Orange benchmark

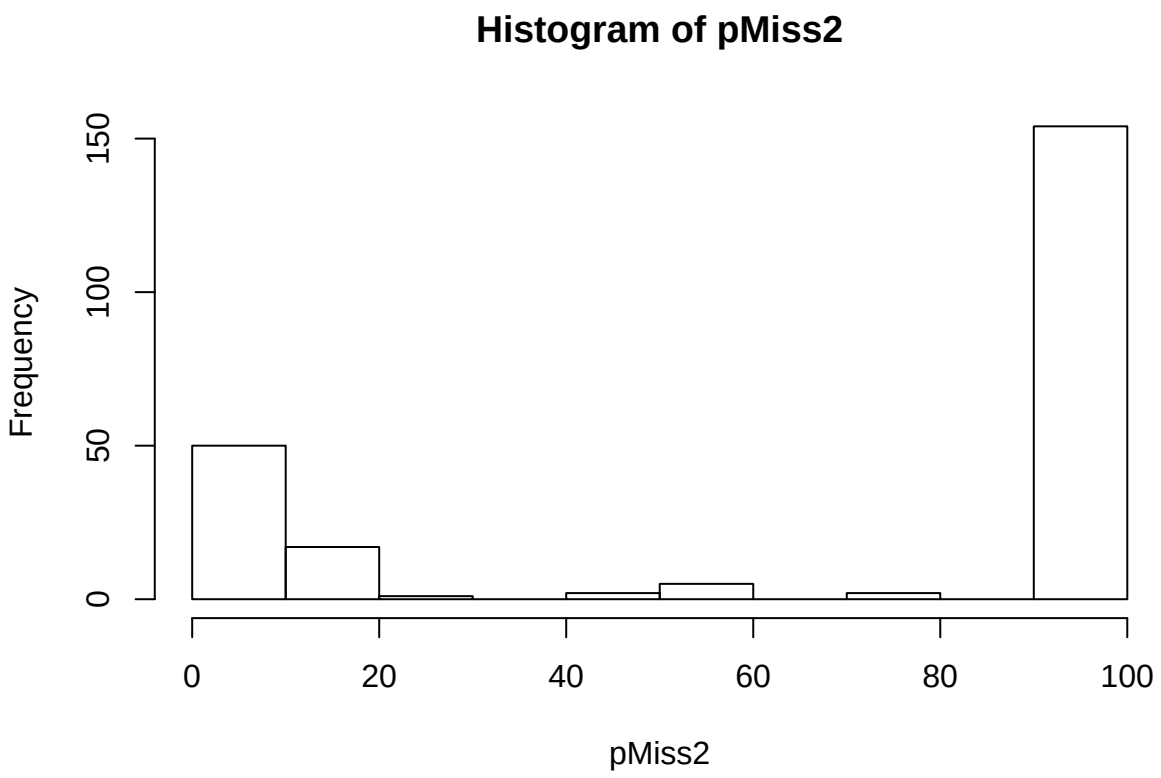
Import data

```
train <- read.csv("train.csv")
test <- read.csv("test.csv")
n = nrow(train)
m = nrow(test)
test$churn=NA
combi = rbind(train,test)
train = combi[1:n,]
test = combi[(n+1):(n+m),]
```

