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

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

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


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

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

3 Summary statistics

```
summary(td)
```

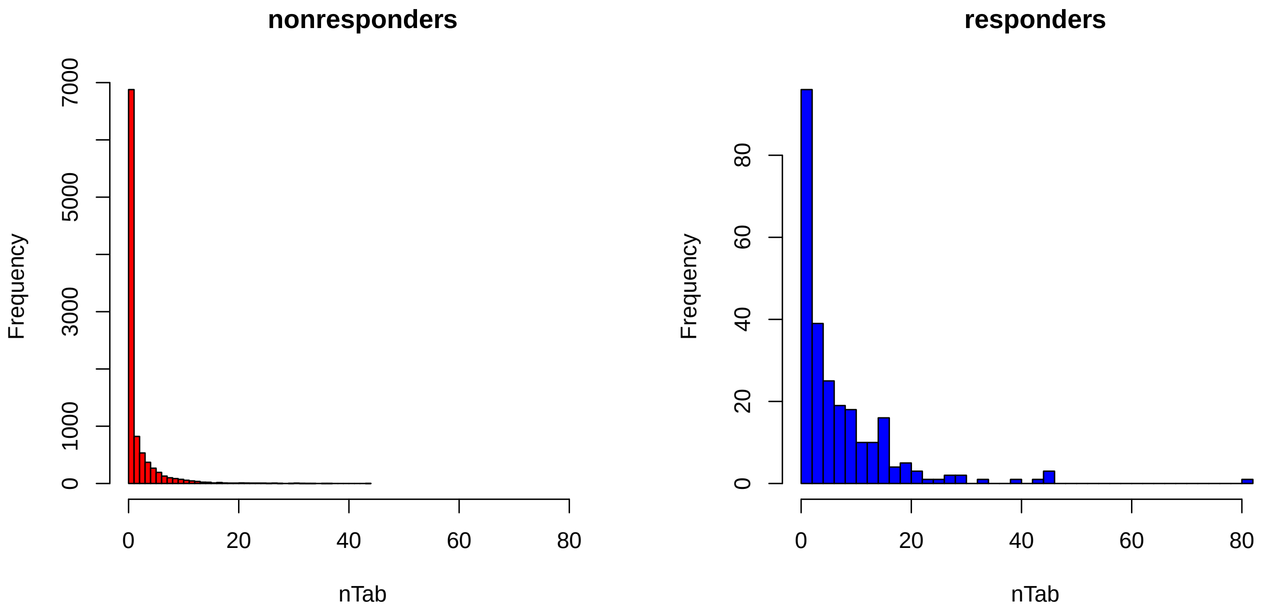
```
##  purchase       nTab          moCbook          iRecMer1          llDol
##  0:9742   Min.   : 0.0   Min.   : 1.2   Min.   :0.020   Min.   :-2.30
##  1: 258   1st Qu.: 0.0   1st Qu.:50.0   1st Qu.:0.020   1st Qu.:-2.30
##           Median : 0.0   Median :50.0   Median :0.020   Median :-2.30
## 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

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

$258/10000 = 0.0258$

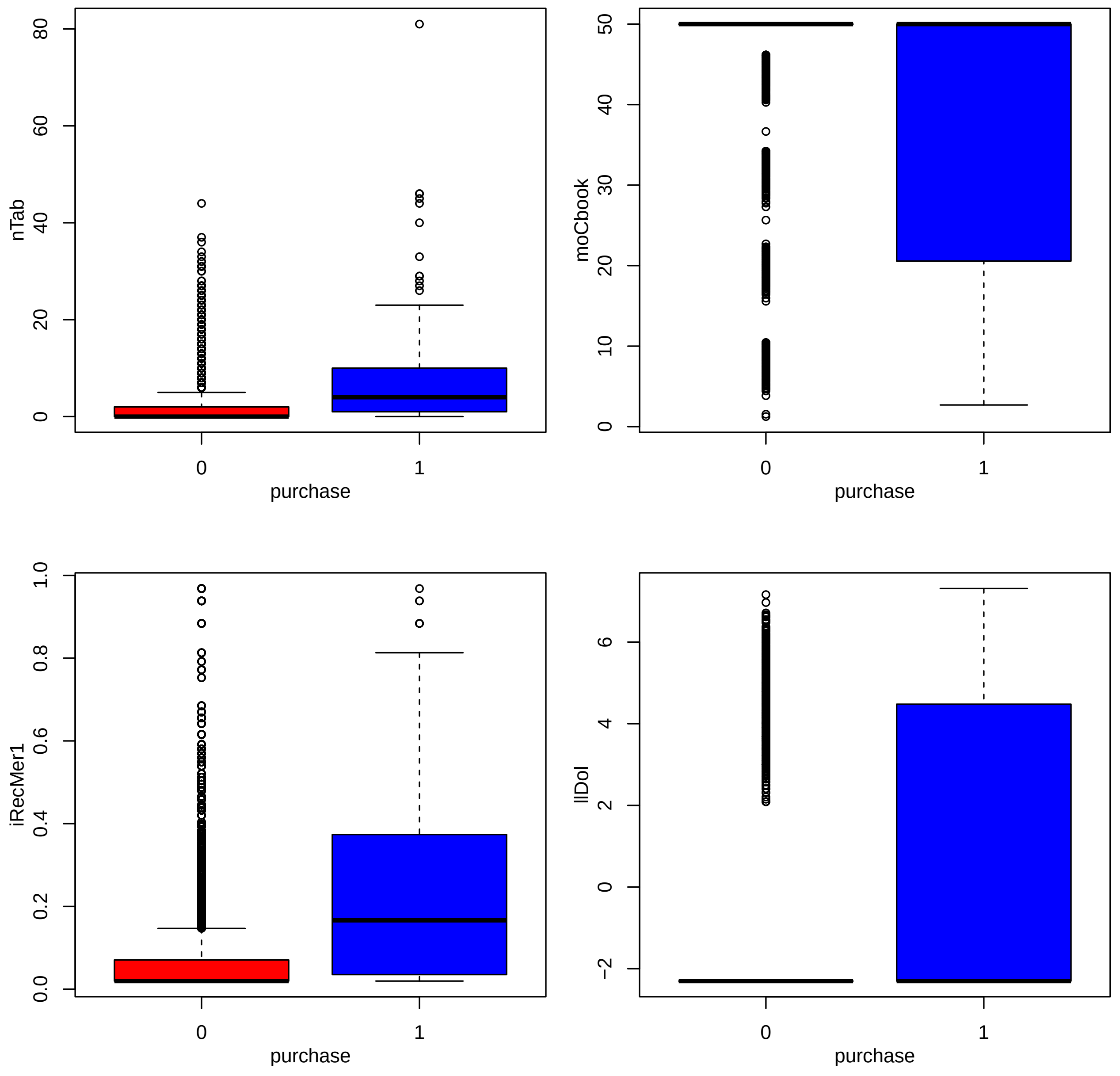
Illustration of how `nTab` is related to `responders`.

