Predicting Churn of imbalance data using Neural Net, Random Forrest and Logistics Regression

```
library(readxl)
dm4 <- read_xlsx("ChurnData.xlsx",sheet = 2)
str(dm4)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    6347 obs. of  12 variables:
##  $ CustomerAge          : num  67 67 55 63 57 58 57 46 56 56 ...
##  $ Churn                : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ CHIScoreMonth0       : num  0 62 0 231 43 138 180 116 78 78 ...
##  $ CHIScore0_1          : num  0 4 0 1 -1 -10 -5 -11 -7 -37 ...
##  $ SupportCasesMonth0   : num  0 0 0 1 0 0 1 0 1 0 ...
##  $ SupportCases0_1      : num  0 0 0 -1 0 0 1 0 -2 0 ...
##  $ SPMonth_0            : num  0 0 0 3 0 0 3 0 3 0 ...
##  $ SP0_1                : num  0 0 0 0 0 0 3 0 0 0 ...
##  $ Logins0_1            : num  0 0 0 167 0 43 13 0 -9 -7 ...
##  $ BlogArticles0_1      : num  0 0 0 -8 0 0 -1 0 1 0 ...
##  $ Views0_1             : num  0 -16 0 21996 9 ...
##  $ DaysSinceLastLogin0_1: num  31 31 31 0 31 0 0 6 7 14 ...
```

```
dm4$Churn<-as.factor(dm4$Churn)
```

```
apply(dm4,2,function(x) sum(is.na(x)))
```

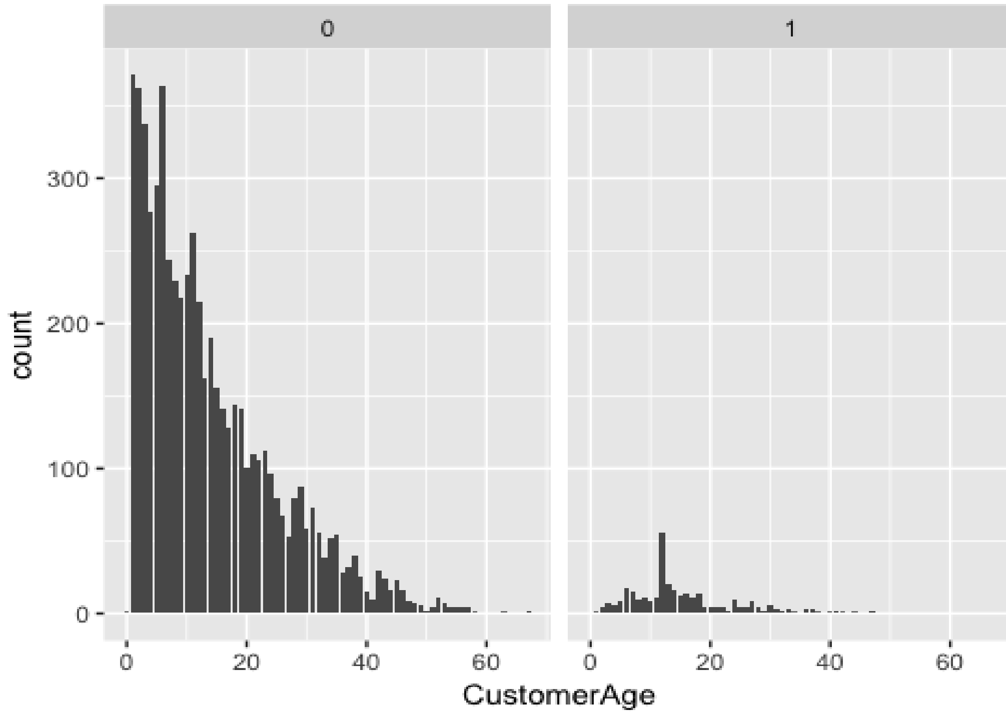
```
##           CustomerAge                 Churn        CHIScoreMonth0 
##                     0                     0                     0 
##           CHIScore0_1    SupportCasesMonth0       SupportCases0_1 
##                     0                     0                     0 
##             SPMonth_0                 SP0_1             Logins0_1 
##                     0                     0                     0 
##       BlogArticles0_1              Views0_1 DaysSinceLastLogin0_1 
##                     0                     0                     0
```

