# Give Me Some Credit Data Set from a Kaggle competition

## 1 Description

This is a kaggle competition data set.

There are 150,000 observations in the kaggle training data.

The Y is: "Person experienced 90 days past due delinquency or worse: Y/N"

Can you predict when an account is going to be delinquent!

Data can be obtained from here: https://www.kaggle.com/c/GiveMeSomeCredit

## 2 Preprocessing

We download the data and preprocess it first. We split the kaggle training data into a 50% train and 50% test. The kaggle test does not come with y! We made y=1 if delinquent and 0 else.

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

trainDf = read.csv("CreditScoring.csv")
trainDf = trainDf[,-1]
```

Add `y` as a factor and get rid of old `y = SeriousDlqin2yrs`.

```
trainDf$y = as.factor(trainDf$SeriousDlqin2yrs)
trainDf = trainDf[,-1]
```

We don't want to deal with NA's, so we drop `NumberOfDependents` and `MonthlyIncome`

```
trainDf=trainDf[,-10]
trainDf=trainDf[,-5]
```

Split data into train and validation.

```
set.seed(99)
n=nrow(trainDf)
ii = sample(1:n,n)
nvalid = floor(n/2)
td = trainDf[ii[1:nvalid],]
td_validation = trainDf[ii[(nvalid+1):n],]
```

# 3 Summary statistics

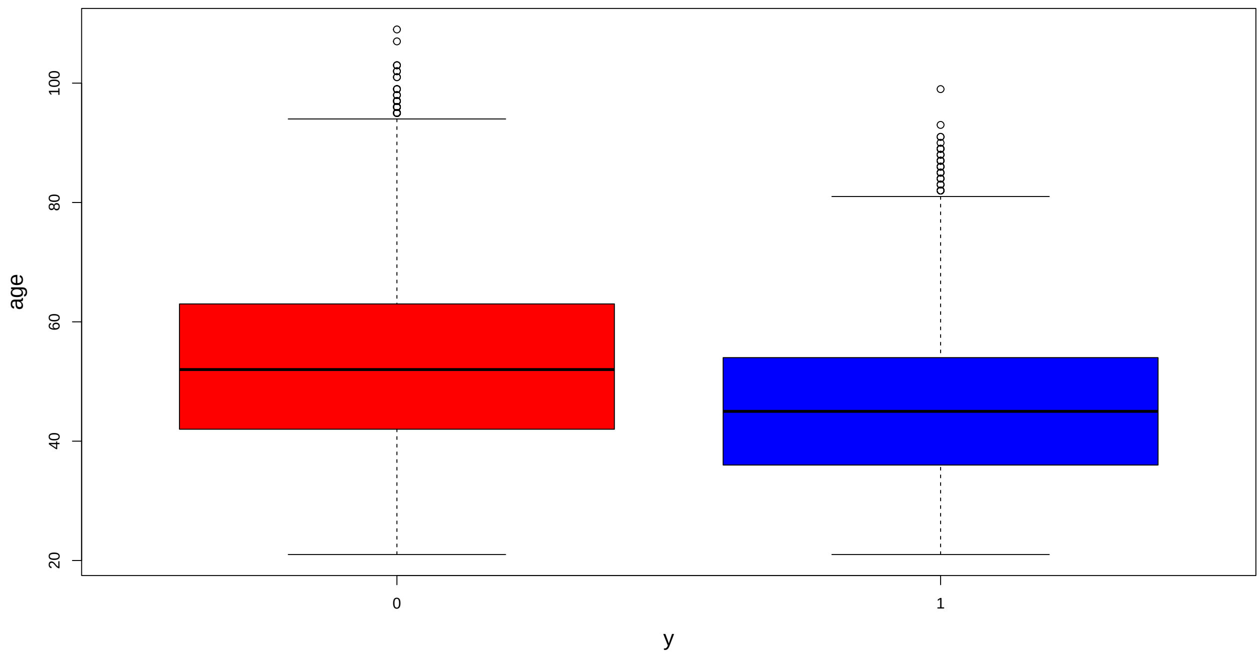
```
table(trainDf$y)
```

```
##
##      0      1
## 139974  10026
```

6 to 7 % of accounts are delinquent.

For example, it looks like older people are less likely to be delinquent.

