BUS41204: Week 4 Review Session

JungHo Lee

2023-01-28

Plan of Attack

- Review: Classification and Evaluating Classifiers
- Example: Credit Scoring dataset from Kaggle

1. Review: Classification and Evaluating Classifiers

Classification

Classification is about predicting a qualitative response (category) for an observation. Some popular methods include:

- k-NN
- Logistic Regression
- NaiveBayes
- Tree-based Methods (Classification Tree, Random Forests, Boosting)
- Linear/Quadratic Discriminant Analyses
- etc.

Evaluating Classifiers

Misclassification

- Difficult to optimize
- Too simplistic (reduces performance of a classifier to a single number)
  - Does not make the distinction between false positive and false negative errors
- Does not work well with unbalanced classes
  - Categorizing all instances to the majority would yield low missclassification rate, but useless

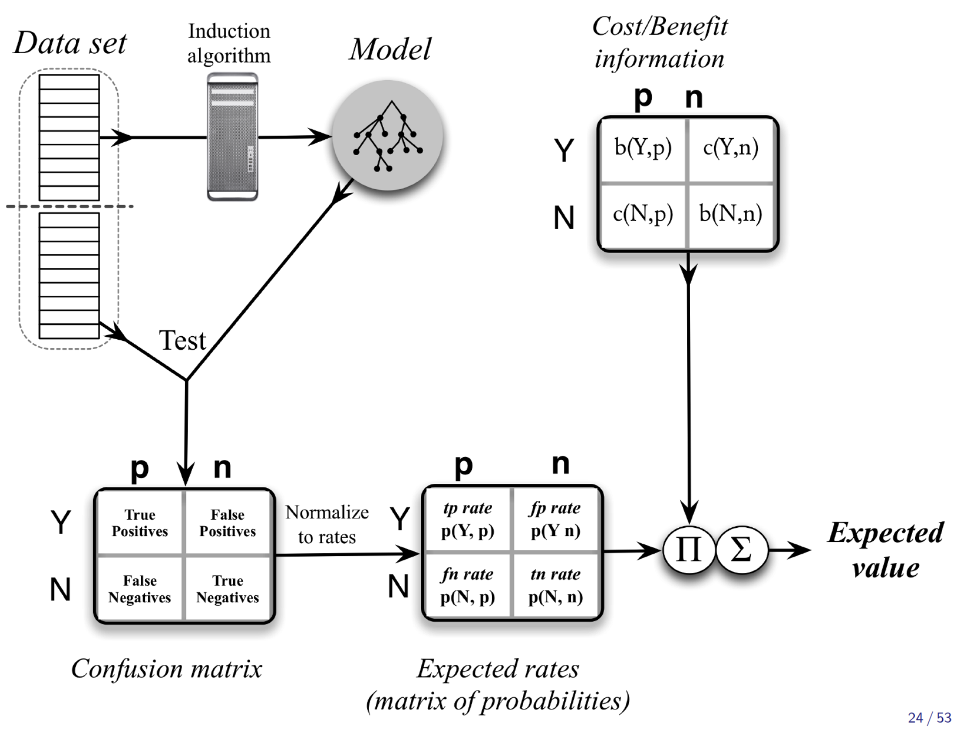
Confusion matrix

- Columns are labeled by actual classes
- Rows are labeled by predicted classes

	positive	**n**egative
Y	True positives	False positives
N	False negatives	True negatives

Expected Values

- Useful in organizing thinking about data-analytic problems



ROC (Receiver Operating Characteristic) Curve

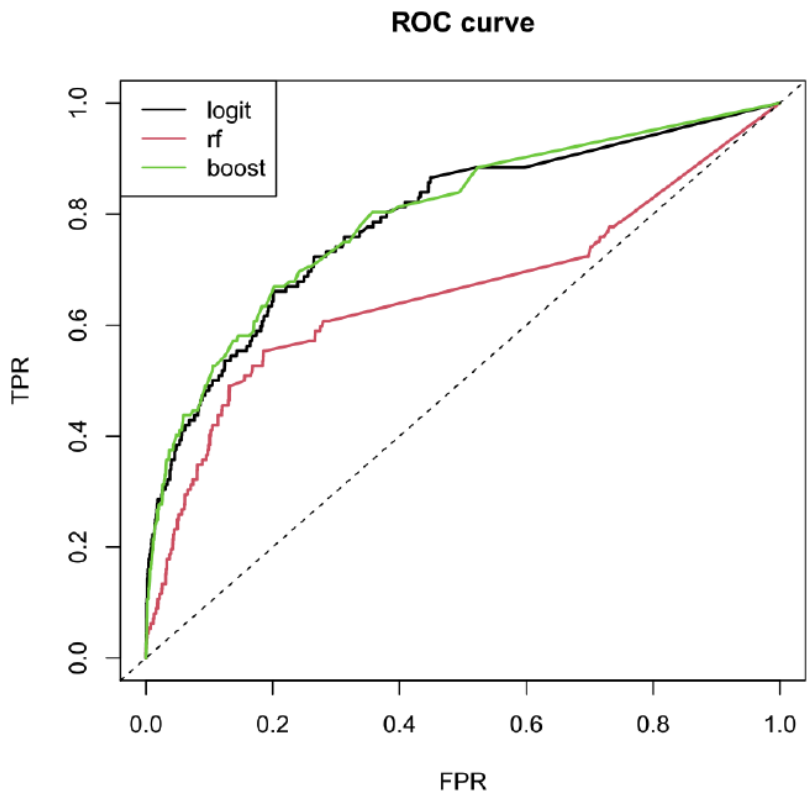
- Depicts relative trade-offs that a classifier makes between benefits (true positives) and costs (false positives).
- An ideal ROC curve will hug the top left corner, the larger the *are under the (ROC) curve* (AUC), the better the classifier.

