Predictive_Modeling_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
```

```
library(pmml)
```

```
## Warning: package 'pmml' was built under R version 3.4.4

## Loading required package: XML

## Warning: package 'XML' was built under R version 3.4.4
```

```
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 3.4.4

## randomForest 4.6-14

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:dplyr':
##
##     combine

## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
library(caret)
```

```
# 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 m

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

## NOTE:       oversample - Samples a user-specified proportion of data from the event level and adjusts

## 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"), copy
                 )
```

```
## 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),
                                      saveState = list(caslib="lg", name="R_SWAT_GB", replace=T),
                                      inputs = inputs,
                                      nominals = nominals,
                                      target = target,
                                      table = train
                                     )
```

```
## NOTE: Wrote 1946372 bytes to the savestate file R_SWAT_GB.
```

```
# 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    23.0
## 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    12.0
## 16            Maximum Size of Leaves 19688.0
## 17            Minimum Size of Leaves     5.0
## 18                Random Number Seed     0.0
##
## $OutputCasTables
##   casLib     Name Rows Columns
## 1     lg gb_model 2688      35
```

```
cas.table.promote(s, caslib="lg", name="R_SWAT_GB", targetCaslib="casuser")
```

```
## ERROR: The target table R_SWAT_GB of the promotion already exists. Please specify a different name.
```

```
## ERROR: The action stopped due to errors.
```

```
## list()
```

```
# 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      0.06318270
## 2        2            0              0        0      0.13746120
## 3        3            0              0        0      0.09871583
## 4        4            0              0        0      0.04420076
## 5        5            0              0        0      0.10690921
## 6        6            0              0        0      0.04801676
```

```
## 7        7            0              0        0      0.05982377
## 8        8            0              0        0      0.06146335
## 9        9            0              0        0      0.06251867
## 10      10            0              0        0      0.04566238
## 11      11            1              0        1      0.39290974
## 12      12            0              0        0      0.06057614
## 13      13            0              0        0      0.04555630
## 14      14            0              0        0      0.08112893
## 15      15            0              0        0      0.27335204
## 16      16            1              0        1      0.21591021
## 17      17            0              1        1      0.64808148
## 18      18            0              0        0      0.05073451
## 19      19            0              0        0      0.05679624
## 20      20            0              0        0      0.04894956
##    P_upsell_xsell0
## 1        0.9368173
## 2        0.8625388
## 3        0.9012842
## 4        0.9557992
## 5        0.8930908
## 6        0.9519832
## 7        0.9401762
## 8        0.9385367
## 9        0.9374813
## 10       0.9543376
## 11       0.6070903
## 12       0.9394239
## 13       0.9544437
## 14       0.9188711
## 15       0.7266480
## 16       0.7840898
## 17       0.3519185
## 18       0.9492655
## 19       0.9432038
## 20       0.9510504
```

```
# Train an R Random Forest 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

train_df$upsell_xsell = as.factor(train_df$upsell_xsell)
val_df$upsell_xsell = as.factor(val_df$upsell_xsell)

train_df$days_openwrkorders = train_df$IMP_days_openwrkorders
train_df$ever_days_over_plan = train_df$IMP_ever_days_over_plan

val_df$days_openwrkorders = val_df$IMP_days_openwrkorders
```

```
val_df$ever_days_over_plan = val_df$IMP_ever_days_over_plan

train_df$IMP_days_openwrkorders = NULL
train_df$IMP_ever_days_over_plan = NULL

val_df$IMP_days_openwrkorders = NULL
val_df$IMP_ever_days_over_plan = NULL
```

```
# Train a RF model on the data

rf_model <- randomForest(upsell_xsell ~ . , ntree=2, mtry=5, data=train_df[,c(3,8,9,10,11,12,14)], impo

# Make predictions on test data

pred <- predict(rf_model, val_df[,c(3,8,9,10,11,12,14)], type="prob")
```

```
# 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 1819 11472  191  3485   0.9049751
## 3 P_upsell_xsell1     1   0.10 1248  2821  762 12136   0.6208955
## 4 P_upsell_xsell1     1   0.15 1094  1532  916 13425   0.5442786
## 5 P_upsell_xsell1     1   0.20 1007  1001 1003 13956   0.5009950
## 6 P_upsell_xsell1     1   0.25  948   719 1062 14238   0.4716418
##   Specificity KS       KS2    F_HALF        FPR       ACC       FDR
## 1   0.0000000  0 0.0000000 0.1438221 1.00000000 0.1184653 0.8815347
## 2   0.2330013  0 0.1379764 0.1648421 0.76699873 0.3126068 0.8631405
## 3   0.8113927  0 0.4322882 0.3412447 0.18860734 0.7888254 0.6932907
## 4   0.8975730  1 0.4418516 0.4371104 0.10242696 0.8557199 0.5833968
## 5   0.9330748  0 0.4340698 0.5013941 0.06692519 0.8818884 0.4985060
## 6   0.9519289  0 0.4235707 0.5462088 0.04807114 0.8950315 0.4313137
##          F1         C      Gini     Gamma       Tau MISCEVENT
## 1 0.2118354 0.7585858 0.5171715 0.6437269 0.1080241 0.8815347
## 2 0.2377622 0.7585858 0.5171715 0.6437269 0.1080241 0.6873932
## 3 0.4105938 0.7585858 0.5171715 0.6437269 0.1080241 0.2111746
## 4 0.4719586 0.7585858 0.5171715 0.6437269 0.1080241 0.1442801
## 5 0.5012444 0.7585858 0.5171715 0.6437269 0.1080241 0.1181116
## 6 0.5156377 0.7585858 0.5171715 0.6437269 0.1080241 0.1049685
```