```
plot(age~y,td,col=c("red","blue"),cex.lab=1.4)
```



# 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(y~., td, family=binomial)
print(summary(lgfit))
```

```
##
## Call:
## glm(formula = y ~ ., family = binomial, data = td)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -3.224  -0.389  -0.318  -0.259   4.927
##
## Coefficients:
##                                       Estimate
## (Intercept)                          -1.29e+00
## RevolvingUtilizationOfUnsecuredLines -1.30e-04
## age                                  -3.06e-02
## NumberOfTime30.59DaysPastDueNotWorse  5.15e-01
## DebtRatio                            -1.98e-05
## NumberOfOpenCreditLinesAndLoans      -1.38e-02
## NumberOfTimes90DaysLate               4.69e-01
## NumberRealEstateLoansOrLines          5.67e-02
## NumberOfTime60.89DaysPastDueNotWorse -9.52e-01
##                                      Std. Error
## (Intercept)                            5.61e-02
## RevolvingUtilizationOfUnsecuredLines   1.26e-04
## age                                    1.15e-03
## NumberOfTime30.59DaysPastDueNotWorse   1.56e-02
## DebtRatio                              1.46e-05
## NumberOfOpenCreditLinesAndLoans        3.56e-03
## NumberOfTimes90DaysLate                2.13e-02
## NumberRealEstateLoansOrLines           1.37e-02
## NumberOfTime60.89DaysPastDueNotWorse   2.48e-02
##                                      z value
## (Intercept)                           -22.92
```

```
## RevolvingUtilizationOfUnsecuredLines   -1.03
## age                                  -26.60
## NumberOfTime30.59DaysPastDueNotWorse   32.95
## DebtRatio                             -1.36
## NumberOfOpenCreditLinesAndLoans       -3.89
## NumberOfTimes90DaysLate               22.04
## NumberRealEstateLoansOrLines           4.13
## NumberOfTime60.89DaysPastDueNotWorse  -38.37
##                                      Pr(>|z|)
## (Intercept)                           < 2e-16
## RevolvingUtilizationOfUnsecuredLines     0.30
## age                                   < 2e-16
## NumberOfTime30.59DaysPastDueNotWorse  < 2e-16
## DebtRatio                                0.17
## NumberOfOpenCreditLinesAndLoans       9.9e-05
## NumberOfTimes90DaysLate               < 2e-16
## NumberRealEstateLoansOrLines          3.7e-05
## NumberOfTime60.89DaysPastDueNotWorse  < 2e-16
##
## (Intercept)                          ***
## RevolvingUtilizationOfUnsecuredLines
## age                                  ***
## NumberOfTime30.59DaysPastDueNotWorse ***
## DebtRatio
## NumberOfOpenCreditLinesAndLoans      ***
## NumberOfTimes90DaysLate              ***
## NumberRealEstateLoansOrLines         ***
## NumberOfTime60.89DaysPastDueNotWorse ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 36977  on 74999  degrees of freedom
## Residual deviance: 33960  on 74991  degrees of freedom
## AIC: 33978
##
## Number of Fisher Scoring iterations: 6
```

Predictions are stored for later analysis

```
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.

```
p=ncol(trainDf)-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         = y~.,
    data            = td,
    num.trees       = 1000,
    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
}
```

```
## Growing trees.. Progress: 25%. Estimated remaining time: 1 minute, 34 seconds.
## Growing trees.. Progress: 52%. Estimated remaining time: 57 seconds.
## Growing trees.. Progress: 79%. Estimated remaining time: 24 seconds.
## Growing trees.. Progress: 77%. Estimated remaining time: 9 seconds.
## Growing trees.. Progress: 29%. Estimated remaining time: 1 minute, 16 seconds.
## Growing trees.. Progress: 58%. Estimated remaining time: 44 seconds.
## Growing trees.. Progress: 88%. Estimated remaining time: 13 seconds.
## Growing trees.. Progress: 69%. Estimated remaining time: 14 seconds.
## Growing trees.. Progress: 36%. Estimated remaining time: 54 seconds.
## Growing trees.. Progress: 73%. Estimated remaining time: 22 seconds.
## Growing trees.. Progress: 92%. Estimated remaining time: 2 seconds.
```

### 4.3 Boosting

We fit boosting models for a few different settings. Remember that we need to put our data into a suitable form.