```
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

```
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(caret)
library(tree)
library(ranger)
library(xgboost)
```

I am creating a list to store results of all models.

```
phat.list = 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 Pr(>|z|)
## (Intercept) -2.62131    0.25667  -10.21  < 2e-16 ***
## nTab         0.05530    0.01209    4.57  4.8e-06 ***
## moCbook     -0.03249    0.00527   -6.17  7.0e-10 ***
## iRecMer1     1.72688    0.31282    5.52  3.4e-08 ***
## llDol        0.07842    0.02630    2.98   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. Note that we use `type="response"` which specifies the type of prediction used. For a default binomial model the default predictions are of log-odds (probabilities on logit scale). By using `type = "response"`, we obtain the predicted probabilities.

```
phat = predict(lgfit, td_validation, type="response")
phat.list$logit = matrix(phat,ncol=1)
```

4.2 Random Forest

We fit random forest models for a few different settings.

```r
p=ncol(td)-1

hyper_grid_rf <- expand.grid(
  mtry       = c(p, ceiling(sqrt(p))),
  node_size  = c(5, 10, 20)
)

# we will store phat values here
phat.list$rf = matrix(0.0, nrow(td_validation), nrow(hyper_grid_rf))

for(i in 1:nrow(hyper_grid_rf)) {
  # train model
  rf.model <- ranger(
    formula         = purchase~.,
    data            = td,
    num.trees       = 250,
    mtry            = hyper_grid_rf$mtry[i],
    min.node.size   = hyper_grid_rf$node_size[i],
    probability     = TRUE,
    seed            = 99
  )

   # predict for random forest returns
   # a matrix of class probabilities
   #    one column for each class and one row for each input
   # we want to record probability for class=1,
   # which is the second column of the output
   phat = predict(rf.model, data=td_validation)$predictions[,2]
   phat.list$rf[,i]=phat
}
```

4.3 Boosting

Remember that we need to put our data into a suitable form.

