Biostat 212a Homework 4

Due Mar. 4, 2025 @ 11:59PM

Wenqiang Ge UID:106371961

2025-03-02

Table of contents


```r
# Load necessary libraries
library(ggplot2)
library(dplyr)
library(rpart)
library(rpart.plot)
library(GGally)
library(gtsummary)
library(ranger)
library(tidyverse)
library(tidymodels)
library(ISLR2)
library(MASS)
library(randomForest)
library(gbm)
library(caret)
library(Metrics)
library(doParallel)
library(future)
library(vip)
library(xgboost)
```

ISL Exercise 8.4.3 (10pts)

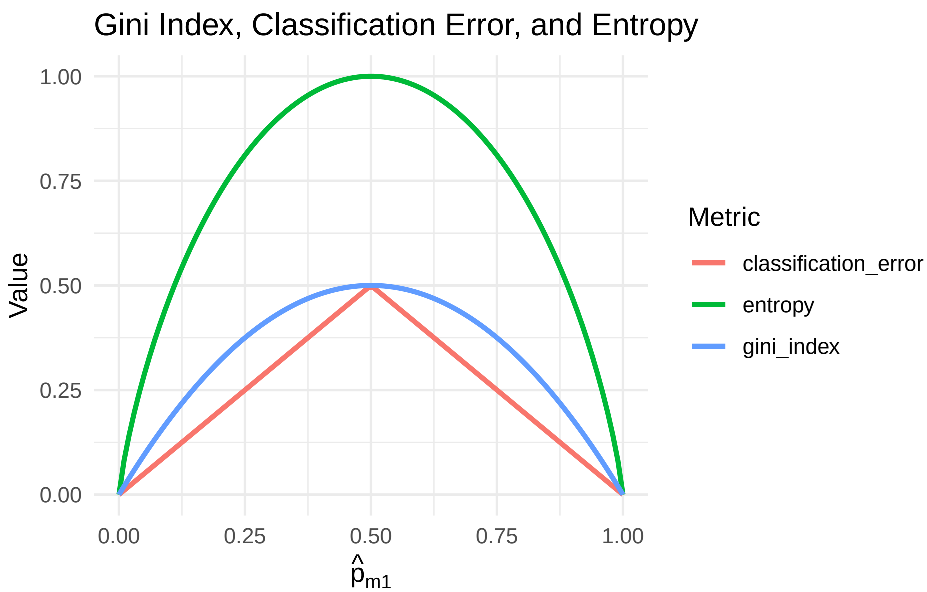
3. Consider the Gini index, classification error, and entropy in a simple classification setting with two classes. Create a single plot that displays each of these quantities as a function of $\hat{p}_{m1}$. The x-axis should display $\hat{p}_{m1}$, ranging from 0 to 1, and the y-axis should display the value of the Gini index, classification error, and entropy.

   Hint: In a setting with two classes, $\hat{p}_{m1} = 1 - \hat{p}_{m2}$. You could make this plot by hand, but it will be much easier to make in R.

Solution:

```r
# Define probability range for class 1
p_m1 <- seq(0, 1, length.out = 100)  # Probability values from 0 to 1
p_m2 <- 1 - p_m1  # Probability of the second class

# Compute the three metrics
gini_index <- 1 - (p_m1^2 + p_m2^2)  # Gini index formula
classification_error <- pmin(p_m1, p_m2)  # Classification error (minimum probability)
entropy <- - (p_m1 * log2(p_m1) + p_m2 * log2(p_m2))  # Entropy formula
entropy[is.na(entropy)] <- 0  # Handle log(0) cases (replace NaN with 0)

# Create a dataframe with all values
df <- data.frame(p_m1, gini_index, classification_error, entropy) %>%
  tidyr::pivot_longer(cols = -p_m1, names_to = "Metric", values_to = "Value")

# Plot the metrics as a function of p_m1
ggplot(df, aes(x = p_m1, y = Value, color = Metric)) +
  geom_line(size = 1) +  # Add lines for each metric
  labs(title = "Gini Index, Classification Error, and Entropy",
       x = expression(hat(p)[m1]), y = "Value") +
  theme_minimal()  # Use a clean theme for better visualization
```

```
Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
i Please use `linewidth` instead.
```



ISL Exercise 8.4.4 (10pts)

364 8. Tree-Based Methods

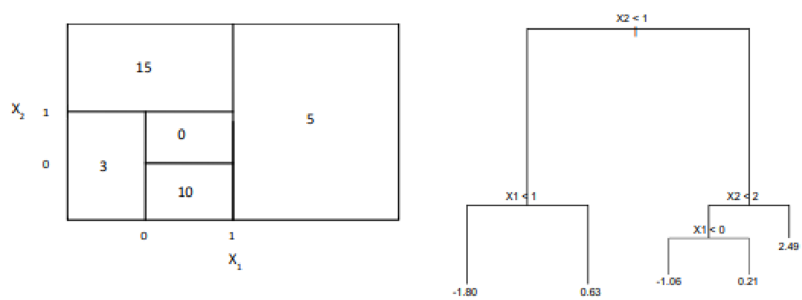
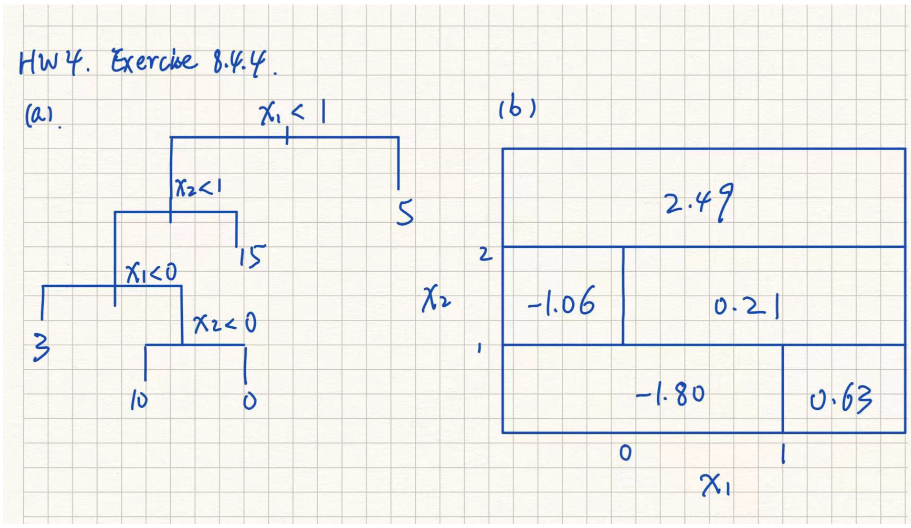


FIGURE 8.14. Left*: A partition of the predictor space corresponding to Exercise 4a.* Right*: A tree corresponding to Exercise 4b.*

(a) Sketch the tree corresponding to the partition of the predictor space illustrated in the left-hand panel of Figure 8.14. The numbers inside the boxes indicate the mean of Y within each region.

(b) Create a diagram similar to the left-hand panel of Figure 8.14, using the tree illustrated in the right-hand panel of the same figure. You should divide up the predictor space into the correct regions, and indicate the mean for each region.

Solution:



ISL Exercise 8.4.5 (10pts)

5. Suppose we produce ten bootstrapped samples from a data set containing red and green classes. We then apply a classification tree to each bootstrapped sample and, for a specific value of X, produce 10 estimates of $P(\text{Class is Red}|X)$:

$$0.1, 0.15, 0.2, 0.2, 0.55, 0.6, 0.6, 0.65, 0.7, \text{ and } 0.75.$$

There are two common ways to combine these results together into a single class prediction. One is the majority vote approach discussed in this chapter. The second approach is to classify based on the average probability. In this example, what is the final classification under each of these two approaches?

Solution:

Given probabilities of $P(Class\ is\ Red \mid X)$: 0.1,0.15,0.2,0.2,0.55,0.6,0.6,0.65,0.7,0.75

Majority Vote Approach: Each estimate can be converted into a binary decision by using a threshold of 0.5 : If $P(Red \mid X) \geq 0.5$, classify as Red. If $P(Red \mid X) < 0.5$,classify as Green.

Now, applying this threshold: Green $P < 0.5$: 4 times.

Red $P >= 0.5$: 6 times.

Since Red occurs more often, the majority vote approach classifies Red.

Average Probability Approach:

$\frac{0.1+0.15+0.2+0.2+0.55+0.6+0.6+0.65+0.7+0.75}{10} = 0.45$
Since $0.45 < 0.5$, the final classification under the average probability approach is Green.

Majority vote: Red

Average probability: Green

ISL Lab 8.3. `Boston` data set (30pts)

Follow the machine learning workflow to train regression tree, random forest, and boosting methods for predicting `medv`. Evaluate out-of-sample performance on a test set.

