

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.

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

trainDf = read.csv("CreditScoring.csv")

##remove X (is 1:n)
trainDf = trainDf[,-1]

##add y as factor
trainDf$y = as.factor(trainDf$SeriousDlqin2yrs)
trainDf = trainDf[,-1] # get rid of old y = SeriousDlqin2yrs

##get rid of NumberOfDependents, don't want to deal with NA's
trainDf=trainDf[,-10]
##get rid of MonthlyIncome, don't want to deal with NA's
trainDf=trainDf[,-5]

##split train in train, test
n=nrow(trainDf)
set.seed(99)
ii = sample(1:n,n)
ntest = floor(n/2)
testDf = trainDf[ii[1:ntest],]
trainDf = trainDf[ii[(ntest+1):n],]
```

3 Summary statistics

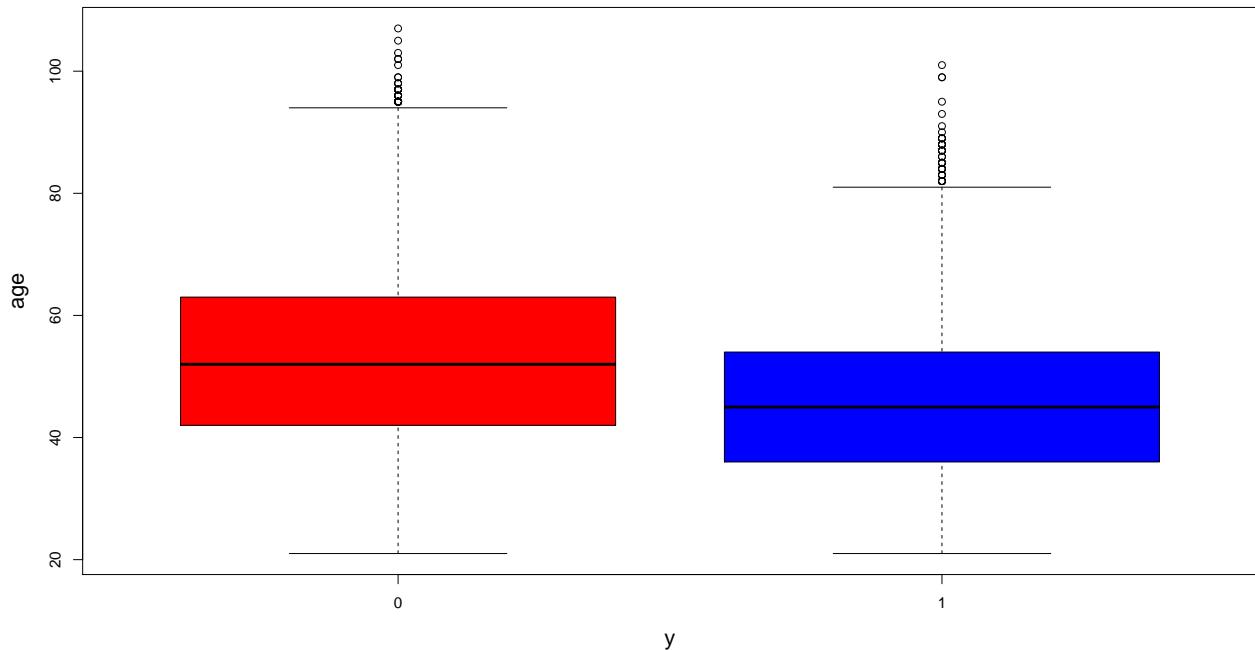
```
table(trainDf$y)
```

```
##  
##      0      1  
## 70021  4979
```

6 to 7 % of accounts are delinquent.

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

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



4 Fit models

We fit

- logistic regression
- random forest model
- boosting

```
library(ROCR)
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(y~, trainDf, family=binomial)
print(summary(lgfit))
```