Distribution of churn rate

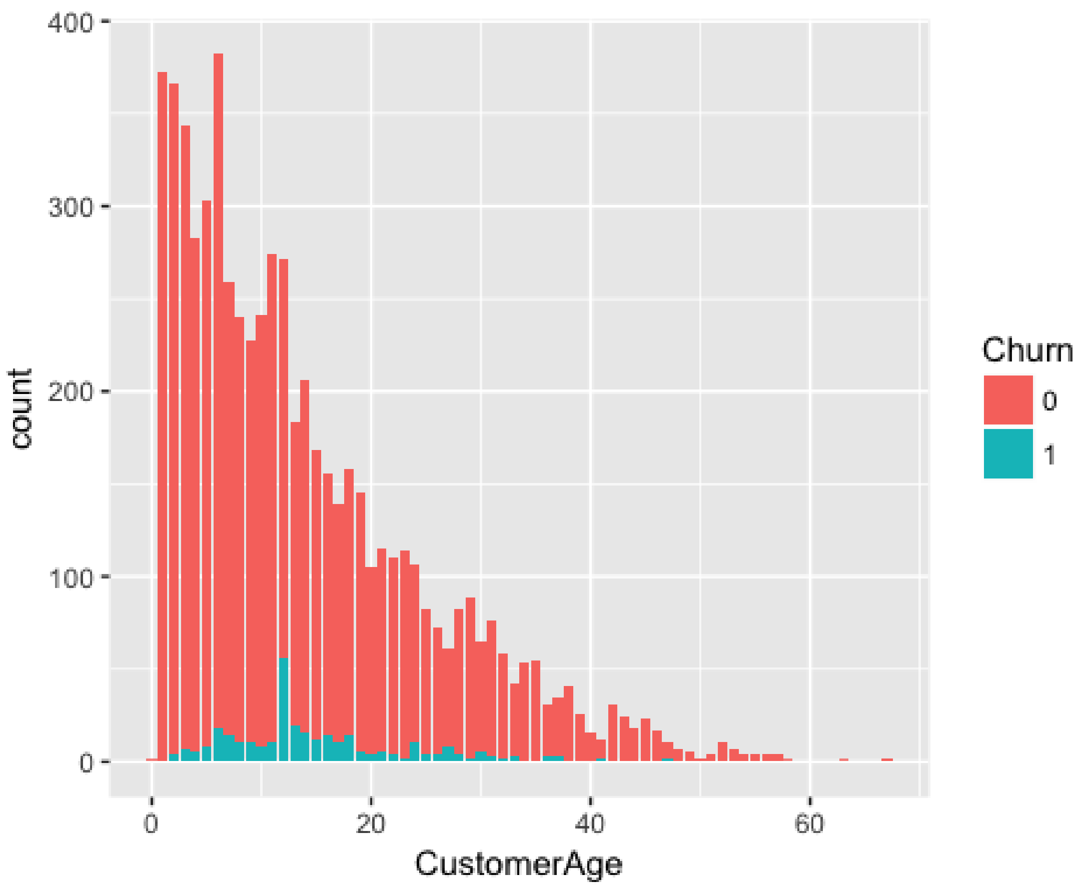
```
t.churn<-table(dm4$Churn)
prop.table(t.churn)*100
```

```
## 
##         0         1 
## 94.910982  5.089018
```

```
library(ggplot2)
ggplot(dm4, aes(x=CustomerAge)) + geom_bar()+facet_wrap(~Churn)
```



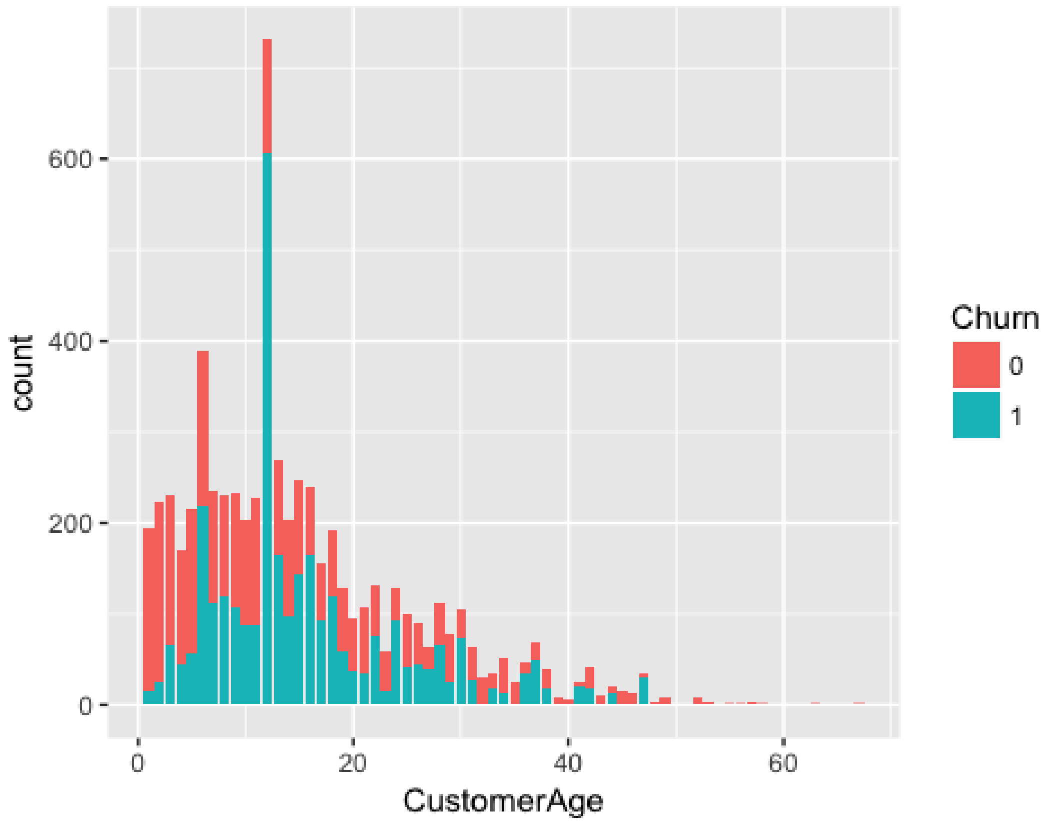

```r
ggplot(dm4, aes(x = CustomerAge, fill = Churn)) +    geom_bar()
```



From the above two plots we can see that the churn rate is largely distributed around age 5-15 months. Is Wall's belief about the dependence of churn rates on customer age supported by the data.

