ABI Group Assignment Midterm

Detecting Fraudulent Transactions

Each of the 401,146 rows of the data table includes information on one report by some salesman. This information includes his ID, the product ID, and the quantity and total value reported by the salesman. This data has already gone through some analysis at the company. The result of this analysis is shown in the last column, which has the outcome of the inspection of some transactions by the company.

```
#Loading the required libraries and loading the dataset
library(DMwR)

## Loading required package: lattice

## Loading required package: grid

data(sales)

#Printing the headof the dataset
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
```

Data Exploration

Let us explore the dataset to get the idea of the same

```
#Prints the summary of the dataset
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
```

```
##
##
##
##

#Prints the total number of unique values in the column
nlevels(sales$ID)

## [1] 6016

nlevels(sales$Prod)

## [1] 4548
```

We can say from the above results that we have 6016 unique sales ids and 4548 different Product Ids.

Let us now look at the NA values in the relevant columns

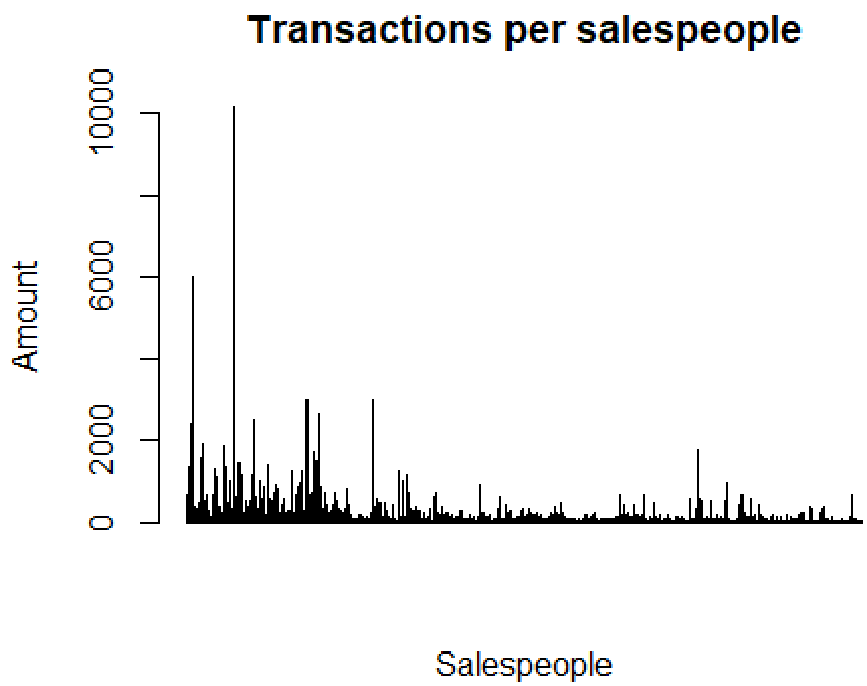
```
#Prints the total number of rows where there are NA values in both the
columns for a particular row
length(which(is.na(sales$Quant) & is.na(sales$Val)))

## [1] 888

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

## [1] 888
```

Here, we can see that there are about 888 transactions where we have NA values for both the columns

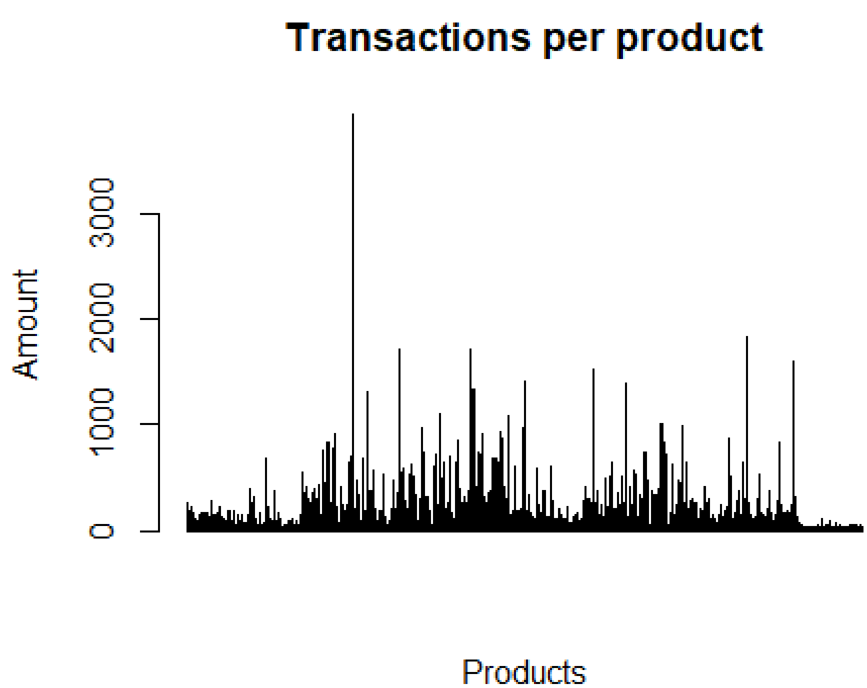
```
#Prints the tabular percentage of the Inspection type
table(sales$Insp)/nrow(sales) * 100

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

Below we print the total number of Transactions per Sales Id and Per Product Id

```
#Stores the Total number of Transaction per Sales Id and per Product Id
respectively
totS <- table(sales$ID)
totP <- table(sales$Prod)

#Shows a barplot of Number of Transactions per Sales Id and per Product
barplot(totS, main = "Transactions per salespeople", names.arg = "", xlab =
"Salespeople", ylab = "Amount")
```




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

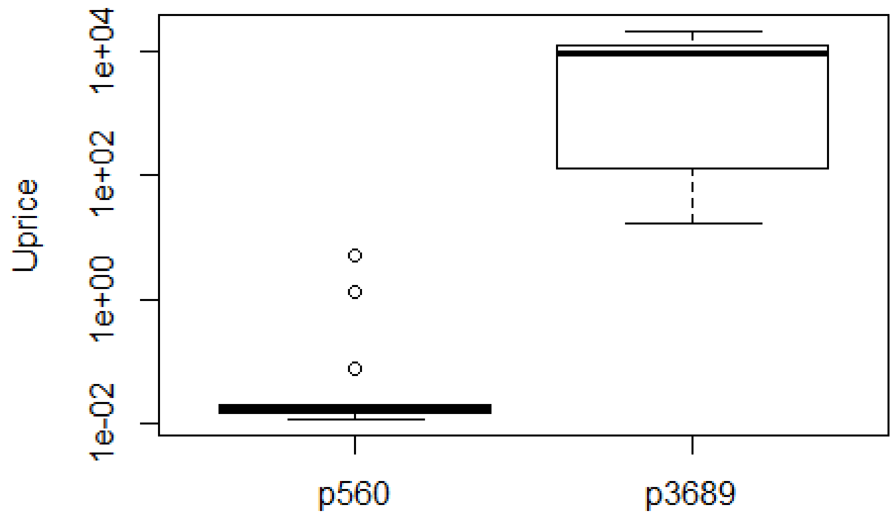


Let us now calculate the unit price of the product which is Value/quantity

```
#Calculates the Unite Price of the Quantity and stores it in new column
sales$Uprice <- sales$Val/sales$Quant

#Prints the summary of the column
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
```

Calculating the most expensive and most cheap products

```
#Attaching the dataset
attach(sales)

#Aggregating the median price per product Id
upp <- aggregate(Uprice,list(Prod),median,na.rm=T)

#Storing the most expensive and most cheap products
topP <- sapply(c(T,F),function(o) upp[order(upp[,2],decreasing=o)[1:5],1])

#Renaming the column names to Expensive and Cheap
colnames(topP) <- c('Expensive','Cheap')

#Printing the above dataframe
topP

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

#Stores the transactions for most expensive and cheap products
tops <- sales[Prod %in% topP[1, ], c("Prod", "Uprice")]