```
## 
## Call:
## glm(formula = y ~ ., family = binomial, data = trainDf)
## 
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max 
## -3.120  -0.389  -0.317  -0.258   4.178 
## 
## Coefficients:
##                               Estimate
## (Intercept)                 -1.28e+00
## RevolvingUtilizationOfUnsecuredLines -4.05e-05
## age                         -3.18e-02
## NumberOfTime30.59DaysPastDueNotWorse  5.00e-01
## DebtRatio                   -2.65e-06
## NumberOfOpenCreditLinesAndLoans      -6.69e-03
## NumberOfTimes90DaysLate            4.49e-01
## NumberRealEstateLoansOrLines       3.65e-02
## NumberOfTime60.89DaysPastDueNotWorse -9.15e-01
##                               Std. Error
## (Intercept)                  5.63e-02
## RevolvingUtilizationOfUnsecuredLines 7.76e-05
## age                          1.16e-03
## NumberOfTime30.59DaysPastDueNotWorse 1.56e-02
## DebtRatio                   9.61e-06
## NumberOfOpenCreditLinesAndLoans 3.51e-03
## NumberOfTimes90DaysLate        2.13e-02
## NumberRealEstateLoansOrLines  1.46e-02
## NumberOfTime60.89DaysPastDueNotWorse 2.49e-02
##                               z value
## (Intercept)                -22.68
## RevolvingUtilizationOfUnsecuredLines -0.52
## age                      -27.35
## NumberOfTime30.59DaysPastDueNotWorse 32.07
## DebtRatio                  -0.28
## NumberOfOpenCreditLinesAndLoans -1.91
## NumberOfTimes90DaysLate      21.02
## NumberRealEstateLoansOrLines  2.49
## NumberOfTime60.89DaysPastDueNotWorse -36.78
##                               Pr(>|z|) 
## (Intercept)                <2e-16
## RevolvingUtilizationOfUnsecuredLines  0.602
## age                      <2e-16
## NumberOfTime30.59DaysPastDueNotWorse <2e-16
```

```

## DebtRatio          0.783
## NumberOfOpenCreditLinesAndLoans   0.057
## NumberOfTimes90DaysLate        <2e-16
## NumberRealEstateLoansOrLines    0.013
## 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: 36629  on 74999  degrees of freedom
## Residual deviance: 33728  on 74991  degrees of freedom
## AIC: 33746
##
## Number of Fisher Scoring iterations: 6
Predictions are stored for later analysis
phat = predict(lgfit, testDf, 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(trainDf)-1
mtryv = c(p, sqrt(p))
ntreeev = c(500,1000)
(setrf = expand.grid(mtryv,ntreeev)) # this contains all settings to try

##    Var1 Var2
## 1 8.00  500
## 2 2.83  500
## 3 8.00 1000
## 4 2.83 1000

colnames(setrf)=c("mtry","ntree")
phatL$rf = matrix(0.0,nrow(testDf),nrow(setrf)) # we will store results here

####fit rf
for(i in 1:nrow(setrf)) {
  #fit and predict
  frf = randomForest(y~., data=trainDf,
                      mtry=setrf[i,1],
                      ntree=setrf[i,2],
                      nodesize=10)
  phat = predict(frf, newdata=testDf, 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))

##   Var1 Var2 Var3
## 1    2 1000 0.10
## 2    4 1000 0.10
## 3    2 5000 0.10
## 4    4 5000 0.10
## 5    2 1000 0.01
## 6    4 1000 0.01
## 7    2 5000 0.01
## 8    4 5000 0.01

colnames(setboost) = c("tdepth","ntree","shrink")
phatL$boost = matrix(0.0,nrow(testDf),nrow(setboost))
```

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

```
trainDfB = trainDf; trainDfB$y = as.numeric(trainDfB$y)-1
testDfB = testDf; testDfB$y = as.numeric(testDfB$y)-1
```

Fitting

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

  phat = predict(fboost,
                 newdata=testDfB,
                 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(testDf$y, phatL[[1]][,1], 0.5)

##           actual
## predictions      0      1
##   predict_0 69812  4849
##   predict_1   141   198

cat('Missclassification rate = ', lossMR(testDf$y, phatL[[1]][,1], 0.5), '\n')

## Missclassification rate =  0.0665
```

For **random forest** we have:

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

## mtry ntree
## 1     8   500
## [1] "Confusion Matrix:"
##           actual
## predictions 0      1
## predict_0 68986 3963
## predict_1  967 1084
## Missclassification rate =  0.0657
## mtry ntree
## 2 2.83   500
## [1] "Confusion Matrix:"
##           actual
## predictions 0      1
## predict_0 69190 4047
## predict_1  763 1000
## Missclassification rate =  0.0641
## mtry ntree
## 3     8   1000
## [1] "Confusion Matrix:"
##           actual
## predictions 0      1
## predict_0 68992 3982
## predict_1  961 1065
## Missclassification rate =  0.0659
## mtry ntree
## 4 2.83   1000
## [1] "Confusion Matrix:"
##           actual
## predictions 0      1
## predict_0 69189 4039
## predict_1  764 1008
## Missclassification rate =  0.064
```

For **boosting** we have:

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

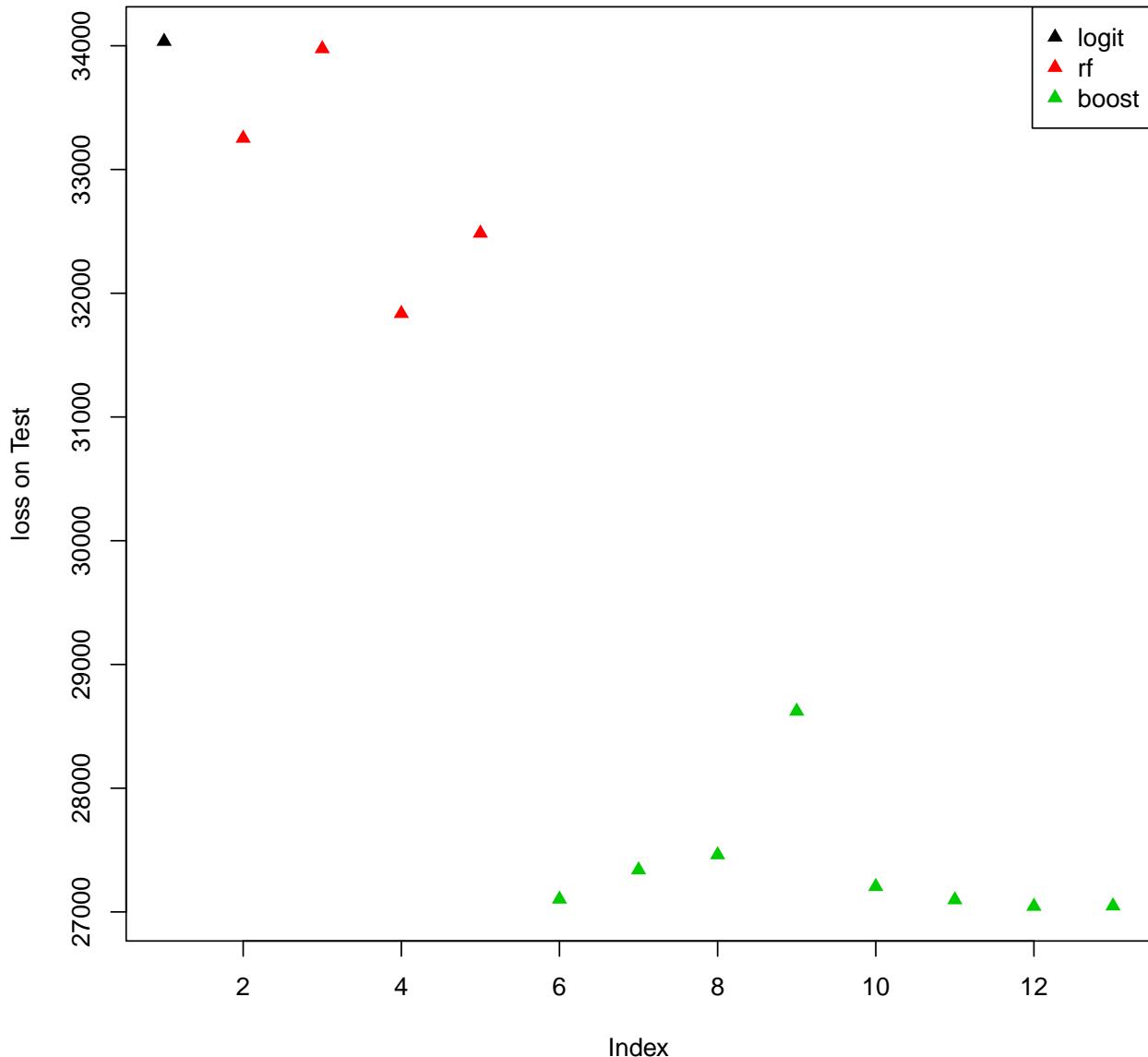
## tdepth ntree shrink
## 1      2 1000   0.1
## [1] "Confusion Matrix:"
##           actual
## predictions 0     1
## predict_0 69257 4045
## predict_1  696 1002
## Missclassification rate =  0.0632
## tdepth ntree shrink
## 2      4 1000   0.1
## [1] "Confusion Matrix:"
##           actual
## predictions 0     1
## predict_0 69184 4077
## predict_1  769 970
## Missclassification rate =  0.0646
## tdepth ntree shrink
## 3      2 5000   0.1
## [1] "Confusion Matrix:"
##           actual
## predictions 0     1
## predict_0 69176 4115
## predict_1  777 932
## Missclassification rate =  0.0652
## tdepth ntree shrink
## 4      4 5000   0.1
## [1] "Confusion Matrix:"
##           actual
## predictions 0     1
## predict_0 68992 4048
## predict_1  961 999
## Missclassification rate =  0.0668
## tdepth ntree shrink
## 5      2 1000   0.01
## [1] "Confusion Matrix:"
##           actual
## predictions 0     1
## predict_0 69310 4077
## predict_1  643 970
## Missclassification rate =  0.0629
## tdepth ntree shrink
## 6      4 1000   0.01
## [1] "Confusion Matrix:"
##           actual
## predictions 0     1
```

```
##   predict_0 69292  4038
##   predict_1   661  1009
## Missclassification rate =  0.0627
##   tdepth ntree shrink
## 7      2 5000  0.01
## [1] "Confusion Matrix:"
##           actual
## predictions 0     1
##   predict_0 69281  4046
##   predict_1   672  1001
## Missclassification rate =  0.0629
##   tdepth ntree shrink
## 8      4 5000  0.01
## [1] "Confusion Matrix:"
##           actual
## predictions 0     1
##   predict_0 69261  4071
##   predict_1   692   976
## Missclassification rate =  0.0635
```

