5. CART Model, Random Forest, and Gradient Boosting

Data Processing

```r
#################### DATA Processing ####################

# read Airbnb data
data <- read.csv("Airbnb_Data.csv")

# delete id and text variables
data <- select(data, -c(id, description, name))

# convert categorical variables to factors
data$room_type <- as.factor(data$room_type)
data$property_type <- as.factor(data$property_type)
data$bed_type <- as.factor(data$bed_type)
data$cancellation_policy <- as.factor(data$cancellation_policy)
data$city <- as.factor(data$city)
data$host_has_profile_pic <- as.factor(data$host_has_profile_pic)
data$host_identity_verified <- as.factor(data$host_identity_verified)
data$host_response_rate <- as.factor(data$host_response_rate)
data$instant_bookable <- as.factor(data$instant_bookable)

# convert host_response_rate to numeric by removing the '%' sign
data$host_response_rate <- as.numeric(gsub("%", "", data$host_response_rate))

# handle amenities variable: transform into multiple binary columns
data$amenities <- str_replace_all(data$amenities, '[{}"]', '')
amenities_list <- str_split(data$amenities, ",")
all_amenities <- unique(unlist(amenities_list))
for (amenity in all_amenities) {
  data[[amenity]] <- sapply(amenities_list, function(x) amenity %in% x)
}
data <- select(data, -amenities)

# handle date variables
data$first_review <- as.Date(data$first_review, format="%m/%d/%Y")
data$last_review <- as.Date(data$last_review, format="%m/%d/%Y")
data$host_since <- as.Date(data$host_since, format="%m/%d/%Y")

# transform date variables into more meaningful variables
data$days_since_first_review <- as.numeric(Sys.Date() - data$first_review)
data$days_since_last_review <- as.numeric(Sys.Date() - data$last_review)
data$host_duration <- as.numeric(Sys.Date() - data$host_since)
# delete first_review, last_review, host_since
data <- select(data, -c(first_review, last_review, host_since))

# turn thumbnail_url into a binary categorical variable
data$has_thumbnail <- ifelse(is.na(data$thumbnail_url) | data$thumbnail_url == "", FALSE, TRUE)
data <- select(data, -thumbnail_url) # delete thumbnail_url

# handle missing values
count_missing <- function(x) {
  sum(is.na(x) | x == "")}
```

```r
# create a summary of missing values (NA and empty strings) for each column
missing_values_summary <- data %>%
  summarise_all(count_missing) %>%
  gather(key = "variable", value = "missing_count") %>%
  arrange(desc(missing_count))
print(missing_values_summary)

# impute the above numerical variables that have missing values with median values
data$host_response_rate[is.na(data$host_response_rate)] <-
  median(data$host_response_rate, na.rm = TRUE)
data$review_scores_rating[is.na(data$review_scores_rating)] <-
  median(data$review_scores_rating, na.rm = TRUE)
data$days_since_first_review[is.na(data$days_since_first_review)] <-
  median(data$days_since_first_review, na.rm = TRUE)
data$days_since_last_review[is.na(data$days_since_last_review)] <-
  median(data$days_since_last_review, na.rm = TRUE)
data$bathrooms[is.na(data$bathrooms)] <-
  median(data$bathrooms, na.rm = TRUE)
data$host_duration[is.na(data$host_duration)] <-
  median(data$host_duration, na.rm = TRUE)
data$beds[is.na(data$beds)] <- median(data$beds, na.rm = TRUE)
data$bedrooms[is.na(data$bedrooms)] <-
  median(data$bedrooms, na.rm = TRUE)

# replace missing values with specific values for categorical columns
data$host_has_profile_pic[is.na(data$host_has_profile_pic)
                          | data$host_has_profile_pic == ''] <- 'f'
data$host_identity_verified[is.na(data$host_identity_verified)
                            | data$host_identity_verified == ''] <- 'f'
data$neighbourhood[is.na(data$neighbourhood)
                   | data$neighbourhood == ''] <- 'Unknown'
data$zipcode[is.na(data$zipcode)
             | data$zipcode == ''] <- 'Unknown'
# convert to factors
data$neighbourhood <- as.factor(data$neighbourhood)
data$zipcode <- as.factor(data$zipcode)

# make columns names unique and legal
names(data) <- make.names(names(data), unique = TRUE)

# extract numerical variable names
numeric_vars <- c("log_price", "accommodates", "bathrooms", "host_response_rate",
                  "latitude", "longitude", "number_of_reviews", "review_scores_rating",
                  "bedrooms", "beds", "days_since_first_review", "days_since_last_review",
                  "host_duration")
# extract categorical variable names
categorical_vars <- setdiff(names(data), numeric_vars)

# standardize numerical variables
data_numeric <- scale(data[numeric_vars])
data_numeric <- as.data.frame(data_numeric)
# combine standardized numerical variables with categorical variables
data_scale <- cbind(data_numeric, data[categorical_vars])
```

