Classification and Regression Trees

2024-09-20

1 Participate in Kaggle's House Prices: Advanced Regression Techniques competition, and implement the following steps:

1. Fit CART models, specifically regression models to a continuous outcome variable y
2. Compare different *estimates* of the RMSLE score on the test data returned by Kaggle
3. Studying the impact of the complexity parameter (cp) on RMSLE scores. This parameter, α, helps minimize the CART objective function $RSS + \alpha|T|$, and is key for controlling the model's learning complexity (also known as a hyperparameter)
4. Optimizing the cp parameter to achieve the best RMSLE score for submission and validation

2 Model using previously seen variables

Here are the variables I use in the CART model:

1. Outcome variable `SalePrice` directly i.e. don't log-transform it
2. The 6 predictor variables from the PS1 solutions
3. The HalfBath variable from the lecture on overfitting
4. LotArea

Below is the data pre-processing necessary to

```
# Replace missing value in KitchenQual variable in test set
test <- test %>%
  mutate(KitchenQual = ifelse(!is.na(KitchenQual), KitchenQual, "TA"))

# Deal with categorical predictor MSSubClass that had a level in test
# data that didn't exist in training data. Do this by lumping all but
# top two levels in new level "other". Remember to do this to both
```

```r
# training and test
training <- training %>%
  mutate(MSSubClassClean = ifelse(MSSubClass %in% c("20", "60"), MSSubClass, "other"))
test <- test %>%
  mutate(MSSubClassClean = ifelse(MSSubClass %in% c("20", "60"), MSSubClass, "other"))

# Restrict to just the variables we'll use
training <- training %>%
  select(Id, GrLivArea, YearBuilt, BedroomAbvGr, SaleCondition, KitchenQual,
         MSSubClassClean, HalfBath, LotArea, SalePrice)
test <- test %>%
  select(Id, GrLivArea, YearBuilt, BedroomAbvGr, SaleCondition, KitchenQual,
         MSSubClassClean, HalfBath, LotArea)
```

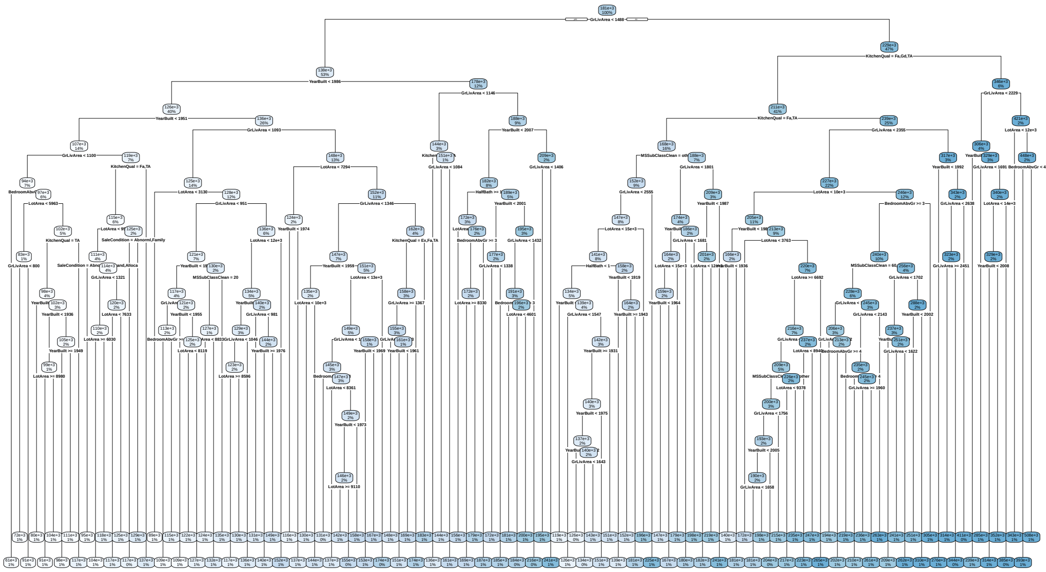
3 MVP Phase 1: Get scores for training and test data

Let's consider a CART model with the above variables and `cp` $= \alpha = 0$. Let's compute both the real Kaggle score on the test data and our first estimate of this score: the score on the training data.

