Tabloid data set

1 Description

A large retailer wants to explore the predictability of response to a tabloid mailing.

If they mail a tabloid to a customer in their data-base, can they predict whether or not the customer will respond by making a purchase.

The dependent variable is 1 if they buy something, 0 if they do not.

They tried to come up with x's based on past purchasing behavior.

The Predictive Analytics team builds a model for the probability the customer responds given information about the customer.

What information about a customer do they use?

- `nTab`: number of past orders.
- `moCbook`: months since last order.
- `iRecMer1`: 1/months since last order in merchandise category 1.
- `llDol`: log of the dollar value of past purchases.

The data for these variables is obtained from the companies operational data base.

2 Preprocessing

We download the data and preprocess it first

```
download.file(
    'https://github.com/ChicagoBoothML/MLClassData/raw/master/Tabloid/Tabloid_test.csv',
    'Tabloid_test.csv')

download.file(
    'https://github.com/ChicagoBoothML/MLClassData/raw/master/Tabloid/Tabloid_train.csv',
    'Tabloid_train.csv')


td = read.csv("Tabloid_train.csv")
td_test = read.csv("Tabloid_test.csv")

td$purchase = as.factor(td$purchase)
td_test$purchase = as.factor(td_test$purchase)
```

3 Summary statistics

```r
summary(td)
```

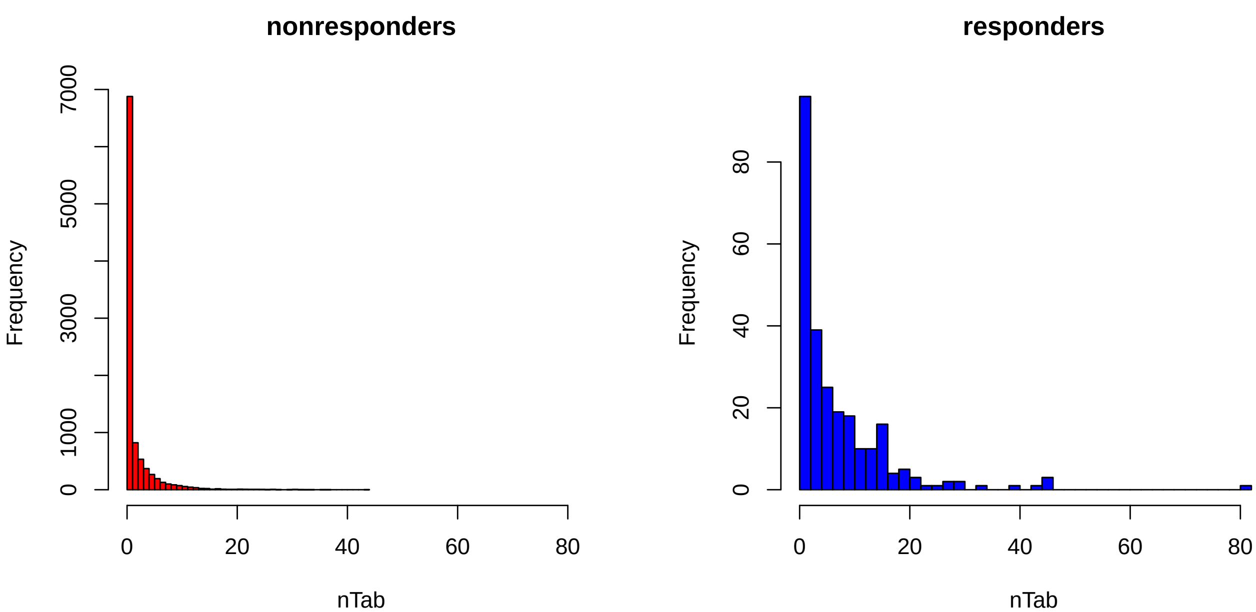
```
##  purchase      nTab         moCbook
##  0:9742   Min.   : 0.0   Min.   : 1.2
##  1: 258   1st Qu.: 0.0   1st Qu.:50.0
##           Median : 0.0   Median :50.0
##           Mean   : 1.9   Mean   :47.6
##           3rd Qu.: 2.0   3rd Qu.:50.0
##           Max.   :81.0   Max.   :50.0
##     iRecMer1          llDol
##  Min.   :0.020   Min.   :-2.30
##  1st Qu.:0.020   1st Qu.:-2.30
##  Median :0.020   Median :-2.30
##  Mean   :0.094   Mean   :-1.39
##  3rd Qu.:0.074   3rd Qu.:-2.30
##  Max.   :0.968   Max.   : 7.31
```

Notice that the percentage of households that make a purchase is pretty small!

