final project

0. Inclide the library and process the data

```r
set.seed(441)
library(MASS)
library(factoextra)
```

```
## Loading required package: ggplot2

## Welcome! Related Books: `Practical Guide To Cluster Analysis in R` at https://goo.gl/13EFCZ
```

```r
library(klaR)
library(nnet)
library(glmnet)
```

```
## Loading required package: Matrix

## Loading required package: foreach

## Loaded glmnet 2.0-18
```

```r
library(mgcv)
```

```
## Loading required package: nlme

## This is mgcv 1.8-26. For overview type 'help("mgcv-package")'.

##
## Attaching package: 'mgcv'

## The following object is masked from 'package:nnet':
##
##     multinom
```

```r
library(car)
```

```
## Loading required package: carData
```

```r
library(e1071)
library(rpart)
library(rpart.plot)
library(randomForest)
```

```
## randomForest 4.6-14

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

##
## Attaching package: 'randomForest'

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

```r
library(gbm)
```

```
## Loaded gbm 2.1.5
```

```r
library(rlist)
```

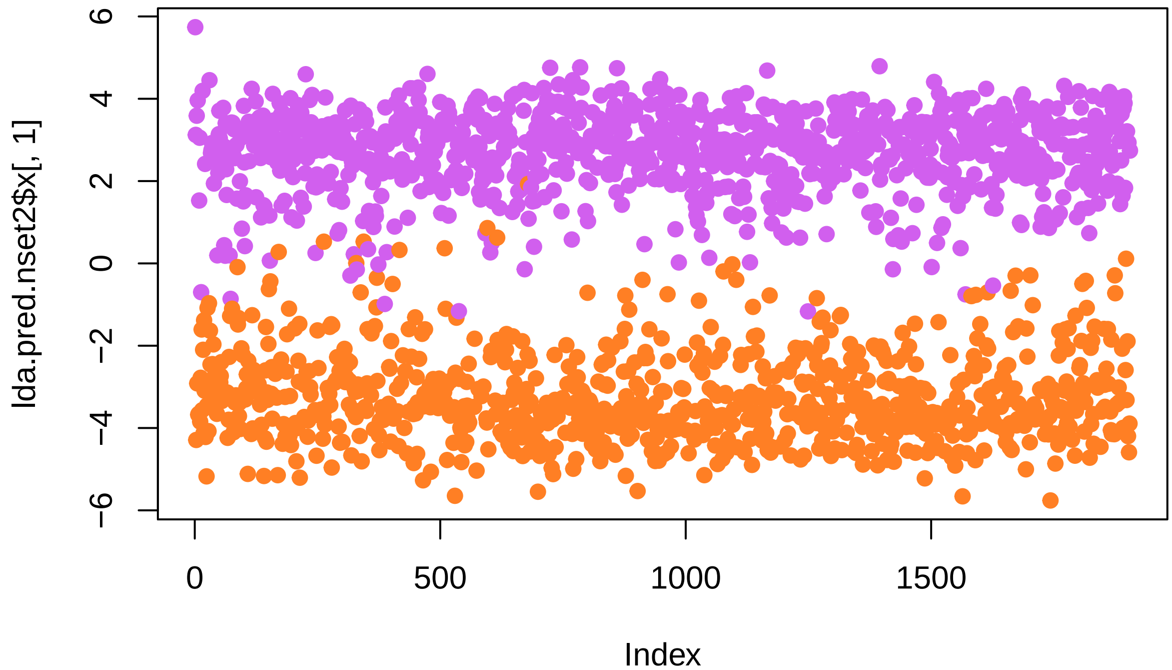
```r
pop <- read.csv("LOL.csv")
getCols = function(pop, name) {
  c(grep(name, colnames(pop)))
}
pop = pop[,-getCols(pop,"Type")]
pop = pop[,-getCols(pop,"Year")]
pop = pop[,-getCols(pop,"Season")]
pop = pop[,-getCols(pop,"League")]
# change it to our data later
pop$bResult = ifelse(pop$bResult == 0, "Defeat", "Victory")
pop$bResult = as.factor(pop$bResult)
perm<-sample(x=nrow(pop))
set1.full <- pop[which(perm<=nrow(pop)/2),]
set2.full <- pop[which(nrow(pop)/2<perm & perm<=3*nrow(pop)/4),]
set3.full <- pop[which(perm>3*nrow(pop)/4),]
set1 = set1.full
set2 = set2.full
set3 = set3.full
```

numeric the data

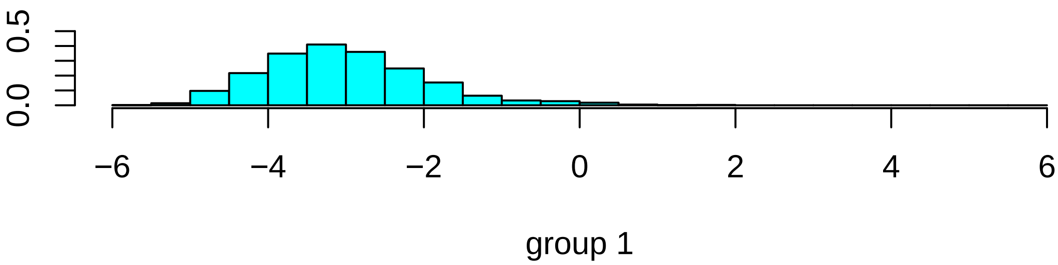
```r
nset1.full = set1.full
nset2.full = set2.full
nset3.full = set3.full
for (i in 1:64) {
  nset1.full[,i] = as.numeric(set1.full[,i])
  nset2.full[,i] = as.numeric(set2.full[,i])
  nset3.full[,i] = as.numeric(set3.full[,i])
}
nset1 = nset1.full
nset2 = nset2.full
nset3 = nset3.full
```

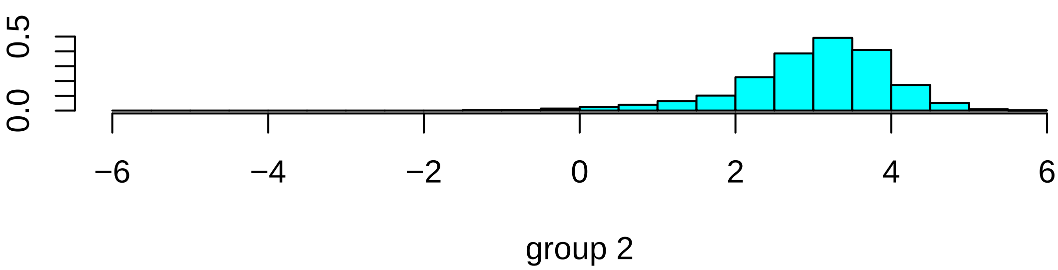
1. Try the lda for variable selcetion

```r
lda.fit = lda(bResult~. , data = nset1)
lda.pred.nset1 <- predict(lda.fit, nset1)
lda.pred.nset2 <- predict(lda.fit, nset2)
#lda.fit
class.col <- ifelse(nset2$bResult==1,y=53,n=464)
plot(lda.pred.nset2$x[,1], col=colors()[class.col], pch = 19)
```




```r
plot(lda.fit)
```






```r
lda.pred.train <- lda.pred.nset1$class
lda.pred.val <- predict(lda.fit, nset2)$class
lda.pred.test <- predict(lda.fit, nset3)$class
mean(ifelse(lda.pred.train == nset1$bResult, yes=0, no=1))
```

```
## [1] 0.01181102
```

```r
mean(ifelse(lda.pred.val == nset2$bResult, yes=0, no=1))
```

```
## [1] 0.01207349
```

```r
mean(ifelse(lda.pred.test == nset3$bResult, yes=0, no=1))
```

```
## [1] 0.009973753
```

With all the explantory variables, linear discrement analysis has already been a perfect split with 1% test error rate. Look into the importance of explantory variable.

```r
lda.table = as.data.frame(as.table(lda.fit$scaling))
lda.table = lda.table[order(abs(lda.table$Freq), decreasing = TRUE),]
print.data.frame(lda.table[1:20,c(1,3)])
```

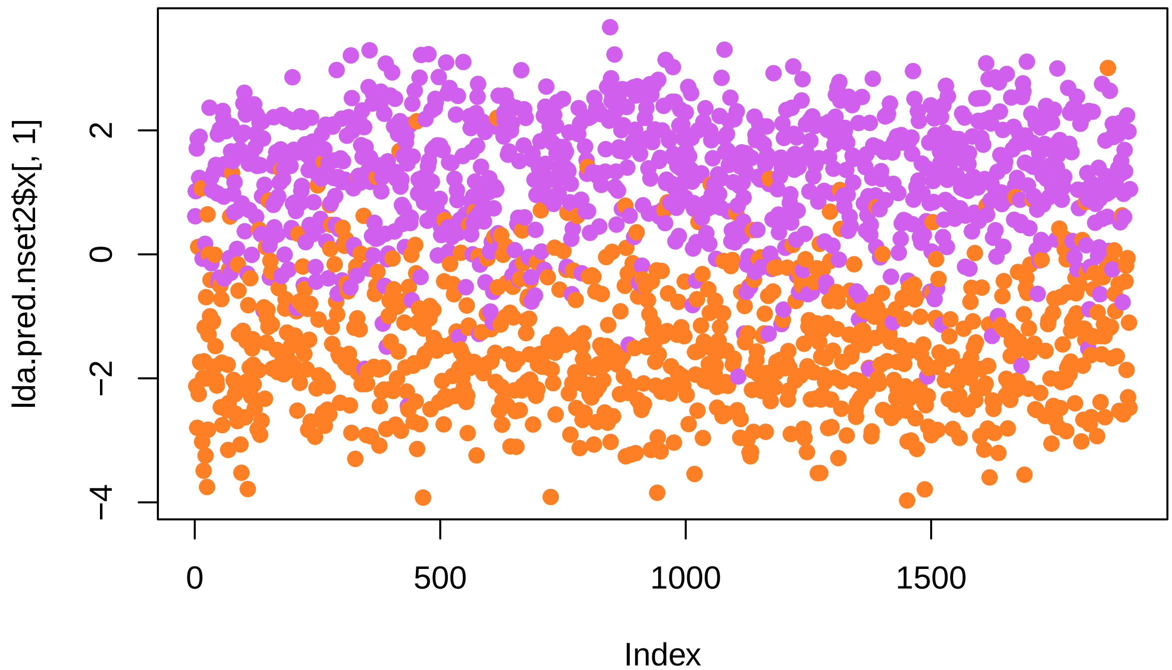
```
##             Var1         Freq
## 53   bNumofTower  0.410788350
## 55   rNumofTower -0.393820917
## 63    rNumofKill -0.113585803
## 61    bNumofKill  0.099056262
## 59   rNumofInhib -0.097574668
## 58   rFirstInhib -0.067208399
## 48   bNumofBaron  0.057220602
## 50  bNumofHerald -0.054479083
## 45  bNumofDragon  0.050641272
## 47  rNumofDragon -0.049729926
## 3     gamelength  0.045057809
## 56   bFirstInhib  0.029179549
## 51  rNumofHerald -0.027240986
## 49   rNumofBaron -0.025040673
## 62    rFirstKill -0.016548768
## 60    bFirstKill  0.014072594
## 54   rFirstTower -0.011681959
## 52   bFirstTower  0.008780541
## 44  bFirstDragon  0.001974395
## 57   bNumofInhib  0.001730735
```

Examining the top 20 explantory variables, we can see that the explantory variables named "number of ***" are especially important. It's reasonable, since these explantory variables are too strong as they are the data of game when the whole game is over. For example, if a team has more kills when the game is end, we can make an intuitive guess that that team win the game. And in terms of prediction, these explantory variables do not help the analysis. We actually cannot achive them until the end of the game. To achive the model for prediction, we should drop these explantory variables. (also the "game length")

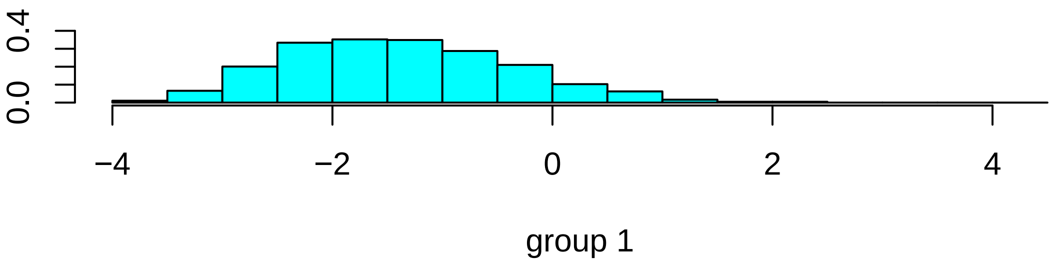
```r
name = "Numof|gamelength"
set1 = set1.full[,-getCols(set1.full, name)]
set2 = set2.full[,-getCols(set2.full, name)]
set3 = set3.full[,-getCols(set3.full, name)]
nset1 = nset1.full[,-getCols(nset1.full, name)]
nset2 = nset2.full[,-getCols(nset2.full, name)]
nset3 = nset3.full[,-getCols(nset3.full, name)]
```

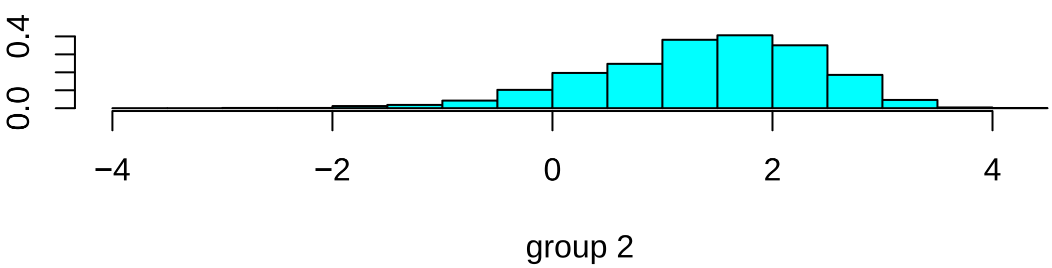
Then try lda again:

```r
lda.fit = lda(bResult~. , data = nset1)
lda.pred.nset1 <- predict(lda.fit, nset1)
lda.pred.nset2 <- predict(lda.fit, nset2)
#lda.fit
class.col <- ifelse(nset2$bResult==1,y=53,n=464)
plot(lda.pred.nset2$x[,1], col=colors()[class.col], pch = 19)
```




```
plot(lda.fit)
```






```
lda.pred.train <- lda.pred.nset1$class
lda.pred.val <- predict(lda.fit, nset2)$class
lda.pred.test <- predict(lda.fit, nset3)$class
mean(ifelse(lda.pred.train == nset1$bResult, yes=0, no=1))
```

```
## [1] 0.09265092
```

```
mean(ifelse(lda.pred.val == nset2$bResult, yes=0, no=1))
```

```
## [1] 0.1013123
```

```r
mean(ifelse(lda.pred.test == nset3$bResult, yes=0, no=1))
```

```
## [1] 0.09081365
```

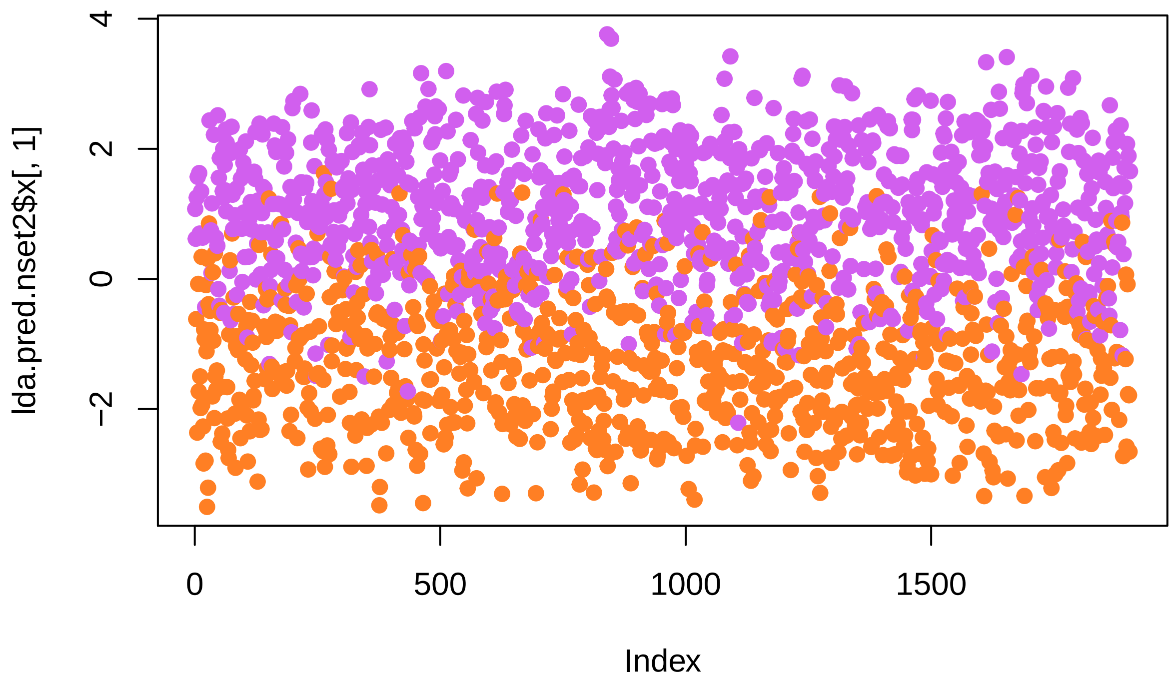
```r
lda.table = as.data.frame(as.table(lda.fit$scaling))
lda.table = lda.table[order(abs(lda.table$Freq), decreasing = TRUE),]
print.data.frame(lda.table[1:20,c(1,3)])
```

