Different Models for Churn Data

Malvika Singh

7/18/2019

In this document, we will be using logistic regression, generalized linear model, random forest and naive bayes classification models to observe and analyze which one of them performs well on our data.

Why do Predictive Analytics?

The focus of this Predictive analytics is to have maximum Return on Investment(ROI). It is strategic and effective to target smaller high revenue generating group who have a higher risk of churning out. Preventing their churning rate is definitely more efficient and fruitful. Customer Retention Cost needs to be minimized by using better predictive models. We will begin with Logistic Regression. Logistic regression is a linear classifier, which makes it easier to interpret than non-linear models. At the same time, because it's a linear model, it has a high bias towards this type of fit, so it may not perform well on non-linear data. Since I am not sure of whether the data set provided is linear or not, I will begin with Logistic Regression.

Step 1: Splitting data into training and test

I will be splitting my data into a training set which comprises of data for the first two months, and test set which comprises of data for the third month. I am going to remove the customerID feature because it's unique for each observation, and probably won't add valuable information to my model.

```
library(randomForest)
```

```
## randomForest 4.6-14
```

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

```
library(caret)
```

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

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

```
##
## Attaching package: 'ggplot2'
```

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

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

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

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

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

```
library(caret)
```

```
data <- read.csv("C:/Users/esaminl/Downloads/TelCh.csv",header = TRUE, sep = ",")
data <- data %>%
  mutate(TotalCharges = replace(TotalCharges,
                                is.na(TotalCharges),
                                mean(TotalCharges, na.rm = T)))
data <- data %>% mutate_if(is.character, as.factor)
data$SeniorCitizen <- as.factor(data$SeniorCitizen)
# removing customerID; doesn't add any value to the model
data <- data %>% select(-customerID)

# train/test split; 75%/25%

# setting the seed for reproducibility
set.seed(5)
inTrain <- createDataPartition(y = data$Churn, p=0.75, list=FALSE)

data_train <- data[inTrain,]
data_test <- data[-inTrain,]

# fitting the model
# fitting the model
fit <- glm(Churn~., data=data_train, family=binomial)

# making predictions
churn.probs <- predict(fit, data_test, type="response")
```

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

```
head(churn.probs)
```

```
##         2         4         5         6         9        16
## 0.04142437 0.02909549 0.68863232 0.79510663 0.60564010 0.03340725
```

```
contrasts(data$Churn)  # Yes = 1, No = 0
```

```
##     Yes
## No    0
## Yes   1
```

```
glm.pred = rep("No", length(churn.probs))
glm.pred[churn.probs > 0.5] = "Yes"

confusionMatrix(table(glm.pred, data_test$Churn))
```

```
## Confusion Matrix and Statistics
##
##
## glm.pred   No  Yes
##      No  1168  213
##      Yes  125  254
##
##                Accuracy : 0.808
##                  95% CI : (0.7888, 0.8261)
##     No Information Rate : 0.7347
##     P-Value [Acc > NIR] : 3.696e-13
##
##                   Kappa : 0.4759
##
##  Mcnemar's Test P-Value : 2.221e-06
##
##             Sensitivity : 0.9033
##             Specificity : 0.5439
##          Pos Pred Value : 0.8458
##          Neg Pred Value : 0.6702
##              Prevalence : 0.7347
##          Detection Rate : 0.6636
##    Detection Prevalence : 0.7847
##       Balanced Accuracy : 0.7236
##
##        'Positive' Class : No
##
```

We are interested in True Positive Rate(Sensitivity); that is, when it's actually yes, how often does it predict yes and False Positive Rate, that is, when it's actually a no how often does it predict a yes. If any of these numbers is low, then our churn out rate will either be deceptively low(TP) or deceptively inflated(FP). Since the sensitivity and recall are 0.00 in this case, even if the accuracy is high, the model is poor.

```
# converting probabilities to classes; "Yes" or "No"
contrasts(data$Churn)  # Yes = 1, No = 0
```

```
##     Yes
## No    0
## Yes   1
```

```
glm.pred = rep("No", length(churn.probs))
glm.pred[churn.probs > 0.5] = "Yes"

confusionMatrix(table(glm.pred, data_test$Churn))
```

```
## Confusion Matrix and Statistics
##
##
## glm.pred   No  Yes
##      No  1168  213
##      Yes  125  254
##
##                Accuracy : 0.808
##                  95% CI : (0.7888, 0.8261)
##     No Information Rate : 0.7347
##     P-Value [Acc > NIR] : 3.696e-13
##
##                   Kappa : 0.4759
##
##  Mcnemar's Test P-Value : 2.221e-06
##
##             Sensitivity : 0.9033
##             Specificity : 0.5439
##          Pos Pred Value : 0.8458
##          Neg Pred Value : 0.6702
##              Prevalence : 0.7347
##          Detection Rate : 0.6636
##    Detection Prevalence : 0.7847
##       Balanced Accuracy : 0.7236
##
##        'Positive' Class : No
##
```