```
X.train = as.matrix( td[,-1] )
Y.train = as.numeric(td$purchase)-1
X.validation = as.matrix( td_validation[,-1] )
Y.validation = as.numeric(td_validation$purchase) - 1
```

We fit boosting models for a few different settings.

```
hyper_grid_xgb <- expand.grid(
  shrinkage = c(.01, .1),        ## controls the learning rate
  interaction.depth = c(1, 2, 4), ## tree depth
  nrounds = c(1000, 3000)         ## number of trees
)

# we will store phat values here
phat.list$boost = matrix(0.0,nrow(td_validation),nrow(hyper_grid_xgb))
```

Fitting

```
for(i in 1:nrow(hyper_grid_xgb)) {
  # create parameter list
  params <- list(
    eta = hyper_grid_xgb$shrinkage[i],
    max_depth = hyper_grid_xgb$interaction.depth[i]
  )

  # reproducibility
  set.seed(4776)

  # train model
  xgb.model <- xgboost(
    data      = X.train,
    label     = Y.train,
    params    = params,
    nrounds   = hyper_grid_xgb$nrounds[i],
    objective = "binary:logistic",     # for regression models
    verbose   = 0,                     # silent
    verbosity = 0                      # silent
  )

  phat = predict(xgb.model, newdata=X.validation)
  phat.list$boost[,i] = phat
}
```

5 Analysis of results

5.1 Misclassification rate

Let us first look at misclassification rate.

The following function computes the confusion matrix from the vector of true class labels `y` and a vector of estimated probabilities. The probabilities are converted into predicted class labels using a threshold `thr`.

```
# y should be 0/1
# phat are probabilities obtained by our algorithm
# thr is the cut off value - everything above thr is classified as 1
getConfusionMatrix = function(y,phat,thr=0.5) {
   yhat = as.factor( ifelse(phat > thr, 1, 0) )
   confusionMatrix(yhat, y)
}
```

This function computes the misclassification rate from the vector of true class labels `y` and a vector of estimated probabilities. The probabilities are converted into predicted class labels using a threshold `thr`.

```
# y should be 0/1
# phat are probabilities obtained by our algorithm
# thr is the cut off value - everything above thr is classified as 1
loss.misclassification.rate = function(y, phat, thr=0.5)
   1 - getConfusionMatrix(y, phat, thr)$overall[1]
```

For **logistic regression** we have:

```
cfm <- getConfusionMatrix(td_validation$purchase, phat.list$logit[,1], 0.5)
print(cfm, printStats = F)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 4884  108
##          1    4    4
```

```
cat('misclassification rate = ',
    loss.misclassification.rate(td_validation$purchase, phat.list[[1]][,1], 0.5),
    '\n')
```

```
## misclassification rate =  0.0224
```

For **random forest** we have:

```
nrun = nrow(hyper_grid_rf)
for(j in 1:nrun) {
  print(hyper_grid_rf[j,])
  cfm <- getConfusionMatrix(td_validation$purchase, phat.list[[2]][,j], 0.5)
  print(cfm, printStats = F)
  cat('misclassification rate = ',
      loss.misclassification.rate(td_validation$purchase, phat.list[[2]][,j], 0.5),
      '\n')
}
```

```
##   mtry node_size
## 1    4         5
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 4867  106
##          1   21    6
## misclassification rate =  0.0254
##   mtry node_size
## 2    2         5
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 4878  107
##          1   10    5
## misclassification rate =  0.0234
##   mtry node_size
## 3    4        10
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 4879  107
##          1    9    5
## misclassification rate =  0.0232
##   mtry node_size
## 4    2        10
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 4882  108
##          1    6    4
## misclassification rate =  0.0228
##   mtry node_size
## 5    4        20
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 4883  108
##          1    5    4
## misclassification rate =  0.0226
##   mtry node_size
## 6    2        20
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 4884  108
##          1    4    4
## misclassification rate =  0.0224
```

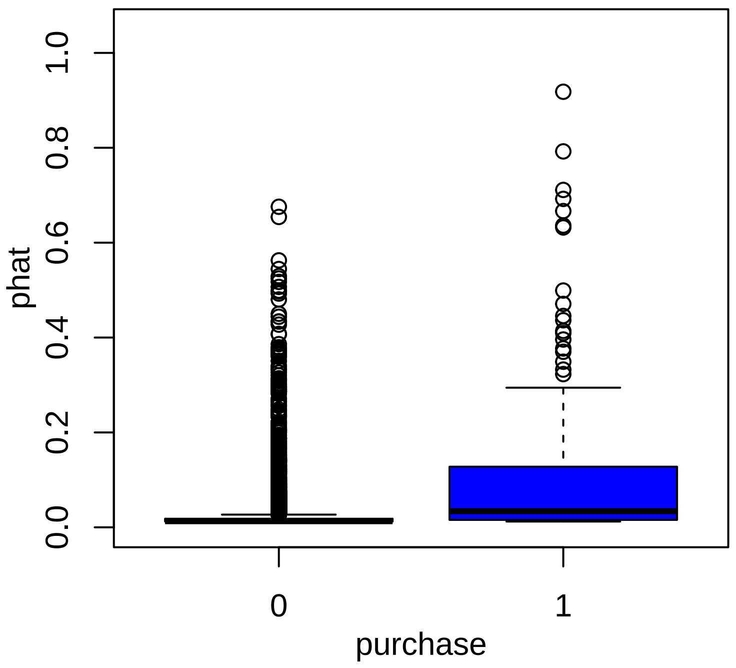
For **boosting** we have:

```r
nrun = nrow(hyper_grid_xgb)
for(j in 1:nrun) {
  print(hyper_grid_xgb[j,])
  cfm <- getConfusionMatrix(td_validation$purchase, phat.list[[3]][,j], 0.5)
  print(cfm, printStats = F)
  cat('misclassification rate = ',
      loss.misclassification.rate(td_validation$purchase, phat.list[[3]][,j], 0.5),
      '\n')
}
```

```
##   shrinkage interaction.depth nrounds
## 1      0.01                 1    1000
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 4885  111
##          1    3    1
## misclassification rate =  0.0228
##   shrinkage interaction.depth nrounds
## 2       0.1                 1    1000
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 4883  111
##          1    5    1
## misclassification rate =  0.0232
##   shrinkage interaction.depth nrounds
## 3      0.01                 2    1000
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 4885  111
##          1    3    1
## misclassification rate =  0.0228
##   shrinkage interaction.depth nrounds
## 4       0.1                 2    1000
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 4871  106
##          1   17    6
## misclassification rate =  0.0246
##   shrinkage interaction.depth nrounds
## 5      0.01                 4    1000
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 4881  109
##          1    7    3
## misclassification rate =  0.0232
##   shrinkage interaction.depth nrounds
## 6       0.1                 4    1000
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 4859  105
##          1   29    7
## misclassification rate =  0.0268
##   shrinkage interaction.depth nrounds
## 7      0.01                 1    3000
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 4885  111
##          1    3    1
## misclassification rate =  0.0228
##   shrinkage interaction.depth nrounds
## 8       0.1                 1    3000
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 4880  108
##          1    8    4
## misclassification rate =  0.0232
##   shrinkage interaction.depth nrounds
## 9      0.01                 2    3000
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 4879  107
##          1    9    5
## misclassification rate =  0.0232
##    shrinkage interaction.depth nrounds
## 10       0.1                 2    3000
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 4861  103
##          1   27    9
## misclassification rate =  0.026
##    shrinkage interaction.depth nrounds
## 11      0.01                 4    3000
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 4873  107
##          1   15    5
## misclassification rate =  0.0244
##    shrinkage interaction.depth nrounds
## 12       0.1                 4    3000
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 4844  103
##          1   44    9
## misclassification rate =  0.0294
```

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)
```