CART Model

Decision Tree

```r
# Split the training data and testing data
set.seed(123)
trainIndex_dt <- createDataPartition(data$log_price, p = .8, list = FALSE, times = 1)
trainData_dt <- data[trainIndex_dt, ]
testData_dt <- data[-trainIndex_dt, ]

# Construct the decision tree model
tree_model <- rpart(log_price ~ ., data = trainData_dt, method = "anova")
```

```r
# Visualize the decision tree
rpart.plot(tree_model)

# In-sample prediction and R^2
train_predictions <- predict(tree_model, newdata = trainData_dt)
train_r2 <- 1 - sum((trainData_dt$log_price - train_predictions)^2)
            / sum((trainData_dt$log_price - mean(trainData_dt$log_price))^2)
print(paste("Decision Tree In-sample R^2:", train_r2))

# Out-of-sample prediction and R^2
tree_predictions <- predict(tree_model, newdata = testData_dt)
tree_r2 <- 1 - sum((testData_dt$log_price - tree_predictions)^2)
            / sum((testData_dt$log_price - mean(testData_dt$log_price))^2)
print(paste("Decision Tree Out-of-sample R^2:", tree_r2))

# Calculate RMSE for out-of-sample
tree_rmse <- sqrt(mean((testData_dt$log_price - tree_predictions)^2))
print(paste("Decision Tree RMSE:", tree_rmse))
```

Feature Importance of Decision Tree

```r
# Calculate importance of features
var_importance <- varImp(tree_model)

# Order the importance and choose the non-zeros
importance_data <-
  data.frame(Variables = rownames(var_importance), Importance = var_importance$Overall)
importance_data <- importance_data[order(-importance_data$Importance), ]  # order
top10_importance <- head(importance_data, 10)  # choose top 10 variables

# Plot the importance graph
ggplot(top10_importance, aes(x = reorder(Variables, Importance), y = Importance)) +
  geom_bar(stat = "identity", color = "indianred1", fill = "indianred1") +
  coord_flip() +
  xlab("Variables") +
  ylab("Importance") +
  ggtitle("Top 10 Variable Importance in Decision Tree") +
  theme_minimal()
```

Complexity Parameter (CP) Plot of Decision Tree

```r
# Visualize "cp" table/plot CV result
plotcp(tree_model, main = "Cross Validation Error vs. Complexity Parameter (CP)")


# Adjust "cp" to prune the tree
optimal_cp <- tree_model$cptable[which.min(tree_model$cptable[,"xerror"]),"CP"]
pruned_tree_model <- prune(tree_model, cp = optimal_cp)

# Use the pruned tree to do in-sample prediction
pruned_train_predictions <- predict(pruned_tree_model, newdata = trainData_dt)

pruned_train_r2 <-
  1 - sum((trainData_dt$log_price - pruned_train_predictions)^2)
  /sum((trainData_dt$log_price - mean(trainData_dt$log_price))^2)

print(paste("Pruned Decision Tree In-sample R^2:", pruned_train_r2))

# Use the pruned tree to do out-of-sample prediction
pruned_tree_predictions <- predict(pruned_tree_model, newdata = testData_dt)
pruned_tree_r2 <-
  1 - sum((testData_dt$log_price - pruned_tree_predictions)^2)
  / sum((testData_dt$log_price - mean(testData_dt$log_price))^2)

print(paste("Pruned Decision Tree Out-of-sample R^2:", pruned_tree_r2))

# Calculate RMSE for pruned tree
pruned_tree_rmse <- sqrt(mean((testData_dt$log_price - pruned_tree_predictions)^2))
print(paste("Pruned Decision Tree RMSE:", pruned_tree_rmse))
```

