Homework 4

Aaron Politsky

October 24, 2015

Load Libraries and source stuff.

```
library(caret)
library(data.table)
library(doParallel)
library(dplyr)

# load modules from the common HelpR repo
source('EvaluationMetrics.R')

# set randomizer's seed
set.seed(99)    # Gretzky was #99

# Set up Parallelization

cl <- makeCluster(detectCores())    # I don't mind using all of my cores
clusterEvalQ(cl, library(foreach))
```

```
## [[1]]
## [1] "foreach"   "methods"   "stats"     "graphics"  "grDevices" "utils"
## [7] "datasets"  "base"
##
## [[2]]
## [1] "foreach"   "methods"   "stats"     "graphics"  "grDevices" "utils"
## [7] "datasets"  "base"
##
## [[3]]
## [1] "foreach"   "methods"   "stats"     "graphics"  "grDevices" "utils"
## [7] "datasets"  "base"
##
## [[4]]
## [1] "foreach"   "methods"   "stats"     "graphics"  "grDevices" "utils"
## [7] "datasets"  "base"
```

```
registerDoParallel(cl)    # register this cluster

# download data and read data into data.table format

data_folder_path <- 'data'

# Common NAs:
na_strings <- c(
  '',
  'na', 'n.a', 'n.a.',
  'nan', 'n.a.n', 'n.a.n.',
  'NA', 'N.A', 'N.A.',
  'NaN', 'N.a.N', 'N.a.N.',
  'NAN', 'N.A.N', 'N.A.N.',
  'nil', 'Nil', 'NIL',
  'null', 'Null', 'NULL')

orange <- as.data.table(read.table(
  file.path("data", 'orange_small_train.data'),
  header=TRUE, sep='\t', stringsAsFactors=TRUE, na.strings=na_strings))

num.features <- ncol(orange)
input.feature.names <- names(orange)
num.samples <- nrow(orange)

churn <- factor(
  read.table(
    file.path(data_folder_path, 'orange_small_train_churn.labels.txt'),
    header=FALSE, sep='\t')[[1]],
  levels=c(-1, 1),
  labels=c('no', 'yes'))

appetency <- factor(
  read.table(
    file.path(data_folder_path, 'orange_small_train_appetency.labels.txt'),
    header=FALSE, sep='\t')[[1]],
  levels=c(-1, 1),
  labels=c('no', 'yes'))

upsell <- factor(
  read.table(
    file.path(data_folder_path, 'orange_small_train_upselling.labels.txt'),
    header=FALSE, sep='\t')[[1]],
  levels=c(-1, 1),
  labels=c('no', 'yes'))
```

Lets see the incidence of yes across these:

```
table(churn)
```

```
## churn
##    no   yes
## 46328  3672
```

```
table(appetency)
```

```
## appetency
##    no   yes
## 49110   890
```

```r
table(upsell)
```

```
## upsell
##    no   yes
## 46318  3682
```

Upselling seems to be the least rare of the three, which might make it easier to predict. Let's choose it and remove the others.

```r
remove(churn)
remove(appetency)
```

Split into our train, validation, and test sets

```r
train.proportion <- .8
train.indices <- createDataPartition(
  y=upsell,
  p=train.proportion,
  list=FALSE)

orange.train <- orange[train.indices, ]
orange.test <- orange[-train.indices, ]
upsell.train <- upsell[train.indices]
upsell.test <- upsell[-train.indices]

num.test.samples <- length(upsell.test)

# and Validation sets

valid.proportion <- .25
valid.indices <- createDataPartition(
  y=upsell.train,
  p=valid.proportion,
  list=FALSE)

orange.valid <- orange.train[valid.indices, ]
orange.train <- orange.train[-valid.indices, ]
upsell.valid <- upsell.train[valid.indices]
upsell.train <- upsell.train[-valid.indices]

num.train.samples <- length(upsell.train)
num.valid.samples <- length(upsell.valid)
```

Let's check how balanced our upsell rates are across these:

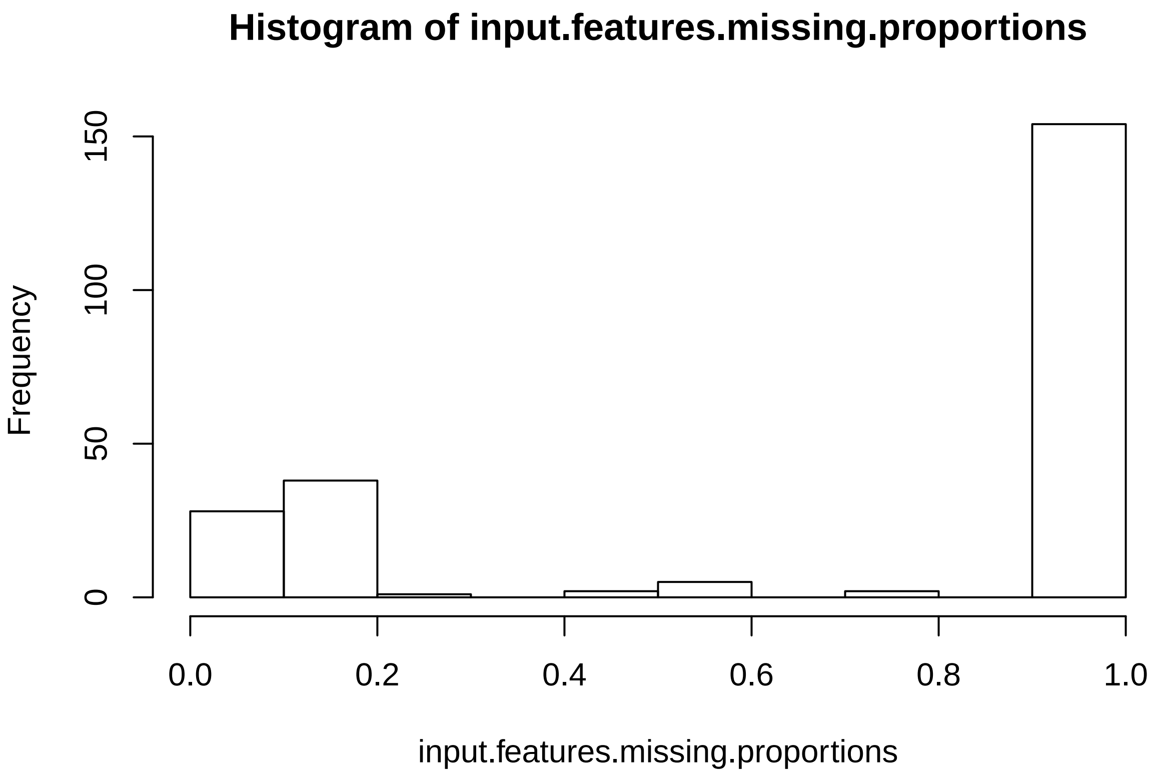
```r
lapply(list(upsell.train, upsell.valid, upsell.test), function(x) sum(x=='yes')/length(x))
```

```
## [[1]]
## [1] 0.07363333
##
## [[2]]
## [1] 0.07369263
##
## [[3]]
## [1] 0.07360736
```