3.1 Fit and visualize

First, fit this model and visualize it using `rpart.plot::rpart.plot()`, which IMO is an improvement over the base R generic `plot()` function

```r
# create CART model with cp of 0
model_CART <- rpart(SalePrice ~ GrLivArea + YearBuilt + BedroomAbvGr + SaleCondition + Kit

# plot CART model
rpart.plot(model_CART)
```



3.2 Compute scores

- Training set score: 0.15220
- Test set score from Kaggle: 0.20428

Question to explore: Why do we observe the discrepancy between these two scores? The complex model caters too much to the training model (score is much lower than test), hindering its applicability to other data.

```
# add predicted values from CART model
training <- data_frame(training, y_hat = predict(model_CART))

# find rmsle for training data
rmsle_training <- sqrt((sum((log(training$y_hat + 1) - log(training$SalePrice + 1))^2))/(n

print(rmsle_training)
```

```
[1] 0.1521967
```

```
# predict sales price on test using fitted/trained model
test$SalePrice <- predict(model_CART, newdata = test)
```

```r
# add sales price values to submission file
submission <- sample_submission %>%
  mutate(SalePrice = test$SalePrice)

# convert into file
write_csv(submission, path = "data/submission.csv")
```

4 Due Diligence Phase 2: Get a better estimate of the Kaggle score

Using the same model with cp = 0 from above, obtain an *estimate* of the test set score Kaggle returns using the method from the `overfitting.R` lecture. Do not change the random number seed nor the training set split method from `overfitting.R`, so that all submissions will have the same split.

- train_validation score: 0.16586
- test_validation score: 0.20169

Questions to explore:

1. Which of the above two scores is the estimate of the Kaggle score? The test_validation score
2. Is this estimate of the Kaggle score better than the previous one? Why? This estimate of the Kaggle score is slightly better than the previous one because the train split is randomly generated. The randomly generated data mimics unseen data and allows it to build a better model for the test data. Using just the test data exposes the model to new characteristics that the model is not familiar with.

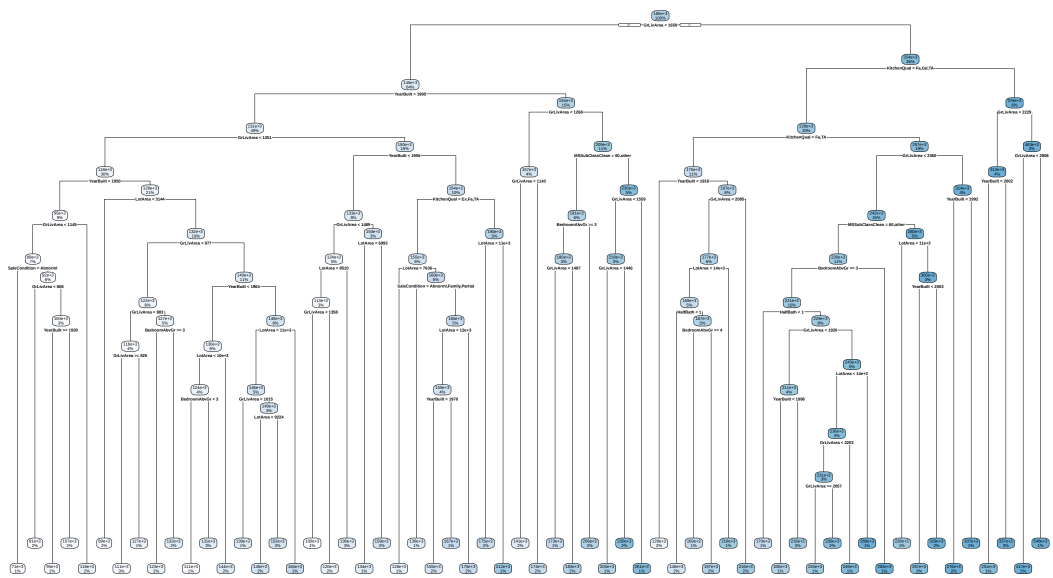
```r
# set seed
set.seed(76)

# split training data into train_validation and test_validation
train_validation <- training %>%
  sample_frac(0.5)
test_validation <- training %>%
  anti_join(train_validation, by = "Id")

# check that the data has been split correctly
nrow(training)
```

```
[1] 1460
```

```r
nrow(train_validation) + nrow(test_validation)
```

```
[1] 1460
```

```r
# CART model for train_validation
model_CART_split <- rpart(SalePrice ~ GrLivArea + YearBuilt + BedroomAbvGr + SaleCondition
                          data = train_validation,
                          control = rpart.control(cp = 0))

# plot CART model
rpart.plot(model_CART_split)
```



```r
# add predicted values from CART model fitted on train_validation
test_validation$SalePrice_hat <- predict(model_CART_split, newdata = test_validation)
# add predicted values from CART model to train_validation
train_validation <- data_frame(train_validation, SalePrice_hat = predict(model_CART_split)

# find rmsle of train and test split
rmsle_train_validation <- sqrt((sum((log(train_validation$SalePrice_hat + 1) - log(train_v
```

```
print(rmsle_train_validation)
```

```
[1] 0.1658643
```

```
rmsle_test_validation <- sqrt((sum((log(test_validation$SalePrice_hat + 1) - log(test_vali

print(rmsle_test_validation)
```

```
[1] 0.2016911
```

