Detecting Fraudulent Transactions.R

Tue Nov 27 06:07:55 2018

```
#loading libraries, loading and viewing data
library(DMwR)
```

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

```
#install.packages('Hmisc')
library(Hmisc)
```

```
## Loading required package: survival
## Loading required package: Formula
## Loading required package: ggplot2
##
## Attaching package: 'Hmisc'
## The following objects are masked from 'package:base':
##
##     format.pval, units
```

```
data(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
```

```
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
##
##
##
##
```

```
#nlevel returns the levels in the arguement
nlevels(sales$ID)
```

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

```
nlevels(sales$Prod)
```

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

```
describe(sales$ID)
```

```
## sales$ID
##        n  missing distinct
##   401146        0     6016
##
## lowest : v1    v2    v3    v4    v5   , highest: v6066 v6067 v6068 v6069 v
6070
```

```
describe(sales$prod)
```

```
##
## NULL
```

```
length(which(is.na(sales$Quant) & is.na(sales$Val)))
```

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

```
sum(is.na(sales$Quant) & is.na(sales$Val))
```

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

```
table(sales$Insp)/nrow(sales) * 100
```

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

```
totS <- table(sales$ID)
totP <- table(sales$Prod)
```

```r
#plotting no. of transactions per salesperson.

barplot(totS, main = "Transactions per salespeople", names.arg = "",xlab = "S
alespeople", ylab = "Amount")
```

```r
#plotting no. of transactions per product.

barplot(totP, main = "Transactions per product", names.arg = "",xlab = "Produ
cts", ylab = "Amount")
```

```r
#Creating a new column of our dataframe, to carry out the analysis because th
e quant and variability show variability.

sales$Uprice <- sales$Val/sales$Quant

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
```

```r
#Checking the most expensive and the most cheap product, and using median pri
ce to present the standard price at which the product is sold.

#Using aggregate function we obtain median unit price of each product.

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"
```

```r
#we have used a log scale in the graph to avoid the values of the cheapest pr
oduct becoming indistinguishable.

tops <- sales[Prod %in% topP[1, ], c("Prod", "Uprice")]

tops$Prod <- factor(tops$Prod)

boxplot(Uprice ~ Prod, data = tops, ylab = "Uprice", log = "y")
```

```r
#Obtaining analysis to discover which salespeople are the ones who bring more money to the company,

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"
```

```r
#We observe massive variations in the sales records of the sales employee, re lying on which actions need to be taken.


#Obtaining distribution of the unit prices of the cheapest and most expensive products.

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

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

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

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

```r
#Obtaining similar analysis in terms of the quantity that is sold for each pr oduct, the results are even 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"
```

```r
#We observe from the 4,548 products, 4,000 represent less than 10% of the sal es volume, with the top 100 representing 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
```

```
sum(as.double(qs[order(qs$x,decreasing=F)[1:4000],2]))/sum(as.double(Quant),n
a.rm=T)*100
```

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

```
#Here we try out find to the transaction of each product and the product with
more outliers.

out <- tapply(Uprice,list(Prod=Prod),function(x) length(boxplot.stats(x)$out)
)

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
```

```
#We observe 29446 are outliers.

sum(out)
```

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

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

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

```
#Obtaining the total number of transactions per salesperson and product.

totS <- table(ID)

totP <- table(Prod)


#Obtaining the salespeople with a larger proportion of transactions with unkn
owns on both Val and Quant.

nas <- sales[which(is.na(Quant) & is.na(Val)), c("ID", "Prod")]

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
```

```
#With respect to the products, these are the numbers.

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
```

```
#Removing all transactions with unknown values on both the quantity and the v
alue.
detach(sales)
sales <- sales[-which(is.na(sales$Quant) & is.na(sales$Val)),]


#Analyzing the remaining reports with unknown values in either the quantity o
r the value of the transaction.
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
```

```
#We have just removed two products from our dataset, we should update the lev
els of the column Prod.
sales <- sales[!sales$Prod %in% c("p2442", "p2443"), ]
nlevels(sales$Prod)
```

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

```
sales$Prod <- factor(sales$Prod)
nlevels(sales$Prod)
```

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

```
#we can try to use this information to fill in these unknowns using the assum
ption.
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 try to fill in these holes using the other transactions, with respect to
salesperson.

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
```

```
#The proportions are not too high. At this stage we have removed all reports
that had insufficient information.