Using multiple algorithms to try and get best model to predict churn. The data is highly imbalanced and data balancing techniques has to be employed so that the prediction is not biased towards the majority class.

```
library(ROSE)

## Loaded ROSE 0.0-3

index <- sample(1:nrow(dm4),round(0.70*nrow(dm4)))
train <- dm4[index,]
test <- dm4[-index,]

# balancing the data by ovum sample
ov.data <- ovun.sample(Churn ~ ., data = train, method = "both", p=0.5, N=6347, s
eed = 321)$data
table(ov.data$Churn)

##
##    0    1
## 3177 3170

library(ggplot2)
ggplot(ov.data, aes(x = CustomerAge, fill = Churn)) +   geom_bar()
```



Neural Network

```
# Neural network

# We need to normalize data before applying neural network model
# Lets use min-max transformation to normalize data
maxs <- sapply(ov.data[-2], max)
mins <- sapply(ov.data[-2], min)
scaled <- as.data.frame(scale(ov.data[-2], center = mins, scale = maxs - mins))
scaled<-cbind(scaled,Churn=ov.data$Churn)

train_scaled <- scaled[index,]

# scaling test data
maxs <- sapply(dm4[-2], max)
mins <- sapply(dm4[-2], min)
scaled <- as.data.frame(scale(dm4[-2], center = mins, scale = maxs - mins))
scaled<-cbind(scaled,Churn=dm4$Churn)

test_scaled <- scaled[-index,]

library(nnet)
```

```r
nn2<-nnet(Churn ~ ., data=train_scaled, linout=F, size=10, decay=0.01, maxit=1000
)
```

```
## # weights:  131
## initial  value 3542.006564
## iter  10 value 2940.761945
## iter  20 value 2738.803457
## iter  30 value 2549.503663
## iter  40 value 2453.517084
## iter  50 value 2390.369656
## iter  60 value 2322.676703
## iter  70 value 2256.900219
## iter  80 value 2222.835310
## iter  90 value 2206.137038
## iter 100 value 2187.969955
## iter 110 value 2159.630919
## iter 120 value 2136.264502
## iter 130 value 2125.701717
## iter 140 value 2115.210643
## iter 150 value 2108.928045
## iter 160 value 2096.699848
## iter 170 value 2083.981254
## iter 180 value 2077.371202
## iter 190 value 2072.405202
## iter 200 value 2069.953139
## iter 210 value 2066.486393
## iter 220 value 2064.467716
## iter 230 value 2063.784087
## iter 240 value 2062.939133
## iter 250 value 2062.008699
## iter 260 value 2061.078766
## iter 270 value 2060.515484
## iter 280 value 2059.346806
## iter 290 value 2053.806258
## iter 300 value 2045.999365
## iter 310 value 2038.856208
## iter 320 value 2034.044300
## iter 330 value 2026.499285
## iter 340 value 2019.274602
## iter 350 value 2012.349294
## iter 360 value 2010.217059
## iter 370 value 2008.199756
## iter 380 value 2006.688880
## iter 390 value 2005.968392
## iter 400 value 2005.456122
## iter 410 value 2005.301023
## iter 420 value 2005.276438
## iter 430 value 2005.264008
## iter 440 value 2005.251408
## iter 450 value 2005.246120
## iter 460 value 2005.243846
## iter 470 value 2005.242087
## final  value 2005.241589
## converged
```

```
summary(nn2)

## a 11-10-1 network with 131 weights
## options were - entropy fitting  decay=0.01
##   b->h1  i1->h1  i2->h1  i3->h1  i4->h1  i5->h1  i6->h1  i7->h1  i8->h1
##   10.97   -8.92   -5.15  -11.65    1.44  -11.80   12.31    3.80    0.11
##  i9->h1 i10->h1 i11->h1
##    5.63   -9.65   -8.13
##   b->h2  i1->h2  i2->h2  i3->h2  i4->h2  i5->h2  i6->h2  i7->h2  i8->h2
##   10.48   -3.20    6.09   15.10    0.54    1.65   -7.52  -19.31   -3.46
##  i9->h2 i10->h2 i11->h2
##    0.30   -0.36  -14.89
##   b->h3  i1->h3  i2->h3  i3->h3  i4->h3  i5->h3  i6->h3  i7->h3  i8->h3
##   -6.01    3.26   -2.19    2.67  -10.56   13.65   -0.03   -7.00    5.33
##  i9->h3 i10->h3 i11->h3
##   -3.08   -5.10    5.23
##   b->h4  i1->h4  i2->h4  i3->h4  i4->h4  i5->h4  i6->h4  i7->h4  i8->h4
##   -5.34    1.76   -9.69   -1.36   -6.25    1.94    2.44  -12.32    1.15
##  i9->h4 i10->h4 i11->h4
##   -9.56   -2.24   20.45
##   b->h5  i1->h5  i2->h5  i3->h5  i4->h5  i5->h5  i6->h5  i7->h5  i8->h5
##    2.60    3.00    4.99    5.64   -1.02   -5.08    8.68   -2.20   -1.85
##  i9->h5 i10->h5 i11->h5
##   -3.62  -10.63   -1.69
##   b->h6  i1->h6  i2->h6  i3->h6  i4->h6  i5->h6  i6->h6  i7->h6  i8->h6
##   -5.14  -13.57  -15.98   16.89  -16.83   -0.49    2.38   -5.36    1.21
##  i9->h6 i10->h6 i11->h6
##   -1.21   -3.74    4.95
##   b->h7  i1->h7  i2->h7  i3->h7  i4->h7  i5->h7  i6->h7  i7->h7  i8->h7
##    5.72   -6.96   -3.09   -0.23  -13.14   -6.98    6.04  -10.08   12.95
##  i9->h7 i10->h7 i11->h7
##    8.69    2.26   -9.81
##   b->h8  i1->h8  i2->h8  i3->h8  i4->h8  i5->h8  i6->h8  i7->h8  i8->h8
##   -3.09   -9.13   -8.76   10.79    0.86   -7.42   -2.01    1.29    0.13
##  i9->h8 i10->h8 i11->h8
##   -1.70   13.53    1.48
##   b->h9  i1->h9  i2->h9  i3->h9  i4->h9  i5->h9  i6->h9  i7->h9  i8->h9
##   -5.51   -0.26    3.56    3.85    1.63    2.26   -2.72    5.16  -12.32
##  i9->h9 i10->h9 i11->h9
##   -5.87  -24.57    6.00
##   b->h10  i1->h10  i2->h10  i3->h10  i4->h10  i5->h10  i6->h10  i7->h10
##    -1.71     3.29   -19.03     1.70    -6.45   -14.45     6.03   -23.29
##  i8->h10  i9->h10 i10->h10 i11->h10
##     2.69   -11.64    -2.98    31.13
##   b->o  h1->o  h2->o  h3->o  h4->o  h5->o  h6->o  h7->o  h8->o  h9->o
##  -5.34  20.23   8.88  17.59 -26.70 -18.62  15.74 -24.73 -31.50  21.26
## h10->o
##  16.81


nn2.preds<-predict(nn2, test_scaled,type='class')
table(nn2.preds)
```

```
## nn2.preds
##    0    1
## 1286  618
```

```r
library(caret)
```

