Ensemble Techniques

Justin Hardy & Benjamin Frenkel

The Data Set

Starting by reading in the data set. The data set we'll use for the assignment consists of data collected by an airline organization, over their customers' submitted satisfaction surveys, as well as relevant information about their flight and demographic.

If you want to see the data set for yourself, you access the raw data here, or the page where I collected it online.

Please note that this part of the assignment reuses the data set used in the Classification portion of the assignment.

```
# Read data set
CustomerData_raw <- read.csv("Invistico_Airline.csv")
```

We'll also remove 90% of the rows in the data set. We'll do this by removing 90% of the satisfied reviews, and 90% of the dissatisfied reviews, and combining what remains.

We're doing this purely to make the Ensemble Techniques run quicker, as otherwise, they'll take way too long to run.

```
# Cut down data set, aiming for equal divide between satisfied/dissatisfied
CustomerData_sat <- CustomerData_raw[CustomerData_raw$satisfaction == "satisfied",]
CustomerData_sat <- CustomerData_sat[sample(1:nrow(CustomerData_sat),
                                            nrow(CustomerData_sat)*0.1, replace=FALSE),]
CustomerData_dis <- CustomerData_raw[CustomerData_raw$satisfaction == "dissatisfied",]
CustomerData_dis <- CustomerData_dis[sample(1:nrow(CustomerData_dis),
                                            nrow(CustomerData_dis)*0.1, replace=FALSE),]
CustomerData <- rbind(CustomerData_sat[1:(nrow(CustomerData_sat)),],
                      CustomerData_dis[1:(nrow(CustomerData_dis)),])
```

Cleaning Up The Data Set

Cleaning up data set for logistic regression, by converting qualitative columns into factors.

```
# Factor columns
CustomerData$satisfaction <- factor(CustomerData$satisfaction) # satisfaction
CustomerData$Gender <- factor(CustomerData$Gender) # gender
CustomerData$Customer.Type <- factor(CustomerData$Customer.Type) # customer type
CustomerData$Type.of.Travel <- factor(CustomerData$Type.of.Travel) # travel type
CustomerData$Class <- factor(CustomerData$Class) # class
```

```r
# Normalize factor names
levels(CustomerData$satisfaction) <- c("Dissatisfied", "Satisfied")
levels(CustomerData$Customer.Type) <- c("Disloyal", "Loyal")
levels(CustomerData$Type.of.Travel) <- c("Business", "Personal")

# Continue factoring numeric finite columns
for(i in 8:21) {
  CustomerData[,i] <-
    factor(CustomerData[,i], levels=c(0,1,2,3,4,5)) # out-of-5 ratings
}

# Normalize column names
names(CustomerData) <- gsub("\\.", " ", names(CustomerData))
names(CustomerData) <- str_to_title(names(CustomerData))
names(CustomerData) <- gsub("\\ ", ".", names(CustomerData))

# Remove na rows
CustomerData <- CustomerData[complete.cases(CustomerData),]
```

Dividing Into Train/Test

Dividing the data set into train/test...

```r
# reset seed
set.seed(1234)

# train/test division
i <- sample(1:nrow(CustomerData), nrow(CustomerData)*0.8, replace=FALSE)
train <- CustomerData[i,]
test <- CustomerData[-i,]
```

Data Exploration

Please refer to the Classification portion of this assignment for exploration of this data.

Models

Model Training

DT Random Forest

```r
# Start time capture
time_start_rt1 <- Sys.time()

# create model
rf1 <- randomForest(Satisfaction~., data=train, importance=TRUE)
summary(rf1)
```

```
##                 Length Class  Mode
## call               4  -none- call
## type               1  -none- character
## predicted      10361  factor numeric
## err.rate        1500  -none- numeric
## confusion          6  -none- numeric
## votes          20722  matrix numeric
## oob.times      10361  -none- numeric
## classes            2  -none- character
## importance        88  -none- numeric
## importanceSD      66  -none- numeric
## localImportance    0  -none- NULL
## proximity          0  -none- NULL
## ntree              1  -none- numeric
## mtry               1  -none- numeric
## forest            14  -none- list
## y              10361  factor numeric
## test               0  -none- NULL
## inbag              0  -none- NULL
## terms              3  terms  call
```

```
# Stop time capture
time_end_rt1 <- Sys.time()
runtime_rt1 <- time_end_rt1-time_start_rt1
```