```r
X = Matrix::sparse.model.matrix(y ~ ., data = trainDf)[,-1]
X.train = X[ii[1:nvalid],]
Y.train = as.numeric(td$y)-1
X.validation = X[ii[(nvalid+1):n],]
Y.validation = as.numeric(td_validation$y) - 1

hyper_grid_xgb <- expand.grid(
  shrinkage = c(.01, .1),        ## controls the learning rate
  interaction.depth = c(1, 2, 4), ## tree depth
  nrounds = c(1000, 5000)         ## number of trees
)

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

Fitting

```r
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 Miss-classification rate

Let us first look at miss-classification 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$y, phat.list$logit[,1], 0.5)
print(cfm, printStats = F)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 69859  4791
##          1   160   190
```

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

```
## misclassification rate =  0.066
```

For **random forest** we have:

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

```
##   mtry node_size
## 1    8         5
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 69058  3978
##          1   961  1003
## misclassification rate =  0.0659
##   mtry node_size
## 2    3         5
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 69258  4030
##          1   761   951
## misclassification rate =  0.0639
##   mtry node_size
## 3    8        10
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 69140  3979
##          1   879  1002
## misclassification rate =  0.0648
##   mtry node_size
## 4    3        10
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 69294  4037
##          1   725   944
## misclassification rate =  0.0635
##   mtry node_size
## 5    8        20
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 69187  4001
```

```
##          1   832   980
## misclassification rate =  0.0644
##   mtry node_size
## 6    3        20
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 69353  4077
##          1   666   904
## misclassification rate =  0.0632
```

For **boosting** we have:

```
nrun = nrow(hyper_grid_xgb)
for(j in 1:nrun) {
  print(hyper_grid_xgb[j,])
  cfm <- getConfusionMatrix(td_validation$y, phat.list[[3]][,j], 0.5)
  print(cfm, printStats = F)
  cat('misclassification rate = ',
      loss.misclassification.rate(td_validation$y, 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 69514  4243
##          1   505   738
## misclassification rate =  0.0633
##   shrinkage interaction.depth nrounds
## 2       0.1                 1    1000
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 69340  4029
##          1   679   952
## misclassification rate =  0.0628
##   shrinkage interaction.depth nrounds
## 3      0.01                 2    1000
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 69467  4141
##          1   552   840
## misclassification rate =  0.0626
##   shrinkage interaction.depth nrounds
## 4       0.1                 2    1000
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 69356  4025
##          1   663   956
## misclassification rate =  0.0625
##   shrinkage interaction.depth nrounds
## 5      0.01                 4    1000
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 69445  4086
```

```
##          1   574   895
## misclassification rate =  0.0621
##   shrinkage interaction.depth nrounds
## 6       0.1                 4    1000
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 69282  4047
##          1   737   934
## misclassification rate =  0.0638
##   shrinkage interaction.depth nrounds
## 7      0.01                 1    5000
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 69361  4037
##          1   658   944
## misclassification rate =  0.0626
##   shrinkage interaction.depth nrounds
## 8       0.1                 1    5000
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 69319  4016
##          1   700   965
## misclassification rate =  0.0629
##   shrinkage interaction.depth nrounds
## 9      0.01                 2    5000
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 69363  4037
##          1   656   944
## misclassification rate =  0.0626
##    shrinkage interaction.depth nrounds
## 10       0.1                 2    5000
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 69216  3998
##          1   803   983
## misclassification rate =  0.064
##    shrinkage interaction.depth nrounds
## 11      0.01                 4    5000
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 69335  4054
```

```
##          1   684   927
## misclassification rate =  0.0632
##    shrinkage interaction.depth nrounds
## 12       0.1                 4    5000
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 68971  3976
##          1  1048  1005
## misclassification rate =  0.067
```