  Given p and $\hat{p}$, Consider k classifier C (threshold) such as $C_1 = 0.1, C_2 = 0.2, ..., C_i = 0.5, ..., C_k = 0.9$

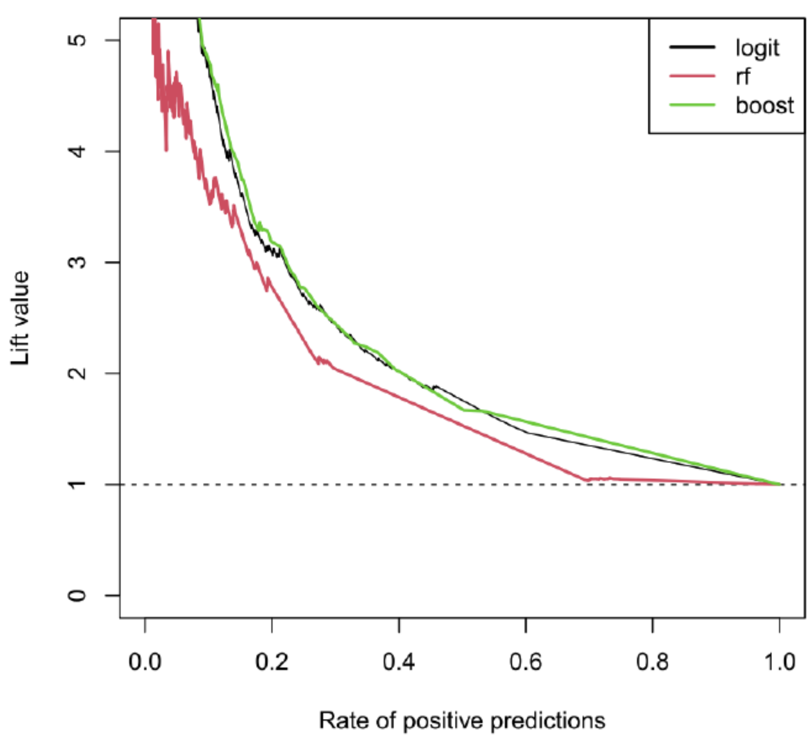
  Step 1: According to each threshold C_i, calculate the related TPR_i and FPR_i

  Step 2: Collect these K ($\text{TPR}_i, \text{FPR}_i$)

  Step 3: Create the plot of $y = \text{TPR}$ versus $x = \text{FPR}$


ROC curve
logit
rf
boost
TPR
FPR
0.0
0.2
0.4
0.6
0.8
1.0

Lift Curve


logit
rf
boost
Lift value
Rate of positive predictions
0
1
2
3
4
5
0.0
0.2
0.4
0.6
0.8
1.0

- Lift $= \frac{\frac{TP}{TP+FP}}{\frac{TP+FP}{TP+FP+TN+FN}}$
- How much the customer conversion is multiplied for a certain percentage of customer contacted
- Closely related with the cumulative response curve

2. CreditSCoring Dataset from Kaggle

Background

Banks play a crucial role in market economies. They decide who can get finance and on what terms and can make or break investment decisions. For markets and society to function, individuals and companies need access to credit.

Credit scoring algorithms, which make a guess at the probability of default, are the method banks use to determine whether or not a loan should be granted. This competition requires participants to improve on the state of the art in credit scoring, by predicting the probability that somebody will experience financial distress in the next two years.

The goal of this competition is to build a model that borrowers can use to help make the best financial decisions.