#Changing the levels of the Product column since we just have 2 levels we can
just factorize it
tops$Prod <- factor(tops$Prod)

#Plotting a boxplot of Unit price for particular Product
boxplot(Uprice ~ Prod, data = tops, ylab = "Uprice", log = "y")
```



Carrying out similar analysis for Sales Ids

```
#Aggregating the total transactions per Sales Id
vs <- aggregate(Val,list(ID),sum,na.rm=T)

#Storing the top increasing and decreasing values of transaction per sales Id
scoresSs <- sapply(c(T,F), function(o) vs[order(vs[,2],decreasing
=o)[1:5],1])

#Prints the created list
scoresSs

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

#Calculating the percentage of sales for top 100 sales ID
sum(vs[order(vs$x, decreasing = T)[1:100],2])/sum(Val, na.rm=T )*100

## [1] 38.33277

#Calculating the percentage of sales for bottom 2000 sales ID
sum(vs[order(vs$x, decreasing = F)[1:2000], 2])/sum(Val, na.rm = T) * 100

## [1] 1.988716
```

```r
#Aggregating the total transactions per Product
qs <- aggregate(Quant,list(Prod),sum,na.rm=T)

#Storing the top increasing and decreasing values of transaction per Product Id
scoresPs <- sapply(c(T,F),function(o) qs[order(qs$x,decreasing=o)[1:5],1])

#Changing the column names to Most and Least
colnames(scoresPs) <- c('Most','Least')

#Printing the dataframe to console
scoresPs

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

#Calculating the percentage of sales for top 100 Product ID
sum(as.double(qs[order(qs$x,decreasing=T)[1:100],2]))/
sum(as.double(Quant),na.rm=T)*100

## [1] 74.63478

#Calculating the percentage of sales for bottom 4000 Product ID
sum(as.double(qs[order(qs$x,decreasing=F)[1:4000],2]))/
sum(as.double(Quant),na.rm=T)*100

## [1] 8.944681
```

We can say from the above numbers that top 100 products contribute to almost 75% of the sales where as the bottom 4000 Product Id contribute to less than 10% of the sales volume

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

#Printing the most outliers
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

#Prints the total number of outlier transactions
sum(out)

## [1] 29446
```

```r
#Calculates the percentage of the Outlier transactions
sum(out)/nrow(sales) * 100

## [1] 7.34047
```

We can say from ythe above output that , 29,446 transactions are considered outliers, which corresponds to approximately 7% of the total number of transactions.

Data Problems As mentioned before, the main concern are transactions that have both the value of Quant and Val missing. Removing all 888 cases may be problematic if this leads to removing most transactions of some product or salesperson. Let us check this.

```r
#Prints the total number of Transactions per Sales Id and ProductId
respectively
totS <- table(ID)
totP <- table(Prod)

#Storing the dataframe of the Id and Product whose both the values are NA
nas <- sales[which(is.na(Quant) & is.na(Val)), c("ID", "Prod")]
```

We now obtain the salespeople with a larger proportion of transactions with unknowns on both Val and Quant:

```r
#Calculate the percentage of Sales Id
propS <- 100 * table(nas$ID)/totS

#Printing the most Sales Ids for whom we have both the NA values
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

#Calculating and Printing the most Product Id for whom we have both NA values
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
```

We can say from the above table that more than 20% of their transactionsremoved; and in particular, product p2689 would have almost 40% of them removed.

In summary, the option of removing all transactions with unknown values on both the quantity and the value is the best option we have:

```r
#Detaching the sales dataset
detach(sales)

#Deleting the rows where we have NA values for both the columns
sales <- sales[-which(is.na(sales$Quant) & is.na(sales$Val)),]
```

Let us now analyze the remaining reports with unknown values in either the quantity or the value of the transaction. We start by calculating the proportion of transactions of each product that have the quantity unknown:

```r
#Calculating the number of NA values we have for A particular ProductId
nnasQp <- tapply(sales$Quant,list(sales$Prod),function(x) sum(is.na(x)))

#Calculating the proportion of NA values we have for the Product Id
propNAsQp <- nnasQp/table(sales$Prod)

#Printing the top 10 ProductIds which have most number of NA values for
Quantity
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
```

There are two products (p2442 and p2443) that have all their transactions with unknown values of the quantity. Omitting these rows where we have all NA values for Quantity

```r
#Omitting the rows of the two desired ProductIds
sales <- sales[!sales$Prod %in% c("p2442", "p2443"), ]

#Printing the levels of the column
nlevels(sales$Prod)

## [1] 4548

#Refactoring the column since we have dropped the rows for which we have all
NA values for Quantity
sales$Prod <- factor(sales$Prod)

#Printing the new levels of the dataset
nlevels(sales$Prod)

## [1] 4546
```

Now, we find the sales people with all the Unkown Quantities

```r
#Calculating the Sales Id for which we have Unknown values of Quantity
nnasQs <- tapply(sales$Quant, list(sales$ID), function(x) sum(is.na(x)))

#Calculating the fraction of NA values for that particular Sales Id
```

```
propNAsQs <- nnasQs/table(sales$ID)

#Printing the first 10 SalesID for Which we have most NA values
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
```

Now we carry similar analysis for Value column

```
#Calculating the NULl values for the Product ID
nnasVp <- tapply(sales$Val,list(sales$Prod), function(x) sum(is.na(x)))

#Calculating the proportion
propNAsVp <- nnasVp/table(sales$Prod)

#Printing the most 10 Product Ids
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
```

Calculating the Null Values for SalesId

```
#Calculating the NA values for VAL for the Sales ID
nnasVs <- tapply(sales$Val, list(sales$ID), function(x) sum(is.na(x)))

#Calculating the proportion
propNAsVs <- nnasVs/table(sales$ID)

#Printing the most NA values
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
```

Since the numbers are not large we donot drop any columns

Let us now calculate the Median Price per ProductID

```
#Calculates the median UnitPrice for each Prduct ID
tPrice <- tapply(sales[sales$Insp != "fraud", "Uprice"],list(sales[sales$Insp
!= "fraud", "Prod"]), median, na.rm=T)
```

We shall use these Median Price to calculate the missing Quantity or Missing VAL

```r
#Storing the row numberd for which we have missing Quantity
noQuant <- which(is.na(sales$Quant))

#Storing the celing value for unknown Quantity and known VAL
sales[noQuant,'Quant'] <- ceiling(sales[noQuant,'Val'] /
tPrice[sales[noQuant,'Prod']])

#Storing the row numberd for which we have missing VAL
noVal <- which(is.na(sales$Val))

#Storing the Median Price*Quantity for Missing VAL entries
sales[noVal,'Val'] <- sales[noVal,'Quant'] *  tPrice[sales[noVal,'Prod']]
```

We have just filled in 12,900 unknown quantity values plus 294 total values of transaction

We can recalculate the Uprice column to fill in the previously unknown unit prices:

```r
#Recalculating the Unit Price of the Products
sales$Uprice <- sales$Val/sales$Quant
```

We now have the dataset free of unknown values after all these preprocessing. Let us now save this clean data.

```r
#Saving the cleaned data
save(sales, file = "salesClean.Rdata")
```

Few Transactions of Some Products

```r
#Attaching the dataset
attach(sales)

#Storring the row numbers for which we have no fradulent transaction
notF <- which(Insp != 'fraud')

#Calculating the boxplot statistics of Unit price per Product Id and storing
the median and inter quartile range for the Product ID
ms <- tapply(Uprice[notF],list(Prod=Prod[notF]),function(x) {
 bp <- boxplot.stats(x)$stats
 c(median=bp[3],iqr=bp[4]-bp[2])
 })

#Storing ms in the form of MAtrix
ms <- matrix(unlist(ms), length(ms), 2 , byrow = T, dimnames =
list(names(ms),c('median','iqr')))

#Printing the head of the matrix
head(ms)

##       median      iqr
## p1 11.346154 8.575599
## p2 10.877863 5.609731
```