```
##                Var1          Freq
## 45      bFirstTower  0.0442744476
## 47      bFirstInhib -0.0422787937
## 48      rFirstInhib  0.0414374952
## 46      rFirstTower -0.0370357324
## 49       bFirstKill  0.0066132611
## 44     rFirstDragon -0.0036243264
## 50       rFirstKill  0.0036218007
## 12  redSupportChamp  0.0028242302
## 6      blueADCChamp  0.0022695492
## 9    redJungleChamp  0.0018962756
## 5   blueMiddleChamp -0.0013468863
## 7  blueSupportChamp  0.0011142474
## 11      redADCChamp  0.0009726689
## 10   redMiddleChamp  0.0007457420
## 8       redTopChamp -0.0006365685
## 4   blueJungleChamp -0.0005671495
## 3      blueTopChamp -0.0003013770
## 33    goldblueADC30  0.0002507009
## 43     bFirstDragon -0.0002440582
## 21 goldblueJungle30  0.0002435573
```

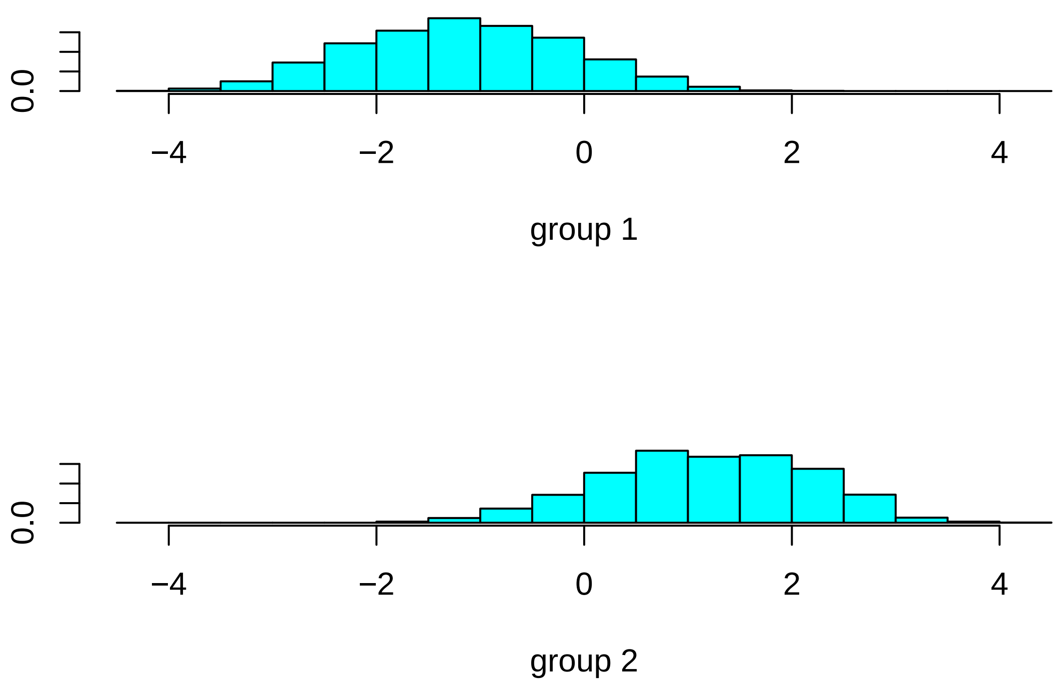
(option) delete the “first **" term (TODO: reason)

```r
name = "Numof|gamelength|First"
set1 = set1.full[,-getCols(set1.full, name)]
set2 = set2.full[,-getCols(set2.full, name)]
set3 = set3.full[,-getCols(set3.full, name)]
nset1 = nset1.full[,-getCols(nset1.full, name)]
nset2 = nset2.full[,-getCols(nset2.full, name)]
nset3 = nset3.full[,-getCols(nset3.full, name)]
```

```r
lda.fit = lda(bResult~. , data = nset1)
lda.pred.nset1 <- predict(lda.fit, nset1)
lda.pred.nset2 <- predict(lda.fit, nset2)
#lda.fit
class.col <- ifelse(nset2$bResult==1,y=53,n=464)
plot(lda.pred.nset2$x[,1], col=colors()[class.col], pch = 19)
```




```
plot(lda.fit)
```




```
lda.pred.train <- lda.pred.nset1$class
lda.pred.val <- predict(lda.fit, nset2)$class
lda.pred.test <- predict(lda.fit, nset3)$class
mean(ifelse(lda.pred.train == nset1$bResult, yes=0, no=1))
```

```
## [1] 0.1225722
```

```
mean(ifelse(lda.pred.val == nset2$bResult, yes=0, no=1))
```

```
## [1] 0.1333333
```

```r
mean(ifelse(lda.pred.test == nset3$bResult, yes=0, no=1))
```

```
## [1] 0.128084
```

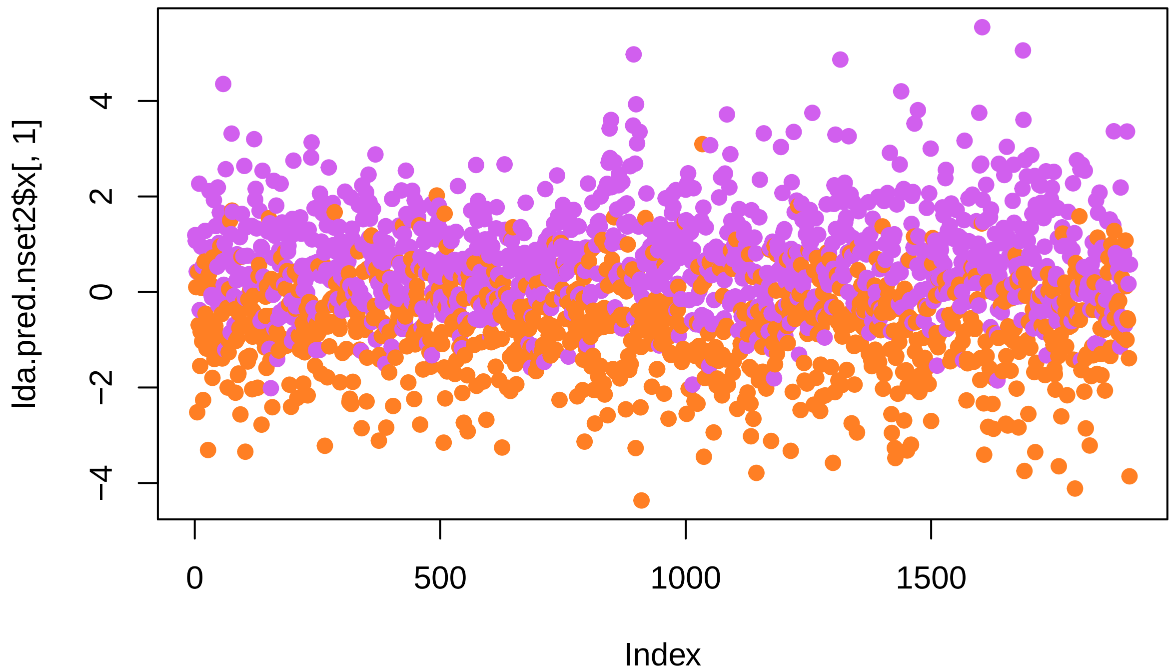
```r
lda.table = as.data.frame(as.table(lda.fit$scaling))
lda.table = lda.table[order(abs(lda.table$Freq), decreasing = TRUE),]
print.data.frame(lda.table[1:20,c(1,3)])
```

```
##                  Var1          Freq
## 12   redSupportChamp  0.0029063322
## 9     redJungleChamp  0.0027204556
## 5    blueMiddleChamp -0.0013393192
## 10    redMiddleChamp  0.0012563808
## 8        redTopChamp -0.0011045086
## 4    blueJungleChamp -0.0005574673
## 7   blueSupportChamp  0.0005469909
## 3       blueTopChamp -0.0003970469
## 6       blueADCChamp  0.0003105739
## 21  goldblueJungle30  0.0002916389
## 33     goldblueADC30  0.0002775524
## 36      goldredADC30 -0.0002564034
## 30   goldredMiddle30 -0.0002481973
## 1        blueTeamTag  0.0002468375
## 27  goldblueMiddle30  0.0002233387
## 20  goldblueJungle20 -0.0002138735
## 24   goldredJungle30 -0.0001980228
## 38 goldblueSupport20  0.0001849578
## 42  goldredSupport30 -0.0001798704
## 32     goldblueADC20 -0.0001721818
```

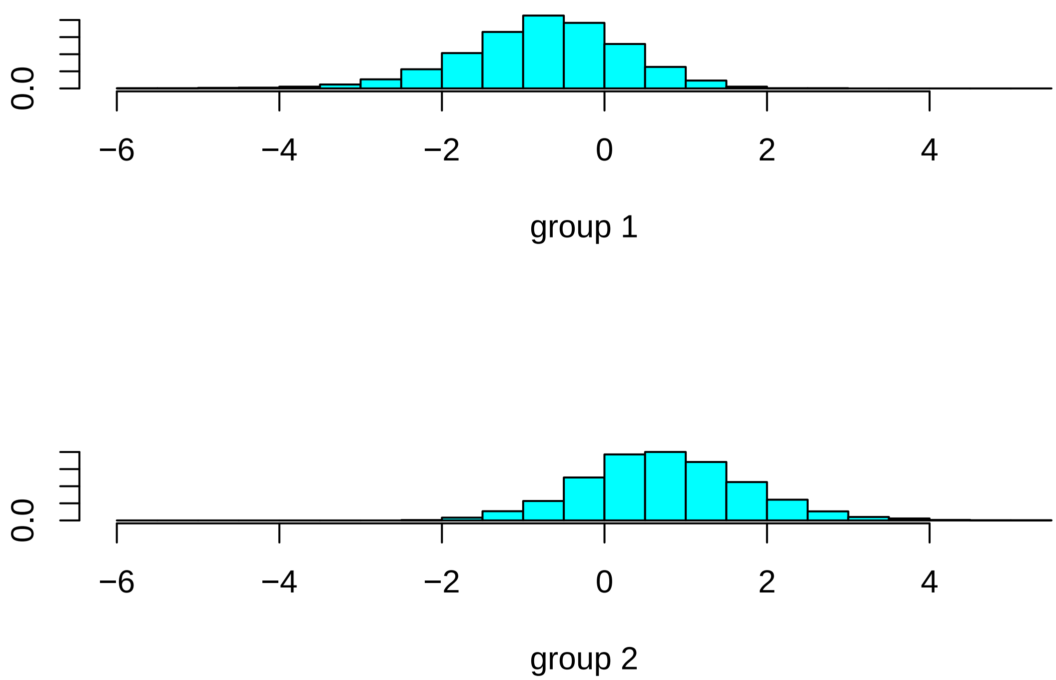
delete 30**

```r
name = "Numof|gamelength|First|30"
set1 = set1.full[,-getCols(set1.full, name)]
set2 = set2.full[,-getCols(set2.full, name)]
set3 = set3.full[,-getCols(set3.full, name)]
nset1 = nset1.full[,-getCols(nset1.full, name)]
nset2 = nset2.full[,-getCols(nset2.full, name)]
nset3 = nset3.full[,-getCols(nset3.full, name)]
```

```r
lda.fit = lda(bResult~. , data = nset1)
lda.pred.nset1 <- predict(lda.fit, nset1)
lda.pred.nset2 <- predict(lda.fit, nset2)
#lda.fit
class.col <- ifelse(nset2$bResult==1,y=53,n=464)
plot(lda.pred.nset2$x[,1], col=colors()[class.col], pch = 19)
```




```r
plot(lda.fit)
```




```r
lda.pred.train <- lda.pred.nset1$class
lda.pred.val <- predict(lda.fit, nset2)$class
lda.pred.test <- predict(lda.fit, nset3)$class
mean(ifelse(lda.pred.train == nset1$bResult, yes=0, no=1))
```

```
## [1] 0.2209974
```

```r
mean(ifelse(lda.pred.val == nset2$bResult, yes=0, no=1))
```

```
## [1] 0.2204724
```

```r
mean(ifelse(lda.pred.test == nset3$bResult, yes=0, no=1))
```

```
## [1] 0.2309711
```

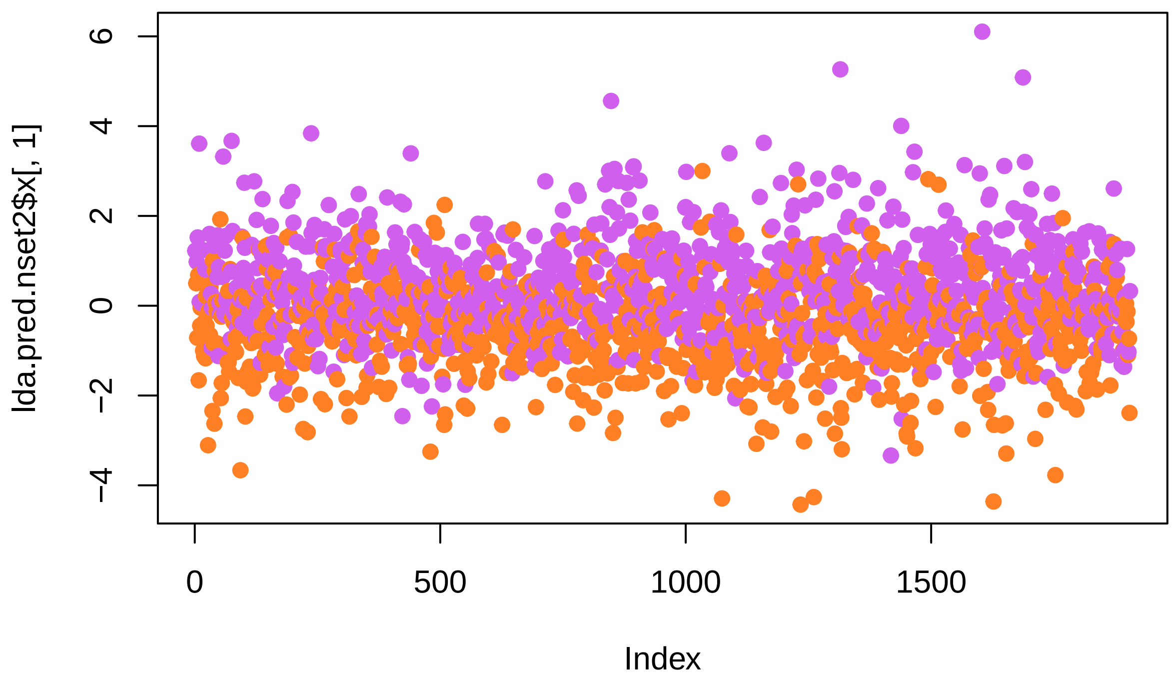
```r
lda.table = as.data.frame(as.table(lda.fit$scaling))
lda.table = lda.table[order(abs(lda.table$Freq), decreasing = TRUE),]
print.data.frame(lda.table[1:20,c(1,3)])
```