Pretty balanced. Well done, caret.

Here we clean. This is very much in the spirit of the "get me outta here" code, but I tried to write things my own way, in some cases, with hopefully a cleaner/faster approach.

```
## Getting Rid of Input Features $x$'s with Too Many Missing Values

# examine the proportions of missing values per input feature column:

input.features.missing.proportions <-
  sapply(orange.train, function(col) sum(is.na(col))) / num.train.samples

hist(input.features.missing.proportions)
```



Based on this, we will remove the following features:

```
input.feature.names[input.features.missing.proportions > .2]
```

```
##   [1] "Var1"   "Var2"   "Var3"   "Var4"   "Var5"   "Var8"   "Var9"
##   [8] "Var10"  "Var11"  "Var12"  "Var14"  "Var15"  "Var16"  "Var17"
##  [15] "Var18"  "Var19"  "Var20"  "Var23"  "Var26"  "Var27"  "Var29"
##  [22] "Var30"  "Var31"  "Var32"  "Var33"  "Var34"  "Var36"  "Var37"
##  [29] "Var39"  "Var40"  "Var41"  "Var42"  "Var43"  "Var45"  "Var46"
##  [36] "Var47"  "Var48"  "Var49"  "Var50"  "Var51"  "Var52"  "Var53"
##  [43] "Var54"  "Var55"  "Var56"  "Var58"  "Var59"  "Var60"  "Var61"
##  [50] "Var62"  "Var63"  "Var64"  "Var66"  "Var67"  "Var68"  "Var69"
##  [57] "Var70"  "Var71"  "Var72"  "Var75"  "Var77"  "Var79"  "Var80"
##  [64] "Var82"  "Var84"  "Var86"  "Var87"  "Var88"  "Var89"  "Var90"
##  [71] "Var91"  "Var92"  "Var93"  "Var94"  "Var95"  "Var96"  "Var97"
##  [78] "Var98"  "Var99"  "Var100" "Var101" "Var102" "Var103" "Var104"
##  [85] "Var105" "Var106" "Var107" "Var108" "Var110" "Var111" "Var114"
##  [92] "Var115" "Var116" "Var117" "Var118" "Var120" "Var121" "Var122"
##  [99] "Var124" "Var126" "Var127" "Var128" "Var129" "Var130" "Var131"
## [106] "Var135" "Var136" "Var137" "Var138" "Var139" "Var141" "Var142"
## [113] "Var145" "Var146" "Var147" "Var148" "Var150" "Var151" "Var152"
## [120] "Var154" "Var155" "Var156" "Var157" "Var158" "Var159" "Var161"
## [127] "Var162" "Var164" "Var165" "Var166" "Var167" "Var168" "Var169"
## [134] "Var170" "Var171" "Var172" "Var174" "Var175" "Var176" "Var177"
## [141] "Var178" "Var179" "Var180" "Var182" "Var183" "Var184" "Var185"
## [148] "Var186" "Var187" "Var188" "Var189" "Var190" "Var191" "Var194"
## [155] "Var200" "Var201" "Var209" "Var213" "Var214" "Var215" "Var224"
## [162] "Var225" "Var229" "Var230"
```

Let's remove features which exceed 20% missing data.

```r
input.feature.names <-
  input.feature.names[input.features.missing.proportions <= .2]

num.input.features <- length(input.feature.names)

# Throw out unused columns
orange.train <- orange.train[ , input.feature.names, with=FALSE]

# Which of these are numeric and which are categorical?
input.feature.classes <- factor(sapply(orange.train, class))

input.feature.classes
```

```
##    Var6    Var7   Var13   Var21   Var22   Var24   Var25   Var28   Var35
## integer integer integer integer integer integer integer numeric integer
##   Var38   Var44   Var57   Var65   Var73   Var74   Var76   Var78   Var81
## integer integer numeric integer integer integer integer integer numeric
##   Var83   Var85  Var109  Var112  Var113  Var119  Var123  Var125  Var132
## integer integer integer integer numeric integer integer integer integer
##  Var133  Var134  Var140  Var143  Var144  Var149  Var153  Var160  Var163
## integer integer integer integer integer integer integer integer integer
##  Var173  Var181  Var192  Var193  Var195  Var196  Var197  Var198  Var199
## integer integer  factor  factor  factor  factor  factor  factor  factor
##  Var202  Var203  Var204  Var205  Var206  Var207  Var208  Var210  Var211
##  factor  factor  factor  factor  factor  factor  factor  factor  factor
##  Var212  Var216  Var217  Var218  Var219  Var220  Var221  Var222  Var223
##  factor  factor  factor  factor  factor  factor  factor  factor  factor
##  Var226  Var227  Var228
##  factor  factor  factor
## Levels: factor integer numeric
```

Replacing missing values: Numeric features

```
numeric.input.feature.names <-
  input.feature.names[input.feature.classes != 'factor']
```

Below, I attempt to improve upon style and execution speed using asdf as my copy of the orange.train data, just in case you're wondering what's going on.

```
orig.orange.train <- copy(orange.train)

# Lets check if any of these are constant across row
sd <-
  summarise_each(orange.train[,numeric.input.feature.names, with=F],
                 funs(sd(., na.rm=T)))

numeric.input.feature.means <-
  sapply(orange.train[ , numeric.input.feature.names, with=FALSE],
         function(col) mean(col, na.rm=TRUE))

asdf <- copy(orig.orange.train)
mus <- sapply(asdf[, numeric.input.feature.names, with=F],
              function(col) mean(col, na.rm=T))
```

