Simple\_Predictive\_Modeling\_with\_SWAT

# Loading the required SWAT package and other R libraries necessary  
library(swat)

## NOTE: The extension module for binary protocol support is not available.

## Only the CAS REST interface can be used.

## SWAT 1.4.0

library(ggplot2)  
library(reshape2)  
library(xgboost)

## Warning: package 'xgboost' was built under R version 3.4.4

library(caret)

## Warning: package 'caret' was built under R version 3.4.4

## Loading required package: lattice

library(dplyr)

## Warning: package 'dplyr' was built under R version 3.4.4

##   
## Attaching package: 'dplyr'

## The following object is masked from 'package:xgboost':  
##   
## slice

## The following objects are masked from 'package:stats':  
##   
## filter, lag

## The following objects are masked from 'package:base':  
##   
## intersect, setdiff, setequal, union

library(pROC)

## Type 'citation("pROC")' for a citation.

##   
## Attaching package: 'pROC'

## The following object is masked from 'package:swat':  
##   
## cov

## The following objects are masked from 'package:stats':  
##   
## cov, smooth, var

library(e1071)

## Warning: package 'e1071' was built under R version 3.4.4

library(ROCR)

## Warning: package 'ROCR' was built under R version 3.4.4

## Loading required package: gplots

## Warning: package 'gplots' was built under R version 3.4.4

##   
## Attaching package: 'gplots'

## The following object is masked from 'package:stats':  
##   
## lowess

# Connect to CAS server using appropriate credentials   
  
s = CAS()

## NOTE: Connecting to CAS and generating CAS action functions for loaded

## action sets...

## NOTE: To generate the functions with signatures (for tab completion), set

## options(cas.gen.function.sig=TRUE).

# Create a CAS library called lg pointing to the defined directory  
# Need to specify the srctype as path, otherwise it defaults to HDFS  
  
cas.table.addCaslib(s,  
 name = "lg",  
 description = "Looking glass data",  
 dataSource = list(srcType="path"),  
 path = "/viyafiles/tmp"  
 )

## NOTE: 'lg' is now the active caslib.

## NOTE: Cloud Analytic Services added the caslib 'lg'.

## $CASLibInfo  
## Name Type Description Path Definition Subdirs Local  
## 1 lg PATH Looking glass data /viyafiles/tmp/ 0 1  
## Active Personal Hidden Transient  
## 1 1 0 0 0

# Load the data into the in-memory CAS server  
  
data = cas.read.csv(s,   
 "C:/Users/Looking\_glass.csv",   
 casOut=list(name="castbl", caslib="lg", replace=TRUE)   
 )

## NOTE: Cloud Analytic Services made the uploaded file available as table CASTBL in caslib lg.

# Invoke the overloaded R functions to view the head and summary of the input table  
  
print(head(data))

## lifetime\_value calls\_in\_offpk mou\_onnet\_pct\_MOM mb\_data\_usg\_m01  
## 1 9616.9 604.38 0 1388.947  
## 2 7619.3 793.57 0 2930.470  
## 3 2765.7 529.50 0 69.000  
## 4 6426.5 333.39 1 1739.512  
## 5 5372.8 -16.42 0 1075.152  
## 6 1746.9 364.10 0 1191.598  
## mb\_data\_usg\_m02 mb\_data\_usg\_m03 region upsell\_xsell  
## 1 1243.291 1299.693 Pacific 0  
## 2 2856.150 3030.931 Southwest 0  
## 3 431.056 412.150 Mid Atlantic 0  
## 4 1766.006 1702.673 Midwest 0  
## 5 854.023 829.591 South 0  
## 6 1222.585 1254.263 Pacific 0  
## ever\_days\_over\_plan ever\_times\_over\_plan avg\_days\_susp  
## 1 2 6 6  
## 2 10 1 5  
## 3 9 2 0  
## 4 0 2 4  
## 5 11 5 2  
## 6 14 3 12  
## mou\_onnet\_6m\_normal unsolv\_tsupcomplnt wrk\_orders days\_openwrkorders  
## 1 0 0 0 15  
## 2 0 0 0 0  
## 3 -3 0 0 11  
## 4 -2 0 0 0  
## 5 1 1 0 16  
## 6 0 0 0 6

print(summary(data))