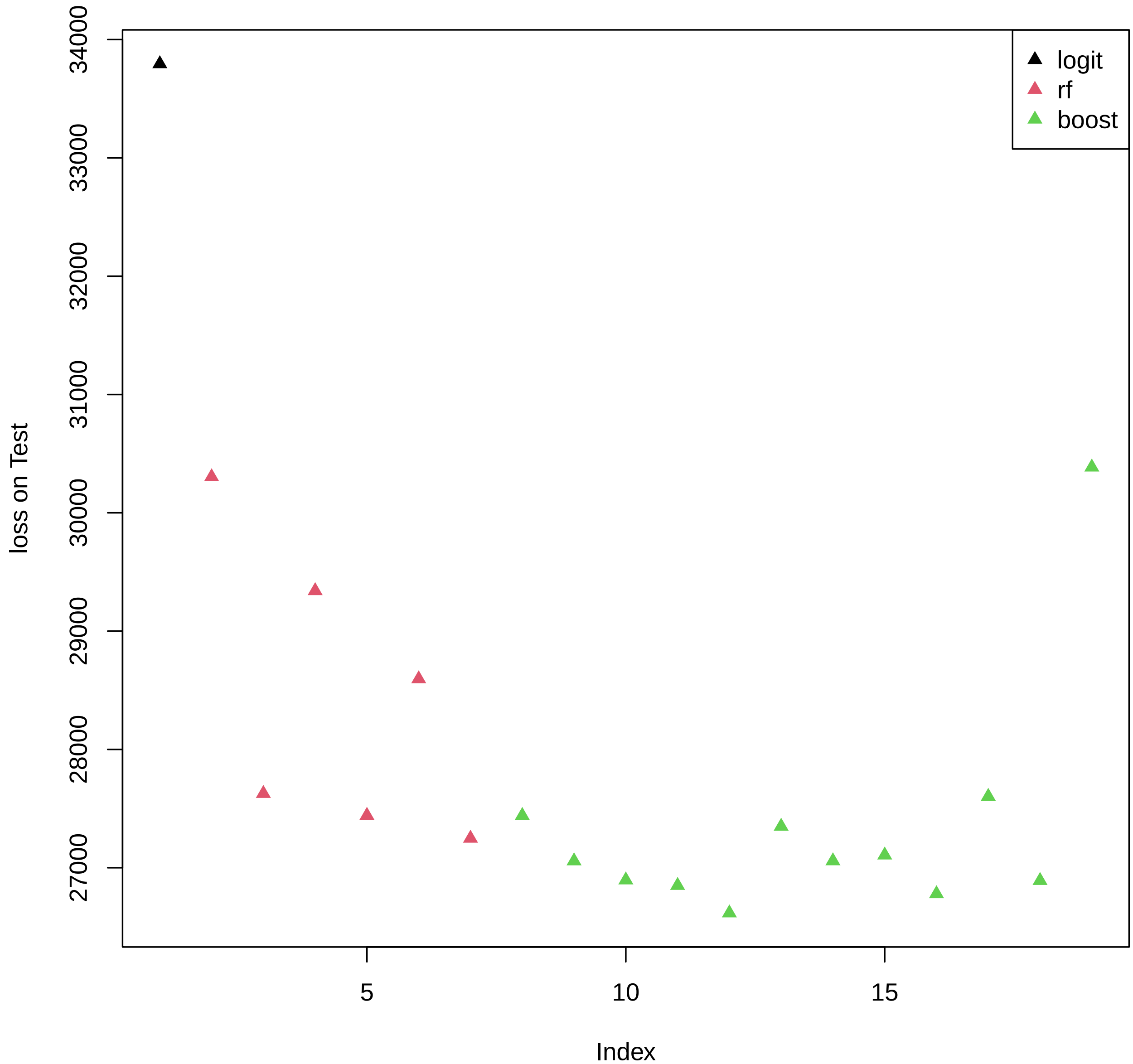
## 5.2 Deviance

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

```
# 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 test set loss — deviance:

```
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$y, phat.list[[i]][,j])
   lossL[[i]]=lvec; names(lossL)[i] = names(phat.list)[i]
}
lossv = unlist(lossL)
plot(lossv, ylab="loss on Test", 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.