My way:

```
system.time({
  lapply(numeric.input.feature.names,
         function(col) {
           na.rows <- is.na(asdf[[col]])
           if(sum(na.rows) > 0) {
             asdf[, col := replace(asdf[[col]], which(na.rows), mus[col]), with=F]
           }
         })
})
```

```
##    user  system elapsed
##   0.042   0.007   0.049
```

The instructor's style:

```
system.time({
  for (numeric.col in numeric.input.feature.names) {
    x <- orange.train[[numeric.col]]
    missing.value.row.yesno <- is.na(x)
    if (sum(missing.value.row.yesno) > 0) {
      orange.train[ , numeric.col := as.numeric(x), with=FALSE]
      mu <- numeric.input.feature.means[numeric.col]
      orange.train[missing.value.row.yesno, numeric.col := mu, with=FALSE]
    }
  }
}
)
```

```
##    user  system elapsed
##   0.058   0.004   0.063
```

How'd I do?

```
identical(asdf, orange.train)
```

```
## [1] TRUE
```

```
# check our work
all.equal(
  numeric.input.feature.means,
  sapply(orange.train[ , numeric.input.feature.names, with=FALSE], mean))
```

```
## [1] TRUE
```

Now we need to clean up the categorical data

```
#########
# Categorical features:  cleaning

categorical.input.feature.names <-
  input.feature.names[input.feature.classes == 'factor']

num.categorical.input.feature.levels <-
  sapply(orange.train[ , categorical.input.feature.names, with=FALSE],
         function(col) length(levels(col)))

num.categorical.input.feature.levels
```

```
## Var192 Var193 Var195 Var196 Var197 Var198 Var199 Var202 Var203 Var204
##    361     51     23      4    225   4291   5073   5713      5    100
## Var205 Var206 Var207 Var208 Var210 Var211 Var212 Var216 Var217 Var218
##      3     21     14      2      6      2     81   2016  13990      2
## Var219 Var220 Var221 Var222 Var223 Var226 Var227 Var228
##     22   4291      7   4291      4     23      7     30
```

```
# discard those with greater than 500 levels, as they're probably text.

categorical.input.feature.names <-
  categorical.input.feature.names[num.categorical.input.feature.levels <= 500]

orange.train <-
  orange.train[ , c(numeric.input.feature.names, categorical.input.feature.names), with=FALSE]

# let's make NA values its own category called zzzMissing

orig.orange.train <- copy(orange.train)
system.time(
  lapply(categorical.input.feature.names, function(colname) {
    col <- orange.train[[colname]]
    col <- addNA(col, ifany=T)
    levels(col)[is.na(levels(col))] <- 'zzzMISSING'
    orange.train[, colname := col, with=F]
  })
)
```

```
##    user  system elapsed
##   0.050   0.005   0.055
```

```
# For each categorical feature, collapse the long tail categories (less than 5%)
# keep track of the categories we collapsed, per feature

collapsed.categories.per.feature <-
  lapply(categorical.input.feature.names, function(x) {character()})

orig.categorical.input.feature.names <- categorical.input.feature.names

asdf <- copy(orange.train)


system.time( {
  lapply(categorical.input.feature.names, function(colname) {
    col <- asdf[[colname]]
    lapply(levels(col), function(level) {

      level.rows <-  col == level
      if(sum(level.rows) < 0.05 * length(col)) {
        collapsed.categories.per.feature[[colname]] <-
          c(collapsed.categories.per.feature[[colname]], level)
        asdf[level.rows, colname := 'zzzOTHER', with=F]
      }
    })
    col <- droplevels(col)
    asdf[,colname := col, with=F]
  })

  for(colname in categorical.input.feature.names) {
    # discard columns which have only one non-missing or non-other category
    levels <- levels(asdf[[colname]])
    if (length(levels[(levels != 'zzzMISSING') & (levels != 'zzzOTHER')]) < 2) {
      #removed.columns <- c(removed.columns, colname)
      categorical.input.feature.names <- setdiff(categorical.input.feature.names, colname)
    }
  }
}
)
```

```
##    user  system elapsed
##   2.873   0.205   3.093
```

```
my.cat.features <- categorical.input.feature.names


orig.orange.train <- copy(orange.train)

categorical.input.feature.names <- orig.categorical.input.feature.names

system.time({
  collapsed_categories <- list()
  for (cat_col in categorical.input.feature.names) {

    missing_value_row_yesno <- is.na(orange.train[[cat_col]])
    if (sum(missing_value_row_yesno) > 0) {
      orange.train[missing_value_row_yesno, cat_col := 'zzzMISSING', with=FALSE]
    }

    x <- orange.train[[cat_col]]
    for (cat in levels(x)) {
      cat_rows_yesno <- x == cat
      if (sum(cat_rows_yesno) < .05 * num.train.samples) {
        if (!(cat_col %in% names(collapsed_categories))) {
          collapsed_categories[[cat_col]] <- character()
        }
        collapsed_categories[[cat_col]] <- c(collapsed_categories[[cat_col]], cat)
        orange.train[cat_rows_yesno, cat_col := 'zzzOTHER', with=FALSE]
        levels(orange.train[[cat_col]])[levels(orange.train[[cat_col]]) == cat] <- NA
      }
    }

    cats <- levels(orange.train[[cat_col]])
    if ((length(cats) == 1) ||
        (length(cats[(cats != 'zzzMISSING') & (cats != 'zzzOTHER')]) < 2)) {
      categorical.input.feature.names <- setdiff(categorical.input.feature.names, cat_col)
    }
  }
})
```

```
##    user  system elapsed
##   4.378   0.436   4.828
```

```
identical(orange.train, asdf)
```

```
## [1] TRUE
```

```
identical(my.cat.features, categorical.input.feature.names)
```

```
## [1] TRUE
```