Feature Selection

```
summary(fit)
```

```
##
## Call:
## glm(formula = Churn ~ ., family = binomial, data = data_train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.8863  -0.6847  -0.2883   0.7310   3.3551
##
## Coefficients: (7 not defined because of singularities)
##                                    Estimate Std. Error z value
## (Intercept)                         1.217e+00  9.436e-01   1.290
## genderMale                         -6.886e-03  7.457e-02  -0.092
## SeniorCitizen1                      8.951e-02  9.835e-02   0.910
## PartnerYes                         -4.538e-02  9.011e-02  -0.504
## DependentsYes                      -3.981e-02  1.019e-01  -0.391
## tenure                             -5.394e-02  6.901e-03  -7.816
## PhoneServiceYes                     3.575e-01  7.514e-01   0.476
## MultipleLinesNo phone service              NA         NA      NA
## MultipleLinesYes                    4.793e-01  2.044e-01   2.344
## InternetServiceFiber optic          1.972e+00  9.236e-01   2.136
## InternetServiceNo                  -1.904e+00  9.353e-01  -2.036
## OnlineSecurityNo internet service          NA         NA      NA
## OnlineSecurityYes                  -1.690e-01  2.082e-01  -0.812
## OnlineBackupNo internet service            NA         NA      NA
## OnlineBackupYes                     9.485e-02  2.033e-01   0.467
## DeviceProtectionNo internet service        NA         NA      NA
## DeviceProtectionYes                 2.156e-01  2.034e-01   1.060
## TechSupportNo internet service             NA         NA      NA
## TechSupportYes                     -1.311e-01  2.084e-01  -0.629
## StreamingTVNo internet service             NA         NA      NA
## StreamingTVYes                      6.836e-01  3.773e-01   1.812
## StreamingMoviesNo internet service         NA         NA      NA
## StreamingMoviesYes                  6.974e-01  3.786e-01   1.842
## ContractOne year                   -6.886e-01  1.251e-01  -5.506
## ContractTwo year                   -1.438e+00  2.021e-01  -7.119
## PaperlessBillingYes                 3.714e-01  8.579e-02   4.329
## PaymentMethodCredit card (automatic) -2.352e-01  1.318e-01  -1.785
## PaymentMethodElectronic check       2.692e-01  1.088e-01   2.474
## PaymentMethodMailed check          -3.950e-02  1.314e-01  -0.301
## MonthlyCharges                     -4.703e-02  3.676e-02  -1.279
## TotalCharges                        2.564e-04  7.907e-05   3.242
##                                    Pr(>|z|)
## (Intercept)                         0.19717
## genderMale                          0.92643
## SeniorCitizen1                      0.36276
## PartnerYes                          0.61454
## DependentsYes                       0.69591
## tenure                             5.44e-15 ***
## PhoneServiceYes                     0.63420
## MultipleLinesNo phone service            NA
## MultipleLinesYes                    0.01905 *
## InternetServiceFiber optic          0.03270 *
## InternetServiceNo                   0.04177 *
```

```
## OnlineSecurityNo internet service          NA
## OnlineSecurityYes                      0.41696
## OnlineBackupNo internet service            NA
## OnlineBackupYes                        0.64081
## DeviceProtectionNo internet service        NA
## DeviceProtectionYes                    0.28904
## TechSupportNo internet service             NA
## TechSupportYes                         0.52923
## StreamingTVNo internet service             NA
## StreamingTVYes                         0.07000 .
## StreamingMoviesNo internet service         NA
## StreamingMoviesYes                     0.06551 .
## ContractOne year                      3.68e-08 ***
## ContractTwo year                      1.09e-12 ***
## PaperlessBillingYes                   1.50e-05 ***
## PaymentMethodCredit card (automatic)  0.07431 .
## PaymentMethodElectronic check         0.01335 *
## PaymentMethodMailed check             0.76370
## MonthlyCharges                        0.20083
## TotalCharges                          0.00119 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 6113.6  on 5282  degrees of freedom
## Residual deviance: 4394.0  on 5259  degrees of freedom
## AIC: 4442
##
## Number of Fisher Scoring iterations: 6
```

```
varImp(fit)
```

```
##                                       Overall
## genderMale                         0.09233886
## SeniorCitizen1                     0.91011530
## PartnerYes                         0.50360441
## DependentsYes                      0.39085161
## tenure                             7.81623229
## PhoneServiceYes                    0.47581683
## MultipleLinesYes                   2.34448095
## InternetServiceFiber optic         2.13572954
## InternetServiceNo                  2.03576272
## OnlineSecurityYes                  0.81171414
## OnlineBackupYes                    0.46656109
## DeviceProtectionYes                1.06022456
## TechSupportYes                     0.62918448
## StreamingTVYes                     1.81190282
## StreamingMoviesYes                 1.84176756
## ContractOne year                   5.50554026
## ContractTwo year                   7.11851746
## PaperlessBillingYes                4.32904589
## PaymentMethodCredit card (automatic) 1.78468062
## PaymentMethodElectronic check      2.47431686
## PaymentMethodMailed check          0.30062814
## MonthlyCharges                     1.27919199
## TotalCharges                       3.24209276
```

Even when we filter and implement with the important features, we do not find a better model representation.

```
churn.rf = randomForest(Churn~., data = data_train, importance = T)

churn.rf
```

```
##
## Call:
##  randomForest(formula = Churn ~ ., data = data_train, importance = T)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
##         OOB estimate of  error rate: 20.63%
## Confusion matrix:
##       No Yes class.error
## No  3518 363  0.09353259
## Yes  727 675  0.51854494
```

```
churn.predict.prob <- predict(churn.rf, data_test, type="prob")

churn.predict <- predict(churn.rf, data_test)
confusionMatrix(churn.predict, data_test$Churn, positive = "Yes")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   No  Yes
##        No  1182  226
##        Yes  111  241
##
##                Accuracy : 0.8085
##                  95% CI : (0.7893, 0.8267)
##     No Information Rate : 0.7347
##     P-Value [Acc > NIR] : 2.414e-13
##
##                   Kappa : 0.4669
##
##  Mcnemar's Test P-Value : 5.299e-10
##
##             Sensitivity : 0.5161
##             Specificity : 0.9142
##          Pos Pred Value : 0.6847
##          Neg Pred Value : 0.8395
##              Prevalence : 0.2653
##          Detection Rate : 0.1369
##    Detection Prevalence : 0.2000
##       Balanced Accuracy : 0.7151
##
##        'Positive' Class : Yes
##
```

```
library(ROCR)
```

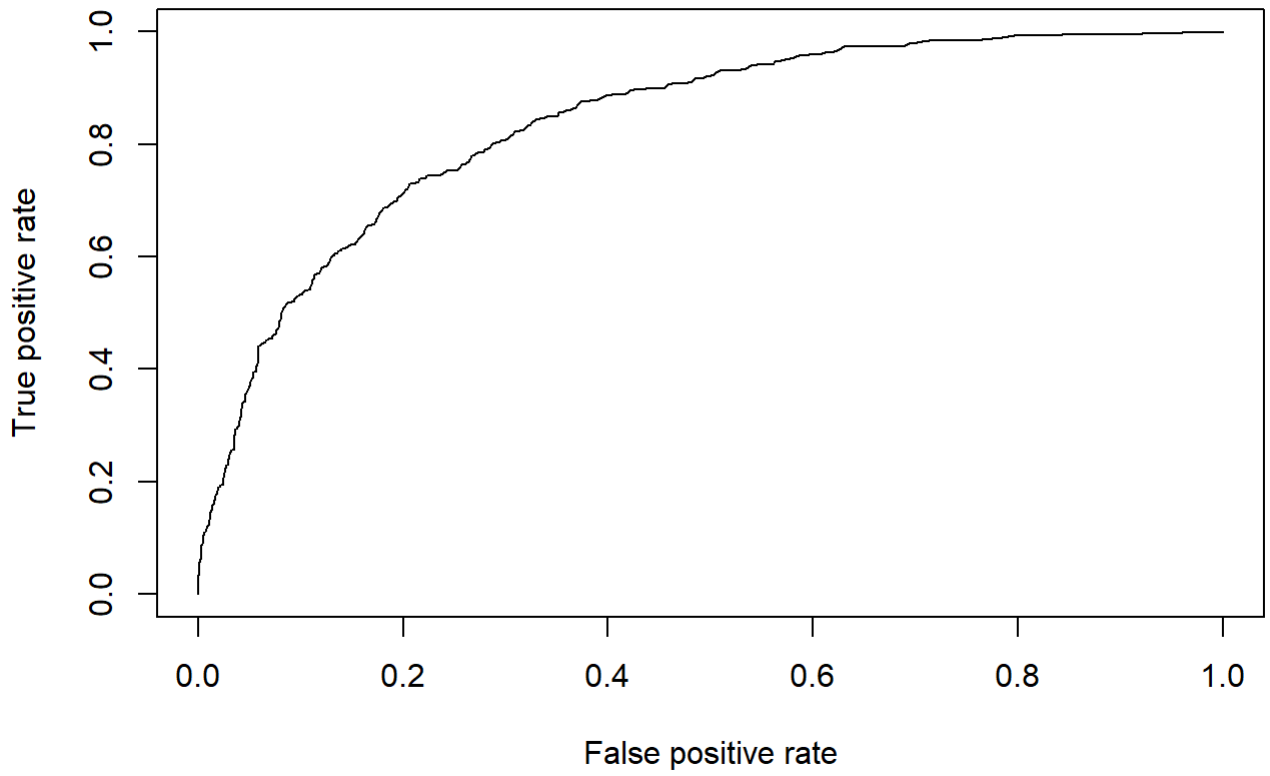
```
## Loading required package: gplots
```

