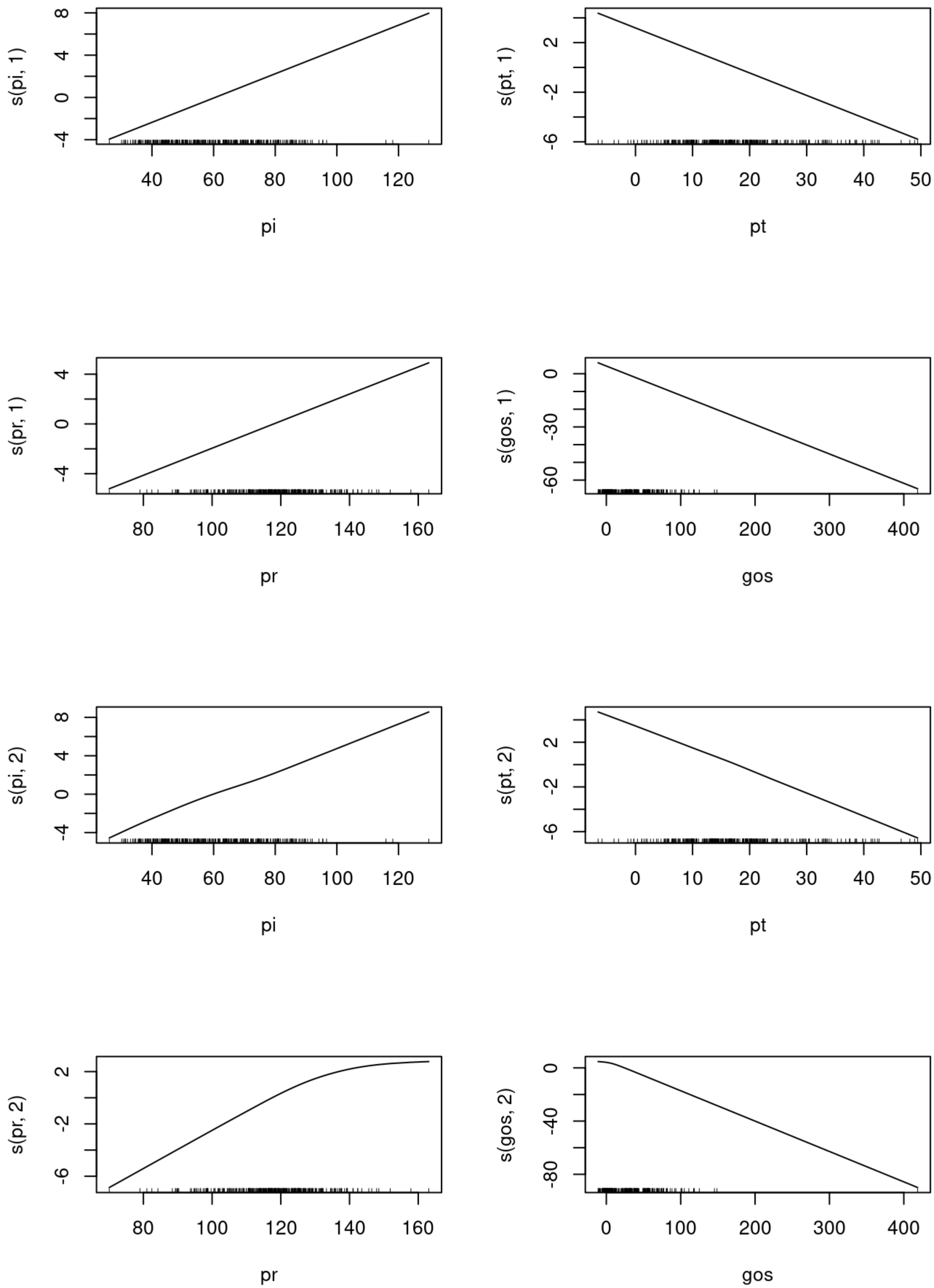
```
## Call:
## gam(formula = newclass ~ s(pi, 4) + s(pt, 4) + s(pr, 4) + s(gos,
##     4), family = binomial(), data = vc)
##
## Degrees of Freedom: 309 total; 293.0001 Residual
## Residual Deviance: 154.4163
```

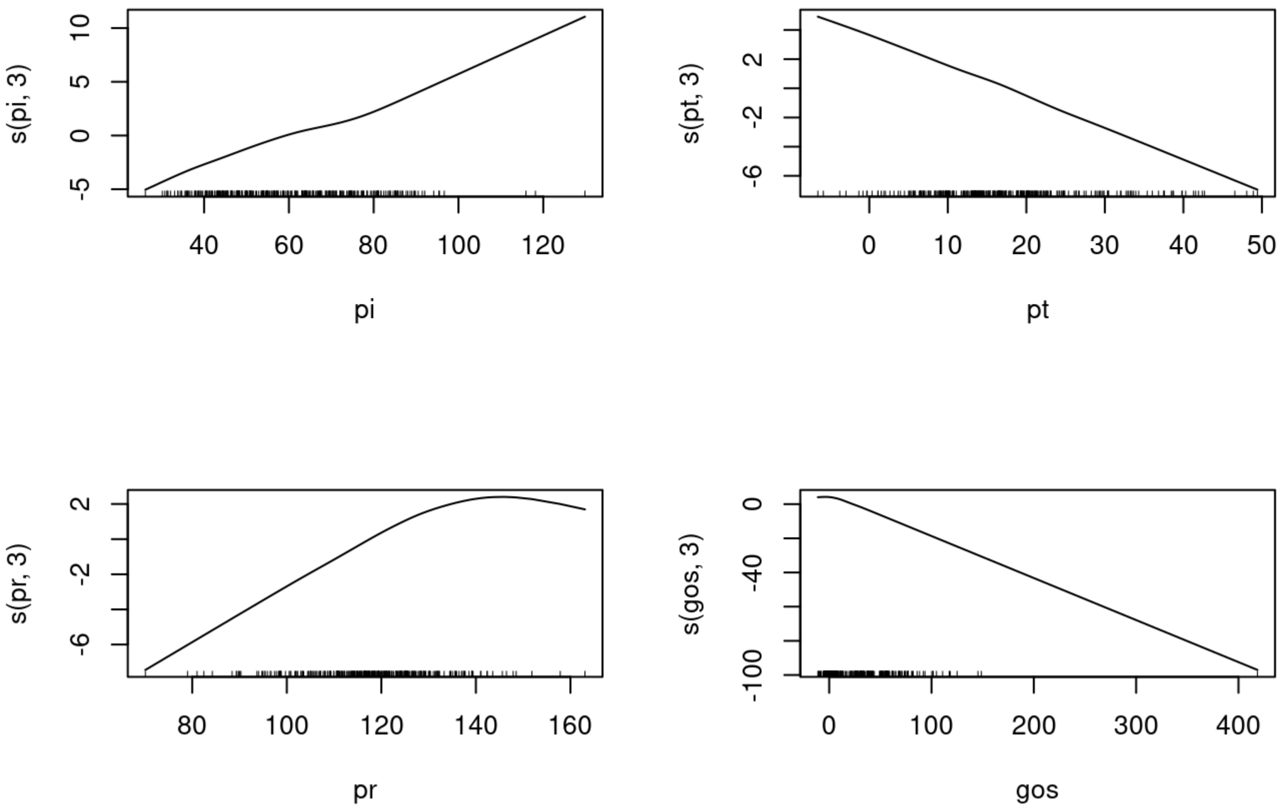
```
(r5 = gam(newclass ~ s(pi, 5) + s(pt, 5) + s(pr, 5) + s(gos,5), data=vc, family=binomial()))
```

```
## Call:
## gam(formula = newclass ~ s(pi, 5) + s(pt, 5) + s(pr, 5) + s(gos,
##     5), family = binomial(), data = vc)
##
## Degrees of Freedom: 309 total; 288.9995 Residual
## Residual Deviance: 149.6359
```

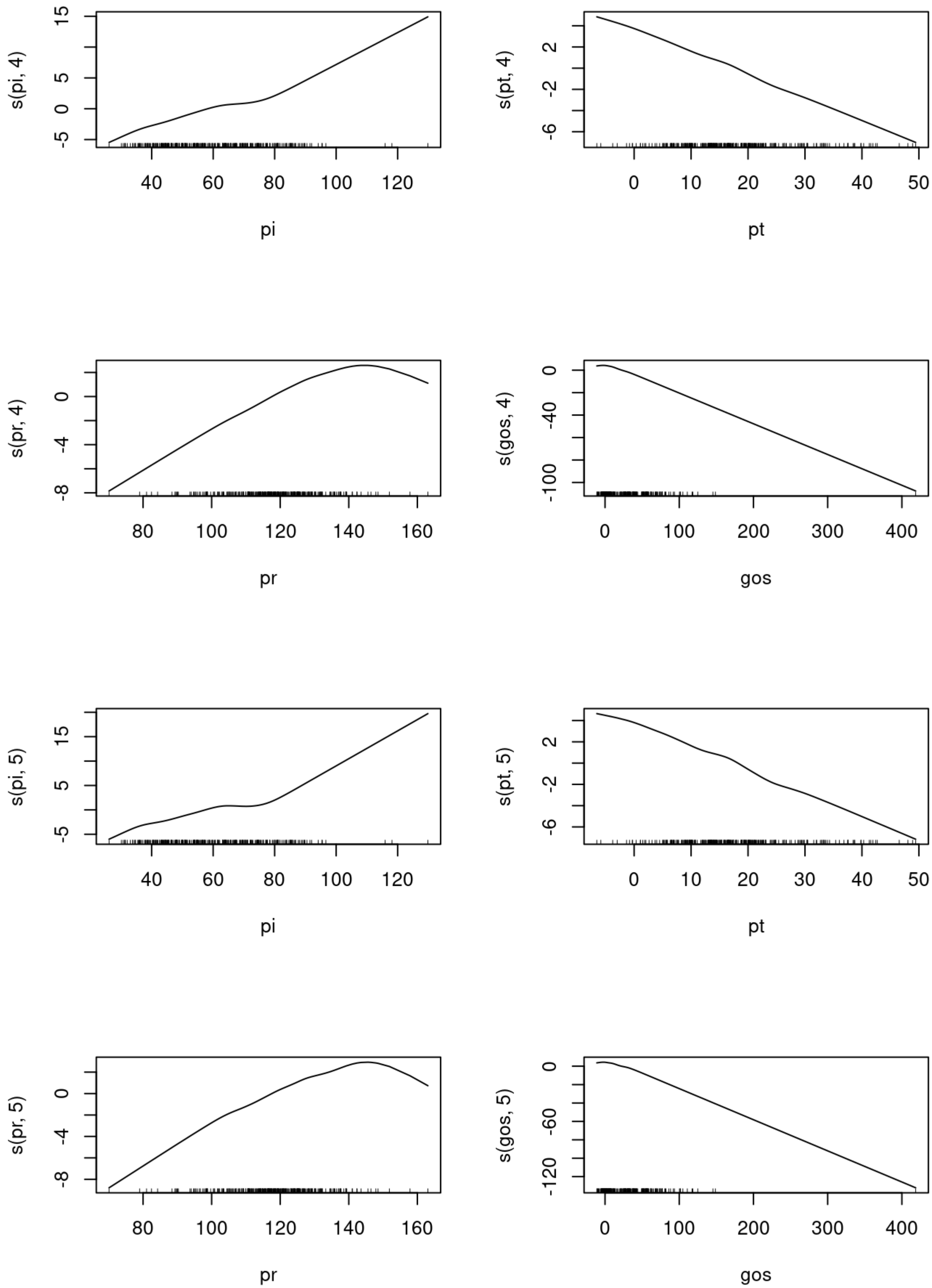
```
par(mfrow=c(2,2))
plot(r1);plot(r2);plot(r3);plot(r4);plot(r5)
```

```
## Warning in pchisq(nl.chisq, nldf): NaNs produced
```


s(pi, 1)
-4
0
4
8
40
60
80
100
120
pi
s(pt, 1)
-6
-2
2
0
10
20
30
40
50
pt
s(pr, 1)
-4
0
4
80
100
120
140
160
pr
s(gos, 1)
-60
-30
0
0
100
200
300
400
gos
s(pi, 2)
-4
0
4
8
40
60
80
100
120
pi
s(pt, 2)
-6
-2
2
0
10
20
30
40
50
pt
s(pr, 2)
-6
-2
2
80
100
120
140
160
pr
s(gos, 2)
-80
-40
0
0
100
200
300
400
gos



s(pi, 3)
-5
0
5
10
40
60
80
100
120
pi
s(pt, 3)
-6
-2
2
0
10
20
30
40
50
pt
s(pr, 3)
-6
-2
2
80
100
120
140
160
pr
s(gos, 3)
-100
-40
0
0
100
200
300
400
gos


s(pi, 4)
15
5
0
-5
40
60
80
100
120
pi
s(pt, 4)
2
-2
-6
0
10
20
30
40
50
pt
s(pr, 4)
0
-4
-8
80
100
120
140
160
pr
s(gos, 4)
0
-40
-100
0
100
200
300
400
gos
s(pi, 5)
15
5
-5
40
60
80
100
120
pi
s(pt, 5)
2
-2
-6
0
10
20
30
40
50
pt
s(pr, 5)
0
-4
-8
80
100
120
140
160
pr
s(gos, 5)
0
-60
-120
0
100
200
300
400
gos

```
## the 2 degree freedom shows better fitting
p2 = predict(r2, vc)   # Pr(Y = 1)
table(vc$newclass, as.numeric(p2 > 0.5) + 1)  # confusion table
```

```
##
##        1   2
##   AB 197  13
##   NO  24  76
```

```
# 2 degree classification accuracy
sum(diag(table(vc$newclass, as.numeric(p2 > 0.5) + 1)))/sum(table(vc$newclass, as.numeric(p2 >
0.5) + 1))
```

```
## [1] 0.8806452
```

Cross-validation and Parallel Computing

11, Use 10 repetitions of 10-fold cross-validation to evaluate the performance of the 5 classification methods.

The resulting classification accuracy of all five classification methods are shown in a table. The result shows the Multinomial Logistic Regression has highest classification accuracy for this dataset.

```
R = 10                    # number of repetitions
m = 5                     # Five classification methods we want to use
n = nrow(vc)              # data size
K = 10                    # K-fold CV

## function for shuffle the i-th sample from n size of data by K fold selection.
test.set = function(i, n, K=10) {
  if(i < 1 || i > K)
    stop(" i out of range (1, K)")
  start = round(1 + (i-1) * n / K)
  end = ifelse(i == K, n, round(i * n / K))
  start:end
}

set.seed(389)            # set a random seed
Atable = matrix(nrow=R*K, ncol=m)
colnames(Atable) = c("Linear","Quadratic","NaiveBayes","Multinomial","KNN")
for(i in 1:R) {                 # for each repetition
  ind = sample(n)               # shuffle index

    for(k in 1:K) {                 # for each fold
    index = ind[test.set(k, n, K)]
    test = vc[index,]
    train = vc[-index,]
    # Classification methods
      ## Linear discriminant analysis
      (r1 = lda(class ~ pi+pt+lla+ss+pr+gos, data=train))
      A1 = sum(diag(table(test$class, predict(r1, newdata=test)$class))/sum(table(test$class, pr
edict(r1, newdata=test)$class)))
      ## Quardratic discriminant analysis
      (r2 = qda(class ~ pi+pt+lla+ss+pr+gos, data=train))
      A2 = sum(diag(table(test$class, predict(r2, newdata=test)$class))/ sum(table(test$class, p
redict(r2, newdata=test)$class)))
      ## Naive Bayes
      (r3 = naiveBayes(class ~pi+pt+lla+ss+pr+gos, data=train))
      A3 = sum(diag(table(test$class, predict(r3, newdata=test)))/ sum(table(test$class, predict
(r3, newdata=test))))
      ## Multinomial Logistic Regression
      (r4 = multinom(class ~pi+pt+lla+ss+pr+gos, data=train))
      A4 = sum(diag(table(test$class, predict(r4, test)))/  sum(table(test$class, predict(r4, te
st))))
      ##KNN K = 10
      A5 = sum(diag(table(test[,7], knn(train=train[,1:6], test=test[,1:6], cl=train[,7], k=10