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.

```r
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!

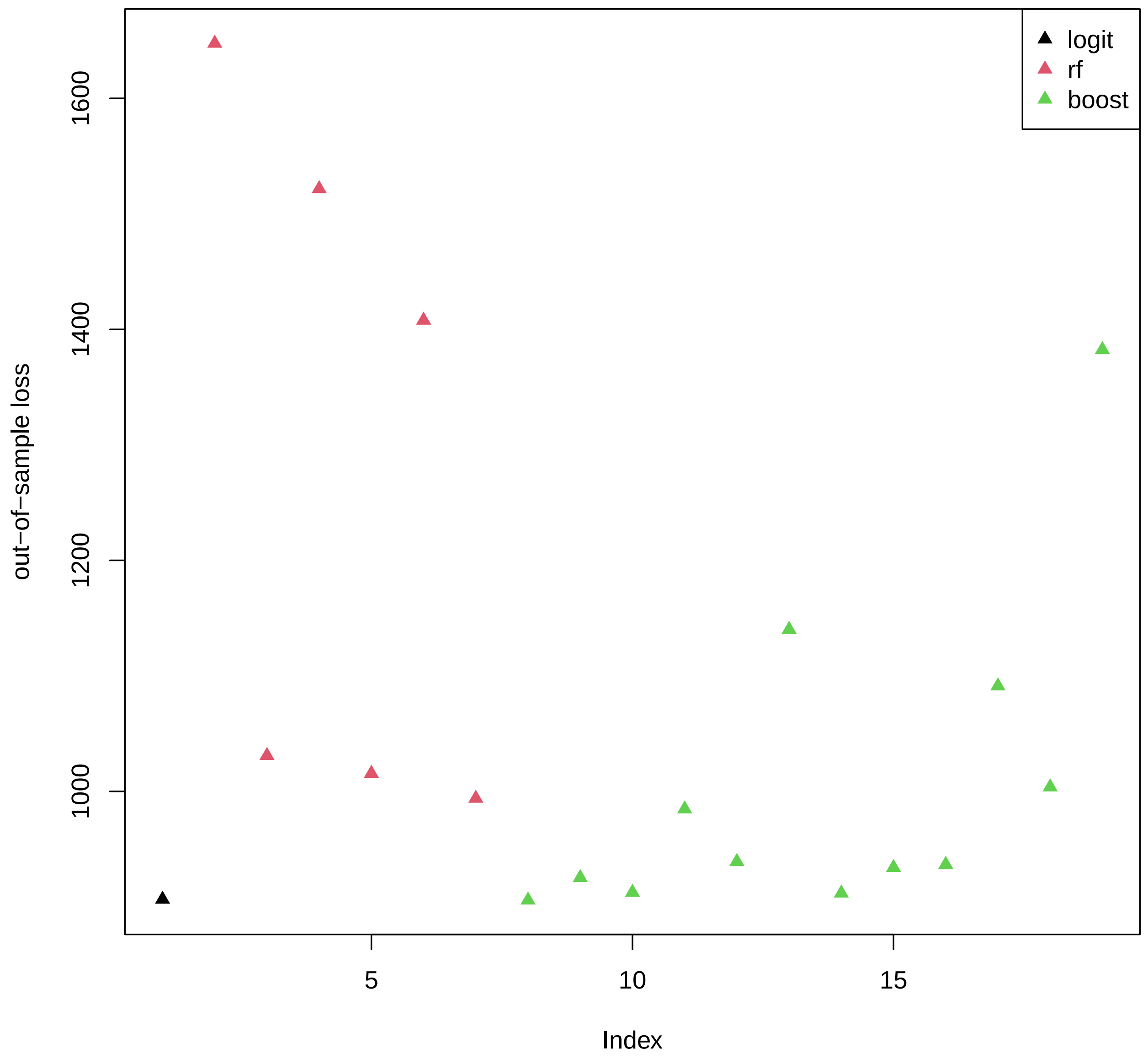
5.2 Deviance

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

```r
# deviance loss function
# y should be 0/1
# phat are probabilities obtained by our algorithm
# wht shrinks probabilities 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)))
}
```

Plot validation set loss — deviance:

```r
lossL = list()
nmethod = length(phat.list)
for(i in 1:nmethod) {
   nrun = ncol(phat.list[[i]])
   lvec = rep(0,nrun)
   for(j in 1:nrun) lvec[j] = lossf(td_validation$purchase, phat.list[[i]][,j])
   lossL[[i]]=lvec; names(lossL)[i] = names(phat.list)[i]
}
lossv = unlist(lossL)
plot(lossv, ylab="out-of-sample loss", type="n")
nloss=0
for(i in 1:nmethod) {
   ii = nloss + 1:ncol(phat.list[[i]])
   points(ii,lossv[ii],col=i,pch=17)
   nloss = nloss + ncol(phat.list[[i]])
}
legend("topright",legend=names(phat.list),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(phat.list)
phatBest = matrix(0.0,nrow(td_validation),nmethod) #pick off best from each method
colnames(phatBest) = names(phat.list)
for(i in 1:nmethod) {
   nrun = ncol(phat.list[[i]])
   lvec = rep(0,nrun)
   for(j in 1:nrun) lvec[j] = lossf(td_validation$purchase,phat.list[[i]][,j])
   imin = which.min(lvec)
   phatBest[,i] = phat.list[[i]][,imin]
}
```

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_validation$purchase, phatBest[,1], 0.02)
print(confMat, printStats = F)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 3730   37
##          1 1158   75
```

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

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

```
confMat = getConfusionMatrix(td_validation$purchase, phatBest[,2], 0.02)
print(confMat, printStats = F)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 3753   44
##          1 1135   68
```

