

# Untitled

Mengdi Hao

2024-05-19

## CART Model

### Decision Tree

```
# 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")

# 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

```
# 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 Plot of Decision Tree

```
# 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 -
pruned_train_predictions)^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 -
pruned_tree_predictions)^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

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

## IS and OOS R2 of the Random Forest

```
# 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$log_price - in_sample_predictions)^2) / sum((trainData_rf$log_price - in_sample_predictions)^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 - out_sample_predictions)^2)
print(paste("Out-of-sample R2:", out_sample_r2))

```

## Feature Importance of the Random Forest

```

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

```

## Scatter Plot for the Random Forest

```

# 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
)
```

## IS and OOS R2 for the XGBoost

```
# 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 the XGBoost

```
# obtain the feature importance and plot it
importance_matrix <- as.data.frame(head(xgb.importance(feature_names = colnames(train_matrix), model = xgb_model), 10))
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")
)

```

## Scatter Plot of the XGBoost

```

# 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") +
  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)

```