Solution:

```
Boston %>% tbl_summary()
```

```
Boston <- Boston %>% filter(!is.na(medv))
```

Regression tree

Initial split into test and non-test sets

```
# For reproducibility
set.seed(203)

data_split <- initial_split(
  Boston,
  prop = 0.5
  )
data_split
```

```
<Training/Testing/Total>
<253/253/506>
```

```
Boston_other <- training(data_split)
dim(Boston_other)
```

```
[1] 253  14
```

Characteristic	**N = 506**[1]
crim	0.3 (0.1, 3.7)
zn	0 (0, 13)
indus	9.7 (5.2, 18.1)
chas	35 (6.9%)
nox	0.54 (0.45, 0.62)
rm	6.21 (5.89, 6.63)
age	78 (45, 94)
dis	3.21 (2.10, 5.21)
rad	
1	20 (4.0%)
2	24 (4.7%)
3	38 (7.5%)
4	110 (22%)
5	115 (23%)
6	26 (5.1%)
7	17 (3.4%)
8	24 (4.7%)
24	132 (26%)
tax	330 (279, 666)
ptratio	19.05 (17.40, 20.20)
black	391 (375, 396)
lstat	11 (7, 17)
medv	21 (17, 25)

[1]Median (Q1, Q3); n (%)

```r
Boston_test <- testing(data_split)
dim(Boston_test)
```

```
[1] 253  14
```

Recipe (R)

```r
# Define an untrained recipe
tree_recipe <- recipe(medv ~ ., data = Boston) %>%
  step_naomit(all_predictors()) %>%
  step_dummy(all_nominal_predictors()) %>%
  step_zv(all_numeric_predictors()) %>%
  step_normalize(all_numeric_predictors())

tree_recipe
```

Model

```r
#Model
regtree_mod <- decision_tree(
  cost_complexity = tune(),
  tree_depth = tune(),
  min_n = 5,
  mode = "regression",
  engine = "rpart"
  )
```

Workflow

```r
#Workflow
tree_wf <- workflow() %>%
  add_recipe(tree_recipe) %>%
  add_model(regtree_mod)
tree_wf
```

```
== Workflow ====================================================================
Preprocessor: Recipe
Model: decision_tree()

-- Preprocessor ----------------------------------------------------------------
4 Recipe Steps

* step_naomit()
```

```
* step_dummy()
* step_zv()
* step_normalize()

-- Model -----------------------------------------------------------------------
Decision Tree Model Specification (regression)

Main Arguments:
  cost_complexity = tune()
  tree_depth = tune()
  min_n = 5

Computational engine: rpart
```

Tuning grid

```r
#Tuning
tree_grid <- grid_regular(cost_complexity(),
                          tree_depth(),
                          levels = c(100, 5))
```

Cross-validation

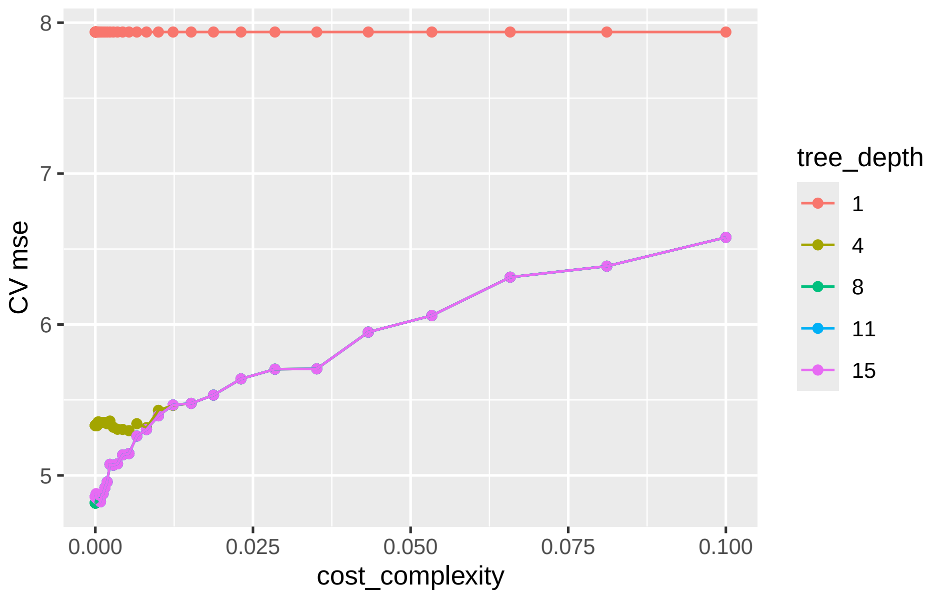
```r
#Cross-validation
set.seed(203)

folds <- vfold_cv(Boston_other, v = 5)
#Fit cross-validation
tree_fit <- tree_wf %>%
  tune_grid(
    resamples = folds,
    grid = tree_grid,
    metrics = metric_set(yardstick::rmse, yardstick::rsq)
    )
tree_fit
```

```
# Tuning results
# 5-fold cross-validation
# A tibble: 5 x 4
  splits           id    .metrics             .notes
  <list>           <chr> <list>               <list>
1 <split [202/51]> Fold1 <tibble [1,000 x 6]> <tibble [0 x 3]>
2 <split [202/51]> Fold2 <tibble [1,000 x 6]> <tibble [0 x 3]>
3 <split [202/51]> Fold3 <tibble [1,000 x 6]> <tibble [0 x 3]>
4 <split [203/50]> Fold4 <tibble [1,000 x 6]> <tibble [0 x 3]>
```

```
5 <split [203/50]> Fold5 <tibble [1,000 x 6]> <tibble [0 x 3]>
```

```
#Visualize CV results
tree_fit %>%
  collect_metrics() %>%
  print(width = Inf) %>%
  filter(.metric == "rmse") %>%
  mutate(tree_depth = as.factor(tree_depth)) %>%
  ggplot(mapping = aes(x = cost_complexity, y = mean, color = tree_depth)) +
  geom_point() +
  geom_line() +
  labs(x = "cost_complexity", y = "CV mse")
```

```
# A tibble: 1,000 x 8
   cost_complexity tree_depth .metric .estimator  mean     n std_err
             <dbl>      <int> <chr>   <chr>      <dbl> <int>   <dbl>
 1        1   e-10          1 rmse    standard   7.94      5  0.442
 2        1   e-10          1 rsq     standard   0.347     5  0.0740
 3        1.23e-10          1 rmse    standard   7.94      5  0.442
 4        1.23e-10          1 rsq     standard   0.347     5  0.0740
 5        1.52e-10          1 rmse    standard   7.94      5  0.442
 6        1.52e-10          1 rsq     standard   0.347     5  0.0740
 7        1.87e-10          1 rmse    standard   7.94      5  0.442
 8        1.87e-10          1 rsq     standard   0.347     5  0.0740
 9        2.31e-10          1 rmse    standard   7.94      5  0.442
10        2.31e-10          1 rsq     standard   0.347     5  0.0740
   .config
   <chr>
 1 Preprocessor1_Model001
 2 Preprocessor1_Model001
 3 Preprocessor1_Model002
 4 Preprocessor1_Model002
 5 Preprocessor1_Model003
 6 Preprocessor1_Model003
 7 Preprocessor1_Model004
 8 Preprocessor1_Model004
 9 Preprocessor1_Model005
10 Preprocessor1_Model005
# i 990 more rows
```



Finalize the model

```
tree_fit %>%
  show_best(metric = "rmse", n = 5)
```

```
# A tibble: 5 x 8
  cost_complexity tree_depth .metric .estimator  mean     n std_err .config
            <dbl>      <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
1        1   e-10          8 rmse    standard    4.82     5   0.726 Preprocesso~
2        1.23e-10          8 rmse    standard    4.82     5   0.726 Preprocesso~
3        1.52e-10          8 rmse    standard    4.82     5   0.726 Preprocesso~
4        1.87e-10          8 rmse    standard    4.82     5   0.726 Preprocesso~
5        2.31e-10          8 rmse    standard    4.82     5   0.726 Preprocesso~
```

```
best_tree <- tree_fit %>%
  select_best(metric = "rmse")
best_tree
```

```
# A tibble: 1 x 3
  cost_complexity tree_depth .config
            <dbl>      <int> <chr>
1    0.0000000001          8 Preprocessor1_Model201
```

```r
# Final workflow
final_wf <- tree_wf %>%
  finalize_workflow(best_tree)
final_wf
```

```
== Workflow ===================================================================
Preprocessor: Recipe
Model: decision_tree()

-- Preprocessor ---------------------------------------------------------------
4 Recipe Steps

* step_naomit()
* step_dummy()
* step_zv()
* step_normalize()

-- Model ----------------------------------------------------------------------
Decision Tree Model Specification (regression)

Main Arguments:
  cost_complexity = 1e-10
  tree_depth = 8
  min_n = 5

Computational engine: rpart
```

```r
# Fit the whole training set, then predict the test cases
final_fit <-
  final_wf %>%
  last_fit(data_split)
final_fit
```

```
# Resampling results
# Manual resampling
# A tibble: 1 x 6
  splits            id               .metrics .notes   .predictions .workflow
  <list>            <chr>            <list>   <list>   <list>       <list>
1 <split [253/253]> train/test split <tibble> <tibble> <tibble>     <workflow>
```

```r
# Test metrics
final_fit %>%
  collect_metrics()
```

```
# A tibble: 2 x 4
```

```
  .metric .estimator .estimate .config
  <chr>   <chr>          <dbl> <chr>
1 rmse    standard       4.97  Preprocessor1_Model1
2 rsq     standard       0.716 Preprocessor1_Model1
```

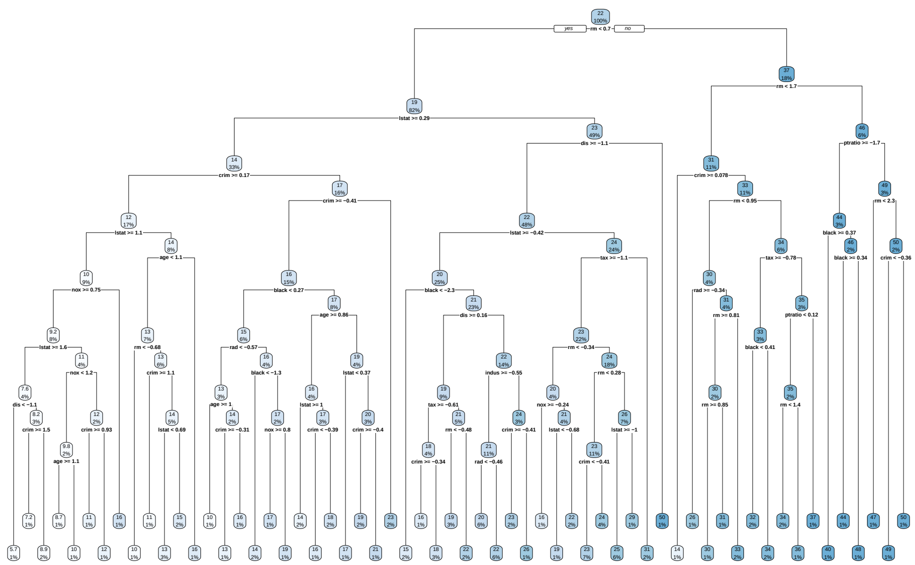
Visualize the final model

```r
final_tree <- extract_workflow(final_fit)
final_tree
```