Choosing Input variables

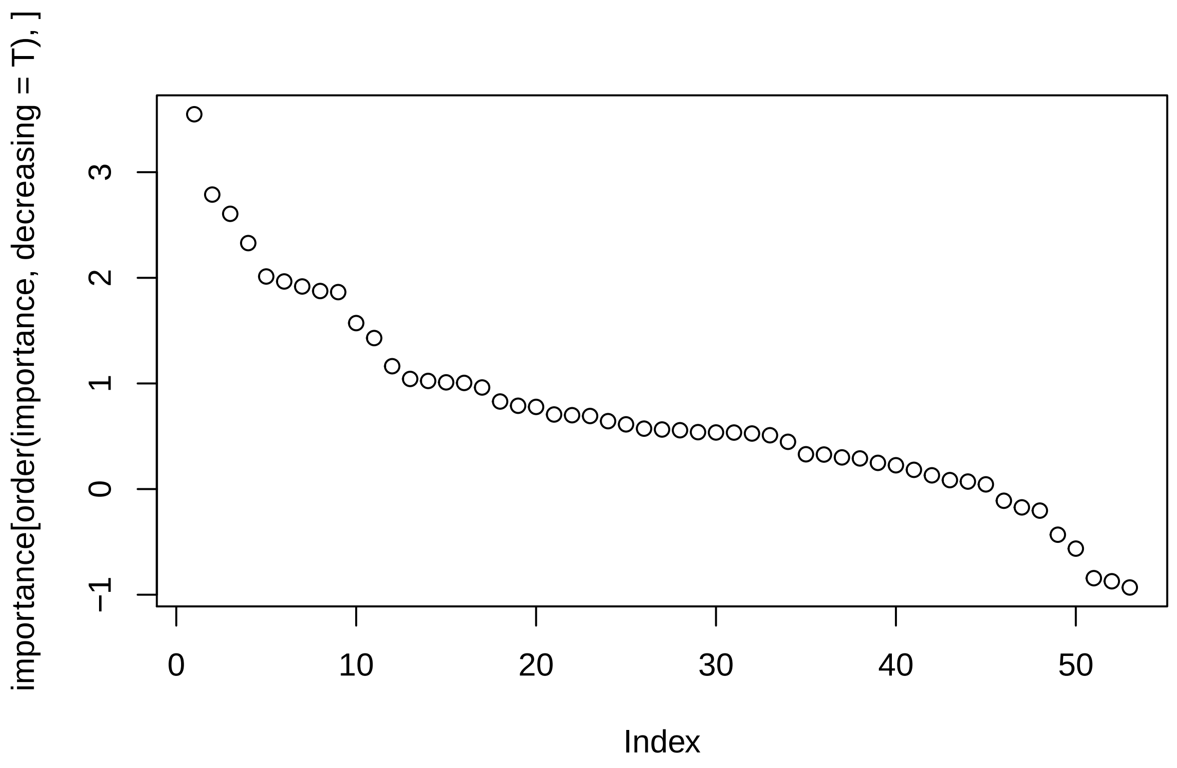
Now that we've cleaned our data, we want to choose input variables. I decided to pit the variables against each other in a random forest competition structure, sort of like major league baseball, but if a bunch of teams compete each game.

First will be the regular season. I split the features into four divisions and will have them "play" each other in a random forest per division. From each division forest we can assess importance. I then order the variables by importance, and 32 variables make the playoffs.

```
#########################################################################################
# Choosing input variables
input.feature.names <- c(numeric.input.feature.names, categorical.input.feature.names)

orange.train <- orange.train[, input.feature.names, with=F]

caret_optimized_metric <- 'logLoss'   # equivalent to 1 / 2 of Deviance

caret_train_control <- trainControl(
  classProbs=TRUE,             # compute class probabilities
  summaryFunction=mnLogLoss,   # equivalent to 1 / 2 of Deviance
  method='repeatedcv',         # repeated Cross Validation
  number=5,                    # number of folds
  repeats=1,                   # number of repeats
  allowParallel=TRUE)
```

```
# First, shuffle the order of features and set up divisions
set.seed(99)
competitors <- sample(input.feature.names)

north <- (1:13)
south <- (14:26)
east <- (27:39)
west <- (40:53)

B <- 500

importance <-data.frame()

# division play
division.rf.models <- list()
for(i in 1:4) {
  division <- list(north, south, east, west)[[i]]
  print(system.time(
    division.rf.models[[i]] <- train(
      x=orange.train[, competitors[division], with=FALSE],
      y=upsell.train,
      method='parRF',     # parallel Random Forest
      metric=caret_optimized_metric,
      ntree=B,            # number of trees in the Random Forest
      nodesize=100,       # minimum node size set small enough to allow for complex trees,
      # but not so small as to require too large B to eliminate high variance
      importance=T,       #  evaluate importance of predictors
      keep.inbag=FALSE,   # not relevant as we're using Cross Validation
      trControl=caret_train_control,
      tuneGrid=NULL)
  ))
  div.imp <- varImp(division.rf.models[[i]], scale = F)
  importance <- rbind(importance, div.imp$importance[2])
}
```

```
load("importance.Rda")
plot(importance[order(importance, decreasing = T),])
```



```r
playoff.teams <- rownames(importance)[order(importance$yes, decreasing=T)][1:32]
```

The top 32 “teams” (features) to make the playoffs are:

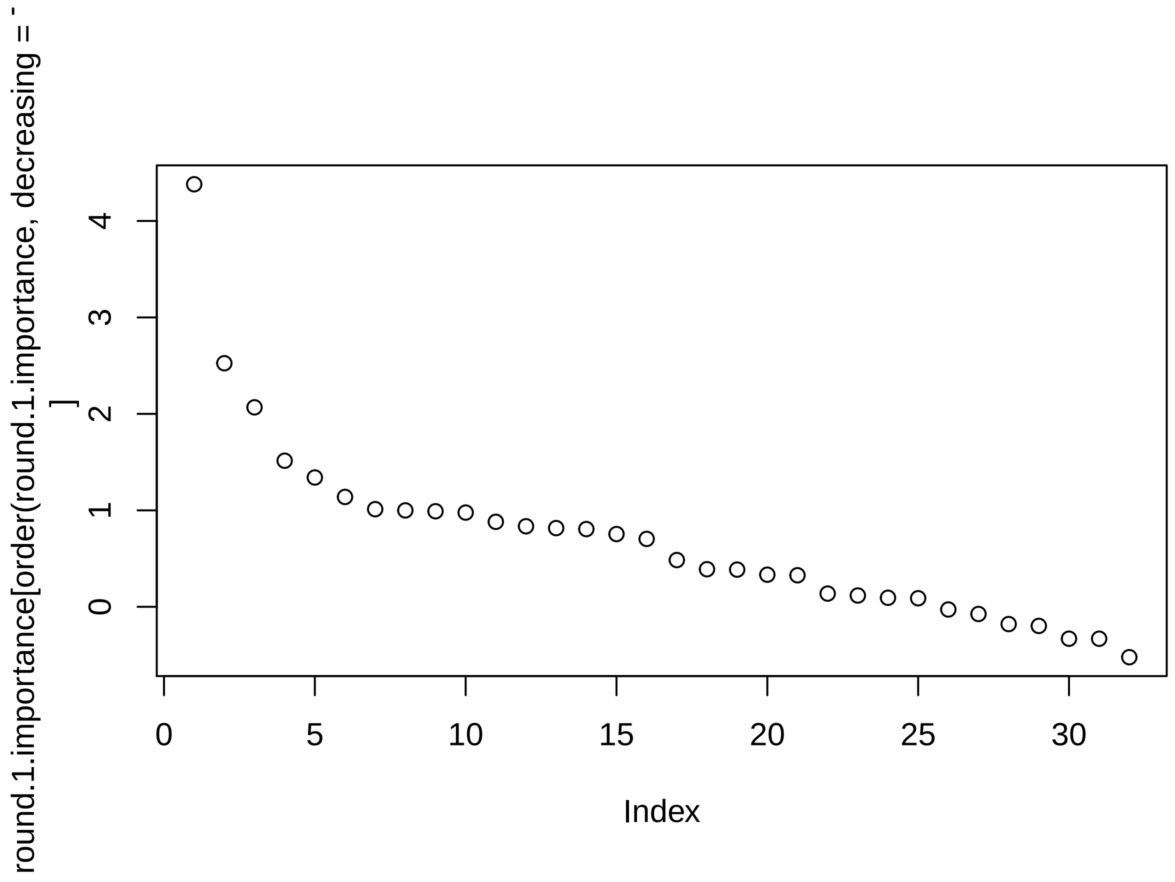
```r
playoff.teams
```

```
##  [1] "Var211" "Var218" "Var28"  "Var223" "Var113" "Var119" "Var81"
##  [8] "Var226" "Var181" "Var78"  "Var25"  "Var227" "Var125" "Var85"
## [15] "Var133" "Var221" "Var143" "Var163" "Var228" "Var132" "Var38"
## [22] "Var13"  "Var22"  "Var123" "Var160" "Var7"   "Var153" "Var193"
## [29] "Var207" "Var83"  "Var109" "Var74"
```

For the first round of the playoffs, I set up seeded groups, where top seed and bottom seed are in the same group, 2nd seed and 2nd worst seed are in another group, and so on. Then we let each group grow a forest, and see how each teams do.

```r
# Round 1:

group.a <- playoff.teams[c(seq(1,16,4),seq(32,17,-4))]
group.b <- playoff.teams[c(seq(2,16,4),seq(31,17,-4))]
group.c <- playoff.teams[c(seq(3,16,4),seq(30,17,-4))]
group.d <- playoff.teams[c(seq(4,16,4),seq(29,17,-4))]

B <- 300
set.seed(99)
round.1.importance <- data.frame()
round.1.rf.models <- list()
for(i in 1:4) {
  g <- list(group.a, group.b, group.c, group.d)[[i]]
  print(system.time(
    round.1.rf.models[[i]] <- train(
      x=orange.train[, g, with=FALSE],
      y=upsell.train,
      method='parRF',      # parallel Random Forest
      metric=caret_optimized_metric,
      ntree=B,             # number of trees in the Random Forest
      nodesize=100,        # minimum node size set small enough to allow for complex trees,
      # but not so small as to require too large B to eliminate high variance
      importance=T,        #  evaluate importance of predictors
      keep.inbag=FALSE,    # not relevant as we're using Cross Validation
      trControl=caret_train_control,
      tuneGrid=NULL)
  ))
  imp <- varImp(round.1.rf.models[[i]], scale = F)
  round.1.importance <- rbind(round.1.importance, imp$importance[2])
}
```

```r
load("round.1.importance.Rda")
plot(round.1.importance[order(round.1.importance, decreasing = T),])
```



```r
round.2.teams <-
  rownames(round.1.importance)[order(round.1.importance$yes, decreasing=T)][1:16]
```

Our advancing teams are

```
round.2.teams
```

```
##  [1] "Var211" "Var28"  "Var113" "Var38"  "Var181" "Var125" "Var123"
##  [8] "Var22"  "Var228" "Var119" "Var25"  "Var153" "Var218" "Var160"
## [15] "Var7"   "Var74"
```

Now on to the League Championship Series. Here we reseed the teams into the ALCS and NLCS and let them do their things.