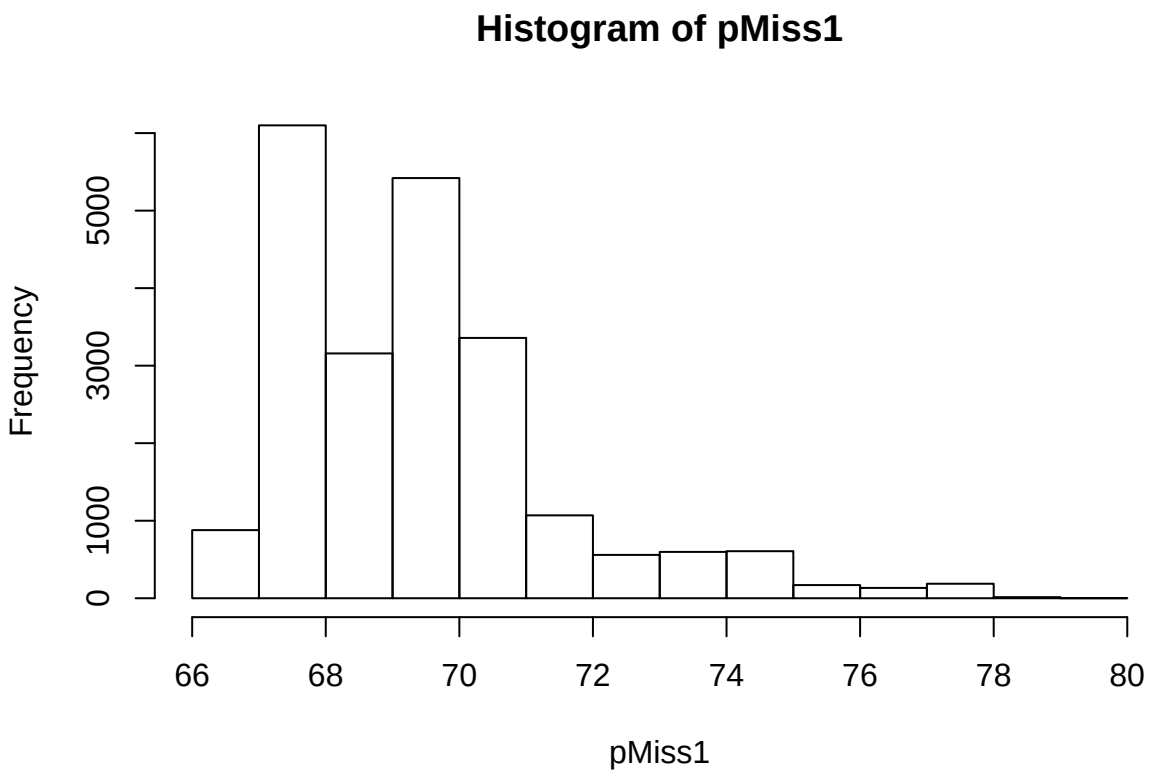
Missing values

```
pMiss <- function(x){sum(is.na(x))/length(x)*100}

pMiss2 = apply(train,2,pMiss)
hist(pMiss2)
```




```
pMiss1 = apply(train,1,pMiss)
hist(pMiss1)
```




```
vars_miss = which(pMiss2==100)
vars_miss
```

```
##   Var8  Var15  Var20  Var31  Var32  Var39  Var42  Var48  Var52  Var55
##      8     15     20     31     32     39     42     48     52     55
##  Var79 Var141 Var167 Var169 Var175 Var185 Var209 Var230
##     79    141    167    169    175    185    209    230
```

Zero- and Near Zero-Variance Predictors

In some situations, the data generating mechanism can create predictors that only have a single unique value (i.e. a "zero-variance predictor"). For many models (excluding tree-based models), this may cause the model to crash or the fit to be unstable.

To identify these types of predictors, the following two metrics can be calculated:

- the frequency of the most prevalent value over the second most frequent value (called the "frequency ratio"), which would be near one for well-behaved predictors and very large for highly-unbalanced data
- the "percent of unique values" is the number of unique values divided by the total number of samples (times 100) that approaches zero as the granularity of the data increases

If the frequency ratio is greater than a pre-specified threshold freqCut = 95/5 and the unique value percentage is less than a threshold uniqueCut = 10, we might consider a predictor to be near zero-variance.

```
library(caret)
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
vars_zv = nearZeroVar(train)
setdiff(vars_zv, vars_miss)
```

```
##  [1]   2   3   4   5  10  14  19  26  27  29  33  34  36  37  44  49  50
## [18]  51  53  56  58  59  67  69  70  78  80  84  86  90  93  95  98 106
## [35] 111 114 116 117 118 122 124 131 138 139 140 143 150 162 165 173 176
## [52] 177 182 183 191 194 195 196 201 210 213 215 219 224
```