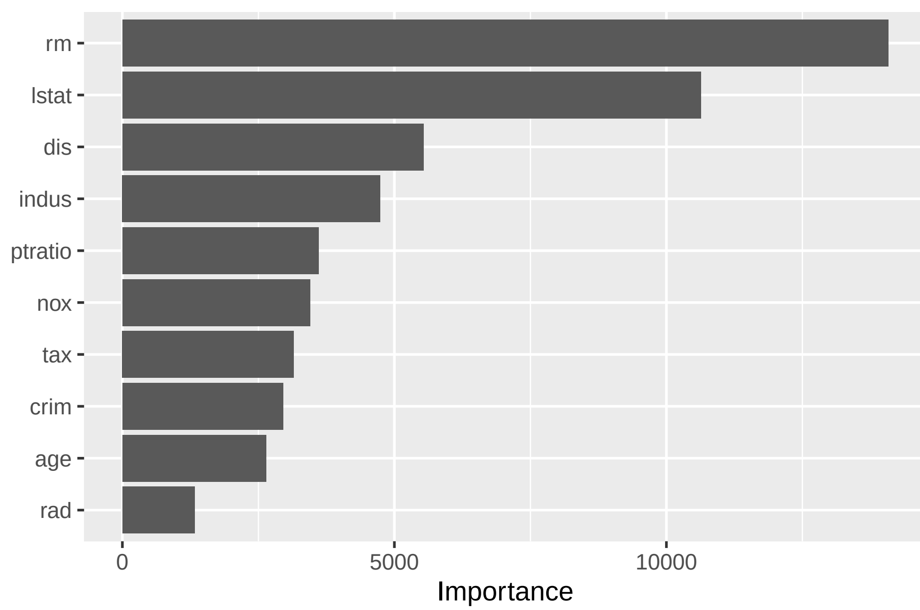
```
== Workflow [trained] ================================================================
Preprocessor: Recipe
Model: decision_tree()

-- Preprocessor ----------------------------------------------------------------------
4 Recipe Steps

* step_naomit()
* step_dummy()
* step_zv()
* step_normalize()

-- Model -----------------------------------------------------------------------------
n= 253

node), split, n, deviance, yval
      * denotes terminal node

  1) root 253 2.337441e+04 22.451380
    2) rm< 0.697884 208 8.878604e+03 19.383650
      4) lstat>=0.2888357 83 1.457504e+03 14.308430
        8) crim>=0.16539 42 4.284764e+02 11.764290
         16) lstat>=1.065022 22 2.211150e+02 10.150000
           32) nox>=0.7459829 19 8.364000e+01  9.200000
             64) lstat>=1.639685 9 1.810889e+01  7.611111
              128) dis< -1.083781 2 8.450000e-01  5.650000 *
              129) dis>=-1.083781 7 7.374286e+00  8.171429
                258) crim>=1.486993 3 8.000000e-02  7.200000 *
                259) crim< 1.486993 4 2.340000e+00  8.900000 *
             65) lstat< 1.639685 10 2.236100e+01 10.630000
              130) nox< 1.203607 5 9.652000e+00  9.760000
                260) age>=1.120657 2 4.500000e+00  8.700000 *
                261) age< 1.120657 3 1.406667e+00 10.466670 *
              131) nox>=1.203607 5 5.140000e+00 11.500000
                262) crim>=0.9254287 2 4.500000e-02 10.650000 *
```

```
          263) crim< 0.9254287 3 2.686667e+00 12.066670 *
        33) nox< 0.7459829 3 1.172667e+01 16.166670 *
      17) lstat< 1.065022 20 8.696800e+01 13.540000
        34) age< 1.117024 17 6.234471e+01 13.082350
          68) rm< -0.683543 2 5.120000e+00 10.100000 *
          69) rm>=-0.683543 15 3.706400e+01 13.480000
           138) crim>=1.111666 2 2.205000e+00 10.650000 *
           139) crim< 1.111666 13 1.637692e+01 13.915380
             278) lstat< 0.6861861 8 2.820000e+00 13.450000 *
             279) lstat>=0.6861861 5 9.052000e+00 14.660000 *
        35) age>=1.117024 3 8.866667e-01 16.133330 *
     9) crim< 0.16539 41 4.786912e+02 16.914630
      18) crim>=-0.4060389 37 3.045276e+02 16.291890
        36) black< 0.2746026 16 1.295544e+02 14.731250
          72) rad< -0.5746709 7 5.816000e+01 13.100000
           144) age>=1.048007 2 2.178000e+01 10.300000 *
           145) age< 1.048007 5 1.442800e+01 14.220000
             290) crim>=-0.3052468 3 1.400000e-01 13.000000 *
             291) crim< -0.3052468 2 3.125000e+00 16.050000 *
          73) rad>=-0.5746709 9 3.828000e+01 16.000000
           146) black< -1.317335 4 7.370000e+00 14.250000 *
           147) black>=-1.317335 5 8.860000e+00 17.400000
             294) nox>=0.7968301 3 2.906667e+00 16.533330 *
             295) nox< 0.7968301 2 3.200000e-01 18.700000 *
        37) black>=0.2746026 21 1.063124e+02 17.480950
          74) age>=0.8609327 11 3.508727e+01 15.954550
           148) lstat>=1.036538 4 1.347500e+00 13.925000 *
           149) lstat< 1.036538 7 7.848571e+00 17.114290

...
and 74 more lines.
```

```r
final_tree %>%
  extract_fit_engine() %>%
  rpart.plot(roundint = FALSE)
```



```r
final_tree %>%
  extract_fit_parsnip() %>%
  vip()
```



Random forest

```
#Recipe
rf_recipe <-
  recipe(
    medv ~ .,
    data = Boston_other
  ) %>%
  step_naomit(medv) %>%
  step_zv(all_numeric_predictors())
rf_recipe

#Model
rf_mod <-
  rand_forest(
    mode = "regression",
    mtry = tune(),
    trees = tune()
  ) %>%
  set_engine("ranger")
rf_mod
```

```
Random Forest Model Specification (regression)

Main Arguments:
  mtry = tune()
  trees = tune()

Computational engine: ranger
```

```
#Workflow
rf_wf <- workflow() %>%
  add_recipe(rf_recipe) %>%
  add_model(rf_mod)
rf_wf
```

```
== Workflow ====================================================================
Preprocessor: Recipe
Model: rand_forest()

-- Preprocessor ----------------------------------------------------------------
2 Recipe Steps

* step_naomit()
* step_zv()
```

```
-- Model -----------------------------------------------------------------------
Random Forest Model Specification (regression)

Main Arguments:
  mtry = tune()
  trees = tune()

Computational engine: ranger
```

```r
#Tuning
param_grid <- grid_regular(
  trees(range = c(100L, 300L)),
  mtry(range = c(1L, 5L)),
  levels = c(3, 5)
  )
param_grid
```

```
# A tibble: 15 x 2
   trees  mtry
   <int> <int>
 1   100     1
 2   200     1
 3   300     1
 4   100     2
 5   200     2
 6   300     2
 7   100     3
 8   200     3
 9   300     3
10   100     4
11   200     4
12   300     4
13   100     5
14   200     5
15   300     5
```

```r
#Cross-validation
set.seed(203)

folds <- vfold_cv(Boston_other, v = 5)
folds
```

```
#  5-fold cross-validation
# A tibble: 5 x 2
```

```
  splits           id   
  <list>           <chr>
1 <split [202/51]> Fold1
2 <split [202/51]> Fold2
3 <split [202/51]> Fold3
4 <split [203/50]> Fold4
5 <split [203/50]> Fold5
```

