GCC Project Report

Xiaoshuo Liu

April 8, 2019

I. Introduction

GCC is a project that poses a real-world business problem, how does a magazine company improve its renewal rate? I found this case on a business analytics textbook named, . The goal of this project is to predict whether a customer is to renew his or her subscription or not. In this project, I mainly use the three models focused from the courses, logistic regression, k nearest neighbors, and classification tree. A principal component analysis is also done to test the feasibility of using less predictors to generate similar and high accuracy.

For all the codes and files, please visit the following Github page:

1. Preparing the Data Set

The dataset is provided in a .xlsx file. The first sheet lists descriptions of all variables and the second sheet contains all the data. I read through all the descriptions to become familiar with the general overview of the set. The set covers quite a wide range of information on GCC’s subscribers. For the ease of reading the data, I converted the .xlsx file to .csv file. The file is imported into R as a data frame.

First, load all the packages needed.

library(tidyverse)

## -- Attaching packages ------------------------------------------------------------------- tidyverse 1.2.1 --

## v ggplot2 3.1.1 v purrr 0.3.2   
## v tibble 2.1.1 v dplyr 0.8.0.1  
## v tidyr 0.8.3 v stringr 1.4.0   
## v readr 1.3.1 v forcats 0.4.0

## -- Conflicts ---------------------------------------------------------------------- tidyverse\_conflicts() --  
## x dplyr::filter() masks stats::filter()  
## x dplyr::lag() masks stats::lag()

library(purrr)  
library(caret)

## Loading required package: lattice

##   
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':  
##   
## lift

library(mlr)

## Loading required package: ParamHelpers

##   
## Attaching package: 'mlr'

## The following object is masked from 'package:caret':  
##   
## train

gcc <- read.csv("GCC.csv")

I use the following code to view the dataset.

head(gcc)

## ï..CustomerID Renewal Age HomeOwner ResidenceLength DwellingType Gender  
## 1 1 0 70 10 1 M F  
## 2 2 0 65 10 4 M F  
## 3 3 0 76 10 1 M F  
## 4 4 0 54 10 1 M F  
## 5 5 0 78 10 7 M M  
## 6 6 1 40 10 1 M F  
## Marital HouseholdSize ChildPresent Child0.5 Child6.12 Child13.18 Income  
## 1 M 3 Y 45 35 100 29000  
## 2 S 1 U 35 15 35 13000  
## 3 M 2 N 25 15 25 36000  
## 4 S 1 U 0 15 0 23000  
## 5 M 2 N 35 15 15 5000  
## 6 M 2 Y 65 55 85 30000  
## Occupation HomeValue MagazineStatus PaidDirectMailOrders  
## 1 R 4 B 2  
## 2 M 8 E 1  
## 3 U 7 E 3  
## 4 M 3 E 0  
## 5 W 9 E 0  
## 6 U 2 E 0  
## YearsSinceLastOrder TotalAmountPaid DollarsPerIssue TotalPaidOrders  
## 1 8 50.00 0.36 3  
## 2 4 40.00 0.48 4  
## 3 3 60.00 0.62 3  
## 4 5 41.50 0.38 3  
## 5 3 97.00 1.47 4  
## 6 4 21.97 0.61 2  
## MonthsSinceLastPayment LastPaymentType UnpaidMagazines PaidCashMagazines  
## 1 89 0 3 2  
## 2 39 A 0 2  
## 3 53 1 0 0  
## 4 68 2 0 0  
## 5 37 A 0 2  
## 6 59 A 0 2  
## PaidReinstateMagazines PaidCreditMagazines ActiveSubscriptions  
## 1 0 0 0  
## 2 0 0 0  
## 3 0 2 0  
## 4 0 1 0  
## 5 0 0 0  
## 6 0 0 0  
## ExpiredSubscriptions RequestedCancellations NoPayCancellations  
## 1 2 0 3  
## 2 2 0 0  
## 3 2 0 0  
## 4 1 0 0  
## 5 1 0 1  
## 6 2 0 0  
## PaidComplaints GiftDonor NumberGiftDonations MonthsSince1stOrder  
## 1 0 N 0 118  
## 2 0 Y 2 90  
## 3 0 N 0 91  
## 4 0 N 0 158  
## 5 0 N 0 50  
## 6 0 N 0 59  
## MonthsSinceLastOrder MonthsSinceExpire  
## 1 89 52  
## 2 90 52  
## 3 54 17  
## 4 60 20  
## 5 37 34  
## 6 59 46