Type of predictors

```
vars <- setdiff(names(train),c('churn',
                               names(train)[vars_zv]
                               ))
table(sapply(train[,vars],class))
```

```
##
##  factor integer numeric
##      28     106      14
```

```
# categorical and numerical
vars_cat <- vars[sapply(train[,vars],class) %in% c('factor','logical')]
vars_cat
```

```
##  [1] "Var192" "Var193" "Var197" "Var198" "Var199" "Var200" "Var202"
##  [8] "Var203" "Var204" "Var205" "Var206" "Var207" "Var208" "Var211"
## [15] "Var212" "Var214" "Var216" "Var217" "Var218" "Var220" "Var221"
## [22] "Var222" "Var223" "Var225" "Var226" "Var227" "Var228" "Var229"
```

```
vars_num<- vars[sapply(train[,vars],class) %in% c('numeric','integer')]
vars_num
```

```
##   [1] "Var1"   "Var6"   "Var7"   "Var9"   "Var11"  "Var12"  "Var13"
##   [8] "Var16"  "Var17"  "Var18"  "Var21"  "Var22"  "Var23"  "Var24"
##  [15] "Var25"  "Var28"  "Var30"  "Var35"  "Var38"  "Var40"  "Var41"
##  [22] "Var43"  "Var45"  "Var46"  "Var47"  "Var54"  "Var57"  "Var60"
##  [29] "Var61"  "Var62"  "Var63"  "Var64"  "Var65"  "Var66"  "Var68"
##  [36] "Var71"  "Var72"  "Var73"  "Var74"  "Var75"  "Var76"  "Var77"
##  [43] "Var81"  "Var82"  "Var83"  "Var85"  "Var87"  "Var88"  "Var89"
##  [50] "Var91"  "Var92"  "Var94"  "Var96"  "Var97"  "Var99"  "Var100"
##  [57] "Var101" "Var102" "Var103" "Var104" "Var105" "Var107" "Var108"
##  [64] "Var109" "Var110" "Var112" "Var113" "Var115" "Var119" "Var120"
##  [71] "Var121" "Var123" "Var125" "Var126" "Var127" "Var128" "Var129"
##  [78] "Var130" "Var132" "Var133" "Var134" "Var135" "Var136" "Var137"
##  [85] "Var142" "Var144" "Var145" "Var146" "Var147" "Var148" "Var149"
##  [92] "Var151" "Var152" "Var153" "Var154" "Var155" "Var156" "Var157"
##  [99] "Var158" "Var159" "Var160" "Var161" "Var163" "Var164" "Var166"
## [106] "Var168" "Var170" "Var171" "Var172" "Var174" "Var178" "Var179"
## [113] "Var180" "Var181" "Var184" "Var186" "Var187" "Var188" "Var189"
## [120] "Var190"
```

```
length(c(vars_cat, vars_num))
```

```
## [1] 148
```

Single categorical predictor

To create a table comparing the levels of variable 218 against the labeled churn outcome

```
# Tabulate levels of Var218.
table218 <- table(
   Var218=train[,'Var218'],
   churn=train[,'churn'],
   useNA='ifany')
print(table218)
```

```
##       churn
## Var218    -1     1
##   cJvF 10634   680
##   UYBR  9748   864
##   <NA>   238    89
```

```
print(table218[,2]/(table218[,1]+table218[,2]))
```

```
##       cJvF       UYBR       <NA>
## 0.06010253 0.08141726 0.27217125
```

This summary tells us that when variable 218 takes on a value of cJvF, around 6% of the customers churn; when it's UYBR, 8% of the customers churn; and when it's not recorded (NA), 27% of the customers churn. The utility of any variable level is a combination of how often the level occurs (rare levels aren't very useful) and how extreme the distribution of the outcome is for records matching a given level. Variable 218 seems like a predictor that's easy to use and helpful with prediction.

Calibration set

Split the data into three sets: training, calibration, and test.