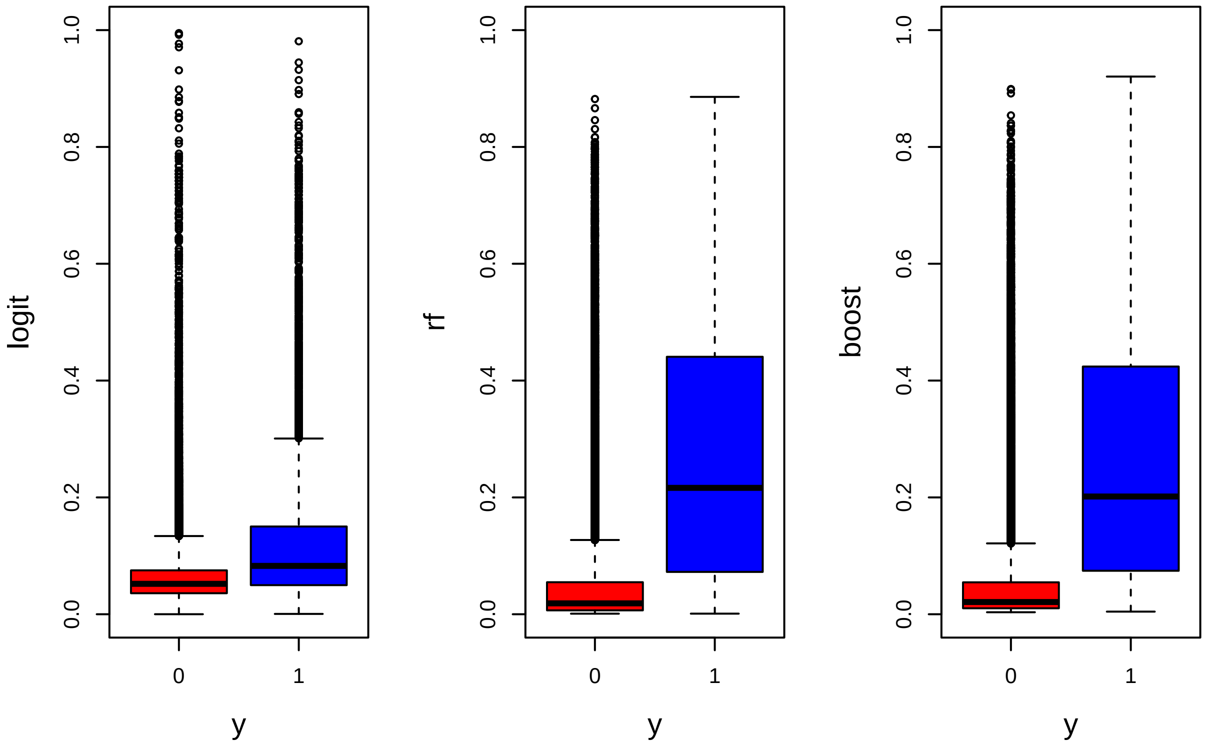
```
nmethod = length(phat.list)
phatBest = matrix(0.0,nrow(td_validation),nmethod) #pick off best from each method
colnames(phatBest) = names(phat.list)
for(i in 1:nmethod) {
   nrun = ncol(phat.list[[i]])
   lvec = rep(0,nrun)
   for(j in 1:nrun) lvec[j] = lossf(td_validation$y,phat.list[[i]][,j])
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

Each plot relates $\hat{p}$ to $y$.

Going from left to right, $\hat{p}$ is from logit, random forests, and boosting.