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

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

```
# need to create prediction object from ROCR
pr <- prediction(churn.predict.prob[,2], data_test$Churn)

# plotting ROC curve
prf <- performance(pr, measure = "tpr", x.measure = "fpr")
plot(prf)
```




```
auc <- performance(pr, measure = "auc")
auc <- auc@y.values[[1]]
auc
```

```
## [1] 0.8381981
```

```
library(party)
```

```
## Loading required package: grid
```

```
## Loading required package: mvtnorm
```

```
## Loading required package: modeltools
```

```
## Loading required package: stats4
```

```
## Loading required package: strucchange
```

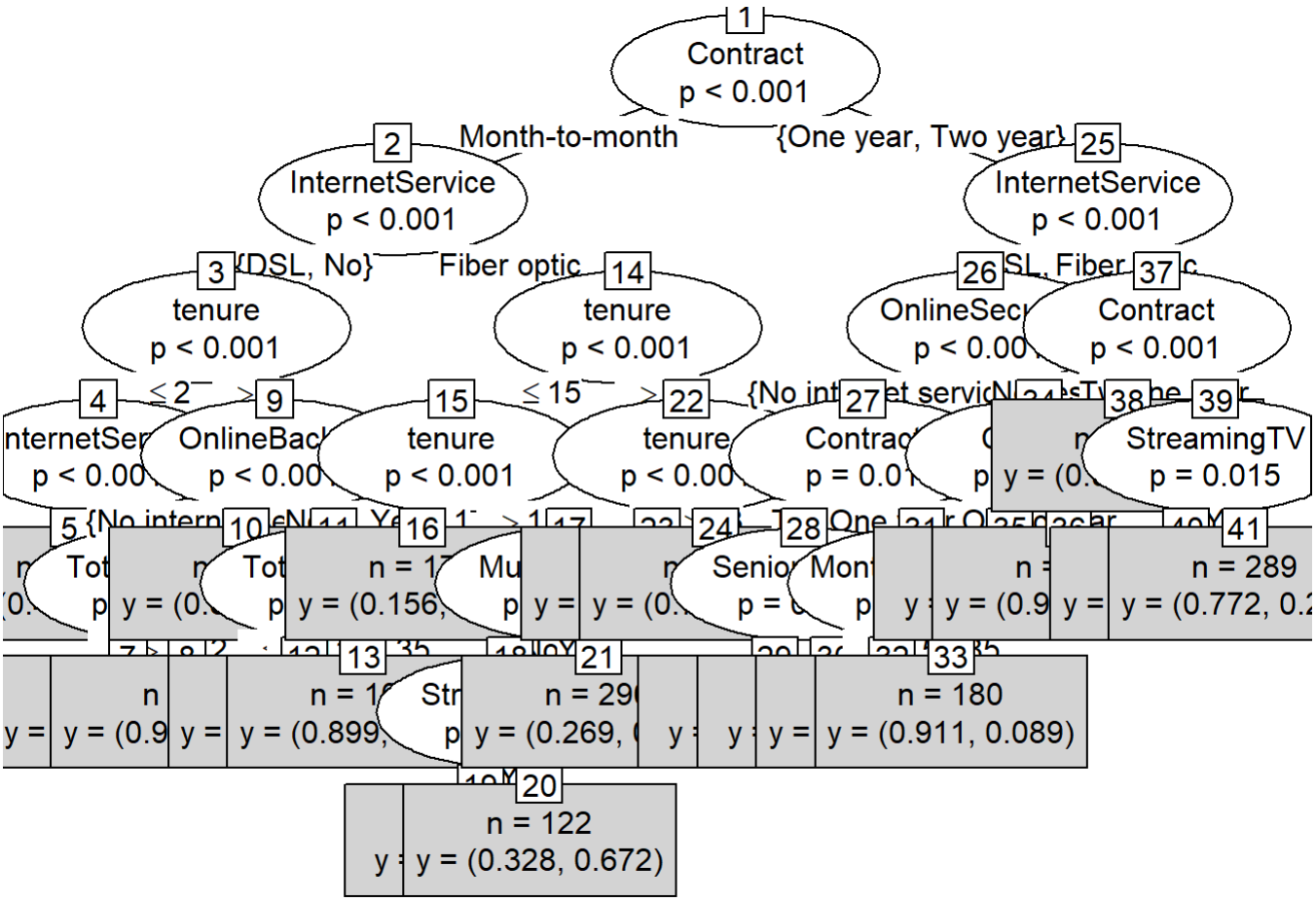
```
## Loading required package: zoo
```

```
##
## Attaching package: 'zoo'
```

```
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

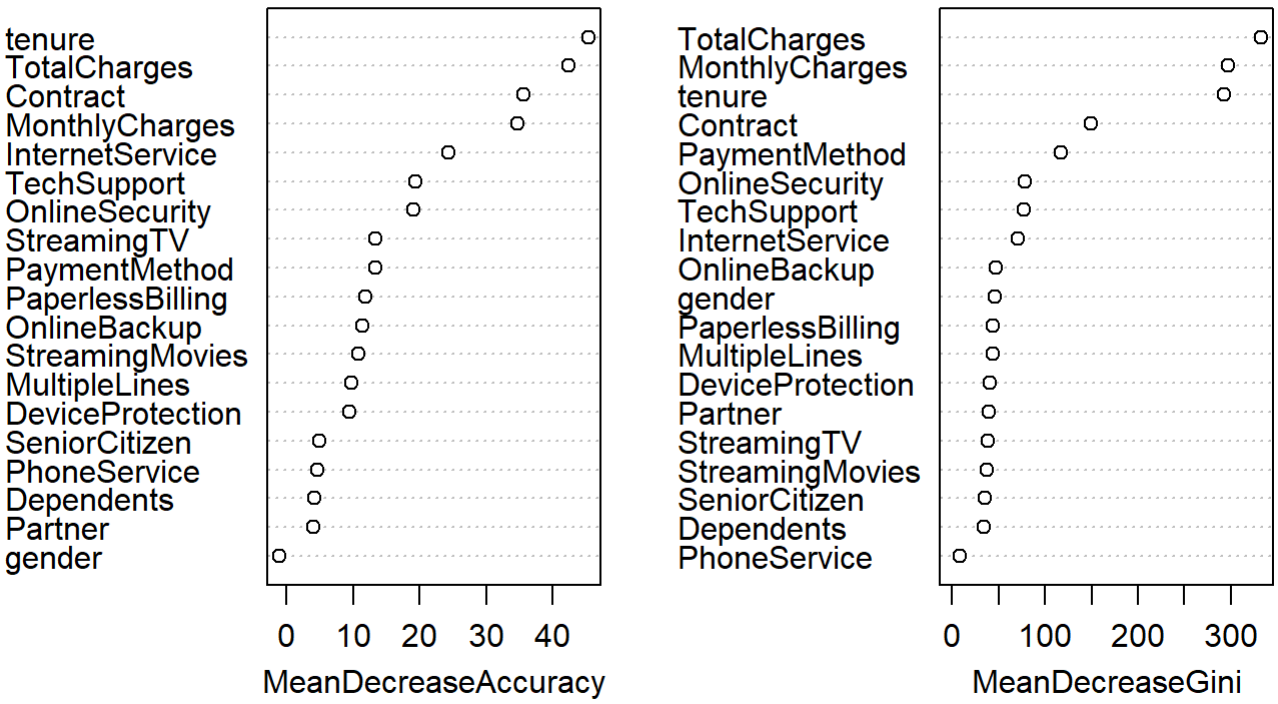
```
## Loading required package: sandwich
```