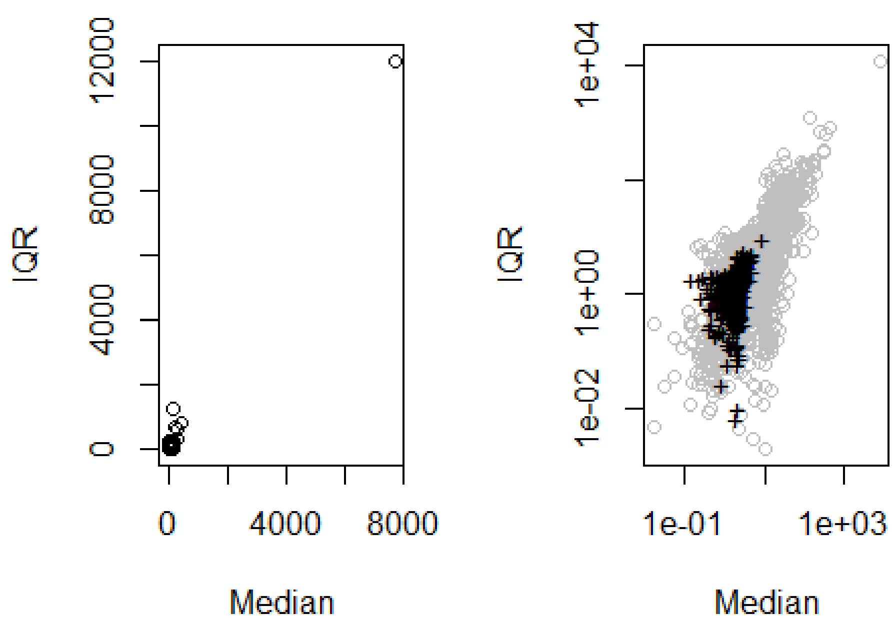
```
## p3 10.000000 4.809092
## p4  9.911243 5.998530
## p5 10.957447 7.136601
## p6 13.223684 6.685185
```

Let us now visualize some of the parameters

```r
#Setting the frames for visualization
par(mfrow = c(1, 2))

#Printing the scatterplot
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
```