Random Forest

```r
#################### Random Forest ####################

# delete zipcode and neighbourhood, because they have too much
# categories that rf model does not support
sample_data <- select(data, -c(zipcode, neighbourhood))

# split training and testing set
set.seed(123)
trainIndex_rf <- createDataPartition(sample_data$log_price, p = .8, list = FALSE, times = 1)
trainData_rf <- sample_data[trainIndex_rf, ]
testData_rf <- sample_data[-trainIndex_rf, ]

# construct a ramdom forest model
rf <- randomForest(log_price ~ ., data = trainData_rf, ntree = 250, nodesize = 25, importance = TRUE)
```

```r
# plot the rf: this is an error graph instead of the tree visualization
plot(rf)

# calculate in-sample R2
in_sample_predictions <- predict(rf, trainData_rf)
in_sample_r2 <- 1 - sum((trainData_rf$log_price - in_sample_predictions)^2)
```

```r
/ sum((trainData_rf$log_price - mean(trainData_rf$log_price))^2)

print(paste("In-sample R2:", in_sample_r2))

# calculate out-of-sample R2
out_sample_predictions <- predict(rf, testData_rf)
out_sample_r2 <- 1 - sum((testData_rf$log_price - out_sample_predictions)^2)
/ sum((testData_rf$log_price - mean(testData_rf$log_price))^2)

print(paste("Out-of-sample R2:", out_sample_r2))
```

Feature Importance of Random Forest

```r
# plot the importance of features
varImpPlot(rf, type=1, pch=21, bg="indianred1", main='RF variable importance')
```

Actual vs. Predicted Values of Random Forest

```r
# predict using rf
predicted <- predict(rf, newdata = sample_data)

# calculate residuals
residuals <- sample_data$log_price - predicted

# prepare data for ggplot
rf_graph_data <- data.frame(
  Actual = sample_data$log_price,
  Predicted = predicted,
  Residuals = residuals
)

range_vals_rf <- range(c(rf_graph_data$Actual, rf_graph_data$Predicted))

# plot predicted values vs. actual values & residual plot
p1 <- ggplot(rf_graph_data, aes(x = Actual, y = Predicted)) +
  geom_point(shape = 21, color = "indianred1", fill = "mistyrose1", size = 1.5, alpha = 0.15) +
  geom_abline(intercept = 0, slope = 1, color = "black", linewidth = 0.6) +
  labs(title = "Actual VS. Predicted (RF)",
       x = "Actual Value (log_price)", y = "RF Predicted Value (log_price)") +
  xlim(range_vals) + ylim(range_vals) +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5, face = "bold"))

p2 <- ggplot(rf_graph_data, aes(x = Predicted, y = Residuals)) +
  geom_point(shape = 21, color = "indianred1", fill = "mistyrose1", size = 1.5, alpha = 0.15) +
  geom_hline(yintercept = 0, color = "black", linewidth = 0.6) +
  labs(title = "Residual Plot of Random Forest",
       x = "Predicted Value (RF) (log_price)", y = "Residual") +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5, face = "bold"))

grid.arrange(p1, p2, ncol = 2)
```