$258/10000 = 0.0258$

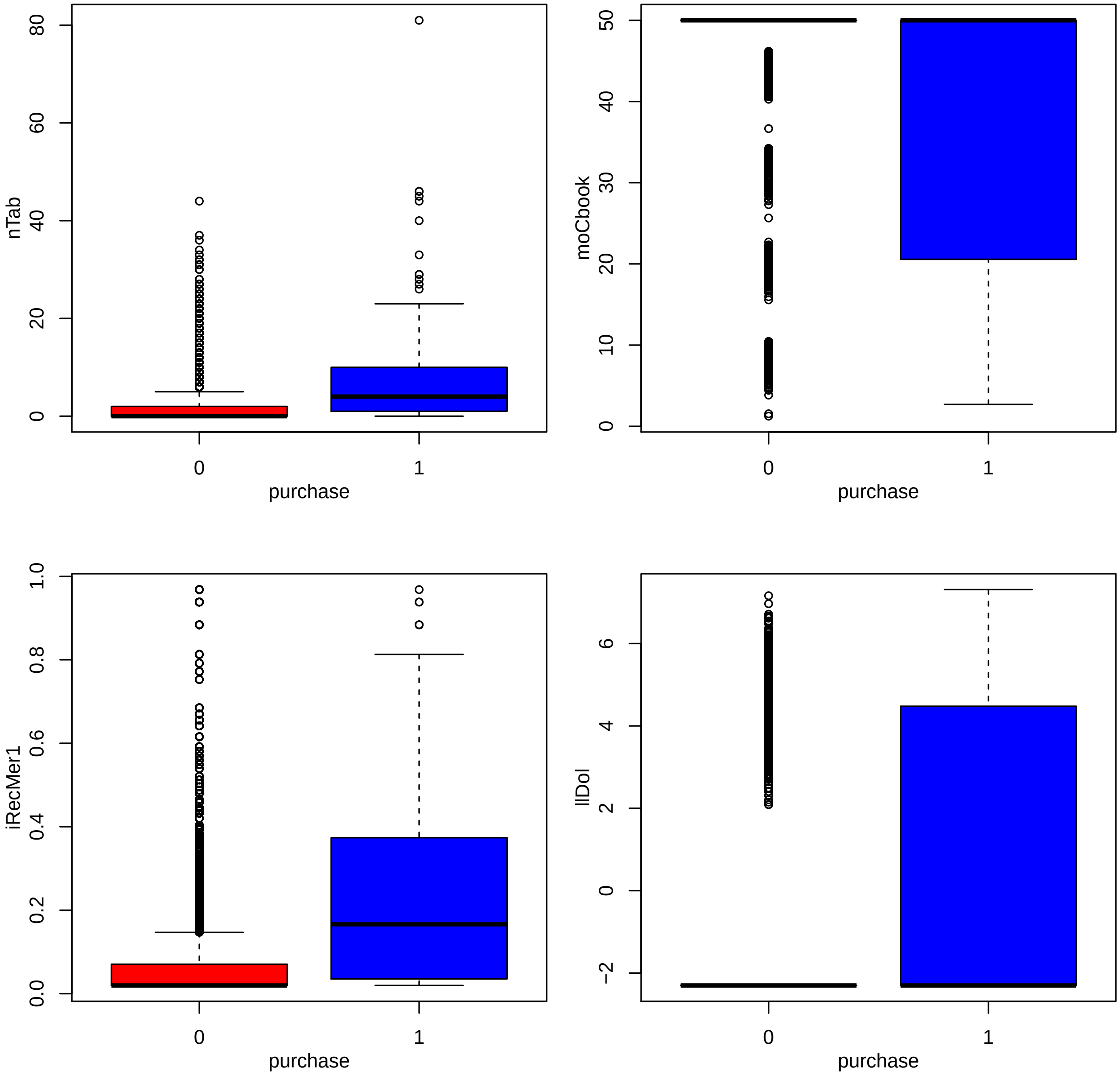
Illustration of how `nTab` is related to `responders`.

```r
par(mfrow=c(1,2))
hist(td[td$purchase==0, "nTab"], breaks=40, col="red",
     main="nonresponders", xlab="nTab", xlim=c(0,85))
hist(td[td$purchase==1, "nTab"], breaks=40, col="blue",
     main="responders", xlab="nTab", xlim=c(0,85))
```



Here is `Y` plotted vs. each of the four `X`'s

```r
par(mfrow=c(2,2), mar=c(3,3,3,1), mgp=c(2,1,0))
plot(nTab~purchase,td,col=c("red", "blue"))
plot(moCbook~purchase,td,col=c("red", "blue"))
plot(iRecMer1~purchase,td,col=c("red", "blue"))
plot(llDol~purchase,td,col=c("red", "blue"))
```



4 Fit models

We fit

- logistic regression
- random forest model
- boosting

```
library(tree)
library(randomForest)
library(gbm)
```

Create some helper function used for evaluation.

The following function is used to compute the deviance of a model

```
# deviance loss function
# y should be 0/1
# phat are probabilities obtain by our algorithm
# wht shrinks probs in phat towards .5 --- this helps avoid numerical problems don't use log(0)!
lossf = function(y,phat,wht=0.0000001) {
   if(is.factor(y)) y = as.numeric(y)-1
   phat = (1-wht)*phat + wht*.5
   py = ifelse(y==1, phat, 1-phat)
   return(-2*sum(log(py)))
}
```

The following will get confucion matrix:

```
# deviance loss function
# y should be 0/1
# phat are probabilities obtain by our algorithm
# thr is the cut off value - everything above thr is classified as 1
getConfusionMatrix = function(y,phat,thr=0.5) {
   if(is.factor(y)) y = as.numeric(y)-1
   yhat = ifelse(phat > thr, 1, 0)
   tb = table(predictions = yhat,
                   actual = y)
   rownames(tb) = c("predict_0", "predict_1")
   return(tb)
}
```

And finally, this function gives miss-classification rate:

```
# deviance loss function
# y should be 0/1
# phat are probabilities obtain by our algorithm
# thr is the cut off value - everything above thr is classified as 1
lossMR = function(y,phat,thr=0.5) {
   if(is.factor(y)) y = as.numeric(y)-1
   yhat = ifelse(phat > thr, 1, 0)
   return(1 - mean(yhat == y))
}
```