## Warning: package 'bindrcpp' was built under R version 3.4.4

## Selecting by Frequency

## lifetime\_value calls\_in\_offpk mou\_onnet\_pct\_MOM mb\_data\_usg\_m01   
## Min. :-14006 Min. :-1410.3 Min. :-45.0000 Min. :-2425.0   
## 1st Qu.: 1587 1st Qu.: 123.9 1st Qu.: -0.5280 1st Qu.: 540.2   
## Median : 3822 Median : 296.1 Median : 0.0000 Median : 1425.0   
## Mean : 5281 Mean : 388.6 Mean : -0.1368 Mean : 1697.2   
## 3rd Qu.: 7435 3rd Qu.: 545.5 3rd Qu.: 0.0000 3rd Qu.: 2417.2   
## Max. : 60740 Max. : 4640.2 Max. :124.7270 Max. :40568.7   
##   
## mb\_data\_usg\_m02 mb\_data\_usg\_m03 region   
## Min. :-2171.1 Min. :-1621.0 Great Lakes :10900   
## 1st Qu.: 538.7 1st Qu.: 535.2 South :10580   
## Median : 1431.1 Median : 1422.9 Mid Atlantic :10357   
## Mean : 1698.6 Mean : 1696.2 Pacific : 9157   
## 3rd Qu.: 2418.3 3rd Qu.: 2417.5 Greater Texas: 7236   
## Max. :40761.3 Max. :40784.2   
##   
## upsell\_xsell ever\_days\_over\_plan ever\_times\_over\_plan  
## Min. :0.0000 Min. : 0.00 Min. : 0.00   
## 1st Qu.:0.0000 1st Qu.: 0.00 1st Qu.: 0.00   
## Median :0.0000 Median : 9.00 Median : 2.00   
## Mean :0.1213 Mean :13.65 Mean : 2.53   
## 3rd Qu.:0.0000 3rd Qu.:22.00 3rd Qu.: 4.00   
## Max. :1.0000 Max. :99.00 Max. :26.00   
## NA's :58.00   
## avg\_days\_susp mou\_onnet\_6m\_normal unsolv\_tsupcomplnt wrk\_orders   
## Min. : 0.000 Min. :-27.1355 Min. :0.0000 Min. :0.000   
## 1st Qu.: 0.000 1st Qu.: -0.6147 1st Qu.:0.0000 1st Qu.:0.000   
## Median : 2.000 Median : 0.0000 Median :0.0000 Median :0.000   
## Mean : 3.474 Mean : -0.1175 Mean :0.6858 Mean :0.112   
## 3rd Qu.: 6.000 3rd Qu.: 0.0000 3rd Qu.:1.0000 3rd Qu.:0.000   
## Max. :62.000 Max. : 72.0113 Max. :5.0000 Max. :6.000   
##   
## days\_openwrkorders  
## Min. : 0.000   
## 1st Qu.: 0.000   
## Median : 0.000   
## Mean : 5.332   
## 3rd Qu.: 5.000   
## Max. : 99.000   
## NA's :155.000

# Check for any missingness in the data   
  
dist\_tabl = cas.simple.distinct(data)$Distinct[,c('Column','NMiss')]  
print(dist\_tabl)

## Column NMiss  
## 1 lifetime\_value 0  
## 2 calls\_in\_offpk 0  
## 3 mou\_onnet\_pct\_MOM 0  
## 4 mb\_data\_usg\_m01 0  
## 5 mb\_data\_usg\_m02 0  
## 6 mb\_data\_usg\_m03 0  
## 7 region 0  
## 8 upsell\_xsell 0  
## 9 ever\_days\_over\_plan 58  
## 10 ever\_times\_over\_plan 0  
## 11 avg\_days\_susp 0  
## 12 mou\_onnet\_6m\_normal 0  
## 13 unsolv\_tsupcomplnt 0  
## 14 wrk\_orders 0  
## 15 days\_openwrkorders 155

dist\_tabl = as.data.frame(dist\_tabl)  
sub = subset(dist\_tabl, dist\_tabl$NMiss != 0)  
imp\_cols = sub$Column  
  