Data Preprocessing

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

train = read.csv("CreditScoring.csv")
```

```
dim(train)
```

```
## [1] 150000     12
```

```
head(train)
```

```
##   X SeriousDlqin2yrs RevolvingUtilizationOfUnsecuredLines age
## 1 1                1                            0.7661266  45
## 2 2                0                            0.9571510  40
## 3 3                0                            0.6581801  38
## 4 4                0                            0.2338098  30
## 5 5                0                            0.9072394  49
## 6 6                0                            0.2131787  74
##   NumberOfTime30.59DaysPastDueNotWorse  DebtRatio MonthlyIncome
## 1                                    2 0.80298213          9120
## 2                                    0 0.12187620          2600
## 3                                    1 0.08511338          3042
## 4                                    0 0.03604968          3300
## 5                                    1 0.02492570         63588
## 6                                    0 0.37560697          3500
```

```
##   NumberOfOpenCreditLinesAndLoans NumberOfTimes90DaysLate
## 1                              13                       0
## 2                               4                       0
## 3                               2                       1
## 4                               5                       0
## 5                               7                       0
## 6                               3                       0
##   NumberRealEstateLoansOrLines NumberOfTime60.89DaysPastDueNotWorse
## 1                            6                                    0
## 2                            0                                    0
## 3                            0                                    0
## 4                            0                                    0
## 5                            1                                    0
## 6                            1                                    0
##   NumberOfDependents
## 1                  2
## 2                  1
## 3                  0
## 4                  0
## 5                  0
## 6                  1
```

```
summary(train)
```

```
##        X          SeriousDlqin2yrs  RevolvingUtilizationOfUnsecuredLines
##  Min.   :     1   Min.   :0.00000   Min.   :    0.00
##  1st Qu.: 37501   1st Qu.:0.00000   1st Qu.:    0.03
##  Median : 75000   Median :0.00000   Median :    0.15
##  Mean   : 75000   Mean   :0.06684   Mean   :    6.05
##  3rd Qu.:112500   3rd Qu.:0.00000   3rd Qu.:    0.56
##  Max.   :150000   Max.   :1.00000   Max.   :50708.00
##
##       age        NumberOfTime30.59DaysPastDueNotWorse   DebtRatio
##  Min.   :  0.0   Min.   : 0.000                       Min.   :     0.0
##  1st Qu.: 41.0   1st Qu.: 0.000                       1st Qu.:     0.2
##  Median : 52.0   Median : 0.000                       Median :     0.4
##  Mean   : 52.3   Mean   : 0.421                       Mean   :   353.0
##  3rd Qu.: 63.0   3rd Qu.: 0.000                       3rd Qu.:     0.9
##  Max.   :109.0   Max.   :98.000                       Max.   :329664.0
##
##  MonthlyIncome     NumberOfOpenCreditLinesAndLoans NumberOfTimes90DaysLate
##  Min.   :      0   Min.   : 0.000                  Min.   : 0.000
##  1st Qu.:   3400   1st Qu.: 5.000                  1st Qu.: 0.000
##  Median :   5400   Median : 8.000                  Median : 0.000
##  Mean   :   6670   Mean   : 8.453                  Mean   : 0.266
##  3rd Qu.:   8249   3rd Qu.:11.000                  3rd Qu.: 0.000
##  Max.   :3008750   Max.   :58.000                  Max.   :98.000
##  NA's   :29731
##  NumberRealEstateLoansOrLines NumberOfTime60.89DaysPastDueNotWorse
##  Min.   : 0.000               Min.   : 0.0000
##  1st Qu.: 0.000               1st Qu.: 0.0000
##  Median : 1.000               Median : 0.0000
##  Mean   : 1.018               Mean   : 0.2404
##  3rd Qu.: 2.000               3rd Qu.: 0.0000
```

```
##  Max.   :54.000               Max.   :98.0000
##
##  NumberOfDependents
##  Min.   : 0.000
##  1st Qu.: 0.000
##  Median : 0.000
##  Mean   : 0.757
##  3rd Qu.: 1.000
##  Max.   :20.000
##  NA's   :3924
```

```
str(train)
```

```
## 'data.frame':    150000 obs. of  12 variables:
##  $ X                                   : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ SeriousDlqin2yrs                    : int  1 0 0 0 0 0 0 0 0 0 ...
##  $ RevolvingUtilizationOfUnsecuredLines: num  0.766 0.957 0.658 0.234 0.907 ...
##  $ age                                 : int  45 40 38 30 49 74 57 39 27 57 ...
##  $ NumberOfTime30.59DaysPastDueNotWorse: int  2 0 1 0 1 0 0 0 0 0 ...
##  $ DebtRatio                           : num  0.803 0.1219 0.0851 0.036 0.0249 ...
##  $ MonthlyIncome                       : int  9120 2600 3042 3300 63588 3500 NA 3500 NA 23684 ...
##  $ NumberOfOpenCreditLinesAndLoans     : int  13 4 2 5 7 3 8 8 2 9 ...
##  $ NumberOfTimes90DaysLate             : int  0 0 1 0 0 0 0 0 0 0 ...
##  $ NumberRealEstateLoansOrLines        : int  6 0 0 0 1 1 3 0 0 4 ...
##  $ NumberOfTime60.89DaysPastDueNotWorse: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ NumberOfDependents                  : int  2 1 0 0 0 1 0 0 NA 2 ...
```

Rename `SeriousDlqin2yrs` to `y` and convert it to a factor.

```
library(tidyverse)

df = train %>% rename(y = SeriousDlqin2yrs) %>% mutate(across(y, as.factor))
```

Get rid of variables `X` (index), `MonthlyIncome` and `NumberOfDependents` as we do not want to deal with NAs.

