rcodemidterm.R

Yash

Thu Nov 29 04:03:46 2018

```
#GROUP MIDTERM ASSIGNMENT : DETECTING FRAUDULENT TRANSACTIONS


#The data we will be using in this case study refers to the transactions
#reported by the salespeople of some company. These salespeople sell a set of
#products of the company and report these sales with a certain periodicity.
#load from the file a data frame named sales.
#load("sales.Rdata")

#installs the package "DMwR" from the library
library(DMwR)
```

```
## Loading required package: lattice
```

```
## Loading required package: grid
```

```
#loads the data sales
data(sales)

#displays the first six rows of the data of sales
head(sales)
```

```
##   ID Prod Quant   Val Insp
## 1 v1   p1   182  1665 unkn
## 2 v2   p1  3072  8780 unkn
## 3 v3   p1 20393 76990 unkn
## 4 v4   p1   112  1100 unkn
## 5 v3   p1  6164 20260 unkn
## 6 v5   p2   104  1155 unkn
```

```
#to find the initial overview of the statistical properties of the data
#displays the summary of the data
summary(sales)
```

```
##        ID              Prod           Quant                 Val
##  v431   : 10159   p1125  :  3923   Min.   :      100   Min.   :   1005
##  v54    :  6017   p3774  :  1824   1st Qu.:      107   1st Qu.:   1345
##  v426   :  3902   p1437  :  1720   Median :      168   Median :   2675
##  v1679  :  3016   p1917  :  1702   Mean   :     8442   Mean   :  14617
##  v1085  :  3001   p4089  :  1598   3rd Qu.:      738   3rd Qu.:   8680
##  v1183  :  2642   p2742  :  1519   Max.   :473883883   Max.   :4642955
##  (Other):372409   (Other):388860   NA's   :13842       NA's   :1182
##     Insp
##  ok   : 14462
##  unkn :385414
##  fraud:  1270
##
##
##
##
```

```
#to confirm significant number of products and salespeople
#gives the number of salesids
nlevels(sales$ID)
```

```
## [1] 6016
```

```
#gives the number of sales products
nlevels(sales$Prod)
```

```
## [1] 4548
```

```
#gives the number of transactions
length(which(is.na(sales$Quant) & is.na(sales$Val)))
```

```
## [1] 888
```

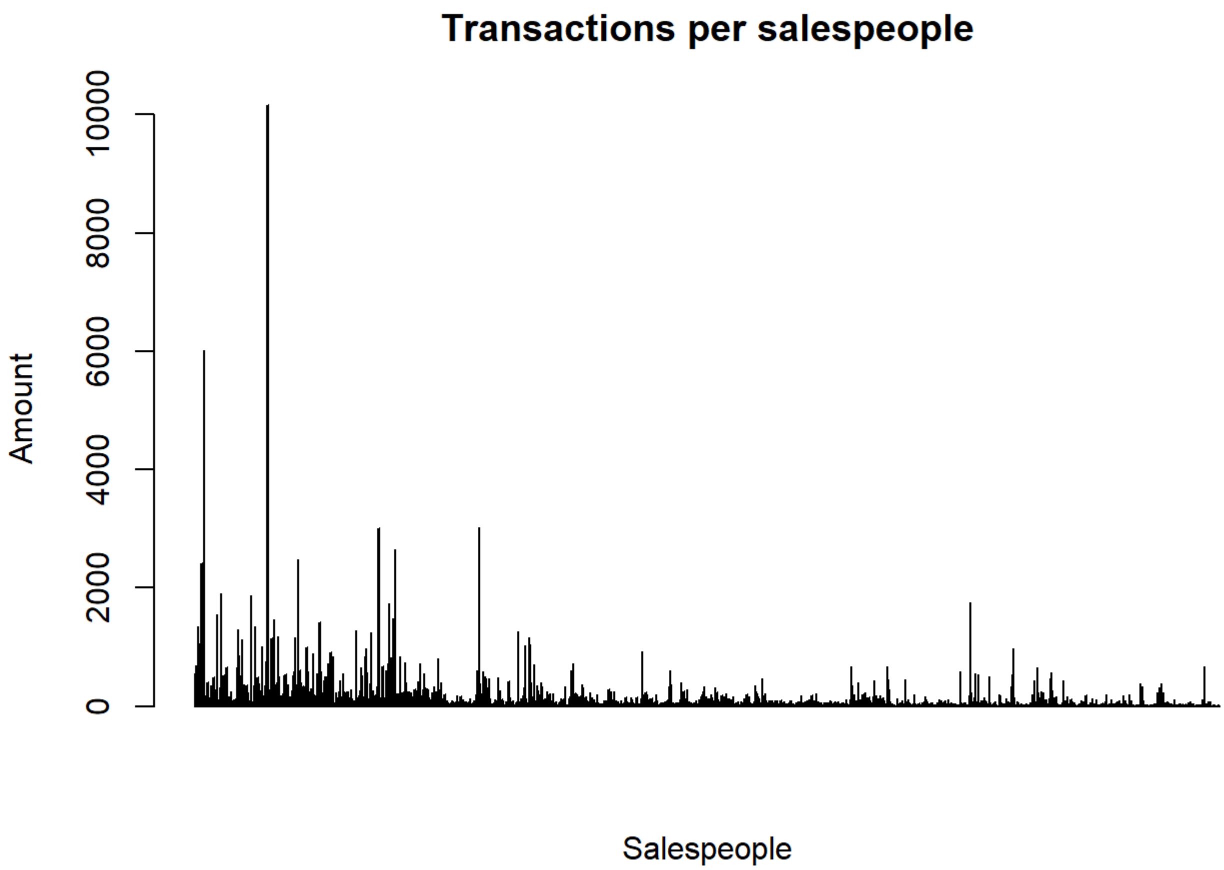
```
#taking advantage of the logical values in R, this is a more efficient way to obtain the number
 of transactions
sum(is.na(sales$Quant) & is.na(sales$Val))
```

```
## [1] 888
```

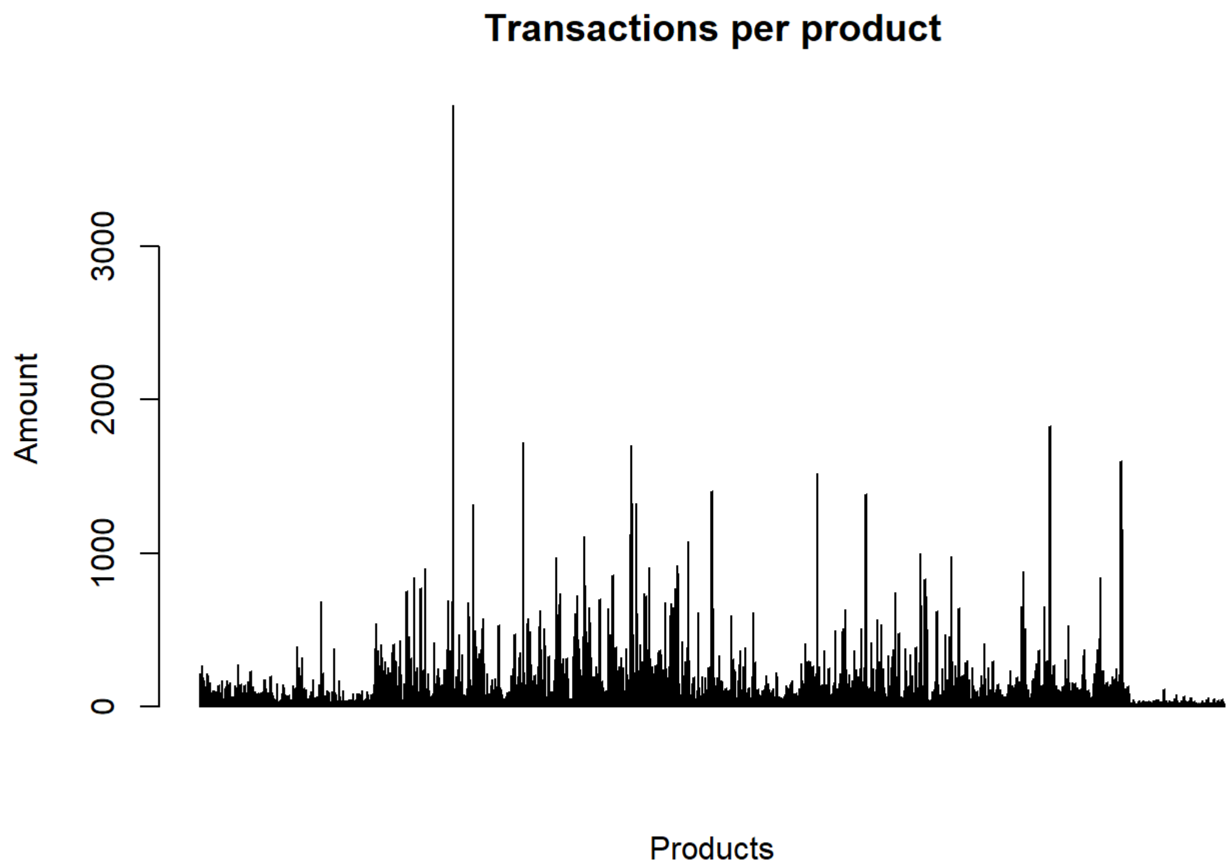
```
#gives a tabular view of the types of transactions
table(sales$Insp)/nrow(sales) * 100
```

```
##
##        ok      unkn     fraud
##  3.605171 96.078236  0.316593
```

```
#shows the number of reports per salesperson
#the numbers are rather diverse across the salespeople.
totS <- table(sales$ID)
totP <- table(sales$Prod)
barplot(totS, main = "Transactions per salespeople", names.arg = "",
        xlab = "Salespeople", ylab = "Amount")
```




```
#shows the number of reports per product
#we observe strong variability
barplot(totP, main = "Transactions per product", names.arg = "",
        xlab = "Products", ylab = "Amount")
```