Since the goal is to predict renewing customers, I want to see the overall renewal rate.

mean(gcc$Renewal)

## [1] 0.02035556

The rate is ~2%.

We can see that there are 38 total columns. Some of the colunms have character-based indications. I need to transform into dummy variables for further analysis. But before doing so, I want to check if there are NA values in the set that could cause malfunctions to our modeling functions.

any(is.na.data.frame(gcc))

## [1] TRUE

From the result, I can see that there are indeed NA values across the data frame. Thus, I want to remove them.

gcc <- na.omit(gcc)

Now, I am going to transform the character values into dummy variables. I will my using the “createDummyFeature” function from the “mlr” package.

chaCol <- c("DwellingType", "Gender", "Marital", "ChildPresent", "Occupation","MagazineStatus","LastPaymentType","GiftDonor")  
gcc <- createDummyFeatures(gcc, cols = chaCol)

Let’s examine the new data frame.

head(gcc)

## ï..CustomerID Renewal Age HomeOwner ResidenceLength HouseholdSize  
## 1 1 0 70 10 1 3  
## 2 2 0 65 10 4 1  
## 3 3 0 76 10 1 2  
## 4 4 0 54 10 1 1  
## 5 5 0 78 10 7 2  
## 6 6 1 40 10 1 2  
## Child0.5 Child6.12 Child13.18 Income HomeValue PaidDirectMailOrders  
## 1 45 35 100 29000 4 2  
## 2 35 15 35 13000 8 1  
## 3 25 15 25 36000 7 3  
## 4 0 15 0 23000 3 0  
## 5 35 15 15 5000 9 0  
## 6 65 55 85 30000 2 0  
## YearsSinceLastOrder TotalAmountPaid DollarsPerIssue TotalPaidOrders  
## 1 8 50.00 0.36 3  
## 2 4 40.00 0.48 4  
## 3 3 60.00 0.62 3  
## 4 5 41.50 0.38 3  
## 5 3 97.00 1.47 4  
## 6 4 21.97 0.61 2  
## MonthsSinceLastPayment UnpaidMagazines PaidCashMagazines  
## 1 89 3 2  
## 2 39 0 2  
## 3 53 0 0  
## 4 68 0 0  
## 5 37 0 2  
## 6 59 0 2  
## PaidReinstateMagazines PaidCreditMagazines ActiveSubscriptions  
## 1 0 0 0  
## 2 0 0 0  
## 3 0 2 0  
## 4 0 1 0  
## 5 0 0 0  
## 6 0 0 0  
## ExpiredSubscriptions RequestedCancellations NoPayCancellations  
## 1 2 0 3  
## 2 2 0 0  
## 3 2 0 0  
## 4 1 0 0  
## 5 1 0 1  
## 6 2 0 0  
## PaidComplaints NumberGiftDonations MonthsSince1stOrder  
## 1 0 0 118  
## 2 0 2 90  
## 3 0 0 91  
## 4 0 0 158  
## 5 0 0 50  
## 6 0 0 59  
## MonthsSinceLastOrder MonthsSinceExpire DwellingType.M DwellingType.S  
## 1 89 52 1 0  
## 2 90 52 1 0  
## 3 54 17 1 0  
## 4 60 20 1 0  
## 5 37 34 1 0  
## 6 59 46 1 0  
## DwellingType.U Gender.F Gender.M Gender.U Marital.M Marital.S Marital.U  
## 1 0 1 0 0 1 