```
colnames(phatBest) = c("logit", "rf", "boost")
tempdf = data.frame(phatBest,y = td_validation$y)

par(mfrow=c(1,3))
plot(logit~y,tempdf,ylim=c(0,1),cex.lab=1.4,col=c("red","blue"))
plot(rf~y,tempdf,ylim=c(0,1),cex.lab=1.4,col=c("red","blue"))
plot(boost~y,tempdf,ylim=c(0,1),cex.lab=1.4,col=c("red","blue"))
```




Boosting and random forests both look **pretty good**!
Both are **dramatically better than logit**!

## 5.3 Expected value of a classifier

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

```
cost_benefit = matrix(c(0,-0.25,0,1), nrow=2)
print(cost_benefit)
```

```
##       [,1] [,2]
## [1,]  0.00    0
## [2,] -0.25    1
```

If $\hat{p} > 0.2$, we extend credit.

Expected values of classifiers is below:

```
confMat = getConfusionMatrix(td_validation$y, phatBest[,1], 0.2)
print(confMat, printStats = F)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 68886  4134
##          1  1133   847
```

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

```
## Expected value of targeting using logistic regression =  564
```

```
confMat = getConfusionMatrix(td_validation$y, phatBest[,2], 0.2)
print(confMat, printStats = F)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 65320  2380
##          1  4699  2601
```

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

```
## Expected value of targeting using random forests =  1426
```

```
confMat = getConfusionMatrix(td_validation$y, phatBest[,3], 0.2)
print(confMat, printStats = F)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 66086  2480
##          1  3933  2501
```

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

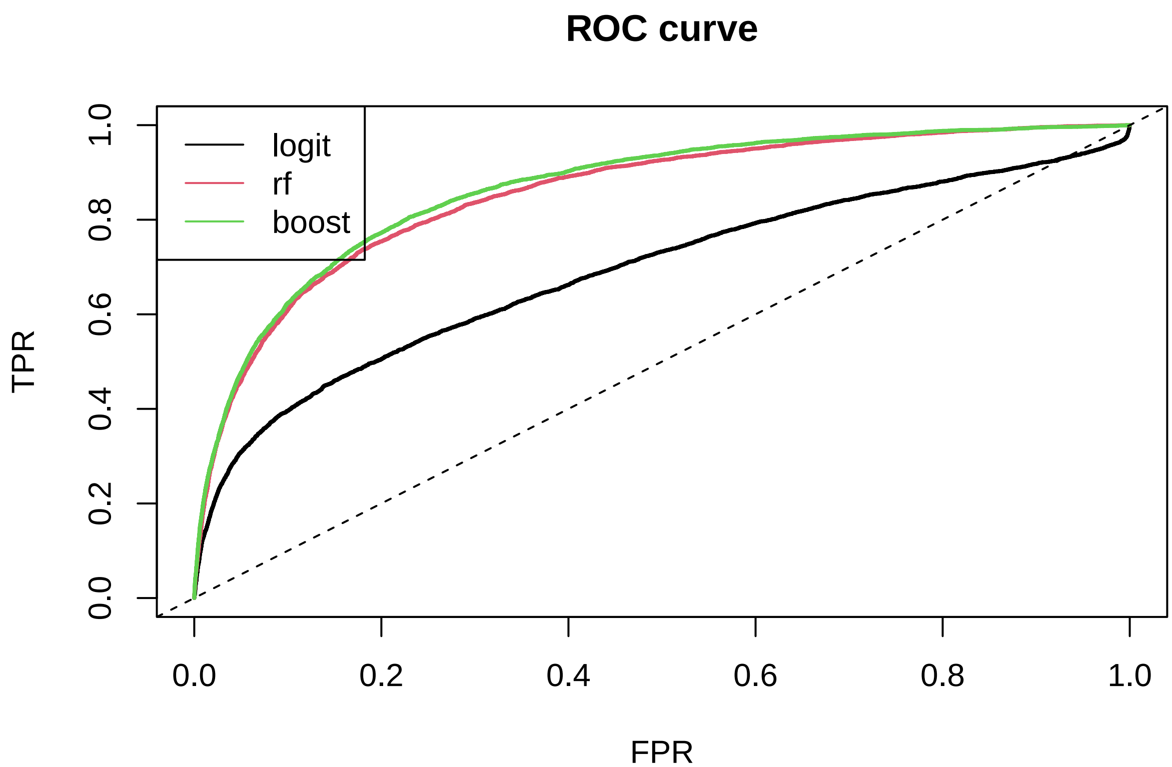
```
## Expected value of targeting using boosting =  1518
```

### 5.4 ROC curves

```
library(ROCR)
```

```
for(i in 1:ncol(phatBest)) {
   pred = prediction(phatBest[,i], td_validation$y)
   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).