)))/ sum(table(test[,7], knn(train=train[,1:6], test=test[,1:6], cl=train[,7], k=10))))
      ## Classification accuracy
      Atable[K*(i-1)+k,] = c(A1,A2,A3,A4,A5)

  }
}
```

```
Atable
```

```
##          Linear Quadratic NaiveBayes Multinomial       KNN
##   [1,] 0.8709677 0.8709677  0.7096774   0.9354839 0.8387097
##   [2,] 0.9032258 0.8387097  0.8709677   0.9354839 0.8387097
##   [3,] 0.8709677 0.9354839  0.9032258   0.8709677 0.9354839
##   [4,] 0.7419355 0.8064516  0.7419355   0.8064516 0.7419355
##   [5,] 0.7419355 0.8709677  0.8387097   0.9032258 0.9354839
##   [6,] 0.8064516 0.7741935  0.8064516   0.8064516 0.8064516
##   [7,] 0.8387097 0.7741935  0.7741935   0.8709677 0.8064516
##   [8,] 0.8387097 0.7741935  0.8387097   0.8387097 0.8064516
##   [9,] 0.7741935 0.7741935  0.8387097   0.8064516 0.7741935
##  [10,] 0.7419355 0.8709677  0.9032258   0.8064516 0.7741935
##  [11,] 0.8064516 0.8064516  0.8709677   0.8387097 0.8064516
##  [12,] 0.7741935 0.8064516  0.7096774   0.8709677 0.8064516
##  [13,] 0.9354839 0.8709677  0.9032258   0.9354839 0.9354839
##  [14,] 0.7741935 0.8064516  0.7096774   0.8064516 0.7741935
##  [15,] 0.9032258 0.9354839  0.9032258   0.9354839 0.9032258
##  [16,] 0.6774194 0.7419355  0.7419355   0.8064516 0.7419355
##  [17,] 0.8387097 0.7419355  0.7419355   0.8064516 0.8387097
##  [18,] 0.8064516 0.8387097  0.9032258   0.8387097 0.8387097
##  [19,] 0.8387097 0.9354839  0.8064516   0.8709677 0.8064516
##  [20,] 0.8064516 0.9354839  0.9354839   0.9032258 0.9032258
##  [21,] 0.8387097 0.8064516  0.8387097   0.9032258 0.8387097
##  [22,] 0.6774194 0.8064516  0.8064516   0.8064516 0.7419355
##  [23,] 0.8709677 0.9032258  0.8709677   0.8709677 0.8709677
##  [24,] 0.8387097 0.8709677  0.8064516   0.9032258 0.8709677
##  [25,] 0.7741935 0.6451613  0.7741935   0.7096774 0.7419355
##  [26,] 0.8064516 0.7419355  0.7096774   0.8387097 0.8064516
##  [27,] 0.7096774 0.8387097  0.7741935   0.8064516 0.8709677
##  [28,] 0.8709677 0.8709677  0.8064516   0.9032258 0.9354839
##  [29,] 0.8387097 0.9354839  0.9032258   0.9354839 0.9032258
##  [30,] 0.8387097 0.9354839  0.9032258   0.8709677 0.9032258
##  [31,] 0.8387097 0.9032258  0.8387097   0.8387097 0.9032258
##  [32,] 0.6774194 0.8387097  0.8064516   0.8064516 0.7419355
##  [33,] 0.8709677 0.8709677  0.9032258   0.9032258 0.8387097
##  [34,] 0.7419355 0.7741935  0.7096774   0.7741935 0.8064516
##  [35,] 0.8387097 0.9032258  0.8709677   0.9032258 0.8387097
##  [36,] 0.8709677 0.9354839  0.8709677   0.9032258 0.8387097
##  [37,] 0.7096774 0.7741935  0.7741935   0.7419355 0.8064516
##  [38,] 0.8387097 0.9032258  0.9032258   0.8064516 0.8709677
##  [39,] 0.8387097 0.8064516  0.7419355   0.9032258 0.8709677
##  [40,] 0.8064516 0.7419355  0.8387097   0.8387097 0.8064516
##  [41,] 0.8709677 0.8387097  0.8709677   0.9032258 0.8709677
##  [42,] 0.7741935 0.9032258  0.8387097   0.9354839 0.9032258
##  [43,] 0.7096774 0.9032258  0.7741935   0.8064516 0.8064516
##  [44,] 0.7419355 0.7741935  0.7741935   0.8387097 0.7741935
##  [45,] 0.8387097 0.8709677  0.8387097   0.8709677 0.7419355
##  [46,] 0.8709677 0.8064516  0.8709677   0.8709677 0.7741935
##  [47,] 0.8064516 0.8709677  0.8387097   0.8709677 0.8709677
##  [48,] 0.8387097 0.8709677  0.8387097   0.8709677 0.9354839
##  [49,] 0.7741935 0.8387097  0.7741935   0.8064516 0.7096774
##  [50,] 0.7419355 0.8064516  0.8387097   0.8387097 0.8387097
##  [51,] 0.9032258 0.9032258  0.8709677   0.9354839 0.9354839
```

```
##  [52,] 0.7741935 0.7419355  0.8064516   0.8064516 0.8064516
##  [53,] 0.7741935 0.7741935  0.7419355   0.7419355 0.7096774
##  [54,] 0.8709677 0.8387097  0.8709677   0.9032258 0.9354839
##  [55,] 0.8064516 0.8709677  0.8709677   0.9032258 0.9032258
##  [56,] 0.6774194 0.8387097  0.7419355   0.8709677 0.7741935
##  [57,] 0.8709677 0.9032258  0.8064516   0.9354839 0.8387097
##  [58,] 0.8387097 0.8387097  0.9354839   0.8387097 0.8387097
##  [59,] 0.7741935 0.8709677  0.8709677   0.9354839 0.9677419
##  [60,] 0.7419355 0.7741935  0.7741935   0.7419355 0.7096774
##  [61,] 0.7419355 0.8387097  0.8387097   0.8387097 0.8387097
##  [62,] 0.8064516 0.8387097  0.8387097   0.8709677 0.8387097
##  [63,] 0.7419355 0.8064516  0.7419355   0.8387097 0.8709677
##  [64,] 0.8064516 0.7741935  0.7741935   0.8709677 0.8064516
##  [65,] 0.8064516 0.8387097  0.8064516   0.8709677 0.8709677
##  [66,] 0.8709677 0.9354839  0.9032258   0.9677419 0.9354839
##  [67,] 0.9032258 0.8709677  0.9032258   0.8709677 0.8387097
##  [68,] 0.8387097 0.8387097  0.9032258   0.8387097 0.8387097
##  [69,] 0.6774194 0.7096774  0.7096774   0.8064516 0.6774194
##  [70,] 0.8709677 0.8387097  0.8387097   0.9032258 0.9677419
##  [71,] 0.8064516 0.8064516  0.7741935   0.8064516 0.8064516
##  [72,] 0.8064516 0.8387097  0.7419355   0.8709677 0.8064516
##  [73,] 0.7419355 0.9032258  0.9032258   0.8709677 0.8387097
##  [74,] 0.8064516 0.8387097  0.8709677   0.9032258 0.8709677
##  [75,] 0.8064516 0.8064516  0.8387097   0.7741935 0.7419355
##  [76,] 0.8064516 0.8709677  0.8387097   0.9677419 0.8709677
##  [77,] 0.9032258 0.8709677  0.8709677   0.8709677 0.8387097
##  [78,] 0.8064516 0.7741935  0.7741935   0.8387097 0.8709677
##  [79,] 0.7741935 0.8387097  0.8709677   0.9032258 0.8709677
##  [80,] 0.8064516 0.8387097  0.7419355   0.8709677 0.8709677
##  [81,] 0.8387097 0.8387097  0.8709677   0.9032258 0.8709677
##  [82,] 0.7741935 0.8064516  0.8709677   0.8709677 0.8387097
##  [83,] 0.8064516 0.7741935  0.6451613   0.8387097 0.6774194
##  [84,] 0.6451613 0.7419355  0.7096774   0.7419355 0.7096774
##  [85,] 0.9354839 0.9032258  0.9032258   0.9032258 0.8709677
##  [86,] 0.7419355 0.8064516  0.7741935   0.7741935 0.8064516
##  [87,] 0.7096774 0.8709677  0.9032258   0.7741935 0.8064516
##  [88,] 0.7741935 0.7741935  0.7419355   0.8064516 0.9032258
##  [89,] 0.9032258 0.9354839  0.9354839   0.9354839 0.8709677
##  [90,] 0.8387097 0.9032258  0.8064516   0.9032258 0.8709677
##  [91,] 0.7741935 0.9032258  0.8709677   0.8387097 0.7419355
##  [92,] 0.8387097 0.8064516  0.8387097   0.9354839 0.9032258
##  [93,] 0.8387097 0.8064516  0.7741935   0.7741935 0.7741935
##  [94,] 0.8709677 0.9354839  0.9354839   0.9354839 0.9677419
##  [95,] 0.9032258 0.9032258  0.9354839   0.8709677 0.8709677
##  [96,] 0.8064516 0.8387097  0.7419355   0.8709677 0.8387097
##  [97,] 0.6451613 0.7741935  0.7419355   0.7096774 0.8064516
##  [98,] 0.7741935 0.8064516  0.8064516   0.8387097 0.8064516
##  [99,] 0.9032258 0.9032258  0.8709677   0.9032258 0.9032258
## [100,] 0.7419355 0.8387097  0.7419355   0.7741935 0.7096774
```

```
colMeans(Atable)
```

```
##      Linear   Quadratic  NaiveBayes Multinomial         KNN
##   0.8058065   0.8390323   0.8235484   0.8564516   0.8348387
```

```
## the result shows the Multinomial Logistic Regression has highest classification accuracy for
 this dataset.
```

Additional attempt to search for the optimum k value in KNN. It shows better results when K>=5, among which 7 neighboors KNN gives best prediction accuracy.

```
R = 10                   # number of repetitions
m = 10                   # K values we want to search (1:10)
n = nrow(vc)             # data size
K = 10                   # K-fold CV

## function for shuffle the i-th sample from n size of data by K fold selection.
test.set = function(i, n, K=10) {
  if(i < 1 || i > K)
    stop(" i out of range (1, K)")
  start = round(1 + (i-1) * n / K)
  end = ifelse(i == K, n, round(i * n / K))
  start:end
}

set.seed(389)           # set a random seed
Atable = matrix(nrow=R*K, ncol=m)

for(i in 1:R) {                 # for each repetition
  ind = sample(n)               # shuffle index
  for(k in 1:K) {               # for each fold
    index = ind[test.set(k, n, K)]
    test = vc[index,]
    train = vc[-index,]
     # for each classification accuracy
    for(j in 1:m) {
      ##KNN K = 10
      Aknn1 = sum(diag(table(test[,7], knn(train=train[,1:6], test=test[,1:6], cl=train[,7], k=
j)))/ sum(table(test[,7], knn(train=train[,1:6], test=test[,1:6], cl=train[,7], k=j))))
      ## Classification accuracy
      Atable[K*(i-1)+k,j] = Aknn1
    }
  }
}

colMeans(Atable)
```

```
##  [1] 0.8316129 0.7983871 0.8135484 0.8167742 0.8358065 0.8329032 0.8393548
##  [8] 0.8354839 0.8354839 0.8322581
```

```
which.max(colMeans(Atable)) ## the 7 neigbours KNN methods has highest classification accuracy.