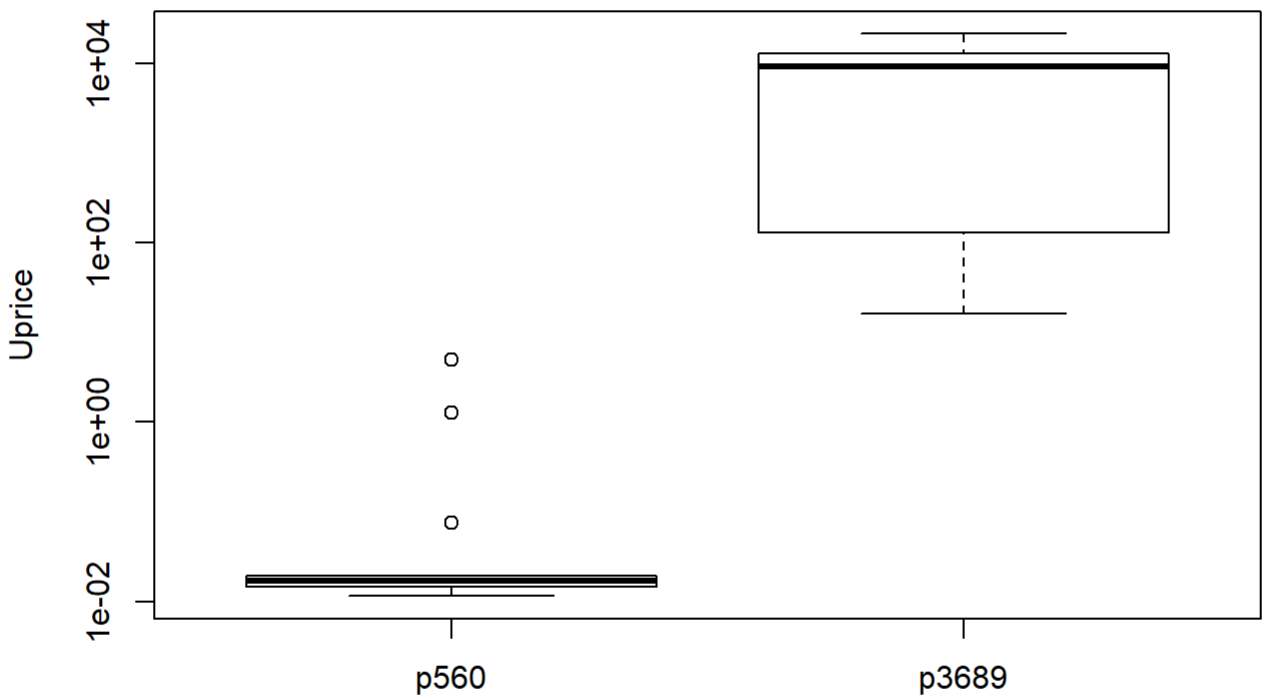

```
#adds new column to the data frame to carryout the analysis over unit price
sales$Uprice <- sales$Val/sales$Quant

#we observe marked variability
#summarizes the distribution of the unit price
summary(sales$Uprice)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.     NA's
##     0.00     8.46    11.89    20.30    19.11 26460.70    14136
```

```
#we have generated the five most expensive (cheapest) products by varying the parameter decreasi
ng of the function order(), using the sapply() function.
attach(sales)
upp <- aggregate(Uprice,list(Prod),median,na.rm=T)
topP <- sapply(c(T,F),function(o)
  upp[order(upp[,2],decreasing=o)[1:5],1])
colnames(topP) <- c('Expensive','Cheap')
topP
```

```
##      Expensive Cheap
## [1,] "p3689"   "p560"
## [2,] "p2453"   "p559"
## [3,] "p2452"   "p4195"
## [4,] "p2456"   "p601"
## [5,] "p2459"   "p563"
```

```
# we show the completely di???erent price distribution of the top products
tops <- sales[Prod %in% topP[1, ], c("Prod", "Uprice")]
tops$Prod <- factor(tops$Prod)
#displays the boxplot of their unit prices
boxplot(Uprice ~ Prod, data = tops, ylab = "Uprice", log = "y")
```




```
#  we carry out a similar analysis to discover which salespeople are the ones who bring more (le
ss) money to the company,
#displays the  distribution of the unit prices of the cheapest and most expensive products.
vs <- aggregate(Val,list(ID),sum,na.rm=T)
scoresSs <- sapply(c(T,F),function(o)
  vs[order(vs$x,decreasing=o)[1:5],1])
colnames(scoresSs) <- c('Most','Least')
scoresSs
```

```
##      Most    Least
## [1,] "v431"  "v3355"
## [2,] "v54"   "v6069"
## [3,] "v19"   "v5876"
## [4,] "v4520" "v6058"
## [5,] "v955"  "v4515"
```

```
sum(vs[order(vs$x, decreasing = T)[1:100], 2])/sum(Val, na.rm = T) *
  100
```

```
## [1] 38.33277
```

```
sum(vs[order(vs$x, decreasing = F)[1:2000], 2])/sum(Val,
                                                    na.rm = T) * 100
```

```
## [1] 1.988716
```

```
#analysis based on the quantity sold for each product
#results are more unbalanced
qs <- aggregate(Quant,list(Prod),sum,na.rm=T)
scoresPs <- sapply(c(T,F),function(o)
  qs[order(qs$x,decreasing=o)[1:5],1])
colnames(scoresPs) <- c('Most','Least')
scoresPs
```

```
##      Most    Least
## [1,] "p2516" "p2442"
## [2,] "p3599" "p2443"
## [3,] "p314"  "p1653"
## [4,] "p569"  "p4101"
## [5,] "p319"  "p3678"
```

```
#From the 4,548 products, 4,000 represent less than 10% of the sales volume,  with the top 100 r
epresenting nearly 75%.
sum(as.double(qs[order(qs$x,decreasing=T)[1:100],2]))/
  sum(as.double(Quant),na.rm=T)*100
```

```
## [1] 74.63478
```

```
#these may be more profitable if they have a larger profit margin.
sum(as.double(qs[order(qs$x,decreasing=F)[1:4000],2]))/
  sum(as.double(Quant),na.rm=T)*100
```

```
## [1] 8.944681
```

```
#determines the number of outliers of each product
out <- tapply(Uprice,list(Prod=Prod),
              function(x) length(boxplot.stats(x)$out))

#The boxplot.stats() function obtains several statistics that are used in the construction of bo
x plots. It returns a list with this information.
out[order(out, decreasing = T)[1:10]]
```

```
## Prod
## p1125 p1437 p2273 p1917 p1918 p4089  p538 p3774 p2742 p3338
##   376   181   165   156   156   137   129   125   120   117
```

```
#29,446 transactions are considered outliers, which corresponds to approximately 7% of the total
 number of transactions,
sum(out)
```

```
## [1] 29446
```

```
sum(out)/nrow(sales) * 100
```

```
## [1] 7.34047
```

```
#gives the  total number of transactions per salesperson and product
totS <- table(ID)
totP <- table(Prod)

#gives the The salespeople and products involved in the problematic transactions
nas <- sales[which(is.na(Quant) & is.na(Val)), c("ID", "Prod")]

#obtains the salespeople with a larger proportion of transactions
# with unknowns on both Val and Quant:
propS <- 100 * table(nas$ID)/totS
propS[order(propS, decreasing = T)[1:10]]
```

```
##
##     v1237     v4254     v4038     v5248     v3666     v4433     v4170
## 13.793103  9.523810  8.333333  8.333333  6.666667  6.250000  5.555556
##     v4926     v4664     v4642
##  5.555556  5.494505  4.761905
```

```
#the alternative of trying to fill in both columns seems much more risky.
#It seems reasonable to delete these transactions
propP <- 100 * table(nas$Prod)/totP
propP[order(propP, decreasing = T)[1:10]]
```

```
##
##    p2689    p2675    p4061    p2780    p4351    p2686    p2707    p2690
## 39.28571 35.41667 25.00000 22.72727 18.18182 16.66667 14.28571 14.08451
##    p2691    p2670
## 12.90323 12.76596
```

```r
#removing all transactions with unknown values on both the quantity and the value
#join their transactions with the ones from similar products to increase the statistical reliabi
lity of any outlier detection tests.
detach(sales)
sales <- sales[-which(is.na(sales$Quant) & is.na(sales$Val)),]


#calculating the proportion of transactions of each product that have the quantity unknown
nnasQp <- tapply(sales$Quant,list(sales$Prod),
                 function(x) sum(is.na(x)))
propNAsQp <- nnasQp/table(sales$Prod)
propNAsQp[order(propNAsQp,decreasing=T)[1:10]]
```

```
##     p2442     p2443     p1653     p4101     p4243      p903     p3678
## 1.0000000 1.0000000 0.9090909 0.8571429 0.6842105 0.6666667 0.6666667
##     p3955     p4464     p1261
## 0.6428571 0.6363636 0.6333333
```

```r
#These are 54 reports, and two of them are tagged as frauds while another was found to be OK.
#There are two products (p2442 and p2443) that have all their transactions with unknown values o
f the quantity.
sales <- sales[!sales$Prod %in% c("p2442", "p2443"), ]
# we will delete them
#updates the levels of the coulmn Prod
nlevels(sales$Prod)
```

```
## [1] 4548
```

```r
#we have just removed two products from our dataset,
sales$Prod <- factor(sales$Prod)
nlevels(sales$Prod)
```

```
## [1] 4546
```

```r
#there are several salespeople who have not filled in the information on the quantity in their r
eports
nnasQs <- tapply(sales$Quant, list(sales$ID), function(x) sum(is.na(x)))
propNAsQs <- nnasQs/table(sales$ID)
propNAsQs[order(propNAsQs, decreasing = T)[1:10]]
```

```
##     v2925     v5537     v5836     v6058     v6065     v4368     v2923
## 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 0.8888889 0.8750000
##     v2970     v4910     v4542
## 0.8571429 0.8333333 0.8095238
```

```
#we can try to use this information to fill in these unknowns using the assumption that the unit
 price should be similar.
# we will not delete these transactions.
nnasVp <- tapply(sales$Val,list(sales$Prod),
                 function(x) sum(is.na(x)))
propNAsVp <- nnasVp/table(sales$Prod)
propNAsVp[order(propNAsVp,decreasing=T)[1:10]]
```

```
##      p1110      p1022      p4491      p1462        p80      p4307
## 0.25000000 0.17647059 0.10000000 0.07500000 0.06250000 0.05882353
##      p4471      p2821      p1017      p4287
## 0.05882353 0.05389222 0.05263158 0.05263158
```

```
#similar analysis for the transactions with an unknown  value in the Val column.
nnasVs <- tapply(sales$Val, list(sales$ID), function(x) sum(is.na(x)))
propNAsVs <- nnasVs/table(sales$ID)
propNAsVs[order(propNAsVs, decreasing = T)[1:10]]
```

```
##      v5647        v74      v5946      v5290      v4472      v4022
## 0.37500000 0.22222222 0.20000000 0.15384615 0.12500000 0.09756098
##       v975      v2814      v2892      v3739
## 0.09574468 0.09090909 0.09090909 0.08333333
```

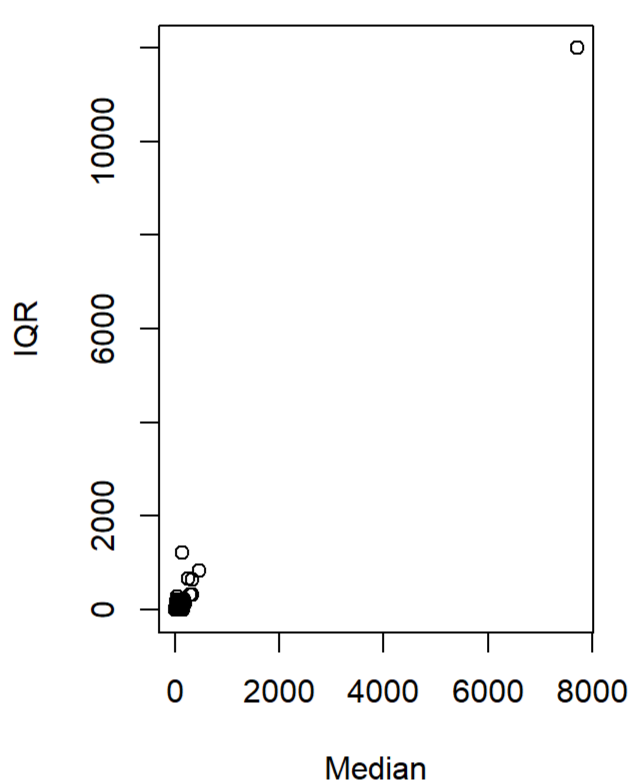
```
#. We will skip the prices of transactions that were found to be frauds in the calculation of th
e typical price. For the remaining transactions we will use the median unit price of the transac
tions as the typical
# price of the respective products
tPrice <- tapply(sales[sales$Insp != "fraud", "Uprice"],
                 list(sales[sales$Insp != "fraud", "Prod"]), median, na.rm = T)


# fills in all remaining unknown values
#we can use it to calculate any of the two possibly missing values (Quant and Val)
noQuant <- which(is.na(sales$Quant))
sales[noQuant,'Quant'] <- ceiling(sales[noQuant,'Val'] /
                                    tPrice[sales[noQuant,'Prod']])
noVal <- which(is.na(sales$Val))
sales[noVal,'Val'] <- sales[noVal,'Quant'] *
  tPrice[sales[noVal,'Prod']]


#we can recalculate the  Uprice column to fill in the previously unknown unit prices

sales$Uprice <- sales$Val/sales$Quant
#we have a dataset free of unknown values.

# saves the data frame
save(sales, file = "salesClean.Rdata")

# obtains both statistics for all transactions of each product
#uses the median as the statistic of centrality and the inter-quartile range (IQR) as the statis
tic of spread
attach(sales)
notF <- which(Insp != 'fraud')
ms <- tapply(Uprice[notF],list(Prod=Prod[notF]),function(x) {
  bp <- boxplot.stats(x)$stats
  c(median=bp[3],iqr=bp[4]-bp[2])
})
ms <- matrix(unlist(ms),
             length(ms),2,
             byrow=T,dimnames=list(names(ms),c('median','iqr')))
head(ms)
```

```
##       median      iqr
## p1 11.346154 8.575599
## p2 10.877863 5.609731
## p3 10.000000 4.809092
## p4  9.911243 5.998530
## p5 10.957447 7.136601
## p6 13.223684 6.685185
```

```
#uses the boxplot.stats() function to obtain the values of the median, first and third quartile
s.
#We calculate these values for all sets of transactions of each product, eliminating the fraudul
ent transactions from our analysis.
par(mfrow = c(1, 2))

#plots each product according to its respective median and IQR.
#there are many products that have approximately the same median and IQR
plot(ms[, 1], ms[, 2], xlab = "Median", ylab = "IQR", main = "")
plot(ms[, 1], ms[, 2], xlab = "Median", ylab = "IQR", main = "",
     col = "grey", log = "xy")
```

```
## Warning in xy.coords(x, y, xlabel, ylabel, log): 3 y values <= 0 omitted
## from logarithmic plot
```

```
smalls <- which(table(Prod) < 20)
points(log(ms[smalls, 1]), log(ms[smalls, 2]), pch = "+")
```