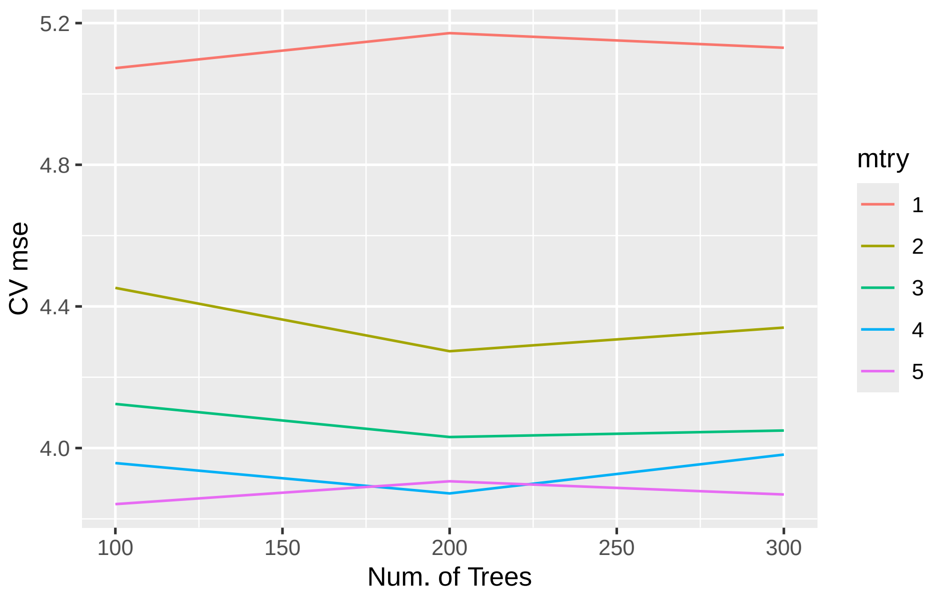
```
rf_fit <- rf_wf %>%
  tune_grid(
    resamples = folds,
    grid = param_grid,
    metrics = metric_set(yardstick::rmse, yardstick::rsq)
    )
rf_fit
```

```
# Tuning results
# 5-fold cross-validation
# A tibble: 5 x 4
  splits           id    .metrics          .notes          
  <list>           <chr> <list>            <list>          
1 <split [202/51]> Fold1 <tibble [30 x 6]> <tibble [0 x 3]>
2 <split [202/51]> Fold2 <tibble [30 x 6]> <tibble [0 x 3]>
3 <split [202/51]> Fold3 <tibble [30 x 6]> <tibble [0 x 3]>
4 <split [203/50]> Fold4 <tibble [30 x 6]> <tibble [0 x 3]>
5 <split [203/50]> Fold5 <tibble [30 x 6]> <tibble [0 x 3]>
```

```
rf_fit %>%
  collect_metrics() %>%
  print(width = Inf) %>%
  filter(.metric == "rmse") %>%
  mutate(mtry = as.factor(mtry)) %>%
  ggplot(mapping = aes(x = trees, y = mean, color = mtry)) +
  # geom_point() +
  geom_line() +
  labs(x = "Num. of Trees", y = "CV mse")
```

```
# A tibble: 30 x 8
    mtry trees .metric .estimator  mean     n std_err .config             
   <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>               
 1     1   100 rmse    standard   5.07      5  0.417  Preprocessor1_Model01
 2     1   100 rsq     standard   0.749     5  0.0838 Preprocessor1_Model01
 3     1   200 rmse    standard   5.17      5  0.417  Preprocessor1_Model02
 4     1   200 rsq     standard   0.741     5  0.0863 Preprocessor1_Model02
 5     1   300 rmse    standard   5.13      5  0.465  Preprocessor1_Model03
```

```
 6     1   300 rsq     standard   0.741     5  0.0904 Preprocessor1_Model03
 7     2   100 rmse    standard   4.45      5  0.637  Preprocessor1_Model04
 8     2   100 rsq     standard   0.778     5  0.0988 Preprocessor1_Model04
 9     2   200 rmse    standard   4.27      5  0.564  Preprocessor1_Model05
10     2   200 rsq     standard   0.796     5  0.0859 Preprocessor1_Model05
# i 20 more rows
```




```
rf_fit %>%
  show_best(metric = "rmse")
```

```
# A tibble: 5 x 8
   mtry trees .metric .estimator  mean     n std_err .config
  <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
1     5   100 rmse    standard    3.84     5   0.612 Preprocessor1_Model13
2     5   300 rmse    standard    3.87     5   0.598 Preprocessor1_Model15
3     4   200 rmse    standard    3.87     5   0.590 Preprocessor1_Model11
4     5   200 rmse    standard    3.91     5   0.593 Preprocessor1_Model14
5     4   100 rmse    standard    3.96     5   0.602 Preprocessor1_Model10
```

```
best_rf <- rf_fit %>%
  select_best(metric = "rmse")
best_rf
```

```
# A tibble: 1 x 3
   mtry trees .config
  <int> <int> <chr>
```

```
1     5   100 Preprocessor1_Model13
```

```r
# Final workflow
final_wf <- rf_wf %>%
  finalize_workflow(best_rf)
final_wf
```

```
== Workflow ====================================================================
Preprocessor: Recipe
Model: rand_forest()

-- Preprocessor ----------------------------------------------------------------
2 Recipe Steps

* step_naomit()
* step_zv()

-- Model -----------------------------------------------------------------------
Random Forest Model Specification (regression)

Main Arguments:
  mtry = 5
  trees = 100

Computational engine: ranger
```

```r
# Fit the whole training set, then predict the test cases
final_fit <-
  final_wf %>%
  last_fit(data_split)
final_fit
```

```
# Resampling results
# Manual resampling
# A tibble: 1 x 6
  splits            id               .metrics .notes   .predictions .workflow
  <list>            <chr>            <list>   <list>   <list>       <list>
1 <split [253/253]> train/test split <tibble> <tibble> <tibble>     <workflow>
```

```r
# Test metrics
final_fit %>%
  collect_metrics()
```

```
# A tibble: 2 x 4
  .metric .estimator .estimate .config
```

```
  <chr>   <chr>          <dbl> <chr>
1 rmse    standard       3.18  Preprocessor1_Model1
2 rsq     standard       0.869 Preprocessor1_Model1
```

Boosting methods

Recipe (R)

```
#Recipe
gb_recipe <-
  recipe(
    medv ~ .,
    data = Boston_other
  ) %>%
  step_naomit(medv) %>%
  step_zv(all_numeric_predictors())
gb_recipe
```

Model

```
#Model
gb_mod <-
  boost_tree(
    mode = "regression",
    trees = 1000,
    tree_depth = tune(),
    learn_rate = tune()
  ) %>%
  set_engine("xgboost")
gb_mod
```

```
Boosted Tree Model Specification (regression)

Main Arguments:
  trees = 1000
  tree_depth = tune()
  learn_rate = tune()

Computational engine: xgboost
```

Workflow & Tuning

```
#Workflow
gb_wf <- workflow() %>%
```

```r
  add_recipe(gb_recipe) %>%
  add_model(gb_mod)
gb_wf
```

```
== Workflow ====================================================================
Preprocessor: Recipe
Model: boost_tree()

-- Preprocessor ----------------------------------------------------------------
2 Recipe Steps

* step_naomit()
* step_zv()

-- Model -----------------------------------------------------------------------
Boosted Tree Model Specification (regression)

Main Arguments:
  trees = 1000
  tree_depth = tune()
  learn_rate = tune()

Computational engine: xgboost
```

```r
#Tuning
param_grid <- grid_regular(
  tree_depth(range = c(1L, 4L)),
  learn_rate(range = c(-3, -0.5), trans = log10_trans()),
  levels = c(4, 10)
  )
param_grid
```

```
# A tibble: 40 x 2
   tree_depth learn_rate
        <int>      <dbl>
 1          1    0.001
 2          2    0.001
 3          3    0.001
 4          4    0.001
 5          1    0.00190
 6          2    0.00190
 7          3    0.00190
 8          4    0.00190
 9          1    0.00359
10          2    0.00359
```

```
# i 30 more rows
```

Cross-validation

```r
#Cross-validation
set.seed(203)

folds <- vfold_cv(Boston_other, v = 5)
folds
```

```
#  5-fold cross-validation
# A tibble: 5 x 2
  splits           id
  <list>           <chr>
1 <split [202/51]> Fold1
2 <split [202/51]> Fold2
3 <split [202/51]> Fold3
4 <split [203/50]> Fold4
5 <split [203/50]> Fold5
```