```

```
## [1] 7
```

12, Modify code so that parallel computing can be used, in which each job running in parallel is only for the computation about one fold out of 10 (i.e., one training set and one test set). Make sure that the same subsamples are used by different methods and that the results are reproducible.Compare timings with 1, 5, 10, 20 cores.

```
## function for shuffle the i-th sample from n size of data by K fold selection.
 test.set = function(i, n, K) {
    if(i < 1 || i > K)
    stop(" i out of range (1, K)")
    start = round(1 + (i-1) * n / K)
    end = ifelse(i == K, n, round(i * n / K))
    start:end
 }

## function AClass for 5 different classification methods, A1:A5 are Classification accuracy
Aclass<-function(i) {
        set.seed(389+i%%K)
        ## shuffle data index when i%%10 =1
        if (i%%K == 1) {
        ind = sample(n)
        }
        ## generate test / train datasets
        index = ind[test.set(i%%K+1, n, K)]
        test  = vc[index,]
        head(test)
        train = vc[-index,]
# Classification methods: classification accuracy 'A' is sum of diagonal of confusion table / su
m of all.
      ## Linear discriminant analysis
      (r1 = lda(class ~ pi+pt+lla+ss+pr+gos, data=train))
      A1 = sum(diag(table(test$class, predict(r1, newdata=test)$class))/sum(table(test$class, pr
edict(r1, newdata=test)$class)))
      ## Quardratic discriminant analysis
      (r2 = qda(class ~ pi+pt+lla+ss+pr+gos, data=train))
      A2 = sum(diag(table(test$class, predict(r2, newdata=test)$class))/ sum(table(test$class, p
redict(r2, newdata=test)$class)))
      ## Naive Bayes
      (r3 = naiveBayes(class ~pi+pt+lla+ss+pr+gos, data=train))
      A3 = sum(diag(table(test$class, predict(r3, newdata=test)))/ sum(table(test$class, predict
(r3, newdata=test))))
      ## Multinomial Logistic Regression
      (r4 = multinom(class ~pi+pt+lla+ss+pr+gos, data=train))
      A4 = sum(diag(table(test$class, predict(r4, test)))/  sum(table(test$class, predict(r4, te
st))))
      ##KNN K = 10
      A5 = sum(diag(table(test[,7], knn(train=train[,1:6], test=test[,1:6], cl=train[,7], k=10
)))/ sum(table(test[,7], knn(train=train[,1:6], test=test[,1:6], cl=train[,7], k=10))))
      ## Classification accuracy
       c(A1,A2,A3,A4,A5)

}
```

```
## one Core computing time

system.time({
       R = 10                          # number of repetitions
       n = nrow(vc)                    # data size
       K = 10                          # K-fold CV
       Atable1<- mclapply(1:(R*K), function(x) Aclass(x), mc.set.seed=TRUE, mc.cores=1)
})
```

```
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 167.658623
## iter  20 value 77.465948
## iter  30 value 75.293132
## iter  40 value 75.268395
## iter  50 value 75.250990
## iter  60 value 75.209317
## iter  70 value 75.189242
## iter  80 value 75.142851
## final  value 75.100479
## converged
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 175.294456
## iter  20 value 75.351877
## iter  30 value 72.904903
## iter  40 value 72.789516
## iter  50 value 72.753296
## iter  60 value 72.732250
## iter  70 value 72.723833
## iter  80 value 72.715731
## iter  90 value 72.696543
## iter 100 value 72.685617
## final  value 72.685617
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 156.565002
## iter  20 value 79.328671
## iter  30 value 78.430061
## iter  40 value 78.426664
## iter  50 value 78.425666
## iter  60 value 78.425019
## final  value 78.424746
## converged
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 161.494008
## iter  20 value 79.420411
## iter  30 value 77.734451
## iter  40 value 77.725471
## iter  50 value 77.717311
## iter  60 value 77.622055
## iter  70 value 77.561991
## iter  80 value 77.553110
## iter  90 value 77.328736
## iter 100 value 77.220229
## final  value 77.220229
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 306.512829
```

```
## iter  10 value 170.649165
## iter  20 value 78.014427
## iter  30 value 75.648129
## iter  40 value 75.614429
## iter  50 value 75.581798
## iter  60 value 75.553213
## iter  70 value 75.541566
## iter  80 value 75.530252
## iter  90 value 75.522912
## iter 100 value 75.518911
## final  value 75.518911
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 165.775977
## iter  20 value 80.533840
## iter  30 value 78.580430
## iter  40 value 78.571567
## iter  50 value 78.556777
## iter  60 value 78.501597
## iter  70 value 78.477724
## iter  80 value 78.472945
## iter  90 value 78.461880
## iter 100 value 78.458230
## final  value 78.458230
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 167.346216
## iter  20 value 84.766956
## iter  30 value 83.284513
## iter  40 value 83.282332
## iter  50 value 83.279313
## iter  60 value 83.195110
## iter  70 value 83.175368
## iter  80 value 83.173982
## iter  90 value 83.172897
## final  value 83.172559
## converged
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 167.479971
## iter  20 value 86.602091
## iter  30 value 84.888486
## iter  40 value 84.880201
## iter  50 value 84.874257
## iter  60 value 84.854747
## iter  70 value 84.843247
## iter  80 value 84.840131
## iter  90 value 84.814170
## iter 100 value 84.799689
## final  value 84.799689
```

```
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 168.364703
## iter  20 value 86.079584
## iter  30 value 83.754829
## iter  40 value 83.740322
## iter  50 value 83.729164
## iter  60 value 83.705846
## iter  70 value 83.692057
## iter  80 value 83.689319
## iter  90 value 83.675849
## iter 100 value 83.670916
## final  value 83.670916
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 163.632039
## iter  20 value 85.066929
## iter  30 value 83.713257
## iter  40 value 83.698069
## iter  50 value 83.688379
## iter  60 value 83.657425
## iter  70 value 83.649456
## iter  80 value 83.648702
## final  value 83.639789
## converged
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 167.658623
## iter  20 value 77.465948
## iter  30 value 75.293132
## iter  40 value 75.268395
## iter  50 value 75.250990
## iter  60 value 75.209317
## iter  70 value 75.189242
## iter  80 value 75.142851
## final  value 75.100479
## converged
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 175.294456
## iter  20 value 75.351877
## iter  30 value 72.904903
## iter  40 value 72.789516
## iter  50 value 72.753296
## iter  60 value 72.732250
## iter  70 value 72.723833
## iter  80 value 72.715731
## iter  90 value 72.696543
## iter 100 value 72.685617
## final  value 72.685617
```

```
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 156.565002
## iter  20 value 79.328671
## iter  30 value 78.430061
## iter  40 value 78.426664
## iter  50 value 78.425666
## iter  60 value 78.425019
## final  value 78.424746
## converged
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 161.494008
## iter  20 value 79.420411
## iter  30 value 77.734451
## iter  40 value 77.725471
## iter  50 value 77.717311
## iter  60 value 77.622055
## iter  70 value 77.561991
## iter  80 value 77.553110
## iter  90 value 77.328736
## iter 100 value 77.220229
## final  value 77.220229
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 170.649165
## iter  20 value 78.014427
## iter  30 value 75.648129
## iter  40 value 75.614429
## iter  50 value 75.581798
## iter  60 value 75.553213
## iter  70 value 75.541566
## iter  80 value 75.530252
## iter  90 value 75.522912
## iter 100 value 75.518911
## final  value 75.518911
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 165.775977
## iter  20 value 80.533840
## iter  30 value 78.580430
## iter  40 value 78.571567
## iter  50 value 78.556777
## iter  60 value 78.501597
## iter  70 value 78.477724
## iter  80 value 78.472945
## iter  90 value 78.461880
## iter 100 value 78.458230
## final  value 78.458230
```

```
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 167.346216
## iter  20 value 84.766956
## iter  30 value 83.284513
## iter  40 value 83.282332
## iter  50 value 83.279313
## iter  60 value 83.195110
## iter  70 value 83.175368
## iter  80 value 83.173982
## iter  90 value 83.172897
## final  value 83.172559
## converged
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 167.479971
## iter  20 value 86.602091
## iter  30 value 84.888486
## iter  40 value 84.880201
## iter  50 value 84.874257
## iter  60 value 84.854747
## iter  70 value 84.843247
## iter  80 value 84.840131
## iter  90 value 84.814170
## iter 100 value 84.799689
## final  value 84.799689
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 168.364703
## iter  20 value 86.079584
## iter  30 value 83.754829
## iter  40 value 83.740322
## iter  50 value 83.729164
## iter  60 value 83.705846
## iter  70 value 83.692057
## iter  80 value 83.689319
## iter  90 value 83.675849
## iter 100 value 83.670916
## final  value 83.670916
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 163.632039