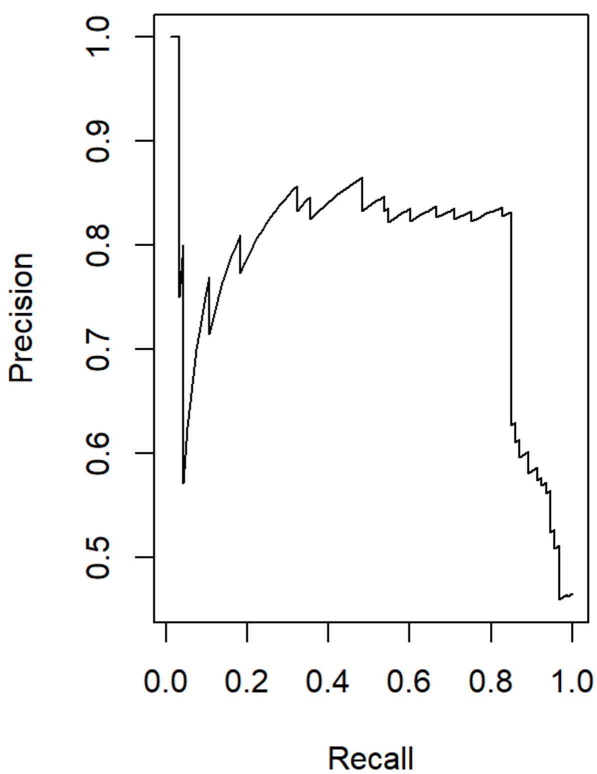
```
#obtains a matrix  with the information on this type of test for each of the products with less
 than 20 transactions
#similarity between the products is being calculated using the information on the median and IQR
 of the respective unit prices.
#dms <- scale(ms)
#smalls <- which(table(Prod) < 20)
#prods <- tapply(sales$Uprice, sales$Prod, list)
#similar <- matrix(NA, length(smalls), 7, dimnames = list(names(smalls),
#                                                         c("Simil", "ks.stat", "ks.p", "medP",
 "iqrP", "medS",
#                                                           "iqrS")))
#for (i in seq(along = smalls)) {
#  d <- scale(dms, dms[smalls[i], ], FALSE)
#  d <- sqrt(drop(d^2 %*% rep(1, ncol(d))))
 # stat <- ks.test(prods[[smalls[i]]], prods[[order(d)[2]]])
  #similar[i, ] <- c(order(d)[2], stat$statistic, stat$p.value,
   #                ms[smalls[i], ], ms[order(d)[2], ])
#}


#we show the first few lines of the resulting similar object
#The row names indicate the product for which we are obtaining the most  similar product
#head(similar)

#The respective product ID can be obtained for the first row of similar
#levels(Prod)[similar[1, 1]]

# checks how many products have a product whose unit price distribution  is significantly simila
r with 90% confidence:
#nrow(similar[similar[, "ks.p"] >= 0.9, ])

#more efficient method is to use "sum"
#sum(similar[, "ks.p"] >= 0.9)

# save the similar object in case we decide to use this similarity between products later
#save(similar, file = "similarProducts.Rdata")

#loads the package "ROCR"
#
library(ROCR)
```

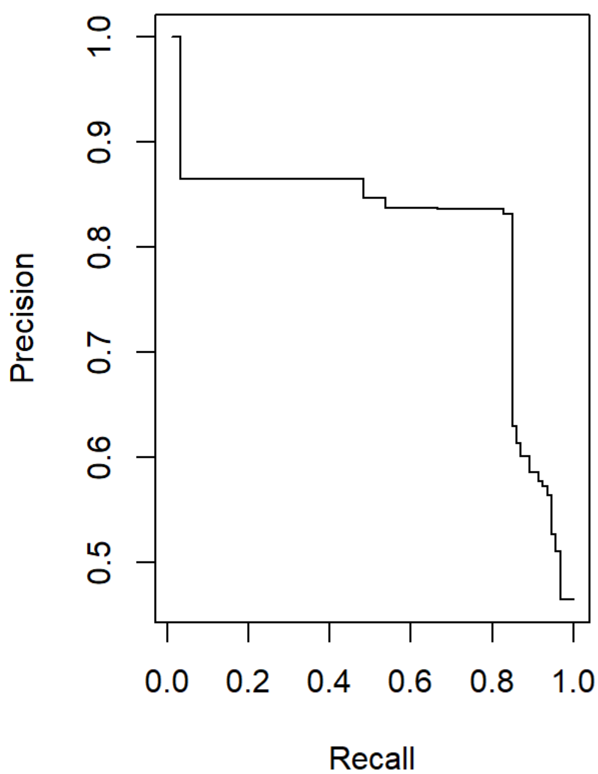
```
## Loading required package: gplots
```

```
##
## Attaching package: 'gplots'
```

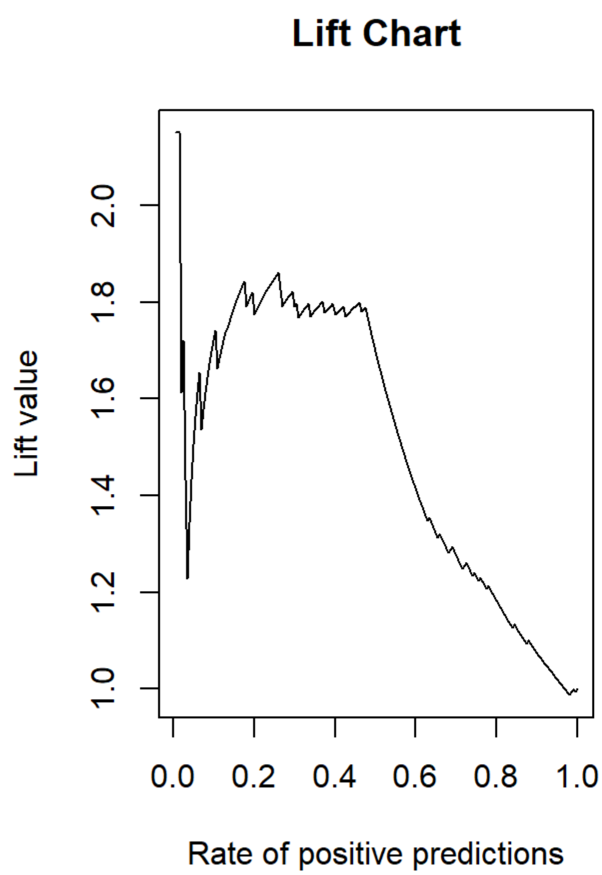
```
## The following object is masked from 'package:stats':
##
##     lowess
```

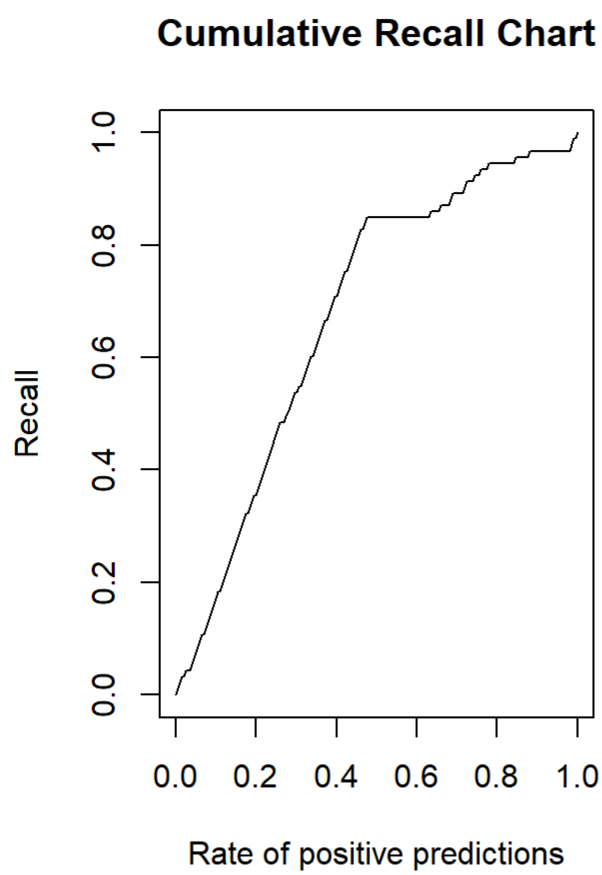
```
data(ROCR.simple)
#obtains an object of the class prediction using the predictions of the model and the true value
s of the test set.
pred <- prediction(ROCR.simple$predictions, ROCR.simple$labels)
perf <- performance(pred, "prec", "rec")

#the result of this latter function can be used with the function plot() to obtain di???erent pe
rformance curves.
plot(perf)

#it has a slot named y.values with the values of the y axis of the graph
#We can obtain a PR curve without the sawtooth e???ect by simply substituting this slot with the
 values of the interpolated precision

PRcurve <- function(preds, trues, ...) {
  require(ROCR, quietly = T)
  pd <- prediction(preds, trues)
  pf <- performance(pd, "prec", "rec")
  pf@y.values <- lapply(pf@y.values, function(x) rev(cummax(rev(x))))
  plot(pf, ...)
}

#use PRcurve() function to produce the graphs
PRcurve(ROCR.simple$predictions, ROCR.simple$labels)
```





```
#plots Lift charts using ROCR package
#not so useful for our application
pred <- prediction(ROCR.simple$predictions, ROCR.simple$labels)
perf <- performance(pred, "lift", "rpp")
plot(perf, main = "Lift Chart")

#plots cumulative recall chart using ROCR package
#shows the recall values in terms of the inspection e???ort that is captured by the RPP
CRchart <- function(preds, trues, ...) {
  require(ROCR, quietly = T)
  pd <- prediction(preds, trues)
  pf <- performance(pd, "rec", "rpp")
  plot(pf, ...)
}

# the nearer the curve of a model is to the topleft  corner of the graph, the better
CRchart(ROCR.simple$predictions, ROCR.simple$labels,
        main='Cumulative Recall Chart')
```





```
#we will use the average value of NDTPp as one of  the evaluation metrics to characterize the pe
rformance of the models.
# increase the computational eciency of repeated calls to this function.
avgNDTP <- function(toInsp,train,stats) {
  if (missing(train) && missing(stats))
    stop('Provide either the training data or the product stats')
  if (missing(stats)) {
    notF <- which(train$Insp != 'fraud')
    stats <- tapply(train$Uprice[notF],
                    list(Prod=train$Prod[notF]),
                    function(x) {
                      bp <- boxplot.stats(x)$stats
                      c(median=bp[3],iqr=bp[4]-bp[2])
                    })
    stats <- matrix(unlist(stats),
                    length(stats),2,byrow=T,
                    dimnames=list(names(stats),c('median','iqr')))
    stats[which(stats[,'iqr']==0),'iqr'] <-
      stats[which(stats[,'iqr']==0),'median']
  }

  mdtp <- mean(abs(toInsp$Uprice-stats[toInsp$Prod,'median']) /
                 stats[toInsp$Prod,'iqr'])
  return(mdtp)
}

#calculates precision, recall and average NDTP
evalOutlierRanking <- function(testSet,rankOrder,Threshold,statsProds) {
  ordTS <- testSet[rankOrder,]
  N <- nrow(testSet)
  nF <- if (Threshold < 1) as.integer(Threshold*N) else Threshold
  cm <- table(c(rep('fraud',nF),rep('ok',N-nF)),ordTS$Insp)
  prec <- cm['fraud','fraud']/sum(cm['fraud',])
  rec <- cm['fraud','fraud']/sum(cm[,'fraud'])
  AVGndtp <- avgNDTP(ordTS[nF,],stats=statsProds)
  return(c(Precision=prec,Recall=rec,avgNDTP=AVGndtp))
}
#This test set will be given to di???erent  modeling techniques that should return a ranking of
 these transactions according
#to their estimated probability of being frauds. Internally, the models
#may decide to analyze the products individually or all together.


#receives a set of transactions and obtains their ranking order and score
#we can mix together the values for the di???erent  products and thus produce a global ranking o
f all test cases
# it returns a list with this score and the rank order of the test set observations
BPrule <- function(train,test) {
  notF <- which(train$Insp != 'fraud')
  ms <- tapply(train$Uprice[notF],list(Prod=train$Prod[notF]),
               function(x) {
                 bp <- boxplot.stats(x)$stats
                 c(median=bp[3],iqr=bp[4]-bp[2])
```

```
                })
  ms <- matrix(unlist(ms),length(ms),2,byrow=T,
               dimnames=list(names(ms),c('median','iqr')))
  ms[which(ms[,'iqr']==0),'iqr'] <- ms[which(ms[,'iqr']==0),'median']
  ORscore <- abs(test$Uprice-ms[test$Prod,'median']) /
    ms[test$Prod,'iqr']
  return(list(rankOrder=order(ORscore,decreasing=T),
              rankScore=ORscore))
}


# calculating the values of the median and IQR for each product required to calculate the averag
e NDTP score.
#obtaining more reliable estimates of the ability of our models for detecting unusual values.
notF <- which(sales$Insp != 'fraud')
globalStats <- tapply(sales$Uprice[notF],
                      list(Prod=sales$Prod[notF]),
                      function(x) {
                        bp <- boxplot.stats(x)$stats
                        c(median=bp[3],iqr=bp[4]-bp[2])
                      })
globalStats <- matrix(unlist(globalStats),
                      length(globalStats),2,byrow=T,
                      dimnames=list(names(globalStats),c('median','iqr')))
globalStats[which(globalStats[,'iqr']==0),'iqr'] <-
  globalStats[which(globalStats[,'iqr']==0),'median']

#returns the value of the evaluation statistics with an attached attribute with the predicted an
d true values:
#To plot the PR and cumulative recall curves, the ROCR package functions need to know the predic
ted and true values of each test observation.
ho.BPrule <- function(form, train, test, ...) {
  res <- BPrule(train,test)
  structure(evalOutlierRanking(test,res$rankOrder,...),
            itInfo=list(preds=res$rankScore,
                        trues=ifelse(test$Insp=='fraud',1,0)
            )
  )
}

#The holdOut() function stores this information for each iteration of the experimental process.
# A more global perspective of the performance of the system over di???erent limits will be give
n by
#the PR and cumulative recall curves. The hold-out estimates will be obtained based on three rep
etitions of this process.
bp.res <- holdOut(learner('ho.BPrule',
                          pars=list(Threshold=0.1,
                                    statsProds=globalStats)),
                  dataset(Insp ~ .,sales),
                  hldSettings(3,0.3,1234,T),
                  itsInfo=TRUE
)
```