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

```r
confusionMatrix(nn2.preds,test_scaled$Churn,positive = '1',dnn=c('Predicted','Actual'))
```

```
## Confusion Matrix and Statistics
##
##          Actual
## Predicted    0    1
##         0 1233   53
##         1  561   57
##
##                Accuracy : 0.6775
##                  95% CI : (0.656, 0.6985)
##     No Information Rate : 0.9422
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0649
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.51818
##             Specificity : 0.68729
##          Pos Pred Value : 0.09223
##          Neg Pred Value : 0.95879
##              Prevalence : 0.05777
##          Detection Rate : 0.02994
##    Detection Prevalence : 0.32458
##       Balanced Accuracy : 0.60274
##
##        'Positive' Class : 1
```

Random Forest

```r
# Running Random Forest
train<-ov.data
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:ggplot2':
##
##     margin
```

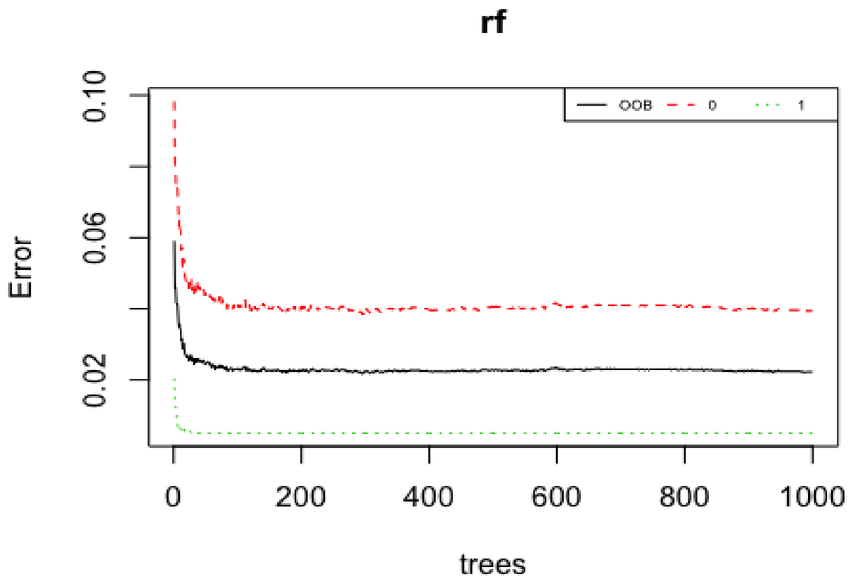
```r
rf = randomForest(Churn ~ ., data = train, ntree = 1000, proximity = T, replace=
T, importance = T, mtry = sqrt(ncol(train)-1))
rf
```

```
##
## Call:
##  randomForest(formula = Churn ~ ., data = train, ntree = 1000,      proximity
= T, replace = T, importance = T, mtry = sqrt(ncol(train) -          1))
##                Type of random forest: classification
##                      Number of trees: 1000
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 2.22%
## Confusion matrix:
##      0    1 class.error
## 0 3052  125 0.039345294
## 1   16 3154 0.005047319
```