# Print the names of the columns to be imputed   
print(imp\_cols)

## [1] "ever\_days\_over\_plan" "days\_openwrkorders"

# Impute the missing values   
  
cas.dataPreprocess.impute(data,  
 methodContinuous = 'MEDIAN',  
 methodNominal = 'MODE',  
 inputs = imp\_cols,  
 copyAllVars = TRUE,  
 casOut = list(name = 'castbl', replace = TRUE)  
 )

## $ImputeInfo  
## Variable ImputeTech ResultVar N NMiss  
## 1 ever\_days\_over\_plan Median IMP\_ever\_days\_over\_plan 56498 58  
## 2 days\_openwrkorders Median IMP\_days\_openwrkorders 56401 155  
## ImputedValueContinuous  
## 1 9  
## 2 0  
##   
## $OutputCasTables  
## casLib Name Rows Columns  
## 1 lg castbl 56556 17

# Split the data into training and validation and view the partitioned table  
  
loadActionSet(s,"sampling")

## NOTE: Added action set 'sampling'.

## NOTE: Information for action set 'sampling':

## NOTE: sampling

## NOTE: srs - Samples a proportion of data from the input table or partitions the data into no more than three portions

## NOTE: stratified - Samples a proportion of data or partitions the data into no more than three portions within each stratum

## NOTE: oversample - Samples a user-specified proportion of data from the event level and adjusts the ratio between rare events and non-rare events to a user-specified ratio

## NOTE: kfold - K-fold partitioning.

cas.sampling.srs( s,  
 table = list(name="castbl", caslib="lg"),  
 samppct = 30,  
 seed = 123456,  
 partind = TRUE,  
 output = list(casOut = list(name = "sampled\_castbl", replace = T, caslib="lg"), copyVars = 'ALL')  
 )

## NOTE: Using SEED=123456 for sampling.

## $OutputCasTables  
## casLib Name Label Rows Columns  
## 1 lg sampled\_castbl 56556 18  
##   
## $SRSFreq  
## NObs NSamp  
## 1 56556 16967  
##   
## $outputSize  
## $outputSize$outputNObs  
## [1] 56556  
##   
## $outputSize$outputNVars  
## [1] 18

# Check for frequency distribution of partitioned data  
  
cas.simple.freq(s,table="sampled\_castbl", inputs="\_PartInd\_")

## $Frequency  
## Column NumVar FmtVar Level Frequency  
## 1 \_PartInd\_ 0 0 1 39589  
## 2 \_PartInd\_ 1 1 2 16967

# Partition data into train and validation based on \_PartInd\_  
  
train = defCasTable(s, tablename = "sampled\_castbl", where = " \_PartInd\_ = 0 ")  
  
val = defCasTable(s, tablename = "sampled\_castbl", where = " \_PartInd\_ = 1 ")

# Create the appropriate input and target variables  
  
info = cas.table.columnInfo(s, table = train)  
  
colinfo = info$ColumnInfo  
  
## nominal variables are: region, upsell\_xsell  
  
nominals = colinfo$Column[c(7,8)]  
  
intervals = colinfo$Column[c(-7,-8,-9,-15,-18)]  
  
target = colinfo$Column[8]  
  
inputs = colinfo$Column[c(-8,-9,-15,-18)]

# Build a GB model for predictive classification  
  
loadActionSet(s, "decisionTree")