We need a place to store results

```
phatL = list() #store the test phat for the different methods here
```

4.1 Logistic regression

We fit a logistic regression model using all variables

```
lgfit = glm(purchase~., td, family=binomial)
print(summary(lgfit))
```

```
##
## Call:
## glm(formula = purchase ~ ., family = binomial, data = td)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.501  -0.188  -0.161  -0.157   2.968
##
## Coefficients:
##             Estimate Std. Error z value
## (Intercept) -2.62131    0.25667  -10.21
## nTab         0.05530    0.01209    4.57
## moCbook     -0.03249    0.00527   -6.17
## iRecMer1     1.72688    0.31282    5.52
## llDol        0.07842    0.02630    2.98
##             Pr(>|z|)
## (Intercept)  < 2e-16 ***
## nTab         4.8e-06 ***
## moCbook      7.0e-10 ***
## iRecMer1     3.4e-08 ***
## llDol         0.0029 **
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2396.5  on 9999  degrees of freedom
## Residual deviance: 2063.5  on 9995  degrees of freedom
## AIC: 2073
##
## Number of Fisher Scoring iterations: 7
```

Predictions are stored for later analysis

```
phat = predict(lgfit, td_test, type="response")
phatL$logit = matrix(phat,ncol=1)
```

4.2 Random Forest

We fit random forest models for a few different settings.

```
set.seed(99)

##settings for randomForest
p=ncol(td)-1
mtryv = c(p, sqrt(p))
ntreev = c(500,1000)
setrf = expand.grid(mtryv,ntreev)  # this contains all settings to try
colnames(setrf)=c("mtry","ntree")
phatL$rf = matrix(0.0,nrow(td_test),nrow(setrf))  # we will store results here

###fit rf
for(i in 1:nrow(setrf)) {
   #fit and predict
   frf = randomForest(purchase~., data=td,
                      mtry=setrf[i,1],
                      ntree=setrf[i,2],
                      nodesize=10)
   phat = predict(frf, newdata=td_test, type="prob")[,2]
   phatL$rf[,i]=phat
}
```

4.3 Boosting

We fit boosting models for a few different settings.

```
##settings for boosting
idv = c(2,4)
ntv = c(1000,5000)
shv = c(.1,.01)
setboost = expand.grid(idv,ntv,shv)
colnames(setboost) = c("tdepth","ntree","shrink")
phatL$boost = matrix(0.0,nrow(td_test),nrow(setboost))
```

Remember to convert to numeric 0,1 values for boosting.

```
tdB = td; tdB$purchase = as.numeric(tdB$purchase)-1
td_testB = td_test; td_testB$purchase = as.numeric(td_testB$purchase)-1
```

Fitting

```
for(i in 1:nrow(setboost)) {
   ##fit and predict
   fboost = gbm(purchase~., data=tdB, distribution="bernoulli",
              n.trees=setboost[i,2],
              interaction.depth=setboost[i,1],
              shrinkage=setboost[i,3])

   phat = predict(fboost,
                  newdata=td_testB,
                  n.trees=setboost[i,2],
                  type="response")

   phatL$boost[,i] = phat
}
```

5 Analysis of results

5.1 Miss-classification rate

Let us first look at miss-classification rate.

For **logistic regression** we have:

```
getConfusionMatrix(td_test$purchase, phatL[[1]][,1], 0.5)
```

```
##             actual
## predictions    0    1
##   predict_0 4884  108
##   predict_1    4    4
```

```
cat('Missclassification rate = ', lossMR(td_test$purchase, phatL[[1]][,1], 0.5), '\n')
```

```
## Missclassification rate =  0.0224
```

For **random forest** we have:

```
nrun = nrow(setrf)
for(j in 1:nrun) {
  print(setrf[j,])
  print("Confusion Matrix:")
  print(getConfusionMatrix(td_test$purchase, phatL[[2]][,j], 0.5))
  cat('Missclassification rate = ', lossMR(td_test$purchase, phatL[[2]][,j], 0.5), '\n')
}
```

```
##   mtry ntree
## 1    4   500
## [1] "Confusion Matrix:"
##             actual
## predictions    0    1
##   predict_0 4881  108
##   predict_1    7    4
## Missclassification rate =  0.023
##   mtry ntree
## 2    2   500
## [1] "Confusion Matrix:"
##             actual
## predictions    0    1
##   predict_0 4882  108
##   predict_1    6    4
## Missclassification rate =  0.0228
##   mtry ntree
## 3    4  1000
## [1] "Confusion Matrix:"
##             actual
## predictions    0    1
##   predict_0 4882  107
##   predict_1    6    5
## Missclassification rate =  0.0226
##   mtry ntree
## 4    2  1000
## [1] "Confusion Matrix:"
##             actual
## predictions    0    1
##   predict_0 4882  108
##   predict_1    6    4
## Missclassification rate =  0.0228
```

For **boosting** we have:

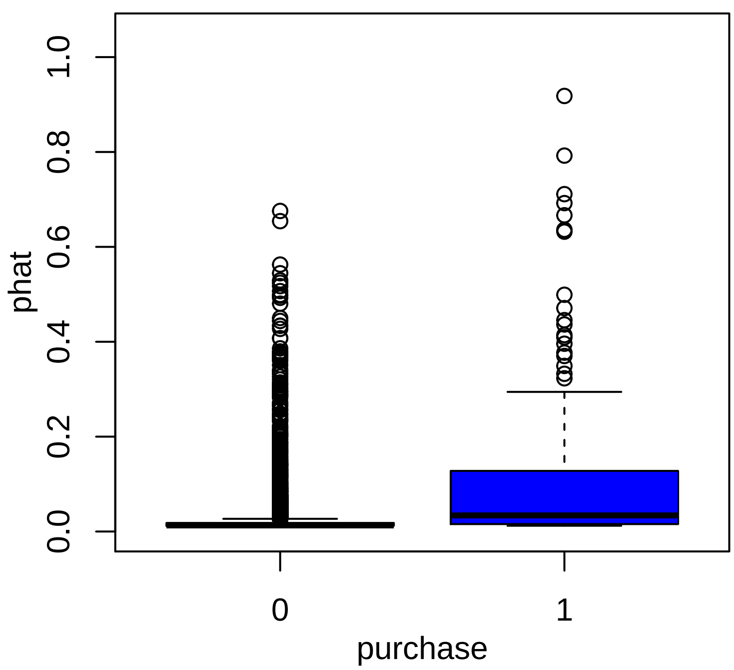
```r
nrun = nrow(setboost)
for(j in 1:nrun) {
  print(setboost[j,])
  print("Confusion Matrix:")
  print(getConfusionMatrix(td_test$purchase, phatL[[3]][,j], 0.5))
  cat('Missclassification rate = ', lossMR(td_test$purchase, phatL[[3]][,j], 0.5), '\n')
}
```

```
##   tdepth ntree shrink
## 1      2  1000    0.1
## [1] "Confusion Matrix:"
##             actual
## predictions    0    1
##   predict_0 4874  105
##   predict_1   14    7
## Missclassification rate =  0.0238
##   tdepth ntree shrink
## 2      4  1000    0.1
## [1] "Confusion Matrix:"
##             actual
## predictions    0    1
##   predict_0 4863  105
##   predict_1   25    7
## Missclassification rate =  0.026
##   tdepth ntree shrink
## 3      2  5000    0.1
## [1] "Confusion Matrix:"
##             actual
## predictions    0    1
##   predict_0 4864  106
##   predict_1   24    6
## Missclassification rate =  0.026
##   tdepth ntree shrink
## 4      4  5000    0.1
## [1] "Confusion Matrix:"
##             actual
## predictions    0    1
##   predict_0 4845  106
##   predict_1   43    6
## Missclassification rate =  0.0298
##   tdepth ntree shrink
## 5      2  1000   0.01
## [1] "Confusion Matrix:"
##             actual
## predictions    0    1
##   predict_0 4886  110
##   predict_1    2    2
## Missclassification rate =  0.0224
##   tdepth ntree shrink
## 6      4  1000   0.01
## [1] "Confusion Matrix:"
##             actual
## predictions    0    1
```

```
##   predict_0 4880  107
##   predict_1    8    5
## Missclassification rate =  0.023
##   tdepth ntree shrink
## 7      2  5000   0.01
## [1] "Confusion Matrix:"
##            actual
## predictions    0    1
##   predict_0 4878  107
##   predict_1   10    5
## Missclassification rate =  0.0234
##   tdepth ntree shrink
## 8      4  5000   0.01
## [1] "Confusion Matrix:"
##            actual
## predictions    0    1
##   predict_0 4874  106
##   predict_1   14    6
## Missclassification rate =  0.024
```

This is strange… There seems to be fit in the model.