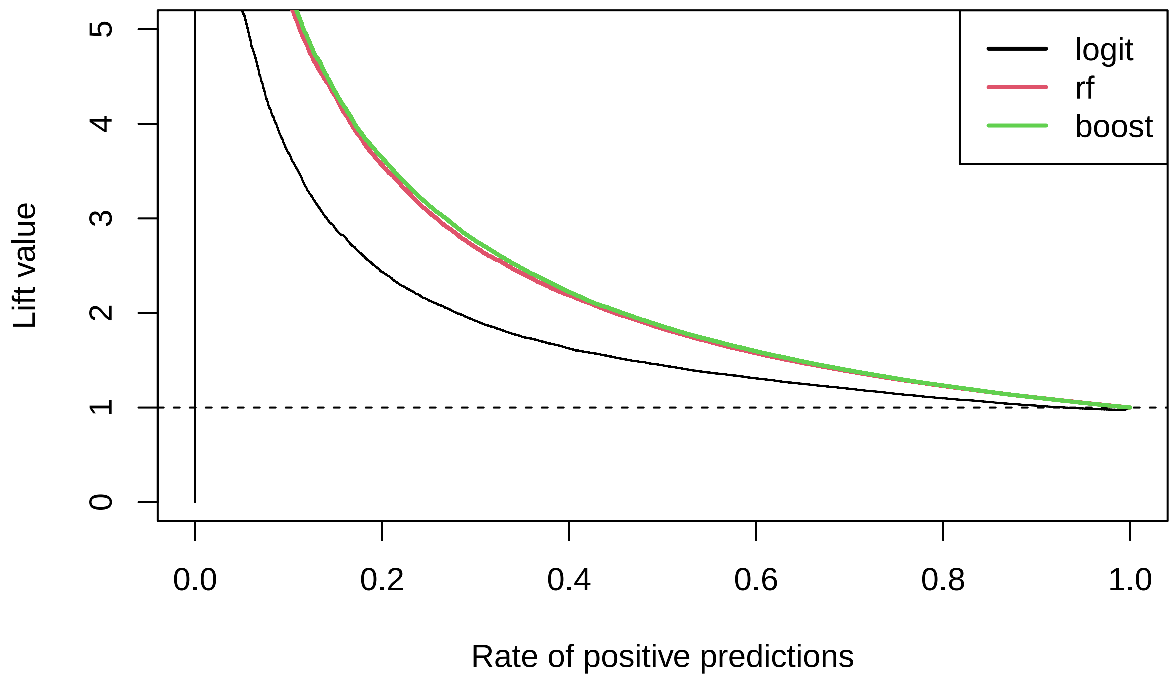
```
for(i in 1:ncol(phatBest)) {
  pred = prediction(phatBest[,i], td_validation$y)
  perf <- performance(pred, measure = "auc")
  print(paste0("AUC ", names(phat.list)[i], " :: ", perf@y.values[[1]]))
}
```

```
## [1] "AUC logit :: 0.689726238272716"
## [1] "AUC rf :: 0.855168271517304"
## [1] "AUC boost :: 0.86546139357898"
```

### 5.5 Lift curves

```
pred = prediction(phatBest[,1], td_validation$y)
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$y)
   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

```r
pred = prediction(phatBest[,1], td_validation$y)
perf = performance(pred, measure = "tpr", x.measure = "rpp")
plot(perf, col=1, ylim=c(0,1),lwd=2)
abline(h=1, lty=2)
abline(0,1,lty=2)
for(i in 2:ncol(phatBest)) {
   pred = prediction(phatBest[,i], td_validation$y)
   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)
```