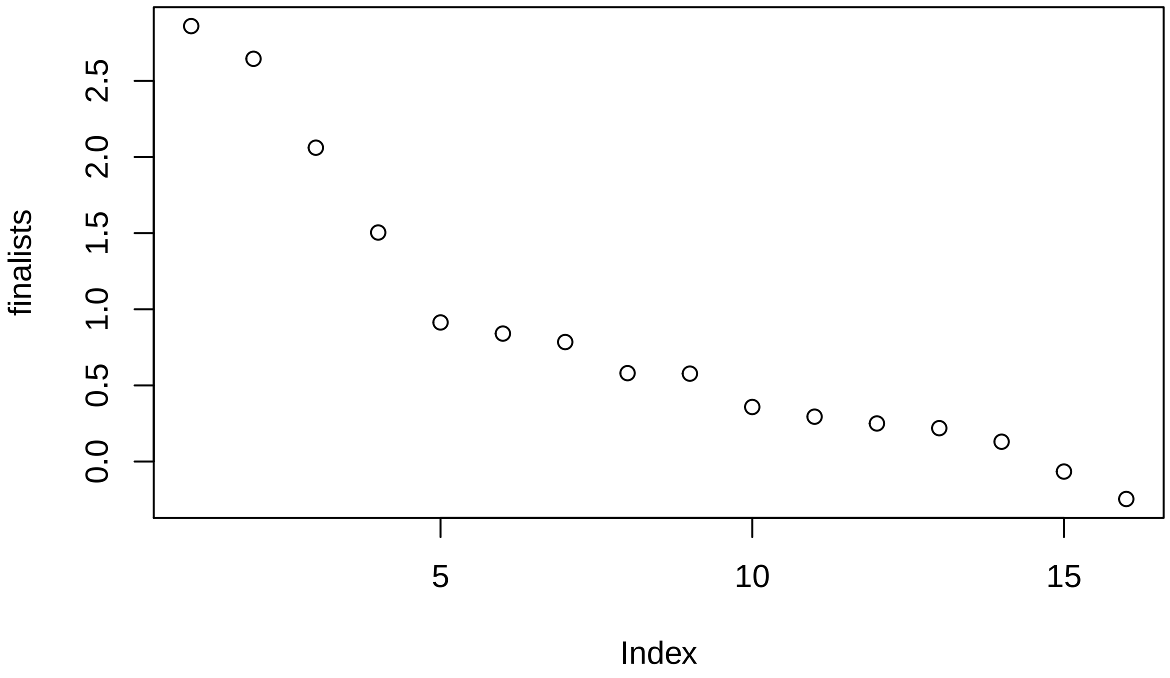
```
# Round 2
alcs <- round.2.teams[c(1,3,5,7,16,14,12,10)]
nlcs <- round.2.teams[c(2,4,6,8,15,13,11,9)]

B <- 300
set.seed(99)
round.2.importance <- data.frame()
round.2.rf.models <- list()
for(i in 1:2) {
  teams <- list(alcs, nlcs)[[i]]
  print(system.time(
    round.2.rf.models[[i]] <- train(
      x=orange.train[, teams, with=FALSE],
      y=upsell.train,
      method='parRF',     # parallel Random Forest
      metric=caret_optimized_metric,
      ntree=B,            # number of trees in the Random Forest
      nodesize=100,       # minimum node size set small enough to allow for complex trees,
      # but not so small as to require too large B to eliminate high variance
      importance=T,       #  evaluate importance of predictors
      keep.inbag=FALSE,   # not relevant as we're using Cross Validation
      trControl=caret_train_control,
      tuneGrid=NULL)
  ))
  imp <- varImp(round.2.rf.models[[i]], scale = F)
  round.2.importance <- rbind(round.2.importance, imp$importance[2])
}


finalists <-
  round.2.importance[order(round.2.importance, decreasing = T),]
names(finalists) <-
  rownames(round.2.importance)[order(round.2.importance$yes, decreasing=T)]
```

Our Finalists (and their importances) are

```
load("finalists.Rda")
finalists
```

```
##      Var211      Var28     Var218      Var74     Var228     Var153
##  2.85966836 2.64492842 2.06106973 1.50415765 0.91353240 0.84037019
##      Var125     Var119     Var181      Var25      Var22     Var113
##  0.78484767  0.58067842  0.57737336  0.35822150  0.29473417  0.25008886
##      Var160      Var123       Var38        Var7
##  0.21929890  0.13020338 -0.06576606 -0.24590767
```

```
plot(finalists)
```



```
# lets choose the ones that have positive importance.
champs <- finalists[finalists >= 0]
champs
```

```
##    Var211     Var28    Var218     Var74    Var228    Var153    Var125
## 2.8596684 2.6449284 2.0610697 1.5041577 0.9135324 0.8403702 0.7848477
##    Var119    Var181     Var25     Var22    Var113    Var160    Var123
## 0.5806784 0.5773734 0.3582215 0.2947342 0.2500889 0.2192989 0.1302034
```

What kind of predictors are we working with?

```
# what type of predictors are we working with?
input.feature.classes[names(champs)]
```

```
##  Var211   Var28  Var218   Var74  Var228  Var153  Var125  Var119  Var181
##  factor numeric  factor integer  factor integer integer integer integer
##   Var25   Var22  Var113  Var160  Var123
## integer integer numeric integer integer
## Levels: factor integer numeric
```

Some factors and some numerics.
Lets train our championship model. Here we no longer care about importance, but we still cross validate. (Note: training set includes validation set, so we are at once training on validation and train)

```
##################
# Fit our championship model
B <- 1000 #YOLO
set.seed(99)
print(system.time(
  champ.model <- train(
    x=orange.train[, names(champs), with=FALSE],
    y=upsell.train,
    method='parRF',      # parallel Random Forest
    metric=caret_optimized_metric,
    ntree=B,             # number of trees in the Random Forest
    nodesize=100,        # minimum node size set small enough to allow for complex trees,
    # but not so small as to require too large B to eliminate high variance
    importance=FALSE,        #  Don't much care now.
    keep.inbag=FALSE,    # not relevant as we're using Cross Validation
    trControl=caret_train_control,
    tuneGrid=NULL)
))
```

Now we need to make sure our test inputs don't have missing or unexpected values. (Note: shouldn't need to scrub valid set here as it's within train)

```
##########################################################################################
# Prep our validation and test data

champ.factors <- names(champs)[input.feature.classes[names(champs)]=='factor']

# NA values?
sapply(champ.factors, function(cf) sum(is.na(orange.valid[[cf]])))
```

```
## Var211 Var218 Var228
##      0    132      0
```

```
sapply(champ.factors, function(cf) sum(is.na(orange.test[[cf]])))
```

```
## Var211 Var218 Var228
##      0    150      0
```

```
# replace with missing tag
lapply(champ.factors, function(cf) {
  col <- orange.valid[[cf]]
  col <- addNA(col, ifany=T)
  levels(col)[is.na(levels(col))] <- 'zzzMISSING'
  orange.valid[, cf := col, with=F]
})
lapply(champ.factors, function(cf) {
  col <- orange.test[[cf]]
  col <- addNA(col, ifany=T)
  levels(col)[is.na(levels(col))] <- 'zzzMISSING'
  orange.test[, cf := col, with=F]
})
```