The intent of the three-way split is this: we'll use the training set for most of our work, and we'll never look at the test set (we'll reserve it for our final report of model performance).

The calibration set is used to simulate the unseen test set during modeling - we'll look at performance on the calibration set to estimate if we're overfitting.

```
set.seed(123)
train.all = train
is.calib <- rbinom(n=nrow(train.all),size=1,prob=0.25)>0
# further split training data into training and calibration.
train = train.all[!is.calib,]
calib = train.all[is.calib,]
```

Scoring categorical predictors

We also need to design a strategy for what to do if a new level not seen during training were to occur during model use. We'll build a function that converts NA to a level (as it seems to be pretty informative) and also treats novel values as uninformative.

```
y = train$churn
x=train[ ,"Var218"]
xstar = calib[,'Var218']

# how often (%) y is positive for the training data
```

```
pPos <- sum(y==1)/length(y)
pPos
```

```
## [1] 0.07302934
```

```
# how often (%) y is positive for NA values for the training data
pPosNA <- prop.table(table(as.factor(y[is.na(x)])))["1"]
pPosNA
```

```
##         1
## 0.2510288
```

```
# how often (%) y is positive for the levels of the categorical predictor for the training data
tab <- table(as.factor(y),x)
tab
```

```
##     x
##      cJvF UYBR
##   -1 7999 7330
##   1   520  641
```

```
pPosLev <- (tab["1",]+1.0e-3*pPos)/(colSums(tab)+1.0e-3)
pPosLev
```

```
##       cJvF       UYBR
## 0.06104003 0.08041651
```

```
# compute predictons
pred <- pPosLev[xstar]
pred[1:10]
```

```
##       <NA>       UYBR       cJvF       UYBR       cJvF       cJvF
##         NA 0.08041651 0.06104003 0.08041651 0.06104003 0.06104003
##       cJvF       cJvF       UYBR       cJvF
## 0.06104003 0.06104003 0.08041651 0.06104003
```

```
# compute predictons for NA levels of xstar
pred[is.na(xstar)] <- pPosNA

# compute predictions for levels of xstar that weren't seen during training
pred[is.na(pred)] <- pPos
pred[1:10]
```

```
##       <NA>       UYBR       cJvF       UYBR       cJvF       cJvF
## 0.25102881 0.08041651 0.06104003 0.08041651 0.06104003 0.06104003
##       cJvF       cJvF       UYBR       cJvF
## 0.06104003 0.06104003 0.08041651 0.06104003
```

We calculate the score (the predicted probability of churning) for each categorical variable

```
# Given a vector of training response (y), a categorical training predictor (x), and a calibration cate
score_cat <- function(y,x,xstar){
  pPos <- sum(y==1)/length(y)
  pPosNA <- prop.table(table(as.factor(y[is.na(x)])))["1"]
  tab <- table(as.factor(y),x)
  pPosLev <- (tab["1",]+1.0e-3*pPos)/(colSums(tab)+1.0e-3)
  pred <- pPosLev[xstar]
  pred[is.na(xstar)] <- pPosNA
  pred[is.na(pred)] <- pPos
```

```
  pred
}



library('ROCR')
calcAUC <- function(phat,truth) {
  perf <- performance(prediction(phat,truth=="1"),'auc')
  as.numeric(perf@y.values)
}

# Once we have the predictions, we can find the categorical variables that have a good AUC both on the
for(v in vars_cat) {
  pi <- paste('score', v, sep='')
  train[,pi] <- score_cat(train$churn,train[,v],train[,v])
  calib[,pi] <- score_cat(train$churn,train[,v],calib[,v])
  test[,pi] <- score_cat(train$churn,train[,v],test[,v])
  train.auc <- calcAUC(train[,pi],train$churn)
  if(train.auc >= 0.8) {
    calib.auc <- calcAUC(calib[,pi],calib$churn)
    print(sprintf("%s, trainAUC: %4.3f calibrationAUC: %4.3f",
                  pi,train.auc,calib.auc))
  }
}
```

```
## [1] "scoreVar198, trainAUC: 0.819 calibrationAUC: 0.522"
## [1] "scoreVar199, trainAUC: 0.815 calibrationAUC: 0.565"
## [1] "scoreVar200, trainAUC: 0.841 calibrationAUC: 0.562"
## [1] "scoreVar202, trainAUC: 0.905 calibrationAUC: 0.536"
## [1] "scoreVar214, trainAUC: 0.841 calibrationAUC: 0.562"
## [1] "scoreVar217, trainAUC: 0.941 calibrationAUC: 0.575"
## [1] "scoreVar220, trainAUC: 0.819 calibrationAUC: 0.522"
## [1] "scoreVar222, trainAUC: 0.819 calibrationAUC: 0.522"
```