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

```
## Expected value of targeting using random forests =  1758
```

```
confMat = getConfusionMatrix(td_validation$purchase, phatBest[,3], 0.02)
print(confMat, printStats = F)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 3359   28
##          1 1529   84
```

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

```
## Expected value of targeting using boosting =  2070
```

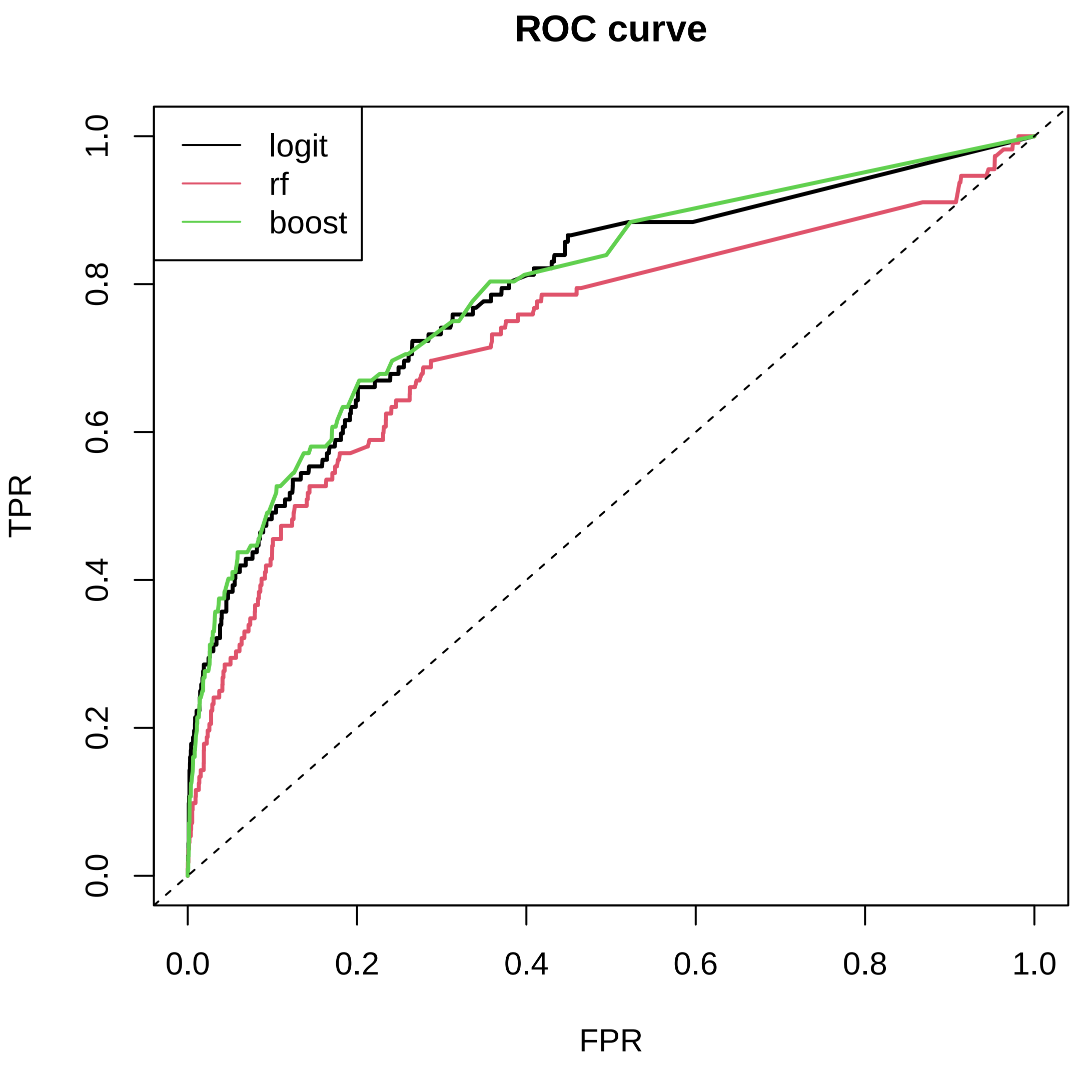
5.4 ROC curves

Library for plotting various summary curves

```
library(ROCR)
```

```
for(i in 1:ncol(phatBest)) {
   pred = prediction(phatBest[,i], td_validation$purchase)
   perf = performance(pred, measure = "tpr", x.measure = "fpr")

   if (i == 1) {
     plot(perf, col = 1, lwd = 2,
          main= 'ROC curve', xlab='FPR', ylab='TPR', cex.lab=1)
   } else {
     plot(perf, add = T, col = i, lwd = 2)
   }
}
abline(0,1,lty=2)
legend("topleft",legend=names(phat.list),col=1:nmethod,lty=rep(1,nmethod))
```



We can also compute AUC (area under the ROC curve).

