FEM11152 – Data Science for Marketing Analytics Week 5: Case B

Eleftherios Tranakos - EBB5 Team 5

2023-11-30

Introduction

This report aims at predicting the mode of transportation chosen by people for their daily routines. More specifically, the mode of transportation are consisted of public transport and private car. The dataset we are going to analyse includes 5.896 options of people who chose to use either means of transportation or their car based on the influence of other features such as duration of trip, distance, income and education.

Ensemble Models & Boosting

Before we explain what boosting is, we will start by diving first to ensemble methods. So, ensemble methods is a machine learning technique that mix a variety OF models in order to improve the overall performance of decision trees. By combining multiple models, ensemble models enhance the accuracy of the predictions and become more resilient against uncertainties in the data. In addition, these methods deals with model over fitting and model complexity, being at the same time robust to noise and outliers.

Heading now to boosting method, it attempts to create a powerful classifier based on the number of weak classifiers/learners. It's a repetitive procedure for modifying the distribution of training data automatically by emphasizing more on the formerly misclassified records by assigning weights. Specifically, in the beginninhg, a model is created from the training data and all records have the same weight. On the next rounds, records that are not classified correctly will have higher weights in contrast to the correctly classified which will have their weights decreased. Thus, the model tries to correct the errors present in the previous rounds.

On the other hand, apart from boosting, there is also the bagging (Bootstrap Aggregating) and Random forest method. The main difference is that the former focuses on correcting errors while the later aim at reducing the variance by aggregating multiple independent models. Particularly, bagging tend to overfitting less and its independent models are trained on randomly sampled subsets of the data(with replacement). Furthermore, Random Forest is a specific type of bagging that decreases overfitting even more by introducing randomness as we built the tree. It randomly choose subsets of features at each split and mix the predictions of multiple decision trees to enhance the accuracy and the variance reduction.

Optimal Model & Tuning the number of Trees

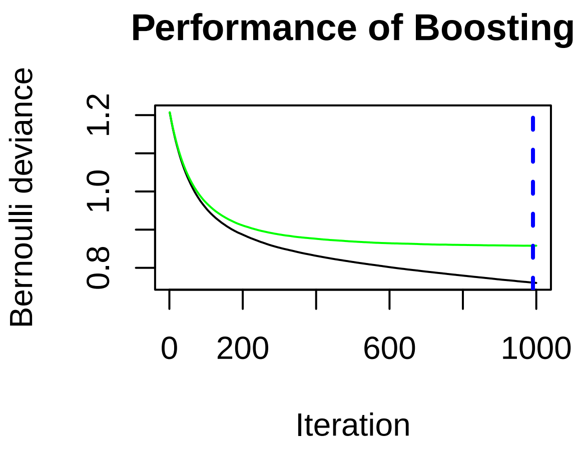
Boosting includes several packages to be used such as gradient boosting, extreme gradient boosting and Adaptive boosting. Applying them all, the accuracy of the predictions was on similar levels around 0.79/1. However, we decided to proceed with the first one as the rest have high computational complexity. Firstly, we used bernoulli distribution indicating binary classification as our response variable has only two outcomes 0-1 (public transport and private car). The boosting iteration of trees was set up to 1000(choosing 991 as the optimal number of trees with Cross Validation) and the interaction depth(max depth of individual trees) up to 5 which is a quite common value. Also the learning rate(shrinkage) and the bag fraction were 0.01 and 0.07, respectively, reasonable starting points, as the former's smaller values often lead to better results and the later's prevent overfitting. Ending to the most significant parameter, cross validation , was set up equal to 10 and helped us estimate the model's performance and tune the hyperparameters.

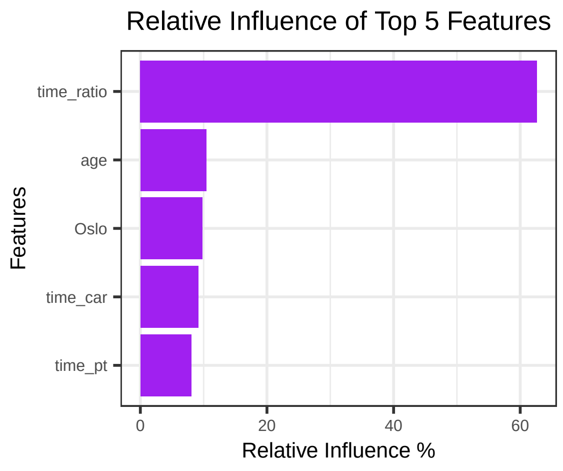
Predictions & Results

Heading to the predictions, as was already mentioned, we used gradient boosting in order to train our model and predict the results. The left plot below represents the performance of our boosting model with 1000 iterations. The black line shows the performance on the training set while the green line indicates the performance on the test set. When the line reaches a plateau means that additional trees will not lead to important improvements. The optimal number of trees that was chosen is 991 represented by the blue dashed line. Lastly, the fact that the black and green line are quite close suggests that the model is generalizing and does not overfit.