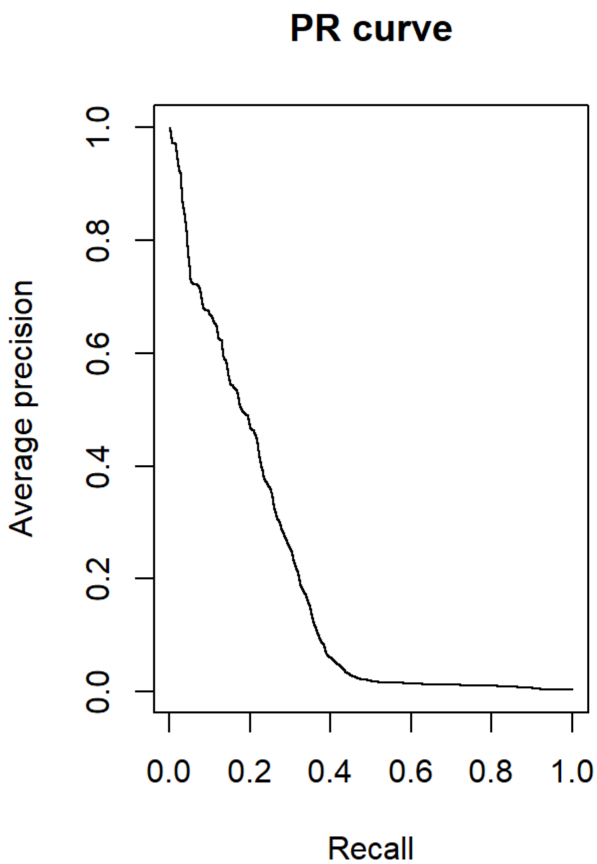
```
##
##  Stratified  3 x 70 %/ 30 % Holdout run with seed =  1234
## Repetition  1
## Repetition  2
## Repetition  3
```

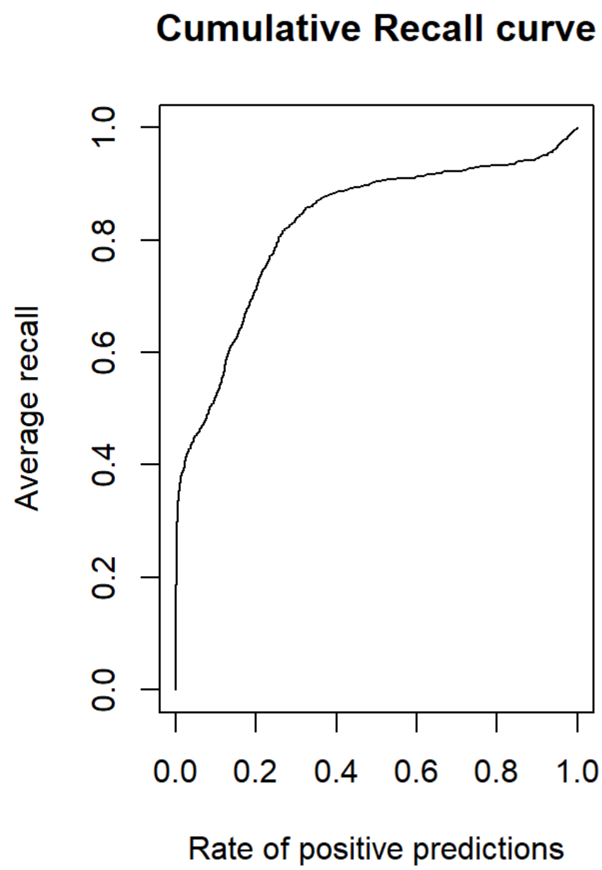
```
#summarizes the results of the Hold out experiment
summary(bp.res)
```

```
##
## == Summary of a Hold Out Experiment ==
##
##  Stratified  3 x 70 %/ 30 % Holdout run with seed =  1234
##
## * Data set ::  sales
## * Learner  ::  ho.BPrule  with parameters:
##   Threshold  =  0.1
##   statsProds  =  11.34  ...
##
## * Summary of Experiment Results:
```

```
##            Precision     Recall    avgNDTP
## avg     0.0166305736 0.52293272 1.87123901
## std     0.0008983669 0.01909992 0.05379945
## min     0.0159920040 0.51181102 1.80971393
## max     0.0176578377 0.54498715 1.90944329
## invalid 0.0000000000 0.00000000 0.00000000
```

```
#To obtain the PR and cumulative recall charts, we need access to the actual  outlier scores of
 the method on each hold-out repetition, as well as the true  "class" labels.
#The function holdOut() collects this extra information for each iteration on a list. This list
 is returned as an attribute named itsInfo of the objected produced by the holdOut() function
# divide the graph window in two to visualize both figures side by side.
par(mfrow=c(1,2))

# extract the list that contains the predicted and true values returned by the
#ho.BPrule() on each iteration
info <- attr(bp.res,'itsInfo')
PTs.bp <- aperm(array(unlist(info),dim=c(length(info[[1]]),2,3)),
                c(1,3,2)
)
PRcurve(PTs.bp[,,1],PTs.bp[,,2],
        main='PR curve',avg='vertical')
CRchart(PTs.bp[,,1],PTs.bp[,,2],
        main='Cumulative Recall curve',avg='vertical')
```


PR curve
Average precision
Recall
0.0
0.2
0.4
0.6
0.8
1.0


Cumulative Recall curve
Average recall
Rate of positive predictions
0.0
0.2
0.4
0.6
0.8
1.0

```
#eliminating both columns and treating the products separately seems clearly more reasonable tha
n the option of  re-coding the variables.
#apply the LOF algorithm to a dataset of reports described only by the unit price
#obtains the evaluation statistics resulting from applying the LOF method to the given training
 and test sets
ho.LOF <- function(form, train, test, k, ...) {
  ntr <- nrow(train)
  all <- rbind(train,test)
  N <- nrow(all)
  ups <- split(all$Uprice,all$Prod)
  r <- list(length=ups)
  for(u in seq(along=ups))
    r[[u]] <- if (NROW(ups[[u]]) > 3)
      lofactor(ups[[u]],min(k,NROW(ups[[u]]) %/% 2))
  else if (NROW(ups[[u]])) rep(0,NROW(ups[[u]]))
  else NULL
  all$lof <- vector(length=N)

  #The function split() was used to divide the unit prices of this full datase by product. The r
esult is a list whose components are the unit prices of the respective products
  split(all$lof,all$Prod) <- r
  all$lof[which(!(is.infinite(all$lof) | is.nan(all$lof)))] <-
    SoftMax(all$lof[which(!(is.infinite(all$lof) | is.nan(all$lof)))])
  structure(evalOutlierRanking(test,order(all[(ntr+1):N,'lof'],
                                          decreasing=T),...),
            itInfo=list(preds=all[(ntr+1):N,'lof'],
                        trues=ifelse(test$Insp=='fraud',1,0))
  )
}


#use a hold-out process to obtain the estimates of our evaluation metrics
lof.res <- holdOut(learner('ho.LOF',
                           pars=list(k=7,Threshold=0.1,
                                     statsProds=globalStats)),
                   dataset(Insp ~ .,sales),
                   hldSettings(3,0.3,1234,T),
                   itsInfo=TRUE
)
```

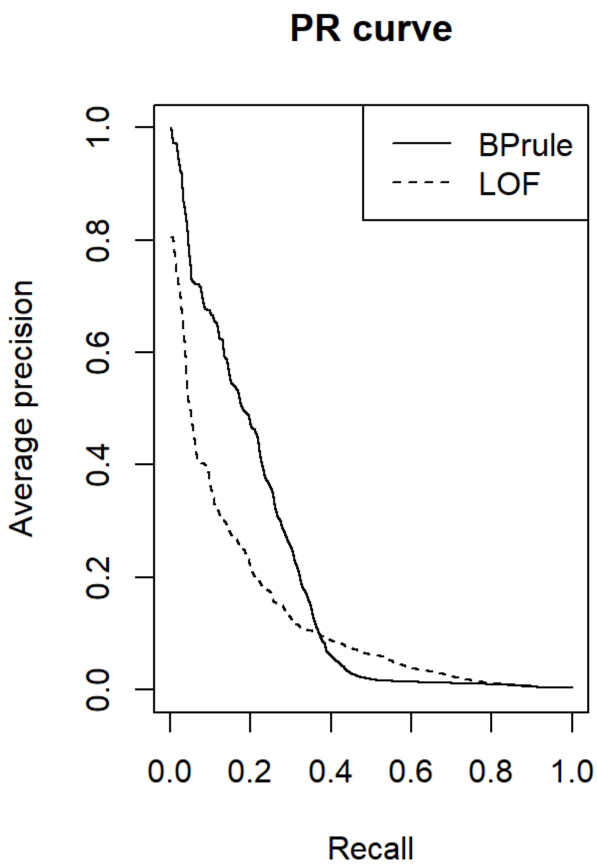
```
##
##  Stratified  3 x 70 %/ 30 % Holdout run with seed =  1234
## Repetition  1
## Repetition  2
## Repetition  3
```

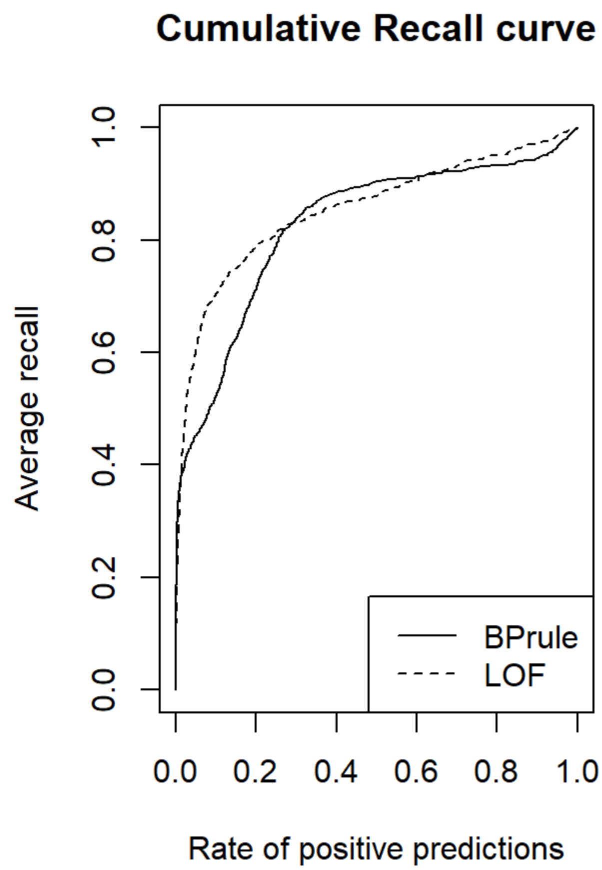
```
#The results of the LOF method are displayed
summary(lof.res)
```

```
##
## == Summary of a Hold Out Experiment ==
##
##  Stratified  3 x 70 %/ 30 % Holdout run with seed =  1234
##
## * Data set ::  sales
## * Learner  ::  ho.LOF  with parameters:
##   k  =  7
##   Threshold  =  0.1
##   statsProds  =  11.34  ...
##
## * Summary of Experiment Results:
```

```
##            Precision     Recall   avgNDTP
## avg     0.0221278250 0.69595344 2.4631856
## std     0.0009136811 0.02019331 0.9750265
## min     0.0214059637 0.67454068 1.4420851
## max     0.0231550891 0.71465296 3.3844572
## invalid 0.0000000000 0.00000000 0.0000000
```