5 Due Diligence Phase 3: Using a different cp value

Choose any different value of cp and obtain a new estimate of the Kaggle score.

After choosing cp = 0.05, I got the following scores: - train_validation score: 0.27760 - test_validation score: 0.25121

Questions to explore:

1. **Without** making a submission to Kaggle to see your actual score, which cp value would you use for the Kaggle submission? cp = 0 or your new value? The new value
2. Why? I would use the model of cp = 0.05 because too much complexity when cp = 0 has many drawbacks. The overly complex CART model does not predict well to unseen data due to overfitting and leads to expensive, increased computational costs when cp = 0.

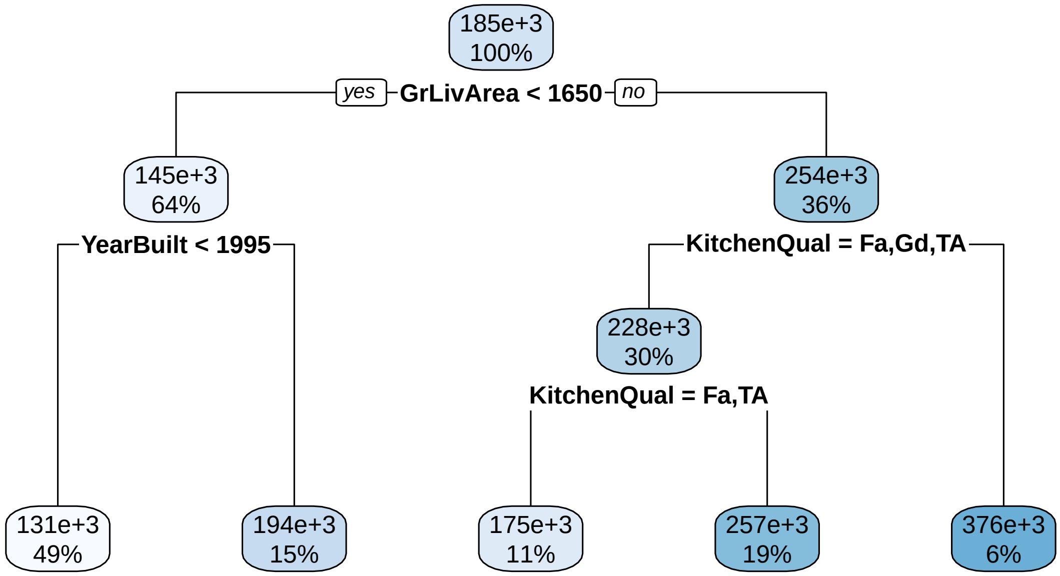
```
# set seed
set.seed(76)

# split training data into train_validation and test_validation
train_validation_new_cp <- training %>%
  sample_frac(0.5)
test_validation_new_cp <- training %>%
  anti_join(train_validation_new_cp, by = "Id")

# check that the data has been split correctly
nrow(training)
```

```
[1] 1460
```

```r
nrow(train_validation_new_cp) + nrow(test_validation_new_cp)
```

```
[1] 1460
```

```r
# CART model for train_validation
model_CART_split_new_cp <- rpart(SalePrice ~ GrLivArea + YearBuilt + BedroomAbvGr + SaleCo
                    data = train_validation_new_cp,
                    control = rpart.control(cp = 0.05))

# plot CART model
rpart.plot(model_CART_split_new_cp)
```




```r
# add predicted values from CART model fitted on train_validation
test_validation_new_cp$SalePrice_hat <- predict(model_CART_split_new_cp, newdata = test_va
# add predicted values from CART model to train_validation
train_validation_new_cp <- data_frame(train_validation_new_cp, SalePrice_hat = predict(mod

# find rmsle of train and test split
rmsle_train_validation_new_cp <- sqrt((sum((log(train_validation_new_cp$SalePrice_hat + 1)
```

```r
print(rmsle_train_validation_new_cp)
```

```
[1] 0.2775991
```

```r
rmsle_test_validation_new_cp <- sqrt((sum((log(test_validation_new_cp$SalePrice_hat + 1) -

print(rmsle_test_validation_new_cp)
```

```
[1] 0.2512112
```