0 0  
## 2 0 1 0 0 0 1 0  
## 3 0 1 0 0 1 0 0  
## 4 0 1 0 0 0 1 0  
## 5 0 0 1 0 1 0 0  
## 6 0 1 0 0 1 0 0  
## ChildPresent.N ChildPresent.U ChildPresent.Y Occupation.B Occupation.H  
## 1 0 0 1 0 0  
## 2 0 1 0 0 0  
## 3 1 0 0 0 0  
## 4 0 1 0 0 0  
## 5 1 0 0 0 0  
## 6 0 0 1 0 0  
## Occupation.M Occupation.R Occupation.U Occupation.W MagazineStatus.A  
## 1 0 1 0 0 0  
## 2 1 0 0 0 0  
## 3 0 0 1 0 0  
## 4 1 0 0 0 0  
## 5 0 0 0 1 0  
## 6 0 0 1 0 0  
## MagazineStatus.B MagazineStatus.C MagazineStatus.E MagazineStatus.N  
## 1 1 0 0 0  
## 2 0 0 1 0  
## 3 0 0 1 0  
## 4 0 0 1 0  
## 5 0 0 1 0  
## 6 0 0 1 0  
## MagazineStatus.O MagazineStatus.S LastPaymentType.0 LastPaymentType.1  
## 1 0 0 1 0  
## 2 0 0 0 0  
## 3 0 0 0 1  
## 4 0 0 0 0  
## 5 0 0 0 0  
## 6 0 0 0 0  
## LastPaymentType.2 LastPaymentType.3 LastPaymentType.4 LastPaymentType.5  
## 1 0 0 0 0  
## 2 0 0 0 0  
## 3 0 0 0 0  
## 4 1 0 0 0  
## 5 0 0 0 0  
## 6 0 0 0 0  
## LastPaymentType.6 LastPaymentType.7 LastPaymentType.8 LastPaymentType.9  
## 1 0 0 0 0  
## 2 0 0 0 0  
## 3 0 0 0 0  
## 4 0 0 0 0  
## 5 0 0 0 0  
## 6 0 0 0 0  
## LastPaymentType.A LastPaymentType.E LastPaymentType.F LastPaymentType.G  
## 1 0 0 0 0  
## 2 1 0 0 0  
## 3 0 0 0 0  
## 4 0 0 0 0  
## 5 1 0 0 0  
## 6 1 0 0 0  
## LastPaymentType.I LastPaymentType.K LastPaymentType.L LastPaymentType.M  
## 1 0 0 0 0  
## 2 0 0 0 0  
## 3 0 0 0 0  
## 4 0 0 0 0  
## 5 0 0 0 0  
## 6 0 0 0 0  
## LastPaymentType.S LastPaymentType.U GiftDonor.N GiftDonor.Y  
## 1 0 0 1 0  
## 2 0 0 0 1  
## 3 0 0 1 0  
## 4 0 0 1 0  
## 5 0 0 1 0  
## 6 0 0 1 0

We can see that for each character-based variable, the “createDummyFeature” function created a dummy variable for all the values, which makes “dummy trap” likely to happen later in creating models. Therefore, I have removed one dummy from each set of dummy variables. The original character-based variables have also been removed for the slimness of the data.

gcc$DwellingType.U <- NULL  
gcc$Gender.U <- NULL  
gcc$Marital.U <- NULL  
gcc$ChildPresent.U <- NULL  
gcc$Occupation.U <- NULL  
gcc$MagazineStatus.S <- NULL  
gcc$LastPaymentType.0 <- NULL  
gcc$GiftDonor.N <- NULL  
  
gcc$DwellingType <- NULL  
gcc$Gender <- NULL  
gcc$Marital <- NULL  
gcc$ChildPresent <- NULL  
gcc$Occupation <- NULL  
gcc$MagazineStatus <- NULL  
gcc$LastPaymentType <- NULL  
gcc$GiftDonor <- NULL

After cleaning up the data, partitions can be created.

