Assign-4 Ensemble Learning

Jonathan De Los Santos

2/14/2021

# Prompt

The example data in 4.4-UniversalBank.csv contains 5,000 lines of bank customer data (“Universal Bank”).

The bank is interested in growing customers to bring in more loan business. The bank encouraged the marketing department to come up with an idea for better target marketing and to determine what factors make a customer accept a personal loan. The attached dataset includes demographic variables of its customers, “customer response to the personal loan campaign (Personal Loan), etc.

Build a decision tree and develop ensemble models including bagging, boosting, and random forests, and compare the outcomes.

# Import and Prepare Data

bank <- read.csv("Data Sets/4.4-UniversalBank.csv")  
#str(bank)  
  
train\_sample <- sample(5000,4000)  
train <- bank[train\_sample,]  
test <- bank[-train\_sample,]  
  
str(train)

## 'data.frame': 4000 obs. of 9 variables:  
## $ Age : int 59 36 39 49 25 47 35 27 50 27 ...  
## $ Experience : int 35 11 15 24 0 21 5 0 25 1 ...  
## $ Income : int 38 44 89 75 30 82 203 38 42 134 ...  
## $ Family : int 1 2 2 1 2 3 1 4 2 1 ...  
## $ Education : int 1 2 1 2 2 1 3 3 2 2 ...  
## $ Mortgage : int 0 85 0 160 0 0 0 154 110 307 ...  
## $ Personal.Loan : int 0 0 0 0 0 0 1 0 0 1 ...  
## $ Securities.Account: int 0 0 0 0 0 0 0 0 0 0 ...  
## $ CreditCard : int 0 0 0 1 0 0 0 0 1 0 ...

# Decision Tree

To begin we build a decision tree with training data (80%) and evaluate with the test data.

## Model with C5.0 Algorithm

This model misclassifies roughly 1.6%. It identifies Income, then Education as the first two split variables.

#install.packages("C50")  
library(C50)

## Warning: package 'C50' was built under R version 4.0.2

model\_bank <- C5.0(train[-7], factor(train$Personal.Loan))  
model\_bank

##   
## Call:  
## C5.0.default(x = train[-7], y = factor(train$Personal.Loan))  
##   
## Classification Tree  
## Number of samples: 4000   
## Number of predictors: 8   
##   
## Tree size: 6   
##   
## Non-standard options: attempt to group attributes

summary(model\_bank)

##   
## Call:  
## C5.0.default(x = train[-7], y = factor(train$Personal.Loan))  
##   
##   
## C5.0 [Release 2.07 GPL Edition] Sun Feb 14 20:06:17 2021  
## -------------------------------  
##   
## Class specified by attribute `outcome'  
##   
## Read 4000 cases (9 attributes) from undefined.data  
##   
## Decision tree:  
##   
## Education > 1:  
## :...Income <= 115: 0 (2064/58)  
## : Income > 115: 1 (248)  
## Education <= 1:  
## :...Income <= 93: 0 (1063/7)  
## Income > 93:  
## :...Family <= 2: 0 (544/5)  
## Family > 2:  
## :...Income <= 113: 0 (28/7)  
## Income > 113: 1 (53)  
##   
##   
## Evaluation on training data (4000 cases):  
##   
## Decision Tree   
## ----------------   
## Size Errors   
##   
## 6 77( 1.9%) <<  
##   
##   
## (a) (b) <-classified as  
## ---- ----  
## 3622 (a): class 0  
## 77 301 (b): class 1  
##   
##   
## Attribute usage:  
##   
## 100.00% Income  
## 100.00% Education  
## 15.62% Family  
##   
##   
## Time: 0.0 secs

## Evaluating C5.0 Model

The model based on the 80% training data is evaluated against the 20% test data. We can see the comparisons of the data in both the table and crosstable below. This evaluates to a **97.7% success rate**, or a **2.3% error rate.**

predict\_bank <- predict(model\_bank, test)  
summary(predict\_bank)

## 0 1   
## 918 82

table(test$Personal.Loan)

##   
## 0 1   
## 898 102

library(gmodels)

## Warning: package 'gmodels' was built under R version 4.0.2

CrossTable(test$Personal.Loan, predict\_bank,   
 porp.chisq = FALSE,  
 dnn = c("Actual Loans", "Predicted Loans"))