```
##                 Var1          Freq
## 9    redJungleChamp  0.0033971426
## 6      blueADCChamp  0.0024761486
## 10   redMiddleChamp  0.0010762361
## 12  redSupportChamp  0.0010715003
## 4   blueJungleChamp  0.0010658762
## 8       redTopChamp  0.0008103837
## 5   blueMiddleChamp -0.0005500536
## 24  goldredMiddle20 -0.0004544234
## 2         redTeamTag  0.0004084316
## 28     goldredADC20 -0.0003975645
## 3      blueTopChamp -0.0003790599
## 14    goldblueTop20  0.0003329866
## 22 goldblueMiddle20  0.0003208088
## 26    goldblueADC20  0.0003165761
## 18 goldblueJungle20  0.0003060609
## 16     goldredTop20 -0.0003044151
## 20  goldredJungle20 -0.0002523991
## 23  goldredMiddle10  0.0002401095
## 7  blueSupportChamp  0.0002280046
## 32 goldredSupport20 -0.0001837216
```

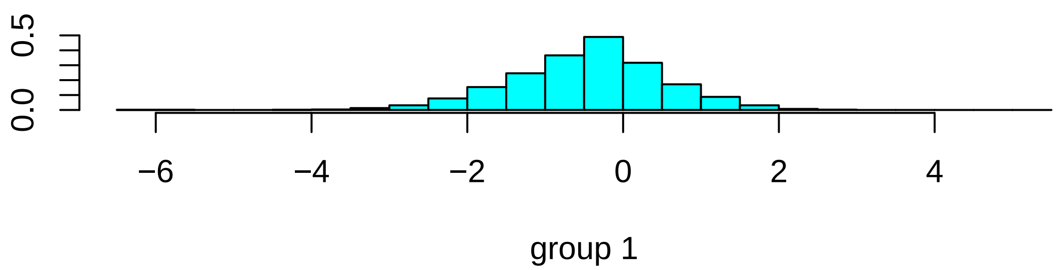
delete 20

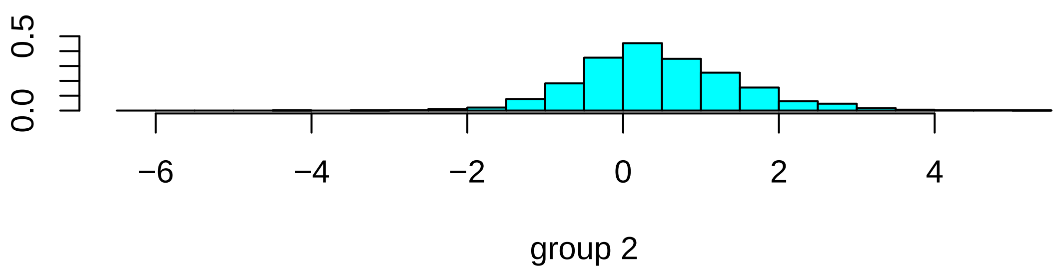
```r
name = "Numof|gamelength|20|30|First"
set1 = set1.full[,-getCols(set1.full, name)]
set2 = set2.full[,-getCols(set2.full, name)]
set3 = set3.full[,-getCols(set3.full, name)]
nset1 = nset1.full[,-getCols(nset1.full, name)]
nset2 = nset2.full[,-getCols(nset2.full, name)]
nset3 = nset3.full[,-getCols(nset3.full, name)]
```

```r
lda.fit = lda(bResult~. , data = nset1)
lda.pred.nset1 <- predict(lda.fit, nset1)
lda.pred.nset2 <- predict(lda.fit, nset2)
#lda.fit
class.col <- ifelse(nset2$bResult==1,y=53,n=464)
plot(lda.pred.nset2$x[,1], col=colors()[class.col], pch = 19)
```




```r
plot(lda.fit)
```






```r
lda.pred.train <- lda.pred.nset1$class
lda.pred.val <- predict(lda.fit, nset2)$class
lda.pred.test <- predict(lda.fit, nset3)$class
mean(ifelse(lda.pred.train == nset1$bResult, yes=0, no=1))
```

```
## [1] 0.3170604
```

```r
mean(ifelse(lda.pred.val == nset2$bResult, yes=0, no=1))
```

```
## [1] 0.3144357
```

```r
mean(ifelse(lda.pred.test == nset3$bResult, yes=0, no=1))
```

```
## [1] 0.3259843
```

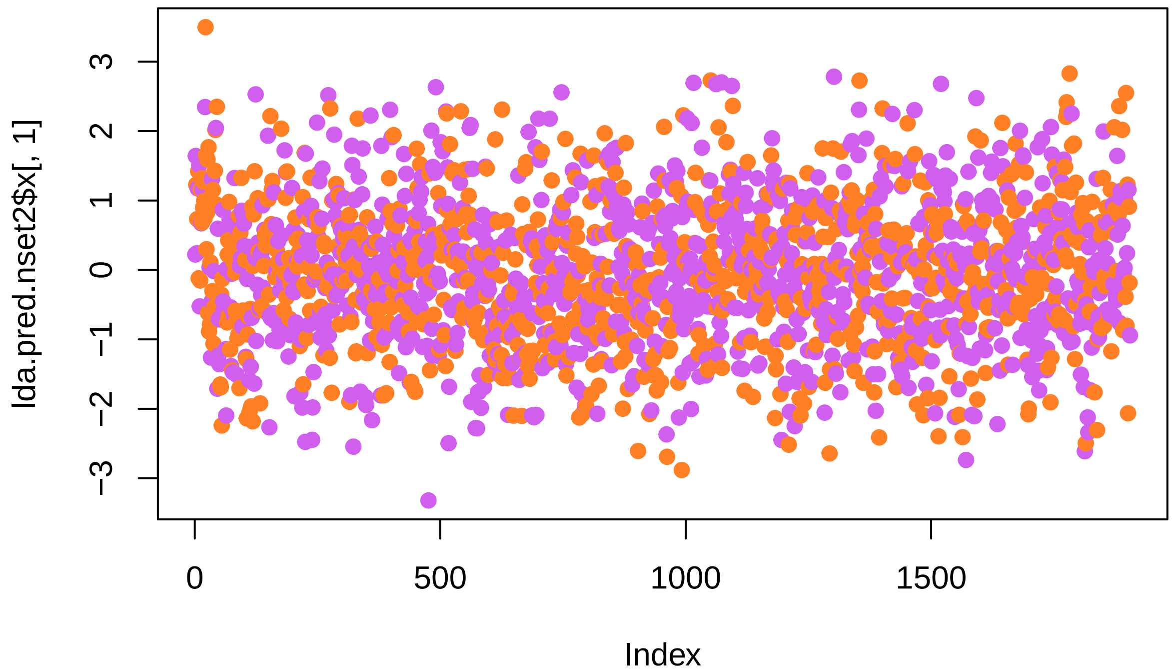
```r
lda.table = as.data.frame(as.table(lda.fit$scaling))
lda.table = lda.table[order(abs(lda.table$Freq), decreasing = TRUE),]
print.data.frame(lda.table[1:20,c(1,3)])
```

```
##                 Var1          Freq
## 6       blueADCChamp  0.0071321921
## 9     redJungleChamp  0.0049395902
## 11       redADCChamp -0.0043927982
## 4    blueJungleChamp  0.0041639468
## 10    redMiddleChamp  0.0015932032
## 8        redTopChamp  0.0012218416
## 7   blueSupportChamp  0.0011551937
## 14      goldredTop10 -0.0010533161
## 17  goldblueMiddle10  0.0010332178
## 19     goldblueADC10  0.0009979821
## 20      goldredADC10 -0.0009899164
## 18   goldredMiddle10 -0.0009704859
## 3       blueTopChamp -0.0008552236
## 13     goldblueTop10  0.0008352487
## 15  goldblueJungle10  0.0007716004
## 16   goldredJungle10 -0.0007567218
## 5    blueMiddleChamp -0.0005307952
## 21 goldblueSupport10  0.0003930752
## 22  goldredSupport10 -0.0003878921
## 2         redTeamTag  0.0003868535
```

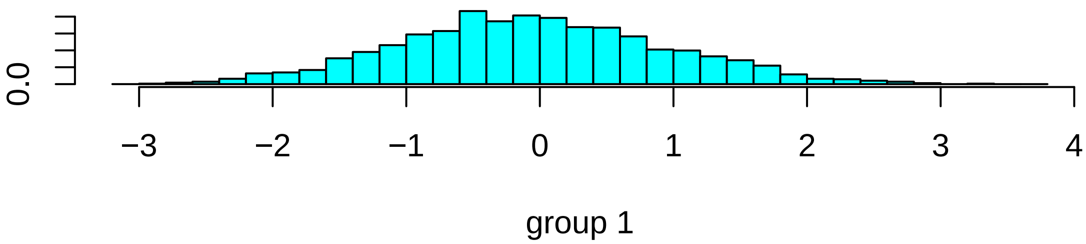
delete 10**

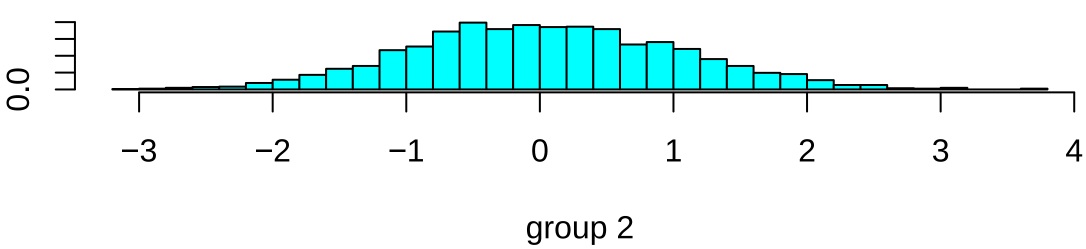
```r
name = "Numof|gamelength|10|20|30|First"
set1 = set1.full[,-getCols(set1.full, name)]
set2 = set2.full[,-getCols(set2.full, name)]
set3 = set3.full[,-getCols(set3.full, name)]
nset1 = nset1.full[,-getCols(nset1.full, name)]
nset2 = nset2.full[,-getCols(nset2.full, name)]
nset3 = nset3.full[,-getCols(nset3.full, name)]
```

```r
lda.fit = lda(bResult~. , data = nset1)
lda.pred.nset1 <- predict(lda.fit, nset1)
lda.pred.nset2 <- predict(lda.fit, nset2)
#lda.fit
class.col <- ifelse(nset2$bResult==1,y=53,n=464)
plot(lda.pred.nset2$x[,1], col=colors()[class.col], pch = 19)
```




```
plot(lda.fit)
```






```
lda.pred.train <- lda.pred.nset1$class
lda.pred.val <- predict(lda.fit, nset2)$class
lda.pred.test <- predict(lda.fit, nset3)$class
mean(ifelse(lda.pred.train == nset1$bResult, yes=0, no=1))
```

```
## [1] 0.452231
```

```
mean(ifelse(lda.pred.val == nset2$bResult, yes=0, no=1))
```

```
## [1] 0.4608924
```

```r
mean(ifelse(lda.pred.test == nset3$bResult, yes=0, no=1))
```

```
## [1] 0.4598425
```

```r
lda.table = as.data.frame(as.table(lda.fit$scaling))
lda.table = lda.table[order(abs(lda.table$Freq), decreasing = TRUE),]
print.data.frame(lda.table[1:20,c(1,3)])
```

```
##                  Var1          Freq
## 4     blueJungleChamp  0.0455370853
## 6        blueADCChamp  0.0407048160
## 9      redJungleChamp  0.0290550138
## 11        redADCChamp -0.0287936087
## 7    blueSupportChamp  0.0234590155
## 3        blueTopChamp -0.0115003342
## 12    redSupportChamp -0.0109086831
## 10     redMiddleChamp  0.0070048759
## 5     blueMiddleChamp -0.0044080017
## 1         blueTeamTag -0.0027425780
## 2          redTeamTag  0.0002854603
## 8         redTopChamp -0.0002391423
## NA               <NA>            NA
## NA.1             <NA>            NA
## NA.2             <NA>            NA
## NA.3             <NA>            NA
## NA.4             <NA>            NA
## NA.5             <NA>            NA
## NA.6             <NA>            NA
## NA.7             <NA>            NA
```

1. Final data set selection

```r
name = "Numof|gamelength|20|30|First|Champ|Tag"
set1.prediction.10 = set1.full[,-getCols(set1.full, name)]
set2.prediction.10 = set2.full[,-getCols(set2.full, name)]
set3.prediction.10 = set3.full[,-getCols(set3.full, name)]
nset1.prediction.10 = nset1.full[,-getCols(nset1.full, name)]
nset2.prediction.10 = nset2.full[,-getCols(nset2.full, name)]
nset3.prediction.10 = nset3.full[,-getCols(nset3.full, name)]
```

```r
name = "Numof|gamelength|30|First|Champ|Tag"
set1.prediction.20 = set1.full[,-getCols(set1.full, name)]
set2.prediction.20 = set2.full[,-getCols(set2.full, name)]
set3.prediction.20 = set3.full[,-getCols(set3.full, name)]
nset1.prediction.20 = nset1.full[,-getCols(nset1.full, name)]
nset2.prediction.20 = nset2.full[,-getCols(nset2.full, name)]
nset3.prediction.20 = nset3.full[,-getCols(nset3.full, name)]
```

```r
name = "Numof|gamelength|First|Champ|Tag"
set1.prediction.30 = set1.full[,-getCols(set1.full, name)]
set2.prediction.30 = set2.full[,-getCols(set2.full, name)]
set3.prediction.30 = set3.full[,-getCols(set3.full, name)]
nset1.prediction.30 = nset1.full[,-getCols(nset1.full, name)]
nset2.prediction.30 = nset2.full[,-getCols(nset2.full, name)]
nset3.prediction.30 = nset3.full[,-getCols(nset3.full, name)]
```

```
set1.prediction.list = list(set1.prediction.10, set1.prediction.20, set1.prediction.30)
set2.prediction.list = list(set2.prediction.10, set2.prediction.20, set2.prediction.30)
set3.prediction.list = list(set3.prediction.10, set3.prediction.20, set3.prediction.30)
nset1.prediction.list = list(nset1.prediction.10, nset1.prediction.20, nset1.prediction.30)
nset2.prediction.list = list(nset2.prediction.10, nset2.prediction.20, nset2.prediction.30)
nset3.prediction.list = list(nset3.prediction.10, nset3.prediction.20, nset3.prediction.30)
title.list = list("at 10", "at 20", "at 30")
```

pca rotated data

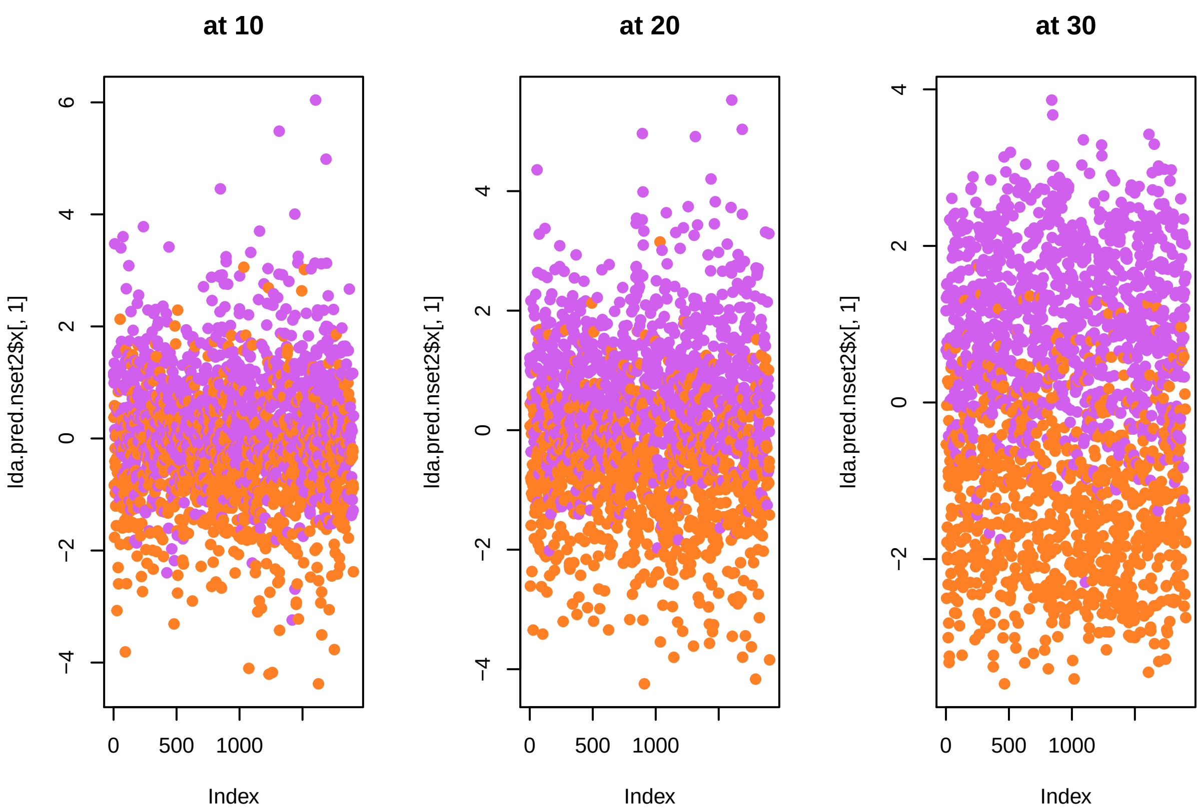
```
set1.pca.prediction.list = list()
set2.pca.prediction.list = list()
set3.pca.prediction.list = list()
for (i in 1:3) {
  set1.prediction = set1.prediction.list[[i]]
  set2.prediction = set2.prediction.list[[i]]
  set3.prediction = set3.prediction.list[[i]]
  pc <-  prcomp(x=set1.prediction[,-1], scale.=TRUE)
  set1.pca.prediction.list <- list.append(set1.pca.prediction.list,
                                         data.frame(bResult = set1.prediction$bResult, pc$x))
  set2.pca.prediction.list <- list.append(set2.pca.prediction.list,
                                         data.frame(bResult = set2.prediction$bResult,
                                                    predict(pc, newdata=set2.prediction[,-1])))
  set3.pca.prediction.list <- list.append(set3.pca.prediction.list,
                                         data.frame(bResult = set3.prediction$bResult,
                                                   predict(pc, newdata=set3.prediction[,-1])))
}
```

2. Model selection

2.1 LDA