```
# 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  14749    208
## Actual:1   1358    652
```

```
# 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 0.09229681145753"
```

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

```
## [1] "Accuracy is 0.90770318854246"
```

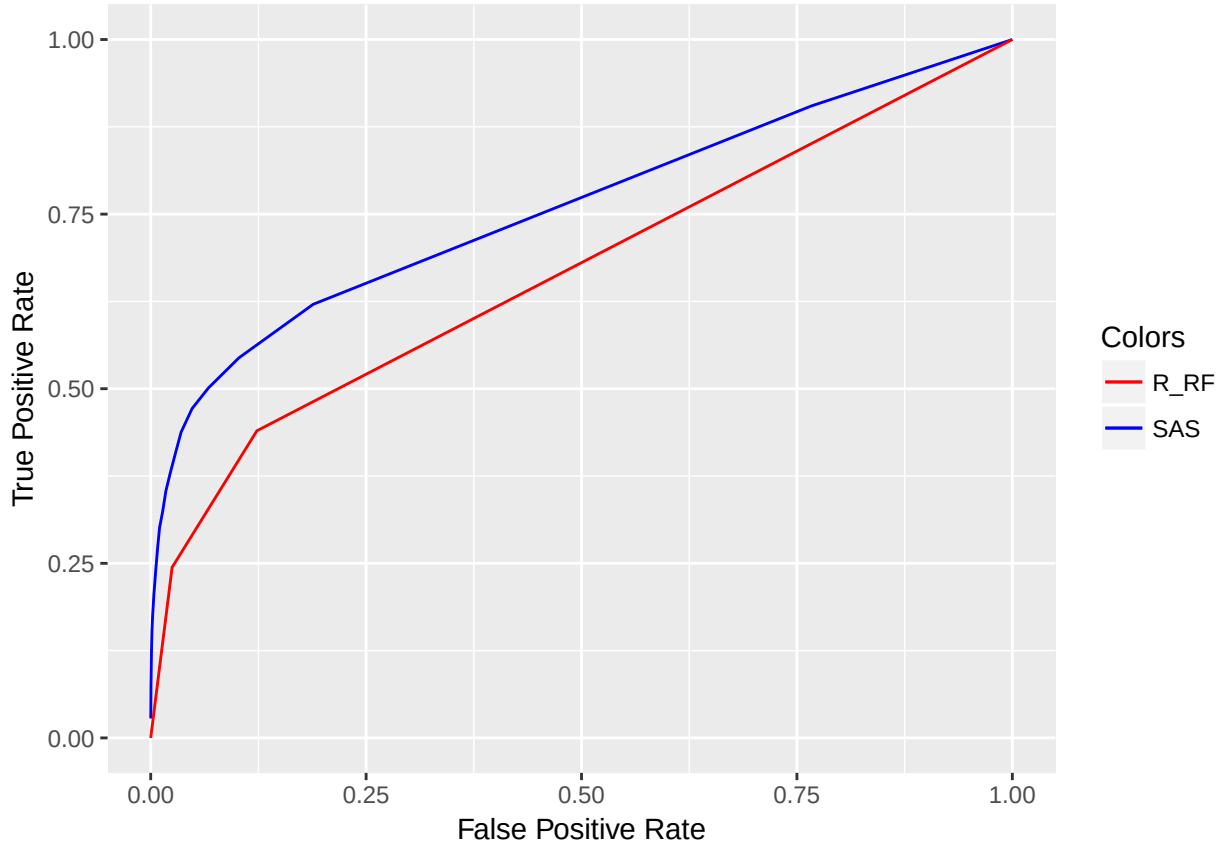
```
## Assessing the performance metric of R-RF model

# Create a confusion matrix for cutoff threshold at 0.5

conf.matrix = table(val_df$upsell_xsell, as.numeric(pred[,2]>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[,2] > 0.5) != val_df$upsell_xsell)

print(paste("Misclassification rate is",err))
```

```
## [1] "Misclassification rate is 0.111451641421583"
```

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

```
## [1] "Accuracy is 0.888548358578417"
```

```
# 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[,2], val_df$upsell_xsell)
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_RF"),
  ) +

  scale_color_manual(
    name = "Colors",
    values = c("SAS" = "blue", "R_RF" = "red")
  ) +

  xlab('False Positive Rate') + ylab('True Positive Rate')
```




```
# Generating PMML code to export R model to Model Manager

rf.pmml = pmml(rf_model)
```

```
## [1] "Now converting tree  1  to PMML"
## [1] "Now converting tree  2  to PMML"
```

```
format(object.size(rf.pmml))
```

```
## [1] "60414848 bytes"
```

```
savePMML(rf.pmml, "C:/Users/neveng/rf.xml", version=4.2 )
```

```
# Terminate the CAS session

cas.session.endSession(s)
```

```
## list()
```