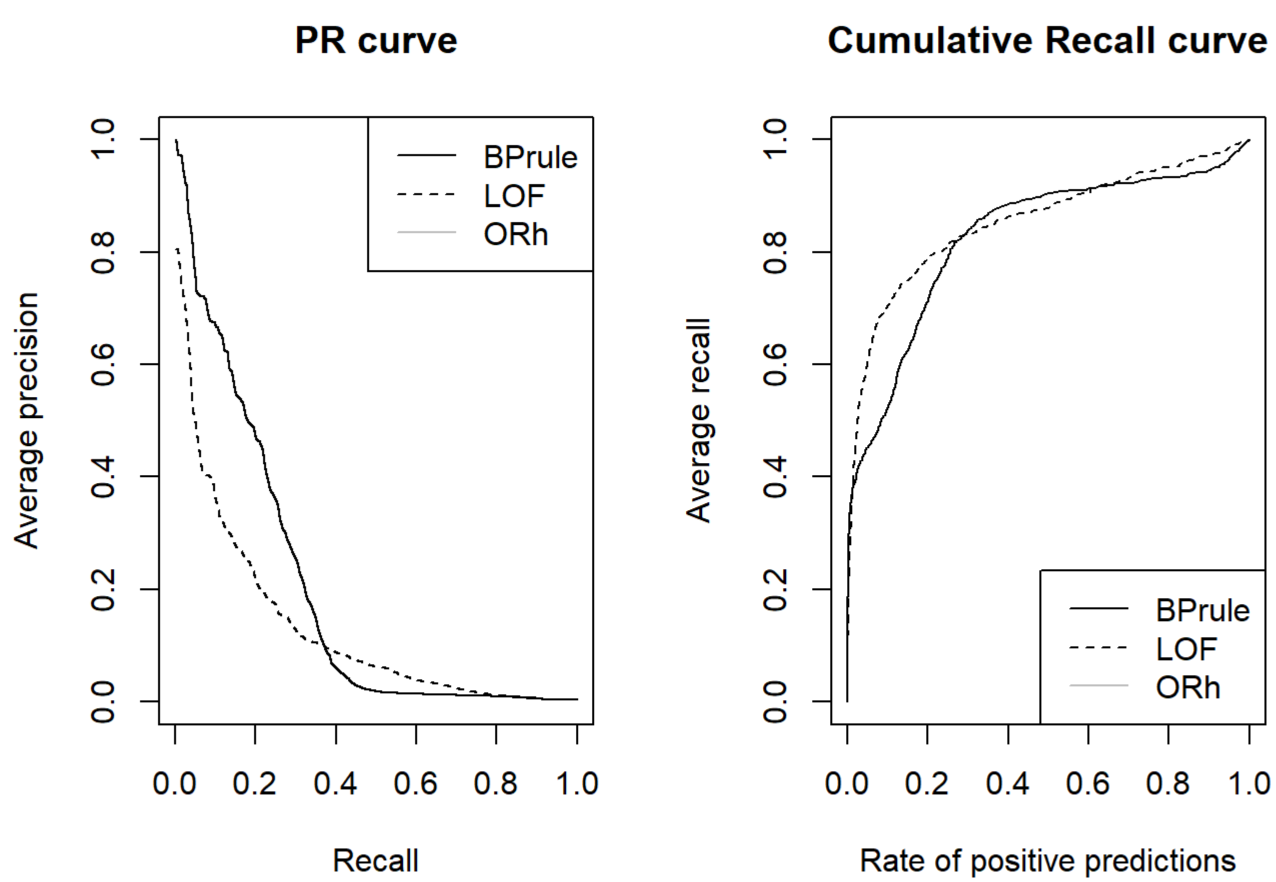
```
#plots PR and cumulative recall curves to enable better comparison with the BPrule method
par(mfrow=c(1,2))
info <- attr(lof.res,'itsInfo')
PTs.lof <- aperm(array(unlist(info),dim=c(length(info[[1]]),2,3)),
                 c(1,3,2)
)
PRcurve(PTs.bp[,,1],PTs.bp[,,2],
        main='PR curve',lty=1,xlim=c(0,1),ylim=c(0,1),
        avg='vertical')
PRcurve(PTs.lof[,,1],PTs.lof[,,2],
        add=T,lty=2,
        avg='vertical')
legend('topright',c('BPrule','LOF'),lty=c(1,2))
CRchart(PTs.bp[,,1],PTs.bp[,,2],
        main='Cumulative Recall curve',lty=1,xlim=c(0,1),ylim=c(0,1),
        avg='vertical')
CRchart(PTs.lof[,,1],PTs.lof[,,2],
        add=T,lty=2,
        avg='vertical')
legend('bottomright',c('BPrule','LOF'),lty=c(1,2))
```


PR curve
BPrule
LOF
Average precision
Recall
0.0
0.2
0.4
0.6
0.8
1.0


Cumulative Recall curve
BPrule
LOF
Average recall
Rate of positive predictions
0.0
0.2
0.4
0.6
0.8
1.0

```
#shows that for smaller recall values, the BP rule generally achieves a considerably higher prec
ision.

#obtain the outlier score of a test set of reports and obtains the usual evaluation statistics
#ho.ORh <- function(form, train, test, ...) {
 # ntr <- nrow(train)
#  all <- rbind(train,test)
 # N <- nrow(all)
#  ups <- split(all$Uprice,all$Prod)
#  r <- list(length=ups)
 # for(u in seq(along=ups))
  #  r[[u]] <- if (NROW(ups[[u]]) > 3)
  #    outliers.ranking(ups[[u]])$prob.outliers
#  else if (NROW(ups[[u]])) rep(0,NROW(ups[[u]]))
#  else NULL
#  all$orh <- vector(length=N)
#  split(all$orh,all$Prod) <- r
#  all$orh[which(!(is.infinite(all$orh) | is.nan(all$orh)))] <-
  #  SoftMax(all$orh[which(!(is.infinite(all$orh) | is.nan(all$orh)))])
  #structure(evalOutlierRanking(test,order(all[(ntr+1):N,'orh'],
  #                                        decreasing=T),...),
   #         itInfo=list(preds=all[(ntr+1):N,'orh'],
  #                      trues=ifelse(test$Insp=='fraud',1,0))
  #)
#}


#we can obtain this matrix using any distance function that handles mixed-mode data
#orh.res <- holdOut(learner('ho.ORh',
 #                          pars=list(Threshold=0.1,
  #                                   statsProds=globalStats)),
   #               dataset(Insp ~ .,sales),
    #              hldSettings(3,0.3,1234,T),
    #              itsInfo=TRUE
#)

#summarizes of the results of the ORh method
#summary(orh.res)


#the results of the ORh system in terms of both precision and recall are very similar to the val
ues of BP rule and LOF
#the average is lower than the scores of the other two methods
#par(mfrow=c(1,2))
#info <- attr(orh.res,'itsInfo')
#PTs.orh <- aperm(array(unlist(info),dim=c(length(info[[1]]),2,3)),
 #                c(1,3,2)
#)

#the results of the ORh method are comparable to those of LOF in terms of the cumulative recall
 curve

#regarding the PR curve, the ORh system clearly dominates the score of LOF, with a smaller advan
```

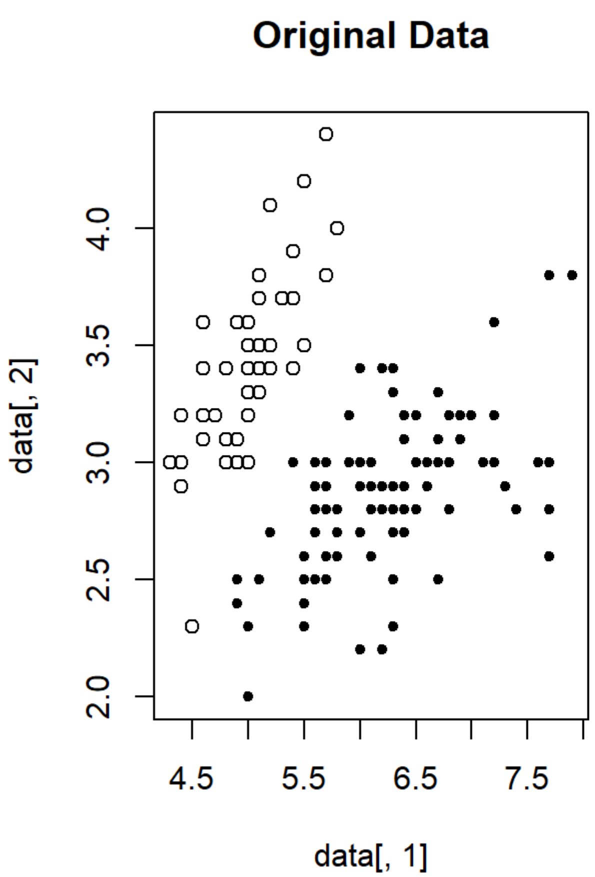
```
tage over BP rule.
PRcurve(PTs.bp[,,1],PTs.bp[,,2],
        main='PR curve',lty=1,xlim=c(0,1),ylim=c(0,1),
        avg='vertical')
PRcurve(PTs.lof[,,1],PTs.lof[,,2],
        add=T,lty=2,
        avg='vertical')
#PRcurve(PTs.orh[,,1],PTs.orh[,,2],
#        add=T,lty=1,col='grey',
#        avg='vertical')
legend('topright',c('BPrule','LOF','ORh'),
       lty=c(1,2,1),col=c('black','black','grey'))
CRchart(PTs.bp[,,1],PTs.bp[,,2],
        main='Cumulative Recall curve',lty=1,xlim=c(0,1),ylim=c(0,1),
        avg='vertical')
CRchart(PTs.lof[,,1],PTs.lof[,,2],
        add=T,lty=2,
        avg='vertical')
#CRchart(PTs.orh[,,1],PTs.orh[,,2],
 #       add=T,lty=1,col='grey',
  #      avg='vertical')
legend('bottomright',c('BPrule','LOF','ORh'),
       lty=c(1,2,1),col=c('black','black','grey'))
```

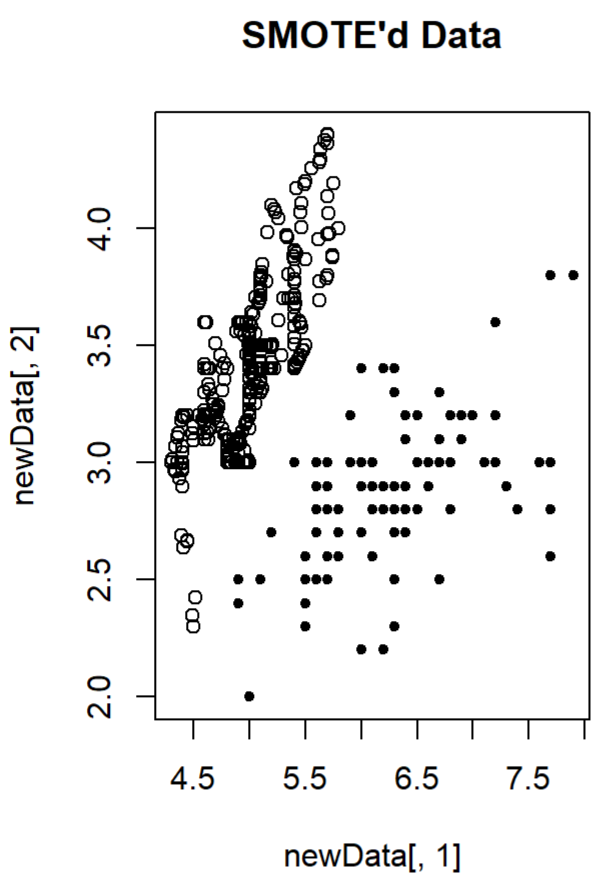


```
#uses the iris data to create an artificial dataset with two predictor variables and a new targe
t variable that has an unbalanced class distribution
data(iris)
data <- iris[, c(1, 2, 5)]
data$Species <- factor(ifelse(data$Species == "setosa", "rare",
                              "common"))
newData <- SMOTE(Species ~ ., data, perc.over = 600)
table(newData$Species)
```

```
##
## common   rare
##    600    350
```

```
#new cases are created by some form of random interpolation between the case and its nearest nei
ghbors

#by plotting the original and SMOTE'd datasets, we get a better idea of what was done
par(mfrow = c(1, 2))
plot(data[, 1], data[, 2], pch = 19 + as.integer(data[, 3]),
     main = "Original Data")
#we create more rare class examples
plot(newData[, 1], newData[, 2], pch = 19 + as.integer(newData[,
                                                                 3]), main = "SMOTE'd Data")
```





```
#obtain the ranking scores of a test set of reports.
#The outlier ranking is obtained using the estimated probabilities of the class being fraud
nb <- function(train, test) {
  require(e1071, quietly = T)
  sup <- which(train$Insp != "unkn")
  data <- train[sup, c("ID", "Prod", "Uprice", "Insp")]
  data$Insp <- factor(data$Insp, levels = c("ok", "fraud"))
  model <- naiveBayes(Insp ~ ., data)
  preds <- predict(model, test[, c("ID", "Prod", "Uprice",
                                   "Insp")], type = "raw")
  return(list(rankOrder = order(preds[, "fraud"], decreasing = T),
              rankScore = preds[, "fraud"]))
}

#obtains the selected evaluation statistics for the Naive Bayes predictions
ho.nb <- function(form, train, test, ...) {
  res <- nb(train,test)
  structure(evalOutlierRanking(test,res$rankOrder,...),
            itInfo=list(preds=res$rankScore,
                        trues=ifelse(test$Insp=='fraud',1,0)
            )
  )
}


#we call our holdOut() function to carry out the experiments with this model using the same sett
ings as for the unsupervised models
nb.res <- holdOut(learner('ho.nb',
                          pars=list(Threshold=0.1,
                                    statsProds=globalStats)),
                  dataset(Insp ~ .,sales),
                  hldSettings(3,0.3,1234,T),
                  itsInfo=TRUE
)
```

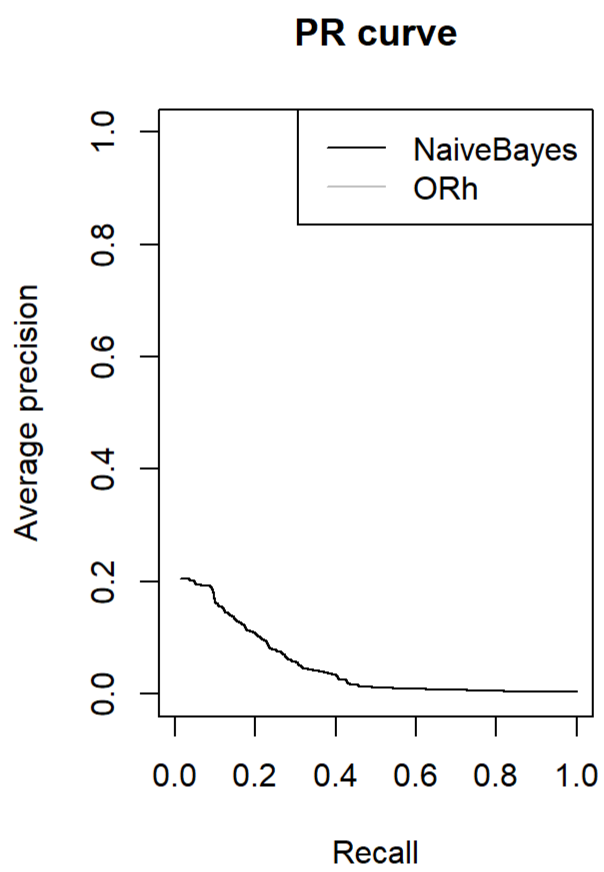
```
##
##  Stratified  3 x 70 %/ 30 % Holdout run with seed =  1234
## Repetition  1
## Repetition  2
## Repetition  3
```