Many categorical variables that have a very large number of levels and are subject to overfitting. Note how, as expected, each predictor's training AUC is inflated compared to its calibration AUC. This is because many of these predictors have thousands of levels. For example

```
length(unique(train[,"Var217"]))
```

```
## [1] 7564
```

```
nlevels(train[,"Var217"])
```

```
## [1] 13990
```

A good trick to work around this is to sort the variables by their AUC score on the calibration set (not seen during training), which is a better estimate of the variable's true utility.

In our case, the most promising variable is variable 206

```
length(unique(train[,"Var206"]))
```

```
## [1] 22
```

```
nlevels(train[,"Var206"])
```

```
## [1] 21
```

```
tr206 <- score_cat(train$churn,train[,"Var206"],train[,"Var206"])
calcAUC(tr206,train$churn)
```

```
## [1] 0.5818733
```

```
ca206 <- score_cat(train$churn,train[,"Var206"],calib[,"Var206"])
calcAUC(ca206,calib$churn)
```

```
## [1] 0.5635558
```

Scoring numerical predictors

A common method is to bin the numeric predictor into a number of intervals and then use the intervals as a new categorical variable.

```
score_num <- function(y,x,xtilde){
  cuts <- unique(as.numeric(quantile(x,probs=seq(0, 1, 0.1),na.rm=T)))
  x.cut <- cut(x,cuts)
  xtilde.cut <- cut(xtilde,cuts)
  score_cat(y,x.cut,xtilde.cut)
}

for(v in vars_num) {
  pi <- paste('score',v,sep='')
  train[,pi] <- score_num(train$churn,train[,v],train[,v])
  calib[,pi] <- score_num(train$churn,train[,v],calib[,v])
  test[,pi] <- score_num(train$churn,train[,v],test[,v])
  train.auc <- calcAUC(train[,pi],train$churn)
  if(train.auc >= 0.55) {
    calib.auc <- calcAUC(calib[,pi],calib$churn)
    print(sprintf("%s, trainAUC: %4.3f calibrationAUC: %4.3f",
                  pi,train.auc,calib.auc))
   }
}
```