```
tree <- ctree(Churn~., data_train)
plot(tree, type='simple')
```



Tree based model: Random Forest

```
varImpPlot(churn.rf)
```




```
# changing the number of variables to try at each split
# mtry = 8, 12, 16, 20

# fitting the model
churn.rf = randomForest(Churn~tenure + PaperlessBilling + TotalCharges, data = data_train, mtry
 = 20, importance = T)
```

```
## Warning in randomForest.default(m, y, ...): invalid mtry: reset to within
## valid range
```

```
churn.predict.prob <- predict(churn.rf, data_test, type="prob")

churn.predict <- predict(churn.rf, data_test)
confusionMatrix(churn.predict, data_test$Churn, positive = "Yes")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   No  Yes
##        No  1117  267
##        Yes  176  200
##
##                Accuracy : 0.7483
##                  95% CI : (0.7273, 0.7684)
##     No Information Rate : 0.7347
##     P-Value [Acc > NIR] : 0.1018
##
##                   Kappa : 0.3115
##
##  Mcnemar's Test P-Value : 1.903e-05
##
##             Sensitivity : 0.4283
##             Specificity : 0.8639
##          Pos Pred Value : 0.5319
##          Neg Pred Value : 0.8071
##              Prevalence : 0.2653
##          Detection Rate : 0.1136
##    Detection Prevalence : 0.2136
##       Balanced Accuracy : 0.6461
##
##        'Positive' Class : Yes
##
```

The sensitivity and Positive Prediction Value is very good in this case.

```
pr <- prediction(churn.predict.prob[,2], data_test$Churn)

# AUC value
auc <- performance(pr, measure = "auc")
auc <- auc@y.values[[1]]
auc
```

```
## [1] 0.7548461
```

```
# changing the number of trees
# ntree = 25, 250, 500, 750

# fitting the model
churn.rf = randomForest(Churn~., data = data_train, ntree = 750, importance = T)

churn.predict.prob <- predict(churn.rf, data_test, type="prob")

churn.predict <- predict(churn.rf, data_test)
confusionMatrix(churn.predict, data_test$Churn, positive = "Yes")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   No  Yes
##        No  1175  229
##        Yes  118  238
##
##                Accuracy : 0.8028
##                  95% CI : (0.7835, 0.8212)
##     No Information Rate : 0.7347
##     P-Value [Acc > NIR] : 1.453e-11
##
##                   Kappa : 0.4527
##
##  Mcnemar's Test P-Value : 3.524e-09
##
##             Sensitivity : 0.5096
##             Specificity : 0.9087
##          Pos Pred Value : 0.6685
##          Neg Pred Value : 0.8369
##              Prevalence : 0.2653
##          Detection Rate : 0.1352
##    Detection Prevalence : 0.2023
##       Balanced Accuracy : 0.7092
##
##        'Positive' Class : Yes
##
```

```
# need to create prediction object from ROCR
pr <- prediction(churn.predict.prob[,2], data_test$Churn)

# AUC value
auc <- performance(pr, measure = "auc")
auc <- auc@y.values[[1]]
auc
```

```
## [1] 0.8392837
```

```
#k-fold cross val in caret
library(caret)
set.seed(10)

# train control
fitControl <- trainControl(## 10-fold CV
                           method = "repeatedcv",
                           number = 10,
                           ## repeated 3 times
                           repeats = 3,
                           classProbs = TRUE,
                           summaryFunction = twoClassSummary)
df <- data
# generalized linear model
logreg <- train(Churn ~ ., data = df, method = "glm", family = "binomial",      trControl = fitC
ontrol,metric = "ROC")
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
print(logreg)
```

```
## Generalized Linear Model
##
## 7043 samples
##   19 predictor
##    2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 6339, 6338, 6338, 6338, 6339, 6339, ...
## Resampling results:
##
##   ROC        Sens       Spec
##   0.8454452  0.8966004  0.550207
```

```
library(h2o)
```

```
##
## ----------------------------------------------------------------------
##
## Your next step is to start H2O:
##     > h2o.init()
##
## For H2O package documentation, ask for help:
##     > ??h2o
##
## After starting H2O, you can use the Web UI at http://localhost:54321
## For more information visit http://docs.h2o.ai
##
## ----------------------------------------------------------------------
```