```
df = df %>% select(-c(X, MonthlyIncome, NumberOfDependents))
str(df)
```

```
## 'data.frame':    150000 obs. of  9 variables:
##  $ y                                   : Factor w/ 2 levels "0","1": 2 1 1 1 1 1 1 1 1 1 ...
##  $ RevolvingUtilizationOfUnsecuredLines: num  0.766 0.957 0.658 0.234 0.907 ...
##  $ age                                 : int  45 40 38 30 49 74 57 39 27 57 ...
##  $ NumberOfTime30.59DaysPastDueNotWorse: int  2 0 1 0 1 0 0 0 0 0 ...
##  $ DebtRatio                           : num  0.803 0.1219 0.0851 0.036 0.0249 ...
##  $ NumberOfOpenCreditLinesAndLoans     : int  13 4 2 5 7 3 8 8 2 9 ...
##  $ NumberOfTimes90DaysLate             : int  0 0 1 0 0 0 0 0 0 0 ...
##  $ NumberRealEstateLoansOrLines        : int  6 0 0 0 1 1 3 0 0 4 ...
##  $ NumberOfTime60.89DaysPastDueNotWorse: int  0 0 0 0 0 0 0 0 0 0 ...
```

Split data into training and validation data:

```r
set.seed(99)

n=nrow(df)
ii = sample(n,n/2)
train_df = df[ii,]
val_df= df[-ii,]
```

Quick Summary Statistics

```r
table(df$y)
```

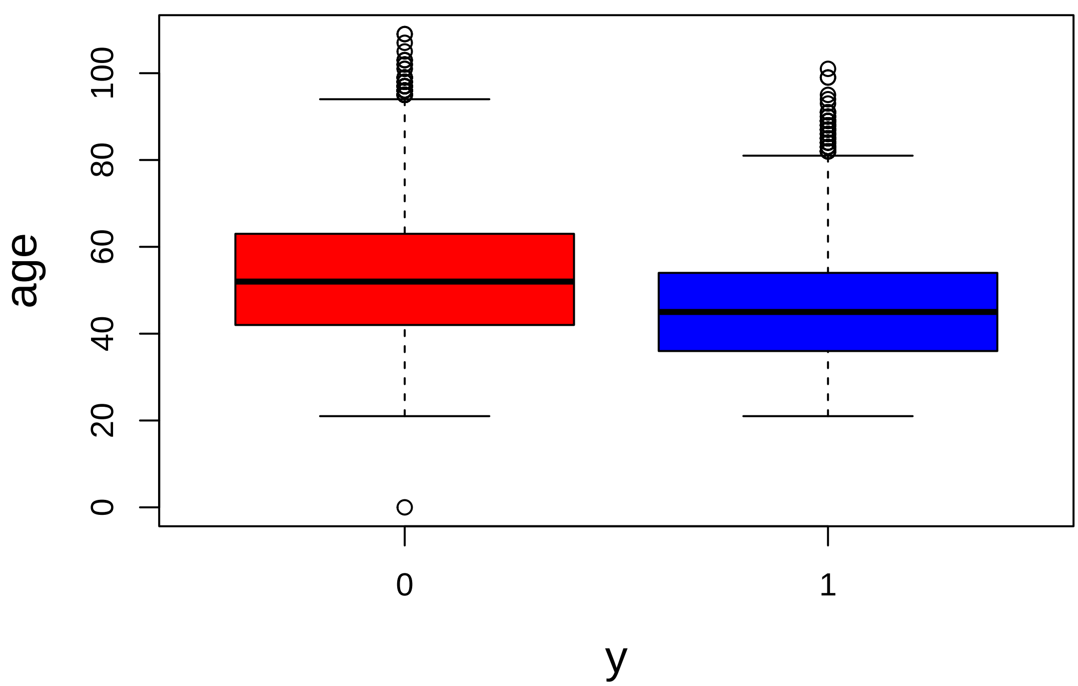
```
##
##      0      1
## 139974  10026
```

```r
table(train_df$y)
```

```
##
##     0     1
## 69955  5045
```

Roughly, 6.7% are delinquent.

```r
plot(age~y, df, col=c('red','blue'), cex.lab=1.4)
```



Model Fitting

We will fit:

- Logistic Regression
- Random Forest
- Boosting

```
phat_list = list() # where the phats will be stored
```

Logistic Regression

```
lg_fit = glm(y~., data=train_df, family=binomial)
lg_phat = predict(lg_fit, val_df, type='response')
phat_list$logit = matrix(lg_phat, ncol=1)
```

Random Forest

```
library(ranger)
```

```
p = ncol(df)-1

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

phat_list$rf = matrix(0, nrow(val_df), nrow(grid_rf)) # where rf phats will be stored

for(i in 1:nrow(grid_rf)){
    rf_fit = ranger(
    formula = y~.,
    data = train_df,
    num.trees = 1000,
    mtry = grid_rf$mtry[i],
    min.node.size = grid_rf$node_size[i],
    probability = TRUE,
    seed = 99
  )

  phat_rf = predict(rf_fit, data=val_df)$predictions[,2]
  phat_list$rf[,i] = phat_rf
}
```

Boosting