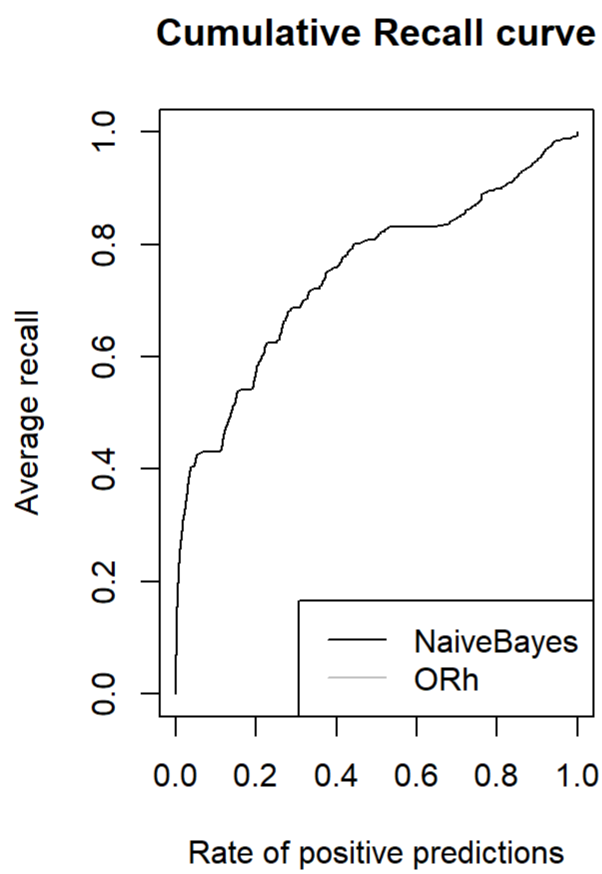
```
#displays the results for the naive bayes model for the 10% inspection effort
summary(nb.res)
```

```
##
## == Summary of a Hold Out Experiment ==
##
##  Stratified  3 x 70 %/ 30 % Holdout run with seed =  1234
##
## * Data set ::  sales
## * Learner  ::  ho.nb  with parameters:
##   Threshold  =  0.1
##   statsProds  =  11.34  ...
##
## * Summary of Experiment Results:
```

```
##           Precision     Recall   avgNDTP
## avg     0.013715365 0.43112103 0.8519657
## std     0.001083859 0.02613164 0.2406771
## min     0.012660336 0.40533333 0.5908980
## max     0.014825920 0.45758355 1.0650114
## invalid 0.000000000 0.00000000 0.0000000
```

```
#The scores are considerably worse than the best scores obtained previously with the unsupervise
d methods

#obtain the usual curves to get a better overall perspective of the performance of the model
#comparing naive bayes with ORh model
par(mfrow=c(1,2))
info <- attr(nb.res,'itsInfo')
PTs.nb <- aperm(array(unlist(info),dim=c(length(info[[1]]),2,3)),
                c(1,3,2)
)
PRcurve(PTs.nb[,,1],PTs.nb[,,2],
        main='PR curve',lty=1,xlim=c(0,1),ylim=c(0,1),
        avg='vertical')
#PRcurve(PTs.orh[,,1],PTs.orh[,,2],
 #       add=T,lty=1,col='grey',
  #      avg='vertical')
legend('topright',c('NaiveBayes','ORh'),
       lty=1,col=c('black','grey'))
CRchart(PTs.nb[,,1],PTs.nb[,,2],
        main='Cumulative Recall curve',lty=1,xlim=c(0,1),ylim=c(0,1),
        avg='vertical')
#CRchart(PTs.orh[,,1],PTs.orh[,,2],
 #       add=T,lty=1,col='grey',
  #      avg='vertical')
legend('bottomright',c('NaiveBayes','ORh'),
       lty=1,col=c('black','grey'))
```


PR curve
NaiveBayes
ORh
Average precision
0.0
0.2
0.4
0.6
0.8
1.0
Recall


Cumulative Recall curve
NaiveBayes
ORh
Average recall
0.0
0.2
0.4
0.6
0.8
1.0
Rate of positive predictions

```
# Naive Bayes method is inferior to the ORh method for this particular application
#Both curves indicate that the latter method dominates over all possible setups.
nb.s <- function(train, test) {
  require(e1071, quietly = T)
  sup <- which(train$Insp != "unkn")
  data <- train[sup, c("ID", "Prod", "Uprice", "Insp")]
  data$Insp <- factor(data$Insp, levels = c("ok", "fraud"))
  newData <- SMOTE(Insp ~ ., data, perc.over = 700)
  model <- naiveBayes(Insp ~ ., newData)
  preds <- predict(model, test[, c("ID", "Prod", "Uprice",
                                   "Insp")], type = "raw")
  return(list(rankOrder = order(preds[, "fraud"], decreasing = T),
              rankScore = preds[, "fraud"]))
}

#obtain the hold-out estimates for this SMOTE'd version of Naive Bayes:

ho.nbs <- function(form, train, test, ...) {
  res <- nb.s(train,test)
  structure(evalOutlierRanking(test,res$rankOrder,...),
            itInfo=list(preds=res$rankScore,
                        trues=ifelse(test$Insp=='fraud',1,0)
            )
  )
}
nbs.res <- holdOut(learner('ho.nbs',
                           pars=list(Threshold=0.1,
                                     statsProds=globalStats)),
                   dataset(Insp ~ .,sales),
                   hldSettings(3,0.3,1234,T),
                   itsInfo=TRUE
)
```

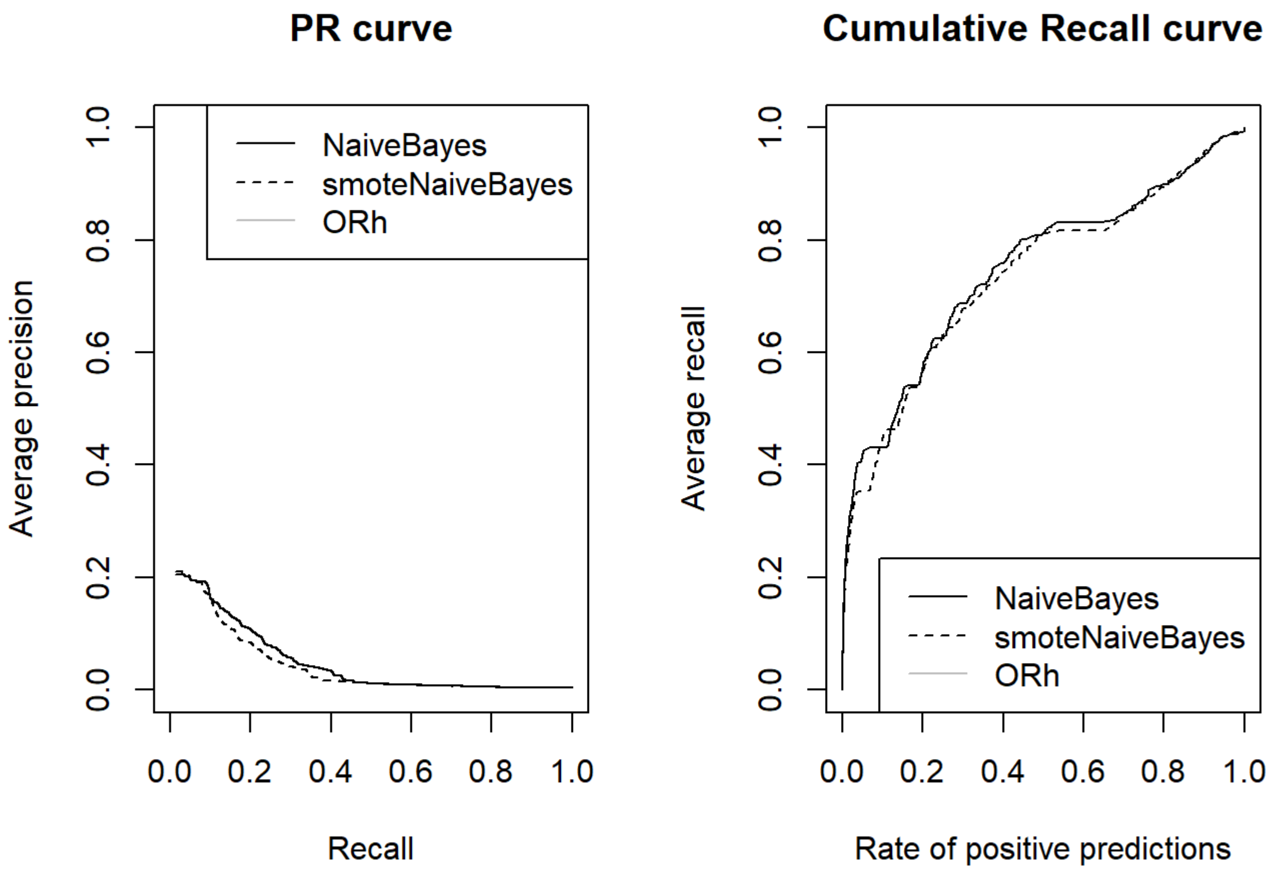
```
##
##  Stratified  3 x 70 %/ 30 % Holdout run with seed =  1234
## Repetition  1
## Repetition  2
## Repetition  3
```

```
#displays the results for naive bayes for the 10% inspection effort

summary(nbs.res)
```

```
##
## == Summary of a Hold Out Experiment ==
##
##  Stratified  3 x 70 %/ 30 % Holdout run with seed =  1234
##
## * Data set ::  sales
## * Learner  ::  ho.nbs  with parameters:
##   Threshold  =  0.1
##   statsProds  =  11.34  ...
##
## * Summary of Experiment Results:
```

```
##           Precision     Recall   avgNDTP
## avg     0.014215115 0.44686510 0.8913330
## std     0.001109167 0.02710388 0.8482740
## min     0.013493253 0.43044619 0.1934613
## max     0.015492254 0.47814910 1.8354999
## invalid 0.000000000 0.00000000 0.0000000
```

```
#The scores are only slightly superior but still very far from the best results of the unsupervi
sed models
#despite the oversampling of the minority class carried out by SMOTE, Naive Bayes is still not a
ble to correctly predict which are the fraudulent reports
#checks the  graphs for a more global perspective of the performance of this variant
par(mfrow=c(1,2))
info <- attr(nbs.res,'itsInfo')
PTs.nbs <- aperm(array(unlist(info),dim=c(length(info[[1]]),2,3)),
                 c(1,3,2)
)
PRcurve(PTs.nb[,,1],PTs.nb[,,2],
        main='PR curve',lty=1,xlim=c(0,1),ylim=c(0,1),
        avg='vertical')
PRcurve(PTs.nbs[,,1],PTs.nbs[,,2],
        add=T,lty=2,
        avg='vertical')
#PRcurve(PTs.orh[,,1],PTs.orh[,,2],
 #       add=T,lty=1,col='grey',
  #      avg='vertical')
legend('topright',c('NaiveBayes','smoteNaiveBayes','ORh'),
       lty=c(1,2,1),col=c('black','black','grey'))
CRchart(PTs.nb[,,1],PTs.nb[,,2],
        main='Cumulative Recall curve',lty=1,xlim=c(0,1),ylim=c(0,1),
        avg='vertical')
CRchart(PTs.nbs[,,1],PTs.nbs[,,2],
        add=T,lty=2,
        avg='vertical')
#CRchart(PTs.orh[,,1],PTs.orh[,,2],
 #       add=T,lty=1,col='grey',
  #      avg='vertical')
legend('bottomright',c('NaiveBayes','smoteNaiveBayes','ORh'),
       lty=c(1,2,1),col=c('black','black','grey'))
```




```
#loads the package "RWeka" from the library

library(RWeka)
#The function WOW() allows you to check which parameters are available for a particular Weka lea
rning algorithm
WOW(AdaBoostM1)
```

```
## -P <num>
##         Percentage of weight mass to base training on.  (default
##         100, reduce to around 90 speed up)
##  Number of arguments: 1.
## -Q      Use resampling for boosting.
## -S <num>
##         Random number seed.  (default 1)
##  Number of arguments: 1.
## -I <num>
##         Number of iterations.  (current value 10)
##  Number of arguments: 1.
## -W <classifier name>
##         Full name of base classifier.  (default:
##         weka.classifiers.trees.DecisionStump)
##  Number of arguments: 1.
## -output-debug-info
##         If set, classifier is run in debug mode and may output
##         additional info to the console
## -do-not-check-capabilities
##         If set, classifier capabilities are not checked before
##         classifier is built (use with caution).
## -num-decimal-places
##         The number of decimal places for the output of numbers in
##         the model (default 2).
##  Number of arguments: 1.
## -batch-size
##         The desired batch size for batch prediction (default 100).
##  Number of arguments: 1.
##
## Options specific to classifier weka.classifiers.trees.DecisionStump:
##
## -output-debug-info
##         If set, classifier is run in debug mode and may output
##         additional info to the console
## -do-not-check-capabilities
##         If set, classifier capabilities are not checked before
##         classifier is built (use with caution).
## -num-decimal-places
##         The number of decimal places for the output of numbers in
##         the model (default 2).
##  Number of arguments: 1.
## -batch-size
##         The desired batch size for batch prediction (default 100).
##  Number of arguments: 1.
```

```
# applying AdaBoostM1() to the well-known iris data set, using 100 iterations instead  of the de
fault 10
#loads the data iris
data(iris)
idx <- sample(150,100)
model <- AdaBoostM1(Species ~ .,iris[idx,],
                    control=Weka_control(I=100))
preds <- predict(model,iris[-idx,])
#lists the levels or columns
head(preds)
```

```
## [1] setosa setosa setosa setosa setosa setosa
## Levels: setosa versicolor virginica
```

```
#lists the prediction table
table(preds,iris[-idx,'Species'])
```

```
##
## preds        setosa versicolor virginica
##   setosa         18          0         0
##   versicolor      0         17         1
##   virginica       0          1        13
```

```
#lists the values of the six rows
prob.preds <- predict(model,iris[-idx,],type='probability')
head(prob.preds)
```

```
##       setosa   versicolor    virginica
## 1  0.9999911 8.886596e-06 9.658103e-11
## 6  0.9999911 8.886596e-06 9.658103e-11
## 7  0.9999911 8.886596e-06 9.658103e-11
## 9  0.9999953 4.725223e-06 1.816388e-10
## 11 0.9999911 8.886596e-06 9.658103e-11
## 18 0.9999911 8.886596e-06 9.658103e-11
```

```
#obtain probabilistic classifications with this model.


#obtains the report rankings for the given train and test sets
ab <- function(train,test) {
  require(RWeka,quietly=T)
  sup <- which(train$Insp != 'unkn')
  data <- train[sup,c('ID','Prod','Uprice','Insp')]
  data$Insp <- factor(data$Insp,levels=c('ok','fraud'))
  model <- AdaBoostM1(Insp ~ .,data,
                      control=Weka_control(I=100))
  preds <- predict(model,test[,c('ID','Prod','Uprice','Insp')],
                   type='probability')
  return(list(rankOrder=order(preds[,'fraud'],decreasing=T),
              rankScore=preds[,'fraud'])
  )
}
#this function is used to run the hold out experiments
ho.ab <- function(form, train, test, ...) {
  res <- ab(train,test)
  structure(evalOutlierRanking(test,res$rankOrder,...),
            itInfo=list(preds=res$rankScore,
                        trues=ifelse(test$Insp=='fraud',1,0)
            )
  )
}


ab.res <- holdOut(learner('ho.ab',
                          pars=list(Threshold=0.1,
                                    statsProds=globalStats)),
                  dataset(Insp ~ .,sales),
                  hldSettings(3,0.3,1234,T),
                  itsInfo=TRUE
)
```