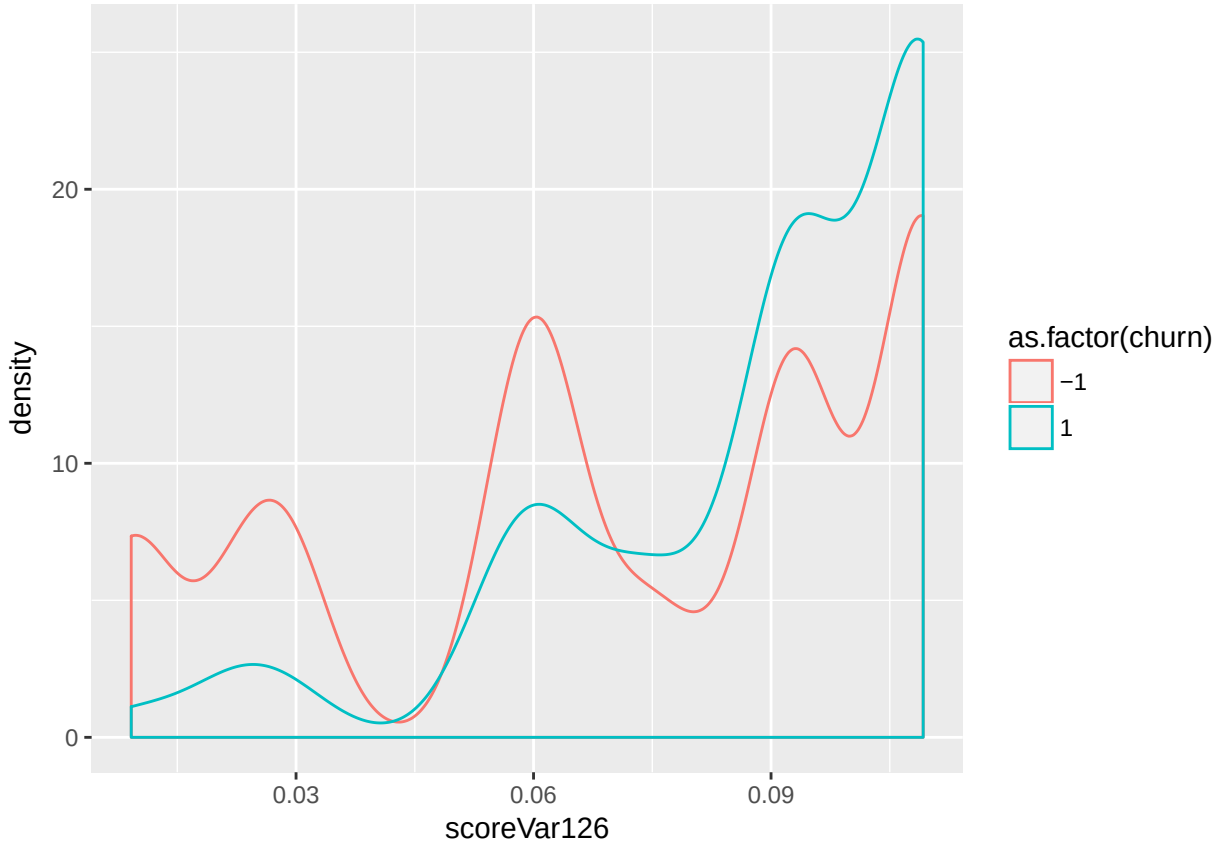
```
## [1] "scoreVar6, trainAUC: 0.556 calibrationAUC: 0.543"
## [1] "scoreVar7, trainAUC: 0.553 calibrationAUC: 0.551"
## [1] "scoreVar13, trainAUC: 0.570 calibrationAUC: 0.552"
## [1] "scoreVar28, trainAUC: 0.552 calibrationAUC: 0.530"
## [1] "scoreVar73, trainAUC: 0.601 calibrationAUC: 0.607"
## [1] "scoreVar74, trainAUC: 0.578 calibrationAUC: 0.560"
## [1] "scoreVar81, trainAUC: 0.551 calibrationAUC: 0.549"
## [1] "scoreVar113, trainAUC: 0.568 calibrationAUC: 0.512"
## [1] "scoreVar125, trainAUC: 0.552 calibrationAUC: 0.536"
## [1] "scoreVar126, trainAUC: 0.637 calibrationAUC: 0.646"
## [1] "scoreVar189, trainAUC: 0.570 calibrationAUC: 0.562"
```

Notice in this case the numeric predictors behave similarly on the training and calibration data. This is because our prediction method converts numeric variables into categorical variables with around 10 well-distributed levels, so our training estimate tends to be good and not overfit.

Plotting predictor performance

What the next figure is showing is the conditional distribution of predVar126 for churning accounts (red line) and the distribution of predVar126 for nonchurning accounts (blue line).

```
library(ggplot2)
ggplot(data=calib) +
  geom_density(aes(x=scoreVar126,color=as.factor(churn)))
```



We can deduce that low values of predVar126 are rare for churning accounts and not as rare for non-churning accounts (the graph is read by comparing areas under the curves).