renewal <- gcc[,2]  
valiIndex <- createDataPartition(renewal, p = 0.5, times = 1, list = FALSE)  
validation <- gcc[valiIndex,]  
gccModelData <- gcc[-valiIndex,]  
  
renewalTest <- gccModelData[,2]  
testIndex <- createDataPartition(renewalTest, p = 0.4, times = 1, list = FALSE)  
train <- gccModelData[-testIndex,]  
test <- gccModelData[testIndex,]

The objects used above are removed once the process is over.

rm(renewal)  
rm(renewalTest)  
rm(valiIndex)  
rm(testIndex)  
rm(gcc)  
rm(gccModelData)  
rm(chaCol)

Next, data frames of actual values and predictors are created.

predictors <- train[,-2]  
pred <- train[,2]  
  
testPredictors <- test[, -2]  
testPred <- test[,2]

1. Principal Component Analysis

From the last section, I can see that there are 68 predictors that can be used. I consider this as too many, so I use PCA to see the possibility of shrinking down the number.

pca <- prcomp(predictors, center = FALSE, scale. = FALSE)  
pcaTest <- prcomp(testPredictors, center = FALSE, scale. = FALSE)  
  
summary(pca)

## Importance of components:  
## PC1 PC2 PC3 PC4 PC5 PC6 PC7  
## Standard deviation 1.174e+05 1.874e+04 132.8 60.78 52.68 45.53 36.29  
## Proportion of Variance 9.751e-01 2.487e-02 0.0 0.00 0.00 0.00 0.00  
## Cumulative Proportion 9.751e-01 1.000e+00 1.0 1.00 1.00 1.00 1.00  
## PC8 PC9 PC10 PC11 PC12 PC13 PC14 PC15  
## Standard deviation 31.14 25.23 23.06 15.13 8.4 4.536 2.579 2.008  
## Proportion of Variance 0.00 0.00 0.00 0.00 0.0 0.000 0.000 0.000  
## Cumulative Proportion 1.00 1.00 1.00 1.00 1.0 1.000 1.000 1.000  
## PC16 PC17 PC18 PC19 PC20 PC21 PC22 PC23  
## Standard deviation 1.962 1.531 1.296 1.201 1.066 0.7753 0.7191 0.5695  
## Proportion of Variance 0.000 0.000 0.000 0.000 0.000 0.0000 0.0000 0.0000  
## Cumulative Proportion 1.000 1.000 1.000 1.000 1.000 1.0000 1.0000 1.0000  
## PC24 PC25 PC26 PC27 PC28 PC29 PC30  
## Standard deviation 0.5378 0.5268 0.5009 0.4684 0.4654 0.3986 0.398  
## Proportion of Variance 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.000  
## Cumulative Proportion 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.000  
## PC31 PC32 PC33 PC34 PC35 PC36 PC37  
## Standard deviation 0.3837 0.3806 0.353 0.2954 0.2845 0.2718 0.2603  
## Proportion of Variance 0.0000 0.0000 0.000 0.0000 0.0000 0.0000 0.0000  
## Cumulative Proportion 1.0000 1.0000 1.000 1.0000 1.0000 1.0000 1.0000  
## PC38 PC39 PC40 PC41 PC42 PC43 PC44  
## Standard deviation 0.2547 0.241 0.2081 0.1922 0.1805 0.1715 0.1576  
## Proportion of Variance 0.0000 0.000 0.0000 0.0000 0.0000 0.0000 0.0000  
## Cumulative Proportion 1.0000 1.000 1.0000 1.0000 1.0000 1.0000 1.0000  
## PC45 PC46 PC47 PC48 PC49 PC50 PC51  
## Standard deviation 0.1544 0.1325 0.1168 0.1151 0.1027 0.09603 0.09042  
## Proportion of Variance 0.0000 0.0000 0.0000 0.0000 0.0000 0.00000 0.00000  
## Cumulative Proportion 1.0000 1.0000 1.0000 1.0000 1.0000 1.00000 1.00000  
## PC52 PC53 PC54 PC55 PC56 PC57  
## Standard deviation 0.08757 0.07884 0.0742 0.06685 0.06038 0.05812  
## Proportion of Variance 0.00000 0.00000 0.0000 0.00000 0.00000 0.00000  
## Cumulative Proportion 1.00000 1.00000 1.0000 1.00000 1.00000 1.00000  
## PC58 PC59 PC60 PC61 PC62 PC63  
## Standard deviation 0.05342 0.04714 0.04126 0.03573 0.0331 0.02856  
## Proportion of Variance 0.00000 0.00000 0.00000 0.00000 0.0000 0.00000  
## Cumulative Proportion 1.00000 1.00000 1.00000 1.00000 1.0000 1.00000  
## PC64 PC65 PC66 PC67 PC68  
## Standard deviation 0.02457 0.02155 0.01252 0.01243 8.573e-12  
## Proportion of Variance 0.00000 0.00000 0.00000 0.00000 0.000e+00  
## Cumulative Proportion 1.00000 1.00000 1.00000 1.00000 1.000e+00