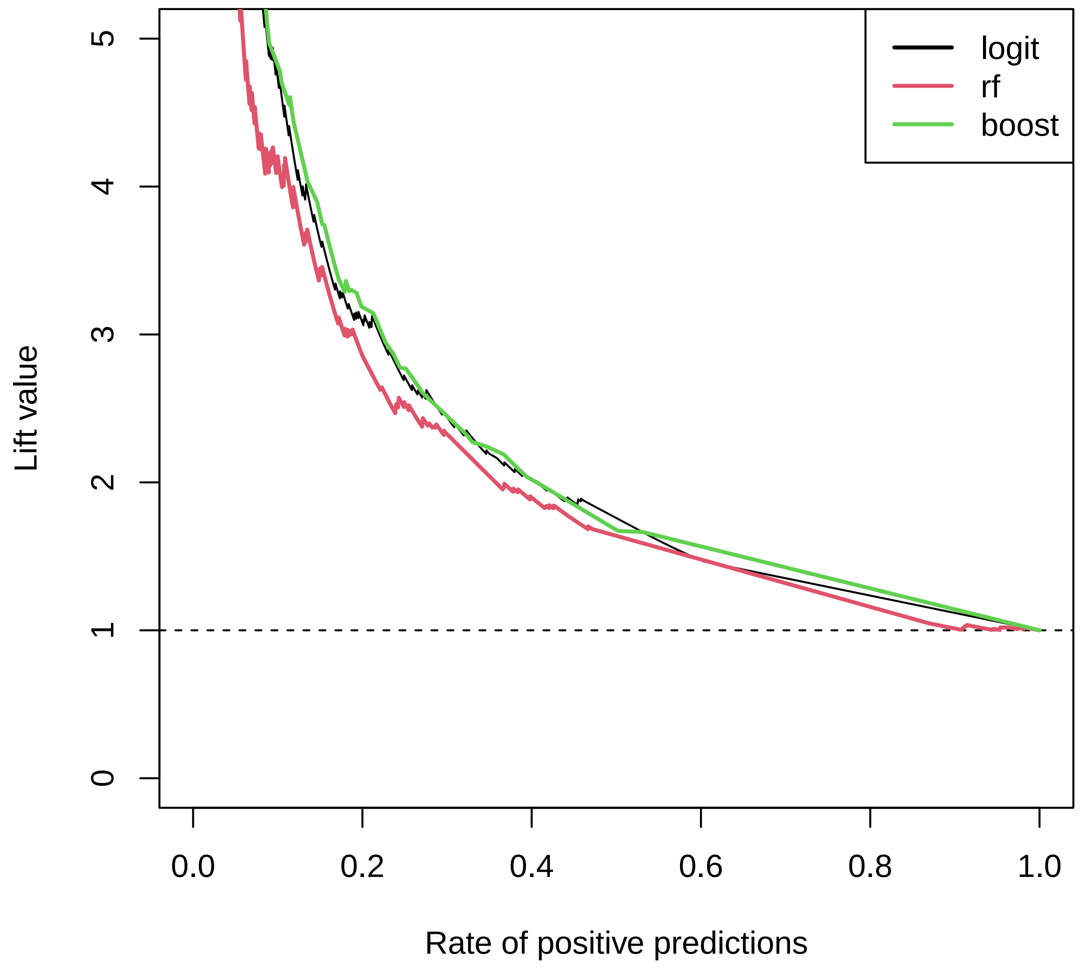
```r
for(i in 1:ncol(phatBest)) {
  pred = prediction(phatBest[,i], td_validation$purchase)
  perf <- performance(pred, measure = "auc")
  print(paste0("AUC ", names(phat.list)[i], " :: ", perf@y.values[[1]]))
}
```

```
## [1] "AUC logit :: 0.788627761865789"
## [1] "AUC rf :: 0.734814304711245"
## [1] "AUC boost :: 0.794628244096329"
```