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



Performing KS test to find the most similar Product IDs for less frequent Transactions

```r
#Sacling the ms matrix
dms <- scale(ms)

#Storing the ProductIds for which we have less than 20 Transactions
smalls <- which(table(Prod) < 20)
```

```r
names(smalls)
```

```
##   [1] "p8"    "p18"   "p38"   "p39"   "p40"   "p47"   "p54"   "p55"  
##   [9] "p58"   "p65"   "p75"   "p76"   "p78"   "p80"   "p85"   "p89"  
##  [17] "p102"  "p120"  "p123"  "p124"  "p125"  "p131"  "p133"  "p134" 
##  [25] "p138"  "p140"  "p141"  "p145"  "p147"  "p149"  "p154"  "p161" 
##  [33] "p162"  "p163"  "p164"  "p167"  "p169"  "p170"  "p172"  "p173" 
##  [41] "p174"  "p175"  "p176"  "p180"  "p183"  "p184"  "p187"  "p189" 
##  [49] "p190"  "p193"  "p195"  "p196"  "p198"  "p208"  "p209"  "p212" 
##  [57] "p217"  "p219"  "p223"  "p224"  "p233"  "p236"  "p239"  "p243" 
##  [65] "p247"  "p248"  "p260"  "p262"  "p263"  "p264"  "p266"  "p267" 
##  [73] "p271"  "p273"  "p280"  "p281"  "p302"  "p303"  "p307"  "p310" 
##  [81] "p311"  "p326"  "p329"  "p333"  "p336"  "p344"  "p345"  "p346" 
##  [89] "p348"  "p349"  "p350"  "p351"  "p353"  "p356"  "p357"  "p359" 
##  [97] "p360"  "p375"  "p385"  "p387"  "p388"  "p392"  "p398"  "p402" 
## [105] "p403"  "p404"  "p415"  "p437"  "p441"  "p466"  "p467"  "p468" 
## [113] "p475"  "p476"  "p503"  "p504"  "p510"  "p513"  "p515"  "p517" 
## [121] "p518"  "p519"  "p521"  "p522"  "p523"  "p525"  "p531"  "p534" 
## [129] "p540"  "p546"  "p550"  "p554"  "p560"  "p563"  "p565"  "p571" 
## [137] "p576"  "p580"  "p581"  "p582"  "p591"  "p597"  "p602"  "p607" 
## [145] "p609"  "p610"  "p611"  "p612"  "p613"  "p617"  "p625"  "p627" 
## [153] "p628"  "p629"  "p631"  "p632"  "p633"  "p634"  "p638"  "p639" 
## [161] "p641"  "p643"  "p648"  "p649"  "p650"  "p652"  "p655"  "p657" 
## [169] "p658"  "p659"  "p660"  "p662"  "p663"  "p667"  "p669"  "p670" 
## [177] "p671"  "p675"  "p678"  "p679"  "p684"  "p685"  "p688"  "p689" 
## [185] "p690"  "p693"  "p695"  "p696"  "p700"  "p702"  "p703"  "p704" 
## [193] "p706"  "p713"  "p718"  "p721"  "p726"  "p732"  "p733"  "p735" 
## [201] "p745"  "p748"  "p750"  "p753"  "p754"  "p756"  "p759"  "p760" 
## [209] "p762"  "p771"  "p773"  "p792"  "p798"  "p826"  "p844"  "p853" 
## [217] "p885"  "p893"  "p894"  "p896"  "p899"  "p900"  "p902"  "p903" 
## [225] "p910"  "p912"  "p945"  "p950"  "p967"  "p1003" "p1007" "p1013"
## [233] "p1017" "p1020" "p1022" "p1045" "p1047" "p1065" "p1073" "p1075"
## [241] "p1079" "p1089" "p1091" "p1094" "p1106" "p1110" "p1132" "p1133"
## [249] "p1137" "p1150" "p1169" "p1170" "p1173" "p1176" "p1178" "p1179"
## [257] "p1180" "p1181" "p1197" "p1198" "p1200" "p1203" "p1217" "p1227"
## [265] "p1249" "p1260" "p1268" "p1270" "p1273" "p1276" "p1284" "p1286"
## [273] "p1288" "p1298" "p1304" "p1306" "p1307" "p1312" "p1318" "p1319"
## [281] "p1320" "p1322" "p1333" "p1339" "p1348" "p1349" "p1352" "p1353"
## [289] "p1355" "p1359" "p1360" "p1365" "p1366" "p1371" "p1382" "p1387"
## [297] "p1388" "p1392" "p1402" "p1403" "p1415" "p1427" "p1432" "p1433"
## [305] "p1461" "p1469" "p1472" "p1474" "p1491" "p1494" "p1500" "p1502"
## [313] "p1508" "p1514" "p1531" "p1532" "p1537" "p1543" "p1558" "p1566"
## [321] "p1568" "p1570" "p1625" "p1632" "p1636" "p1640" "p1642" "p1643"
## [329] "p1647" "p1649" "p1653" "p1655" "p1657" "p1662" "p1677" "p1688"
## [337] "p1695" "p1739" "p1741" "p1742" "p1743" "p1745" "p1750" "p1755"
## [345] "p1758" "p1765" "p1772" "p1786" "p1791" "p1794" "p1812" "p1819"
## [353] "p1820" "p1824" "p1858" "p1888" "p1900" "p1902" "p1934" "p1947"
## [361] "p1953" "p1959" "p2000" "p2003" "p2005" "p2054" "p2058" "p2060"
## [369] "p2069" "p2073" "p2117" "p2131" "p2146" "p2155" "p2172" "p2179"
```

```
## [377] "p2180" "p2184" "p2186" "p2197" "p2202" "p2205" "p2210" "p2227"
## [385] "p2234" "p2237" "p2238" "p2239" "p2241" "p2242" "p2264" "p2274"
## [393] "p2291" "p2297" "p2300" "p2306" "p2313" "p2323" "p2328" "p2329"
## [401] "p2332" "p2333" "p2337" "p2342" "p2350" "p2352" "p2356" "p2358"
## [409] "p2362" "p2383" "p2390" "p2399" "p2400" "p2402" "p2428" "p2432"
## [417] "p2434" "p2436" "p2448" "p2454" "p2466" "p2469" "p2470" "p2471"
## [425] "p2474" "p2480" "p2484" "p2487" "p2488" "p2509" "p2515" "p2518"
## [433] "p2519" "p2520" "p2521" "p2535" "p2536" "p2538" "p2539" "p2543"
## [441] "p2548" "p2550" "p2551" "p2554" "p2557" "p2564" "p2565" "p2566"
## [449] "p2576" "p2579" "p2582" "p2583" "p2585" "p2586" "p2587" "p2588"
## [457] "p2591" "p2594" "p2601" "p2602" "p2603" "p2605" "p2613" "p2615"
## [465] "p2617" "p2622" "p2627" "p2634" "p2636" "p2640" "p2643" "p2644"
## [473] "p2650" "p2657" "p2666" "p2677" "p2680" "p2689" "p2700" "p2730"
## [481] "p2739" "p2743" "p2745" "p2746" "p2747" "p2765" "p2770" "p2776"
## [489] "p2780" "p2791" "p2807" "p2810" "p2824" "p2828" "p2832" "p2847"
## [497] "p2851" "p2867" "p2871" "p2872" "p2886" "p2893" "p2897" "p2906"
## [505] "p2922" "p2941" "p2964" "p2967" "p2971" "p2973" "p2975" "p2976"
## [513] "p2978" "p2979" "p2985" "p2992" "p2993" "p2996" "p3001" "p3002"
## [521] "p3004" "p3006" "p3009" "p3022" "p3039" "p3047" "p3049" "p3052"
## [529] "p3062" "p3068" "p3071" "p3084" "p3090" "p3091" "p3094" "p3097"
## [537] "p3104" "p3109" "p3111" "p3123" "p3125" "p3128" "p3134" "p3137"
## [545] "p3138" "p3145" "p3146" "p3147" "p3149" "p3150" "p3151" "p3158"
## [553] "p3159" "p3161" "p3162" "p3175" "p3183" "p3195" "p3203" "p3220"
## [561] "p3221" "p3226" "p3230" "p3231" "p3233" "p3234" "p3236" "p3239"
## [569] "p3242" "p3243" "p3245" "p3246" "p3250" "p3254" "p3259" "p3260"
## [577] "p3263" "p3268" "p3283" "p3287" "p3288" "p3289" "p3290" "p3302"
## [585] "p3310" "p3313" "p3314" "p3321" "p3323" "p3324" "p3329" "p3341"
## [593] "p3346" "p3380" "p3397" "p3402" "p3403" "p3414" "p3421" "p3425"
## [601] "p3426" "p3445" "p3449" "p3455" "p3458" "p3461" "p3465" "p3466"
## [609] "p3490" "p3494" "p3498" "p3501" "p3502" "p3506" "p3524" "p3529"
## [617] "p3538" "p3540" "p3542" "p3549" "p3560" "p3561" "p3562" "p3565"
## [625] "p3574" "p3577" "p3579" "p3581" "p3585" "p3586" "p3587" "p3588"
## [633] "p3626" "p3632" "p3633" "p3656" "p3671" "p3674" "p3678" "p3680"
## [641] "p3687" "p3689" "p3691" "p3696" "p3707" "p3718" "p3721" "p3728"
## [649] "p3729" "p3732" "p3740" "p3744" "p3756" "p3764" "p3768" "p3770"
## [657] "p3773" "p3775" "p3782" "p3793" "p3794" "p3806" "p3808" "p3810"
## [665] "p3811" "p3813" "p3818" "p3820" "p3823" "p3826" "p3827" "p3829"
## [673] "p3831" "p3833" "p3834" "p3854" "p3857" "p3872" "p3878" "p3906"
## [681] "p3914" "p3932" "p3938" "p3948" "p3952" "p3955" "p3962" "p3963"
## [689] "p3968" "p3983" "p4004" "p4005" "p4014" "p4019" "p4022" "p4031"
## [697] "p4061" "p4064" "p4076" "p4101" "p4105" "p4115" "p4128" "p4130"
## [705] "p4131" "p4132" "p4134" "p4136" "p4137" "p4138" "p4139" "p4140"
## [713] "p4141" "p4142" "p4145" "p4147" "p4148" "p4149" "p4151" "p4152"
## [721] "p4153" "p4154" "p4155" "p4156" "p4157" "p4158" "p4159" "p4160"
## [729] "p4161" "p4162" "p4163" "p4164" "p4165" "p4166" "p4167" "p4168"
## [737] "p4170" "p4171" "p4172" "p4175" "p4177" "p4179" "p4180" "p4181"
## [745] "p4182" "p4183" "p4184" "p4185" "p4186" "p4187" "p4188" "p4189"
## [753] "p4191" "p4192" "p4193" "p4196" "p4198" "p4199" "p4200" "p4201"
## [761] "p4206" "p4207" "p4210" "p4213" "p4214" "p4215" "p4216" "p4217"
## [769] "p4219" "p4220" "p4221" "p4224" "p4225" "p4226" "p4227" "p4228"
```

```
## [777] "p4230" "p4231" "p4233" "p4235" "p4237" "p4238" "p4240" "p4241"
## [785] "p4243" "p4244" "p4246" "p4249" "p4250" "p4251" "p4252" "p4253"
## [793] "p4254" "p4257" "p4259" "p4260" "p4261" "p4262" "p4263" "p4267"
## [801] "p4268" "p4269" "p4271" "p4273" "p4274" "p4276" "p4278" "p4281"
## [809] "p4282" "p4284" "p4287" "p4288" "p4291" "p4292" "p4293" "p4296"
## [817] "p4297" "p4298" "p4300" "p4301" "p4302" "p4306" "p4307" "p4309"
## [825] "p4311" "p4312" "p4313" "p4314" "p4315" "p4318" "p4319" "p4321"
## [833] "p4322" "p4323" "p4324" "p4326" "p4327" "p4330" "p4332" "p4337"
## [841] "p4339" "p4340" "p4343" "p4344" "p4345" "p4346" "p4347" "p4348"
## [849] "p4349" "p4351" "p4352" "p4353" "p4354" "p4356" "p4357" "p4359"
## [857] "p4360" "p4361" "p4362" "p4363" "p4364" "p4365" "p4370" "p4371"
## [865] "p4372" "p4373" "p4374" "p4375" "p4378" "p4379" "p4380" "p4382"
## [873] "p4383" "p4388" "p4389" "p4390" "p4391" "p4392" "p4394" "p4397"
## [881] "p4401" "p4404" "p4405" "p4407" "p4409" "p4416" "p4417" "p4418"
## [889] "p4421" "p4422" "p4424" "p4426" "p4427" "p4428" "p4429" "p4430"
## [897] "p4431" "p4432" "p4433" "p4434" "p4435" "p4436" "p4437" "p4438"
## [905] "p4439" "p4440" "p4441" "p4442" "p4443" "p4444" "p4445" "p4446"
## [913] "p4447" "p4448" "p4450" "p4452" "p4453" "p4454" "p4455" "p4456"
## [921] "p4457" "p4458" "p4459" "p4460" "p4461" "p4463" "p4464" "p4465"
## [929] "p4466" "p4467" "p4468" "p4469" "p4471" "p4472" "p4474" "p4475"
## [937] "p4479" "p4480" "p4481" "p4482" "p4483" "p4484" "p4485" "p4487"
## [945] "p4488" "p4489" "p4490" "p4491" "p4493" "p4494" "p4495" "p4496"
## [953] "p4497" "p4498" "p4500" "p4502" "p4504" "p4507" "p4508" "p4509"
## [961] "p4510" "p4511" "p4514" "p4515" "p4516" "p4517" "p4518" "p4521"
## [969] "p4522" "p4523" "p4524" "p4525" "p4527" "p4528" "p4529" "p4530"
## [977] "p4531" "p4532" "p4533" "p4536" "p4540" "p4542" "p4546" "p4547"
## [985] "p4548"

#Storing the list of all the Unit Prices for the Product ID
prods <- tapply(sales$Uprice, sales$Prod, list)

#Creating a new matrix of length of colums equal to smalls and names for rows
equals names of the smalls list and column names equals as mentioned
similar <- matrix(NA, length(smalls), 7, dimnames = list(names(smalls),
c("Simil", "ks.stat", "ks.p", "medP", "iqrP", "medS", "iqrS")))

#Finding the all the relevant parameters of the matrix
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], ])
 }


## Warning in ks.test(prods[[smalls[i]]], prods[[order(d)[2]]]): cannot
## compute exact p-value with ties
```

```r
#Printing the head of the matrix
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
#Printing the most similar Product Id for the first ProductID
levels(Prod)[similar[1, 1]]
```