```
lda.fit.list = list()
for (nset1.prediction in nset1.prediction.list) {
  #print(set1.prediction)
  lda.fit.list = list.append(lda.fit.list, lda(bResult~. , data = nset1.prediction))
}


lda.train.err = c()
lda.val.err = c()
lda.test.err = c()
par(mfrow=c(1,3))
for (i in 1:3) {
  lda.fit = lda.fit.list[[i]]
  nset1.prediction = nset1.prediction.list[[i]]
  nset2.prediction = nset2.prediction.list[[i]]
  nset3.prediction = nset3.prediction.list[[i]]
  title = title.list[[i]]
  lda.pred.nset1 = predict(lda.fit, nset1.prediction)
  lda.pred.nset2 = predict(lda.fit, nset2.prediction)
  #lda.fit
  class.col <- ifelse(nset2$bResult==1,y=53,n=464)
  plot(lda.pred.nset2$x[,1], col=colors()[class.col], pch = 19, main = title)
  #plot(lda.fit)
```

```
  lda.pred.train <- lda.pred.nset1$class
  lda.pred.val <- predict(lda.fit, nset2.prediction)$class
  lda.pred.test <- predict(lda.fit, nset3.prediction)$class
  lda.train.err = c(lda.train.err, mean(ifelse(lda.pred.train == nset1.prediction$bResult, yes=0, no=1)
  lda.val.err = c(lda.val.err, mean(ifelse(lda.pred.val == nset2.prediction$bResult, yes=0, no=1)))
  lda.test.err = c(lda.test.err, mean(ifelse(lda.pred.test == nset3.prediction$bResult, yes=0, no=1)))
}
```




```
lda.train.err
```

```
## [1] 0.3149606 0.2233596 0.1244094
```

```
lda.val.err
```

```
## [1] 0.3070866 0.2215223 0.1312336
```

```
lda.test.err
```

```
## [1] 0.3275591 0.2283465 0.1270341
```

2.2 QDA

```
qda.fit.list = list()
for (nset1.prediction in nset1.prediction.list) {
  #print(set1.prediction)
  qda.fit.list = list.append(qda.fit.list, qda(bResult~. , data = nset1.prediction))
}


qda.train.err = c()
qda.val.err = c()
qda.test.err = c()
```

```r
for (i in 1:3) {
  qda.fit = qda.fit.list[[i]]
  nset1.prediction = nset1.prediction.list[[i]]
  nset2.prediction = nset2.prediction.list[[i]]
  nset3.prediction = nset3.prediction.list[[i]]
  title = title.list[[i]]
  qda.pred.train <- predict(qda.fit, nset1.prediction)$class
  qda.pred.val <- predict(qda.fit, nset2.prediction)$class
  qda.pred.test <- predict(qda.fit, nset3.prediction)$class
  qda.train.err = c(qda.train.err, mean(ifelse(qda.pred.train == nset1.prediction$bResult, yes=0, no=1)
  qda.val.err = c(qda.val.err, mean(ifelse(qda.pred.val == nset2.prediction$bResult, yes=0, no=1)))
  qda.test.err = c(qda.test.err, mean(ifelse(qda.pred.test == nset3.prediction$bResult, yes=0, no=1)))
}

qda.train.err
```

```
## [1] 0.3312336 0.2380577 0.1320210
```

```r
qda.val.err
```

```
## [1] 0.3191601 0.2524934 0.1522310
```

```r
qda.test.err
```

```
## [1] 0.3517060 0.2530184 0.1522310
```

2.3 logistic

```r
logit.fit.list = list()
for (nset1.prediction in nset1.prediction.list) {
  #print(set1.prediction)
  logit.fit.list = list.append(logit.fit.list, glm(bResult - 1~. , data = nset1.prediction, family = bi
}


logit.train.err = c()
logit.val.err = c()
logit.test.err = c()
for (i in 1:3) {
  logit.fit = logit.fit.list[[i]]
  nset1.prediction = nset1.prediction.list[[i]]
  nset2.prediction = nset2.prediction.list[[i]]
  nset3.prediction = nset3.prediction.list[[i]]
  title = title.list[[i]]
  logit.pred.train <- ifelse(predict(logit.fit, nset1.prediction, type="response") < 0.5, 1, 2)
  logit.pred.val <- ifelse(predict(logit.fit, nset2.prediction, type="response") < 0.5, 1, 2)
  logit.pred.test <- ifelse(predict(logit.fit, nset3.prediction, type="response") < 0.5, 1, 2)
  logit.train.err = c(logit.train.err, mean(ifelse(logit.pred.train == nset1.prediction$bResult, yes=0,
  logit.val.err = c(logit.val.err, mean(ifelse(logit.pred.val == nset2.prediction$bResult, yes=0, no=1)
  logit.test.err = c(logit.test.err, mean(ifelse(logit.pred.test == nset3.prediction$bResult, yes=0, no
}

logit.train.err
```

```
## [1] 0.3165354 0.2233596 0.1225722
```

```
logit.val.err
```

```
## [1] 0.3039370 0.2251969 0.1307087
```

```
logit.test.err
```

```
## [1] 0.3270341 0.2246719 0.1291339
```

(option) with pca rotation (?)

```
logit.pca.fit.list = list()
for (nset1.prediction in set1.pca.prediction.list) {
  #print(nset1.prediction)
  logit.pca.fit.list = list.append(logit.pca.fit.list, glm(bResult~. , data = nset1.prediction, family =
}


logit.pca.train.err = c()
logit.pca.val.err = c()
logit.pca.test.err = c()
logit.train.prob.list = list()
for (i in 1:3) {
  logit.pca.fit = logit.pca.fit.list[[i]]
  nset1.prediction = set1.pca.prediction.list[[i]]
  nset2.prediction = set2.pca.prediction.list[[i]]
  nset3.prediction = set3.pca.prediction.list[[i]]
  title = title.list[[i]]
  logit.train.prob.list = list.append(logit.train.prob.list, predict(logit.pca.fit, nset1.prediction, ty
  logit.pca.pred.train <- ifelse(predict(logit.pca.fit, nset1.prediction, type="response") < 0.5, 1, 2)
  logit.pca.pred.val <- ifelse(predict(logit.pca.fit, nset2.prediction, type="response") < 0.5, 1, 2)
  logit.pca.pred.test <- ifelse(predict(logit.pca.fit, nset3.prediction, type="response") < 0.5, 1, 2)
  logit.pca.train.err = c(logit.pca.train.err, mean(ifelse(logit.pca.pred.train == as.numeric(nset1.pred
  logit.pca.val.err = c(logit.pca.val.err, mean(ifelse(logit.pca.pred.val == as.numeric(nset2.predictio
  logit.pca.test.err = c(logit.pca.test.err, mean(ifelse(logit.pca.pred.test == as.numeric(nset3.predict
}

logit.pca.train.err
```

```
## [1] 0.3165354 0.2233596 0.1225722
```

```
logit.pca.val.err
```

```
## [1] 0.3039370 0.2251969 0.1307087
```

```
logit.pca.test.err
```

```
## [1] 0.3270341 0.2246719 0.1291339
```

2.4 LASSO logistic

```
lasso.logit.fit.list = list()
for (nset1.prediction in nset1.prediction.list) {
  #print(set1.prediction)
  lasso.logit.fit.list = list.append(lasso.logit.fit.list, cv.glmnet(as.matrix(nset1.prediction[,-1]), r
}

lasso.variable.list = list()
```

```r
lasso.logit.1se.train.err = c()
lasso.logit.1se.val.err = c()
lasso.logit.1se.test.err = c()
lasso.logit.min.train.err = c()
lasso.logit.min.val.err = c()
lasso.logit.min.test.err = c()
for (i in 1:3) {
  lasso.logit.fit = lasso.logit.fit.list[[i]]
  nset1.prediction = nset1.prediction.list[[i]]
  nset2.prediction = nset2.prediction.list[[i]]
  nset3.prediction = nset3.prediction.list[[i]]
  title = title.list[[i]]
  lasso.variable.list <- list.append(lasso.variable.list, predict(lasso.logit.fit, as.matrix(nset1.predi
  lasso.logit.pred.train <- predict(lasso.logit.fit, as.matrix(nset1.prediction[,-1]), type="class", s=l
  lasso.logit.pred.val <- predict(lasso.logit.fit, as.matrix(nset2.prediction[,-1]), type="class", s=las
  lasso.logit.pred.test <- predict(lasso.logit.fit, as.matrix(nset3.prediction[,-1]), type="class", s=la
  lasso.logit.1se.train.err = c(lasso.logit.1se.train.err, mean(ifelse(lasso.logit.pred.train == nset1.p
  lasso.logit.1se.val.err = c(lasso.logit.1se.val.err, mean(ifelse(lasso.logit.pred.val == nset2.predict
  lasso.logit.1se.test.err = c(lasso.logit.1se.test.err, mean(ifelse(lasso.logit.pred.test == nset3.pred
  lasso.logit.pred.train <- predict(lasso.logit.fit, as.matrix(nset1.prediction[,-1]), type="class", s=l
  lasso.logit.pred.val <- predict(lasso.logit.fit, as.matrix(nset2.prediction[,-1]), type="class", s=las
  lasso.logit.pred.test <- predict(lasso.logit.fit, as.matrix(nset3.prediction[,-1]), type="class", s=la
  lasso.logit.min.train.err = c(lasso.logit.min.train.err, mean(ifelse(lasso.logit.pred.train == nset1.p
  lasso.logit.min.val.err = c(lasso.logit.min.val.err, mean(ifelse(lasso.logit.pred.val == nset2.predict
  lasso.logit.min.test.err = c(lasso.logit.min.test.err, mean(ifelse(lasso.logit.pred.test == nset3.pred
}

lasso.logit.1se.train.err
```

```
## [1] 0.3162730 0.2241470 0.1270341
```

```r
lasso.logit.1se.val.err
```

```
## [1] 0.3154856 0.2225722 0.1338583
```

```r
lasso.logit.1se.test.err
```

```
## [1] 0.3275591 0.2272966 0.1291339
```

```r
lasso.logit.min.train.err
```

```
## [1] 0.3162730 0.2238845 0.1278215
```

```r
lasso.logit.min.val.err
```

```
## [1] 0.3034121 0.2225722 0.1312336
```

```r
lasso.logit.min.test.err
```

```
## [1] 0.3275591 0.2262467 0.1291339
```

```r
coef(lasso.logit.fit.list[[1]],s='lambda.1se',exact=TRUE)[[1]]
```

```
## 11 x 1 sparse Matrix of class "dgCMatrix"
##                              1
## (Intercept)      -2.011900e-01
## goldblueTop10    -2.622985e-04
## goldredTop10      3.535871e-04
```

```
## goldblueJungle10  -2.889760e-04
## goldredJungle10    2.564655e-04
## goldblueMiddle10  -3.561603e-04
## goldredMiddle10    3.473625e-04
## goldblueADC10     -3.726350e-04
## goldredADC10       3.799433e-04
## goldblueSupport10 -4.553304e-05
## goldredSupport10   2.001430e-05
```

```
coef(lasso.logit.fit.list[[2]],s='lambda.1se',exact=TRUE)[[1]]
```

```
## 21 x 1 sparse Matrix of class "dgCMatrix"
##                               1
## (Intercept)       -6.630985e-01
## goldblueTop10      .
## goldblueTop20     -1.690440e-04
## goldredTop10       .
## goldredTop20       2.053124e-04
## goldblueJungle10   .
## goldblueJungle20  -1.614956e-04
## goldredJungle10    .
## goldredJungle20    1.499272e-04
## goldblueMiddle10   .
## goldblueMiddle20  -2.140704e-04
## goldredMiddle10    .
## goldredMiddle20    2.615721e-04
## goldblueADC10      .
## goldblueADC20     -2.299511e-04
## goldredADC10       .
## goldredADC20       2.538956e-04
## goldblueSupport10  .
## goldblueSupport20 -7.847875e-05
## goldredSupport10   .
## goldredSupport20   5.679283e-05
```

```
coef(lasso.logit.fit.list[[3]],s='lambda.1se',exact=TRUE)[[1]]
```

```
## 31 x 1 sparse Matrix of class "dgCMatrix"
##                               1
## (Intercept)       -8.458597e-01
## goldblueTop10      .
## goldblueTop20      .
## goldblueTop30     -1.203618e-04
## goldredTop10       .
## goldredTop20       .
## goldredTop30       1.497625e-04
## goldblueJungle10   .
## goldblueJungle20   .
## goldblueJungle30  -1.080547e-04
## goldredJungle10    .
## goldredJungle20    .
## goldredJungle30    1.134042e-04
## goldblueMiddle10   .
## goldblueMiddle20   .
## goldblueMiddle30  -1.806173e-04
```

```
## goldredMiddle10    .
## goldredMiddle20    .
## goldredMiddle30    2.179847e-04
## goldblueADC10      .
## goldblueADC20      .
## goldblueADC30     -1.812957e-04
## goldredADC10       .
## goldredADC20       .
## goldredADC30       2.098924e-04
## goldblueSupport10  .
## goldblueSupport20  .
## goldblueSupport30 -1.382088e-04
## goldredSupport10   .
## goldredSupport20   .
## goldredSupport30   9.325413e-05
```

```
coef(lasso.logit.fit.list[[2]],s='lambda.min',exact=TRUE)[[1]]
```

```
## 21 x 1 sparse Matrix of class "dgCMatrix"
##                               1
## (Intercept)       -6.721365e-01
## goldblueTop10      1.784715e-05
## goldblueTop20     -2.335584e-04
## goldredTop10       .
## goldredTop20       2.637568e-04
## goldblueJungle10   .
## goldblueJungle20  -2.008038e-04
## goldredJungle10    .
## goldredJungle20    1.943511e-04
## goldblueMiddle10   .
## goldblueMiddle20  -2.567918e-04
## goldredMiddle10   -6.844015e-05
## goldredMiddle20    3.337530e-04
## goldblueADC10     -3.776322e-05
## goldblueADC20     -2.571756e-04
## goldredADC10       .
## goldredADC20       2.964127e-04
## goldblueSupport10  .
## goldblueSupport20 -1.497697e-04
## goldredSupport10   .
## goldredSupport20   1.254019e-04
```

2.5 Naive Bayes w/o kernel

2.5.1. without rotation