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(testDf$y, 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.

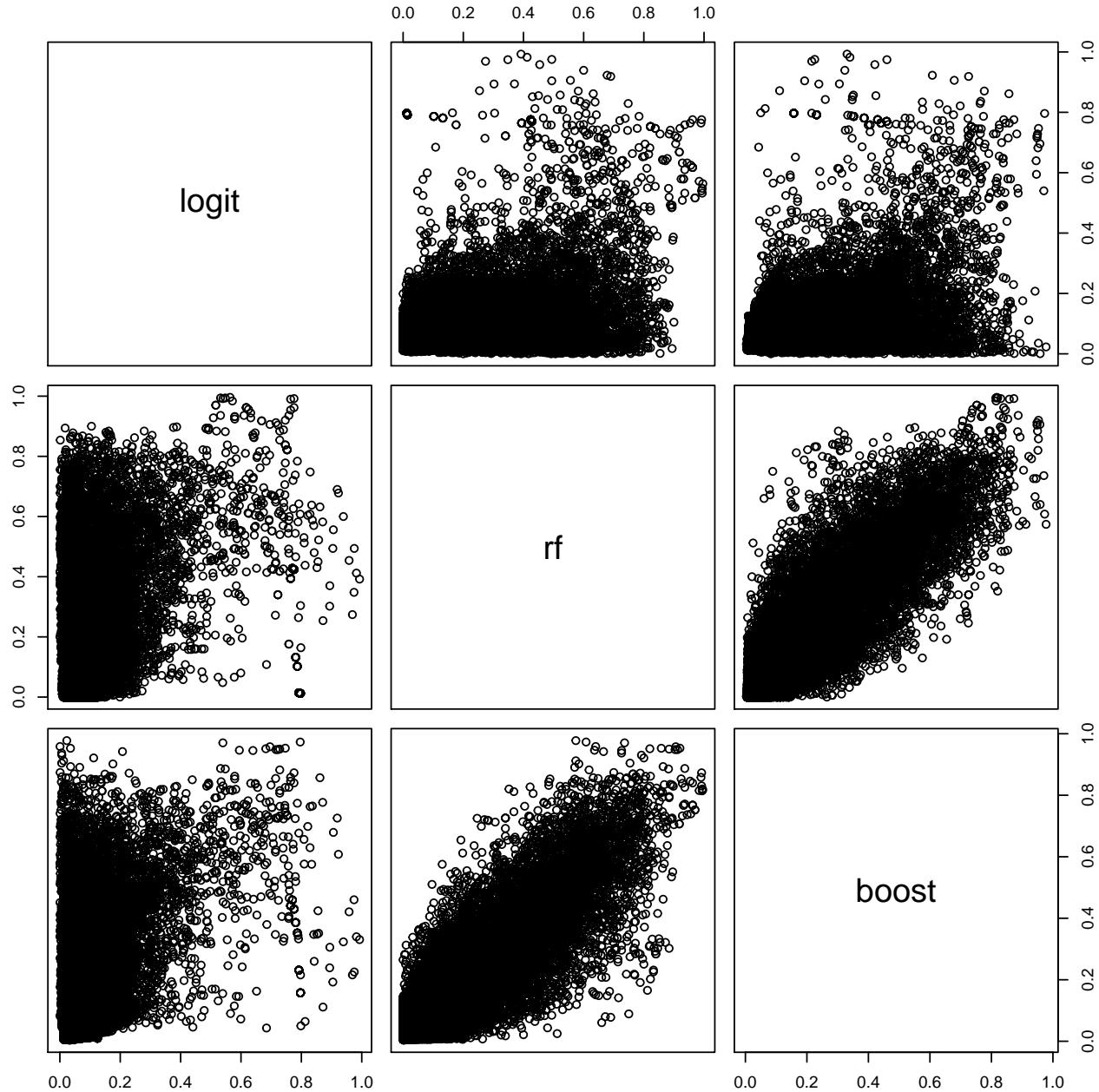
```

nmethod = length(phatL)
phatBest = matrix(0.0,nrow=testDf ,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(testDf$y,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)
```