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

```r
#Finding the ProductId with 90% confidence intervals
nrow(similar[similar[, "ks.p"] >= 0.9, ])
```

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

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

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

```r
#Saving the similar data
save(similar, file = "similarProducts.Rdata")
```

Defining the Data Mining Tasks:

Precision and Recall

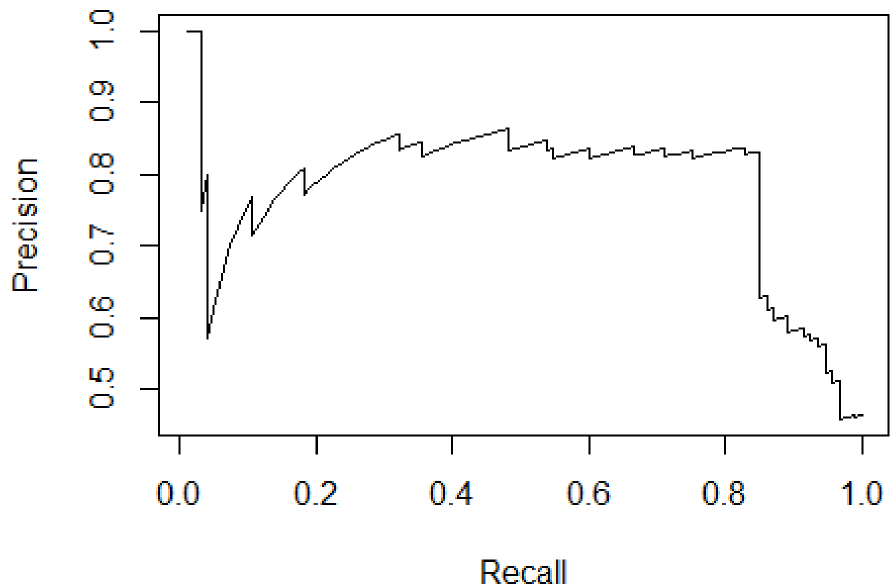
```r
#Loading the required libraries
library(ROCR)
```

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

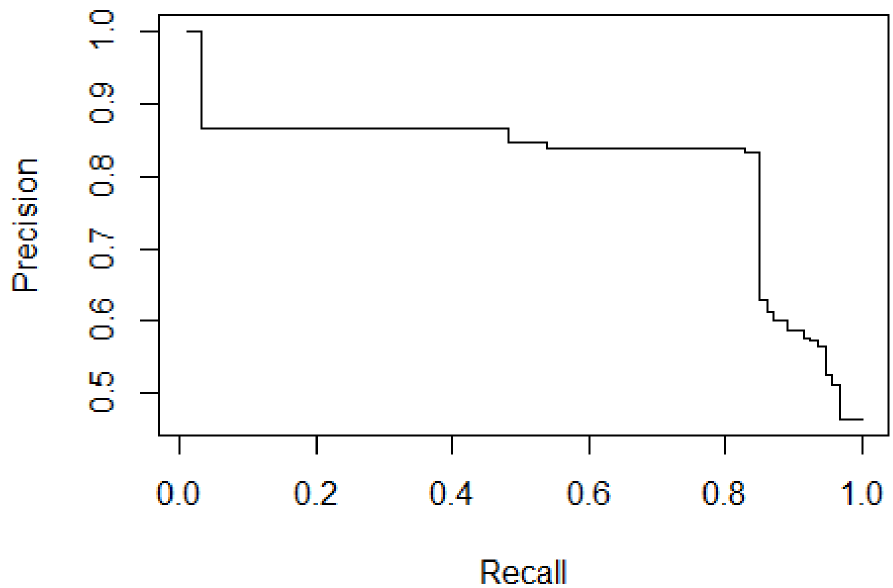
```
##
## Attaching package: 'gplots'
```

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

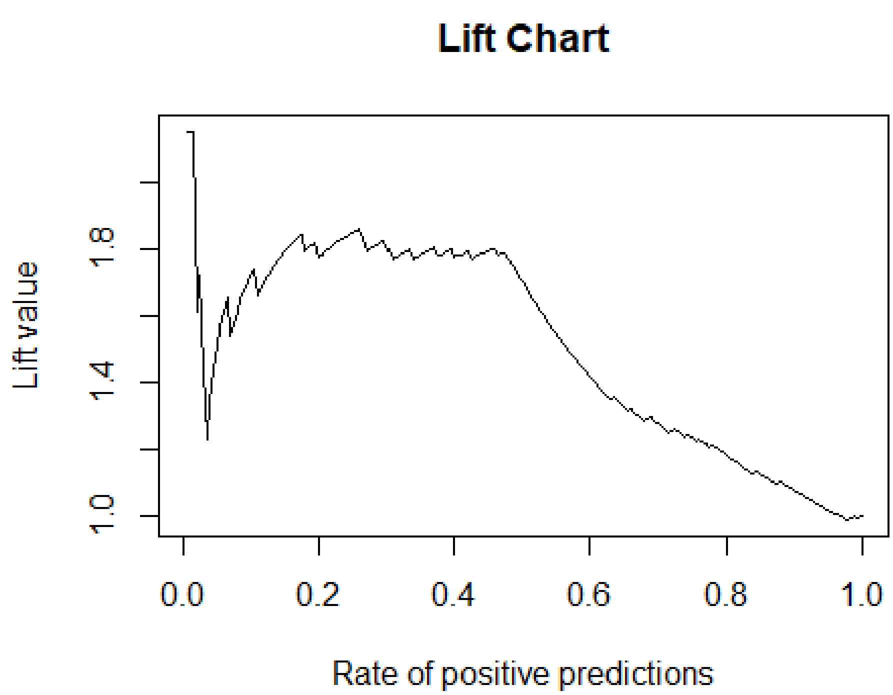
```r
data(ROCR.simple)

pred <- prediction(ROCR.simple$predictions, ROCR.simple$labels)

perf <- performance(pred, "prec", "rec")

plot(perf)
```