```
##
## Attaching package: 'h2o'
```

```
## The following objects are masked from 'package:stats':
##
##     cor, sd, var
```

```
## The following objects are masked from 'package:base':
##
##     %*%, %in%, &&, ||, apply, as.factor, as.numeric, colnames,
##     colnames<-, ifelse, is.character, is.factor, is.numeric, log,
##     log10, log1p, log2, round, signif, trunc
```

```
h2o.no_progress()
h2o.init()
```

```
##
## H2O is not running yet, starting it now...
##
## Note:  In case of errors look at the following log files:
##     C:\Users\esaminl\AppData\Local\Temp\Rtmpm4rMLB/h2o_esaminl_started_from_r.out
##     C:\Users\esaminl\AppData\Local\Temp\Rtmpm4rMLB/h2o_esaminl_started_from_r.err
##
##
## Starting H2O JVM and connecting: . Connection successful!
##
## R is connected to the H2O cluster:
##     H2O cluster uptime:         4 seconds 137 milliseconds
##     H2O cluster timezone:       America/Chicago
##     H2O data parsing timezone:  UTC
##     H2O cluster version:        3.24.0.5
##     H2O cluster version age:    1 month and 20 days
##     H2O cluster name:           H2O_started_from_R_esaminl_jee669
##     H2O cluster total nodes:    1
##     H2O cluster total memory:   7.08 GB
##     H2O cluster total cores:    8
##     H2O cluster allowed cores:  8
##     H2O cluster healthy:        TRUE
##     H2O Connection ip:          localhost
##     H2O Connection port:        54321
##     H2O Connection proxy:       NA
##     H2O Internal Security:      FALSE
##     H2O API Extensions:         Amazon S3, Algos, AutoML, Core V3, Core V4
##     R Version:                  R version 3.6.1 (2019-07-05)
```

```
# create feature names
y <- "Churn"
x <- setdiff(names(data_train), y)

# h2o cannot ingest ordered factors
train.h2o <- data_train %>%
  mutate_if(is.factor, factor, ordered = FALSE) %>%
  as.h2o()

# train model
nb.h2o <- h2o.naiveBayes(
  x = x,
  y = y,
  training_frame = train.h2o,
  nfolds = 10,
  laplace = 0
)

# assess results
h2o.confusionMatrix(nb.h2o)
```

```
## Confusion Matrix (vertical: actual; across: predicted)  for max f1 @ threshold = 0.7700698795
59401:
##          No  Yes    Error        Rate
## No     2981  900 0.231899   =900/3881
## Yes     355 1047 0.253210   =355/1402
## Totals 3336 1947 0.237554  =1255/5283
```

```
# do a little preprocessing
preprocess <- preProcess(data_train, method = c("BoxCox", "center", "scale", "pca"))
train_pp   <- predict(preprocess, data_train)
test_pp    <- predict(preprocess, data_test)

# convert to h2o objects
train_pp.h2o <- train_pp %>%
  mutate_if(is.factor, factor, ordered = FALSE) %>%
  as.h2o()

test_pp.h2o <- test_pp %>%
  mutate_if(is.factor, factor, ordered = FALSE) %>%
  as.h2o()

# get new feature names --> PCA preprocessing reduced and changed some features
x <- setdiff(names(train_pp), "Churn")

# create tuning grid
hyper_params <- list(
  laplace = seq(0, 5, by = 0.5)
  )

# build grid search
grid <- h2o.grid(
  algorithm = "naivebayes",
  grid_id = "nb_grid",
  x = x,
  y = y,
  training_frame = train_pp.h2o,
  nfolds = 10,
  hyper_params = hyper_params
  )

# Sort the grid models by mse
sorted_grid <- h2o.getGrid("nb_grid", sort_by = "accuracy", decreasing = TRUE)
sorted_grid
```

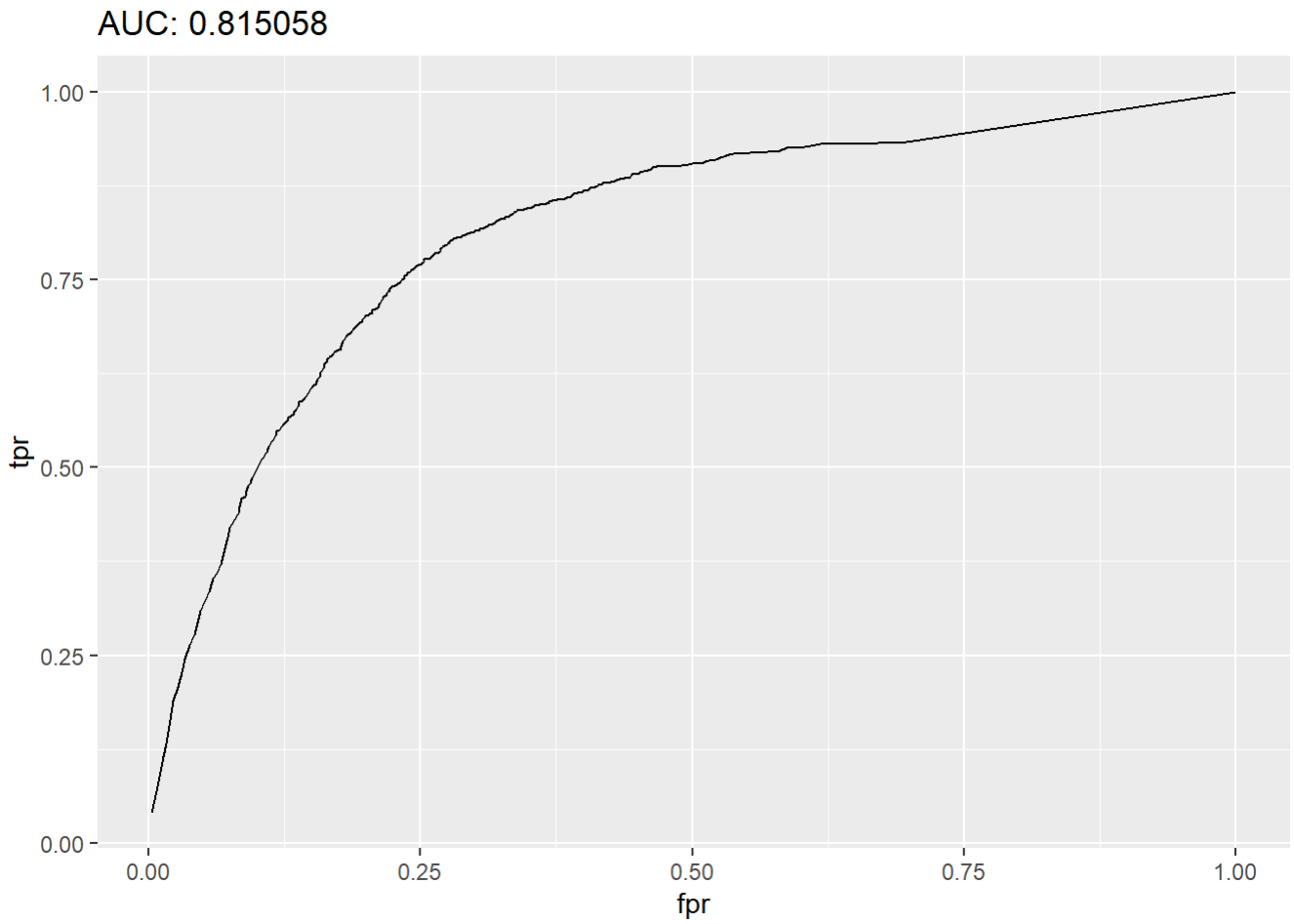
```
## H2O Grid Details
## ================
##
## Grid ID: nb_grid
## Used hyper parameters:
##   -  laplace
## Number of models: 11
## Number of failed models: 0
##
## Hyper-Parameter Search Summary: ordered by decreasing accuracy
##    laplace        model_ids           accuracy
## 1      2.5  nb_grid_model_6 0.7675563127011168
## 2      4.5 nb_grid_model_10 0.7664205943592656
## 3      3.5  nb_grid_model_8 0.7664205943592656
## 4      3.0  nb_grid_model_7  0.766231307968957
## 5      1.0  nb_grid_model_3 0.7624455801627863
## 6      1.5  nb_grid_model_4 0.7597955706984668
## 7      4.0  nb_grid_model_9 0.7597955706984668
## 8      0.0  nb_grid_model_1 0.7590384251372326
## 9      2.0  nb_grid_model_5 0.7586598523566156
## 10     5.0 nb_grid_model_11 0.7577134204050728
## 11     0.5  nb_grid_model_2  0.755631270111679
```