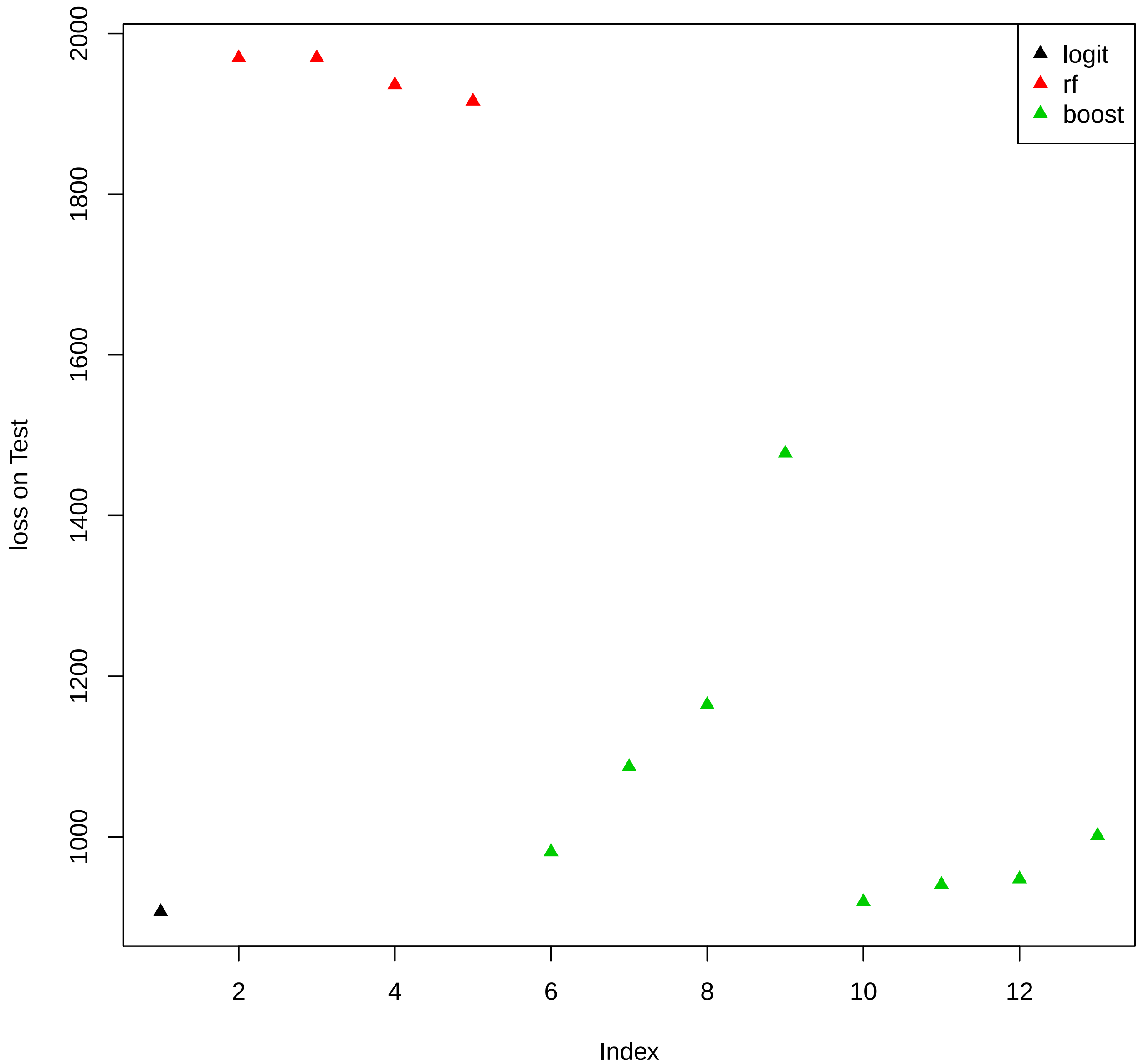
```
par(mar=c(3,3,3,1), mgp=c(2,1,0))
phat = predict(lgfit, newdata=td, type="response")
plot(phat~td$purchase, col=c("red","blue"),
     xlab="purchase", ylab="phat", ylim=c(0,1.05), cex.text=0.7)
```



5.2 Deviance

Plot test set loss — deviance:

```
lossL = list()
nmethod = length(phatL)
for(i in 1:nmethod) {
   nrun = ncol(phatL[[i]])
   lvec = rep(0,nrun)
   for(j in 1:nrun) lvec[j] = lossf(td_test$purchase, phatL[[i]][,j])
   lossL[[i]]=lvec; names(lossL)[i] = names(phatL)[i]
}
lossv = unlist(lossL)
plot(lossv, ylab="loss on Test", type="n")
nloss=0
for(i in 1:nmethod) {
   ii = nloss + 1:ncol(phatL[[i]])
   points(ii,lossv[ii],col=i,pch=17)
   nloss = nloss + ncol(phatL[[i]])
}
legend("topright",legend=names(phatL),col=1:nmethod,pch=rep(17,nmethod))
```