XGBOOST

```
#################### XGBOOST ########################

# split training and testing data
set.seed(123)
trainIndex_xg <- createDataPartition(data$log_price, p = .8, list = FALSE, times = 1)
trainData_xg <- data[trainIndex_xg, ]
testData_xg <- data[-trainIndex_xg, ]

# turn factor variables into one-hot-code
train_matrix <- model.matrix(log_price ~ . - 1, data = trainData_xg)
test_matrix <- model.matrix(log_price ~ . - 1, data = testData_xg)

# prepare data matrix
train_label <- trainData_xg$log_price
test_label <- testData_xg$log_price
dtrain <- xgb.DMatrix(data = train_matrix, label = train_label)
dtest <- xgb.DMatrix(data = test_matrix, label = test_label)

# set XGBoost model parameters
params <- list(
  objective = "reg:squarederror",
  eta = 0.1,
  max_depth = 6,
  eval_metric = "rmse"
)
```

```
# train the model
xgb_model <- xgb.train(
  params = params,
  data = dtrain,
  nrounds = 250,
  watchlist = list(train = dtrain, eval = dtest),
  early_stopping_rounds = 10,
  print_every_n = 10
)
```

```
# calculate in-sample R2
xgb_train_predictions <- predict(xgb_model, dtrain)
xgb_train_r2 <- 1 - sum((train_label - xgb_train_predictions)^2)
                / sum((train_label - mean(train_label))^2)
print(paste("XGBoost In-sample R2:", xgb_train_r2))

# calculate out-of-sample R2
xgb_test_predictions <- predict(xgb_model, dtest)
xgb_test_r2 <- 1 - sum((test_label - xgb_test_predictions)^2)
                / sum((test_label - mean(test_label))^2)
print(paste("XGBoost Out-of-sample R2:", xgb_test_r2))
```

Feature Importance of XGBoost

```
# obtain the feature importance and plot it
importance_matrix <- as.data.frame(head(xgb.importance(
```

```r
  feature_names = colnames(train_matrix), model = xgb_model), 20))
ggplot(importance_matrix, aes(x = reorder(Feature, Gain), y = Gain)) +
  geom_bar(stat = "identity", fill = "indianred1") +
  coord_flip() +
  xlab("Features") +
  ylab("Importance (Gain)") +
  ggtitle("Top 20 Important Features") +
  theme_minimal() +
  theme(
    plot.title = element_text(hjust = 0.5, face = "bold")
  )
```

Actual vs. Predicted Values of XGBoost

```r
# prepare data for scatter plot
xg_graph_data <- data.frame(
  Actual = test_label,
  Predicted = xgb_test_predictions,
  Residuals = test_label - xgb_test_predictions
)

range_vals_xg <- range(c(xg_graph_data$Actual, xg_graph_data$Predicted))

# predicted vs. actual
p3 <- ggplot(xg_graph_data, aes(x = Actual, y = Predicted)) +
  geom_point(shape = 21, color = "indianred1", fill = "mistyrose1", size = 1.5, alpha = 0.15) +
  geom_abline(intercept = 0, slope = 1, color = "black", linewidth = 0.6) +
  labs(title = "Actual VS. Predicted (XGBoost)",
       x = "Actual Value (log_price)", y = "XGBoost Predicted Value (log_price)") +
  xlim(range_vals_xg) + ylim(range_vals_xg) +
  theme_minimal() +
  coord_fixed(ratio = 1) +
  theme(
    plot.title = element_text(hjust = 0.5, face = "bold")
  )

# residual plot
p4 <- ggplot(xg_graph_data, aes(x = Predicted, y = Residuals)) +
  geom_point(shape = 21, color = "indianred1", fill = "mistyrose1", size = 1.5, alpha = 0.15) +
  geom_hline(yintercept = 0, color = "black", linewidth = 0.6) +
  labs(title = "Residual Plot of XGBoost",
       x = "Predicted Value (XGBoost) (log_price)", y = "Residual") +
  theme_minimal() +
  coord_fixed(ratio = 1) +
  theme(
    plot.title = element_text(hjust = 0.5, face = "bold")
  )

grid.arrange(p3, p4, ncol = 2)
```