5.5 Lift curves

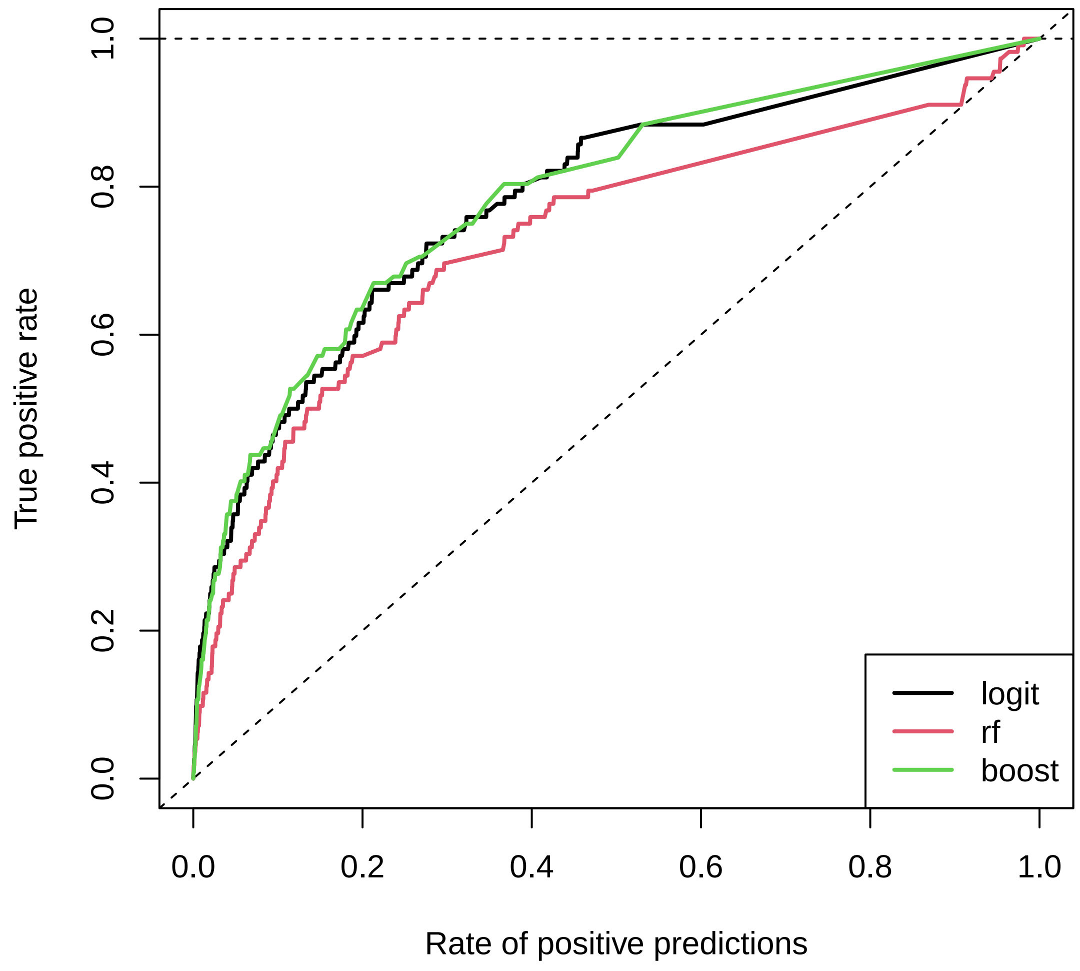
```
pred = prediction(phatBest[,1], td_validation$purchase)
perf = performance(pred, measure = "lift", x.measure = "rpp", lwd=2)
plot(perf, col=1, ylim=c(0,5))
abline(h=1, lty=2)

for(i in 2:ncol(phatBest)) {
   pred = prediction(phatBest[,i], td_validation$purchase)
   perf = performance(pred, measure = "lift", x.measure = "rpp")
   plot(perf, add = T, col = i, lwd = 2)
}
legend("topright",legend=names(phat.list),col=1:nmethod,lty=rep(1,nmethod), lwd=2)
```



5.6 Cummulative response

```
pred = prediction(phatBest[,1], td_validation$purchase)
perf = performance(pred, measure = "tpr", x.measure = "rpp")
plot(perf, col=1, ylim=c(0,1),lwd=2)
abline(h=1, lty=2)
abline(0,1,lty=2)
for(i in 2:ncol(phatBest)) {
   pred = prediction(phatBest[,i], td_validation$purchase)
   perf = performance(pred, measure = "tpr", x.measure = "rpp")
   plot(perf, add = T, col = i, lwd = 2)
}
legend("bottomright",legend=names(phat.list),col=1:nmethod,lty=rep(1,nmethod),lwd=2)
```


True positive rate
Rate of positive predictions
logit
rf
boost