```
# grab top model id
best_h2o_model <- sorted_grid@model_ids[[1]]
best_model <- h2o.getModel(best_h2o_model)

# confusion matrix of best model
h2o.confusionMatrix(best_model)
```

```
## Confusion Matrix (vertical: actual; across: predicted)  for max f1 @ threshold = 0.7418560804
2583:
##          No  Yes    Error        Rate
## No     2948  933 0.240402   =933/3881
## Yes     333 1069 0.237518   =333/1402
## Totals 3281 2002 0.239637  =1266/5283
```

```
# ROC curve
auc <- h2o.auc(best_model, xval = TRUE)
fpr <- h2o.performance(best_model, xval = TRUE) %>% h2o.fpr() %>% .[['fpr']]
tpr <- h2o.performance(best_model, xval = TRUE) %>% h2o.tpr() %>% .[['tpr']]
data.frame(fpr = fpr, tpr = tpr) %>%
  ggplot(aes(fpr, tpr) ) +
  geom_line() +
  ggtitle( sprintf('AUC: %f', auc) )
```




```
# evaluate on test set
h2o.performance(best_model, newdata = test_pp.h2o)
```

```
## H2OBinomialMetrics: naivebayes
##
## MSE:  0.2203908
## RMSE:  0.469458
## LogLoss:  0.9291591
## Mean Per-Class Error:  0.232946
## AUC:  0.8329533
## pr_auc:  0.6167042
## Gini:  0.6659065
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
##          No Yes    Error       Rate
## No     1045 248 0.191802  =248/1293
## Yes     128 339 0.274090   =128/467
## Totals 1173 587 0.213636  =376/1760
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##                         metric threshold    value idx
## 1                       max f1  0.848224 0.643264 109
## 2                       max f2  0.100153 0.743743 333
## 3                 max f0point5  0.954922 0.636611  48
## 4                 max accuracy  0.954922 0.806818  48
## 5                max precision  0.998991 0.896552   1
## 6                   max recall  0.000010 1.000000 399
## 7              max specificity  0.999392 0.997680   0
## 8             max absolute_mcc  0.887248 0.500910  91
## 9   max min_per_class_accuracy  0.755804 0.764454 145
## 10 max mean_per_class_accuracy  0.804953 0.769164 126
##
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, va
lid=<T/F>, xval=<T/F>)`
```

```
## H2OBinomialMetrics: naivebayes

# predict new data
h2o.predict(nb.h2o, newdata = test_pp.h2o)
```

```
## Warning in doTryCatch(return(expr), name, parentenv, handler): Test/
## Validation dataset is missing column 'tenure': substituting in a column of
## NaN
```

```
## Warning in doTryCatch(return(expr), name, parentenv, handler): Test/
## Validation dataset is missing column 'MonthlyCharges': substituting in a
## column of NaN
```

```
## Warning in doTryCatch(return(expr), name, parentenv, handler): Test/
## Validation dataset is missing column 'TotalCharges': substituting in a
## column of NaN
```

```
##   predict          No         Yes
## 1      No 0.904246335 0.095753665
## 2      No 0.979467909 0.020532091
## 3     Yes 0.006409447 0.993590553
## 4     Yes 0.015556176 0.984443824
## 5     Yes 0.110465703 0.889534297
## 6      No 0.998859597 0.001140403
##
## [1760 rows x 3 columns]
```

```
# shut down h2o
h2o.shutdown(prompt = FALSE)
```

```
## [1] TRUE
```

```
## [1] TRUE
```

```
library(e1071)
NBclassfier=naiveBayes(Churn~., data=data_train)
print(NBclassfier)
```

```
##
## Naive Bayes Classifier for Discrete Predictors
##
## Call:
## naiveBayes.default(x = X, y = Y, laplace = laplace)
##
## A-priori probabilities:
## Y
##        No       Yes
## 0.7346205 0.2653795
##
## Conditional probabilities:
##      gender
## Y        Female      Male
##   No  0.4893069 0.5106931
##   Yes 0.5007133 0.4992867
##
##      SeniorCitizen
## Y             0         1
##   No  0.8680752 0.1319248
##   Yes 0.7603424 0.2396576
##
##      Partner
## Y            No       Yes
##   No  0.4741046 0.5258954
##   Yes 0.6476462 0.3523538
##
##      Dependents
## Y            No       Yes
##   No  0.6580778 0.3419222
##   Yes 0.8109843 0.1890157
##
##      tenure
## Y         [,1]     [,2]
##   No  37.73460 24.18417
##   Yes 18.02211 19.55395
##
##      PhoneService
## Y             No        Yes
##   No  0.09610925 0.90389075
##   Yes 0.08915835 0.91084165
##
##      MultipleLines
## Y             No No phone service        Yes
##   No  0.48698789       0.09610925 0.41690286
##   Yes 0.45435093       0.08915835 0.45649073
##
##      InternetService
## Y            DSL Fiber optic         No
##   No  0.37876836  0.34862149 0.27261015
##   Yes 0.24108417  0.69400856 0.06490728
##
##      OnlineSecurity
```