##   
##   
## Cell Contents  
## |-------------------------|  
## | N |  
## | Chi-square contribution |  
## | N / Row Total |  
## | N / Col Total |  
## | N / Table Total |  
## |-------------------------|  
##   
##   
## Total Observations in Table: 1000   
##   
##   
## | Predicted Loans   
## Actual Loans | 0 | 1 | Row Total |   
## -------------|-----------|-----------|-----------|  
## 0 | 898 | 0 | 898 |   
## | 6.578 | 73.636 | |   
## | 1.000 | 0.000 | 0.898 |   
## | 0.978 | 0.000 | |   
## | 0.898 | 0.000 | |   
## -------------|-----------|-----------|-----------|  
## 1 | 20 | 82 | 102 |   
## | 57.908 | 648.286 | |   
## | 0.196 | 0.804 | 0.102 |   
## | 0.022 | 1.000 | |   
## | 0.020 | 0.082 | |   
## -------------|-----------|-----------|-----------|  
## Column Total | 918 | 82 | 1000 |   
## | 0.918 | 0.082 | |   
## -------------|-----------|-----------|-----------|  
##   
##

# Bagging Model

We will build on the single-tree analysis with a 5-tree bagging model.

library(adabag)

## Warning: package 'adabag' was built under R version 4.0.2

## Loading required package: rpart

## Warning: package 'rpart' was built under R version 4.0.2

## Loading required package: caret

## Warning: package 'caret' was built under R version 4.0.2

## Loading required package: lattice

## Loading required package: ggplot2

## Warning: package 'ggplot2' was built under R version 4.0.2

## Loading required package: foreach

## Warning: package 'foreach' was built under R version 4.0.2

## Loading required package: doParallel

## Warning: package 'doParallel' was built under R version 4.0.2

## Loading required package: iterators

## Warning: package 'iterators' was built under R version 4.0.2

## Loading required package: parallel

library(rpart)  
library(rpart.plot)

## Warning: package 'rpart.plot' was built under R version 4.0.2

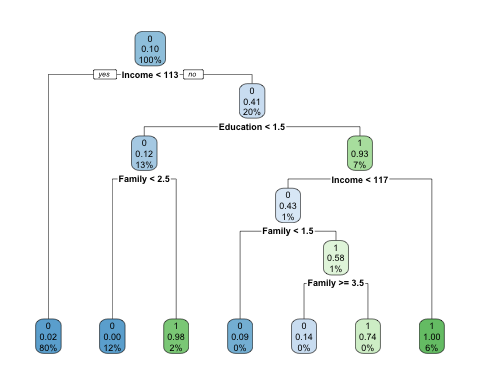
train$Personal.Loan <- factor(train$Personal.Loan)  
bank.bagging <- bagging(Personal.Loan~ ., data = train, mfinal = 5,  
 control = rpart.control(maxdepth=5, minsplit=1))  
  
# See variable importance  
bank.bagging$importance

## Age CreditCard Education Experience   
## 0.4250164 0.0000000 42.8475038 0.0000000   
## Family Income Mortgage Securities.Account   
## 17.1452987 39.5821811 0.0000000 0.0000000

## Plot Bagging Model

rpart.plot(bank.bagging$trees[[1]])

## Warning: Cannot retrieve the data used to build the model (so cannot determine roundint and is.binary for the variables).  
## To silence this warning:  
## Call rpart.plot with roundint=FALSE,  
## or rebuild the rpart model with model=TRUE.

 ## Evaluate Bagging Model

By creating a confusion matrix of the bagging model and the test data, we find that our model resulted in a **97.9% success rate**, or a **2.1% error rate.**

library(e1071)  
pred <- predict(bank.bagging, test, type = "class")  
confusionMatrix(factor(pred$class), factor(test$Personal.Loan))

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1  
## 0 897 20  
## 1 1 82  
##   
## Accuracy : 0.979   
## 95% CI : (0.9681, 0.987)  
## No Information Rate : 0.898   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.8751   
##   
## Mcnemar's Test P-Value : 8.568e-05   
##   
## Sensitivity : 0.9989   
## Specificity : 0.8039   
## Pos Pred Value : 0.9782   
## Neg Pred Value : 0.9880   
## Prevalence : 0.8980   
## Detection Rate : 0.8970   
## Detection Prevalence : 0.9170   
## Balanced Accuracy : 0.9014   
##   
## 'Positive' Class : 0   
##

# Boosting Model

We repeat this process with a boosting model which adjusts the weights of incorrect responses between iterations.