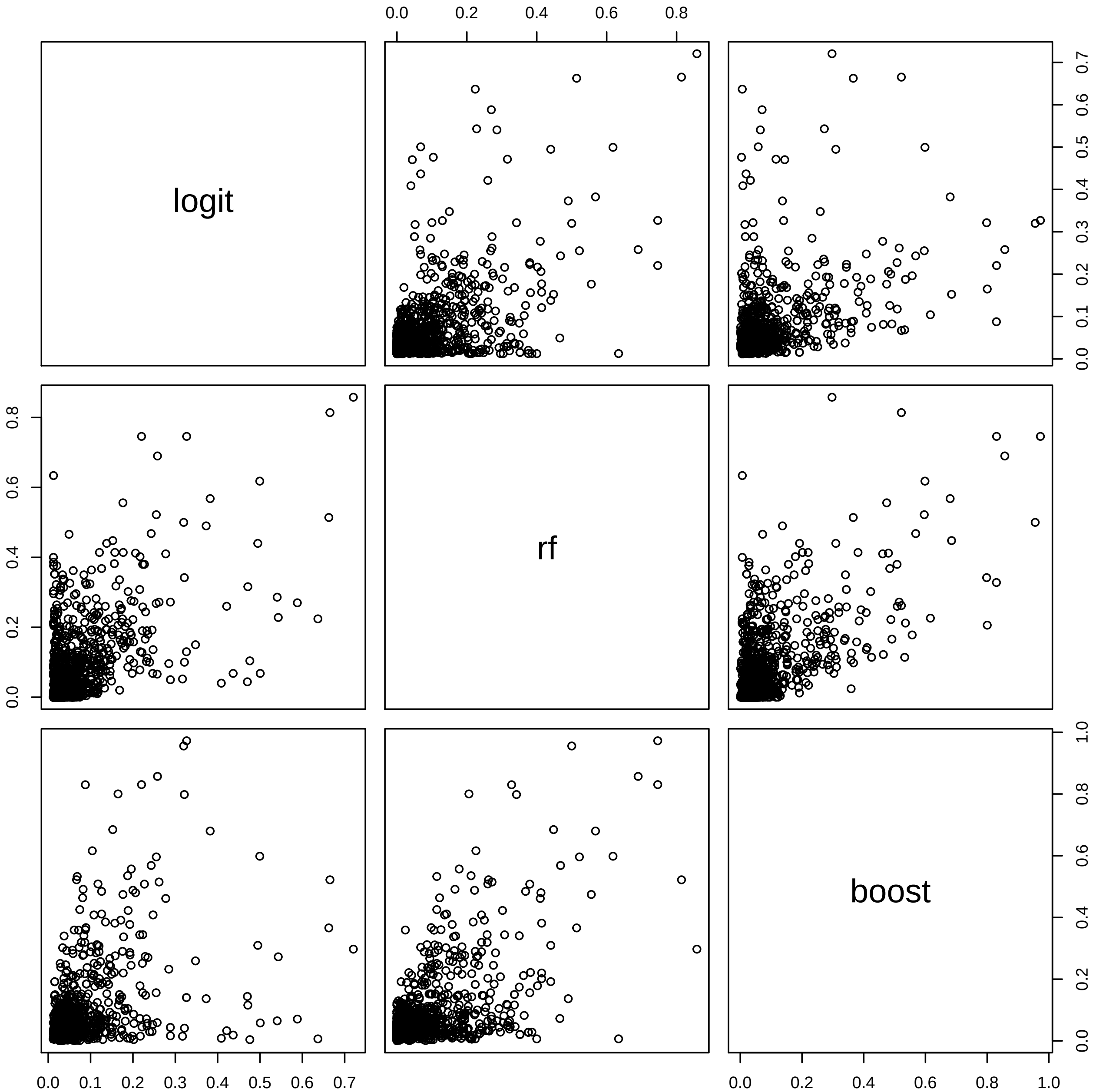


From each method class, we choose the one that has the lowest error on the validation set.

```
nmethod = length(phatL)
phatBest = matrix(0.0,nrow(td_test),nmethod) #pick off best from each method
colnames(phatBest) = names(phatL)
for(i in 1:nmethod) {
   nrun = ncol(phatL[[i]])
   lvec = rep(0,nrun)
   for(j in 1:nrun) lvec[j] = lossf(td_test$purchase,phatL[[i]][,j])
   imin = which.min(lvec)
   phatBest[,i] = phatL[[i]][,imin]
   phatBest[,i] = phatL[[i]][,1]
}
```

We can plot $\hat{p}$ for best models on the test set

```
pairs(phatBest)
```



The idea behind the tabloid example is that if we can predict who will buy we can target those customers and send them the tabloid.

To get an idea of how well our model is working, we can imagine choosing a customer from the data set to mail to first - did they buy?

We can look at the y value to see if they bought.

Whom would you mail to first?

You could mail the first 40 people in your database.

```
td$phat = phat
td[1:40, c("purchase", "phat")]
```

```
##    purchase   phat
## 1         0 0.0122
## 2         1 0.0670
## 3         0 0.0153
## 4         0 0.0129
## 5         0 0.0122
## 6         0 0.0429
## 7         0 0.0124
## 8         0 0.0122
## 9         0 0.0223
## 10        0 0.0122
## 11        0 0.0399
## 12        0 0.0122
## 13        0 0.0353
## 14        0 0.0163
## 15        0 0.0288
## 16        0 0.0125
## 17        0 0.0175
## 18        0 0.0122
## 19        0 0.0200
## 20        0 0.0122
## 21        0 0.0184
## 22        0 0.0203
## 23        0 0.0122
## 24        0 0.0122
## 25        0 0.0144
## 26        0 0.0122
## 27        0 0.0122
## 28        0 0.0131
## 29        0 0.0160
## 30        0 0.0122
## 31        0 0.0122
## 32        0 0.0122
## 33        0 0.0265
## 34        0 0.0122
## 35        0 0.0122
## 36        0 0.0274
## 37        0 0.0122
## 38        0 0.0123
## 39        0 0.0122
## 40        0 0.0136
```

Out of the first 40, there is only one purchase.

If you believe your model, you might mail to the household with the largest $\hat{p}$ (estimated prob of buying) first. Then you would mail to the household with the second largest $\hat{p}$ and so on.

```
td$phat = phat
sorted_phat = order(-phat)
td[sorted_phat[1:40], c("purchase", "phat")]
```

```
##      purchase  phat
## 2000        1 0.918
## 2755        1 0.792
## 8862        1 0.711
## 3628        1 0.692
## 1284        0 0.676
## 529         1 0.667
## 8086        0 0.654
## 2072        1 0.636
## 1435        1 0.632
## 4524        0 0.563
## 4626        0 0.544
## 978         0 0.529
## 9351        0 0.524
## 7040        0 0.517
## 7424        0 0.507
## 6545        1 0.499
## 5716        0 0.499
## 1218        0 0.497
## 374         0 0.493
## 521         0 0.480
## 1887        1 0.471
## 7703        0 0.450
## 789         1 0.446
## 8931        0 0.444
## 3853        1 0.436
## 5239        0 0.434
## 2999        0 0.434
## 6997        0 0.427
## 3526        1 0.414
## 8566        1 0.409
## 891         0 0.407
## 2417        0 0.407
## 5214        1 0.396
## 8490        0 0.386
## 6594        0 0.380
## 4548        0 0.378
## 6147        1 0.377
## 6548        0 0.376
## 1637        0 0.373
## 4748        1 0.370
```

You got 16 purchases out of the first 40 customers you targeted. Using only $40/10000 = 0.004$ of the data we got $16/258 = .062$ of the purchases!