Regarding to the right plot below, on this are represented the features which have a great impact on the prediction of whether an individual will choose public transport or private car. Specifically, the feature "time_ratio" suggests that the time it takes to travel by public transport compared to a car is the most important factor in a decision-making option. To support further the significance of this feature, on the Appendix A can be found a partial dependence plot which shows the strong relationship between this feature and its response variable.

Unquestionably, it is worth mentioning that three out of the five features are related with time and duration meaning that individuals take it into account as a major factor to choose the mode of transportation.





Having seen the most important features, the confusion matrix in Table 1 can help us evaluate the performance of the classification model. The bottom right cell, number 1104, represents the number of true positive which is the number of times the model correctly forecast the actual car users. On the other hand, the top left cell, number 301, shows the number of true negatives which is the number of times the model predicted accurately the actual public transport users. The rest of the numbers are false positives and false negatives, representing the times the model incorrectly predicted the mode of transportation users. Summing up the correct predictions and dividing it by the sum of all the predictions we have an accuracy equals to 0,79 which is the proportion of all predictions that were correct. In conclusion, we forecast that most of the people will use their private car in the future.

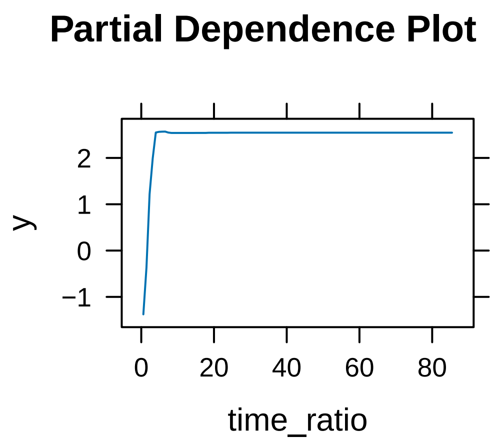
Table 1: Confusion Matrix

	0	1
0	301	144
1	219	1104

References

- https://www.analyticsvidhya.com/blog/2018/06/comprehensive-guide-for-ensemble-models/
- https://www.geeksforgeeks.org/boosting-in-r/
- An Introduction to Statistical Learning with Applications in R (Second Edition) by Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani.