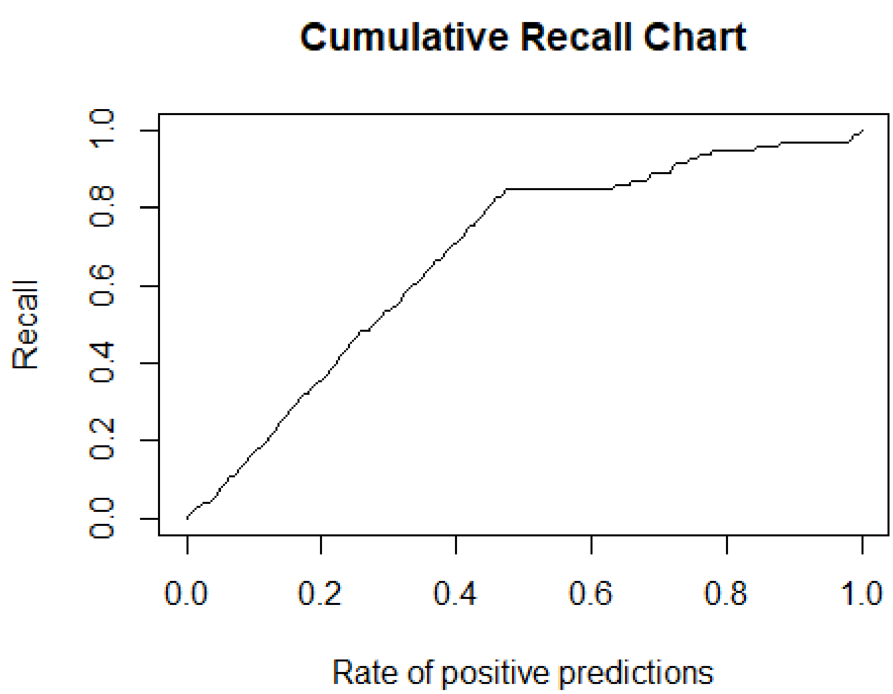

```r
PRcurve <- function(preds, trues, ...) {
  require(ROCR, quietly = T)
  pd <- prediction(preds, trues)
  pf <- performance(pd, "prec", "rec")
  pf@y.values <- lapply(pf@y.values, function(x) rev(cummax(rev(x))))
  plot(pf, ...)
}

PRcurve(ROCR.simple$predictions, ROCR.simple$labels)
```




```r
pred <- prediction(ROCR.simple$predictions, ROCR.simple$labels)
perf <- performance(pred, "lift", "rpp")
plot(perf, main = "Lift Chart")
```



```r
CRchart <- function(preds, trues, ...) {
 require(ROCR, quietly = T)
 pd <- prediction(preds, trues)
 pf <- performance(pd, "rec", "rpp")
 plot(pf, ...)
 }

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



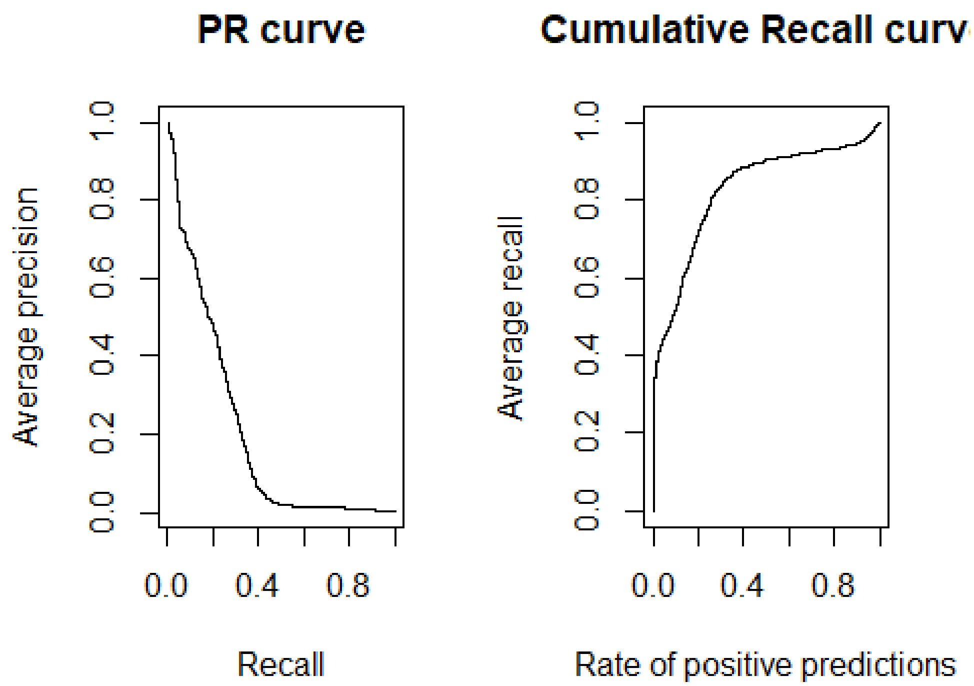

```r
avgNDTP <- function(toInsp,train,stats) {
 if (missing(train) && missing(stats))
 stop('Provide either the training data or the product stats')
 if (missing(stats)) {
 notF <- which(train$Insp != 'fraud')
 stats <- tapply(train$Uprice[notF],
 list(Prod=train$Prod[notF]),
 function(x) {
 bp <- boxplot.stats(x)$stats
 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)
}

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))
}
```

Obtaining Outlier Rankings

```
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(globalStats[,'iqr']==0),'median']

ho.BPrule <- function(form, train, test, ...) {
   res <- BPrule(train,test)
```

```r
 structure(evalOutlierRanking(test,res$rankOrder,...),
 itInfo=list(preds=res$rankScore,
 trues=ifelse(test$Insp=='fraud',1,0)))
 }

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

 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

 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')
```




```
 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))
)
}

lof.res <- holdOut(learner('ho.LOF',
pars=list(k=7,Threshold=0.1,
statsProds=globalStats)),
dataset(Insp ~ .,sales),
```

```r
hldSettings(3,0.3,1234,T),
itsInfo=TRUE
)
```

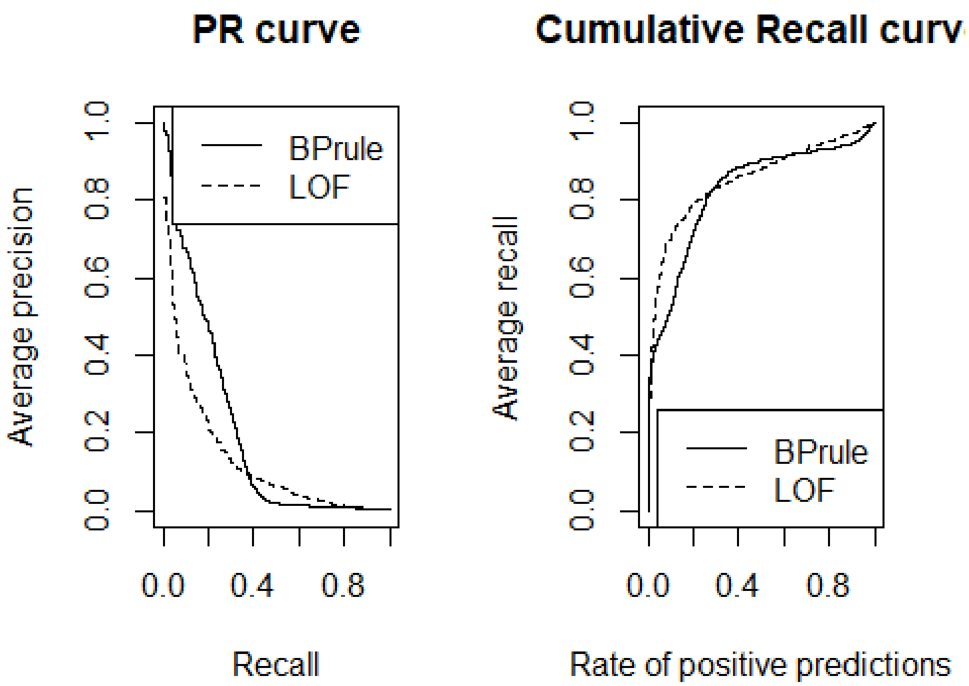
```
##
##  Stratified  3 x 70 %/ 30 % Holdout run with seed =  1234
## Repetition  1
## Repetition  2
## Repetition  3
```