## iter  20 value 85.066929
## iter  30 value 83.713257
## iter  40 value 83.698069
## iter  50 value 83.688379
## iter  60 value 83.657425
## iter  70 value 83.649456
## iter  80 value 83.648702
```

```
## final  value 83.639789
## converged
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 167.658623
## iter  20 value 77.465948
## iter  30 value 75.293132
## iter  40 value 75.268395
## iter  50 value 75.250990
## iter  60 value 75.209317
## iter  70 value 75.189242
## iter  80 value 75.142851
## final  value 75.100479
## converged
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 175.294456
## iter  20 value 75.351877
## iter  30 value 72.904903
## iter  40 value 72.789516
## iter  50 value 72.753296
## iter  60 value 72.732250
## iter  70 value 72.723833
## iter  80 value 72.715731
## iter  90 value 72.696543
## iter 100 value 72.685617
## final  value 72.685617
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 156.565002
## iter  20 value 79.328671
## iter  30 value 78.430061
## iter  40 value 78.426664
## iter  50 value 78.425666
## iter  60 value 78.425019
## final  value 78.424746
## converged
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 161.494008
## iter  20 value 79.420411
## iter  30 value 77.734451
## iter  40 value 77.725471
## iter  50 value 77.717311
## iter  60 value 77.622055
## iter  70 value 77.561991
## iter  80 value 77.553110
## iter  90 value 77.328736
## iter 100 value 77.220229
## final  value 77.220229
## stopped after 100 iterations
```

```
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 170.649165
## iter  20 value 78.014427
## iter  30 value 75.648129
## iter  40 value 75.614429
## iter  50 value 75.581798
## iter  60 value 75.553213
## iter  70 value 75.541566
## iter  80 value 75.530252
## iter  90 value 75.522912
## iter 100 value 75.518911
## final  value 75.518911
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 165.775977
## iter  20 value 80.533840
## iter  30 value 78.580430
## iter  40 value 78.571567
## iter  50 value 78.556777
## iter  60 value 78.501597
## iter  70 value 78.477724
## iter  80 value 78.472945
## iter  90 value 78.461880
## iter 100 value 78.458230
## final  value 78.458230
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 167.346216
## iter  20 value 84.766956
## iter  30 value 83.284513
## iter  40 value 83.282332
## iter  50 value 83.279313
## iter  60 value 83.195110
## iter  70 value 83.175368
## iter  80 value 83.173982
## iter  90 value 83.172897
## final  value 83.172559
## converged
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 167.479971
## iter  20 value 86.602091
## iter  30 value 84.888486
## iter  40 value 84.880201
## iter  50 value 84.874257
## iter  60 value 84.854747
## iter  70 value 84.843247
## iter  80 value 84.840131
## iter  90 value 84.814170
```

```
## iter 100 value 84.799689
## final  value 84.799689
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 168.364703
## iter  20 value 86.079584
## iter  30 value 83.754829
## iter  40 value 83.740322
## iter  50 value 83.729164
## iter  60 value 83.705846
## iter  70 value 83.692057
## iter  80 value 83.689319
## iter  90 value 83.675849
## iter 100 value 83.670916
## final  value 83.670916
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 163.632039
## iter  20 value 85.066929
## iter  30 value 83.713257
## iter  40 value 83.698069
## iter  50 value 83.688379
## iter  60 value 83.657425
## iter  70 value 83.649456
## iter  80 value 83.648702
## final  value 83.639789
## converged
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 167.658623
## iter  20 value 77.465948
## iter  30 value 75.293132
## iter  40 value 75.268395
## iter  50 value 75.250990
## iter  60 value 75.209317
## iter  70 value 75.189242
## iter  80 value 75.142851
## final  value 75.100479
## converged
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 175.294456
## iter  20 value 75.351877
## iter  30 value 72.904903
## iter  40 value 72.789516
## iter  50 value 72.753296
## iter  60 value 72.732250
## iter  70 value 72.723833
## iter  80 value 72.715731
## iter  90 value 72.696543
```

```
## iter 100 value 72.685617
## final  value 72.685617
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 156.565002
## iter  20 value 79.328671
## iter  30 value 78.430061
## iter  40 value 78.426664
## iter  50 value 78.425666
## iter  60 value 78.425019
## final  value 78.424746
## converged
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 161.494008
## iter  20 value 79.420411
## iter  30 value 77.734451
## iter  40 value 77.725471
## iter  50 value 77.717311
## iter  60 value 77.622055
## iter  70 value 77.561991
## iter  80 value 77.553110
## iter  90 value 77.328736
## iter 100 value 77.220229
## final  value 77.220229
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 170.649165
## iter  20 value 78.014427
## iter  30 value 75.648129
## iter  40 value 75.614429
## iter  50 value 75.581798
## iter  60 value 75.553213
## iter  70 value 75.541566
## iter  80 value 75.530252
## iter  90 value 75.522912
## iter 100 value 75.518911
## final  value 75.518911
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 165.775977
## iter  20 value 80.533840
## iter  30 value 78.580430
## iter  40 value 78.571567
## iter  50 value 78.556777
## iter  60 value 78.501597
## iter  70 value 78.477724
## iter  80 value 78.472945
## iter  90 value 78.461880
```

```
## iter 100 value 78.458230
## final  value 78.458230
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 167.346216
## iter  20 value 84.766956
## iter  30 value 83.284513
## iter  40 value 83.282332
## iter  50 value 83.279313
## iter  60 value 83.195110
## iter  70 value 83.175368
## iter  80 value 83.173982
## iter  90 value 83.172897
## final  value 83.172559
## converged
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 167.479971
## iter  20 value 86.602091
## iter  30 value 84.888486
## iter  40 value 84.880201
## iter  50 value 84.874257
## iter  60 value 84.854747
## iter  70 value 84.843247
## iter  80 value 84.840131
## iter  90 value 84.814170
## iter 100 value 84.799689
## final  value 84.799689
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 168.364703
## iter  20 value 86.079584
## iter  30 value 83.754829
## iter  40 value 83.740322
## iter  50 value 83.729164
## iter  60 value 83.705846
## iter  70 value 83.692057
## iter  80 value 83.689319
## iter  90 value 83.675849
## iter 100 value 83.670916
## final  value 83.670916
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 163.632039
## iter  20 value 85.066929
## iter  30 value 83.713257
## iter  40 value 83.698069
## iter  50 value 83.688379
## iter  60 value 83.657425
```

```
## iter  70 value 83.649456
## iter  80 value 83.648702
## final  value 83.639789
## converged
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 167.658623
## iter  20 value 77.465948
## iter  30 value 75.293132
## iter  40 value 75.268395
## iter  50 value 75.250990
## iter  60 value 75.209317
## iter  70 value 75.189242
## iter  80 value 75.142851
## final  value 75.100479
## converged
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 175.294456
## iter  20 value 75.351877
## iter  30 value 72.904903
## iter  40 value 72.789516
## iter  50 value 72.753296
## iter  60 value 72.732250
## iter  70 value 72.723833
## iter  80 value 72.715731
## iter  90 value 72.696543
## iter 100 value 72.685617
## final  value 72.685617
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 156.565002
## iter  20 value 79.328671
## iter  30 value 78.430061
## iter  40 value 78.426664
## iter  50 value 78.425666
## iter  60 value 78.425019
## final  value 78.424746
## converged
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 161.494008
## iter  20 value 79.420411
## iter  30 value 77.734451
## iter  40 value 77.725471
## iter  50 value 77.717311
## iter  60 value 77.622055
## iter  70 value 77.561991
## iter  80 value 77.553110
## iter  90 value 77.328736
## iter 100 value 77.220229
```

```
## final  value 77.220229
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 170.649165
## iter  20 value 78.014427
## iter  30 value 75.648129
## iter  40 value 75.614429
## iter  50 value 75.581798
## iter  60 value 75.553213
## iter  70 value 75.541566
## iter  80 value 75.530252
## iter  90 value 75.522912
## iter 100 value 75.518911
## final  value 75.518911
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 165.775977
## iter  20 value 80.533840
## iter  30 value 78.580430
## iter  40 value 78.571567
## iter  50 value 78.556777
## iter  60 value 78.501597
## iter  70 value 78.477724
## iter  80 value 78.472945
## iter  90 value 78.461880
## iter 100 value 78.458230
## final  value 78.458230
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 167.346216
