Random Forest & Logistic Regression

Chinki

May 20, 2017

Step 2: Exploring & Preparing the data

#reading the data  
credit <- read.csv("http://www.sci.csueastbay.edu/~esuess/classes/Statistics\_6620/Presentations/ml10/credit.csv")  
library(caret)

## Loading required package: lattice

## Loading required package: ggplot2

library(lattice)  
library(ggplot2)

Step 3: Traning a model on the data

I will use random forest package for random forest.

library(randomForest)

## randomForest 4.6-12

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

##   
## Attaching package: 'randomForest'

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

#Set.seed for getting the same output  
set.seed(300)

Step 4: Evaluating the model performance

#Creating random forest  
rf <- randomForest(default ~ ., data = credit)  
rf

##   
## Call:  
## randomForest(formula = default ~ ., data = credit)   
## Type of random forest: classification  
## Number of trees: 500  
## No. of variables tried at each split: 4  
##   
## OOB estimate of error rate: 23.8%  
## Confusion matrix:  
## no yes class.error  
## no 640 60 0.08571429  
## yes 178 122 0.59333333

The randomForest() function creates an ensemble of 500 trees & 4 variable at each split.Estimated error rate is 23.8%.

step 4: Improving model performance

library(caret)  
ctrl <- trainControl(method = "repeatedcv",  
 number = 10, repeats = 10)

In this example, we are repeating 10 cross fold 10 times repeate.

# tunning a random forest  
grid\_rf <- expand.grid(.mtry = c(2, 4, 8, 16))

we need to create a grid with values of 2, 4, 8, and 16.

set.seed(300)  
m\_rf <- train(default ~ ., data = credit, method = "rf",  
 metric = "Kappa", trControl = ctrl,  
 tuneGrid = grid\_rf)  
m\_rf

## Random Forest   
##   
## 1000 samples  
## 16 predictor  
## 2 classes: 'no', 'yes'   
##   
## No pre-processing  
## Resampling: Cross-Validated (10 fold, repeated 10 times)   
## Summary of sample sizes: 900, 900, 900, 900, 900, 900, ...   
## Resampling results across tuning parameters:  
##   
## mtry Accuracy Kappa   
## 2 0.7256 0.1311283  
## 4 0.7476 0.2878470  
## 8 0.7519 0.3346061  
## 16 0.7557 0.3618152  
##   
## Kappa was used to select the optimal model using the largest value.  
## The final value used for the model was mtry = 16.

I am going to train model based on rf & using kappa for accuracy checking. we'll compare that to a boosted tree using 10, 20, 30, and 40 iterations:

# auto-tune a boosted C5.0 decision tree  
grid\_c50 <- expand.grid(.model = "tree",  
 .trials = c(10, 20, 30, 40),  
 .winnow = "FALSE")

set.seed(300)  
library(C50)  
m\_c50 <- train(default ~ ., data = credit, method = "C5.0",  
 metric = "Kappa", trControl = ctrl,  
 tuneGrid = grid\_c50)

## Loading required package: plyr

## Warning in Ops.factor(x$winnow): '!' not meaningful for factors

m\_c50

## C5.0   
##   
## 1000 samples  
## 16 predictor  
## 2 classes: 'no', 'yes'   
##   
## No pre-processing  
## Resampling: Cross-Validated (10 fold, repeated 10 times)   
## Summary of sample sizes: 900, 900, 900, 900, 900, 900, ...   
## Resampling results across tuning parameters:  
##   
## trials Accuracy Kappa   
## 10 0.7325 0.3215655  
## 20 0.7343 0.3268052  
## 30 0.7381 0.3343137  
## 40 0.7388 0.3335082  
##   
## Tuning parameter 'model' was held constant at a value of tree  
##   
## Tuning parameter 'winnow' was held constant at a value of FALSE  
## Kappa was used to select the optimal model using the largest value.  
## The final values used for the model were trials = 30, model = tree  
## and winnow = FALSE.

With a kappa of about 0.361, the random forest model with mtry = 16 was the winner among these eight models. It was higher than the best C5.0 decision tree, which had a kappa of about 0.334, and slightly higher than the AdaBoost.M1 model with a kappa of about 0.360. Based on these results, we would submit the random forest as our final model.