```
naive.k.fit.list = list()
for (i in 1:3) {
  nset1.prediction = nset1.prediction.list[[i]]
  set1.prediction = set1.prediction.list[[i]]
  naive.k.fit.list = list.append(naive.k.fit.list, naiveBayes(x=nset1.prediction[,-1], y=set1.predictio
}



naive.k.train.err = c()
naive.k.val.err = c()
```

```r
naive.k.test.err = c()
for (i in 1:3) {
  naive.k.fit = naive.k.fit.list[[i]]
  set1.prediction = set1.prediction.list[[i]]
  set2.prediction = set2.prediction.list[[i]]
  set3.prediction = set3.prediction.list[[i]]
  nset1.prediction = nset1.prediction.list[[i]]
  nset2.prediction = nset2.prediction.list[[i]]
  nset3.prediction = nset3.prediction.list[[i]]
  title = title.list[[i]]
  naive.k.pred.train <- predict(naive.k.fit, newdata=nset1.prediction[,-1], type="class")
  naive.k.pred.val <- predict(naive.k.fit, newdata=nset2.prediction[,-1], type="class")
  naive.k.pred.test <- predict(naive.k.fit, newdata=nset3.prediction[,-1], type="class")
  naive.k.train.err = c(naive.k.train.err, mean(ifelse(naive.k.pred.train == set1.prediction$bResult, ye
  naive.k.val.err = c(naive.k.val.err, mean(ifelse(naive.k.pred.val == set2.prediction$bResult, yes=0, n
  naive.k.test.err = c(naive.k.test.err, mean(ifelse(naive.k.pred.test == set3.prediction$bResult, yes=
}

naive.k.train.err
```

```
## [1] 0.3265092 0.2343832 0.1517060
```

```r
naive.k.val.err
```

```
## [1] 0.3107612 0.2362205 0.1627297
```

```r
naive.k.test.err
```

```
## [1] 0.3406824 0.2467192 0.1601050
```

2.5.2 with pca rotation

```r
naive.k.pca.fit.list = list()
for (i in 1:3) {
  set1.prediction = set1.pca.prediction.list[[i]]
  naive.k.pca.fit.list = list.append(naive.k.pca.fit.list, naiveBayes(x=set1.prediction[,-1], y=set1.pre
}


naive.k.pca.train.err = c()
naive.k.pca.val.err = c()
naive.k.pca.test.err = c()
for (i in 1:3) {
  naive.k.pca.fit = naive.k.pca.fit.list[[i]]
  set1.prediction = set1.pca.prediction.list[[i]]
  set2.prediction = set2.pca.prediction.list[[i]]
  set3.prediction = set3.pca.prediction.list[[i]]
  title = title.list[[i]]
  naive.k.pca.pred.train <- predict(naive.k.pca.fit, newdata=set1.prediction[,-1], type="class")
  naive.k.pca.pred.val <- predict(naive.k.pca.fit, newdata=set2.prediction[,-1], type="class")
  naive.k.pca.pred.test <- predict(naive.k.pca.fit, newdata=set3.prediction[,-1], type="class")
  naive.k.pca.train.err = c(naive.k.pca.train.err, mean(ifelse(naive.k.pca.pred.train == set1.predictio
  naive.k.pca.val.err = c(naive.k.pca.val.err, mean(ifelse(naive.k.pca.pred.val == set2.prediction$bResu
  naive.k.pca.test.err = c(naive.k.pca.test.err, mean(ifelse(naive.k.pca.pred.test == set3.prediction$bR
}

naive.k.pca.train.err
```

```
## [1] 0.3215223 0.2341207 0.1354331
```

```
naive.k.pca.val.err
```

```
## [1] 0.3107612 0.2362205 0.1396325
```

```
naive.k.pca.test.err
```

```
## [1] 0.3328084 0.2309711 0.1406824
```

2.6 logistic gam

For categorical variabels, set number of knots equal to the number of unique variables

```
gam.logit.fit.list = list()
for (i in 1:3) {
  nset1.prediction = nset1.prediction.list[[i]]
  set1.prediction = set1.prediction.list[[i]]
  gam.formula = as.formula(paste0(colnames(nset1.prediction)[1]," - 1~",paste0("s(",colnames(nset1.predi
  gam.logit.fit.list = list.append(gam.logit.fit.list, gam(data=nset1.prediction, formula = gam.formula
}



gam.logit.train.err = c()
gam.logit.val.err = c()
gam.logit.test.err = c()
for (i in 1:3) {
  gam.logit.fit = gam.logit.fit.list[[i]]
  set1.prediction = set1.prediction.list[[i]]
  set2.prediction = set2.prediction.list[[i]]
  set3.prediction = set3.prediction.list[[i]]
  nset1.prediction = nset1.prediction.list[[i]]
  nset2.prediction = nset2.prediction.list[[i]]
  nset3.prediction = nset3.prediction.list[[i]]
  title = title.list[[i]]
  gam.logit.pred.train <- as.numeric(predict(gam.logit.fit, nset1.prediction, type="link") > 0)
  gam.logit.pred.val <- as.numeric(predict(gam.logit.fit, nset2.prediction, type="link") > 0)
  gam.logit.pred.test <- as.numeric(predict(gam.logit.fit, nset3.prediction, type="link") > 0)
  gam.logit.train.err = c(gam.logit.train.err, mean(ifelse(gam.logit.pred.train == as.numeric(nset1.pred
  gam.logit.val.err = c(gam.logit.val.err, mean(ifelse(gam.logit.pred.val == as.numeric(nset2.prediction
  gam.logit.test.err = c(gam.logit.test.err, mean(ifelse(gam.logit.pred.test == as.numeric(nset3.predict
}

gam.logit.train.err
```

```
## [1] 0.3144357 0.2207349 0.1223097
```

```
gam.logit.val.err
```

```
## [1] 0.3091864 0.2325459 0.1364829
```

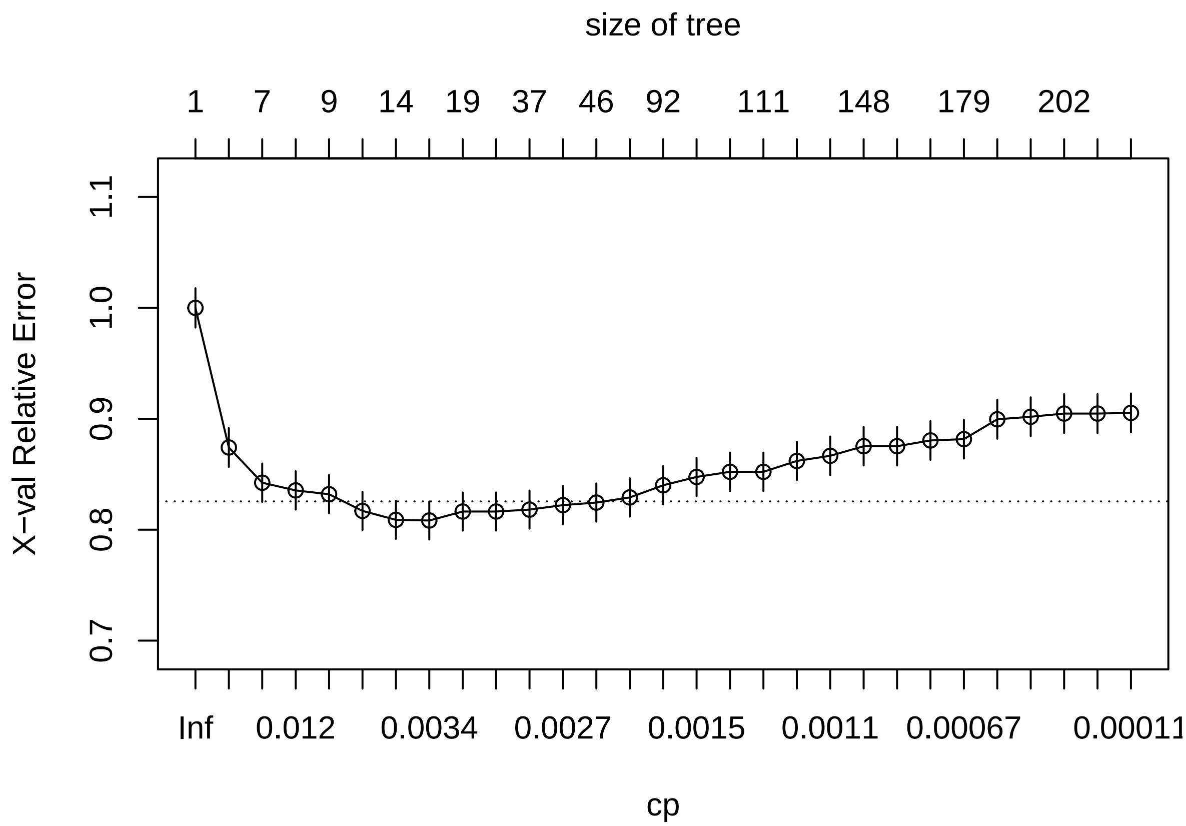
```
gam.logit.test.err
```

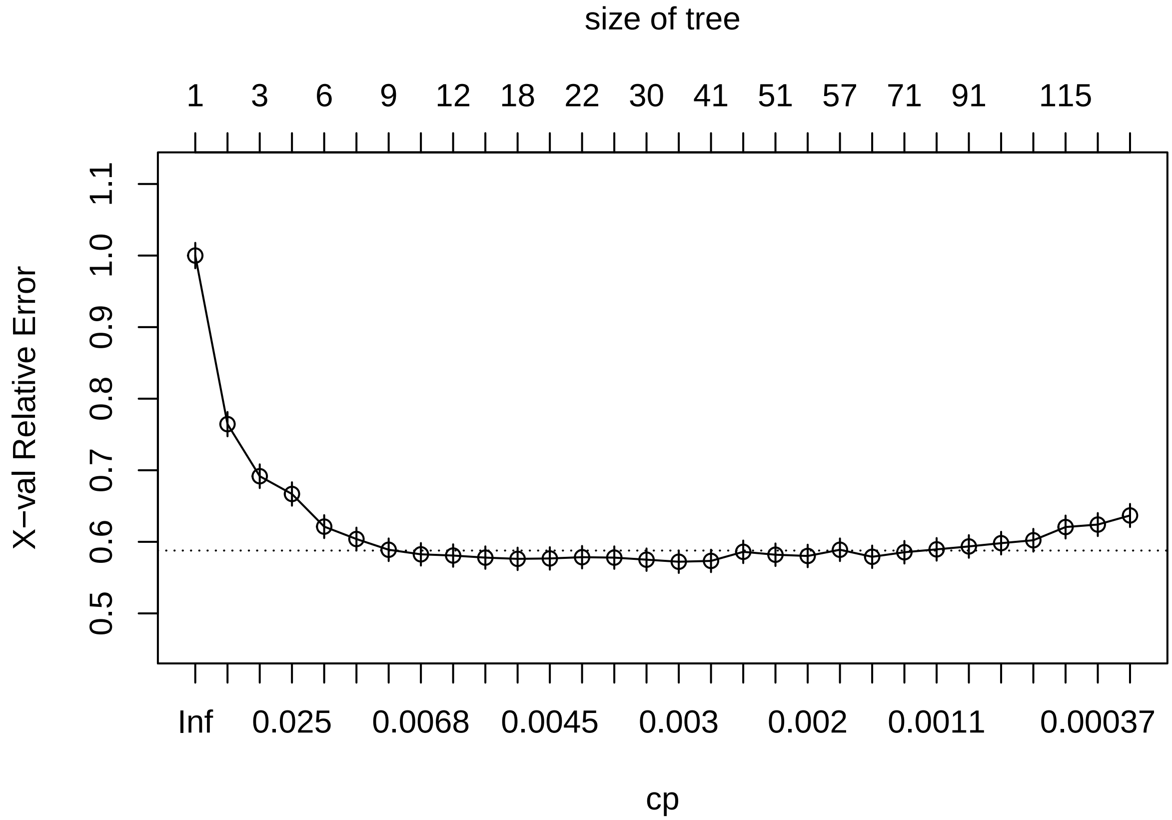
```
## [1] 0.3265092 0.2251969 0.1343832
```

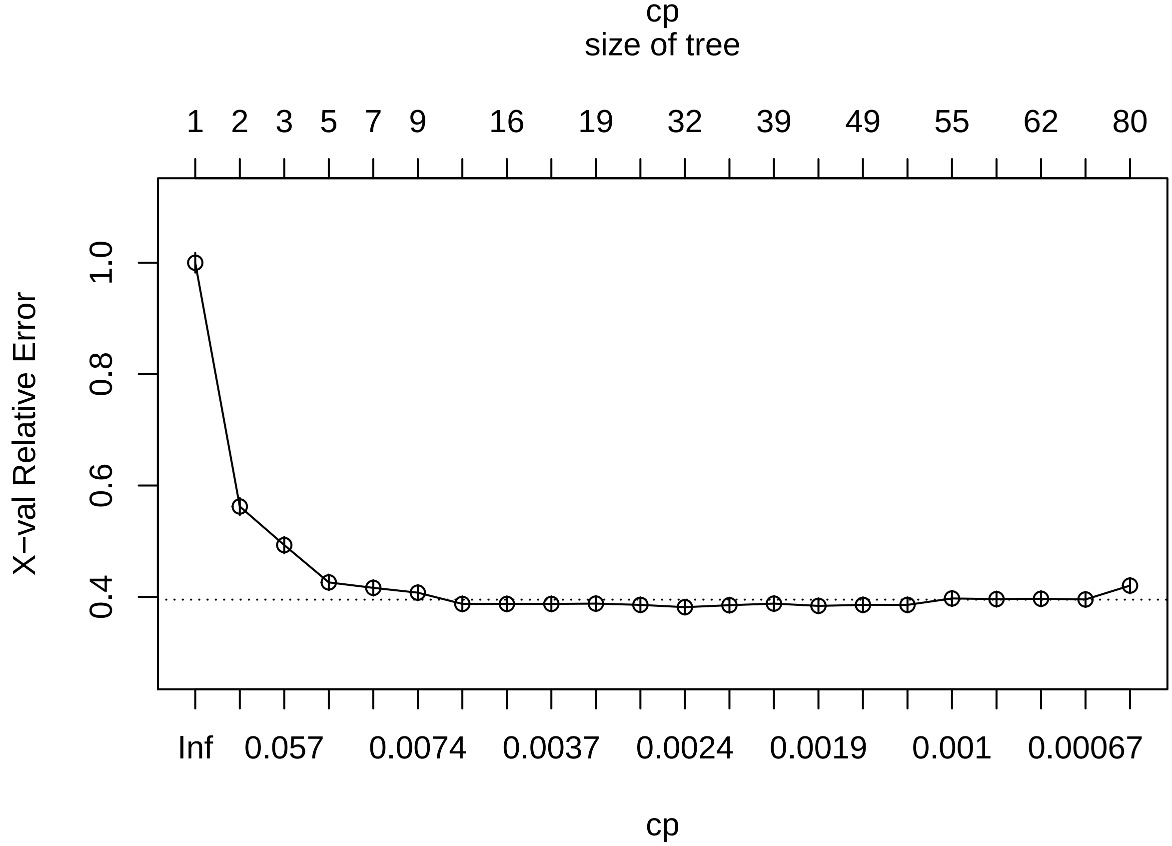
2.7 single classification tree

```
set.seed(441)
# should use validation set, but seems for wheat it's too small?
veh.tree.fit.list = list()
```