Variable selection

A key part of building many variable models is selecting what predictors to use and how the predictors are to be transformed or treated.

An important evaluation of an estimated probability is the log likelihood. The log likelihood is the logarithm of the product of the probability the model assigned to each observation.

For an observation with churn=1 and an estimated probability of 0.9 of being churn, the log likelihood is log(0.9); for an observation with churn=-1, the same score of 0.9 is a log likelihood of log(1-0.9).

The null model has probability of churn = (number of churn clients)/(the total number of clients).

The score is a bit ad hoc, but tends to work well in selecting variables. Notice we're using performance on the calibration set (not the training set) to select predictors.

```
vars_score <- paste('score',c(vars_cat,vars_num),sep='')

# Define a convenience function to compute log likelihood.
loglik <- function(y,x) {
    sum(ifelse(y==1,log(x),log(1-x)))
}

vars_sel <- c()
cutoff <- 5
loglik0 <- loglik(calib$churn,
                  sum(calib$churn==1)/length(calib$churn)
                  )

# Run through categorical predictor and pick based on a deviance improvement (related to difference in
for(v in vars_cat) {
  pi <- paste('score',v,sep='')
  deviance <- 2*( (loglik(calib$churn,calib[,pi]) - loglik0))
  if(deviance>cutoff) {
    print(sprintf("%s, calibrationScore: %g",
                  pi,deviance))
    vars_sel <- c(vars_sel,pi)
  }
}
```

```
## [1] "scoreVar193, calibrationScore: 21.9554"
## [1] "scoreVar205, calibrationScore: 24.1322"
## [1] "scoreVar206, calibrationScore: 28.3265"
## [1] "scoreVar211, calibrationScore: 7.66146"
## [1] "scoreVar218, calibrationScore: 59.7186"
## [1] "scoreVar221, calibrationScore: 11.1505"
## [1] "scoreVar225, calibrationScore: 20.4016"
## [1] "scoreVar227, calibrationScore: 20.5856"
## [1] "scoreVar228, calibrationScore: 8.76659"
## [1] "scoreVar229, calibrationScore: 24.9989"
```

```
# Run through numerical predictor and pick based on a deviance improvement (related to difference in lo
for(v in vars_num) {
  pi <- paste('score',v,sep='')
  deviance <- 2*( (loglik(calib$churn,calib[,pi]) - loglik0))
  if(deviance>cutoff) {
    print(sprintf("%s, calibrationScore: %g",
                  pi,deviance))
    vars_sel <- c(vars_sel,pi)
  }
}
```

```
## [1] "scoreVar6, calibrationScore: 8.93739"
## [1] "scoreVar7, calibrationScore: 10.7107"
## [1] "scoreVar13, calibrationScore: 9.09552"
## [1] "scoreVar38, calibrationScore: 6.68585"
## [1] "scoreVar65, calibrationScore: 5.65415"
## [1] "scoreVar73, calibrationScore: 52.2945"
## [1] "scoreVar74, calibrationScore: 16.2695"
## [1] "scoreVar76, calibrationScore: 5.6245"
## [1] "scoreVar81, calibrationScore: 20.1007"
```

```
## [1] "scoreVar125, calibrationScore: 6.95256"
## [1] "scoreVar126, calibrationScore: 121.793"
## [1] "scoreVar134, calibrationScore: 7.92441"
## [1] "scoreVar144, calibrationScore: 5.87638"
## [1] "scoreVar153, calibrationScore: 5.18528"
## [1] "scoreVar189, calibrationScore: 32.3873"
```

Submission

```
library(rpart)
fml <- paste('churn == 1 ~ ',paste(vars_sel,collapse=' + '),sep='')
fml
```

```
## [1] "churn == 1 ~ scoreVar193 + scoreVar205 + scoreVar206 + scoreVar211 + scoreVar218 + scoreVar221 +
```

```
fit = rpart(fml, data=train)
phat = predict(fit, newdata=test)

write.table(file="mySubmission.txt", phat, row.names = FALSE, col.names = FALSE)
```