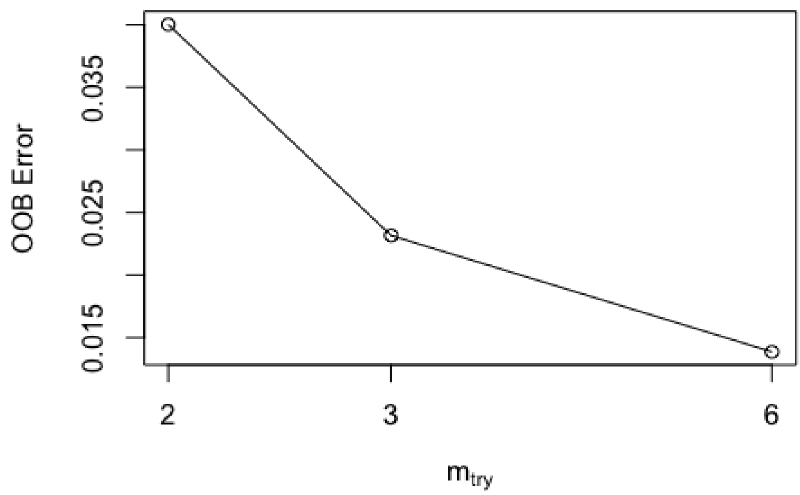
```r
# Confusion matrix
library(caret)
confusionMatrix(predict(rf, type = "class", newdata = test), test$Churn, positive
= '1', dnn = c("Predictions", "Actual Values"))
```

```
## Confusion Matrix and Statistics
##
##            Actual Values
## Predictions    0    1
##           0 1722   76
##           1   91   15
##
##                Accuracy : 0.9123
##                  95% CI : (0.8987, 0.9246)
##     No Information Rate : 0.9522
##     P-Value [Acc > NIR] : 1.0000
##
##                   Kappa : 0.1063
##  Mcnemar's Test P-Value : 0.2787
##
##             Sensitivity : 0.164835
##             Specificity : 0.949807
##          Pos Pred Value : 0.141509
##          Neg Pred Value : 0.957731
##              Prevalence : 0.047794
##          Detection Rate : 0.007878
##    Detection Prevalence : 0.055672
##       Balanced Accuracy : 0.557321
##
##        'Positive' Class : 1
##
```

```r
plot(rf)
legend("topright", legend = colnames(rf$err.rate), cex = 0.5, lty = c(1,2,3), col
= c(1,2,3), horiz = T)
```




```
# Important variables
importance(rf, type = 2)

##                          MeanDecreaseGini
## CustomerAge                     573.64837
## CHIScoreMonth0                  481.92214
## CHIScore0_1                     371.91405
## SupportCasesMonth0               68.58775
## SupportCases0_1                  91.69479
## SPMonth_0                        50.45195
## SP0_1                            65.96992
## Logins0_1                       374.72007
## BlogArticles0_1                 121.39965
## Views0_1                        366.45460
## DaysSinceLastLogin0_1           405.64031

# best mtry with tuneRF
bestMtry<-tuneRF(train[,-2], train[,2] , stepFactor =2.0,improve = TRUE )

## mtry = 3  OOB error = 2.32%
## Searching left ...
## mtry = 2     OOB error = 4%
## -0.7278912 TRUE
## Searching right ...
## mtry = 6     OOB error = 1.39%
## 0.4013605 TRUE
```




```
# Random Forest with optimum mtry
rf_opt_mtry = randomForest(Churn ~ ., data = train, ntree = 1000, proximity = T,
replace= T, importance = T, mtry = 6)
rf_opt_mtry

##
## Call:
##  randomForest(formula = Churn ~ ., data = train, ntree = 1000,      proximity
= T, replace = T, importance = T, mtry = 6)
##                Type of random forest: classification
##                      Number of trees: 1000
## No. of variables tried at each split: 6
##
##         OOB estimate of  error rate: 1.5%
## Confusion matrix:
##      0    1 class.error
## 0 3098   79 0.024866226
## 1   16 3154 0.005047319

confusionMatrix(predict(rf_opt_mtry, type = "class", newdata = test), test$Churn,
dnn = c("Predictions", "Actual Values"))

## Confusion Matrix and Statistics
##
##            Actual Values
## Predictions    0    1
##           0 1749   81
##           1   64   10
##
##                Accuracy : 0.9238
##                  95% CI : (0.911, 0.9354)
##     No Information Rate : 0.9522
##     P-Value [Acc > NIR] : 1.0000
##
##                   Kappa : 0.0819
##  Mcnemar's Test P-Value : 0.1839
```

```
##
##             Sensitivity : 0.9647
##             Specificity : 0.1099
##          Pos Pred Value : 0.9557
##          Neg Pred Value : 0.1351
##              Prevalence : 0.9522
##          Detection Rate : 0.9186
##    Detection Prevalence : 0.9611
##       Balanced Accuracy : 0.5373
##
##        'Positive' Class : 0
##
```

```
# K fold cross validation
k <- 10
nmethod <- 1
folds <- cut(seq(1,nrow(ov.data)),breaks=k,labels=FALSE)
models.err <- matrix(-1,k,nmethod, dimnames=list(paste0("Fold", 1:k), c("rf")))

for(i in 1:k)
{
  testIndexes <- which(folds==i, arr.ind=TRUE)
  Test <- ov.data[testIndexes, ]
  Train <- ov.data[-testIndexes, ]

  library(randomForest)
  rf <- randomForest(Churn~., data = Train, ntree = 10, mtry = 6)
  predicted <- predict(rf, newdata = Test, type = "class")
  models.err[i] <- mean(Test$Churn != predicted)
}

models.err
```

```
##                  rf
## Fold1  0.029921260
## Fold2  0.050393701
## Fold3  0.020504732
## Fold4  0.031496063
## Fold5  0.034645669
## Fold6  0.006309148
## Fold7  0.004724409
## Fold8  0.001577287
## Fold9  0.004724409
## Fold10 0.011023622
```

```
mean(models.err)
```

```
## [1] 0.01953203
```