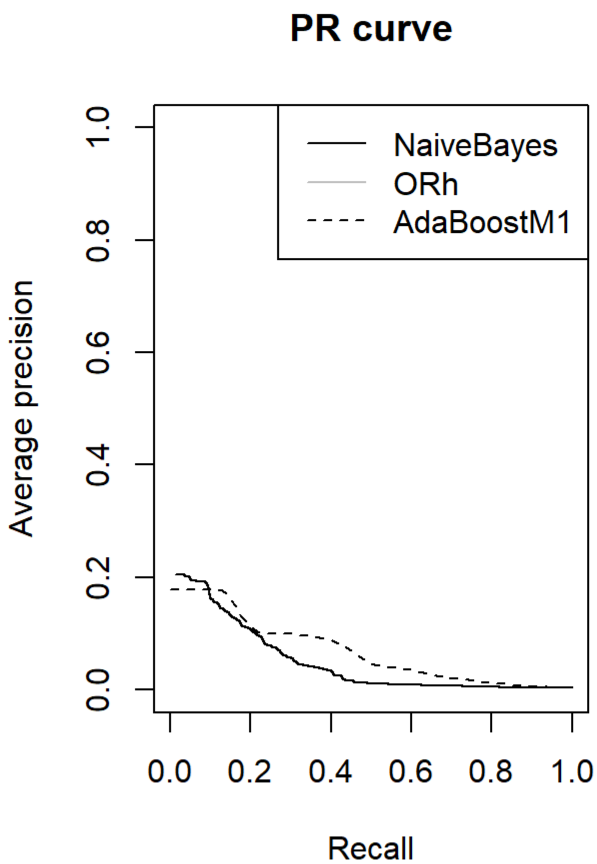
```
##
##  Stratified  3 x 70 %/ 30 % Holdout run with seed =  1234
## Repetition  1
## Repetition  2
## Repetition  3
```

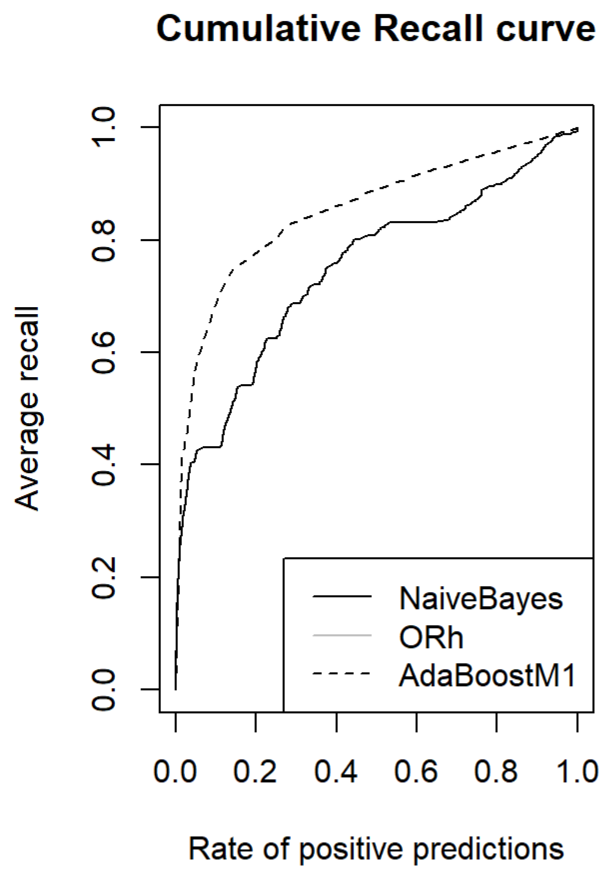
```
#displays the results of AdaBoost for the 10% e???ort
summary(ab.res)
```

```
##
## == Summary of a Hold Out Experiment ==
##
##  Stratified  3 x 70 %/ 30 % Holdout run with seed =  1234
##
## * Data set ::  sales
## * Learner  ::  ho.ab  with parameters:
##   Threshold  =  0.1
##   statsProds  =  11.34  ...
##
## * Summary of Experiment Results:
```

```
##            Precision     Recall   avgNDTP
## avg     0.0220722972 0.69416565 1.5182034
## std     0.0008695907 0.01576555 0.5238575
## min     0.0214892554 0.68241470 0.9285285
## max     0.0230717974 0.71208226 1.9298286
## invalid 0.0000000000 0.00000000 0.0000000
```

```
#these scores compare well with the best scores we have obtained with both LOF and ORh
#achieved a robust 69% of recall with a good 1.5 score in terms of average NDTP.


#PR and cumulative curves are obtained
par(mfrow=c(1,2))
info <- attr(ab.res,'itsInfo')
PTs.ab <- aperm(array(unlist(info),dim=c(length(info[[1]]),2,3)),
                c(1,3,2)
)
PRcurve(PTs.nb[,,1],PTs.nb[,,2],
        main='PR curve',lty=1,xlim=c(0,1),ylim=c(0,1),
        avg='vertical')
#PRcurve(PTs.orh[,,1],PTs.orh[,,2],
 #       add=T,lty=1,col='grey',
#        avg='vertical')
PRcurve(PTs.ab[,,1],PTs.ab[,,2],
        add=T,lty=2,
        avg='vertical')
legend('topright',c('NaiveBayes','ORh','AdaBoostM1'),
       lty=c(1,1,2),col=c('black','grey','black'))
CRchart(PTs.nb[,,1],PTs.nb[,,2],
        main='Cumulative Recall curve',lty=1,xlim=c(0,1),ylim=c(0,1),
        avg='vertical')
#CRchart(PTs.orh[,,1],PTs.orh[,,2],
 #       add=T,lty=1,col='grey',
#        avg='vertical')
CRchart(PTs.ab[,,1],PTs.ab[,,2],
        add=T,lty=2,
        avg='vertical')
legend('bottomright',c('NaiveBayes','ORh','AdaBoostM1'),
       lty=c(1,1,2),col=c('black','grey','black'))
```






```
#This curve shows that for most e???ort levels, AdaBoost.M1 matches the score obtained by ORh
#For higher recall levels, it clearly matches the precision of the best scores we have obtained
#we  can conclude that AdaBoost.M1 is a very competitive algorithm
library(DMwR)
library(e1071)


#Artificially creates a few unlabeled examples in this dataset to make semi-supervised classific
ation potentially useful
#loads the package "DMwR" and package "e1071"
#loads iris data
data(iris)
idx <- sample(150, 100)
tr <- iris[idx, ]
ts <- iris[-idx, ]
nb <- naiveBayes(Species ~ ., tr)
table(predict(nb, ts), ts$Species)
```

```
##
##              setosa versicolor virginica
##   setosa         14          0         0
##   versicolor      0         20         1
##   virginica       0          0        15
```

```
#obtains three di???erent Naive Bayes models

# first (nb) is obtained with a sample of 100 labeled cases.

#the dataset with the mixed labeled and unlabeled cases are given to the SelfTrain() function an
d another  model (nbST) obtained
#the self-trained model is able to almost reach the same level of performance as the
#initial model obtained with all 100 labeled cases.

trST <- tr
nas <- sample(100, 90)
trST[nas, "Species"] <- NA
func <- function(m, d) {
  p <- predict(m, d, type = "raw")
  data.frame(cl = colnames(p)[apply(p, 1, which.max)],
             p = apply(p, 1, max))
}
nbSTbase <- naiveBayes(Species ~ ., trST[-nas, ])
table(predict(nbSTbase, ts), ts$Species)
```

```
##
##              setosa versicolor virginica
##   setosa         14          0         0
##   versicolor      0          9         0
##   virginica       0         11        16
```

```
#nbST <- SelfTrain(Species ~ ., trST, learner("naiveBayes",
 #                                            list()), "func")
#table(predict(nbST, ts), ts$Species)

#implement and run the hold-out experiments with this self-trained Naive Bayes

pred.nb <- function(m,d) {
  p <- predict(m,d,type='raw')
  data.frame(cl=colnames(p)[apply(p,1,which.max)],
             p=apply(p,1,max)
  )
}
nb.st <- function(train,test) {
  require(e1071,quietly=T)
  train <- train[,c('ID','Prod','Uprice','Insp')]
  train[which(train$Insp == 'unkn'),'Insp'] <- NA
  train$Insp <- factor(train$Insp,levels=c('ok','fraud'))
  model <- SelfTrain(Insp ~ .,train,
                     learner('naiveBayes',list()),'pred.nb')
  preds <- predict(model,test[,c('ID','Prod','Uprice','Insp')],
                   type='raw')
  return(list(rankOrder=order(preds[,'fraud'],decreasing=T),
              rankScore=preds[,'fraud'])
  )
}
ho.nb.st <- function(form, train, test, ...) {
  res <- nb.st(train,test)
  structure(evalOutlierRanking(test,res$rankOrder,...),
            itInfo=list(preds=res$rankScore,
                        trues=ifelse(test$Insp=='fraud',1,0)
            )
  )
}


nb.st.res <- holdOut(learner('ho.nb.st',
                             pars=list(Threshold=0.1,
                                       statsProds=globalStats)),
                     dataset(Insp ~ .,sales),
                     hldSettings(3,0.3,1234,T),
                     itsInfo=TRUE



)
```