DT XGBoost

```
# train label/matrix
train_label <- ifelse(train$Satisfaction=="Satisfied", 1, 0)
train_matrix <- data.matrix(train[, -1])

# Start time capture
time_start_xgb1 <- Sys.time()

# create model
xgb1 <- xgboost(data=train_matrix, label=train_label,nrounds=100,
                objective='binary:logistic', verbose=0)
summary(xgb1)
```

```
##                Length Class              Mode
## handle              1 xgb.Booster.handle externalptr
## raw            279532 -none-             raw
## niter               1 -none-             numeric
## evaluation_log      2 data.table         list
## call               14 -none-             call
## params              2 -none-             list
## callbacks           1 -none-             list
## feature_names      22 -none-             character
## nfeatures           1 -none-             numeric
```

```
# Stop time capture
time_end_xgb1 <- Sys.time()
runtime_xgb1 <- time_end_xgb1-time_start_xgb1
```

DT AdaBoost

```
# reset seed
set.seed(1234)

# Start time capture
time_start_ab1 <- Sys.time()

# create model
ab1 <- boosting(Satisfaction~., data=train, boos=TRUE,
                  mfinal=40, coeflearn='Breiman')
summary(ab1)
```

```
##            Length Class   Mode
## formula         3  formula call
## trees          40  -none-  list
## weights        40  -none-  numeric
## votes       20722  -none-  numeric
## prob        20722  -none-  numeric
## class       10361  -none-  character
## importance     22  -none-  numeric
## terms           3  terms   call
## call            6  -none-  call
```

```
# Stop time capture
time_end_ab1 <- Sys.time()
runtime_ab1 <- time_end_ab1-time_start_ab1
```

Model Predictions

DT Random Forest

```
##
## pred_rf1       Dissatisfied Satisfied
##   Dissatisfied         1094        73
##   Satisfied              81      1343
```

```
## Accuracy:  0.940563489000386
## MCC:  0.880037131563128
## Runtime:  20.3644120693207 s
```

DT XGBoost

```
##          test_label
## pred_xgb1    0    1
```

```
##         0 1115   57
##         1   60 1359

## Accuracy:  0.954843689695098
## MCC:  0.90888188927005
## Runtime:  1.14200901985168 s
```

DT AdaBoost

```
##
##                Dissatisfied Satisfied
##   Dissatisfied         1097        67
##   Satisfied              78      1349

## Accuracy:  0.944037051331532
## MCC:  0.887039878458568
## Runtime:  22.5660479068756 s
```

Analysis

Looking at each of the above prediction results, we can observe a number of things about each ensemble technique. We'll discuss the results of ensemble individually.

Random Forest

The random forest ensemble seems to run on this data somewhere between 19-21 seconds. It produced an accuracy of 94.06%, which is a considerably good result. The random forest is strong in that it can discover new trees that outperform the basic decision tree, which simply chooses the strongest predictor first to build from. The MCC score is high in comparison to the results of the Classification portion of this assignment, as well.

XGBoost

XGBoost ran extremely fast on this data set, taking around 1-2 seconds to complete the algorithm. It produced a notably higher accuracy than the random forest ensemble, of 95.48% accuracy, along with a considerably higher MCC value. This algorithm utilizes the machine's multithreading to generate hundreds of trees, which are then aggregated. It's because of this that the algorithm performs at an extremely fast speed, while achieving greater results.

AdaBoost

AdaBoost ran for roughly the same time as the random forest ensemble, between 20-23 seconds. It produced a slightly better accuracy of 94.40%, as well as a slightly improved MCC value. The algorithm works by iterating through a select number of times - 40 times, in this case - and increases/decreases weights for training examples through each iteration depending on whether or not the error was large/negligible correspondingly. Afterwards, the learners themselves are given a weights, with accurate learners having high weights. For this reason, we're able to achieve a more accurate model than random forest through the means of these iterations.

Conclusion

It's clear that XGBoost outperforms both Random Forest and AdaBoost in all metrics. Of course, XGBoost is much more modernized and has the advantage of utilizing the machine's multithreaded processing, so this is to be expected.

Compared with our original dive into decision trees in one of the previous assignments, it's clear that ensemble methods help make the decision tree model more dependable as a basis. Of course, we know from the textbook that in general, these ensemble techniques will perform best with low-bias, high-variance learners, so despite its strength, it's important to understand its weakness lies in the opposite.