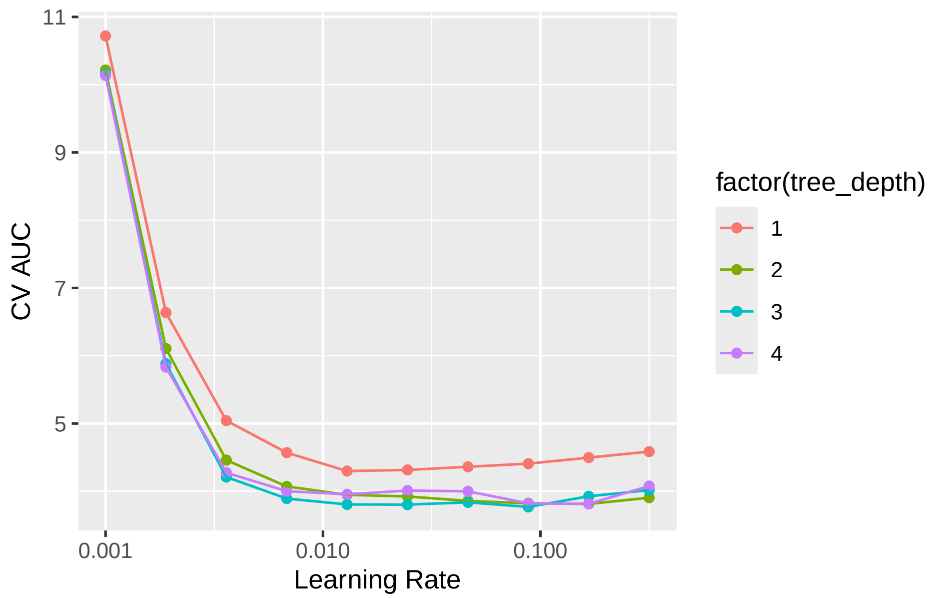
```r
gb_fit <- gb_wf %>%
  tune_grid(
    resamples = folds,
    grid = param_grid,
    metrics = metric_set(yardstick::rmse, yardstick::rsq)
    )
gb_fit
```

```
# Tuning results
# 5-fold cross-validation
# A tibble: 5 x 4
  splits           id    .metrics          .notes
  <list>           <chr> <list>            <list>
1 <split [202/51]> Fold1 <tibble [80 x 6]> <tibble [0 x 3]>
2 <split [202/51]> Fold2 <tibble [80 x 6]> <tibble [0 x 3]>
3 <split [202/51]> Fold3 <tibble [80 x 6]> <tibble [0 x 3]>
4 <split [203/50]> Fold4 <tibble [80 x 6]> <tibble [0 x 3]>
5 <split [203/50]> Fold5 <tibble [80 x 6]> <tibble [0 x 3]>
```

```r
gb_fit %>%
  collect_metrics() %>%
  print(width = Inf) %>%
  filter(.metric == "rmse") %>%
  ggplot(mapping = aes(x = learn_rate, y = mean, color = factor(tree_depth))) +
```

```r
  geom_point() +
  geom_line() +
  labs(x = "Learning Rate", y = "CV AUC") +
  scale_x_log10()
```

```
# A tibble: 80 x 8
   tree_depth learn_rate .metric .estimator   mean     n std_err
        <int>      <dbl> <chr>   <chr>       <dbl> <int>   <dbl>
 1          1    0.001   rmse    standard   10.7       5  0.392
 2          1    0.001   rsq     standard    0.714     5  0.0598
 3          2    0.001   rmse    standard   10.2       5  0.445
 4          2    0.001   rsq     standard    0.750     5  0.0861
 5          3    0.001   rmse    standard   10.2       5  0.434
 6          3    0.001   rsq     standard    0.765     5  0.0837
 7          4    0.001   rmse    standard   10.1       5  0.418
 8          4    0.001   rsq     standard    0.769     5  0.0820
 9          1    0.00190 rmse    standard    6.64      5  0.400
10          1    0.00190 rsq     standard    0.752     5  0.0732
   .config
   <chr>
 1 Preprocessor1_Model01
 2 Preprocessor1_Model01
 3 Preprocessor1_Model02
 4 Preprocessor1_Model02
 5 Preprocessor1_Model03
 6 Preprocessor1_Model03
 7 Preprocessor1_Model04
 8 Preprocessor1_Model04
 9 Preprocessor1_Model05
10 Preprocessor1_Model05
# i 70 more rows
```




```
gb_fit %>%
  show_best(metric = "rmse")
```

```
# A tibble: 5 x 8
  tree_depth learn_rate .metric .estimator  mean     n std_err .config
       <int>      <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
1          3     0.0880 rmse    standard    3.77     5   0.382 Preprocessor1_Mo~
2          3     0.0245 rmse    standard    3.80     5   0.422 Preprocessor1_Mo~
3          3     0.0129 rmse    standard    3.81     5   0.458 Preprocessor1_Mo~
4          2     0.167  rmse    standard    3.81     5   0.351 Preprocessor1_Mo~
5          4     0.167  rmse    standard    3.81     5   0.393 Preprocessor1_Mo~
```

```
best_gb <- gb_fit %>%
  select_best(metric = "rmse")
best_gb
```

```
# A tibble: 1 x 3
  tree_depth learn_rate .config
       <int>      <dbl> <chr>
1          3     0.0880 Preprocessor1_Model31
```

Finalize the model

```r
#Final model
final_wf <- gb_wf %>%
  finalize_workflow(best_gb)
final_wf
```

```
== Workflow ====================================================================
Preprocessor: Recipe
Model: boost_tree()

-- Preprocessor ----------------------------------------------------------------
2 Recipe Steps

* step_naomit()
* step_zv()

-- Model -----------------------------------------------------------------------
Boosted Tree Model Specification (regression)

Main Arguments:
  trees = 1000
  tree_depth = 3
  learn_rate = 0.0879922543569107

Computational engine: xgboost
```

```r
final_fit <-
  final_wf %>%
  last_fit(data_split)
final_fit
```

```
# Resampling results
# Manual resampling
# A tibble: 1 x 6
  splits            id               .metrics .notes   .predictions .workflow
  <list>            <chr>            <list>   <list>   <list>       <list>
1 <split [253/253]> train/test split <tibble> <tibble> <tibble>     <workflow>
```

```r
final_fit %>%
  collect_metrics()
```

```
# A tibble: 2 x 4
  .metric .estimator .estimate .config
  <chr>   <chr>          <dbl> <chr>
1 rmse    standard       3.63  Preprocessor1_Model1
2 rsq     standard       0.836 Preprocessor1_Model1
```

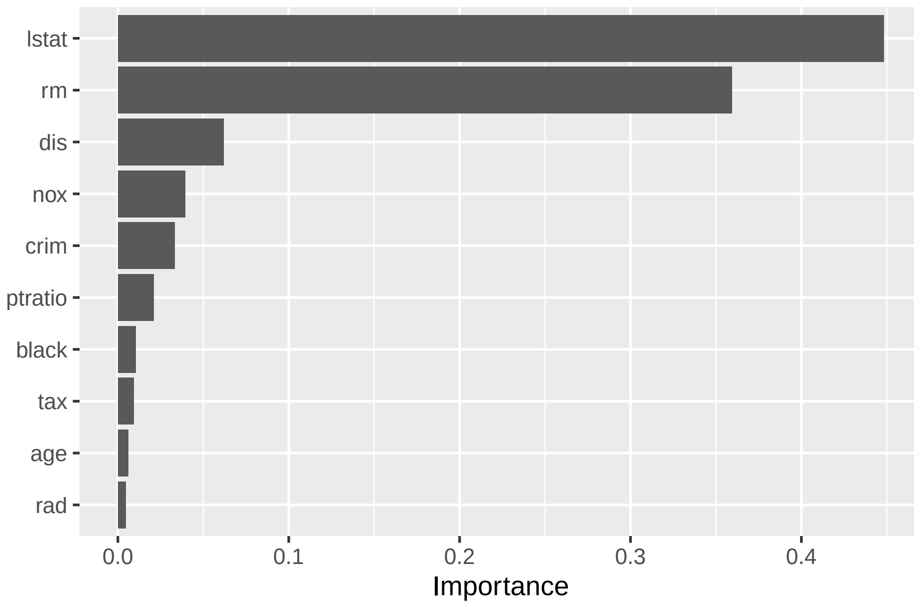
Visualize the final model

```r
#Visualize the final model
final_tree <- extract_workflow(final_fit)
final_tree
```

```
== Workflow [trained] ==========================================================
Preprocessor: Recipe
Model: boost_tree()

-- Preprocessor ----------------------------------------------------------------
2 Recipe Steps

* step_naomit()
* step_zv()

-- Model -----------------------------------------------------------------------
##### xgb.Booster
raw: 1.1 Mb
call:
  xgboost::xgb.train(params = list(eta = 0.0879922543569107, max_depth = 3L,
    gamma = 0, colsample_bytree = 1, colsample_bynode = 1, min_child_weight = 1,
    subsample = 1), data = x$data, nrounds = 1000, watchlist = x$watchlist,
    verbose = 0, nthread = 1, objective = "reg:squarederror")
params (as set within xgb.train):
  eta = "0.0879922543569107", max_depth = "3", gamma = "0", colsample_bytree = "1", colsampl
xgb.attributes:
  niter
callbacks:
  cb.evaluation.log()
# of features: 13
niter: 1000
nfeatures : 13
evaluation_log:
  iter training_rmse
 <num>         <num>
     1   21.98661584
     2   20.17570809
   ---           ---
   999    0.03930784
  1000    0.03925896
```

```r
final_tree %>%
  extract_fit_parsnip() %>%
```

```r
vip()
```



Conclusion

Considering the values of RMSE of each model, the random forest model has the lowest one ,so we can choose it as the final model.

ISL Lab 8.3 `Carseats` data set (30pts)

Follow the machine learning workflow to train classification tree, random forest, and boosting methods for classifying `Sales <= 8` versus `Sales > 8`. Evaluate out-of-sample performance on a test set.

Solution:

```r
# load the data
data("Carseats", package = "ISLR")
Carseats$AHD <- ifelse(Carseats$Sales > 8, "High", "Low")
Carseats$AHD <- as.factor(Carseats$AHD)
Carseats <- Carseats[, !names(Carseats) %in% c("Sales")]
Carseats %>% tbl_summary()
```

Characteristic	**N = 400**[1]
CompPrice	125 (115, 135)
Income	69 (43, 91)
Advertising	5.0 (0.0, 12.0)
Population	272 (139, 399)
Price	117 (100, 131)
ShelveLoc	
Bad	96 (24%)
Good	85 (21%)
Medium	219 (55%)
Age	55 (40, 66)
Education	
10	48 (12%)
11	48 (12%)
12	49 (12%)
13	43 (11%)
14	40 (10%)
15	36 (9.0%)
16	47 (12%)
17	49 (12%)
18	40 (10%)
Urban	282 (71%)
US	258 (65%)
AHD	
High	164 (41%)
Low	236 (59%)

[1]Median (Q1, Q3); n (%)

Classification tree

Initial split into test and non-test sets

```
#Initial split into test and non-test sets
set.seed(212)
data_split <- initial_split(
  Carseats,
  prop = 0.5,
  strata = AHD
  )
data_split
```

```
<Training/Testing/Total>
<200/200/400>
```

```
Carseats_other <- training(data_split)
dim(Carseats_other)
```

```
[1] 200  11
```

```
Carseats_test <- testing(data_split)
dim(Carseats_test)
```

```
[1] 200  11
```

Recipe

```
#Recipe
tree_recipe <-
  recipe(
    AHD ~ .,
    data = Carseats_other
  ) %>%
  step_naomit(all_predictors()) %>%
  # create traditional dummy variables (not necessary for random forest in R)
  step_dummy(all_nominal_predictors()) %>%
  # zero-variance filter
  step_zv(all_numeric_predictors()) %>%
  # # center and scale numeric data (not necessary for random forest)
  step_normalize(all_numeric_predictors())

tree_recipe
```