Appendix A



Appendix B

```
#load the dataset

setwd("C:/Users/lefte/Downloads")
load("TransportModeSweden.RData")

#load necessary libraries
if (!require(gbm)) install.packages("gbm")
if (!require(caret)) install.packages("caret")
if (!require(ggplot2)) install.packages("ggplot2")
library(gbm)
library(caret)
library(ggplot2)

#set seed-split dataset into train-test
set.seed(123)
index <- createDataPartition(data$mode, p = 0.7, list = FALSE)
data$mode<-as.character(data$mode)
train <- data[index, ]
test <- data[-index, ]
# run the gbm model - tune the parameters
model <- gbm(
  formula = mode ~ .,
  distribution = "bernoulli", #binary classification (0-1)
  data = train,
  n.trees = 1000,  #1000 iterations
  interaction.depth = 5, #max depth tree - we do not use high value in order to avoid overfitting
  shrinkage = 0.01, #learning rate - 0.01 for better performance
```

```r
  bag.fraction = 0.7, #0.7 common starting point/avoid overfitting
  n.minobsinnode = 3, #minimum number of observations in a terminal node
  cv.folds = 10) #number of cross-validation folds used for model selection
summary(model)
#finding the optimal number of trees
optimal_tree_numb <- gbm.perf(model, method = "cv")
title(main="Performance of Boosting")

print(optimal_tree_numb) #991

#making predictions on the test data
predictions <- predict(model, newdata = test, n.trees = optimal_tree_numb, type = "response")
#converting probabilities to class | cutoff=0.5
class_pred <- ifelse(predictions > 0.5, "1", "0")
class_pred <- as.factor(class_pred)

#calculating the confusion matrix
conf_mat <- table(Predicted = class_pred, Actual = test$mode)

#Calculating the accuracy
accuracy <- round(sum(diag(conf_mat)) / sum(conf_mat),2)

#printing the confusion matrix and accuracy for GBM
print(conf_mat)
print(accuracy)

#summary.model is the dataframe with the relative influences
summary.model<-as.data.frame(summary(model))
summary.model <- summary.model[order(-summary.model$rel.inf),][1:5,]

# Calculate percentages for the relative influence
total_influence <- sum(summary.model$rel.inf)
summary.model$rel.inf.percent <- summary.model$rel.inf / total_influence * 100

#bar plot of feature importance with a black-and-white theme
feat_import<-ggplot(summary.model, aes(x = reorder(var, rel.inf.percent), y = rel.inf.percent)) +
  geom_col(fill = 'purple') +
  coord_flip() +
  labs(title = "Relative Influence of Top 5 Features",
       y = "Relative Influence %",
       x = "Features") +
  theme_bw() +
  theme(text = element_text(size = 8),
        plot.title = element_text(hjust = 0.5))
# Plotting the Partial Dependence Plot - mode/time_pt variable
plot(model,i="time_pt")
optimal_tree_numb <- gbm.perf(model, method = "cv")
title(main="Performance of Boosting")
feat_import
knitr::kable(conf_mat, caption = "Confusion Matrix")
# Plotting the Partial Dependence Plot - mode/time_pt variable
plot(model,i="time_ratio", main = "Partial Dependence Plot")
```

```r
####Adaboost####

library(rpart.plot)
library(adabag)
library(caret)
set.seed(123)  # Setting a seed for reproducibility
index <- createDataPartition(data$mode, p = 0.7, list = FALSE)
trainSet <- data[index, ]
testSet <- data[-index, ]
devtools::install_github("souravc83/fastAdaboost")
# 10-fold cross-validation
fitControl <- trainControl(method = "cv", number = 10, search = "grid")

# grid of hyperparameters to tune
grid <- expand.grid(nIter = c(100), # Number of boosting iterations
                    method = c("real", "discrete", "gentle"))
# Train the AdaBoost model using caret
set.seed(123)
tunedModel <- train(mode ~ ., data = trainSet,
                    method = "adaboost",
                    trControl = fitControl,
                    tuneGrid = grid,
                    maxdepth = 5,
                    minbucket=3,
                    tree.depth = 3,
                    metric = "Kappa")
#0.44 Kappa - 0.78 Accuracy
#  best-tuned model parameters
print(tunedModel$bestTune)
# Predict using the best-tuned model
bestPredictions <- predict(tunedModel, newdata = testSet)
# Evaluate the best-tuned model
confusionMatrix(bestPredictions, testSet$mode)
#accuracy 0,77


###XGBOOST####
library(xgboost)
library(caret)
library(dplyr)
library(pROC)
# Splitting the dataset into training and test sets
set.seed(123) # for reproducibility
index <- createDataPartition(data$mode, p = 0.7, list = FALSE)
trainData <- data[index,]
testData <- data[-index,]
trainData$mode <- as.numeric(as.factor(trainData$mode)) - 1  # Convert factors to 0-indexed numerics
if (any(!trainData$mode %in% c(0, 1))) {
  stop("Error: Labels are not correctly encoded as 0 or 1.")
}
if (any(is.na(trainData))) {
  trainData <- na.omit(trainData)  # Removing rows with NAs
}
```

```r
# Convert the data into DMatrix format
dtrain <- xgb.DMatrix(as.matrix(trainData[, -1]), label = trainData$mode)
dtest <- xgb.DMatrix(as.matrix(testData[, -1]), label = testData$mode)

# Set the parameters
params <- list(max_depth = 3,
               objective = "binary:logistic",
               eval_metric = "error")
# Hyperparameter tuning using caret
param_grid <- expand.grid(
  nrounds = 10,
  max_depth = 5,
  eta = 0.01,
  gamma = 0,
  colsample_bytree = 0.8,
  min_child_weight = c(3,5),
  subsample = 0.8)
# Convert the response variable to a factor with valid levels
trainData$mode <- factor(trainData$mode, levels = c(0, 1), labels = c("Class0", "Class1"))
trainData$mode <- factor(trainData$mode, levels = unique(trainData$mode)
ctrl <- trainControl(method = "cv", number = 10, classProbs = TRUE)
# Train the model
tuned_model <- train(
  x = as.matrix(trainData[, -1]),  # Features
  y = trainData$mode,  # Response variable as a factor for classification
  method = "xgbTree",
  tuneGrid = param_grid,
  metric = "Accuracy")
# best parameters
best_params <- tuned_model$bestTune
params$max_depth <- best_params$max_depth
params$eta <- best_params$eta
params$subsample <- best_params$subsample
params$colsample_bytree <- best_params$colsample_bytree
# Train the final model with tuned parameters
xgb_model <- xgb.train(
  params = params,
  data = dtrain,
  nrounds = 1000,
  watchlist = list(train = dtrain, test = dtest),
  verbose = FALSE)
# Predict on test data
pred <- predict(xgb_model,dtest)
# Convert probabilities to class predictions
pred_class <- factor(ifelse(pred > 0.5, 1, 0), levels = levels(testData$mode))

conf_matrix <- confusionMatrix(data = pred_class, reference = testData$mode)
# accuracy - confusion matrix
accuracy <- conf_matrix$overall["Accuracy"]
# Print
cat("Accuracy: ", accuracy, "\n")
#accuracy = 0.79 but less correctly classified records 0 291 yes 229 no - 1 1120 yes 128 no
```