The PCA result is telling me that the first two PCA variables can capture 100% of variablilities of the data set. This is quite suspicious. I conducted further testing into the feasibility of using PCA for this set, and the feedback is negative (view the entire process through Github page’s PCA.R file). Therefore, I will be using the original data set for predictive models.

1. Modeling
2. Logistic Regression

#logistic regression  
fit\_glm <- glm(Renewal ~., data = train, family = "binomial")

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

pred\_glm <- predict(fit\_glm, test)

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type =  
## ifelse(type == : prediction from a rank-deficient fit may be misleading

y\_hat\_glm <- ifelse(pred\_glm >= 0.5,yes = 1, no = 0)   
  
cm\_glm <- confusionMatrix(as.factor(y\_hat\_glm), as.factor(testPred))  
rmse\_glm <- RMSE(as.numeric(testPred), as.numeric(y\_hat\_glm))  
cm\_glm

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1  
## 0 8246 162  
## 1 4 4  
##   
## Accuracy : 0.9803   
## 95% CI : (0.9771, 0.9831)  
## No Information Rate : 0.9803   
## P-Value [Acc > NIR] : 0.5206   
##   
## Kappa : 0.0442   
##   
## Mcnemar's Test P-Value : <2e-16   
##   
## Sensitivity : 0.9995   
## Specificity : 0.0241   
## Pos Pred Value : 0.9807   
## Neg Pred Value : 0.5000   
## Prevalence : 0.9803   
## Detection Rate : 0.9798   
## Detection Prevalence : 0.9990   
## Balanced Accuracy : 0.5118   
##   
## 'Positive' Class : 0   
##

rmse\_glm

## [1] 0.1404434

1. K Nearest Neighbors

I use the following codes to run the model. To save time from long processing sessions, I adopt the control method used by Dr. Irizarry. The upper limit of predictors is 45, a subjective number that I have chosen for a balance between coverage and over fitting.

control <- trainControl(method = "cv", number = 5, p = .9)  
pred <- as.factor(pred)  
train\_knn <- caret::train(predictors, pred,  
 method = "knn",  
 tuneGrid = data.frame(k = seq(10:30)),  
 trControl = control)  
train\_knn$bestTune

## k  
## 21 21

pred <- as.factor(pred)  
fit\_knn <- knn3(predictors, pred, k = train\_knn$bestTune)  
y\_hat\_knn <- predict(fit\_knn,  
 testPredictors,  
 type = "class")  
cm\_knn <- confusionMatrix(y\_hat\_knn, factor(testPred))  
rmse\_knn <- RMSE(as.numeric(testPred), as.numeric(y\_hat\_knn))  
cm\_knn