Logistic Regression

```r
train<-ov.data
#training
start.train <- glm(Churn~1, data = train, family = "binomial")
full.train <- glm(Churn~., data = train, family = "binomial")
train.step <- step(start.train, scope =list(lower= start.train, upper = full.trai
n), direction = "both")
```

```
## Start:  AIC=8800.8
## Churn ~ 1
##
##                          Df Deviance    AIC
## + CHIScoreMonth0          1   8522.5 8526.5
## + SPMonth_0               1   8669.7 8673.7
## + CHIScore0_1             1   8686.5 8690.5
## + SupportCasesMonth0      1   8689.1 8693.1
## + DaysSinceLastLogin0_1   1   8713.7 8717.7
## + CustomerAge             1   8743.2 8747.2
## + Views0_1                1   8772.9 8776.9
## + Logins0_1               1   8780.6 8784.6
## + BlogArticles0_1         1   8786.7 8790.7
## + SupportCases0_1         1   8794.1 8798.1
## <none>                        8798.8 8800.8
## + SP0_1                   1   8796.9 8800.9
##
## Step:  AIC=8526.46
## Churn ~ CHIScoreMonth0
##
##                          Df Deviance    AIC
## + CustomerAge             1   8408.7 8414.7
## + CHIScore0_1             1   8483.4 8489.4
## + DaysSinceLastLogin0_1   1   8484.1 8490.1
## + SPMonth_0               1   8485.6 8491.6
## + Views0_1                1   8489.3 8495.3
## + SupportCasesMonth0      1   8493.1 8499.1
## + SupportCases0_1         1   8515.4 8521.4
## + BlogArticles0_1         1   8518.0 8524.0
## + Logins0_1               1   8519.9 8525.9
## <none>                        8522.5 8526.5
## + SP0_1                   1   8521.4 8527.4
## - CHIScoreMonth0          1   8798.8 8800.8
##
## Step:  AIC=8414.74
## Churn ~ CHIScoreMonth0 + CustomerAge
##
##                          Df Deviance    AIC
## + DaysSinceLastLogin0_1   1   8370.9 8378.9
## + Views0_1                1   8375.6 8383.6
## + SPMonth_0               1   8387.4 8395.4
## + CHIScore0_1             1   8388.8 8396.8
## + Logins0_1               1   8395.2 8403.2
## + SupportCasesMonth0      1   8396.0 8404.0
## + SupportCases0_1         1   8397.8 8405.8
## + BlogArticles0_1         1   8406.0 8414.0
## <none>                        8408.7 8414.7
```

```
## + SP0_1                   1   8407.5 8415.5
## - CustomerAge             1   8522.5 8526.5
## - CHIScoreMonth0          1   8743.2 8747.2
##
## Step:  AIC=8378.89
## Churn ~ CHIScoreMonth0 + CustomerAge + DaysSinceLastLogin0_1
##
##                          Df Deviance    AIC
## + Views0_1                1   8338.9 8348.9
## + CHIScore0_1             1   8343.6 8353.6
## + SPMonth_0               1   8351.9 8361.9
## + Logins0_1               1   8358.2 8368.2
## + SupportCasesMonth0      1   8358.9 8368.9
## + SupportCases0_1         1   8359.8 8369.8
## + BlogArticles0_1         1   8367.6 8377.6
## <none>                        8370.9 8378.9
## + SP0_1                   1   8369.9 8379.9
## - DaysSinceLastLogin0_1   1   8408.7 8414.7
## - CustomerAge             1   8484.1 8490.1
## - CHIScoreMonth0          1   8654.9 8660.9
##
## Step:  AIC=8348.87
## Churn ~ CHIScoreMonth0 + CustomerAge + DaysSinceLastLogin0_1 +
##     Views0_1
##
##                          Df Deviance    AIC
## + CHIScore0_1             1   8309.4 8321.4
## + SPMonth_0               1   8318.1 8330.1
## + SupportCasesMonth0      1   8318.6 8330.6
## + Logins0_1               1   8328.1 8340.1
## + SupportCases0_1         1   8332.9 8344.9
## + BlogArticles0_1         1   8335.9 8347.9
## <none>                        8338.9 8348.9
## + SP0_1                   1   8337.8 8349.8
## - Views0_1                1   8370.9 8378.9
## - DaysSinceLastLogin0_1   1   8375.6 8383.6
## - CustomerAge             1   8451.9 8459.9
## - CHIScoreMonth0          1   8630.8 8638.8
##
## Step:  AIC=8321.41
## Churn ~ CHIScoreMonth0 + CustomerAge + DaysSinceLastLogin0_1 +
##     Views0_1 + CHIScore0_1
##
##                          Df Deviance    AIC
## + Logins0_1               1   8282.9 8296.9
## + SupportCases0_1         1   8293.7 8307.7
## + SPMonth_0               1   8293.9 8307.9
## + SupportCasesMonth0      1   8295.7 8309.7
## <none>                        8309.4 8321.4
## + BlogArticles0_1         1   8309.4 8323.4
## + SP0_1                   1   8309.4 8323.4
## - CHIScore0_1             1   8338.9 8348.9
## - Views0_1                1   8343.6 8353.6
## - DaysSinceLastLogin0_1   1   8353.8 8363.8
```

```
## - CustomerAge            1   8399.9 8409.9
## - CHIScoreMonth0         1   8518.2 8528.2
##
## Step:  AIC=8296.94
## Churn ~ CHIScoreMonth0 + CustomerAge + DaysSinceLastLogin0_1 +
##     Views0_1 + CHIScore0_1 + Logins0_1
##
##                          Df Deviance    AIC
## + SupportCasesMonth0     1   8256.7 8272.7
## + SPMonth_0              1   8257.5 8273.5
## + SupportCases0_1        1   8275.8 8291.8
## <none>                       8282.9 8296.9
## + SP0_1                  1   8281.5 8297.5
## + BlogArticles0_1        1   8282.7 8298.7
## - Logins0_1              1   8309.4 8321.4
## - Views0_1               1   8314.6 8326.6
## - CHIScore0_1            1   8328.1 8340.1
## - DaysSinceLastLogin0_1  1   8328.6 8340.6
## - CustomerAge            1   8385.8 8397.8
## - CHIScoreMonth0         1   8517.5 8529.5
##
## Step:  AIC=8272.69
## Churn ~ CHIScoreMonth0 + CustomerAge + DaysSinceLastLogin0_1 +
##     Views0_1 + CHIScore0_1 + Logins0_1 + SupportCasesMonth0
##
##                          Df Deviance    AIC
## + SupportCases0_1        1   8216.5 8234.5
## + SPMonth_0              1   8251.9 8269.9
## <none>                       8256.7 8272.7
## + BlogArticles0_1        1   8256.4 8274.4
## + SP0_1                  1   8256.5 8274.5
## - SupportCasesMonth0     1   8282.9 8296.9
## - Logins0_1              1   8295.7 8309.7
## - CHIScore0_1            1   8296.2 8310.2
## - Views0_1               1   8297.5 8311.5
## - DaysSinceLastLogin0_1  1   8299.7 8313.7
## - CustomerAge            1   8346.2 8360.2
## - CHIScoreMonth0         1   8449.5 8463.5
##
## Step:  AIC=8234.54
## Churn ~ CHIScoreMonth0 + CustomerAge + DaysSinceLastLogin0_1 +
##     Views0_1 + CHIScore0_1 + Logins0_1 + SupportCasesMonth0 +
##     SupportCases0_1
##
##                          Df Deviance    AIC
## + SP0_1                  1   8203.9 8223.9
## + SPMonth_0              1   8212.9 8232.9
## <none>                       8216.5 8234.5
## + BlogArticles0_1        1   8216.3 8236.3
## - Logins0_1              1   8242.2 8258.2
## - Views0_1               1   8250.8 8266.8
## - SupportCases0_1        1   8256.7 8272.7
## - DaysSinceLastLogin0_1  1   8261.1 8277.1
## - CHIScore0_1            1   8267.4 8283.4
```

```
## - SupportCasesMonth0     1   8275.8 8291.8
## - CustomerAge            1   8292.1 8308.1
## - CHIScoreMonth0         1   8352.3 8368.3
##
## Step:  AIC=8223.91
## Churn ~ CHIScoreMonth0 + CustomerAge + DaysSinceLastLogin0_1 +
##     Views0_1 + CHIScore0_1 + Logins0_1 + SupportCasesMonth0 +
##     SupportCases0_1 + SP0_1
##
##                         Df Deviance    AIC
## <none>                      8203.9 8223.9
## + SPMonth_0              1   8203.4 8225.4
## + BlogArticles0_1        1   8203.7 8225.7
## - SP0_1                  1   8216.5 8234.5
## - Views0_1               1   8232.4 8250.4
## - Logins0_1              1   8232.5 8250.5
## - DaysSinceLastLogin0_1  1   8247.3 8265.3
## - CHIScore0_1            1   8253.8 8271.8
## - SupportCases0_1        1   8256.5 8274.5
## - SupportCasesMonth0     1   8265.5 8283.5
## - CustomerAge            1   8283.3 8301.3
## - CHIScoreMonth0         1   8342.3 8360.3
```

```
summary(train.step)
```

```
##
## Call:
## glm(formula = Churn ~ CHIScoreMonth0 + CustomerAge + DaysSinceLastLogin0_1 +
##     Views0_1 + CHIScore0_1 + Logins0_1 + SupportCasesMonth0 +
##     SupportCases0_1 + SP0_1, family = "binomial", data = train)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -2.100  -1.138  -0.134   1.096   1.881
##
## Coefficients:
##                         Estimate Std. Error z value Pr(>|z|)
## (Intercept)            1.185e-01  5.344e-02   2.218 0.026539 *
## CHIScoreMonth0        -5.933e-03  5.106e-04 -11.621  < 2e-16 ***
## CustomerAge            2.458e-02  2.792e-03   8.804  < 2e-16 ***
## DaysSinceLastLogin0_1  8.861e-03  1.360e-03   6.516 7.21e-11 ***
## Views0_1              -9.870e-05  2.201e-05  -4.484 7.33e-06 ***
## CHIScore0_1           -7.313e-03  1.051e-03  -6.957 3.47e-12 ***
## Logins0_1              3.676e-03  6.836e-04   5.378 7.53e-08 ***
## SupportCasesMonth0    -2.217e-01  3.082e-02  -7.193 6.36e-13 ***
## SupportCases0_1        2.210e-01  3.284e-02   6.731 1.68e-11 ***
## SP0_1                 -8.748e-02  2.465e-02  -3.550 0.000386 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 8798.8  on 6346  degrees of freedom
## Residual deviance: 8203.9  on 6337  degrees of freedom
```

```
## AIC: 8223.9
##
## Number of Fisher Scoring iterations: 5
```

```
#predict
result.test <- predict(train.step, newdata = test, type ='response')
fitted.results <- ifelse(result.test > 0.5,1,0)
misClasificError <- mean(fitted.results != test$Churn)
print(paste('Accuracy',1-misClasificError))
```

```
## [1] "Accuracy 0.548844537815126"
```