## NOTE: Added action set 'decisionTree'.

## NOTE: Information for action set 'decisionTree':

## NOTE: decisionTree

## NOTE: dtreeTrain - Trains a decision tree

## NOTE: dtreeScore - Scores a table using a decision tree model

## NOTE: dtreeSplit - Splits decision tree nodes

## NOTE: dtreePrune - Prune a decision tree

## NOTE: dtreeMerge - Merges decision tree nodes

## NOTE: dtreeCode - Generates DATA step scoring code from a decision tree model

## NOTE: forestTrain - Trains a forest

## NOTE: forestScore - Scores a table using a forest model

## NOTE: forestCode - Generates DATA step scoring code from a forest model

## NOTE: gbtreeTrain - Trains a gradient boosting tree

## NOTE: gbtreeScore - Scores a table using a gradient boosting tree model

## NOTE: gbtreeCode - Generates DATA step scoring code from a gradient boosting tree model

model = cas.decisionTree.gbtreeTrain(  
 s,  
 casOut=list(caslib="lg",name="gb\_model",replace=T),   
 inputs = inputs,  
 nominals = nominals,   
 target = target,  
 table = train  
 )  
  
# View the model info  
  
print(model)

## $ModelInfo  
## Descr Value  
## 1 Number of Trees 50.0  
## 2 Distribution 2.0  
## 3 Learning Rate 0.1  
## 4 Subsampling Rate 0.5  
## 5 Number of Selected Variables (M) 14.0  
## 6 Number of Bins 20.0  
## 7 Number of Variables 14.0  
## 8 Max Number of Tree Nodes 63.0  
## 9 Min Number of Tree Nodes 35.0  
## 10 Max Number of Branches 2.0  
## 11 Min Number of Branches 2.0  
## 12 Max Number of Levels 6.0  
## 13 Min Number of Levels 6.0  
## 14 Max Number of Leaves 32.0  
## 15 Min Number of Leaves 18.0  
## 16 Maximum Size of Leaves 18958.0  
## 17 Minimum Size of Leaves 5.0  
## 18 Random Number Seed 0.0  
##   
## $OutputCasTables  
## casLib Name Rows Columns  
## 1 lg gb\_model 2738 35

# Score the model on test data  
  
out = cas.decisionTree.gbtreeScore (   
 s,   
 modelTable = list(name="gb\_model", caslib="lg"),  
 table = val,  
 encodeName = TRUE,   
 assessonerow = TRUE,  
 casOut = list(name="scored\_data", caslib="lg", replace=T),  
 copyVars = target  
 )  
  
# View the scored results  
  
cas.table.fetch(s,table="scored\_data")

## $Fetch  
## \_Index\_ upsell\_xsell I\_upsell\_xsell \_MissIt\_ P\_upsell\_xsell1  
## 1 1 0 0 0 0.05895339  
## 2 2 0 0 0 0.21791656  
## 3 3 0 0 0 0.11431888  
## 4 4 0 0 0 0.04365430  
## 5 5 0 0 0 0.12564108  
## 6 6 0 0 0 0.04626239  
## 7 7 0 0 0 0.05319016  
## 8 8 0 0 0 0.05704202  
## 9 9 0 0 0 0.05970662  
## 10 10 0 0 0 0.05075295  
## 11 11 1 0 1 0.37962623  
## 12 12 0 0 0 0.06068739  
## 13 13 0 0 0 0.05101849  
## 14 14 0 0 0 0.05950617  
## 15 15 0 0 0 0.28303696  
## 16 16 1 0 1 0.22854141  
## 17 17 0 1 1 0.64259389  
## 18 18 0 0 0 0.04994233  
## 19 19 0 0 0 0.05491052  
## 20 20 0 0 0 0.04607341  
## P\_upsell\_xsell0  
## 1 0.9410466  
## 2 0.7820834  
## 3 0.8856811  
## 4 0.9563457  
## 5 0.8743589  
## 6 0.9537376  
## 7 0.9468098  
## 8 0.9429580  
## 9 0.9402934  
## 10 0.9492470  
## 11 0.6203738  
## 12 0.9393126  
## 13 0.9489815  
## 14 0.9404938  
## 15 0.7169630  
## 16 0.7714586  
## 17 0.3574061  
## 18 0.9500577  
## 19 0.9450895  
## 20 0.9539266

# Train an R Extreme Gradient Boosting model  
  
# First, convert the train and test CAS tables to R data frames for training the R-XGB model  
train\_cas\_df = to.casDataFrame(train)  
train\_df = to.data.frame(train\_cas\_df)  
  
val\_cas\_df = to.casDataFrame(val)  
val\_df = to.data.frame(val\_cas\_df)  
  