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1  
## 0 8250 166  
## 1 0 0  
##   
## Accuracy : 0.9803   
## 95% CI : (0.9771, 0.9831)  
## No Information Rate : 0.9803   
## P-Value [Acc > NIR] : 0.5206   
##   
## Kappa : 0   
##   
## Mcnemar's Test P-Value : <2e-16   
##   
## Sensitivity : 1.0000   
## Specificity : 0.0000   
## Pos Pred Value : 0.9803   
## Neg Pred Value : NaN   
## Prevalence : 0.9803   
## Detection Rate : 0.9803   
## Detection Prevalence : 1.0000   
## Balanced Accuracy : 0.5000   
##   
## 'Positive' Class : 0   
##

rmse\_knn

## [1] 0.9900887

As The accuracy seems great at over 98%, and the RMSE is reasonable. However, this is only part of the story. Looking at the confusion matrix results, the algorithm predicted no renewal. The model is then obsolete because the goal of this case is to predict the customers that are more likely to renew so the company can better utilize its marketing campaign. The root of this problem may come from the very low prevelance of renewing customers - only 2%, as found in the first section. The algorithm simply predicts that no one will renew to gain the highest accuracy. This is an indicator that knn does not provide valid solution to this data set.

1. Regression Tree

I first used the “randomForest” package to run the training.

library(randomForest)

## 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

control <- trainControl(method = "cv", number = 5, p = 0.9)  
grid <- expand.grid(mtry = 10)  
pred <- as.factor(pred)  
train\_rf <- caret::train(predictors,  
 pred,  
 method = "cforest",  
 trControl = control,  
 tuneGrid = grid)  
train\_rf$bestTune

## mtry  
## 1 10

fit\_rf <- randomForest(predictors, pred,   
 minNode = train\_rf$bestTune$minNode,  
 predFixed = train\_rf$bestTune$predFixed)  
pred\_rf <- predict(fit\_rf, testPredictors)  
y\_hat\_rf <- as.factor(pred\_rf)  
testPred <- as.factor(testPred)  
cm\_rf <- confusionMatrix(y\_hat\_rf, testPred)  
rmse\_rf <- RMSE(as.numeric(testPred), as.numeric(y\_hat\_rf))  
cm\_rf

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1  
## 0 8247 157  
## 1 3 9  
##   
## Accuracy : 0.981   
## 95% CI : (0.9778, 0.9838)  
## No Information Rate : 0.9803   
## P-Value [Acc > NIR] : 0.3369   
##   
## Kappa : 0.0987   
##   
## Mcnemar's Test P-Value : <2e-16   
##   
## Sensitivity : 0.99964   
## Specificity : 0.05422   
## Pos Pred Value : 0.98132   
## Neg Pred Value : 0.75000   
## Prevalence : 0.98028   
## Detection Rate : 0.97992   
## Detection Prevalence : 0.99857   
## Balanced Accuracy : 0.52693   
##   
## 'Positive' Class : 0   
##