Model & Workflow

```r
#Model
classtree_mod <- decision_tree(
   # Hyperparameter: Complexity parameter (cp) for pruning
  cost_complexity = tune(),
  # Hyperparameter: Maximum depth of the tree
  tree_depth = tune(),
  min_n = 5,
  mode = "classification",
  engine = "rpart"
  )

#Workflow
tree_wf <- workflow() %>%
  add_recipe(tree_recipe) %>%
  add_model(classtree_mod)

# Print the workflow structure
tree_wf
```

```
== Workflow ====================================================================
Preprocessor: Recipe
Model: decision_tree()

-- Preprocessor ----------------------------------------------------------------
4 Recipe Steps

* step_naomit()
* step_dummy()
* step_zv()
* step_normalize()

-- Model -----------------------------------------------------------------------
Decision Tree Model Specification (classification)

Main Arguments:
  cost_complexity = tune()
  tree_depth = tune()
  min_n = 5

Computational engine: rpart
```

Tuning grid

```r
#Tuning
tree_grid <- grid_regular(cost_complexity(),
                          tree_depth(),
                          levels = c(100,5))
```

Cross-validation (CV)

```r
set.seed(212)

folds <- vfold_cv(Carseats_other, v = 5)
folds
```

```
#  5-fold cross-validation
# A tibble: 5 x 2
  splits            id
  <list>            <chr>
1 <split [160/40]> Fold1
2 <split [160/40]> Fold2
3 <split [160/40]> Fold3
4 <split [160/40]> Fold4
5 <split [160/40]> Fold5
```

```r
# Register a parallel backend using future
plan(multisession, workers = parallel::detectCores() - 1)

# Fit cross-validation.
tree_fit <- tree_wf %>%
  tune_grid(
    resamples = folds,
    grid = tree_grid,
    metrics = metric_set(yardstick::accuracy, yardstick::roc_auc),
    control = control_grid(save_pred = TRUE, parallel_over = "resamples")
  )

# Stop parallel processing after computation
plan(sequential)  # Reset to sequential processing after tuning

tree_fit
```

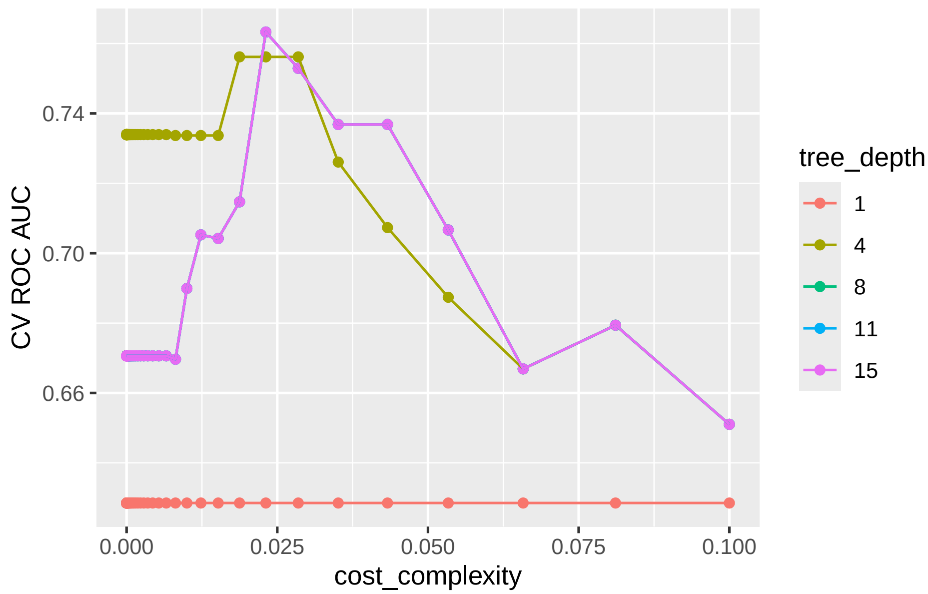
```
# Tuning results
# 5-fold cross-validation
# A tibble: 5 x 5
  splits            id    .metrics             .notes           .predictions
  <list>            <chr> <list>               <list>           <list>
```

```
1 <split [160/40]> Fold1 <tibble [1,000 x 6]> <tibble [0 x 3]> <tibble>
2 <split [160/40]> Fold2 <tibble [1,000 x 6]> <tibble [0 x 3]> <tibble>
3 <split [160/40]> Fold3 <tibble [1,000 x 6]> <tibble [0 x 3]> <tibble>
4 <split [160/40]> Fold4 <tibble [1,000 x 6]> <tibble [0 x 3]> <tibble>
5 <split [160/40]> Fold5 <tibble [1,000 x 6]> <tibble [0 x 3]> <tibble>
```

Visualize CV results

```r
tree_fit %>%
  collect_metrics() %>%
  print(width = Inf) %>%
  filter(.metric == "roc_auc") %>%
  mutate(tree_depth = as.factor(tree_depth)) %>%
  ggplot(mapping = aes(x = cost_complexity, y = mean, color = tree_depth)) +
  geom_point() +
  geom_line() +
  labs(x = "cost_complexity", y = "CV ROC AUC", color = "tree_depth")
```

```
# A tibble: 1,000 x 8
   cost_complexity tree_depth .metric  .estimator  mean     n std_err
             <dbl>      <int> <chr>    <chr>      <dbl> <int>   <dbl>
 1        1   e-10          1 accuracy binary     0.685     5  0.0232
 2        1   e-10          1 roc_auc  binary     0.629     5  0.0298
 3        1.23e-10          1 accuracy binary     0.685     5  0.0232
 4        1.23e-10          1 roc_auc  binary     0.629     5  0.0298
 5        1.52e-10          1 accuracy binary     0.685     5  0.0232
 6        1.52e-10          1 roc_auc  binary     0.629     5  0.0298
 7        1.87e-10          1 accuracy binary     0.685     5  0.0232
 8        1.87e-10          1 roc_auc  binary     0.629     5  0.0298
 9        2.31e-10          1 accuracy binary     0.685     5  0.0232
10        2.31e-10          1 roc_auc  binary     0.629     5  0.0298
   .config
   <chr>
 1 Preprocessor1_Model001
 2 Preprocessor1_Model001
 3 Preprocessor1_Model002
 4 Preprocessor1_Model002
 5 Preprocessor1_Model003
 6 Preprocessor1_Model003
 7 Preprocessor1_Model004
 8 Preprocessor1_Model004
 9 Preprocessor1_Model005
10 Preprocessor1_Model005
# i 990 more rows
```



Finalize the model

```
tree_fit %>%
  show_best(metric = "roc_auc", n = 5)
```

```
# A tibble: 5 x 8
  cost_complexity tree_depth .metric .estimator  mean     n std_err .config
            <dbl>      <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
1          0.0231          8 roc_auc binary     0.763     5  0.0148 Preprocesso~
2          0.0231         11 roc_auc binary     0.763     5  0.0148 Preprocesso~
3          0.0231         15 roc_auc binary     0.763     5  0.0148 Preprocesso~
4          0.0187          4 roc_auc binary     0.756     5  0.0165 Preprocesso~
5          0.0231          4 roc_auc binary     0.756     5  0.0165 Preprocesso~
```

```
# select the best model.
best_tree <- tree_fit %>%
  select_best(metric = "roc_auc")
best_tree
```

```
# A tibble: 1 x 3
  cost_complexity tree_depth .config
            <dbl>      <int> <chr>
1          0.0231          8 Preprocessor1_Model293
```

```r
# Final workflow
final_wf <- tree_wf %>%
  finalize_workflow(best_tree)
final_wf
```

```
== Workflow ================================================================
Preprocessor: Recipe
Model: decision_tree()

-- Preprocessor ------------------------------------------------------------
4 Recipe Steps

* step_naomit()
* step_dummy()
* step_zv()
* step_normalize()

-- Model -------------------------------------------------------------------
Decision Tree Model Specification (classification)

Main Arguments:
  cost_complexity = 0.0231012970008316
  tree_depth = 8
  min_n = 5

Computational engine: rpart
```

Visualize the final model

```r
# Fit the whole training set, then predict the test cases
final_fit <-
  final_wf %>%
  last_fit(data_split)
final_fit
```

```
# Resampling results
# Manual resampling
# A tibble: 1 x 6
  splits            id               .metrics .notes   .predictions .workflow
  <list>            <chr>            <list>   <list>   <list>       <list>
1 <split [200/200]> train/test split <tibble> <tibble> <tibble>     <workflow>
```

```r
# Test metrics
final_fit %>%
```

```r
  collect_metrics()
```