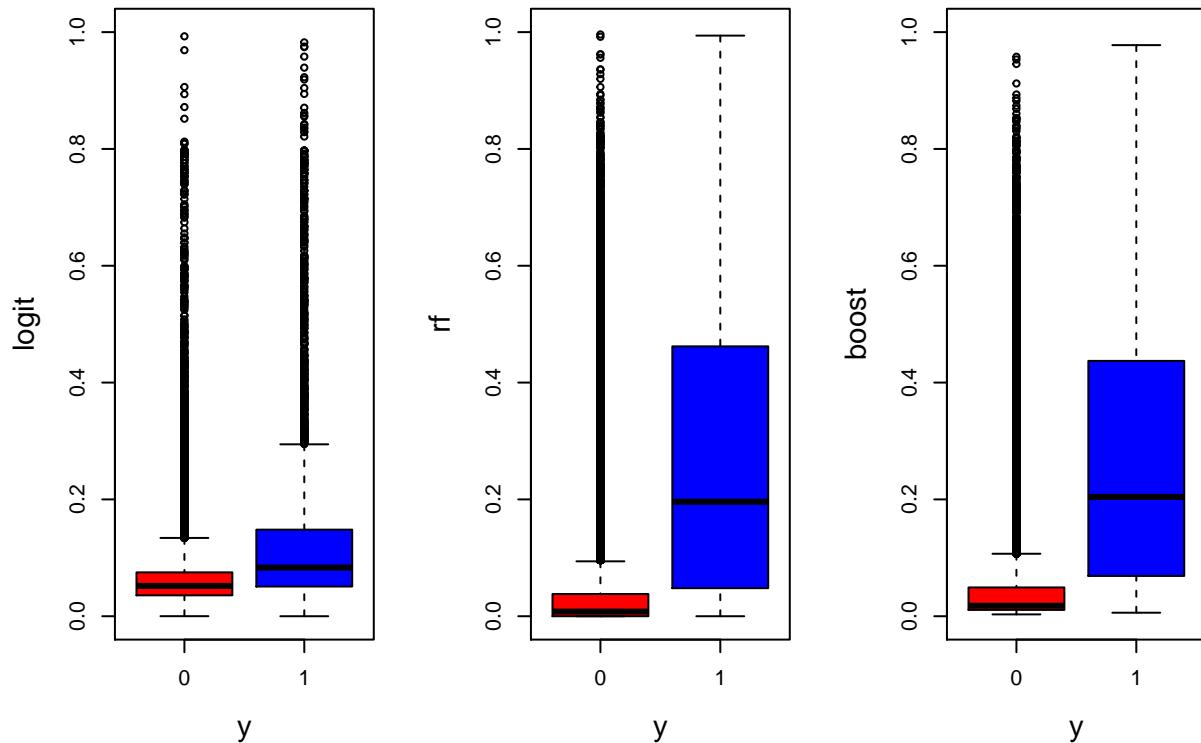


Each plot relates \hat{p} to y .