```
## Y             No No internet service        Yes
##   No  0.39345530          0.27261015 0.33393455
##   Yes 0.77888730          0.06490728 0.15620542
##
##      OnlineBackup
## Y             No No internet service        Yes
##   No  0.35635146          0.27261015 0.37103839
##   Yes 0.64835949          0.06490728 0.28673324
##
##      DeviceProtection
## Y             No No internet service        Yes
##   No  0.36227776          0.27261015 0.36511208
##   Yes 0.63694722          0.06490728 0.29814551
##
##      TechSupport
## Y             No No internet service        Yes
##   No  0.39680495          0.27261015 0.33058490
##   Yes 0.76890157          0.06490728 0.16619116
##
##      StreamingTV
## Y             No No internet service        Yes
##   No  0.36021644          0.27261015 0.36717341
##   Yes 0.49429387          0.06490728 0.44079886
##
##      StreamingMovies
## Y             No No internet service        Yes
##   No  0.35403247          0.27261015 0.37335738
##   Yes 0.49144080          0.06490728 0.44365193
##
##      Contract
## Y     Month-to-month   One year   Two year
##   No      0.43004380 0.24838959 0.32156661
##   Yes     0.88801712 0.08630528 0.02567760
##
##      PaperlessBilling
## Y            No       Yes
##   No  0.4607060 0.5392940
##   Yes 0.2524964 0.7475036
##
##      PaymentMethod
## Y     Bank transfer (automatic) Credit card (automatic) Electronic check
##   No                  0.2447823              0.2576656        0.2483896
##   Yes                 0.1419401              0.1205421        0.5641940
##      PaymentMethod
## Y     Mailed check
##   No     0.2491626
##   Yes    0.1733238
##
##      MonthlyCharges
## Y         [,1]     [,2]
##   No  61.43759 31.09108
##   Yes 74.52967 24.70370
##
##      TotalCharges
```

```
## Y         [,1]     [,2]
##   No  2570.309 2327.716
##   Yes 1533.169 1885.131
```

Naive Bayes Classifier

```
library(e1071)
NBclassfier=naiveBayes(Churn~., data=data_train)
print(NBclassfier)
```

```
##
## Naive Bayes Classifier for Discrete Predictors
##
## Call:
## naiveBayes.default(x = X, y = Y, laplace = laplace)
##
## A-priori probabilities:
## Y
##        No       Yes
## 0.7346205 0.2653795
##
## Conditional probabilities:
##      gender
## Y        Female      Male
##   No  0.4893069 0.5106931
##   Yes 0.5007133 0.4992867
##
##      SeniorCitizen
## Y             0         1
##   No  0.8680752 0.1319248
##   Yes 0.7603424 0.2396576
##
##      Partner
## Y            No       Yes
##   No  0.4741046 0.5258954
##   Yes 0.6476462 0.3523538
##
##      Dependents
## Y            No       Yes
##   No  0.6580778 0.3419222
##   Yes 0.8109843 0.1890157
##
##      tenure
## Y         [,1]     [,2]
##   No  37.73460 24.18417
##   Yes 18.02211 19.55395
##
##      PhoneService
## Y             No        Yes
##   No  0.09610925 0.90389075
##   Yes 0.08915835 0.91084165
##
##      MultipleLines
## Y             No No phone service        Yes
##   No  0.48698789       0.09610925 0.41690286
##   Yes 0.45435093       0.08915835 0.45649073
##
##      InternetService
## Y            DSL Fiber optic         No
##   No  0.37876836  0.34862149 0.27261015
##   Yes 0.24108417  0.69400856 0.06490728
##
##      OnlineSecurity
```

```
## Y             No No internet service        Yes
##   No  0.39345530          0.27261015 0.33393455
##   Yes 0.77888730          0.06490728 0.15620542
##
##      OnlineBackup
## Y             No No internet service        Yes
##   No  0.35635146          0.27261015 0.37103839
##   Yes 0.64835949          0.06490728 0.28673324
##
##      DeviceProtection
## Y             No No internet service        Yes
##   No  0.36227776          0.27261015 0.36511208
##   Yes 0.63694722          0.06490728 0.29814551
##
##      TechSupport
## Y             No No internet service        Yes
##   No  0.39680495          0.27261015 0.33058490
##   Yes 0.76890157          0.06490728 0.16619116
##
##      StreamingTV
## Y             No No internet service        Yes
##   No  0.36021644          0.27261015 0.36717341
##   Yes 0.49429387          0.06490728 0.44079886
##
##      StreamingMovies
## Y             No No internet service        Yes
##   No  0.35403247          0.27261015 0.37335738
##   Yes 0.49144080          0.06490728 0.44365193
##
##      Contract
## Y     Month-to-month   One year   Two year
##   No      0.43004380 0.24838959 0.32156661
##   Yes     0.88801712 0.08630528 0.02567760
##
##      PaperlessBilling
## Y            No       Yes
##   No  0.4607060 0.5392940
##   Yes 0.2524964 0.7475036
##
##      PaymentMethod
## Y     Bank transfer (automatic) Credit card (automatic) Electronic check
##   No                  0.2447823              0.2576656        0.2483896
##   Yes                 0.1419401              0.1205421        0.5641940
##      PaymentMethod
## Y     Mailed check
##   No     0.2491626
##   Yes    0.1733238
##
##      MonthlyCharges
## Y         [,1]     [,2]
##   No  61.43759 31.09108
##   Yes 74.52967 24.70370
##
##      TotalCharges
```

```
## Y         [,1]     [,2]
##   No  2570.309 2327.716
##   Yes 1533.169 1885.131
```

```
printALL=function(model){
  trainPred=predict(model, newdata = data_train, type = "class")
  trainTable=table(data_train$Churn, trainPred)
  testPred=predict(NBclassfier, newdata=data_test, type="class")
  testTable=table(data_test$Churn, testPred)
  trainAcc=(trainTable[1,1]+trainTable[2,2])/sum(trainTable)
  testAcc=(testTable[1,1]+testTable[2,2])/sum(testTable)
  message("Contingency Table for Training Data")
  print(trainTable)
  message("Contingency Table for Test Data")
  print(testTable)
  message("Accuracy")
  print(round(cbind(trainAccuracy=trainAcc, testAccuracy=testAcc),3))
}
printALL(NBclassfier)
```

```
## Contingency Table for Training Data
```

```
##      trainPred
##         No  Yes
##   No  2715 1166
##   Yes  286 1116
```

```
## Contingency Table for Test Data
```

```
##      testPred
##        No Yes
##   No  902 391
##   Yes  84 383
```

```
## Accuracy
```

```
##      trainAccuracy testAccuracy
## [1,]         0.725         0.73
```

Naive Bayes Classifier provides us with good train and test accuracy rates with three important parameters that we derived from the varImportance plot.