```r
library(dbarts)
library(xgboost)

X_train = makeModelMatrixFromDataFrame(train_df[,-1])
y_train = as.numeric(train_df$y)-1
X_val = makeModelMatrixFromDataFrame(val_df[,-1])
y_val = as.numeric(val_df$y)-1
```

```r
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
)

# where boosting phats will be stored
phat_list$boost = matrix(0, nrow(val_df), nrow(grid_xgb))
```

Fitting boosting:

```r
for (i in 1:nrow(grid_xgb)){
  # create param list
  params = list(
    eta = grid_xgb$shrinkage[i],
    max_depth = grid_xgb$interaction.depth[i]
  )

  set.seed(4776)

  xbg_fit = xgboost(
    data = X_train,
    label = y_train,
    params = params,
    nrounds = grid_xgb$nrounds[1],
    objective = 'binary:logistic',
    verbose = 0,
    verbosity = 0
  )

  phat_xgb = predict(xbg_fit, newdata=X_val)
  phat_list$boost[,i] = phat_xgb
}
```

Evaluating Results

Misclassification Rate

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

```r
library(caret)
```

```r
#' @param y: should be 0/1
#' @param phat: probabilities obtained by our algorithm
#' @param thr: threshold: everything above thr will be classified as 1
#' @return confusion matrix
get_confusion_matrix = function(y, phat, thr=0.5){
  yhat = as.factor(ifelse(phat > thr, 1, 0)) # 1 of greater than thr, 0 o.w.
  confusionMatrix(yhat, y)
}
```

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

```r
#' @param y: should be 0/1
#' @param phat: probabilities obtained by our algorithm
#' @param thr: threshold: everything above thr will be classified as 1
#' @return misclassification rate
get_misc_rate = function(y, phat, thr=0.5){
  1 - get_confusion_matrix(y, phat, thr)$overall[1]
}
```

Logistic regression results:

```r
cfm = get_confusion_matrix(val_df$y, phat_list$logit[,1], 0.5)
print(cfm, printStats = F)
```

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

```r
cat('misclassification rate = ',
get_misc_rate(val_df$y, phat_list[[1]][,1], 0.5), '\n')
```

```
## misclassification rate =  0.06601333
```

Random forest:

```r
nrun = nrow(grid_rf)

for(j in 1:nrun) {
  print(grid_rf[j,])
  cfm = get_confusion_matrix(val_df$y, phat_list[[2]][,j], 0.5)
  print(cfm, printStats = F)
  cat('misclassification rate = ',
  get_misc_rate(val_df$y, phat_list[[2]][,j], 0.5), '\n')
}
```

```
##   mtry node_size
## 1    8         5
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 69081  3975
##          1   938  1006
## misclassification rate =  0.06550667
##   mtry node_size
## 2    3         5
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 69260  4028
##          1   759   953
## misclassification rate =  0.06382667
##   mtry node_size
## 3    8        10
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 69149  3985
##          1   870   996
## misclassification rate =  0.06473333
##   mtry node_size
## 4    3        10
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 69284  4036
##          1   735   945
## misclassification rate =  0.06361333
##   mtry node_size
## 5    8        20
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 69212  4016
##          1   807   965
## misclassification rate =  0.06430667
##   mtry node_size
## 6    3        20
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 69348  4067
##          1   671   914
## misclassification rate =  0.06317333
```

Boosting:

```
nrun = nrow(grid_xgb)

for(j in 1:nrun) {
  print(grid_xgb[j,])
  cfm = get_confusion_matrix(val_df$y, phat_list[[3]][,j], 0.5)
  print(cfm, printStats = F)
  cat('misclassification rate = ',
  get_misc_rate(val_df$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 69533  4263
##          1   486   718
## misclassification rate =  0.06332
##   shrinkage interaction.depth nrounds
## 2       0.1                 1    1000
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 69351  4028
##          1   668   953
## misclassification rate =  0.06261333
##   shrinkage interaction.depth nrounds
## 3      0.01                 2    1000
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 69460  4148
##          1   559   833
## misclassification rate =  0.06276
##   shrinkage interaction.depth nrounds
## 4       0.1                 2    1000
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 69337  3999
##          1   682   982
## misclassification rate =  0.06241333
##   shrinkage interaction.depth nrounds
## 5      0.01                 4    1000
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
```

```
##          0 69443  4063
##          1   576   918
## misclassification rate =  0.06185333
##   shrinkage interaction.depth nrounds
## 6       0.1                 4    1000
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 69265  4039
##          1   754   942
## misclassification rate =  0.06390667
##   shrinkage interaction.depth nrounds
## 7      0.01                 1    5000
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 69533  4263
##          1   486   718
## misclassification rate =  0.06332
##   shrinkage interaction.depth nrounds
## 8       0.1                 1    5000
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 69351  4028
##          1   668   953
## misclassification rate =  0.06261333
##   shrinkage interaction.depth nrounds
## 9      0.01                 2    5000
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 69460  4148
##          1   559   833
## misclassification rate =  0.06276
##    shrinkage interaction.depth nrounds
## 10       0.1                 2    5000
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 69337  3999
##          1   682   982
## misclassification rate =  0.06241333
##    shrinkage interaction.depth nrounds
## 11      0.01                 4    5000
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
```