```r
summary(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
```

```r
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))
```

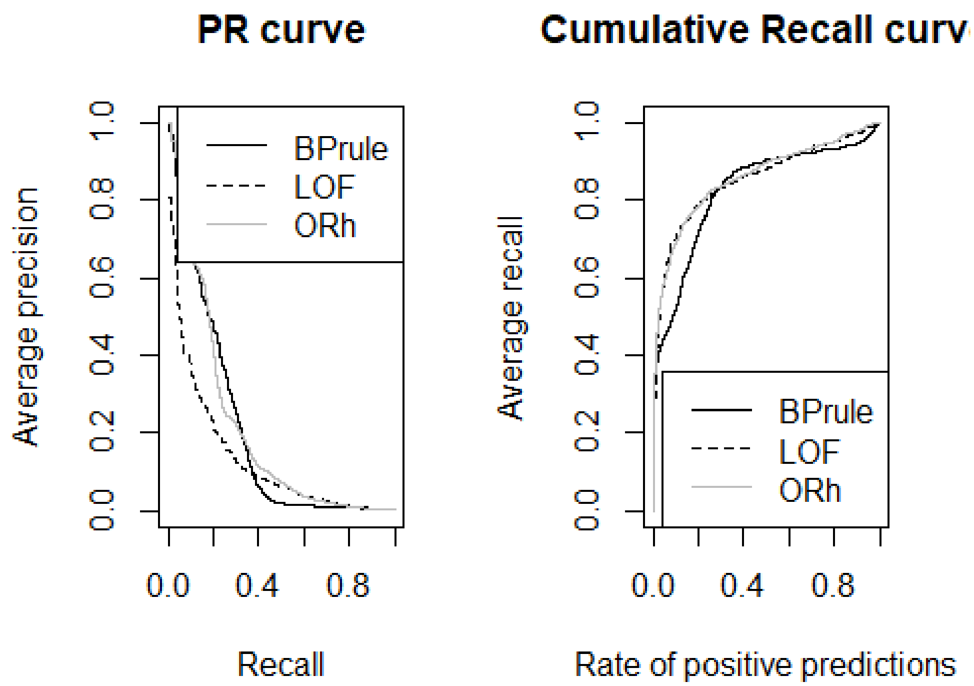



```
ho.ORh <- function(form, train, test, ...) {
ntr <- nrow(train)
all <- rbind(train,test)
N <- nrow(all)
ups <- split(all$Uprice,all$Prod)
r <- list(length=ups)
for(u in seq(along=ups))
r[[u]] <- if (NROW(ups[[u]]) > 3)
outliers.ranking(ups[[u]])$prob.outliers
else if (NROW(ups[[u]])) rep(0,NROW(ups[[u]]))
else NULL
all$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=globalStats)),
dataset(Insp ~ .,sales),
```

```r
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"
```

```r
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'))
```



Supervised Approaches

```
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),
```

```r
hldSettings(3,0.3,1234,T),
itsInfo=TRUE
)
```

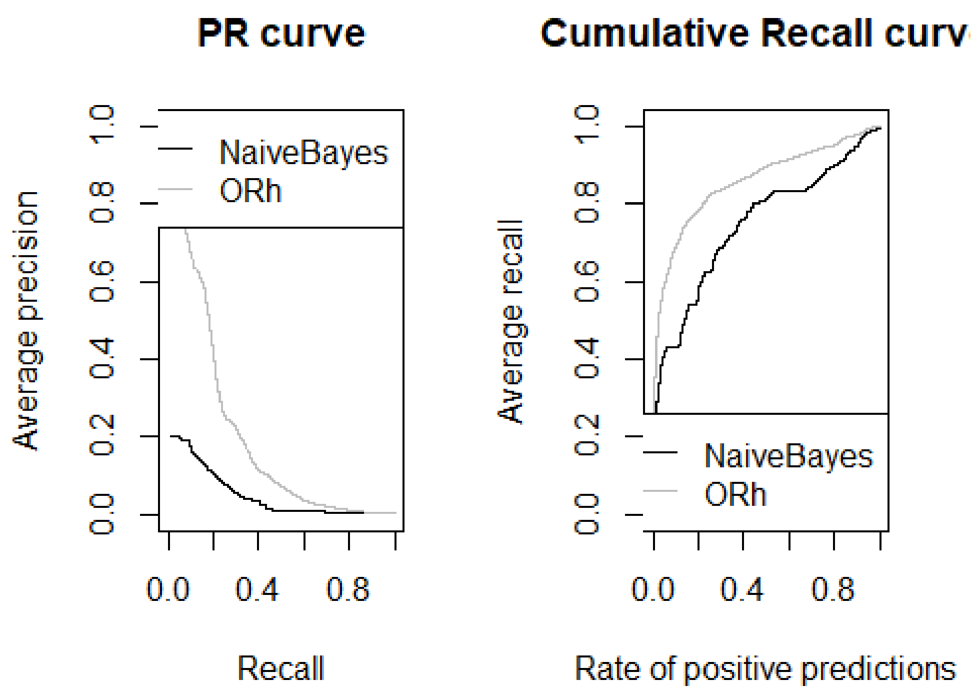
```
##
##  Stratified  3 x 70 %/ 30 % Holdout run with seed =  1234
## Repetition  1
## Repetition  2
## Repetition  3
```

```r
summary(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
```

```r
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),
```

```r
hldSettings(3,0.3,1234,T),
itsInfo=TRUE)
```

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

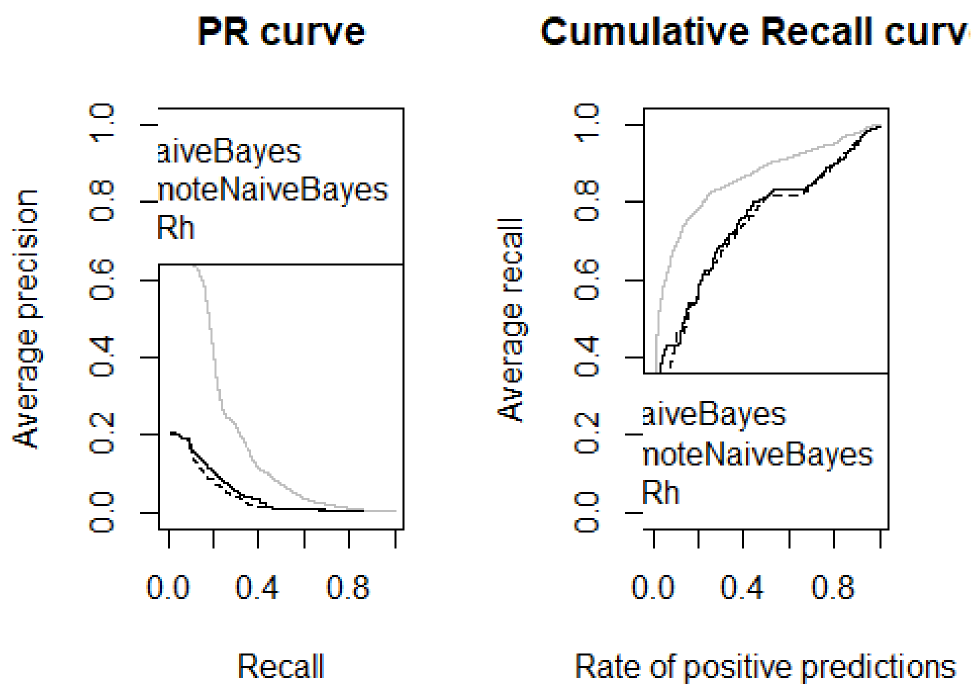
```r
summary(nbs.res)
```