```
library(naivebayes)
```

```
## naivebayes 0.9.6 loaded
```

```
newNBclassifier=naive_bayes(Churn~.,usekernel=T, data=data_train)
printALL(newNBclassifier)
```

```
## Warning: predict.naive_bayes(): More features in the newdata are provided
## as there are probability tables in the object. Calculation is performed
## based on features to be found in the tables.
```

```
## Contingency Table for Training Data
```

```
##      trainPred
##         No  Yes
##   No  2732 1149
##   Yes  277 1125
```

```
## Contingency Table for Test Data
```

```
##      testPred
##        No Yes
##   No  902 391
##   Yes  84 383
```

```
## Accuracy
```

```
##      trainAccuracy testAccuracy
## [1,]          0.73         0.73
```

Naive Bayes Assumptions Testing

With naÃ¯ve Bayes, we assume that the predictor variables are conditionally independent of one another given the response value. This is an extremely strong assumption. We can see quickly that our churn data does not violate this as we have do not have strongly correlated variables.

```
library(dplyr)
tenure_cat <- with(data, ifelse(tenure <= 35, "Low", "High"))
data <- data.frame(data, tenure_cat)

data %>%
  select(tenure_cat, Churn, TotalCharges, tenure) %>%
  filter(tenure_cat == "Low", Churn == "Yes") %>%
  summarise(n = n(),
            total = sum(TotalCharges),
            avg_tenure = sum(tenure)/n)
```

```
##      n   total avg_tenure
## 1 1501 1135107   9.626915
```

```
library(miscset)
```

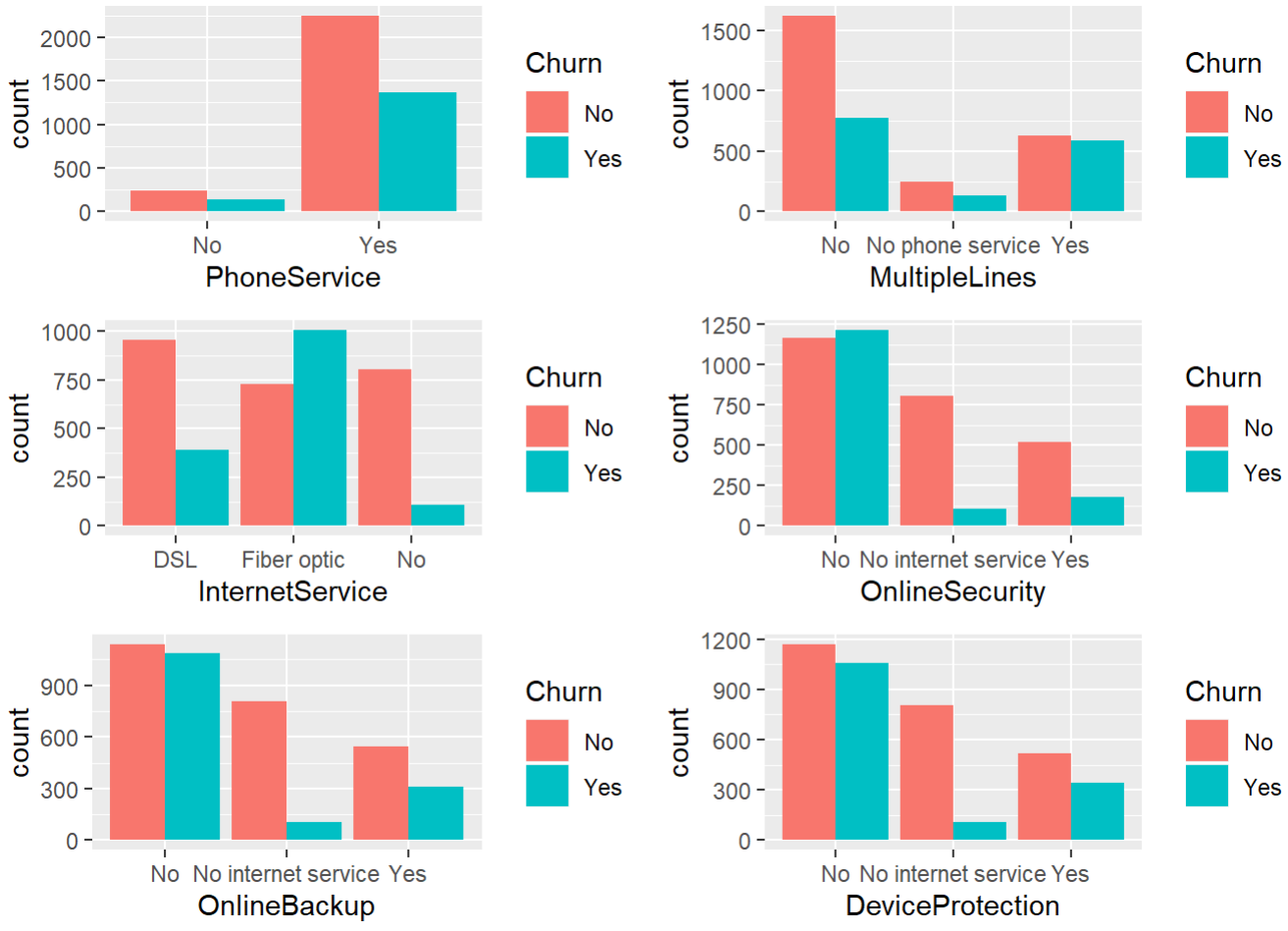
```
##
## Attaching package: 'miscset'
```

```
## The following object is masked from 'package:modeltools':
##
##     info
```

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

```
library(ggplot2)
lowTenure <- data %>% filter(tenure_cat == "Low")

ggplotGrid(ncol=2,
lapply(c("PhoneService","MultipleLines","InternetService","OnlineSecurity","OnlineBackup",
         "DeviceProtection"),
       function(col){
         ggplot(lowTenure,aes_string(col)) + geom_bar(aes(fill=Churn),position="dodge")
       }))
```



```
ggplot(lowTenure) +
  geom_bar(aes(x=PaymentMethod,fill=Churn), position = "dodge")
```