## iter  20 value 84.766956
## iter  30 value 83.284513
## iter  40 value 83.282332
## iter  50 value 83.279313
## iter  60 value 83.195110
## iter  70 value 83.175368
## iter  80 value 83.173982
## iter  90 value 83.172897
## final  value 83.172559
## converged
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 167.479971
## iter  20 value 86.602091
## iter  30 value 84.888486
## iter  40 value 84.880201
## iter  50 value 84.874257
## iter  60 value 84.854747
## iter  70 value 84.843247
```

```
## iter  80 value 84.840131
## iter  90 value 84.814170
## iter 100 value 84.799689
## final  value 84.799689
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 168.364703
## iter  20 value 86.079584
## iter  30 value 83.754829
## iter  40 value 83.740322
## iter  50 value 83.729164
## iter  60 value 83.705846
## iter  70 value 83.692057
## iter  80 value 83.689319
## iter  90 value 83.675849
## iter 100 value 83.670916
## final  value 83.670916
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 163.632039
## iter  20 value 85.066929
## iter  30 value 83.713257
## iter  40 value 83.698069
## iter  50 value 83.688379
## iter  60 value 83.657425
## iter  70 value 83.649456
## iter  80 value 83.648702
## final  value 83.639789
## converged
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 167.658623
## iter  20 value 77.465948
## iter  30 value 75.293132
## iter  40 value 75.268395
## iter  50 value 75.250990
## iter  60 value 75.209317
## iter  70 value 75.189242
## iter  80 value 75.142851
## final  value 75.100479
## converged
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 175.294456
## iter  20 value 75.351877
## iter  30 value 72.904903
## iter  40 value 72.789516
## iter  50 value 72.753296
## iter  60 value 72.732250
## iter  70 value 72.723833
```

```
## iter  80 value 72.715731
## iter  90 value 72.696543
## iter 100 value 72.685617
## final  value 72.685617
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 156.565002
## iter  20 value 79.328671
## iter  30 value 78.430061
## iter  40 value 78.426664
## iter  50 value 78.425666
## iter  60 value 78.425019
## final  value 78.424746
## converged
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 161.494008
## iter  20 value 79.420411
## iter  30 value 77.734451
## iter  40 value 77.725471
## iter  50 value 77.717311
## iter  60 value 77.622055
## iter  70 value 77.561991
## iter  80 value 77.553110
## iter  90 value 77.328736
## iter 100 value 77.220229
## final  value 77.220229
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 170.649165
## iter  20 value 78.014427
## iter  30 value 75.648129
## iter  40 value 75.614429
## iter  50 value 75.581798
## iter  60 value 75.553213
## iter  70 value 75.541566
## iter  80 value 75.530252
## iter  90 value 75.522912
## iter 100 value 75.518911
## final  value 75.518911
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 165.775977
## iter  20 value 80.533840
## iter  30 value 78.580430
## iter  40 value 78.571567
## iter  50 value 78.556777
## iter  60 value 78.501597
## iter  70 value 78.477724
```

```
## iter  80 value 78.472945
## iter  90 value 78.461880
## iter 100 value 78.458230
## final  value 78.458230
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 167.346216
## iter  20 value 84.766956
## iter  30 value 83.284513
## iter  40 value 83.282332
## iter  50 value 83.279313
## iter  60 value 83.195110
## iter  70 value 83.175368
## iter  80 value 83.173982
## iter  90 value 83.172897
## final  value 83.172559
## converged
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 167.479971
## iter  20 value 86.602091
## iter  30 value 84.888486
## iter  40 value 84.880201
## iter  50 value 84.874257
## iter  60 value 84.854747
## iter  70 value 84.843247
## iter  80 value 84.840131
## iter  90 value 84.814170
## iter 100 value 84.799689
## final  value 84.799689
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 168.364703
## iter  20 value 86.079584
## iter  30 value 83.754829
## iter  40 value 83.740322
## iter  50 value 83.729164
## iter  60 value 83.705846
## iter  70 value 83.692057
## iter  80 value 83.689319
## iter  90 value 83.675849
## iter 100 value 83.670916
## final  value 83.670916
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 163.632039
## iter  20 value 85.066929
## iter  30 value 83.713257
## iter  40 value 83.698069
```

```
## iter  50 value 83.688379
## iter  60 value 83.657425
## iter  70 value 83.649456
## iter  80 value 83.648702
## final  value 83.639789
## converged
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 167.658623
## iter  20 value 77.465948
## iter  30 value 75.293132
## iter  40 value 75.268395
## iter  50 value 75.250990
## iter  60 value 75.209317
## iter  70 value 75.189242
## iter  80 value 75.142851
## final  value 75.100479
## converged
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 175.294456
## iter  20 value 75.351877
## iter  30 value 72.904903
## iter  40 value 72.789516
## iter  50 value 72.753296
## iter  60 value 72.732250
## iter  70 value 72.723833
## iter  80 value 72.715731
## iter  90 value 72.696543
## iter 100 value 72.685617
## final  value 72.685617
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 156.565002
## iter  20 value 79.328671
## iter  30 value 78.430061
## iter  40 value 78.426664
## iter  50 value 78.425666
## iter  60 value 78.425019
## final  value 78.424746
## converged
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 161.494008
## iter  20 value 79.420411
## iter  30 value 77.734451
## iter  40 value 77.725471
## iter  50 value 77.717311
## iter  60 value 77.622055
## iter  70 value 77.561991
## iter  80 value 77.553110
```

```
## iter  90 value 77.328736
## iter 100 value 77.220229
## final  value 77.220229
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 170.649165
## iter  20 value 78.014427
## iter  30 value 75.648129
## iter  40 value 75.614429
## iter  50 value 75.581798
## iter  60 value 75.553213
## iter  70 value 75.541566
## iter  80 value 75.530252
## iter  90 value 75.522912
## iter 100 value 75.518911
## final  value 75.518911
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 165.775977
## iter  20 value 80.533840
## iter  30 value 78.580430
## iter  40 value 78.571567
## iter  50 value 78.556777
## iter  60 value 78.501597
## iter  70 value 78.477724
## iter  80 value 78.472945
## iter  90 value 78.461880
## iter 100 value 78.458230
## final  value 78.458230
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 167.346216
## iter  20 value 84.766956
## iter  30 value 83.284513
## iter  40 value 83.282332
## iter  50 value 83.279313
## iter  60 value 83.195110
## iter  70 value 83.175368
## iter  80 value 83.173982
## iter  90 value 83.172897
## final  value 83.172559
## converged
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 167.479971
## iter  20 value 86.602091
## iter  30 value 84.888486
## iter  40 value 84.880201
## iter  50 value 84.874257
```

```
## iter  60 value 84.854747
## iter  70 value 84.843247
## iter  80 value 84.840131
## iter  90 value 84.814170
## iter 100 value 84.799689
## final  value 84.799689
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 168.364703
## iter  20 value 86.079584
## iter  30 value 83.754829
## iter  40 value 83.740322
## iter  50 value 83.729164
## iter  60 value 83.705846
## iter  70 value 83.692057
## iter  80 value 83.689319
## iter  90 value 83.675849
## iter 100 value 83.670916
## final  value 83.670916
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 163.632039
## iter  20 value 85.066929
## iter  30 value 83.713257
## iter  40 value 83.698069
## iter  50 value 83.688379
## iter  60 value 83.657425
## iter  70 value 83.649456
## iter  80 value 83.648702
## final  value 83.639789
## converged
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 167.658623
## iter  20 value 77.465948
## iter  30 value 75.293132
## iter  40 value 75.268395
## iter  50 value 75.250990
## iter  60 value 75.209317
## iter  70 value 75.189242
## iter  80 value 75.142851
## final  value 75.100479
## converged
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 175.294456
## iter  20 value 75.351877
## iter  30 value 72.904903
## iter  40 value 72.789516
## iter  50 value 72.753296
```

```
## iter  60 value 72.732250
## iter  70 value 72.723833
## iter  80 value 72.715731
## iter  90 value 72.696543
## iter 100 value 72.685617
## final  value 72.685617
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 156.565002
## iter  20 value 79.328671
## iter  30 value 78.430061
## iter  40 value 78.426664
## iter  50 value 78.425666
## iter  60 value 78.425019
## final  value 78.424746
## converged
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 161.494008