5.3 Expected value of a classifier

Let us target everyone with $\hat{p} > 0.02$

Our **cost/benefit matrix** looks like this

```
cost_benefit = matrix(c(0,-0.8,0,39.20), nrow=2)
print(cost_benefit)
```

```
##      [,1] [,2]
## [1,]  0.0  0.0
## [2,] -0.8 39.2
```

Expected values of targeting is below:

```
confMat = getConfusionMatrix(td_test$purchase, phatBest[,1], 0.02)
print(confMat)
```

```
##            actual
## predictions    0    1
##   predict_0 3730   37
##   predict_1 1158   75
```

```
cat("Expected value of targeting using logistic regression = ",
    sum(sum(confMat * cost_benefit)), "\n")
```

```
## Expected value of targeting using logistic regression =  2014
```

```
confMat = getConfusionMatrix(td_test$purchase, phatBest[,2], 0.02)
print(confMat)
```

```
##            actual
## predictions    0    1
##   predict_0 4282   60
##   predict_1  606   52
```

```
cat("Expected value of targeting using random forests = ",
    sum(sum(confMat * cost_benefit)), "\n")
```

```
## Expected value of targeting using random forests =  1554
```

```
confMat = getConfusionMatrix(td_test$purchase, phatBest[,3], 0.02)
print(confMat)
```

```
##            actual
## predictions    0    1
##   predict_0 3785   42
##   predict_1 1103   70
```

```
cat("Expected value of targeting using boosting = ",
    sum(sum(confMat * cost_benefit)), "\n")
```

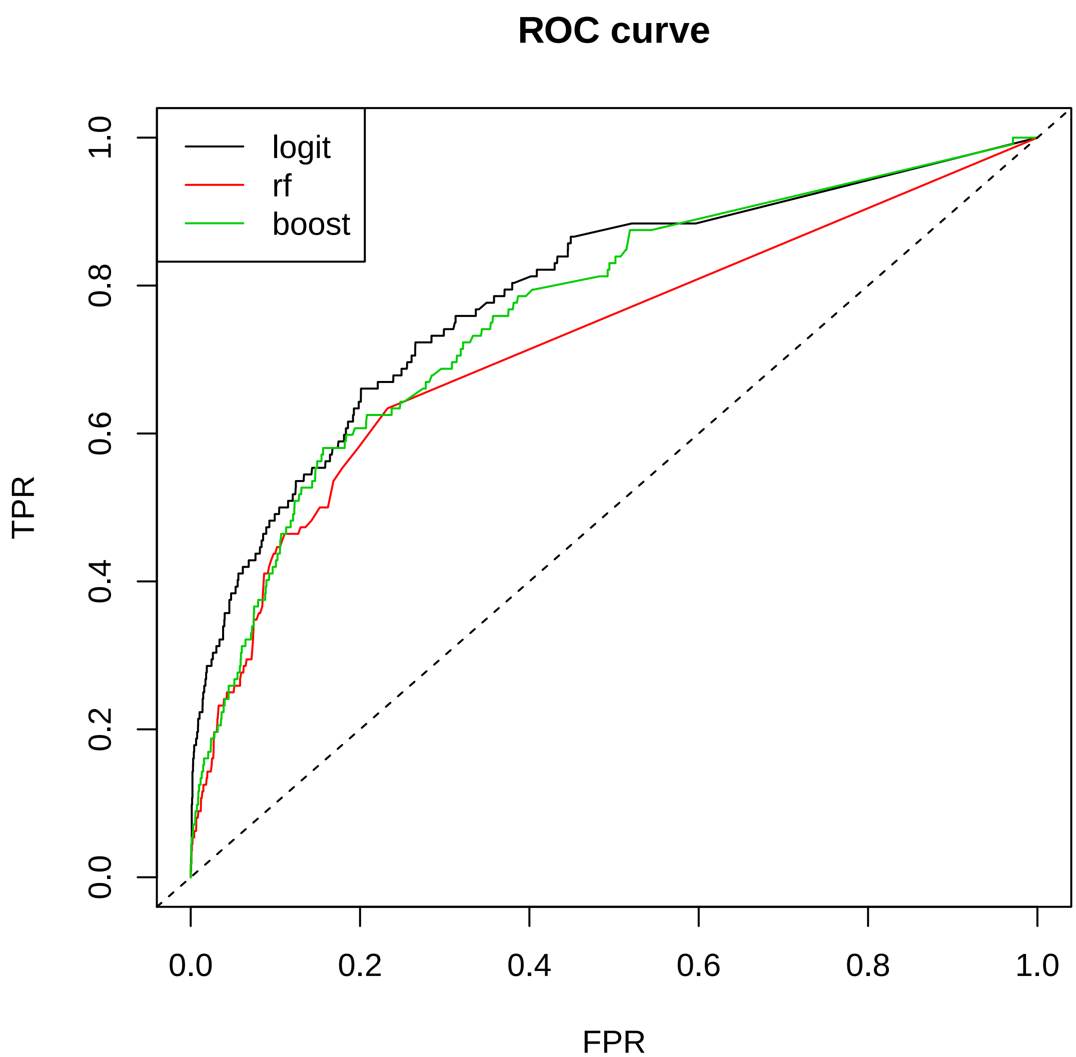
```
## Expected value of targeting using boosting =  1862
```