```
##          0 69443  4063
##          1   576   918
## misclassification rate =  0.06185333
##    shrinkage interaction.depth nrounds
## 12       0.1                 4    5000
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 69265  4039
##          1   754   942
## misclassification rate =  0.06390667
```

Deviance

- One of the surrogate losses discussed in class
- Measures the quality of a model by looking at $\hat{P}(Y = y, | X = x)$
- Intuition: if $\hat{P}(Y = y, | X = x)$ is high, then the observed data is likely under our model.

The total deviance loss for a dataset $\mathcal{D}$ is given by:

$$L(\hat{P}) = \sum_{i \in \mathcal{D}} L(\hat{P}, x_i, y_i) = \sum_{i \in \mathcal{D}} -2 \log(P(Y = y_i \mid x_i)).$$

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

```
#' @param y: should be 0/1
#' @param phat: probabilities obtained by our algorithm
#' @param wht: shrinks probabilities in phat towards .5
#' this helps avoid numerical problems --- don't use log(0)!
#' @return deviance loss
get_deviance = function(y,phat,wht=1e-7) {
  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 loss on validation:

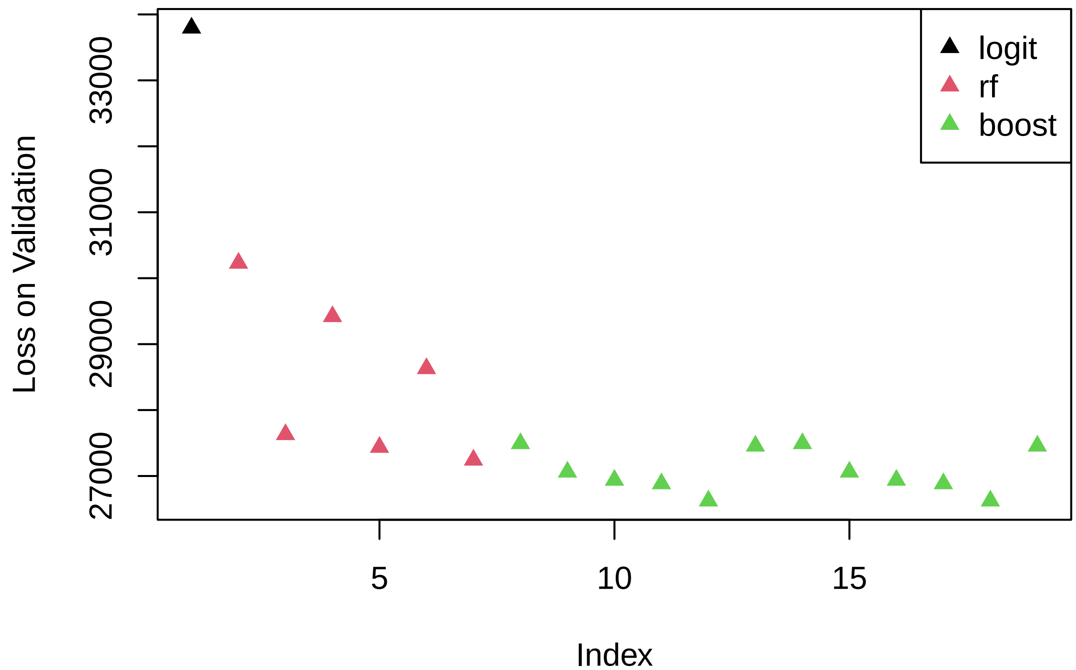
```
loss_list = list()
nmethod = length(phat_list)

for(i in 1:nmethod){
  nrun = ncol(phat_list[[i]])
  lvec = rep(0,nrun)
  for(j in 1:nrun){
    lvec[j] = get_deviance(val_df$y, phat_list[[i]][,j])
    loss_list[[i]] = lvec
    names(loss_list)[i] = names(phat_list)[i]
  }
```

```r
}

lossv = unlist(loss_list)
plot(lossv, ylab="Loss on Validation", 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.

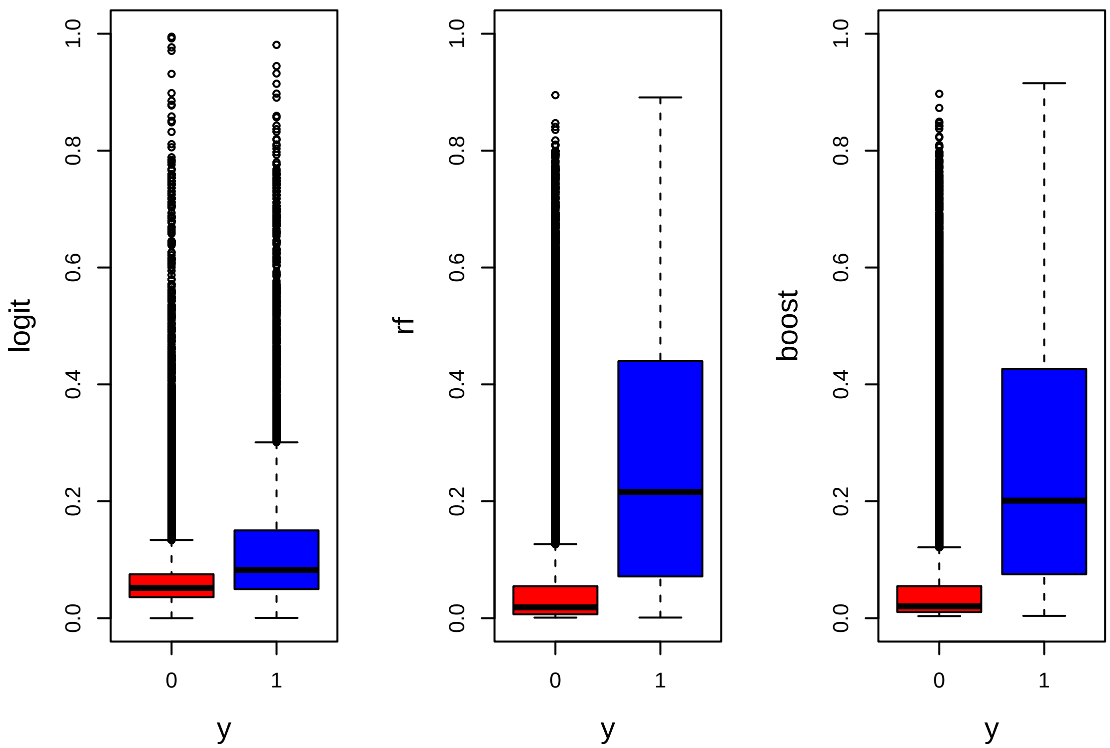
```r
phat_best = matrix(0.0,nrow(val_df),nmethod) #pick off best from each method
colnames(phat_best) = names(phat_list)