6 Due Diligence Phase 4: Using several different cp values

1. Repeat the above process for at least 100 different cp values
2. Plot all the scores in a ggplot with the following aesthetics:
    1. x-axis: cp complexity parameter
    2. y-axis: RMSLE
    3. color: training vs test set (i.e. your visualization should have two curves of different colors)

Qualitatively describe what are the two or three of the most salient patterns in this graph when it comes to understanding model complexity:

1. The rmsle remains constant ($test = 0.38$, $train = 0.44$) when $cp > 0.35$ for the training and test set.

2. The rmsle is more sensitive when cps are small for both training and test sets.

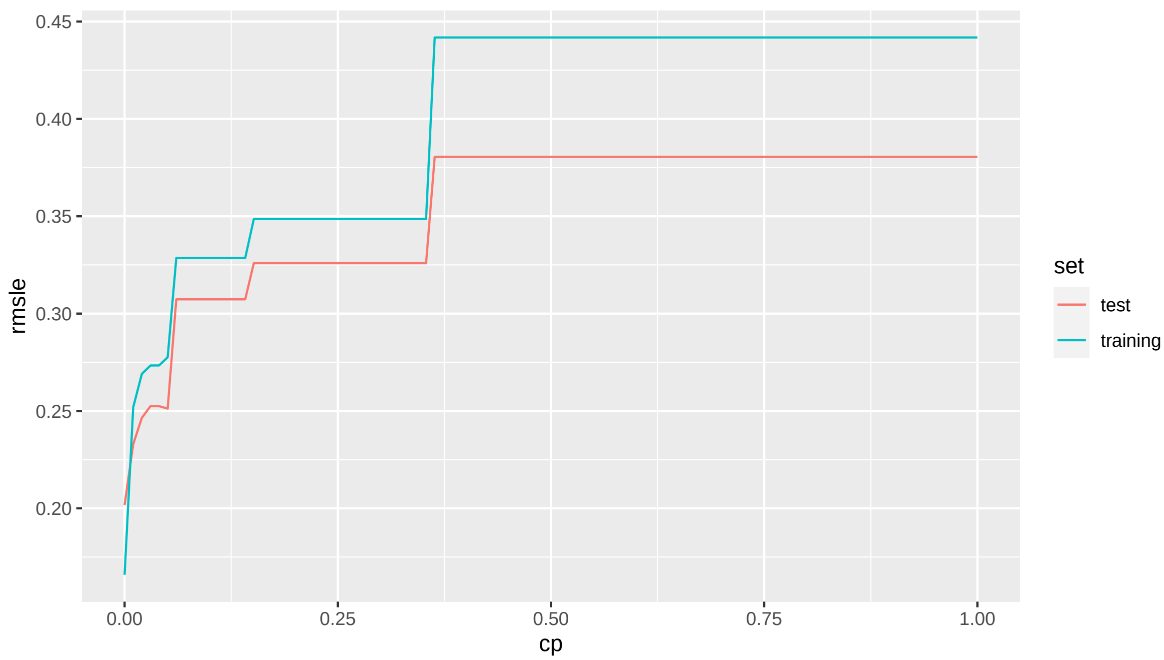
3. When $0 < cp < 0.06$, the rmsle increases at the fastest pace for both training and test sets.

```r
# number of values to generate for test and training
n_values <- 100

# generate 100 cp values from 0 to 1
cp_grid <- seq(from = 0, to = 1, length=n_values)

# generate 100 cp values to include cps from optimal range
cp_grid1 <- seq(from = 0, to = 0.1, length=n_values)

# set seed
```

```r
set.seed(76)

# split training data into train_validation and test_validation
train_validation_cp <- training %>%
  sample_frac(0.5)
test_validation_cp <- training %>%
  anti_join(train_validation_cp, by = "Id")

# get rmsle for train validation
get_rmsle_train <- function(cp) {

  # create model for chosen value for cp
  model_CART_split_cp <- rpart(SalePrice ~ GrLivArea + YearBuilt + BedroomAbvGr + SaleCond
                      data = train_validation_cp,
                      control = rpart.control(cp = cp))

  # predict CART model on train validation
  train_validation_cp <- data_frame(train_validation_cp, SalePrice_hat = predict(model_CAR

  # calculate rmsle for train validation
  rmsle_train_cp <- sqrt((sum((log(train_validation_cp$SalePrice_hat + 1) - log(train_vali

  # remove sale price hat or creates error
  train_validation_cp <- subset(train_validation_cp, select = -c(SalePrice_hat))

  # return rmsle train validation score
  return(rmsle_train_cp)
}

# get rmsle for test validation
get_rmsle_test <- function(cp) {

  # create model for chosen value for cp
  model_CART_split_cp <- rpart(SalePrice ~ GrLivArea + YearBuilt + BedroomAbvGr + SaleCond
                      data = train_validation_cp,
                      control = rpart.control(cp = cp))

  # predict CART model on test validation
  test_validation_cp$SalePrice_hat <- predict(model_CART_split_cp, newdata = test_validati

  # calculate rmsle for test validation
```