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 use the median unit price of the transactions as the typical price o
f the respective products.

tPrice <- tapply(sales[sales$Insp != "fraud", "Uprice"],list(sales[sales$Insp
!= "fraud", "Prod"]), median, na.rm = T)


#we currently have no transactions with both values missing, Hence we will fi
ll all remaining unknown values.

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


#Saving the file after getting rid of the unknown values.

save(sales, file = "salesClean.Rdata")
```

```r
#We are obtaining both statistics for all transactions of each product.
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('media
n','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
```

```r
#Here second graph we have used black "+" signs to indicate the products that
have
#less than 20 transactions.
#Where parameter log=xy sets log scales on both axes of the graph.
par(mfrow = c(1, 2))
plot(ms[, 1], ms[, 2], xlab = "Median", ylab = "IQR", main = "")
par(mar=c(1,1,1,1))
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
```

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

```r
#Here we start by normalizing the data, then calculate the distance between d
istribution property.
```

```r
#We use ks.stats which we have extracted the value of the statistic of the te
st and the respective significance level


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("Sim
il", "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], ])}

#Displaying the first few lines of the results of similar object.

head(similar)
```

```
##      Simil   ks.stat       ks.p     medP      iqrP     medS      iqrS

## p8    2827 0.4339623 0.06470603 3.850211 0.7282168 3.868306 0.7938557

## p18    213 0.2568922 0.25815859 5.187266 8.0359968 5.274884 7.8894149

## p38   1044 0.3650794 0.11308315 5.490758 6.4162095 5.651818 6.3248073

## p39   1540 0.2258065 0.70914769 7.986486 1.6425959 8.080694 1.7668724

## p40   3971 0.3333333 0.13892028 9.674797 1.6104511 9.668854 1.6520147

## p47   1387 0.3125000 0.48540576 2.504092 2.5625835 2.413498 2.6402087
```

```r
#Product ID can be obtained as shown in the following example for the first r
ow of similar.

levels(Prod)[similar[1, 1]]
```

```
## [1] "p2829"
```

```r
nrow(similar[similar[, "ks.p"] >= 0.9, ])
```

```
## [1] 117
```

```r
sum(similar[, "ks.p"] >= 0.9)
```

```
## [1] 117
```

```r
#saving the similar object in case we decide to use this similarity between p
roducts later

save(similar, file = "similarProducts.Rdata")


#Loading required libraries to create graph.
```

```r
#The PR curves produced by the ROCR package have a sawtooth shape.
library(ROCR)
## Loading required package: gplots
##
## Attaching package: 'gplots'
## The following object is masked from 'package:stats':
##
##     lowess
data(ROCR.simple)
pred <- prediction(ROCR.simple$predictions, ROCR.simple$labels)
perf <- performance(pred, "prec", "rec")
plot(perf)


#lapply() because the slot y.values is in effect, a list as it can include th
e results of several iterations of an experimental process.
#Calculating the interpolated precision using the functions cummax() and rev(
).
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, ...)
}
#Generating graph using prcurve() function.
PRcurve(ROCR.simple$predictions, ROCR.simple$labels)
```

```r
#Lift charts can be obtained with the infrastructure provided by the ROCR pac
kage.
pred <- prediction(ROCR.simple$predictions, ROCR.simple$labels)
perf <- performance(pred, "lift", "rpp")
plot(perf, main = "Lift Chart")


#Recall values in terms of the inspection effort that is captured by the RPP.
```

```
#We will call this type of graph the cumulative recall chart and being implem
ented.

CRchart <- function(preds, trues, ...) {

  require(ROCR, quietly = T)

  pd <- prediction(preds, trues)

  pf <- performance(pd, "rec", "rpp")

  plot(pf, ...)

}

#obtaining more smoothed graph.

CRchart(ROCR.simple$predictions, ROCR.simple$labels,main='Cumulative Recall C
hart')
```

```
#Using NDTP as one of the evaluation metrics to characterize the performance
of the models.

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),'med
ian']

  }


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

  return(mdtp)
```

```
}


#This object allows you to specify that a stratified sampling is to be used.
#These statistics are precision, recall and the 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))
}



#We have to decide how to move from these sets into an outlier ranking of all test sets.
#therefore distinguishing the outliers ranking of all test set.
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))
}
```

```
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(globalSt
ats[,'iqr']==0),'median']


#The function structure() can be used to create an object and specify the val
ues of its attributes.
#As experimental settings we will use a 70%/30% division of the full dataset
using a stratified
#sampling strategy, and calculate the precision/recall statistics for a prede
fined
#inspection limit e???ort of 10% of the test set.