# In R, we need to do the data pre-processing explicitly. Hence, convert the "char" region variable to "factor"  
train\_df$region = as.factor(train\_df$region)  
val\_df$region = as.factor(val\_df$region)  
  
# For XGB model, it requires the input to be numeric. Hence, convert cateogrical variables into numeric using 1-hot encoding   
train\_dmy = dummyVars(" ~ .", data = train\_df,fullRank = T)  
val\_dmy = dummyVars(" ~ .", data = val\_df,fullRank = T)  
  
prep\_train = data.frame(predict(train\_dmy, newdata = train\_df))  
prep\_val = data.frame(predict(val\_dmy, newdata = val\_df))  
  
print(head(prep\_train))

## lifetime\_value calls\_in\_offpk mou\_onnet\_pct\_MOM mb\_data\_usg\_m01  
## 1 9616.9 604.38 0 1388.947  
## 2 7619.3 793.57 0 2930.470  
## 3 2765.7 529.50 0 69.000  
## 4 6426.5 333.39 1 1739.512  
## 5 5372.8 -16.42 0 1075.152  
## 6 1746.9 364.10 0 1191.598  
## mb\_data\_usg\_m02 mb\_data\_usg\_m03 region.Greater.Texas region.Mid.Atlantic  
## 1 1243.291 1299.693 0 0  
## 2 2856.150 3030.931 0 0  
## 3 431.056 412.150 0 1  
## 4 1766.006 1702.673 0 0  
## 5 854.023 829.591 0 0  
## 6 1222.585 1254.263 0 0  
## region.Midwest region.Mtn.West region.New.England region.Pacific  
## 1 0 0 0 1  
## 2 0 0 0 0  
## 3 0 0 0 0  
## 4 1 0 0 0  
## 5 0 0 0 0  
## 6 0 0 0 1  
## region.South region.Southwest upsell\_xsell ever\_days\_over\_plan  
## 1 0 0 0 2  
## 2 0 1 0 10  
## 3 0 0 0 9  
## 4 0 0 0 0  
## 5 1 0 0 11  
## 6 0 0 0 14  
## ever\_times\_over\_plan avg\_days\_susp mou\_onnet\_6m\_normal  
## 1 6 6 0  
## 2 1 5 0  
## 3 2 0 -3  
## 4 2 4 -2  
## 5 5 2 1  
## 6 3 12 0  
## unsolv\_tsupcomplnt wrk\_orders days\_openwrkorders IMP\_days\_openwrkorders  
## 1 0 0 15 15  
## 2 0 0 0 0  
## 3 0 0 11 11  
## 4 0 0 0 0  
## 5 1 0 16 16  
## 6 0 0 6 6  
## IMP\_ever\_days\_over\_plan X.\_PartInd\_.  
## 1 2 0  
## 2 10 0  
## 3 9 0  
## 4 0 0  
## 5 11 0  
## 6 14 0

print(head(prep\_val))

## lifetime\_value calls\_in\_offpk mou\_onnet\_pct\_MOM mb\_data\_usg\_m01  
## 1 9165.1 1320.14 0 6813.458  
## 2 1892.9 8.70 5 1584.943  
## 3 9672.0 192.61 0 2924.855  
## 4 5704.2 120.76 0 1353.099  
## 5 5472.7 389.50 0 1864.386  
## 6 6576.7 259.17 -1 1396.245  
## mb\_data\_usg\_m02 mb\_data\_usg\_m03 region.Greater.Texas region.Mid.Atlantic  
## 1 6826.472 6992.660 0 0  
## 2 1695.293 1581.966 0 0  
## 3 2821.905 2764.459 0 0  
## 4 1308.704 1413.062 0 1  
## 5 1799.559 1947.918 0 0  
## 6 1273.867 1536.013 0 0  
## region.Midwest region.Mtn.West region.New.England region.Pacific  
## 1 0 0 0 1  
## 2 0 0 0 0  
## 3 0 0 0 0  
## 4 0 0 0 0  
## 5 0 0 0 1  
## 6 0 0 0 0  
## region.South region.Southwest upsell\_xsell ever\_days\_over\_plan  
## 1 0 0 0 0  
## 2 1 0 0 26  
## 3 1 0 0 15  
## 4 0 0 0 22  
## 5 0 0 0 1  
## 6 1 0 0 31  
## ever\_times\_over\_plan avg\_days\_susp mou\_onnet\_6m\_normal  
## 1 7 1 0  
## 2 0 3 -8  
## 3 6 4 0  
## 4 0 3 0  
## 5 7 12 -1  
## 6 0 0 0  
## unsolv\_tsupcomplnt wrk\_orders days\_openwrkorders IMP\_days\_openwrkorders  
## 1 3 0 0 0  
## 2 1 0 3 3  
## 3 1 0 0 0  
## 4 2 0 18 18  
## 5 1 0 11 11  
## 6 2 0 0 0  
## IMP\_ever\_days\_over\_plan X.\_PartInd\_.  
## 1 0 1  
## 2 26 1  
## 3 15 1  
## 4 22 1  
## 5 1 1  
## 6 31 1