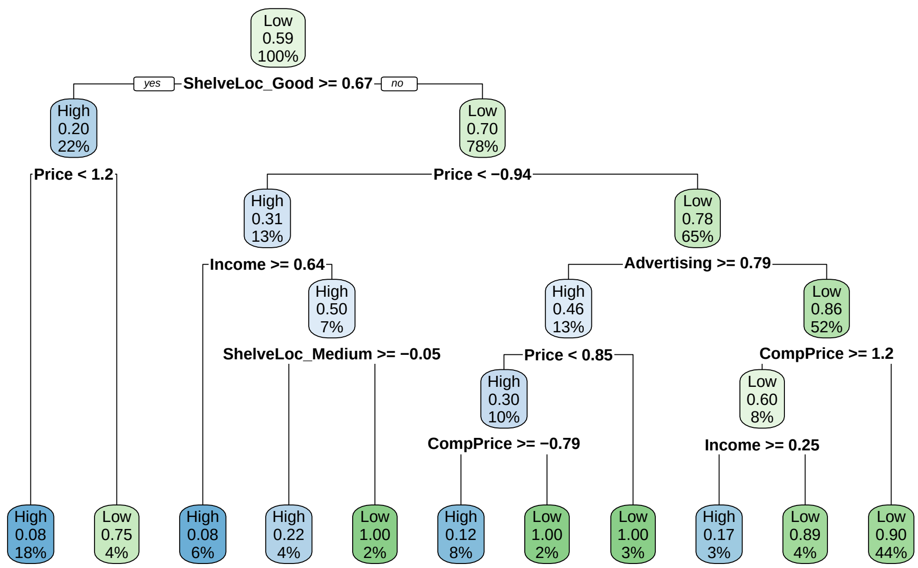
```
# A tibble: 3 x 4
  .metric     .estimator .estimate .config
  <chr>       <chr>          <dbl> <chr>
1 accuracy    binary         0.755 Preprocessor1_Model1
2 roc_auc     binary         0.760 Preprocessor1_Model1
3 brier_class binary         0.208 Preprocessor1_Model1
```

```r
final_tree <- extract_workflow(final_fit)
final_tree
```

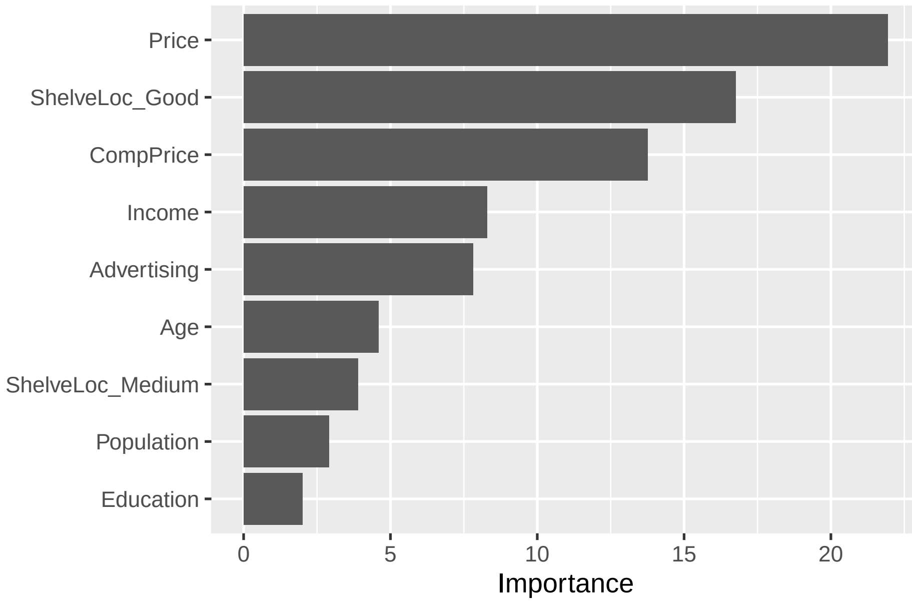
```
== Workflow [trained] ==========================================================
Preprocessor: Recipe
Model: decision_tree()

-- Preprocessor ----------------------------------------------------------------
4 Recipe Steps

* step_naomit()
* step_dummy()
* step_zv()
* step_normalize()

-- Model -----------------------------------------------------------------------
n= 200

node), split, n, loss, yval, (yprob)
      * denotes terminal node

 1) root 200 82 Low (0.41000000 0.59000000)
   2) ShelveLoc_Good>=0.6742344 44  9 High (0.79545455 0.20454545)
     4) Price< 1.204802 36  3 High (0.91666667 0.08333333) *
     5) Price>=1.204802 8  2 Low (0.25000000 0.75000000) *
   3) ShelveLoc_Good< 0.6742344 156 47 Low (0.30128205 0.69871795)
     6) Price< -0.9414209 26  8 High (0.69230769 0.30769231)
      12) Income>=0.6401903 12  1 High (0.91666667 0.08333333) *
      13) Income< 0.6401903 14  7 High (0.50000000 0.50000000)
        26) ShelveLoc_Medium>=-0.0499373 9  2 High (0.77777778 0.22222222) *
        27) ShelveLoc_Medium< -0.0499373 5  0 Low (0.00000000 1.00000000) *
     7) Price>=-0.9414209 130 29 Low (0.22307692 0.77692308)
      14) Advertising>=0.78991 26 12 High (0.53846154 0.46153846)
        28) Price< 0.8505711 20  6 High (0.70000000 0.30000000)
          56) CompPrice>=-0.787038 16  2 High (0.87500000 0.12500000) *
          57) CompPrice< -0.787038 4  0 Low (0.00000000 1.00000000) *
```

```
      29) Price>=0.8505711 6   0 Low (0.00000000 1.00000000) *
    15) Advertising< 0.78991 104 15 Low (0.14423077 0.85576923)
      30) CompPrice>=1.238547 15   6 Low (0.40000000 0.60000000)
        60) Income>=0.247091 6   1 High (0.83333333 0.16666667) *
        61) Income< 0.247091 9   1 Low (0.11111111 0.88888889) *
      31) CompPrice< 1.238547 89   9 Low (0.10112360 0.89887640) *
```

```r
final_tree %>%
  extract_fit_engine() %>%
  rpart.plot(roundint = FALSE)
```




```r
final_tree %>%
  extract_fit_parsnip() %>%
  vip()
```



Random forest

Initial split into test and non-test sets

```
#Initial split into test and non-test sets
set.seed(212)

data_split <- initial_split(
  Carseats,
  prop = 0.75,
  strata = AHD
  )
data_split
```

```
<Training/Testing/Total>
<300/100/400>
```

```
Carseats_other <- training(data_split)
dim(Carseats_other)
```

```
[1] 300  11
```

```
Carseats_test <- testing(data_split)
dim(Carseats_test)
```

```
[1] 100  11
```

Recipe (R)

```
#Recipe
rf_recipe <-
  recipe(
    AHD ~ .,
    data = Carseats_other
  ) %>%
  step_zv(all_numeric_predictors())
rf_recipe
```

Model

```
#Model
rf_mod <-
  rand_forest(
    mode = "classification",
    mtry = tune(),
    trees = tune()
  ) %>%
  set_engine("ranger")
rf_mod
```

```
Random Forest Model Specification (classification)

Main Arguments:
  mtry = tune()
  trees = tune()

Computational engine: ranger
```

Work & Tuning

```
#Workflow
rf_wf <- workflow() %>%
  add_recipe(rf_recipe) %>%
  add_model(rf_mod)
rf_wf
```

```
== Workflow ====================================================================
Preprocessor: Recipe
Model: rand_forest()
```

```
-- Preprocessor ----------------------------------------------------------------
1 Recipe Step

* step_zv()

-- Model -----------------------------------------------------------------------
Random Forest Model Specification (classification)

Main Arguments:
  mtry = tune()
  trees = tune()

Computational engine: ranger
```

```r
#Tuning
param_grid <- grid_regular(
  trees(range = c(100L, 300L)),
  mtry(range = c(1L, 5L)),
  levels = c(3, 5)
  )
param_grid
```

```
# A tibble: 15 x 2
   trees  mtry
   <int> <int>
 1   100     1
 2   200     1
 3   300     1
 4   100     2
 5   200     2
 6   300     2
 7   100     3
 8   200     3
 9   300     3
10   100     4
11   200     4
12   300     4
13   100     5
14   200     5
15   300     5
```

Cross-validation (CV)

```r
#Cross-validation
set.seed(203)

folds <- vfold_cv(Carseats_other, v = 5)
folds
```

```
#  5-fold cross-validation
# A tibble: 5 x 2
  splits           id
  <list>           <chr>
1 <split [240/60]> Fold1
2 <split [240/60]> Fold2
3 <split [240/60]> Fold3
4 <split [240/60]> Fold4
5 <split [240/60]> Fold5
```