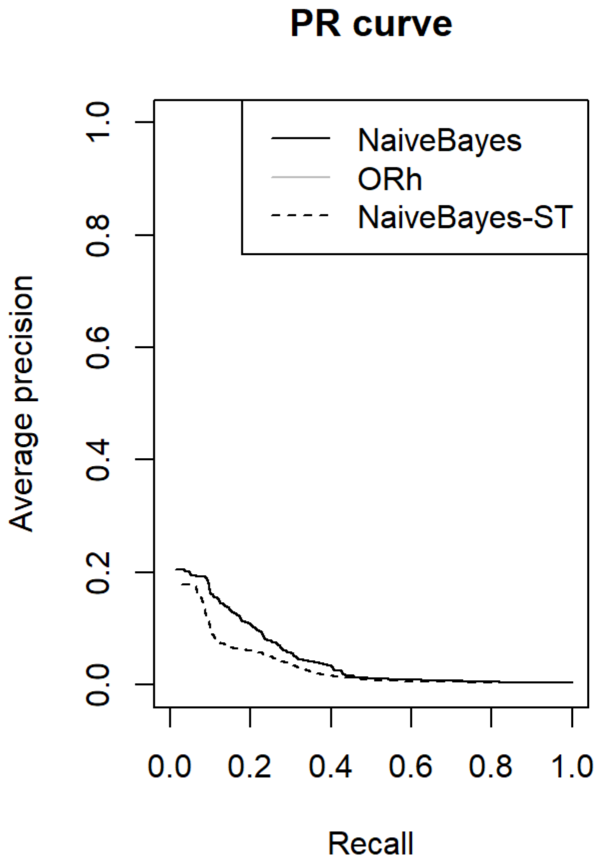
```
##
##  Stratified  3 x 70 %/ 30 % Holdout run with seed =  1234
## Repetition  1
## Repetition  2
## Repetition  3
```

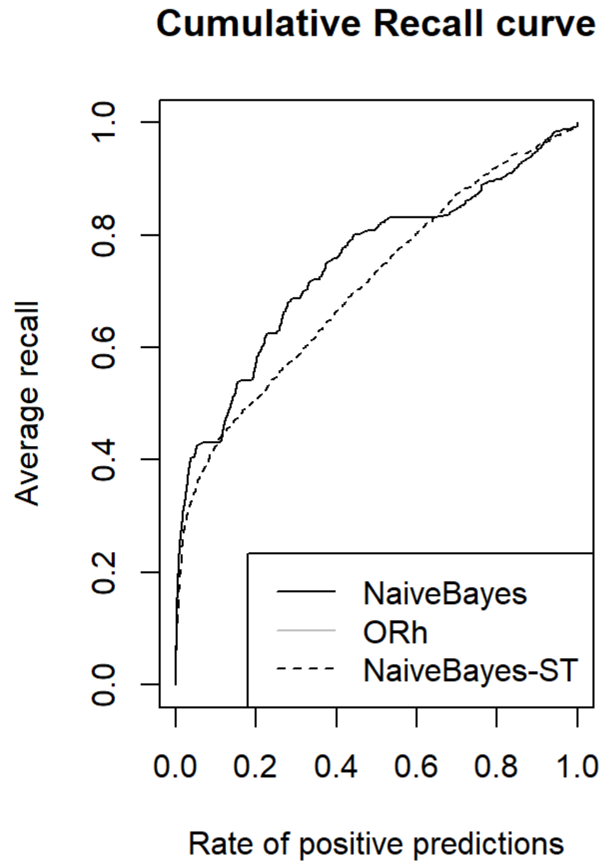
```
summary(nb.st.res)
```

```
##
## == Summary of a Hold Out Experiment ==
##
##  Stratified  3 x 70 %/ 30 % Holdout run with seed =  1234
##
## * Data set ::  sales
## * Learner  ::  ho.nb.st  with parameters:
##   Threshold  =  0.1
##   statsProds  =  11.34  ...
##
## * Summary of Experiment Results:
```

```
##           Precision     Recall    avgNDTP
## avg     0.013521017 0.42513271 1.08220611
## std     0.001346477 0.03895915 1.59726790
## min     0.012077295 0.38666667 0.06717087
## max     0.014742629 0.46456693 2.92334375
## invalid 0.000000000 0.00000000 0.00000000
```

```
#these results are disappointing

#obtains the  PR and cumulative recall curves of this model as well as those of the standard Nai
ve Bayes and ORh methods
par(mfrow=c(1,2))
info <- attr(nb.st.res,'itsInfo')
PTs.nb.st <- aperm(array(unlist(info),dim=c(length(info[[1]]),2,3)),
                   c(1,3,2)
)
PRcurve(PTs.nb[,,1],PTs.nb[,,2],
        main='PR curve',lty=1,xlim=c(0,1),ylim=c(0,1),
        avg='vertical')
#PRcurve(PTs.orh[,,1],PTs.orh[,,2],
#        add=T,lty=1,col='grey',
 #       avg='vertical')
PRcurve(PTs.nb.st[,,1],PTs.nb.st[,,2],
        add=T,lty=2,
        avg='vertical')
legend('topright',c('NaiveBayes','ORh','NaiveBayes-ST'),
       lty=c(1,1,2),col=c('black','grey','black'))
CRchart(PTs.nb[,,1],PTs.nb[,,2],
        main='Cumulative Recall curve',lty=1,xlim=c(0,1),ylim=c(0,1),
        avg='vertical')
#CRchart(PTs.orh[,,1],PTs.orh[,,2],
 #       add=T,lty=1,col='grey',
  #      avg='vertical')
CRchart(PTs.nb.st[,,1],PTs.nb.st[,,2],
        add=T,lty=2,
        avg='vertical')
legend('bottomright',c('NaiveBayes','ORh','NaiveBayes-ST'),
       lty=c(1,1,2),col=c('black','grey','black'))
```


PR curve
Average precision
Recall
0.0
0.2
0.4
0.6
0.8
1.0
NaiveBayes
ORh
NaiveBayes-ST


Cumulative Recall curve
Average recall
Rate of positive predictions
0.0
0.2
0.4
0.6
0.8
1.0
NaiveBayes
ORh
NaiveBayes-ST

```
#graphs show the disappointing performance of the self-trained Naive  Bayes classifier.
#this semi-supervised classifier is  clearly not competitive even with the standard Naive Bayes
 model obtained
#with a considerable smaller dataset.
#using the  self-training approach with the AdaBoost.M1 algorithm.

pred.ada <- function(m,d) {
  p <- predict(m,d,type='probability')
  data.frame(cl=colnames(p)[apply(p,1,which.max)],
             p=apply(p,1,max)
  )
}
ab.st <- function(train,test) {
  require(RWeka,quietly=T)
  train <- train[,c('ID','Prod','Uprice','Insp')]
  train[which(train$Insp == 'unkn'),'Insp'] <- NA
  train$Insp <- factor(train$Insp,levels=c('ok','fraud'))
  model <- SelfTrain(Insp ~ .,train,
                     learner('AdaBoostM1',
                             list(control=Weka_control(I=100))),
                     'pred.ada')
  preds <- predict(model,test[,c('ID','Prod','Uprice','Insp')],
                   type='probability')
  return(list(rankOrder=order(preds[,'fraud'],decreasing=T),
              rankScore=preds[,'fraud'])
  )
}
ho.ab.st <- function(form, train, test, ...) {
  res <- ab.st(train,test)
  structure(evalOutlierRanking(test,res$rankOrder,...),
            itInfo=list(preds=res$rankScore,
                        trues=ifelse(test$Insp=='fraud',1,0)
            )
  )
}
ab.st.res <- holdOut(learner('ho.ab.st',
                             pars=list(Threshold=0.1,
                                       statsProds=globalStats)),
                     dataset(Insp ~ .,sales),
                     hldSettings(3,0.3,1234,T),
                     itsInfo=TRUE
)
```

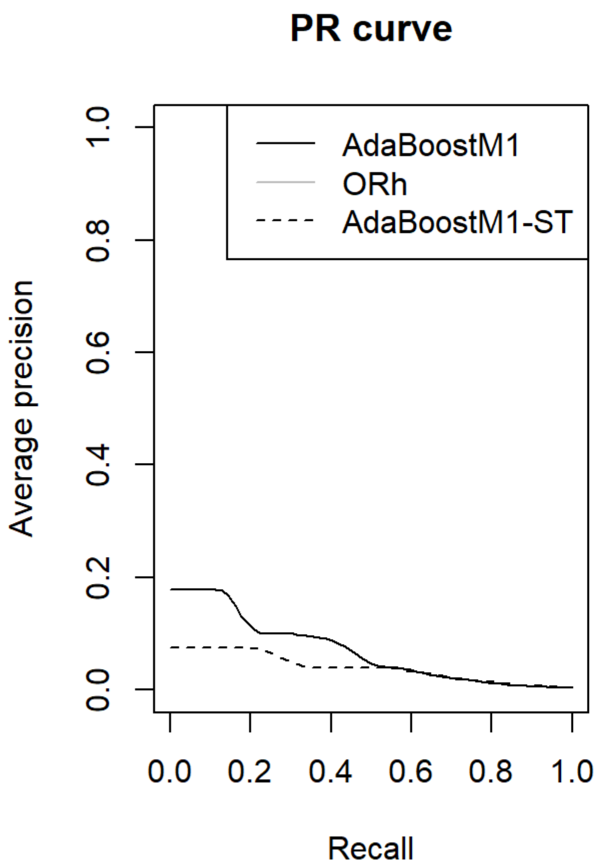
```
##
##  Stratified  3 x 70 %/ 30 % Holdout run with seed =  1234
## Repetition  1
## Repetition  2
## Repetition  3
```

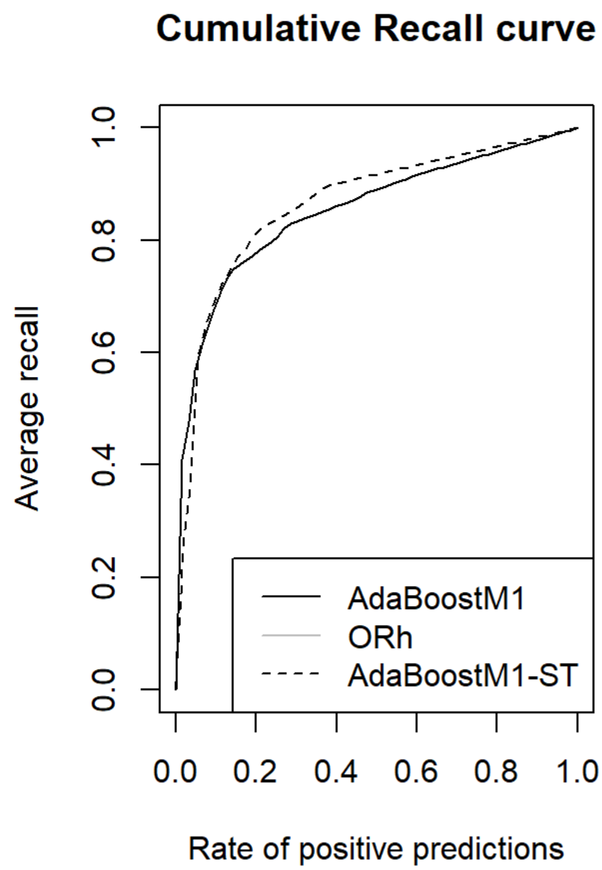
```
#displays the  results of the self-trained AdaBoost for the 10% e???ort
summary(ab.st.res)
```

```
##
## == Summary of a Hold Out Experiment ==
##
##  Stratified  3 x 70 %/ 30 % Holdout run with seed =  1234
##
## * Data set ::  sales
## * Learner  ::  ho.ab.st  with parameters:
##   Threshold  =  0.1
##   statsProds  =  11.34  ...
##
## * Summary of Experiment Results:
```

```
##           Precision     Recall   avgNDTP
## avg     0.022377700 0.70365350 1.6552619
## std     0.001130846 0.02255686 1.5556444
## min     0.021322672 0.68266667 0.5070082
## max     0.023571548 0.72750643 3.4257016
## invalid 0.000000000 0.00000000 0.0000000
```

```
#these scores represent a slight improvement over
#the AdaBoost.M1 model obtained using only the labeled data.
#displays the curves of this self-trained model, together with the standard AdaBoost.M1 and ORh
 methods.
par(mfrow = c(1, 2))
info <- attr(ab.st.res, "itsInfo")
PTs.ab.st <- aperm(array(unlist(info), dim = c(length(info[[1]]),
                                               2, 3)), c(1, 3, 2))
PRcurve(PTs.ab[, , 1], PTs.ab[, , 2], main = "PR curve",
        lty = 1, xlim = c(0, 1), ylim = c(0, 1), avg = "vertical")
#PRcurve(PTs.orh[, , 1], PTs.orh[, , 2], add = T, lty = 1,
 #       col = "grey", avg = "vertical")
PRcurve(PTs.ab.st[, , 1], PTs.ab.st[, , 2], add = T, lty = 2,
        avg = "vertical")
legend("topright", c("AdaBoostM1", "ORh", "AdaBoostM1-ST"),
       lty = c(1, 1, 2), col = c("black", "grey", "black"))
CRchart(PTs.ab[, , 1], PTs.ab[, , 2], main = "Cumulative Recall curve",
        lty = 1, xlim = c(0, 1), ylim = c(0, 1), avg = "vertical")
#CRchart(PTs.orh[, , 1], PTs.orh[, , 2], add = T, lty = 1,
#        col = "grey", avg = "vertical")
CRchart(PTs.ab.st[, , 1], PTs.ab.st[, , 2], add = T, lty = 2,
        avg = "vertical")
legend("bottomright", c("AdaBoostM1", "ORh", "AdaBoostM1-ST"),
       lty = c(1, 1, 2), col = c("black", "grey", "black"))
```






```
#The cumulative recall curve confirms that the self-trained AdaBoost.M1
# is the best model from the ones we have considered for this fraud detection
#problem

#In terms of precision, the scores are not that interesting, but as we mentioned
#before, this is not necessarily bad if the unlabeled reports that the model puts
#on higher positions in the ranking are confirmed as frauds.
```