# Convert the target variable to categorical  
  
train\_labels = as.numeric(as.factor(prep\_train$upsell\_xsell)) - 1  
test\_labels = as.numeric(as.factor(prep\_val$upsell\_xsell)) - 1  
  
prep\_train$upsell\_xsell = NULL  
prep\_val$upsell\_xsell = NULL

# Train a XGBoost model on the data   
  
xgb = xgboost(data = data.matrix(prep\_train[,-1]),   
 label = train\_labels,   
 nround=2,   
 objective = "binary:logistic"  
 )

## [1] train-error:0.095154   
## [2] train-error:0.093048

# Make predictions on test data  
  
pred = predict(xgb, newdata= data.matrix(prep\_val[,-1]))

# Evaluate the performance of SAS and R models  
  
## Assessing the performance metric of SAS-GB model  
  
loadActionSet(s,"percentile")

## NOTE: Added action set 'percentile'.

## NOTE: Information for action set 'percentile':

## NOTE: percentile

## NOTE: percentile - Calculate quantiles and percentiles

## NOTE: boxPlot - Calculate quantiles, high and low whiskers, and outliers

## NOTE: assess - Assess and compare models

tmp = cas.percentile.assess(   
 s,   
 cutStep = 0.05,  
 event = "1",   
 inputs = "P\_upsell\_xsell1",  
 nBins = 20,  
 response = target,  
 table = "scored\_data"   
   
 )$ROCInfo  
  
roc\_df = data.frame(tmp)  
print(head(roc\_df))

## Variable Event CutOff TP FP FN TN Sensitivity  
## 1 P\_upsell\_xsell1 1 0.00 2010 14957 0 0 1.0000000  
## 2 P\_upsell\_xsell1 1 0.05 1782 10958 228 3999 0.8865672  
## 3 P\_upsell\_xsell1 1 0.10 1252 2917 758 12040 0.6228856  
## 4 P\_upsell\_xsell1 1 0.15 1093 1530 917 13427 0.5437811  
## 5 P\_upsell\_xsell1 1 0.20 1006 1009 1004 13948 0.5004975  
## 6 P\_upsell\_xsell1 1 0.25 931 716 1079 14241 0.4631841  
## Specificity KS KS2 F\_HALF FPR ACC FDR  
## 1 0.0000000 0 0.0000000 0.1438221 1.00000000 0.1184653 0.8815347  
## 2 0.2673665 0 0.1539336 0.1682084 0.73263355 0.3407202 0.8601256  
## 3 0.8049743 0 0.4278598 0.3350102 0.19502574 0.7834031 0.6996882  
## 4 0.8977068 1 0.4414879 0.4371301 0.10229324 0.8557789 0.5833016  
## 5 0.9325399 0 0.4330375 0.4995035 0.06746005 0.8813579 0.5007444  
## 6 0.9521294 0 0.4153135 0.5414050 0.04787056 0.8942064 0.4347298  
## F1 C Gini Gamma Tau MISCEVENT  
## 1 0.2118354 0.7569034 0.5138067 0.6289464 0.1073213 0.8815347  
## 2 0.2416271 0.7569034 0.5138067 0.6289464 0.1073213 0.6592798  
## 3 0.4052436 0.7569034 0.5138067 0.6289464 0.1073213 0.2165969  
## 4 0.4718325 0.7569034 0.5138067 0.6289464 0.1073213 0.1442211  
## 5 0.4998758 0.7569034 0.5138067 0.6289464 0.1073213 0.1186421  
## 6 0.5091605 0.7569034 0.5138067 0.6289464 0.1073213 0.1057936