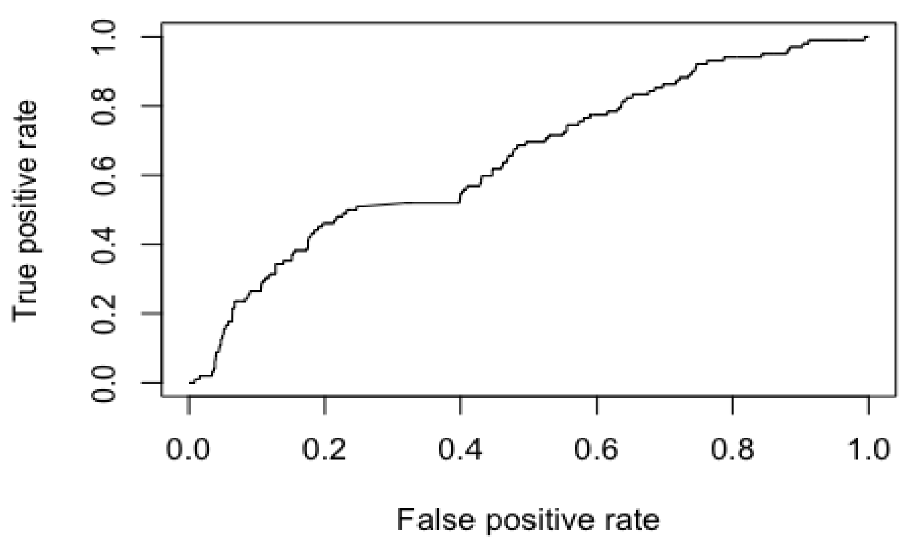
```
#ROC
library(ROCR)
```

```
## Loading required package: gplots
```

```
##
## Attaching package: 'gplots'
```