Are levels in our validation set not in our train set?

```r
lapply(champ.factors, function(cf) {
  levels(orange.valid[[cf]]) %in% levels(orange.train[[cf]])
})
```

```
## [[1]]
## [1] TRUE TRUE
##
## [[2]]
## [1]  TRUE  TRUE FALSE
##
## [[3]]
##  [1] FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE
## [12] FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [23] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

```r
lapply(champ.factors, function(cf) {
  levels(orange.test[[cf]]) %in% levels(orange.train[[cf]])
})
```

```
## [[1]]
## [1] TRUE TRUE
##
## [[2]]
## [1]  TRUE  TRUE FALSE
##
## [[3]]
##  [1] FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE
## [12] FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [23] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

```r
lapply(champ.factors, function(cf) {
  # for each level that is not in our orange.train, we collapse into 'zzzOTHER'
  levels.not.in.train <-
    levels(orange.valid[[cf]])[!(levels(orange.valid[[cf]]) %in%
                                  levels(orange.train[[cf]]))]
  for(l in levels.not.in.train) {
    rows <- orange.valid[[cf]] == l
    orange.valid[rows, cf := 'zzzOTHER', with=F]
  }
  orange.valid[,cf := droplevels(orange.valid[[cf]]), with=F]
})

lapply(champ.factors, function(cf) {
  # for each level that is not in ouor orange.train, we collapse into 'zzzOTHER'
  levels.not.in.train <-
    levels(orange.test[[cf]])[!(levels(orange.test[[cf]]) %in%
                                  levels(orange.train[[cf]]))]
  for(l in levels.not.in.train) {
    rows <- orange.test[[cf]] == l
    orange.test[rows, cf := 'zzzOTHER', with=F]
  }
  orange.test[,cf := droplevels(orange.test[[cf]]), with=F]
})
```

How about now?

```
lapply(champ.factors, function(cf) {
  levels(orange.valid[[cf]]) %in% levels(orange.train[[cf]])
  levels(orange.test[[cf]]) %in% levels(orange.train[[cf]])
})
```

```
## [[1]]
## [1] TRUE TRUE
##
## [[2]]
## [1] TRUE TRUE TRUE
##
## [[3]]
## [1] TRUE TRUE TRUE TRUE
```

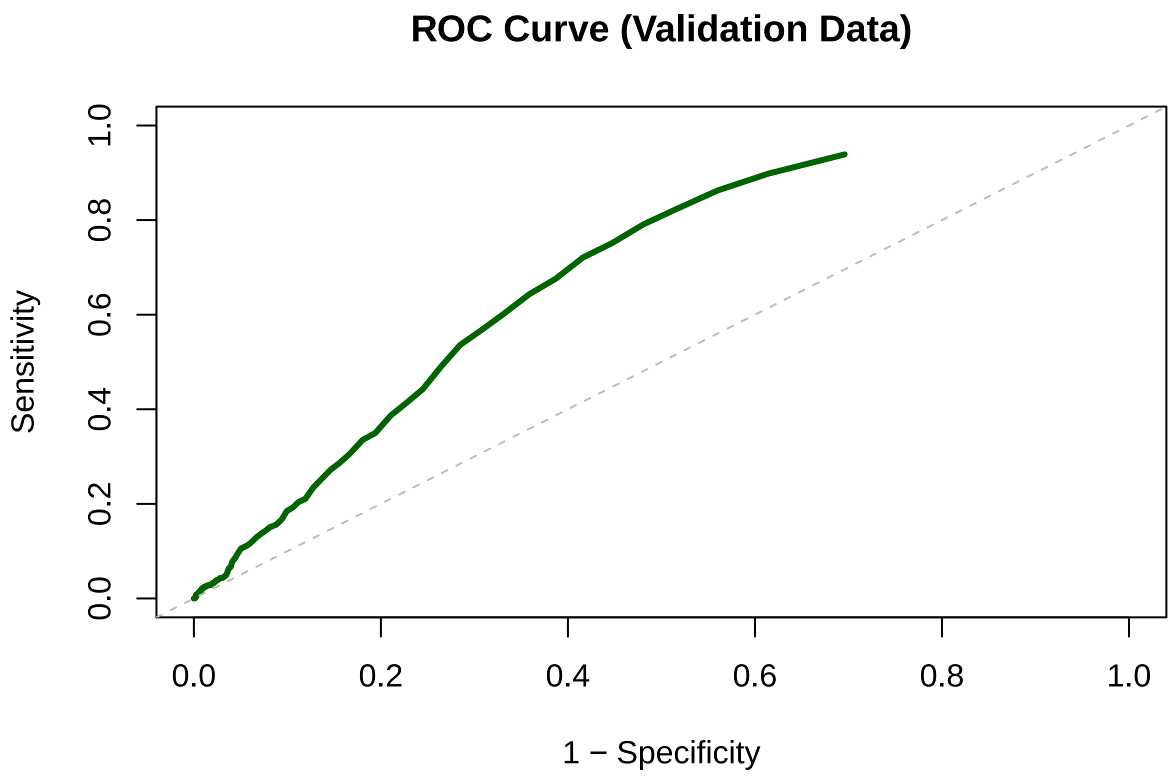
Good. Nothing more to prep with categorical features

```
prep.numeric.data <- function(data, means, numeric.cols) {
  lapply(
    names(data)[names(data) %in% numeric.cols],
    function(col) {
      na.rows <- is.na(data[[col]])
      if(sum(na.rows) > 0) {
        data[, col := replace(data[[col]], which(na.rows), means[col]), with=F]
      }
    })
  data
}

numeric.champs <- names(champs)[input.feature.classes[names(champs)] != 'factor']
champ.means <- numeric.input.feature.means[numeric.champs]

orange.valid <- orange.valid[, names(champs), with=F]
orange.valid <- prep.numeric.data(orange.valid, champ.means, numeric.champs)

orange.test <- orange.test[, names(champs), with=F]
orange.test <- prep.numeric.data(orange.test, champ.means, numeric.champs)
```

```
########################################################################################
# Validate

low_prob <- 1e-6
high_prob <- 1 - low_prob
log_low_prob <- log(low_prob)
log_high_prob <- log(high_prob)
log_prob_thresholds <- seq(from=log_low_prob, to=log_high_prob, length.out=1000)
prob_thresholds <- exp(log_prob_thresholds)
```

```r
champ.probs <- predict(champ.model, newdata = orange.valid, type="prob")
champ.perf <- bin_classif_eval(
  champ.probs$yes, upsell.valid, thresholds=prob_thresholds)
```

```r
load("champ.perf.Rda")
plot(x=1 - champ.perf$specificity,
     y=champ.perf$sensitivity,
     type = "l", col='darkgreen', lwd=3,
     xlim = c(0., 1.), ylim = c(0., 1.),
     main = "ROC Curve (Validation Data)",
     xlab = "1 - Specificity", ylab = "Sensitivity")
abline(a=0,b=1,lty=2,col=8)
```