# Display the confusion matrix for cutoff threshold at 0.5  
  
cutoff = subset(roc\_df, CutOff == 0.5)  
  
tn = cutoff$TN  
fn = cutoff$FN  
tp = cutoff$TP  
fp = cutoff$FP  
a = c(tn,fn)  
p = c(fp,tp)  
mat = data.frame(a,p)  
colnames(mat) = c("Pred:0","Pred:1")  
rownames(mat) = c("Actual:0","Actual:1")  
mat = as.matrix(mat)  
print(mat)

## Pred:0 Pred:1  
## Actual:0 14755 202  
## Actual:1 1354 656

# Print the accuracy and misclassification rates for the model  
  
accuracy = cutoff$ACC  
mis = cutoff$MISCEVENT  
  
print(paste("Misclassification rate is",mis))

## [1] "Misclassification rate is 0.09170743207402"

print(paste("Accuracy is",accuracy))

## [1] "Accuracy is 0.90829256792597"

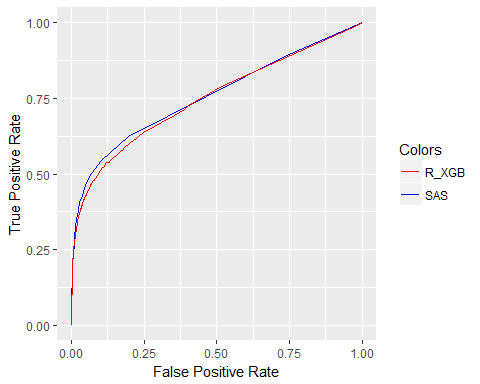
## Assessing the performance metric of R-XGB model  
  
# Create a confusion matrix for cutoff threshold at 0.5  
  
conf.matrix = table(test\_labels, as.numeric(pred>0.5))  
rownames(conf.matrix) = paste("Actual", rownames(conf.matrix), sep = ":")  
colnames(conf.matrix) = paste("Pred", colnames(conf.matrix), sep = ":")  
  
# Print the accuracy and misclassification rates for the model  
  
err = mean(as.numeric(pred > 0.5) != test\_labels)  
  
print(paste("Misclassification rate is",err))

## [1] "Misclassification rate is 0.0941828254847645"

print(paste("Accuracy is",1-err))

## [1] "Accuracy is 0.905817174515235"

# Plot ROC curves for both the models using standard R plotting functions  
  
FPR\_SAS = roc\_df['FPR']  
TPR\_SAS = roc\_df['Sensitivity']  
  
pred1 = prediction(pred, test\_labels)  
perf1 = performance( pred1, "tpr", "fpr" )  
  
FPR\_R = perf1@x.values[[1]]  
TPR\_R = perf1@y.values[[1]]  
  
roc\_df2 = data.frame(FPR = FPR\_R, TPR = TPR\_R)  
  
ggplot() +   
  
geom\_line(  
 data = roc\_df[c('FPR', 'Sensitivity')],   
 aes(x = as.numeric(FPR), y = as.numeric(Sensitivity),color = "SAS"),  
 ) +  
  
geom\_line(  
 data = roc\_df2,   
 aes(x = as.numeric(FPR\_R), y = as.numeric(TPR\_R),color = "R\_XGB"),   
 ) +  
   
scale\_color\_manual(  
 name = "Colors",   
 values = c("SAS" = "blue", "R\_XGB" = "red")  
 ) +  
  
xlab('False Positive Rate') + ylab('True Positive Rate')



# Terminate the CAS session  
  
cas.session.endSession(s)

## list()