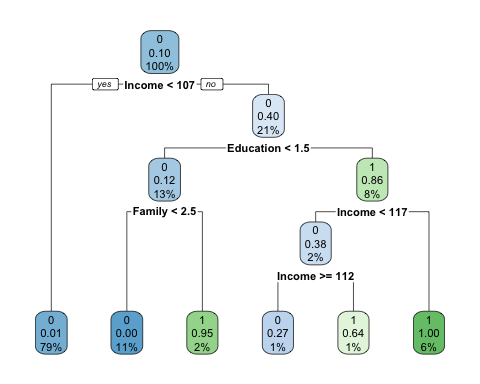
bank.boosting <- boosting(Personal.Loan~ ., data = train, mfinal = 5,  
 control = rpart.control(maxdepth=5, minsplit=1))  
  
# See variable importance  
bank.boosting$importance

## Age CreditCard Education Experience   
## 1.558889 0.000000 23.652418 4.137620   
## Family Income Mortgage Securities.Account   
## 11.601895 58.644009 0.000000 0.405168

## Plot Boosting Model

rpart.plot(bank.boosting$trees[[1]])

## Warning: Cannot retrieve the data used to build the model (so cannot determine roundint and is.binary for the variables).  
## To silence this warning:  
## Call rpart.plot with roundint=FALSE,  
## or rebuild the rpart model with model=TRUE.

 ## Evaluate Boosting Model

Like before, we create a confusion matrix of the boosting model and the test data. The boosting model resulted in a **97.9% success rate**, or a **2.1% error rate** which is the same as the bagging model. In this case, the incorrect predictions were weighted slightly more towards false negatives.

library(e1071)  
pred2 <- predict(bank.boosting, test, type = "class")  
confusionMatrix(factor(pred2$class), factor(test$Personal.Loan))

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1  
## 0 896 20  
## 1 2 82  
##   
## Accuracy : 0.978   
## 95% CI : (0.9669, 0.9862)  
## No Information Rate : 0.898   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.8697   
##   
## Mcnemar's Test P-Value : 0.0002896   
##   
## Sensitivity : 0.9978   
## Specificity : 0.8039   
## Pos Pred Value : 0.9782   
## Neg Pred Value : 0.9762   
## Prevalence : 0.8980   
## Detection Rate : 0.8960   
## Detection Prevalence : 0.9160   
## Balanced Accuracy : 0.9008   
##   
## 'Positive' Class : 0   
##

# Random Forest

The final model we will perform is a random forest, which extends our bagging model above. The result is given in the confusion matrix below.

library(randomForest)

## Warning: package 'randomForest' was built under R version 4.0.2

## 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

# Build model  
myForest <- randomForest (Personal.Loan ~., nodesize = 2, mtry = 2, ntree = 10, data = train)  
  
# Observe confusion table   
myForest$confusion

## 0 1 class.error  
## 0 3555 25 0.00698324  
## 1 118 257 0.31466667

## Evaluate Random Forest

We evaluate random forest similar to before, with the minor difference of manually calculating the success and error rates. The random forest resulted in a **97.9% success rate**, or a **2.1% error rate** which is the same as the bagging and boosting models.

# Success Rate  
sum(tt[row(tt) == col(tt)]) / sum(tt)

## [1] 0.974

# Error Rate (1 - success rate)  
1 - sum(tt[row(tt) == col(tt)]) / sum(tt)

## [1] 0.026

# Conclusion

Unexpectedly, all of the ensemble models ended with the same error rate of 2.1% while the single decision tree yielded 2.3%. Assuming there are no logical errors, this may be because the relationships between the data are highly predictable and therefore there is little variation between the different models. Given this information, we would recommend that the lease resource intensive of the bagging and boosting models be used to build the decision tree for Universal Bank.