rmse\_rf

## [1] 0.1378819

imp\_rf <- importance(fit\_rf)  
imp\_rf

## MeanDecreaseGini  
## ï..CustomerID 2.065416e+01  
## Age 1.811167e+01  
## HomeOwner 1.743114e+00  
## ResidenceLength 1.308026e+01  
## HouseholdSize 9.847236e+00  
## Child0.5 1.035369e+01  
## Child6.12 2.329398e+01  
## Child13.18 1.009224e+01  
## Income 3.175004e+01  
## HomeValue 1.206243e+01  
## PaidDirectMailOrders 7.526543e+00  
## YearsSinceLastOrder 1.283893e+01  
## TotalAmountPaid 2.062539e+01  
## DollarsPerIssue 5.291687e+01  
## TotalPaidOrders 1.414074e+01  
## MonthsSinceLastPayment 1.805976e+01  
## UnpaidMagazines 3.412368e+00  
## PaidCashMagazines 7.823357e+00  
## PaidReinstateMagazines 1.272621e+00  
## PaidCreditMagazines 4.400681e+00  
## ActiveSubscriptions 2.577448e+00  
## ExpiredSubscriptions 6.472397e+00  
## RequestedCancellations 2.798300e+00  
## NoPayCancellations 3.030302e+00  
## PaidComplaints 5.447711e-01  
## NumberGiftDonations 4.642290e+00  
## MonthsSince1stOrder 4.248794e+01  
## MonthsSinceLastOrder 2.590051e+01  
## MonthsSinceExpire 4.181478e+01  
## DwellingType.M 1.715913e+00  
## DwellingType.S 1.740670e+00  
## Gender.F 2.563286e+00  
## Gender.M 2.679998e+00  
## Marital.M 2.306446e+00  
## Marital.S 2.375306e+00  
## ChildPresent.N 1.894596e+00  
## ChildPresent.Y 2.629416e+00  
## Occupation.B 2.049398e+00  
## Occupation.H 1.732659e+00  
## Occupation.M 2.607436e+00  
## Occupation.R 1.397917e+00  
## Occupation.W 3.512668e+00  
## MagazineStatus.A 1.675586e+00  
## MagazineStatus.B 1.531738e+00  
## MagazineStatus.C 6.272670e-01  
## MagazineStatus.E 2.960563e+00  
## MagazineStatus.N 0.000000e+00  
## MagazineStatus.O 2.068702e+00  
## LastPaymentType.1 1.498113e+00  
## LastPaymentType.2 2.078588e-01  
## LastPaymentType.3 5.901308e-01  
## LastPaymentType.4 2.423720e-01  
## LastPaymentType.5 2.940433e-01  
## LastPaymentType.6 9.583817e-03  
## LastPaymentType.7 5.404762e-03  
## LastPaymentType.8 9.142857e-04  
## LastPaymentType.9 0.000000e+00  
## LastPaymentType.A 2.604671e+00  
## LastPaymentType.E 8.309570e-01  
## LastPaymentType.F 6.316489e-01  
## LastPaymentType.G 3.763089e-01  
## LastPaymentType.I 8.264134e-01  
## LastPaymentType.K 2.409661e-03  
## LastPaymentType.L 1.212460e+00  
## LastPaymentType.M 4.209266e-01  
## LastPaymentType.S 1.759187e+00  
## LastPaymentType.U 0.000000e+00  
## GiftDonor.Y 1.731291e+00

To find the optimal method of running the random forest model, I tried using the Rborist package, too.

#The Rborist codes   
library(Rborist)

## Rborist 0.1-17

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

control <- trainControl(method = "cv", number = 5, p = 0.9)  
grid <- expand.grid(minNode = c(2,5), predFixed = c(15,20,25,30,35))  
pred <- as.factor(pred)  
train\_rborist <- caret::train(predictors,  
 pred,  
 method = "Rborist",  
 trControl = control,  
 tuneGrid = grid)  
train\_rborist$bestTune

## predFixed minNode  
## 8 25 5

fit\_rborist <- Rborist(predictors, pred,   
 minNode = train\_rborist$bestTune$minNode,  
 predFixed = train\_rborist$bestTune$predFixed)  
pred\_rborist <- predict(fit\_rborist, testPredictors)  
y\_hat\_rborist <- as.factor(levels(pred)[predict(fit\_rborist, testPredictors)$yPred])  
testPred <- as.factor(testPred)  
cm\_rborist <- confusionMatrix(y\_hat\_rborist, testPred)  
rmse\_rborist <- RMSE(as.numeric(testPred), as.numeric(y\_hat\_rborist))  
cm\_rborist

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1  
## 0 8231 113  
## 1 19 53  
##   
## Accuracy : 0.9843   
## 95% CI : (0.9814, 0.9869)  
## No Information Rate : 0.9803   
## P-Value [Acc > NIR] : 0.003386   
##   
## Kappa : 0.4387   
##   
## Mcnemar's Test P-Value : 5.745e-16   
##   
## Sensitivity : 0.9977   
## Specificity : 0.3193   
## Pos Pred Value : 0.9865   
## Neg Pred Value : 0.7361   
## Prevalence : 0.9803   
## Detection Rate : 0.9780   
## Detection Prevalence : 0.9914   
## Balanced Accuracy : 0.6585   
##   
## 'Positive' Class : 0   
##

rmse\_rborist

## [1] 0.1252374

From the outcome, we could see that the method actually worked.