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 = testDf$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 are below:

```
confMat = getConfusionMatrix(testDf$y, phatBest[,1], 0.2)
print(confMat)
```

```
##           actual
## predictions    0     1
## predict_0 68897 4224
## predict_1 1056   823
cat("Expected value of logistic regression = ",
    sum(sum(confMat * cost_benefit)), "\n")
```

```
## Expected value of logistic regression = 559
```

```
confMat = getConfusionMatrix(testDf$y, phatBest[,2], 0.2)
print(confMat)
```

```
##           actual
## predictions    0     1
## predict_0 65486 2552
## predict_1 4467  2495
```

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

```
## Expected value of random forests = 1378
```

```
confMat = getConfusionMatrix(testDf$y, phatBest[,3], 0.2)
print(confMat)
```

```
##           actual
## predictions    0     1
## predict_0 65997 2507
## predict_1 3956  2540
```

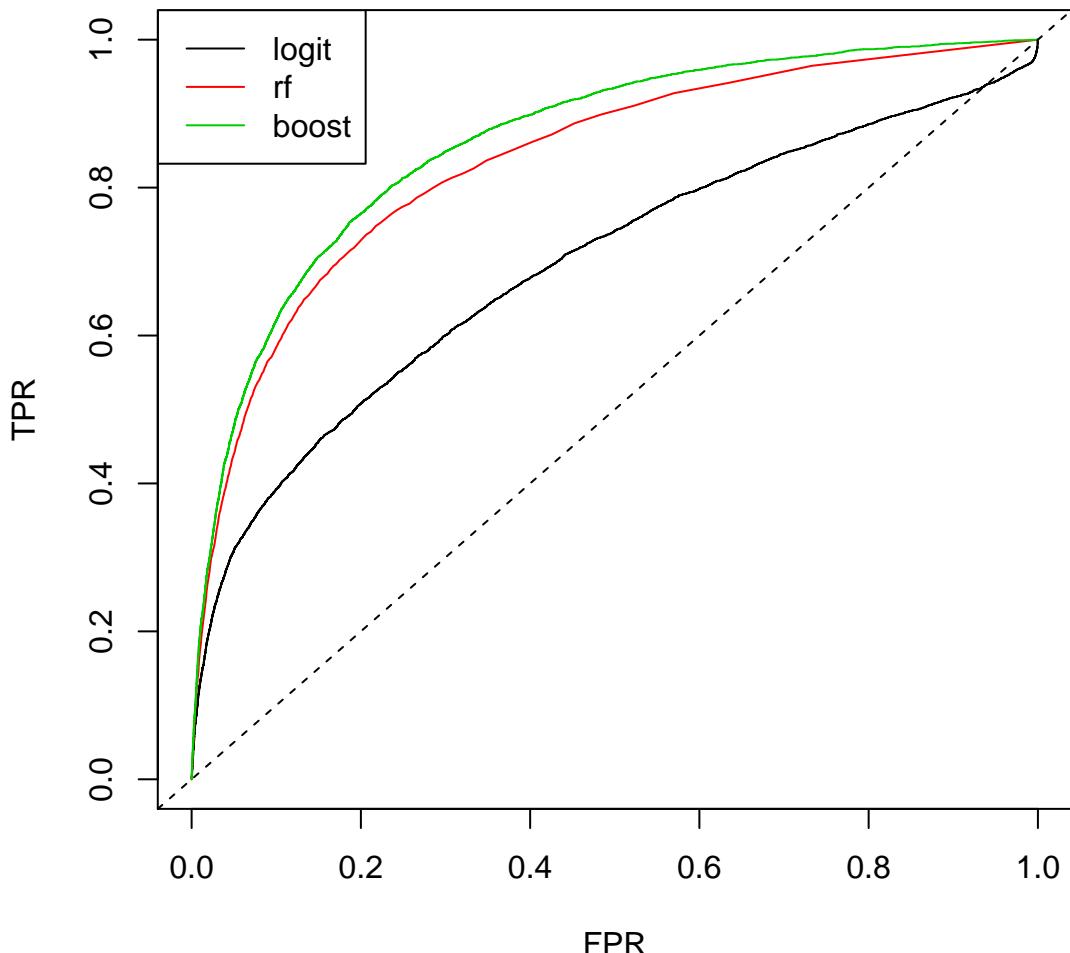
```
cat("Expected value of boosting = ",
    sum(sum(confMat * cost_benefit)), "\n")
```

```
## Expected value of boosting = 1551
```