## iter  20 value 79.420411
## iter  30 value 77.734451
## iter  40 value 77.725471
## iter  50 value 77.717311
## iter  60 value 77.622055
## iter  70 value 77.561991
## iter  80 value 77.553110
## iter  90 value 77.328736
## iter 100 value 77.220229
## final  value 77.220229
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 170.649165
## iter  20 value 78.014427
## iter  30 value 75.648129
## iter  40 value 75.614429
## iter  50 value 75.581798
## iter  60 value 75.553213
## iter  70 value 75.541566
## iter  80 value 75.530252
## iter  90 value 75.522912
## iter 100 value 75.518911
## final  value 75.518911
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 165.775977
## iter  20 value 80.533840
## iter  30 value 78.580430
## iter  40 value 78.571567
## iter  50 value 78.556777
```

```
## iter  60 value 78.501597
## iter  70 value 78.477724
## iter  80 value 78.472945
## iter  90 value 78.461880
## iter 100 value 78.458230
## final  value 78.458230
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 167.346216
## iter  20 value 84.766956
## iter  30 value 83.284513
## iter  40 value 83.282332
## iter  50 value 83.279313
## iter  60 value 83.195110
## iter  70 value 83.175368
## iter  80 value 83.173982
## iter  90 value 83.172897
## final  value 83.172559
## converged
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 167.479971
## iter  20 value 86.602091
## iter  30 value 84.888486
## iter  40 value 84.880201
## iter  50 value 84.874257
## iter  60 value 84.854747
## iter  70 value 84.843247
## iter  80 value 84.840131
## iter  90 value 84.814170
## iter 100 value 84.799689
## final  value 84.799689
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 168.364703
## iter  20 value 86.079584
## iter  30 value 83.754829
## iter  40 value 83.740322
## iter  50 value 83.729164
## iter  60 value 83.705846
## iter  70 value 83.692057
## iter  80 value 83.689319
## iter  90 value 83.675849
## iter 100 value 83.670916
## final  value 83.670916
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 163.632039
## iter  20 value 85.066929
```

```
## iter  30 value 83.713257
## iter  40 value 83.698069
## iter  50 value 83.688379
## iter  60 value 83.657425
## iter  70 value 83.649456
## iter  80 value 83.648702
## final  value 83.639789
## converged
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 167.658623
## iter  20 value 77.465948
## iter  30 value 75.293132
## iter  40 value 75.268395
## iter  50 value 75.250990
## iter  60 value 75.209317
## iter  70 value 75.189242
## iter  80 value 75.142851
## final  value 75.100479
## converged
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 175.294456
## iter  20 value 75.351877
## iter  30 value 72.904903
## iter  40 value 72.789516
## iter  50 value 72.753296
## iter  60 value 72.732250
## iter  70 value 72.723833
## iter  80 value 72.715731
## iter  90 value 72.696543
## iter 100 value 72.685617
## final  value 72.685617
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 156.565002
## iter  20 value 79.328671
## iter  30 value 78.430061
## iter  40 value 78.426664
## iter  50 value 78.425666
## iter  60 value 78.425019
## final  value 78.424746
## converged
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 161.494008
## iter  20 value 79.420411
## iter  30 value 77.734451
## iter  40 value 77.725471
## iter  50 value 77.717311
## iter  60 value 77.622055
```

```
## iter  70 value 77.561991
## iter  80 value 77.553110
## iter  90 value 77.328736
## iter 100 value 77.220229
## final  value 77.220229
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 170.649165
## iter  20 value 78.014427
## iter  30 value 75.648129
## iter  40 value 75.614429
## iter  50 value 75.581798
## iter  60 value 75.553213
## iter  70 value 75.541566
## iter  80 value 75.530252
## iter  90 value 75.522912
## iter 100 value 75.518911
## final  value 75.518911
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 165.775977
## iter  20 value 80.533840
## iter  30 value 78.580430
## iter  40 value 78.571567
## iter  50 value 78.556777
## iter  60 value 78.501597
## iter  70 value 78.477724
## iter  80 value 78.472945
## iter  90 value 78.461880
## iter 100 value 78.458230
## final  value 78.458230
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 167.346216
## iter  20 value 84.766956
## iter  30 value 83.284513
## iter  40 value 83.282332
## iter  50 value 83.279313
## iter  60 value 83.195110
## iter  70 value 83.175368
## iter  80 value 83.173982
## iter  90 value 83.172897
## final  value 83.172559
## converged
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 167.479971
## iter  20 value 86.602091
## iter  30 value 84.888486
```

```
## iter  40 value 84.880201
## iter  50 value 84.874257
## iter  60 value 84.854747
## iter  70 value 84.843247
## iter  80 value 84.840131
## iter  90 value 84.814170
## iter 100 value 84.799689
## final  value 84.799689
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 168.364703
## iter  20 value 86.079584
## iter  30 value 83.754829
## iter  40 value 83.740322
## iter  50 value 83.729164
## iter  60 value 83.705846
## iter  70 value 83.692057
## iter  80 value 83.689319
## iter  90 value 83.675849
## iter 100 value 83.670916
## final  value 83.670916
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 163.632039
## iter  20 value 85.066929
## iter  30 value 83.713257
## iter  40 value 83.698069
## iter  50 value 83.688379
## iter  60 value 83.657425
## iter  70 value 83.649456
## iter  80 value 83.648702
## final  value 83.639789
## converged
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 167.658623
## iter  20 value 77.465948
## iter  30 value 75.293132
## iter  40 value 75.268395
## iter  50 value 75.250990
## iter  60 value 75.209317
## iter  70 value 75.189242
## iter  80 value 75.142851
## final  value 75.100479
## converged
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 175.294456
## iter  20 value 75.351877
## iter  30 value 72.904903
```

```
## iter  40 value 72.789516
## iter  50 value 72.753296
## iter  60 value 72.732250
## iter  70 value 72.723833
## iter  80 value 72.715731
## iter  90 value 72.696543
## iter 100 value 72.685617
## final  value 72.685617
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 156.565002
## iter  20 value 79.328671
## iter  30 value 78.430061
## iter  40 value 78.426664
## iter  50 value 78.425666
## iter  60 value 78.425019
## final  value 78.424746
## converged
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 161.494008
## iter  20 value 79.420411
## iter  30 value 77.734451
## iter  40 value 77.725471
## iter  50 value 77.717311
## iter  60 value 77.622055
## iter  70 value 77.561991
## iter  80 value 77.553110
## iter  90 value 77.328736
## iter 100 value 77.220229
## final  value 77.220229
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 170.649165
## iter  20 value 78.014427
## iter  30 value 75.648129
## iter  40 value 75.614429
## iter  50 value 75.581798
## iter  60 value 75.553213
## iter  70 value 75.541566
## iter  80 value 75.530252
## iter  90 value 75.522912
## iter 100 value 75.518911
## final  value 75.518911
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 165.775977
## iter  20 value 80.533840
## iter  30 value 78.580430
```

```
## iter  40 value 78.571567
## iter  50 value 78.556777
## iter  60 value 78.501597
## iter  70 value 78.477724
## iter  80 value 78.472945
## iter  90 value 78.461880
## iter 100 value 78.458230
## final  value 78.458230
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 167.346216
## iter  20 value 84.766956
## iter  30 value 83.284513
## iter  40 value 83.282332
## iter  50 value 83.279313
## iter  60 value 83.195110
## iter  70 value 83.175368
## iter  80 value 83.173982
## iter  90 value 83.172897
## final  value 83.172559
## converged
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 167.479971
## iter  20 value 86.602091
## iter  30 value 84.888486
## iter  40 value 84.880201
## iter  50 value 84.874257
## iter  60 value 84.854747
## iter  70 value 84.843247
## iter  80 value 84.840131
## iter  90 value 84.814170
## iter 100 value 84.799689
## final  value 84.799689
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 306.512829
## iter  10 value 168.364703
## iter  20 value 86.079584
## iter  30 value 83.754829
## iter  40 value 83.740322
## iter  50 value 83.729164
## iter  60 value 83.705846
## iter  70 value 83.692057
## iter  80 value 83.689319
## iter  90 value 83.675849
## iter 100 value 83.670916
## final  value 83.670916
## stopped after 100 iterations
## # weights:  24 (14 variable)
## initial  value 306.512829
```

```
## iter  10 value 163.632039
## iter  20 value 85.066929
## iter  30 value 83.713257
## iter  40 value 83.698069
## iter  50 value 83.688379
## iter  60 value 83.657425
## iter  70 value 83.649456
## iter  80 value 83.648702
## final  value 83.639789
## converged
```

```
##    user  system elapsed
##   3.809   0.000   3.810
```

```
## 5 Cores computing time
system.time({