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

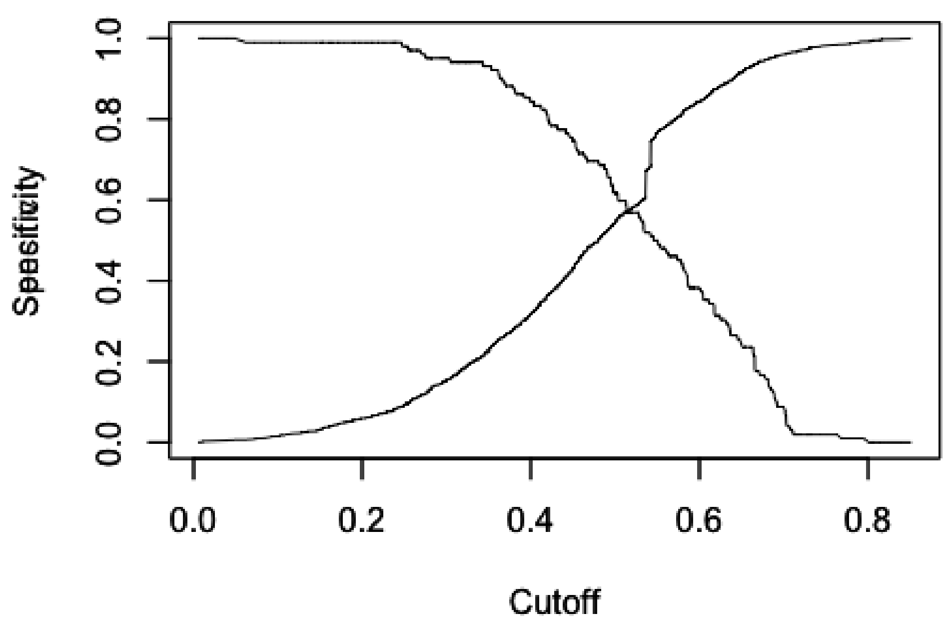
```
pr <- prediction(result.test, test$Churn)
prf <- performance(pr, measure = "tpr", x.measure = "fpr")
plot(prf)
```




```
#Threshold
opt.cut <- function(prf, pr){
  cut.ind <- mapply(FUN = function(x,y,p){d=(x-0)^2+(y-1)^2
  ind<- which(d==min(d))
  c(recall = y[[ind]], specificity = 1-x[[ind]],cutoff = p[[ind]])},prf@x.values,
prf@y.values,prf@alpha.values)
}
print(opt.cut(prf,pr))
```

```
##                   [,1]
## recall      0.5098039
## specificity 0.7524972
## cutoff      0.5452102
```

```r
perfspec <- performance(prediction.obj = pr, measure="spec", x.measure="cutoff")
plot(perfspec)
par(new=TRUE)

perfsens <- performance(prediction.obj = pr, measure="sens", x.measure="cutoff")
plot(perfsens)
```




```r
#Accuracy acc to threshold
fitted.results <- ifelse(result.test > 0.537,1,0)
misClasificError <- mean(fitted.results != test$Churn)
print(paste('Accuracy',1-misClasificError))

## [1] "Accuracy 0.665441176470588"
```

```r
library(caret)
confusionMatrix(test$Churn,fitted.results,positive = "1", dnn = c("Actual Values"
,'Predictions'))
```

```
## Confusion Matrix and Statistics
##
##              Predictions
## Actual Values    0    1
##             0 1214  588
##             1   49   53
##
##                Accuracy : 0.6654
##                  95% CI : (0.6437, 0.6866)
##     No Information Rate : 0.6633
##     P-Value [Acc > NIR] : 0.4336
##
##                   Kappa : 0.0553
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.08268
##             Specificity : 0.96120
##          Pos Pred Value : 0.51961
##          Neg Pred Value : 0.67370
##              Prevalence : 0.33666
##          Detection Rate : 0.02784
##    Detection Prevalence : 0.05357
##       Balanced Accuracy : 0.52194
##
##        'Positive' Class : 1
##
```

- **Finding top 100 instances that is most likely to be churning.**

```r
pred2.f<-predict(rf_opt_mtry, newdata=test[-2],type='prob')
nr <- (1:nrow(test))[order(pred2.f[,2], decreasing=T)[1:100]]
```