for(i in 1:nmethod) {
  nrun = ncol(phat_list[[i]])
  lvec = rep(0,nrun)
  for(j in 1:nrun) lvec[j] = get_deviance(val_df$y,phat_list[[i]][,j])
    imin = which.min(lvec)
  phat_best[,i] = phat_list[[i]][,imin]
}
```

Each plot relates $\hat{p}$ to y. From left to right, $\hat{p}$ is from logit, random forests, and boosting.

```r
colnames(phat_best) = c("logit", "rf", "boost")
tempdf = data.frame(phat_best,y = val_df$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!

Expected Value

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 will extend credit. Expected values of classifiers is below:

```
conf_mat_lg = get_confusion_matrix(val_df$y, phat_best[,1], 0.2) # logit
print(conf_mat_lg, 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(conf_mat_lg) * cost_benefit)))
```

```
## Expected value of targeting using logistic regression =  563.75
```

```
conf_mat_rf = get_confusion_matrix(val_df$y, phat_best[,2], 0.2) # rf
print(conf_mat_rf, printStats = F)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 65298  2386
##          1  4721  2595
```

```
cat("Expected value of targeting using random forest = ",
sum(sum(as.matrix(conf_mat_rf) * cost_benefit)))
```

```
## Expected value of targeting using random forest =  1414.75
```

```
conf_mat_xgb = get_confusion_matrix(val_df$y, phat_best[,3], 0.2) # boosting
print(conf_mat_xgb, printStats=F)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 66079  2487
##          1  3940  2494
```

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

```
## Expected value of targeting using boosting =  1509
```