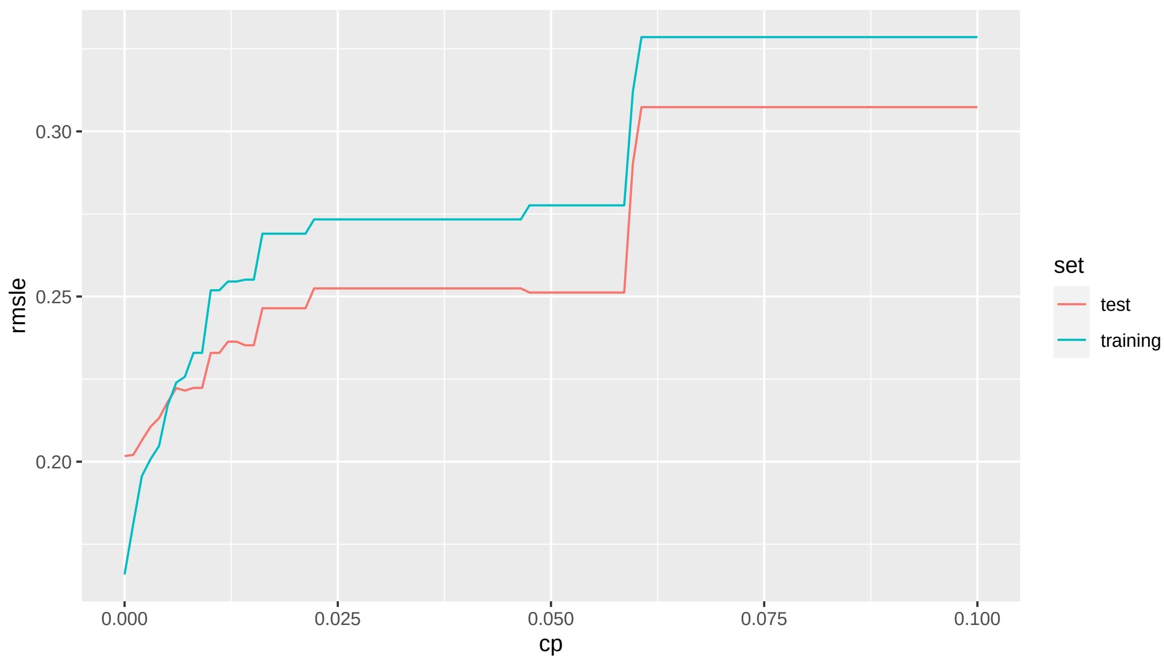
```
  rmsle_test_cp <- sqrt((sum((log(test_validation_cp$SalePrice_hat + 1) - log(test_validat

  # remove sale price hat or creates error
  test_validation_cp <- subset(test_validation_cp, select = -c(SalePrice_hat))

  # return rmsle test validation score
  return(rmsle_test_cp)
}

# initialize train and test rmsle vectors
train_rmsle_vec <- numeric(length(cp_grid))
test_rmsle_vec <- numeric(length(cp_grid))

# Iterate over each cp value
for (i in seq_along(cp_grid)) {

  # Compute RMSLE for the current cp value
  rmsle_train <- get_rmsle_train(cp_grid[i])
  rmsle_test <- get_rmsle_test(cp_grid[i])

  # Insert the computed RMSLE into train_rmsle_vec and test_rmsle_vec
  train_rmsle_vec[i] <- rmsle_train
  test_rmsle_vec[i] <- rmsle_test
}

# focus on a lower cp range for more detailed analysis

# initialize train and test rmsle vectors for lower cp range
train_rmsle_vec_low_cp <- numeric(length(cp_grid1))
test_rmsle_vec_low_cp <- numeric(length(cp_grid1))

# Iterate over each cp value
for (i in seq_along(cp_grid1)) {

  # Compute RMSLE for the current cp value
  rmsle_train1 <- get_rmsle_train(cp_grid1[i])
  rmsle_test1 <- get_rmsle_test(cp_grid1[i])

  # Insert the computed RMSLE into train_rmsle_vec and test_rmsle_vec
  train_rmsle_vec_low_cp[i] <- rmsle_train1
  test_rmsle_vec_low_cp[i] <- rmsle_test1
```

```
}

# create table for rmsle values of train and test validation for cp values from 0 to 1
rmsle <- tibble(
    cp = cp_grid,
    training = unlist(train_rmsle_vec),
    test = unlist(test_rmsle_vec)
  ) %>%
  pivot_longer(-cp, names_to = "set", values_to = "rmsle")

# plot rmsle values for train and test on cps from 0 to 1
ggplot(rmsle, aes(x=cp, y=rmsle, col = set)) +
  geom_line()
```




```
# create table for rmsle values of train and test validation for cp values from 0 to 0.1
rmsle1 <- tibble(
    cp = cp_grid1,
    training = unlist(train_rmsle_vec_low_cp),
    test = unlist(test_rmsle_vec_low_cp)
  ) %>%
  pivot_longer(-cp, names_to = "set", values_to = "rmsle")
```

```r
# plot rmsle values for train and test on cps from 0 to 0.1
ggplot(rmsle1, aes(x=cp, y=rmsle, col = set)) +
  geom_line()
```




```r
minima <- rmsle %>%
  filter(set == "test") %>%
  arrange(rmsle) %>%
  slice(1)
minima
```

```
# A tibble: 1 x 3
     cp set   rmsle
  <dbl> <chr> <dbl>
1     0 test  0.202
```

```r
rmsle_min <- minima$rmsle
cp_star <- minima$cp

print(cp_star)
```