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)
            )
  )
}

#A more global perspective of the performance of the system over different li
mits will be given by
#the PR and cumulative recall curves.
```

```r
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
```

```r
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
```

```r
#This function can be used to obtain the value of any attribute of an object by its name.
#This list is then transformed into an array with three dimensions.
par(mfrow=c(1,2))
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'
)
```

```
#Here we try to approach was to merge the train and test datasets and use LOF
to rank this full set of reports.
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$lof <- vector(length=N)
  split(all$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))
  )
}

#Here we observed,  the values of precision and recall for this 10% inspectio
n
#effort are higher than the values obtained by the BP rule method.
```

```
lof.res <- holdOut(learner('ho.LOF', pars=list(k=7,Threshold=0.1,statsProds=g
lobalStats)),

                   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
```

```
#we can say that generally the LOF method dominates the BP rule for inspectio
n efforts below 25% to 30%.

#For higher inspection efforts, the differences are not so clear, and the res
ults are rather comparable.

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))
```

```
#Here again we have used the approach of handling the products

#individually, primarily for the same reasons described for the LOF method.

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))
  )
}
orh.res <- holdOut(learner('ho.ORh', pars=list(Threshold=0.1, statsProds=glob
alStats)),
                   dataset(Insp ~ .,sales),
                   hldSettings(3,0.3,1234,T),
                   itsInfo=TRUE
)
##
##  Stratified  3 x 70 %/ 30 % Holdout run with seed =  1234
## Repetition  1
## The "ward" method has been renamed to "ward.D"; note new "ward.D2"
#The results of the ORh system in terms of both precision and recall are very
similar to the values of BP rule and LOF.
summary(orh.res)
##
## == Summary of a Hold Out Experiment ==
##
##  Stratified  3 x 70 %/ 30 % Holdout run with seed =  1234
##
## * Data set ::  sales
## * Learner  ::  ho.ORh  with parameters:
##   Threshold  =  0.1
```

```
##   statsProds  =  11.34  ...
##
## * Summary of Experiment Results:
##              Precision     Recall   avgNDTP
## avg     0.0220445333 0.69345072 0.5444893
## std     0.0005545834 0.01187721 0.3712311
## min     0.0215725471 0.67979003 0.2893128
## max     0.0226553390 0.70133333 0.9703665
## invalid 0.0000000000 0.00000000 0.0000000
```

```
#As you can see, the results of the ORh method are comparable to those of LOF in terms of the cumulative recall curve.
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)
)
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'))
```

```
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"]))
}


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)))
}


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
##
## Attaching package: 'e1071'
## The following object is masked from 'package:Hmisc':
##
##     impute
##
## Repetition  2
## Repetition  3
```

```
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
```

```
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'))
```

```
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$Insp, levels = c("ok", "fraud"))
  newData <- SMOTE(Insp ~ ., data, perc.over = 700)
  model <- naiveBayes(Insp ~ ., newData)
  preds <- predict(model, test[, c("ID", "Prod", "Uprice",
                                   "Insp")], type = "raw")
  return(list(rankOrder = order(preds[, "fraud"], decreasing = T),
              rankScore = preds[, "fraud"]))
}
ho.nbs <- function(form, train, test, ...) {
  res <- nb.s(train,test)
  structure(evalOutlierRanking(test,res$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
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
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)
```

```r
)
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'))
```

```r
library(RWeka)
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.
```

```
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$Insp,levels=c('ok','fraud'))
  model <- AdaBoostM1(Insp ~ .,data,
                      control=Weka_control(I=100))
  preds <- predict(model,test[,c('ID','Prod','Uprice','Insp')],
                   type='probability')
  return(list(rankOrder=order(preds[,'fraud'],decreasing=T),
              rankScore=preds[,'fraud'])
  )
}
ho.ab <- function(form, train, test, ...) {
  res <- ab(train,test)
  structure(evalOutlierRanking(test,res$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
```

```
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
```

```
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'))
```

```
library(DMwR)
library(e1071)
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
```

```
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'))
```

```
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
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
```

```r
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"))
```