ROC Curves

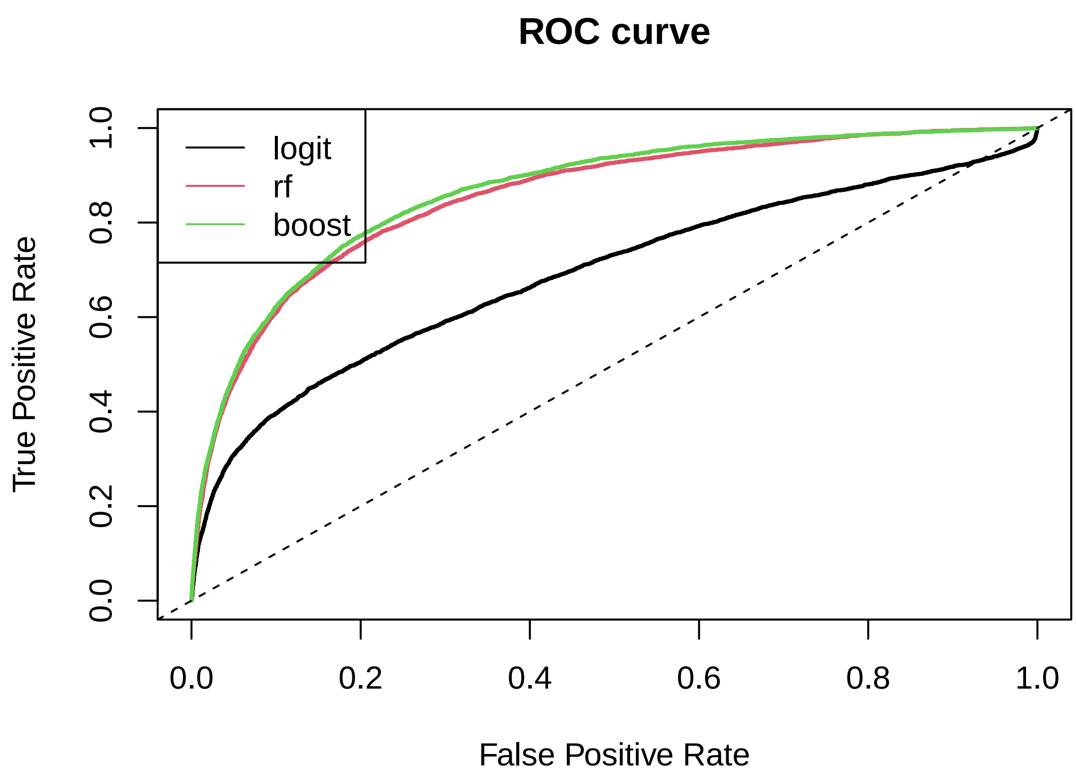
```
library(ROCR)
```

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

  if (i == 1) {
```

```r
    plot(perf, col=1, lwd=2,
    main= 'ROC curve',
    xlab='False Positive Rate',
    ylab='True Positive Rate')
  }

  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).

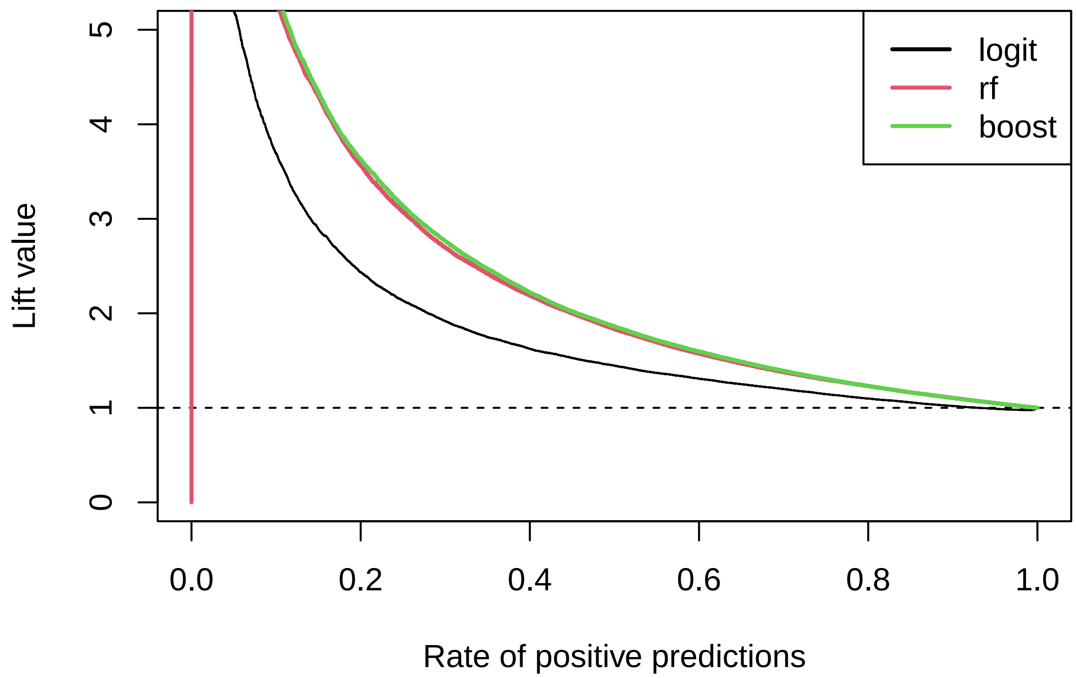
```r
for(i in 1:ncol(phat_best)) {
  pred = prediction(phat_best[,i], val_df$y)
  perf = performance(pred, measure = "auc")
  print(paste0("AUC ", names(phat_list)[i], " :: ", perf@y.values[[1]]))
}
```

```
## [1] "AUC logit :: 0.689726238272716"
## [1] "AUC rf :: 0.855209019627648"
## [1] "AUC boost :: 0.865111825456607"
```

Lift Curves

```
pred = prediction(phat_best[,1], val_df$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(phat_best)) {
  pred = prediction(phat_best[,i], val_df$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)
```



Cumulative Response

Similar to the ROC curve, the diagonal indicates the baseline. As we target more of the population, we target more of the positive cases.

```
pred = prediction(phat_best[,1], val_df$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(phat_best)) {
  pred = prediction(phat_best[,i], val_df$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)
```