5.4 ROC curves

Library for plotting various summary curves

```
library(ROCR)
```

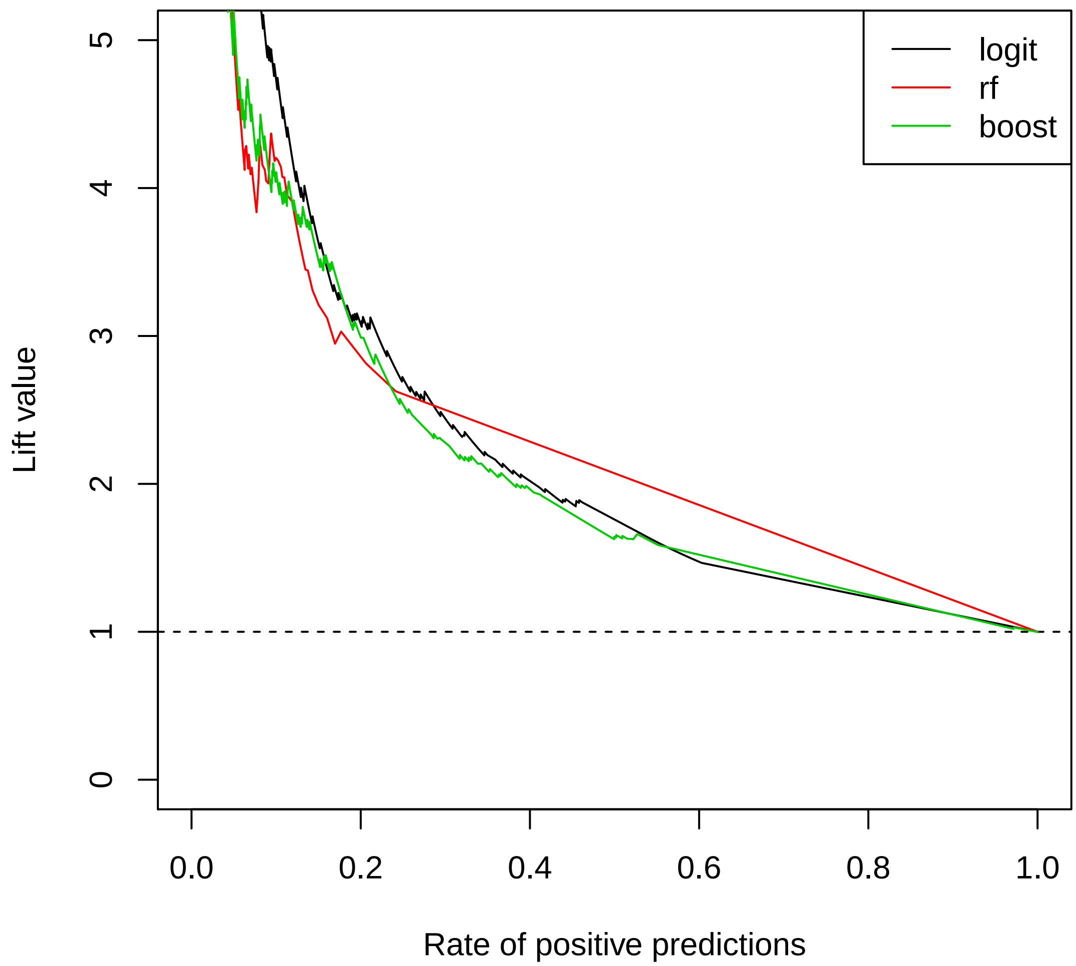
```
plot(c(0,1),c(0,1),xlab='FPR',ylab='TPR',main="ROC curve",cex.lab=1,type="n")
for(i in 1:ncol(phatBest)) {
   pred = prediction(phatBest[,i], td_test$purchase)
   perf = performance(pred, measure = "tpr", x.measure = "fpr")
   lines(perf@x.values[[1]], perf@y.values[[1]],col=i)
}
abline(0,1,lty=2)
legend("topleft",legend=names(phatL),col=1:nmethod,lty=rep(1,nmethod))
```



5.5 Lift curves

```
pred = prediction(phatBest[,1], td_test$purchase)
perf = performance(pred, measure = "lift", x.measure = "rpp")
plot(perf, col=1, ylim=c(0,5))
abline(h=1, lty=2)

for(i in 2:ncol(phatBest)) {
   pred = prediction(phatBest[,i], td_test$purchase)
   perf = performance(pred, measure = "lift", x.measure = "rpp")
   lines(perf@x.values[[1]], perf@y.values[[1]],col=i)
}
legend("topright",legend=names(phatL),col=1:nmethod,lty=rep(1,nmethod))
```



5.6 Cummulative response

```
pred = prediction(phatBest[,1], td_test$purchase)
perf = performance(pred, measure = "tpr", x.measure = "rpp")
plot(perf, col=1, ylim=c(0,1))
abline(h=1, lty=2)
abline(0,1,lty=2)
for(i in 2:ncol(phatBest)) {
   pred = prediction(phatBest[,i], td_test$purchase)
   perf = performance(pred, measure = "tpr", x.measure = "rpp")
   lines(perf@x.values[[1]], perf@y.values[[1]],col=i)
}
legend("bottomright",legend=names(phatL),col=1:nmethod,lty=rep(1,nmethod))
```