Not bad.

```r
summary(champ.perf)
```

```
##    threshold           accuracy       sensitivity      specificity
##  Min.   :0.0000010   Min.   :0.3509   Min.   :0.0000   Min.   :0.3041
##  1st Qu.:0.0000316   1st Qu.:0.3509   1st Qu.:0.1384   1st Qu.:0.3041
##  Median :0.0010000   Median :0.3872   Median :0.9186   Median :0.3449
##  Mean   :0.0728110   Mean   :0.5552   Mean   :0.6370   Mean   :0.5487
##  3rd Qu.:0.0316233   3rd Qu.:0.8689   3rd Qu.:0.9389   3rd Qu.:0.9270
##  Max.   :0.9999990   Max.   :0.9263   Max.   :0.9389   Max.   :1.0000
##
##    precision          f1_score          deviance
##  Min.   :0.00000   Min.   :0.00266   Min.   :1.297
##  1st Qu.:0.09693   1st Qu.:0.17572   1st Qu.:1.297
##  Median :0.09693   Median :0.17572   Median :1.297
##  Mean   :0.10059   Mean   :0.15895   Mean   :1.297
##  3rd Qu.:0.11574   3rd Qu.:0.17572   3rd Qu.:1.297
##  Max.   :0.20000   Max.   :0.20955   Max.   :1.297
##  NA's   :43        NA's   :108
```

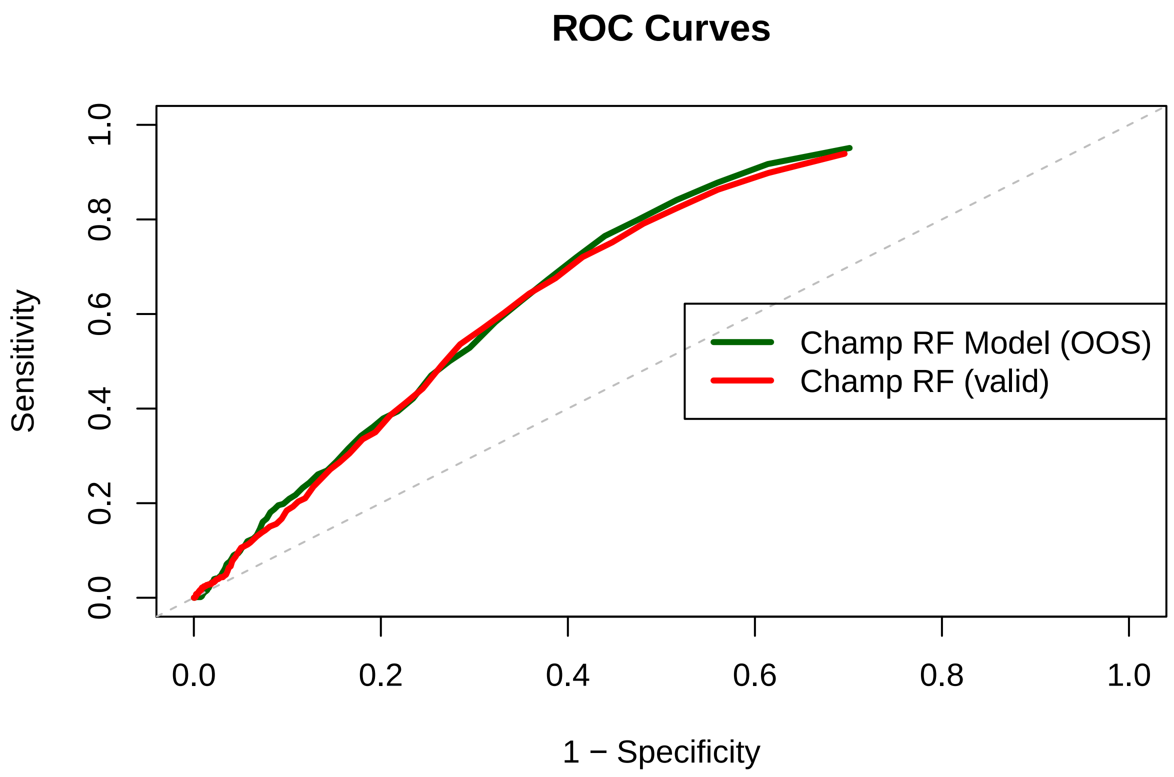
We can see that our min specificity is .3, so we don't have values on that area of the plot.

Out of Sample Performance

We trained on our train set, which includes our validation set, because valid.indices were parts of our train set. So we don't need to retrain here.

```
champ.probs.oos <- predict(champ.model, newdata = orange.test, type="prob")
champ.perf.oos <- bin_classif_eval(
  champ.probs.oos$yes, upsell.test, thresholds=prob_thresholds)
```

```
load("champ.perf.oos.Rda")

plot(x=1 - champ.perf.oos$specificity,
     y=champ.perf.oos$sensitivity,
     type = "l", col='darkgreen', lwd=3,
     xlim = c(0., 1.), ylim = c(0., 1.),
     main = "ROC Curves",
     xlab = "1 - Specificity", ylab = "Sensitivity")
abline(a=0,b=1,lty=2,col=8)
lines(x=1 - champ.perf$specificity,
      y=champ.perf$sensitivity,
      col='red', lwd=3)
legend('right', c('Champ RF Model (OOS)', 'Champ RF (valid)'),
       lty=1, col=c('darkgreen', 'red'), lwd=3, cex=1.)
```



```r
stopCluster(cl)    # shut down the parallel computing cluster
```