```
##
## == 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
```

```r
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'))
```




```
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.
```

```r
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',
```

```r
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
```

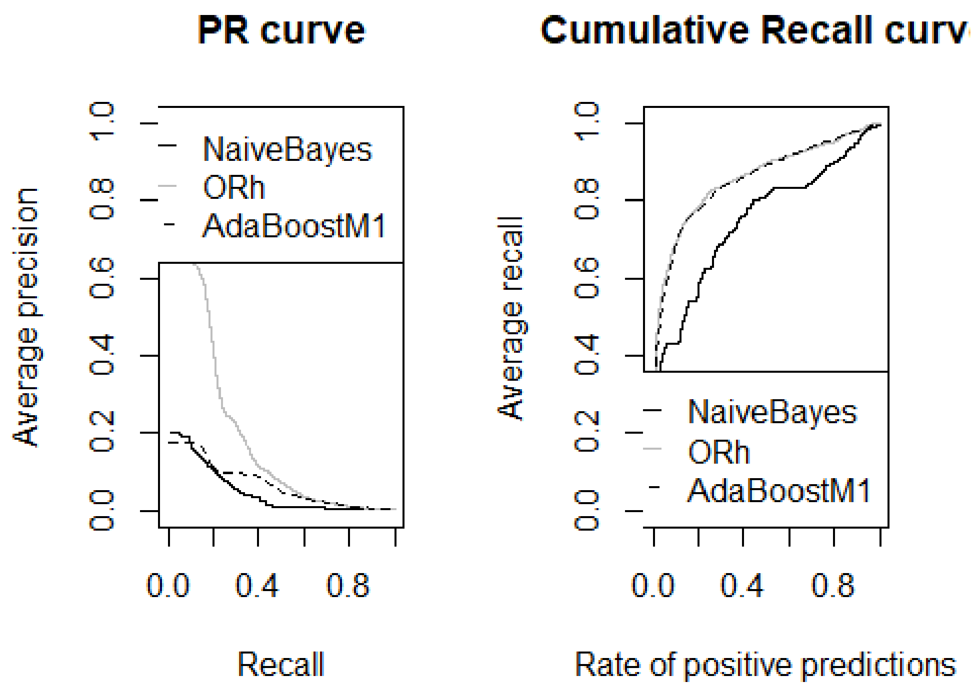
```r
summary(ab.res)
```

```
##
## == 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
```

```r
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],
```

```r
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'))
```



Semi Supervised learning

```r
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'))
```

```r
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
)
```

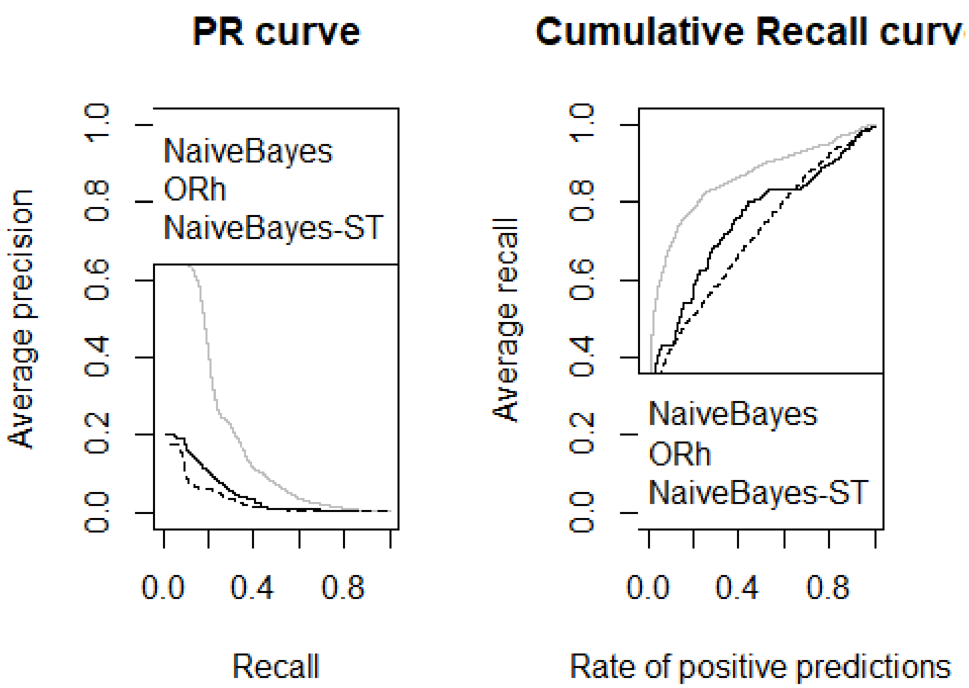
```
##
##  Stratified  3 x 70 %/ 30 % Holdout run with seed =  1234
## Repetition  1
## Repetition  2
## Repetition  3
```

```r
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
```

```r
par(mfrow=c(1,2))
info <- attr(nb.st.res,'itsInfo')
```

```r
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'))
```




```r
pred.ada <- function(m,d) {
p <- predict(m,d,type='probability')
```

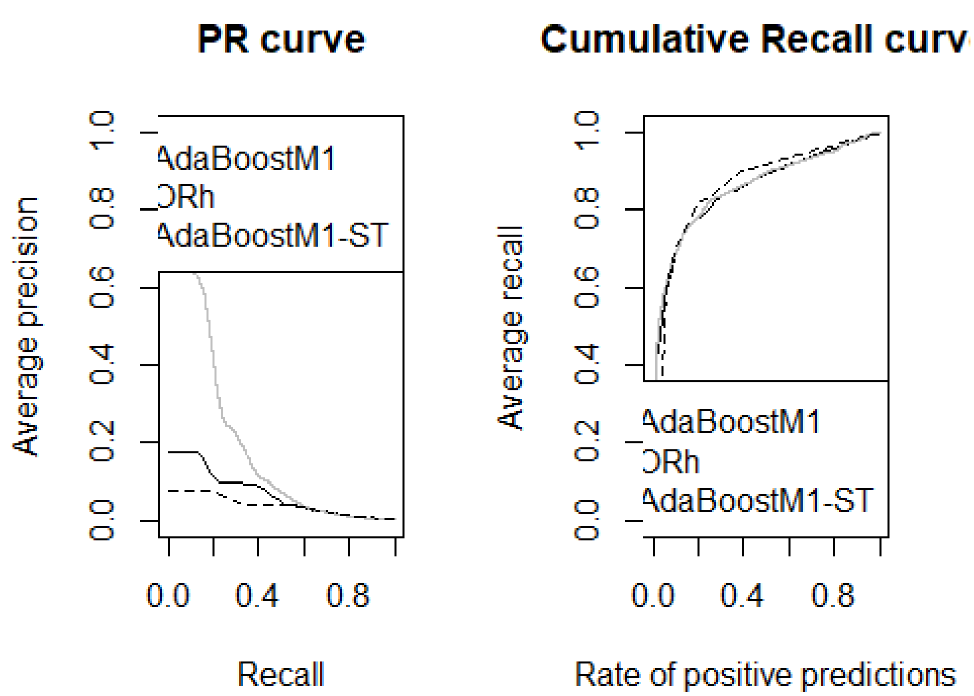
```r
data.frame(cl=colnames(p)[apply(p,1,which.max)],
p=apply(p,1,max))
}
ab.st <- function(train,test) {
require(RWeka,quietly=T)
train <- train[,c('ID','Prod','Uprice','Insp')]
train[which(train$Insp == 'unkn'),'Insp'] <- NA
train$Insp <- factor(train$Insp,levels=c('ok','fraud'))
model <- SelfTrain(Insp ~ .,train,
learner('AdaBoostM1',
list(control=Weka_control(I=100))),'pred.ada')
preds <- predict(model,test[,c('ID','Prod','Uprice','Insp')],
type='probability')
return(list(rankOrder=order(preds[,'fraud'],decreasing=T),
rankScore=preds[,'fraud']))
}
ho.ab.st <- function(form, train, test, ...) {
res <- ab.st(train,test)
structure(evalOutlierRanking(test,res$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
```

```r
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

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 main goal of this chapter was to introduce the reader to a new class of data mining problems: outliers ranking. In particular, we have used a dataset that enabled us to tackle this task from di???erent perspectives. Namely, we used supervised, unsupervised- and semi-supervised approaches to the problem. The application used in this chapter can be regarded as an instantiation of the general problem of finding unusual observations of a phenomenon having a limited amount of resources. Several real-world applications map into this general framework, such as detecting frauds in credit card transactions, telecommunications, tax declarations, etc. In the area of security, there are also several applications of this general concept of outlier ranking.