5.4 ROC curves

```
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], testDf$y)
  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))
```

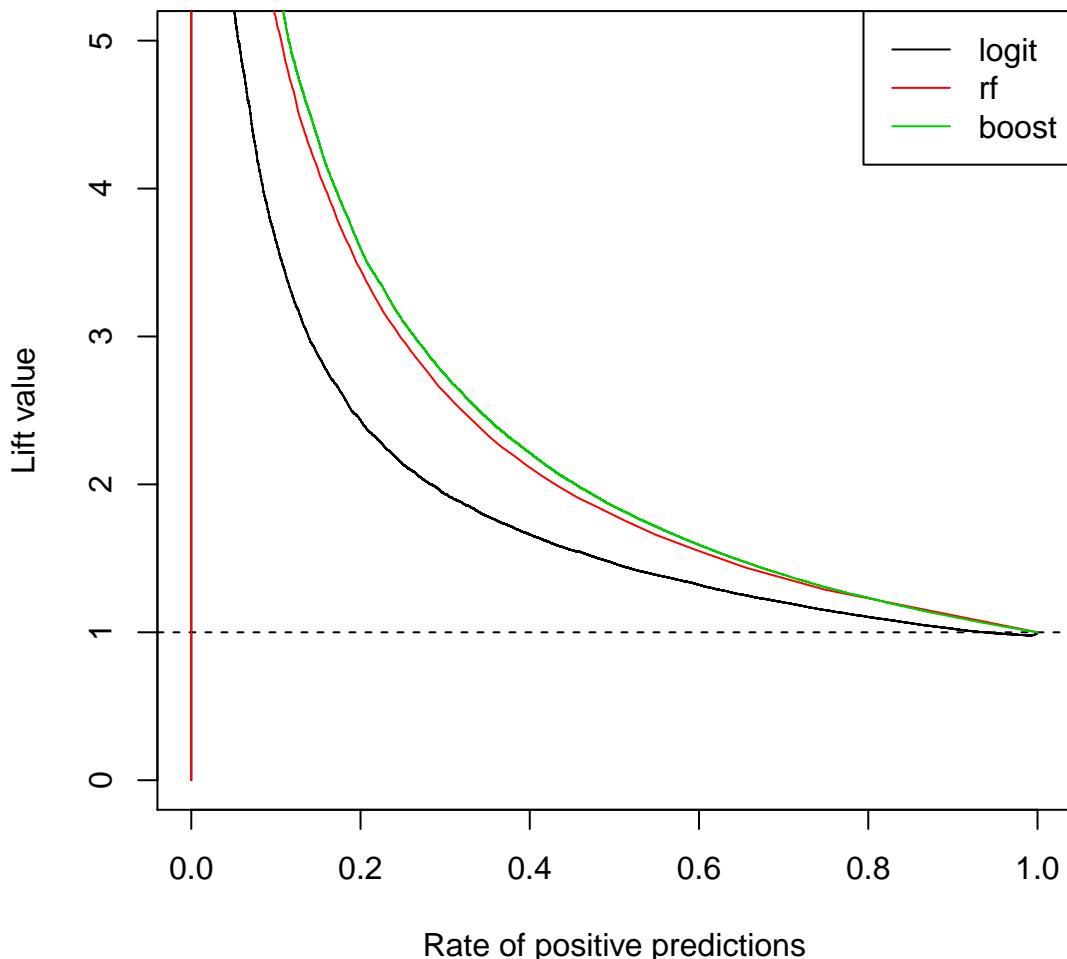
ROC curve



5.5 Lift curves

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