       R = 10                            # number of repetitions
       n = nrow(vc)                      # data size
       K = 10                            # K-fold CV
       Atable5<- mclapply(1:(R*K), function(x) Aclass(x), mc.set.seed=TRUE, mc.cores=5)
})
```

```
##    user  system elapsed
##   4.601   0.193   1.245
```

```
## 10 Cores computing time
system.time({
       R = 10                            # number of repetitions
       n = nrow(vc)                      # data size
       K = 10                            # K-fold CV
       Atable10<- mclapply(1:(R*K), function(x) Aclass(x), mc.set.seed=TRUE, mc.cores=10)
})
```

```
##    user  system elapsed
##   4.499   0.472   0.679
```

```
## 20 Cores computing time
system.time({
       R = 10                            # number of repetitions
       n = nrow(vc)                      # data size
       K = 10                            # K-fold CV
       Atable20<- mclapply(1:(R*K), function(x) Aclass(x), mc.set.seed=TRUE, mc.cores=20)
})
```

```
##    user  system elapsed
##   5.277   0.983   0.509
```

```
## example of 5 cores results
t(matrix(unlist(Atable5),5,100))
```

```
##            [,1]      [,2]      [,3]      [,4]      [,5]
##   [1,] 0.7419355 0.7741935 0.7096774 0.7741935 0.8709677
##   [2,] 0.7419355 0.8709677 0.7741935 0.8387097 0.8064516
##   [3,] 0.7419355 0.8387097 0.8387097 0.8387097 0.8064516
##   [4,] 0.7096774 0.6774194 0.8064516 0.8064516 0.7741935
##   [5,] 0.7419355 0.7741935 0.7741935 0.7741935 0.8387097
##   [6,] 0.7741935 0.8064516 0.8064516 0.7741935 0.8387097
##   [7,] 0.9677419 0.8709677 0.8387097 0.9354839 0.8387097
##   [8,] 0.8387097 0.9354839 0.9354839 0.9354839 0.8709677
##   [9,] 0.8387097 0.9032258 0.7741935 0.8709677 0.8387097
##  [10,] 0.9032258 0.9032258 0.7419355 0.9032258 0.8709677
##  [11,] 0.7419355 0.7741935 0.7096774 0.7741935 0.8709677
##  [12,] 0.7419355 0.8709677 0.7741935 0.8387097 0.8064516
##  [13,] 0.7419355 0.8387097 0.8387097 0.8387097 0.8064516
##  [14,] 0.7096774 0.6774194 0.8064516 0.8064516 0.7741935
##  [15,] 0.7419355 0.7741935 0.7741935 0.7741935 0.8387097
##  [16,] 0.7741935 0.8064516 0.8064516 0.7741935 0.8387097
##  [17,] 0.9677419 0.8709677 0.8387097 0.9354839 0.8387097
##  [18,] 0.8387097 0.9354839 0.9354839 0.9354839 0.8709677
##  [19,] 0.8387097 0.9032258 0.7741935 0.8709677 0.8387097
##  [20,] 0.9032258 0.9032258 0.7419355 0.9032258 0.8709677
##  [21,] 0.7419355 0.7741935 0.7096774 0.7741935 0.8709677
##  [22,] 0.7419355 0.8709677 0.7741935 0.8387097 0.8064516
##  [23,] 0.7419355 0.8387097 0.8387097 0.8387097 0.8064516
##  [24,] 0.7096774 0.6774194 0.8064516 0.8064516 0.7741935
##  [25,] 0.7419355 0.7741935 0.7741935 0.7741935 0.8387097
##  [26,] 0.7741935 0.8064516 0.8064516 0.7741935 0.8387097
##  [27,] 0.9677419 0.8709677 0.8387097 0.9354839 0.8387097
##  [28,] 0.8387097 0.9354839 0.9354839 0.9354839 0.8709677
##  [29,] 0.8387097 0.9032258 0.7741935 0.8709677 0.8387097
##  [30,] 0.9032258 0.9032258 0.7419355 0.9032258 0.8709677
##  [31,] 0.7419355 0.7741935 0.7096774 0.7741935 0.8709677
##  [32,] 0.7419355 0.8709677 0.7741935 0.8387097 0.8064516
##  [33,] 0.7419355 0.8387097 0.8387097 0.8387097 0.8064516
##  [34,] 0.7096774 0.6774194 0.8064516 0.8064516 0.7741935
##  [35,] 0.7419355 0.7741935 0.7741935 0.7741935 0.8387097
##  [36,] 0.7741935 0.8064516 0.8064516 0.7741935 0.8387097
##  [37,] 0.9677419 0.8709677 0.8387097 0.9354839 0.8387097
##  [38,] 0.8387097 0.9354839 0.9354839 0.9354839 0.8709677
##  [39,] 0.8387097 0.9032258 0.7741935 0.8709677 0.8387097
##  [40,] 0.9032258 0.9032258 0.7419355 0.9032258 0.8709677
##  [41,] 0.7419355 0.7741935 0.7096774 0.7741935 0.8709677
##  [42,] 0.7419355 0.8709677 0.7741935 0.8387097 0.8064516
##  [43,] 0.7419355 0.8387097 0.8387097 0.8387097 0.8064516
##  [44,] 0.7096774 0.6774194 0.8064516 0.8064516 0.7741935
##  [45,] 0.7419355 0.7741935 0.7741935 0.7741935 0.8387097
##  [46,] 0.7741935 0.8064516 0.8064516 0.7741935 0.8387097
##  [47,] 0.9677419 0.8709677 0.8387097 0.9354839 0.8387097
##  [48,] 0.8387097 0.9354839 0.9354839 0.9354839 0.8709677
##  [49,] 0.8387097 0.9032258 0.7741935 0.8709677 0.8387097
##  [50,] 0.9032258 0.9032258 0.7419355 0.9032258 0.8709677
##  [51,] 0.7419355 0.7741935 0.7096774 0.7741935 0.8709677
```

```
##  [52,] 0.7419355 0.8709677 0.7741935 0.8387097 0.8064516
##  [53,] 0.7419355 0.8387097 0.8387097 0.8387097 0.8064516
##  [54,] 0.7096774 0.6774194 0.8064516 0.8064516 0.7741935
##  [55,] 0.7419355 0.7741935 0.7741935 0.7741935 0.8387097
##  [56,] 0.7741935 0.8064516 0.8064516 0.7741935 0.8387097
##  [57,] 0.9677419 0.8709677 0.8387097 0.9354839 0.8387097
##  [58,] 0.8387097 0.9354839 0.9354839 0.9354839 0.8709677
##  [59,] 0.8387097 0.9032258 0.7741935 0.8709677 0.8387097
##  [60,] 0.9032258 0.9032258 0.7419355 0.9032258 0.8709677
##  [61,] 0.7419355 0.7741935 0.7096774 0.7741935 0.8709677
##  [62,] 0.7419355 0.8709677 0.7741935 0.8387097 0.8064516
##  [63,] 0.7419355 0.8387097 0.8387097 0.8387097 0.8064516
##  [64,] 0.7096774 0.6774194 0.8064516 0.8064516 0.7741935
##  [65,] 0.7419355 0.7741935 0.7741935 0.7741935 0.8387097
##  [66,] 0.7741935 0.8064516 0.8064516 0.7741935 0.8387097
##  [67,] 0.9677419 0.8709677 0.8387097 0.9354839 0.8387097
##  [68,] 0.8387097 0.9354839 0.9354839 0.9354839 0.8709677
##  [69,] 0.8387097 0.9032258 0.7741935 0.8709677 0.8387097
##  [70,] 0.9032258 0.9032258 0.7419355 0.9032258 0.8709677
##  [71,] 0.7419355 0.7741935 0.7096774 0.7741935 0.8709677
##  [72,] 0.7419355 0.8709677 0.7741935 0.8387097 0.8064516
##  [73,] 0.7419355 0.8387097 0.8387097 0.8387097 0.8064516
##  [74,] 0.7096774 0.6774194 0.8064516 0.8064516 0.7741935
##  [75,] 0.7419355 0.7741935 0.7741935 0.7741935 0.8387097
##  [76,] 0.7741935 0.8064516 0.8064516 0.7741935 0.8387097
##  [77,] 0.9677419 0.8709677 0.8387097 0.9354839 0.8387097
##  [78,] 0.8387097 0.9354839 0.9354839 0.9354839 0.8709677
##  [79,] 0.8387097 0.9032258 0.7741935 0.8709677 0.8387097
##  [80,] 0.9032258 0.9032258 0.7419355 0.9032258 0.8709677
##  [81,] 0.7419355 0.7741935 0.7096774 0.7741935 0.8709677
##  [82,] 0.7419355 0.8709677 0.7741935 0.8387097 0.8064516
##  [83,] 0.7419355 0.8387097 0.8387097 0.8387097 0.8064516
##  [84,] 0.7096774 0.6774194 0.8064516 0.8064516 0.7741935
##  [85,] 0.7419355 0.7741935 0.7741935 0.7741935 0.8387097
##  [86,] 0.7741935 0.8064516 0.8064516 0.7741935 0.8387097
##  [87,] 0.9677419 0.8709677 0.8387097 0.9354839 0.8387097
##  [88,] 0.8387097 0.9354839 0.9354839 0.9354839 0.8709677
##  [89,] 0.8387097 0.9032258 0.7741935 0.8709677 0.8387097
##  [90,] 0.9032258 0.9032258 0.7419355 0.9032258 0.8709677
##  [91,] 0.7419355 0.7741935 0.7096774 0.7741935 0.8709677
##  [92,] 0.7419355 0.8709677 0.7741935 0.8387097 0.8064516
##  [93,] 0.7419355 0.8387097 0.8387097 0.8387097 0.8064516
##  [94,] 0.7096774 0.6774194 0.8064516 0.8064516 0.7741935
##  [95,] 0.7419355 0.7741935 0.7741935 0.7741935 0.8387097
##  [96,] 0.7741935 0.8064516 0.8064516 0.7741935 0.8387097
##  [97,] 0.9677419 0.8709677 0.8387097 0.9354839 0.8387097
##  [98,] 0.8387097 0.9354839 0.9354839 0.9354839 0.8709677
##  [99,] 0.8387097 0.9032258 0.7741935 0.8709677 0.8387097
## [100,] 0.9032258 0.9032258 0.7419355 0.9032258 0.8709677
```

```
### results comparison of parallel computing with multi-cores
core<-rbind(colMeans(t(matrix(unlist(Atable1),5,100))),colMeans(t(matrix(unlist(Atable5),5,100
))),colMeans(t(matrix(unlist(Atable10),5,100))),colMeans(t(matrix(unlist(Atable20),5,100))))
dimnames(core)<- list(c("1 core","5cores","10cores","20cores"),c("Linear","Quadratic","NaiveBaye
s","Multinomial","KNN"))
core
```

```
##         Linear Quadratic NaiveBayes Multinomial       KNN
## 1 core     0.8 0.8354839        0.8   0.8451613 0.8354839
## 5cores     0.8 0.8354839        0.8   0.8451613 0.8354839
## 10cores    0.8 0.8354839        0.8   0.8451613 0.8354839
## 20cores    0.8 0.8354839        0.8   0.8451613 0.8354839
```

```
colMeans(core)
```

```
##      Linear   Quadratic  NaiveBayes Multinomial         KNN
##   0.8000000   0.8354839   0.8000000   0.8451613   0.8354839
```

Summary

In this lab we have learn all different kinds of classification methods, Linear discriminant analysis,Quadratic discriminant analysis,Naive Bayes,Multinomial logistic regression,K-nearest neighbours. All confusion matrix and characterization accuracy are generated. For this dataset the primary evaluation shows that “Quadratic discriminant analysis”, “multinomial logistic regression”, “K-nearest neighbours” have the better results. Cross-validation results show “multinomial logistic regression” gives the highest accuracy that is similar to AIC backward selected model results. Further we explore the parallel coding in VM with 1,5,10,20 cores, the result shows the more cores the less time it takes for computation.