```
for (nset1.prediction in nset1.prediction.list) {
  veh.tree.fit.list = list.append(veh.tree.fit.list, veh.tree.fit <- rpart(data=nset1.prediction, bResul
}

for (veh.tree.fit in veh.tree.fit.list) {

  plotcp(veh.tree.fit)
}
```



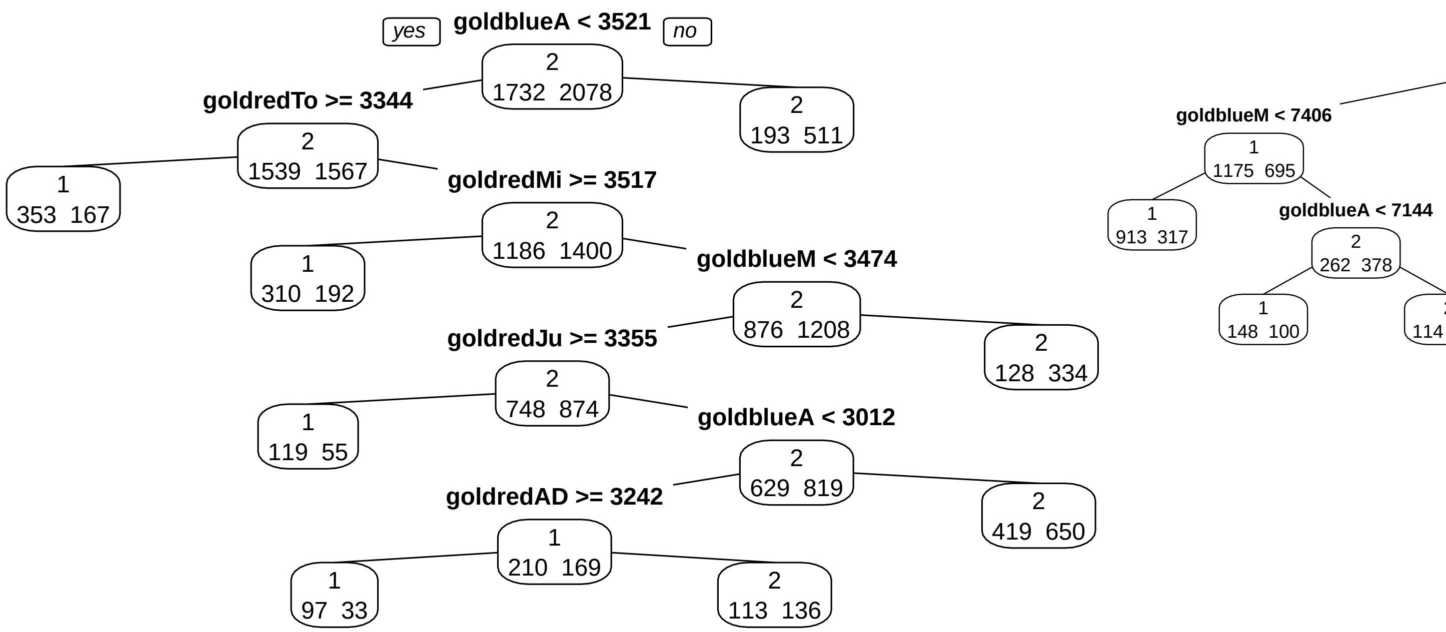


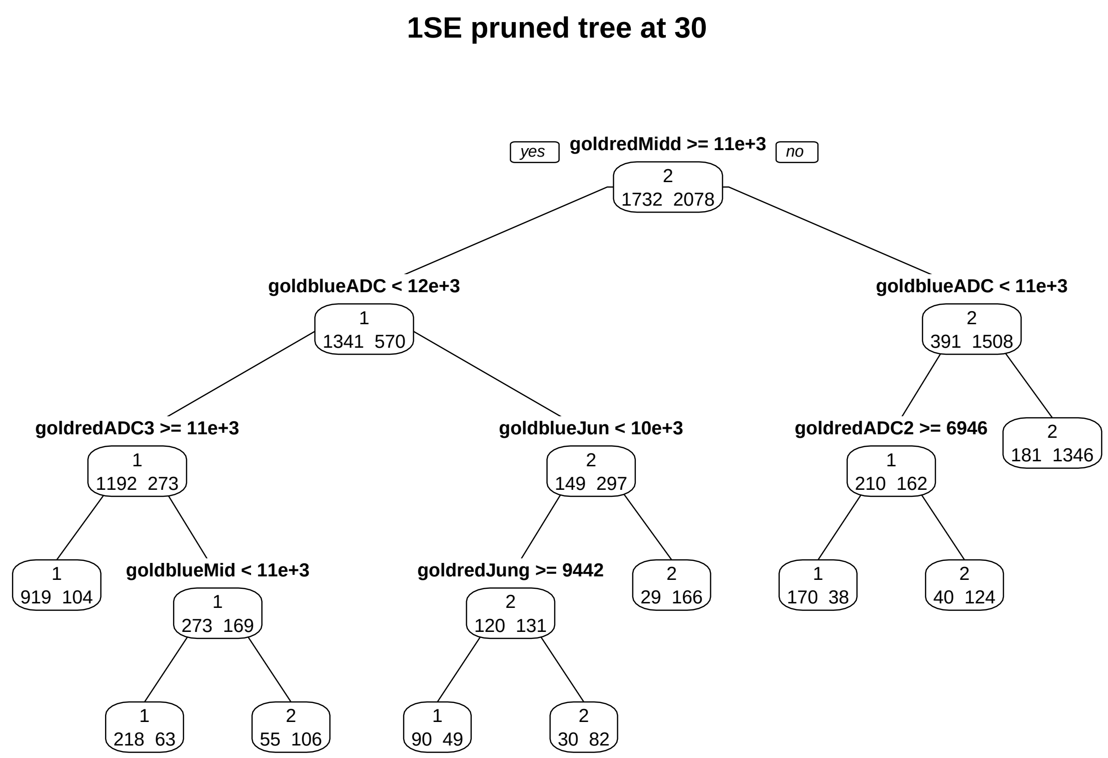



```
a = veh.tree.fit$cptable
```

```r
veh.tree.1se.fit.list = list()
tunedcp.list = c(0.012, 0.0068, 0.0074)
for (i in 1:3) {
  nset1.prediction = nset1.prediction.list[[i]]
  set1.prediction = set1.prediction.list[[i]]
  veh.tree.fit = veh.tree.fit.list[[i]]
  tunedcp = tunedcp.list[[i]]
  title = title.list[[i]]
  veh.tree.1se.fit.list = list.append(veh.tree.1se.fit.list, prune(veh.tree.fit, cp=tunedcp))
  prp(veh.tree.1se.fit.list[[i]], type=1, extra=1, main=paste0("1SE pruned tree ", title))
}
```

1SE pruned tree at 10






```
veh.tree.1se.train.err = c()
veh.tree.1se.val.err = c()
veh.tree.1se.test.err = c()
for (i in 1:3) {
  veh.tree.1se.fit = veh.tree.1se.fit.list[[i]]
  set1.prediction = set1.prediction.list[[i]]
  set2.prediction = set2.prediction.list[[i]]
  set3.prediction = set3.prediction.list[[i]]
  nset1.prediction = nset1.prediction.list[[i]]
  nset2.prediction = nset2.prediction.list[[i]]
  nset3.prediction = nset3.prediction.list[[i]]
  title = title.list[[i]]
  veh.tree.1se.pred.train <- predict(veh.tree.1se.fit, newdata=nset1.prediction[,-1], type="class")
  veh.tree.1se.pred.val <- predict(veh.tree.1se.fit, newdata=nset2.prediction[,-1], type="class")
  veh.tree.1se.pred.test <- predict(veh.tree.1se.fit, newdata=nset3.prediction[,-1], type="class")
  veh.tree.1se.train.err = c(veh.tree.1se.train.err, mean(ifelse(veh.tree.1se.pred.train == nset1.predi
  veh.tree.1se.val.err = c(veh.tree.1se.val.err, mean(ifelse(veh.tree.1se.pred.val == nset2.prediction$
  veh.tree.1se.test.err = c(veh.tree.1se.test.err, mean(ifelse(veh.tree.1se.pred.test == nset3.predicti
}

veh.tree.1se.train.err
```

```
## [1] 0.3412073 0.2475066 0.1545932
```

```
veh.tree.1se.val.err
```

```
## [1] 0.3711286 0.2776903 0.1790026
```

```
veh.tree.1se.test.err
```

```
## [1] 0.3979003 0.2708661 0.1784777
```

2.8 random forest

```r
veh.rftree.fit.list = list()
veh.rftree.fit.500 <- randomForest(data=nset2.prediction.list[[1]], as.factor(bResult)~.,
                       importance=TRUE, ntree=500, mtry=1, keep.forest=TRUE)
veh.rftree.fit.500  # more useful here
```

```
##
## Call:
##  randomForest(formula = as.factor(bResult) ~ ., data = nset2.prediction.list[[1]],      importance =
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 1
##
##         OOB estimate of  error rate: 32.07%
## Confusion matrix:
##     1   2 class.error
## 1 517 356   0.4077892
## 2 255 777   0.2470930
```

```r
veh.rftree.fit.1000 <- randomForest(data=nset2.prediction.list[[1]], as.factor(bResult)~.,
                       importance=TRUE, ntree=1000, mtry=1, keep.forest=TRUE)
veh.rftree.fit.1000  # more useful here
```

```
##
## Call:
##  randomForest(formula = as.factor(bResult) ~ ., data = nset2.prediction.list[[1]],      importance =
##                Type of random forest: classification
##                      Number of trees: 1000
## No. of variables tried at each split: 1
##
##         OOB estimate of  error rate: 32.39%
## Confusion matrix:
##     1   2 class.error
## 1 518 355   0.4066438
## 2 262 770   0.2538760
```

```r
veh.rftree.fit.1500 <- randomForest(data=nset2.prediction.list[[1]], as.factor(bResult)~.,
                       importance=TRUE, ntree=1500, mtry=1, keep.forest=TRUE)
veh.rftree.fit.1500  # more useful here
```

```
##
## Call:
##  randomForest(formula = as.factor(bResult) ~ ., data = nset2.prediction.list[[1]],      importance =
##                Type of random forest: classification
##                      Number of trees: 1500
## No. of variables tried at each split: 1
##
##         OOB estimate of  error rate: 32.34%
## Confusion matrix:
##     1   2 class.error
## 1 513 360   0.4123711
## 2 256 776   0.2480620
```

```r
veh.rftree.fit.list = list.append(veh.rftree.fit.list, veh.rftree.fit.1000)
```

```r
veh.rftree.fit.500 <- randomForest(data=nset2.prediction.list[[2]], as.factor(bResult)~.,
                       importance=TRUE, ntree=500, mtry=1, keep.forest=TRUE)
```

```
veh.rftree.fit.500  # more useful here
```

```
##
## Call:
##  randomForest(formula = as.factor(bResult) ~ ., data = nset2.prediction.list[[2]],      importance =
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 1
##
##         OOB estimate of  error rate: 23.04%
## Confusion matrix:
##     1   2 class.error
## 1 627 246   0.2817869
## 2 193 839   0.1870155
```

```
veh.rftree.fit.1000 <- randomForest(data=nset2.prediction.list[[2]], as.factor(bResult)~.,
                       importance=TRUE, ntree=1000, mtry=1, keep.forest=TRUE)
veh.rftree.fit.1000  # more useful here
```

```
##
## Call:
##  randomForest(formula = as.factor(bResult) ~ ., data = nset2.prediction.list[[2]],      importance =
##                Type of random forest: classification
##                      Number of trees: 1000
## No. of variables tried at each split: 1
##
##         OOB estimate of  error rate: 23.41%
## Confusion matrix:
##     1   2 class.error
## 1 626 247   0.2829324
## 2 199 833   0.1928295
```

```
veh.rftree.fit.1500 <- randomForest(data=nset2.prediction.list[[2]], as.factor(bResult)~.,
                       importance=TRUE, ntree=1500, mtry=1, keep.forest=TRUE)
veh.rftree.fit.1500  # more useful here
```

```
##
## Call:
##  randomForest(formula = as.factor(bResult) ~ ., data = nset2.prediction.list[[2]],      importance =
##                Type of random forest: classification
##                      Number of trees: 1500
## No. of variables tried at each split: 1
##
##         OOB estimate of  error rate: 23.78%
## Confusion matrix:
##     1   2 class.error
## 1 620 253   0.2898053
## 2 200 832   0.1937984
```

```
veh.rftree.fit.list = list.append(veh.rftree.fit.list, veh.rftree.fit.1000)
```

```
veh.rftree.fit.500 <- randomForest(data=nset2.prediction.list[[3]], as.factor(bResult)~.,
                      importance=TRUE, ntree=500, mtry=1, keep.forest=TRUE)
veh.rftree.fit.500  # more useful here
```

```
##
```

```
## Call:
##  randomForest(formula = as.factor(bResult) ~ ., data = nset2.prediction.list[[3]],      importance =
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 1
##
##         OOB estimate of  error rate: 14.28%
## Confusion matrix:
##     1   2 class.error
## 1 719 154   0.1764032
## 2 118 914   0.1143411
```

```
veh.rftree.fit.1000 <- randomForest(data=nset2.prediction.list[[3]], as.factor(bResult)~.,
                       importance=TRUE, ntree=1000, mtry=1, keep.forest=TRUE)
veh.rftree.fit.1000  # more useful here
```

```
##
## Call:
##  randomForest(formula = as.factor(bResult) ~ ., data = nset2.prediction.list[[3]],      importance =
##                Type of random forest: classification
##                      Number of trees: 1000
## No. of variables tried at each split: 1
##
##         OOB estimate of  error rate: 14.44%
## Confusion matrix:
##     1   2 class.error
## 1 718 155   0.1775487
## 2 120 912   0.1162791
```

```
veh.rftree.fit.1500 <- randomForest(data=nset2.prediction.list[[3]], as.factor(bResult)~.,
                       importance=TRUE, ntree=1500, mtry=1, keep.forest=TRUE)
veh.rftree.fit.1500  # more useful here
```

```
##
## Call:
##  randomForest(formula = as.factor(bResult) ~ ., data = nset2.prediction.list[[3]],      importance =
##                Type of random forest: classification
##                      Number of trees: 1500
## No. of variables tried at each split: 1
##
##         OOB estimate of  error rate: 14.38%
## Confusion matrix:
##     1   2 class.error
## 1 715 158   0.1809851
## 2 116 916   0.1124031
```

```
veh.rftree.fit.list = list.append(veh.rftree.fit.list, veh.rftree.fit.1000)
```

```
veh.rftree.train.err = c()
veh.rftree.val.err = c()
veh.rftree.test.err = c()
for (veh.rftree.fit in veh.rftree.fit.list) {
  a = round(importance(veh.rftree.fit),3)
  print.table(as.table(a[order(a[,4], decreasing = TRUE),]))
}
```

```
##                        1      2 MeanDecreaseAccuracy MeanDecreaseGini
## goldredMiddle10    23.763  13.070               25.662          104.316
## goldblueJungle10   20.548   9.334               20.782           97.711
## goldredADC10       12.692  12.654               18.120           97.256
## goldredJungle10    13.503   9.395               16.534           95.226
## goldblueMiddle10   15.125   5.740               14.703           94.185
## goldblueADC10      15.269   8.889               17.439           93.857
## goldblueTop10      14.854   5.673               14.138           92.892
## goldredTop10        7.079   8.220               10.852           92.006
## goldredSupport10    4.253   8.410                9.167           89.255
## goldblueSupport10   7.112  -0.695                4.270           88.643
##                       1      2 MeanDecreaseAccuracy MeanDecreaseGini
## goldredADC20      26.870 24.668               36.125           62.010
## goldredMiddle20   23.904 26.901               34.965           61.590
## goldblueMiddle20  25.368 21.204               32.179           59.290
## goldblueADC20     22.793 21.784               30.862           58.954
## goldredSupport20  17.349 22.058               27.856           55.402
## goldblueJungle20  20.375 18.744               28.183           55.036
## goldredTop20      19.363 18.240               26.391           52.296
## goldblueTop20     22.064 16.249               26.215           51.737
## goldredJungle20   17.827 17.836               25.261           50.862
## goldblueSupport20 16.422 15.549               23.837           49.590
## goldredMiddle10   11.995  9.869               15.845           43.547
## goldredADC10       6.064  7.883               10.475           40.298
## goldblueJungle10   8.353  4.882               10.090           39.836
## goldredTop10       5.082  7.832                9.645           38.750
## goldredJungle10    9.048  4.445                9.936           38.487
## goldblueADC10      9.077  0.994                7.711           38.315
## goldblueMiddle10  11.163  2.297                9.364           38.194
## goldblueTop10     10.505  3.487               10.618           37.991
## goldredSupport10   1.956  3.723                4.080           36.499
## goldblueSupport10  6.780  2.596                6.874           36.412
##                       1      2 MeanDecreaseAccuracy MeanDecreaseGini
## goldredADC30      27.661 28.571               34.577           54.564
## goldredMiddle30   28.003 26.982               33.967           53.435
## goldredSupport30  24.852 25.053               33.365           47.208
## goldredJungle30   26.711 23.977               32.538           47.155
## goldblueSupport30 25.540 23.118               32.371           45.385
## goldblueADC30     22.212 23.102               29.298           43.883
## goldblueJungle30  22.880 21.894               29.019           43.127
## goldblueMiddle30  24.581 21.604               30.405           41.415
## goldredTop30      24.199 24.948               32.568           40.183
## goldblueTop30     25.254 25.277               32.352           37.995
## goldredADC20      15.385 16.395               22.454           33.341
## goldredMiddle20   14.259 18.212               23.283           31.812
## goldblueMiddle20  17.013 12.527               20.715           29.366
## goldblueADC20     13.588 11.838               18.430           28.211
## goldblueJungle20  13.075 11.944               18.537           27.935
## goldblueSupport20 15.280 10.142               19.197           27.881
## goldredSupport20   8.842 14.781               17.581           27.632
## goldredTop20      12.840 13.248               18.994           27.417
## goldblueTop20     13.520  8.001               16.025           27.219
## goldredJungle20   10.735 11.952               16.729           26.990
## goldredMiddle10    7.948  6.802               10.594           22.198
```