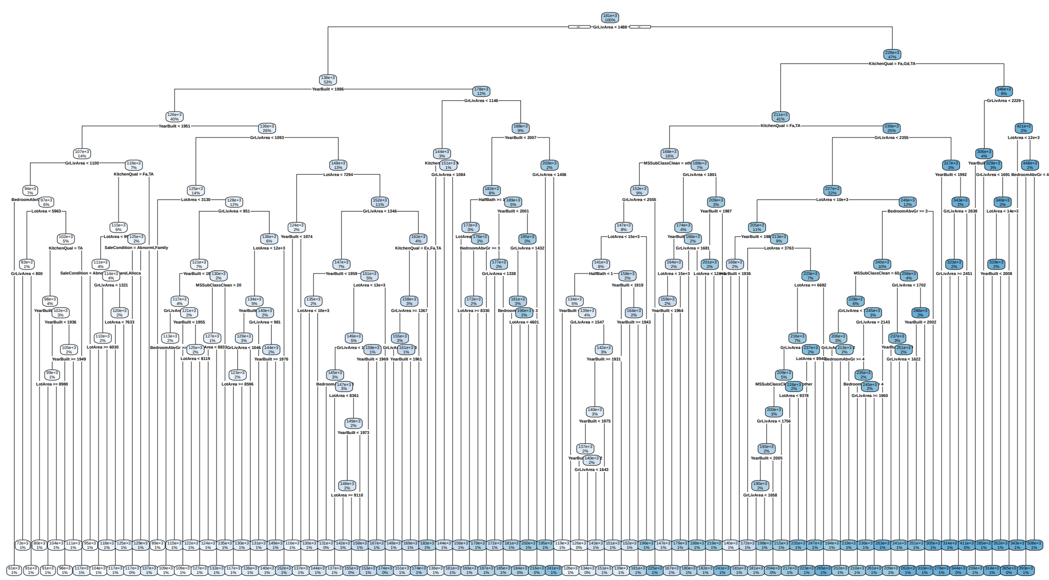
[1] 0

7 Due Diligence Phase 5: Choosing an optimal cp

Based on the results in the previous section, identify the "optimal" cp to use and make a submission to kaggle using this value of cp and obtain the score.

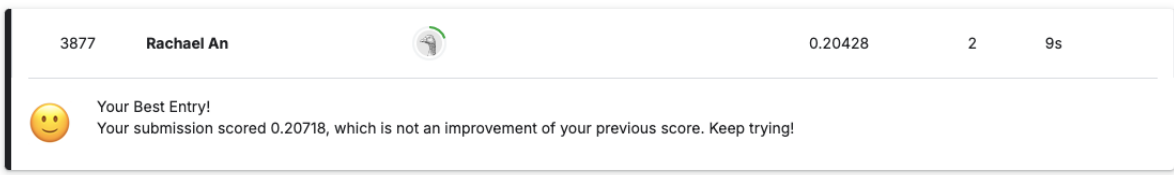
1. What is the optimal cp value? 0.0003808
2. What is the estimate of the RMSLE score returned by Kaggle associated with this optimal cp value? 0.19938
3. What is the actual Kaggle score associated with this optimal cp value? 0.20718
4. Which Kaggle score is better? The one for cp = 0 or for your optimal cp value? The Kaggle score for cp = 0 is better.

```
# create CART model with optimal cp chosen to balance complexity and fit
model_CART_optimal_cp <- rpart(SalePrice ~ GrLivArea + YearBuilt + BedroomAbvGr + SaleCond

# plot CART model of optimal cp
rpart.plot(model_CART_optimal_cp)
```



```r
# predict sales price on test using fitted/trained model
test$SalePrice <- predict(model_CART_optimal_cp, newdata = test)

# add sales price values to submission file
submission_optimal_cp <- sample_submission %>%
  mutate(SalePrice = test$SalePrice)

# convert into file
write_csv(submission_optimal_cp, path = "data/submission_optimal_cp_v2.csv")
```

7.1 Make Kaggle submission



8 Point of Diminished Returns Phase 6: Improving Kaggle test set score estimates

Implement a mechanism for further improving the estimate for the Kaggle score yielded by the train_validation and test_validation split.

I implemented 10-fold cross validation as follows:

```r
# set seed
set.seed(76)

# define 10 folds for 10-fold cross validation
num_folds <- 10

# randomly split the data into 10 folds
folds <- createFolds(training$SalePrice, k = num_folds, list = TRUE, returnTrain = FALSE)

# initialize a vector to hold k-fold numbers
index <- rep(NA, nrow(training))

# iterate along folds
```

```r
for (i in seq_along(folds)) {

  # add fold numbers to index vector
  index[folds[[i]]] <- i
}

# add the index column to your dataset
training <- cbind(index = index, training)

# initialize vector to hold 10 rmsle scores of test_validation
rmsle_score <- rep(0, num_folds)

# iterate along k folds
for (i in 1:num_folds) {

  # initialize test and train validation
  test_validation_k_fold <- training |>
    filter(index == i)
  train_validation_k_fold <- training |>
    filter(index != i)

  # fit a CART model of cp 0
  model_CART_k_fold <- rpart(SalePrice ~ GrLivArea + YearBuilt + BedroomAbvGr + SaleCondit

  # predict sales price on test validation using fitted/trained model on train validation
  test_validation_k_fold$SalePrice_hat <- predict(model_CART_k_fold, newdata = test_valida

  # calculate rmsle score for the ith fold and add it to the rmsle_score vector
  rmsle_score[i] <- sqrt((sum((log(test_validation_k_fold$SalePrice_hat + 1) - log(test_va
}

# find average among k rmsle scores
avg_rmsle <- mean(rmsle_score)


print(avg_rmsle)
```

```
[1] 0.197646
```

The estimated rmsle score of 0.198 is better than the test_validation score of 0.202 when cp = 0.