Last, I want to test the data set on “rpart” function.

library(rpart)  
fit\_tree <- rpart(Renewal ~., data = train, method = "class")  
pred\_tree <- predict(fit\_tree, test, type = "class")  
y\_hat\_tree <- as.factor(pred\_tree)  
cm\_tree <- confusionMatrix(y\_hat\_tree, as.factor(testPred))  
cm\_tree$overall["Accuracy"]

## Accuracy   
## 0.9904943

rmse\_tree <- RMSE(as.numeric(testPred), as.numeric(y\_hat\_tree))  
cm\_tree

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1  
## 0 8201 31  
## 1 49 135  
##   
## Accuracy : 0.9905   
## 95% CI : (0.9882, 0.9925)  
## No Information Rate : 0.9803   
## P-Value [Acc > NIR] : 5.574e-14   
##   
## Kappa : 0.7666   
##   
## Mcnemar's Test P-Value : 0.05735   
##   
## Sensitivity : 0.9941   
## Specificity : 0.8133   
## Pos Pred Value : 0.9962   
## Neg Pred Value : 0.7337   
## Prevalence : 0.9803   
## Detection Rate : 0.9745   
## Detection Prevalence : 0.9781   
## Balanced Accuracy : 0.9037   
##   
## 'Positive' Class : 0   
##

rmse\_tree

## [1] 0.0974972

V. Final Test

#Validating all the valid models with the validation set for the final evaluation.   
##Create data frame for predictors and actual values.   
valiPredictors <- validation[, -2]  
valiPred <- validation[,2]  
  
  
#Logistics  
pred\_glm\_vali <- predict(fit\_glm, validation)

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type =  
## ifelse(type == : prediction from a rank-deficient fit may be misleading

y\_hat\_glm\_vali <- ifelse(pred\_glm\_vali >= 0.5,yes = 1, no = 0)   
cm\_glm\_vali <- confusionMatrix(as.factor(y\_hat\_glm\_vali), as.factor(valiPred))  
rmse\_glm\_vali <- RMSE(as.numeric(valiPred), as.numeric(y\_hat\_glm\_vali))  
  
#Random Forest  
pred\_rborist\_vali <- predict(fit\_rborist, valiPredictors)  
y\_hat\_rborist\_vali <- as.factor(levels(pred)[predict(fit\_rborist, valiPredictors)$yPred])  
valiPred <- as.factor(valiPred)  
cm\_rborist\_vali <- confusionMatrix(y\_hat\_rborist\_vali, valiPred)  
rmse\_rborist\_vali <- RMSE(as.numeric(valiPred), as.numeric(y\_hat\_rborist\_vali))  
  
#Rpart  
pred\_tree\_vali <- predict(fit\_tree, validation, type = "class")  
y\_hat\_tree\_vali <- as.factor(pred\_tree\_vali)  
cm\_tree\_vali <- confusionMatrix(y\_hat\_tree\_vali, as.factor(valiPred))  
rmse\_tree\_vali <- RMSE(as.numeric(valiPred), as.numeric(y\_hat\_tree\_vali))  
  
#Check the sensitivity and RMSE of results.  
names <- c("glm", "Rborist", "rpart")  
  
sensitivities <- c(cm\_glm\_vali$byClass["Sensitivity"],   
 cm\_rborist\_vali$byClass["Sensitivity"],  
 cm\_tree\_vali$byClass["Sensitivity"])  
  
rmses <- c(rmse\_glm\_vali,  
 rmse\_rborist\_vali,  
 rmse\_tree\_vali)  
  
result <- data.frame("Name" = names,  
 "Sensitivity" = sensitivities,  
 "RMSE" = rmses)  
result

## Name Sensitivity RMSE  
## 1 glm 0.9994649 0.15292201  
## 2 Rborist 0.9980540 0.13332858  
## 3 rpart 0.9953296 0.09919108

V. Results