```
## goldredJungle10    8.454  2.750               8.215          21.323
## goldblueJungle10   6.860  4.497               8.189          20.693
## goldredADC10       4.740  7.328               8.828          20.421
## goldblueADC10     10.310  1.631               8.440          20.338
## goldredTop10       4.107  4.028               6.105          20.029
## goldblueMiddle10   5.906  0.206               4.377          19.966
## goldblueTop10      9.117  1.057               7.279          19.943
## goldblueSupport10  5.234  2.466               5.543          19.248
## goldredSupport10   2.825  3.625               4.622          19.036
```

```
for (i in 1:3) {
  veh.rftree.fit = veh.rftree.fit.list[[i]]
  set1.prediction = set1.prediction.list[[i]]
  set2.prediction = set2.prediction.list[[i]]
  set3.prediction = set3.prediction.list[[i]]
  nset1.prediction = nset1.prediction.list[[i]]
  nset2.prediction = nset2.prediction.list[[i]]
  nset3.prediction = nset3.prediction.list[[i]]
  title = title.list[[i]]
  veh.rftree.pred.train <- predict(veh.rftree.fit, nset1.prediction, type="response")
  veh.rftree.pred.val <- predict(veh.rftree.fit, nset2.prediction, type="response")
  veh.rftree.pred.test <- predict(veh.rftree.fit, nset3.prediction, type="response")
  veh.rftree.train.err = c(veh.rftree.train.err, mean(ifelse(veh.rftree.pred.train == nset1.prediction$b
  veh.rftree.val.err = c(veh.rftree.val.err, mean(ifelse(veh.rftree.pred.val == nset2.prediction$bResult
  veh.rftree.test.err = c(veh.rftree.test.err, mean(ifelse(veh.rftree.pred.test == nset3.prediction$bRes
}

veh.rftree.train.err
```

```
## [1] 0.3488189 0.2346457 0.1417323
```

```
veh.rftree.val.err
```

```
## [1] 0 0 0
```

```
veh.rftree.test.err
```

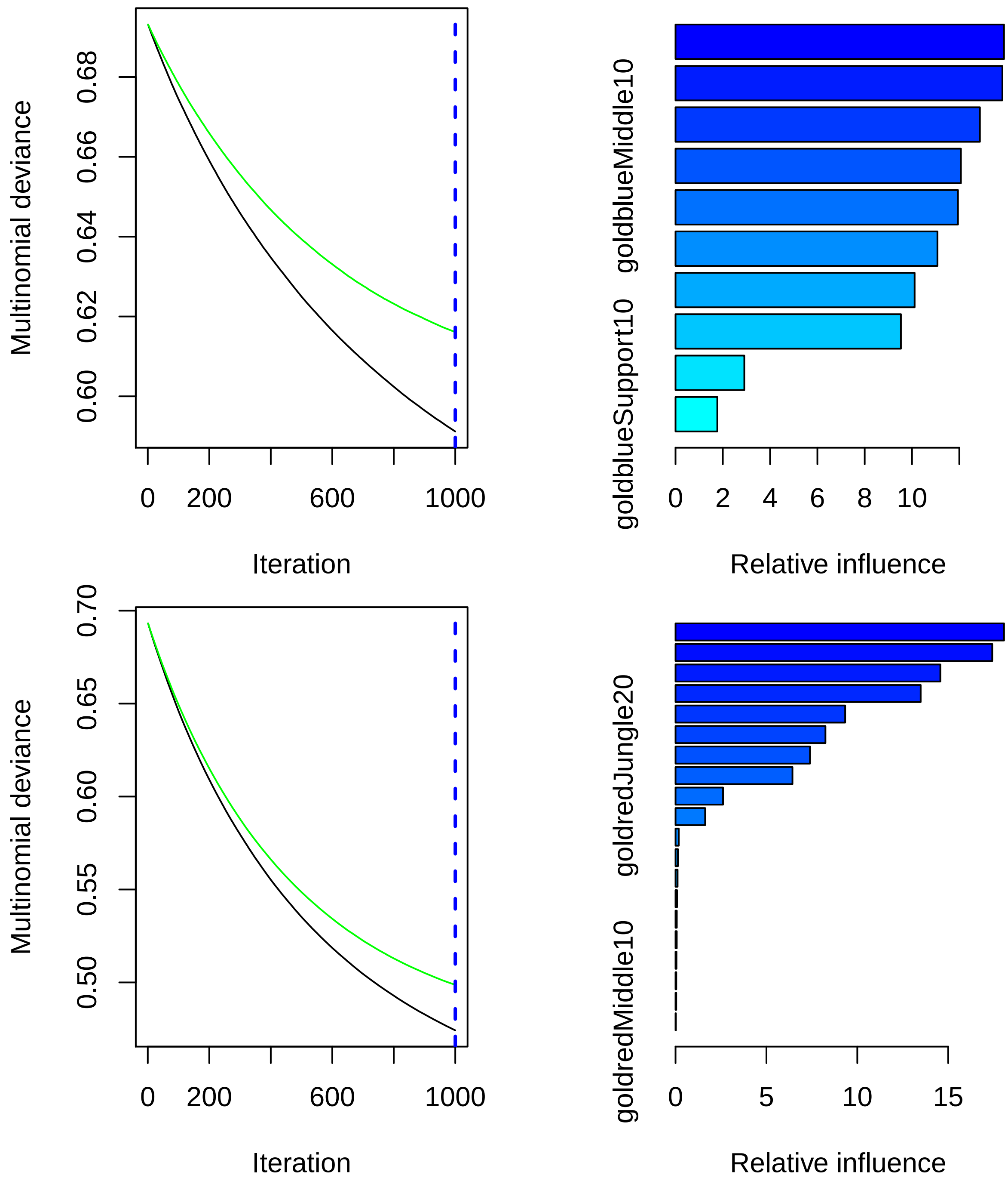
```
## [1] 0.3422572 0.2377953 0.1517060
```

2.9 gbm

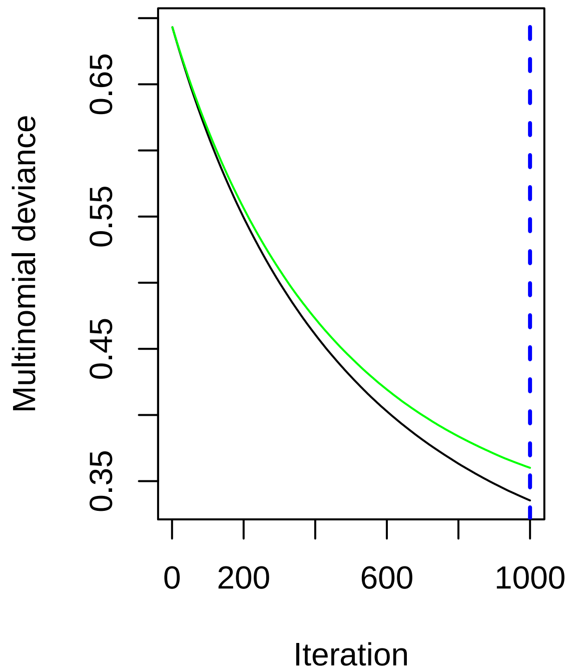
```
gbm.fit.list = list()
for (i in 1:3) {
  nset1.prediction = nset1.prediction.list[[i]]
  set1.prediction = set1.prediction.list[[i]]
  gbm.fit.list = list.append(gbm.fit.list, gbm(data=set1.prediction, formula=bResult~.,
                         distribution="multinomial", verbose=FALSE,
                         n.trees=1000, interaction.depth=6, shrinkage=0.001,
                         bag.fraction=0.5, cv.folds=5))
}
```

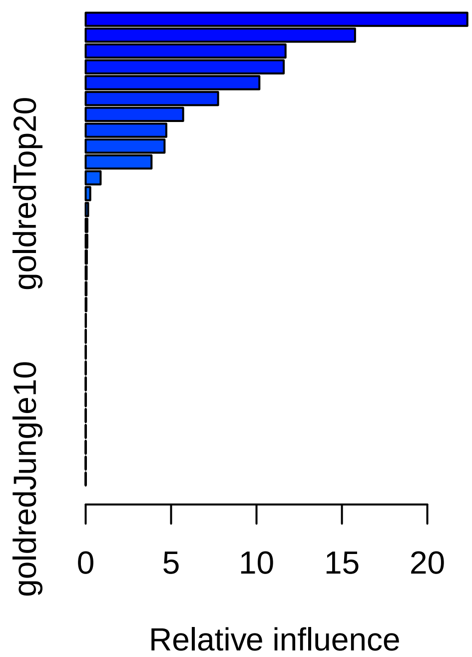
```
gbm.final.tree.list = list()
gbm.rel.inf.list = list()
for (gbm.fit in gbm.fit.list) {
  par(mfrow=c(1,2))
  ntrees.final <- gbm.perf(gbm.fit, method="cv" )
  gbm.rel.inf.list = list.append(gbm.rel.inf.list, summary(gbm.fit, n.trees=ntrees.final))
  gbm.final.tree.list <- list.append(gbm.final.tree.list, gbm.perf(gbm.fit,plot.it = FALSE, method="cv"
```

```
}
```



Multinomial deviance
0.60
0.62
0.64
0.66
0.68
0
200
600
1000
Iteration
goldblueSupport10
goldblueMiddle10
0
2
4
6
8
10
Relative influence
Multinomial deviance
0.50
0.55
0.60
0.65
0.70
0
200
600
1000
Iteration
goldredMiddle10
goldredJungle20
0
5
10
15
Relative influence






```r
for (i in 1:3) {
  print(gbm.rel.inf.list[[i]][1:10,])
}
```

```
##                                  var   rel.inf
## goldblueADC10          goldblueADC10 13.891762
## goldredTop10            goldredTop10 13.822808
## goldredADC10            goldredADC10 12.873100
## goldblueMiddle10    goldblueMiddle10 12.068605
## goldblueJungle10    goldblueJungle10 11.944500
## goldredMiddle10      goldredMiddle10 11.076888
## goldredJungle10      goldredJungle10 10.112194
## goldblueTop10          goldblueTop10  9.533046
## goldredSupport10    goldredSupport10  2.909511
## goldblueSupport10 goldblueSupport10  1.767586
##                                  var   rel.inf
## goldredADC20            goldredADC20 18.072477
## goldredMiddle20      goldredMiddle20 17.424878
## goldblueMiddle20    goldblueMiddle20 14.569090
## goldblueADC20          goldblueADC20 13.486917
## goldredTop20            goldredTop20  9.332578
## goldblueTop20          goldblueTop20  8.247221
## goldblueJungle20    goldblueJungle20  7.397704
## goldredJungle20      goldredJungle20  6.434238
## goldblueSupport20 goldblueSupport20  2.612758
## goldredSupport20    goldredSupport20  1.629303
##                                  var   rel.inf
## goldredMiddle30      goldredMiddle30 22.343114
## goldredADC30            goldredADC30 15.767191
## goldblueADC30          goldblueADC30 11.700690
## goldblueMiddle30    goldblueMiddle30 11.593735
## goldredJungle30      goldredJungle30 10.168727
## goldblueSupport30 goldblueSupport30  7.753663
```

```
## goldblueJungle30   goldblueJungle30  5.707497
## goldredTop30           goldredTop30  4.721555
## goldblueTop30         goldblueTop30  4.617801
## goldredSupport30   goldredSupport30  3.854478
```

```
gbm.train.err = c()
gbm.val.err = c()
gbm.test.err = c()
for (i in 1:3) {
  gbm.fit = gbm.fit.list[[i]]
  gbm.final.tree = gbm.final.tree.list[[i]]
  set1.prediction = set1.prediction.list[[i]]
  set2.prediction = set2.prediction.list[[i]]
  set3.prediction = set3.prediction.list[[i]]
  nset1.prediction = nset1.prediction.list[[i]]
  nset2.prediction = nset2.prediction.list[[i]]
  nset3.prediction = nset3.prediction.list[[i]]
  title = title.list[[i]]
  gbm.pred.train <- predict(gbm.fit, newdata=nset1.prediction, n.trees=gbm.final.tree, type="response")
  class.mul.train.final <- apply(gbm.pred.train[,,1], 1, which.max)
  gbm.pred.val <-predict(gbm.fit, newdata=nset2.prediction, n.trees=gbm.final.tree, type="response")
  class.mul.val.final <- apply(gbm.pred.val[,,1], 1, which.max)
  gbm.pred.test <- predict(gbm.fit, newdata=nset3.prediction, n.trees=gbm.final.tree, type="response")
  class.mul.test.final <- apply(gbm.pred.test[,,1], 1, which.max)
  gbm.train.err = c(gbm.train.err, mean(ifelse(class.mul.train.final == as.numeric(set1.prediction$bResu
  gbm.val.err = c(gbm.val.err, mean(ifelse(class.mul.val.final == as.numeric(set2.prediction$bResult),
  gbm.test.err = c(gbm.test.err, mean(ifelse(class.mul.test.final == as.numeric(set3.prediction$bResult)
}

gbm.train.err
```

```
## [1] 0.2897638 0.2023622 0.1123360
```

```
gbm.val.err
```

```
## [1] 0.3238845 0.2278215 0.1401575
```

```
gbm.test.err
```

```
## [1] 0.3391076 0.2393701 0.1333333
```

2.10 SVM

```
set.seed(441)
veh.tune <-  tune.svm(data=set2.prediction.list[[1]], bResult ~ ., kernel="radial", gamma = 10^(-10:-7)
```

```
summary(veh.tune)
```

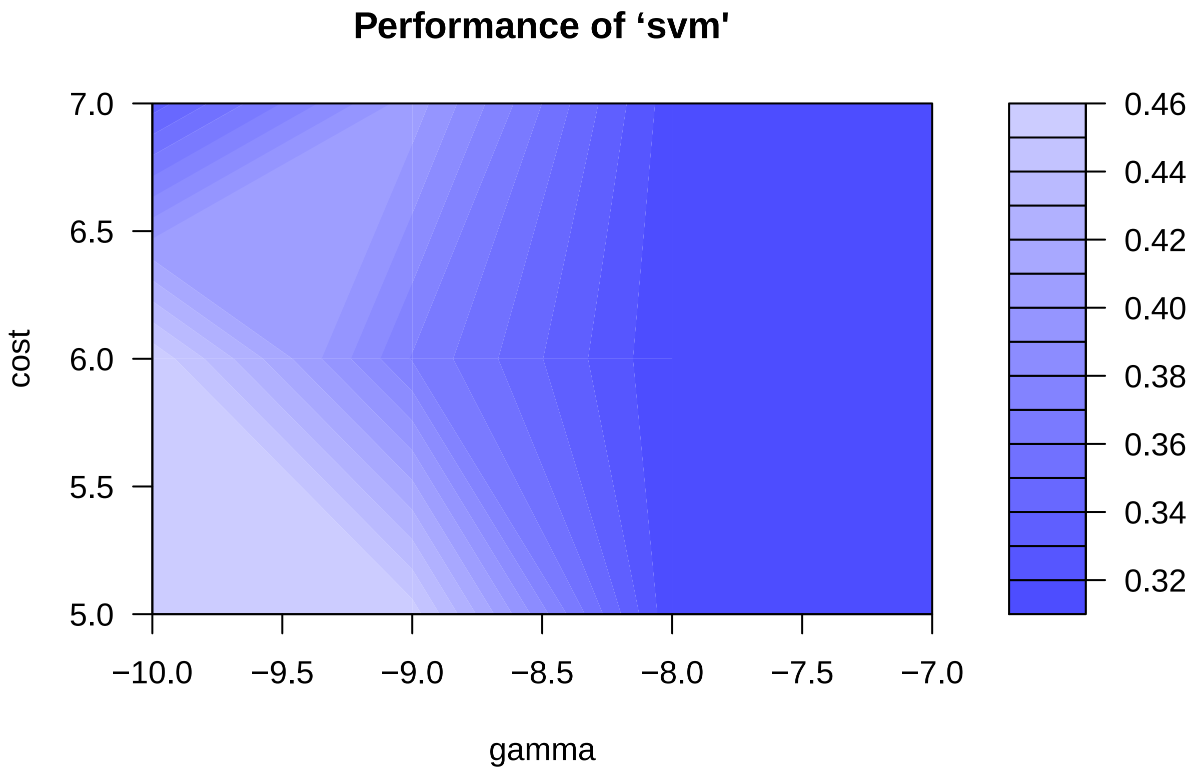
```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  gamma  cost
##  1e-08 1e+06
##
## - best performance: 0.3112648
```

```
##
## - Detailed performance results:
##    gamma  cost     error dispersion
## 1  1e-10 1e+05 0.4582557 0.04069512
## 2  1e-09 1e+05 0.4551061 0.04136905
## 3  1e-08 1e+05 0.3117994 0.04110384
## 4  1e-07 1e+05 0.3154698 0.04612424
## 5  1e-10 1e+06 0.4577322 0.04024181
## 6  1e-09 1e+06 0.3690879 0.04281470
## 7  1e-08 1e+06 0.3112648 0.04367140
## 8  1e-07 1e+06 0.3128410 0.04133893
## 9  1e-10 1e+07 0.3349132 0.04246174
## 10 1e-09 1e+07 0.4058556 0.04673983
## 11 1e-08 1e+07 0.3139157 0.04179841
## 12 1e-07 1e+07 0.3160017 0.03644151
```

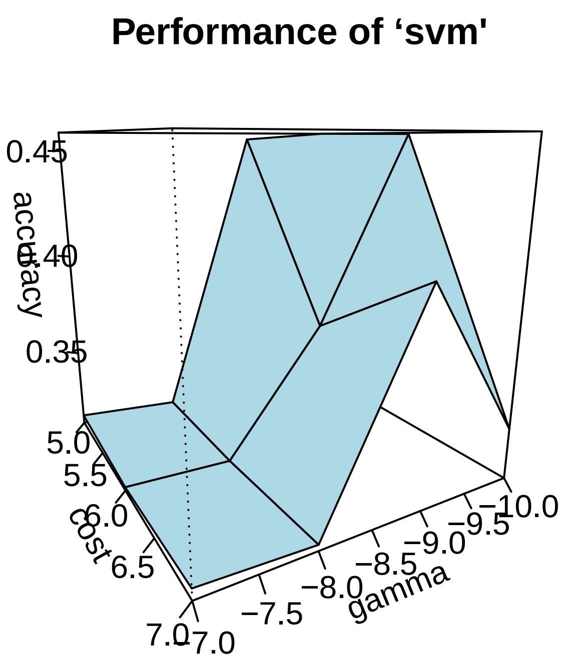
```
aa <- summary(veh.tune)$performances
aa[order(aa[,3]),]
```

```
##    gamma  cost     error dispersion
## 7  1e-08 1e+06 0.3112648 0.04367140
## 3  1e-08 1e+05 0.3117994 0.04110384
## 8  1e-07 1e+06 0.3128410 0.04133893
## 11 1e-08 1e+07 0.3139157 0.04179841
## 4  1e-07 1e+05 0.3154698 0.04612424
## 12 1e-07 1e+07 0.3160017 0.03644151
## 9  1e-10 1e+07 0.3349132 0.04246174
## 6  1e-09 1e+06 0.3690879 0.04281470
## 10 1e-09 1e+07 0.4058556 0.04673983
## 2  1e-09 1e+05 0.4551061 0.04136905
## 5  1e-10 1e+06 0.4577322 0.04024181
## 1  1e-10 1e+05 0.4582557 0.04069512
```

```
#### Note: Optimum is on edge of parameter space. Ought to pursue further (larger) costs
###x11(h=7, w=6, pointsize=12)
plot(veh.tune, type="contour", transform.x=log10, transform.y=log10)
```




```r
#x11(h=7, w=6, pointsize=12)
plot(veh.tune, type="perspective", transform.x=log10, transform.y=log10, theta=150)
```




```r
svm.train.err=c()
svm.val.err=c()
svm.test.err=c()
svm.fit <- svm(data=set1.prediction.list[[1]], bResult ~ ., kernel="radial", gamma=1e-8, cost=1e7, cros
svm.pred.train <- predict(svm.fit, newdata=set1.prediction.list[[1]])
svm.pred.val <- predict(svm.fit, newdata=set2.prediction.list[[1]])
```

```
svm.pred.test <- predict(svm.fit, newdata=set3.prediction.list[[1]])
svm.train.err = c(svm.train.err, mean(ifelse(svm.pred.train == set1.prediction.list[[1]]$bResult, yes=0
svm.val.err = c(svm.val.err, mean(ifelse(svm.pred.val == set2.prediction.list[[1]]$bResult, yes=0, no=1
svm.test.err = c(svm.test.err, mean(ifelse(svm.pred.test == set3.prediction.list[[1]]$bResult, yes=0, no
```

```
svm.fit <- svm(data=set1.prediction.list[[2]], bResult ~ ., kernel="radial", gamma=0.01, cost=0.1, cros
svm.pred.train <- predict(svm.fit, newdata=set1.prediction.list[[2]])
svm.pred.val <- predict(svm.fit, newdata=set2.prediction.list[[2]])
svm.pred.test <- predict(svm.fit, newdata=set3.prediction.list[[2]])
svm.train.err = c(svm.train.err, mean(ifelse(svm.pred.train == set1.prediction.list[[2]]$bResult, yes=0
svm.val.err = c(svm.val.err, mean(ifelse(svm.pred.val == set2.prediction.list[[2]]$bResult, yes=0, no=1
svm.test.err = c(svm.test.err, mean(ifelse(svm.pred.test == set3.prediction.list[[2]]$bResult, yes=0, no
```

```
svm.fit <- svm(data=set1.prediction.list[[3]], bResult ~ ., kernel="radial", gamma=0.01, cost=0.1, cros
svm.pred.train <- predict(svm.fit, newdata=set1.prediction.list[[3]])
svm.pred.val <- predict(svm.fit, newdata=set2.prediction.list[[3]])
svm.pred.test <- predict(svm.fit, newdata=set3.prediction.list[[3]])
svm.train.err = c(svm.train.err, mean(ifelse(svm.pred.train == set1.prediction.list[[3]]$bResult, yes=0
svm.val.err = c(svm.val.err, mean(ifelse(svm.pred.val == set2.prediction.list[[3]]$bResult, yes=0, no=1
svm.test.err = c(svm.test.err, mean(ifelse(svm.pred.test == set3.prediction.list[[3]]$bResult, yes=0, no
```

```
svm.train.err
```

```
## [1] 0.3207349 0.2236220 0.1312336
```

```
svm.val.err
```

```
## [1] 0.3049869 0.2246719 0.1364829
```

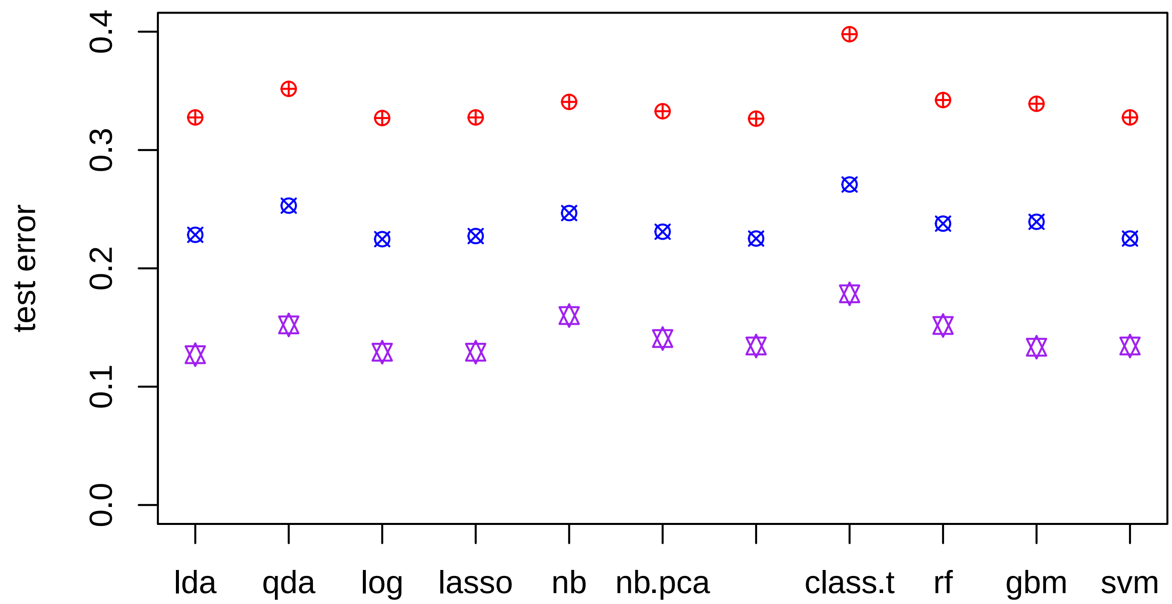
```
svm.test.err
```

```
## [1] 0.3275591 0.2251969 0.1343832
```

2.11 Model selection

```
models = c("lda", "qda", "log", "lasso", "nb", "nb.pca", "gam", "class.t", "rf", "gbm", "svm")
train.err = list(lda.train.err, qda.train.err, logit.train.err, lasso.logit.1se.train.err, naive.k.trai

val.err = list(lda.val.err, qda.val.err, logit.val.err, lasso.logit.1se.val.err, naive.k.val.err, naive

test.err = list(lda.test.err, qda.test.err, logit.test.err, lasso.logit.1se.test.err, naive.k.test.err,
```

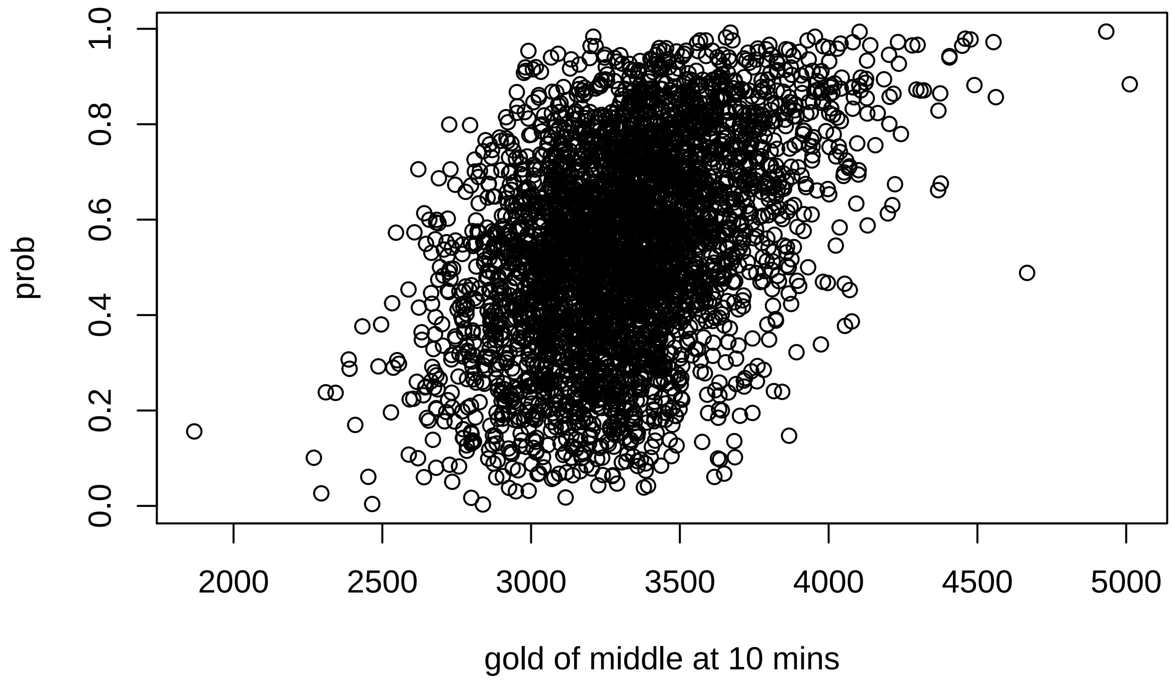
```
for (i in 1:3) {
  curErr = c()
  for (err in test.err) {
    curErr = c(curErr, err[i])
  }
  if (i == 1) {
    plot(curErr, xaxt = "n", ylim=c(0,0.4), col="red", pch = 10, ylab = "test error", xlab ="")
    axis(1, at=1:11, labels=models)
  } else if (i ==2) {
    points(curErr, col="blue", pch = 13)
  } else {
    points(curErr, col="purple", pch = 11)
  }


}
```



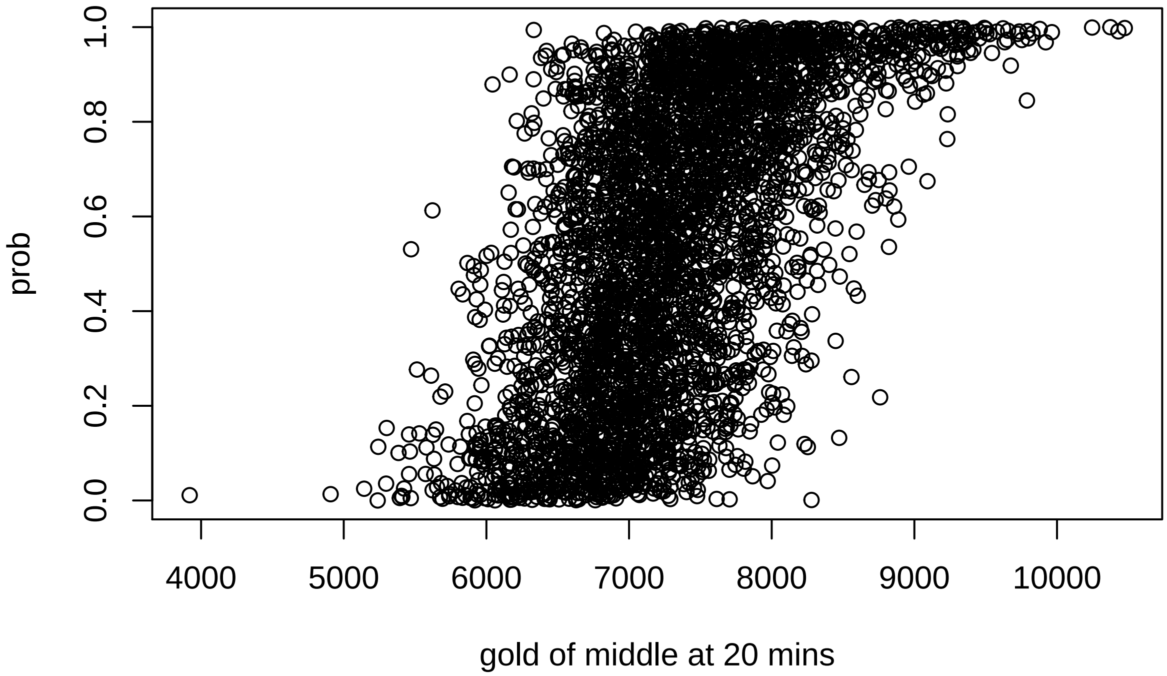
Choose logistic

3.1

```
logit.train.prob = logit.train.prob.list[[1]]
set1.prediction = set1.prediction.list[[1]]
plot(x = set1.prediction$goldblueMiddle10[order(logit.train.prob)], y = logit.train.prob[order(logit.tra
```




```
logit.train.prob = logit.train.prob.list[[2]]
set1.prediction = set1.prediction.list[[2]]
plot(x = set1.prediction$goldblueMiddle20[order(logit.train.prob)], y = logit.train.prob[order(logit.tra
```




```
logit.train.prob = logit.train.prob.list[[3]]
set1.prediction = set1.prediction.list[[3]]
plot(x = set1.prediction$goldblueMiddle30[order(logit.train.prob)], y = logit.train.prob[order(logit.tra
```