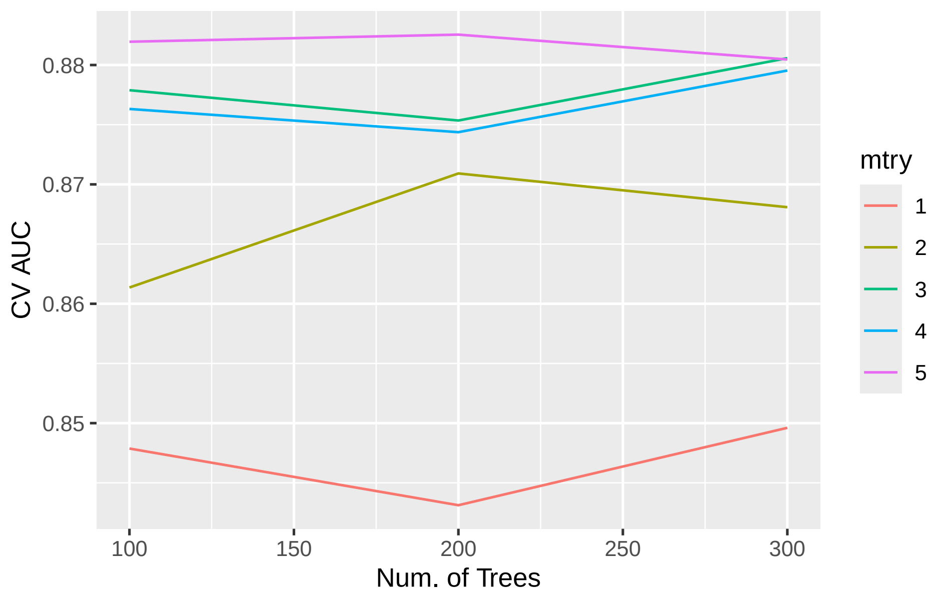
```r
#Fit cross-validation
rf_fit <- rf_wf %>%
  tune_grid(
    resamples = folds,
    grid = param_grid,
    metrics = metric_set(yardstick::roc_auc,
                         yardstick::accuracy)
    )
rf_fit
```

```
# Tuning results
# 5-fold cross-validation
# A tibble: 5 x 4
  splits           id    .metrics          .notes
  <list>           <chr> <list>            <list>
1 <split [240/60]> Fold1 <tibble [30 x 6]> <tibble [0 x 3]>
2 <split [240/60]> Fold2 <tibble [30 x 6]> <tibble [0 x 3]>
3 <split [240/60]> Fold3 <tibble [30 x 6]> <tibble [0 x 3]>
4 <split [240/60]> Fold4 <tibble [30 x 6]> <tibble [0 x 3]>
5 <split [240/60]> Fold5 <tibble [30 x 6]> <tibble [0 x 3]>
```

```r
#Visualize CV results
rf_fit %>%
  collect_metrics() %>%
  print(width = Inf) %>%
  filter(.metric == "roc_auc") %>%
  mutate(mtry = as.factor(mtry)) %>%
  ggplot(mapping = aes(x = trees, y = mean, color = mtry)) +
```

```
  # geom_point() +
  geom_line() +
  labs(x = "Num. of Trees", y = "CV AUC")
```

```
# A tibble: 30 x 8
    mtry trees .metric  .estimator  mean     n std_err .config
   <int> <int> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
 1     1   100 accuracy binary     0.753     5  0.0207 Preprocessor1_Model01
 2     1   100 roc_auc  binary     0.848     5  0.0214 Preprocessor1_Model01
 3     1   200 accuracy binary     0.747     5  0.0276 Preprocessor1_Model02
 4     1   200 roc_auc  binary     0.843     5  0.0297 Preprocessor1_Model02
 5     1   300 accuracy binary     0.76      5  0.0327 Preprocessor1_Model03
 6     1   300 roc_auc  binary     0.850     5  0.0283 Preprocessor1_Model03
 7     2   100 accuracy binary     0.767     5  0.0497 Preprocessor1_Model04
 8     2   100 roc_auc  binary     0.861     5  0.0324 Preprocessor1_Model04
 9     2   200 accuracy binary     0.777     5  0.0327 Preprocessor1_Model05
10     2   200 roc_auc  binary     0.871     5  0.0274 Preprocessor1_Model05
# i 20 more rows
```




```
#Show the top 5 models.
rf_fit %>%
  show_best(metric = "roc_auc")
```

```
# A tibble: 5 x 8
   mtry trees .metric .estimator  mean     n std_err .config
```

```
  <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
1     5   200 roc_auc binary     0.883     5  0.0264 Preprocessor1_Model14
2     5   100 roc_auc binary     0.882     5  0.0317 Preprocessor1_Model13
3     3   300 roc_auc binary     0.881     5  0.0306 Preprocessor1_Model09
4     5   300 roc_auc binary     0.880     5  0.0322 Preprocessor1_Model15
5     4   300 roc_auc binary     0.880     5  0.0284 Preprocessor1_Model12
```

```
#Select the best model
best_rf <- rf_fit %>%
  select_best(metric = "roc_auc")
best_rf
```

```
# A tibble: 1 x 3
   mtry trees .config
  <int> <int> <chr>
1     5   200 Preprocessor1_Model14
```

Finalize the model

```
#Final model
final_wf <- rf_wf %>%
  finalize_workflow(best_rf)
final_wf
```

```
== Workflow ====================================================================
Preprocessor: Recipe
Model: rand_forest()

-- Preprocessor ----------------------------------------------------------------
1 Recipe Step

* step_zv()

-- Model -----------------------------------------------------------------------
Random Forest Model Specification (classification)

Main Arguments:
  mtry = 5
  trees = 200

Computational engine: ranger
```

```
final_fit <-
  final_wf %>%
```

```r
  last_fit(data_split)
final_fit
```

```
# Resampling results
# Manual resampling
# A tibble: 1 x 6
  splits            id               .metrics .notes   .predictions .workflow
  <list>            <chr>            <list>   <list>   <list>       <list>
1 <split [300/100]> train/test split <tibble> <tibble> <tibble>     <workflow>
```

```r
final_fit %>%
  collect_metrics()
```

```
# A tibble: 3 x 4
  .metric     .estimator .estimate .config
  <chr>       <chr>          <dbl> <chr>
1 accuracy    binary         0.81  Preprocessor1_Model1
2 roc_auc     binary         0.895 Preprocessor1_Model1
3 brier_class binary         0.140 Preprocessor1_Model1
```

Boosting methods

Initial split into test and non-test sets

```r
library(xgboost)
#Initial split into test and non-test sets
set.seed(212)

data_split <- initial_split(
  Carseats,
  prop = 0.75,
  strata = AHD
  )
data_split
```

```
<Training/Testing/Total>
<300/100/400>
```

```r
Carseats_other <- training(data_split)
dim(Carseats_other)
```

```
[1] 300  11
```

```
Carseats_test <- testing(data_split)
dim(Carseats_test)
```

```
[1] 100  11
```

Recipe (R)

```
#Recipe
gb_recipe <-
  recipe(
    AHD ~ .,
    data = Carseats_other
  ) %>%
  step_dummy(all_nominal_predictors()) %>%
  step_zv(all_numeric_predictors())
gb_recipe
```

Model

```
#Model
gb_mod <-
  boost_tree(
    mode = "classification",
    trees = 1000,
    tree_depth = tune(),
    learn_rate = tune()
  ) %>%
  set_engine("xgboost")
gb_mod
```

```
Boosted Tree Model Specification (classification)

Main Arguments:
  trees = 1000
  tree_depth = tune()
  learn_rate = tune()

Computational engine: xgboost
```

Workflow & Tuning

```
#Workflow
gb_wf <- workflow() %>%
```

```
  add_recipe(gb_recipe) %>%
  add_model(gb_mod)
gb_wf
```

```
== Workflow ====================================================================
Preprocessor: Recipe
Model: boost_tree()

-- Preprocessor ----------------------------------------------------------------
2 Recipe Steps

* step_dummy()
* step_zv()

-- Model -----------------------------------------------------------------------
Boosted Tree Model Specification (classification)

Main Arguments:
  trees = 1000
  tree_depth = tune()
  learn_rate = tune()

Computational engine: xgboost
```

```
#Tuning
param_grid <- grid_regular(
  tree_depth(range = c(1L, 3L)),
  learn_rate(range = c(-5, 2), trans = log10_trans()),
  levels = c(3, 10)
  )
param_grid
```

```
# A tibble: 30 x 2
   tree_depth learn_rate
        <int>      <dbl>
 1          1  0.00001
 2          2  0.00001
 3          3  0.00001
 4          1  0.0000599
 5          2  0.0000599
 6          3  0.0000599
 7          1  0.000359
 8          2  0.000359
 9          3  0.000359
10          1  0.00215
```

```
# i 20 more rows
```

Cross-validation

```r
#Cross-validation
set.seed(203)

folds <- vfold_cv(Carseats_other, v = 5)
folds
```

```
#  5-fold cross-validation
# A tibble: 5 x 2
  splits           id
  <list>           <chr>
1 <split [240/60]> Fold1
2 <split [240/60]> Fold2
3 <split [240/60]> Fold3
4 <split [240/60]> Fold4
5 <split [240/60]> Fold5
```

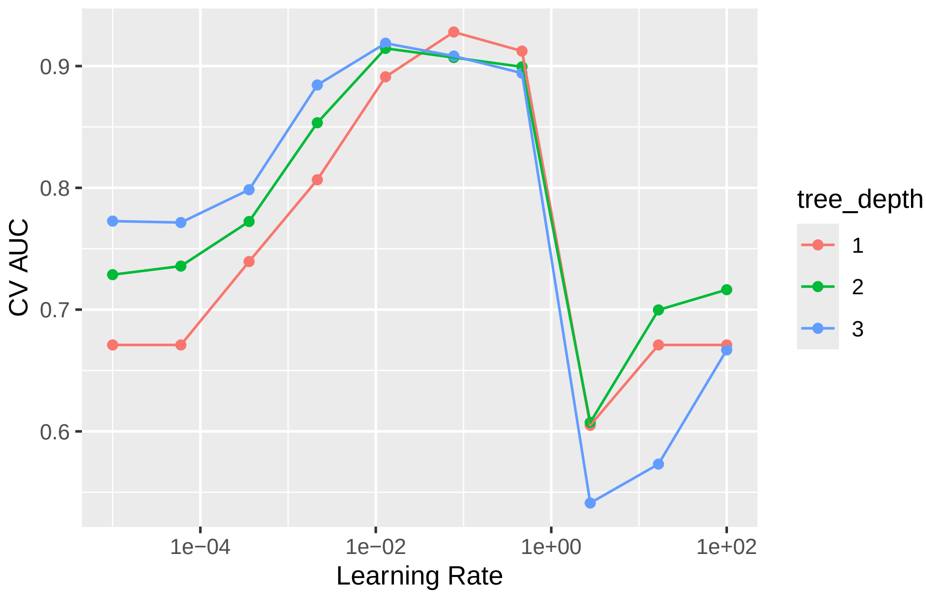
```r
gb_fit <- gb_wf %>%
  tune_grid(
    resamples = folds,
    grid = param_grid,
    metrics = metric_set(yardstick::roc_auc,
                         yardstick::accuracy)
    )
gb_fit
```

```
# Tuning results
# 5-fold cross-validation
# A tibble: 5 x 4
  splits           id    .metrics          .notes
  <list>           <chr> <list>            <list>
1 <split [240/60]> Fold1 <tibble [60 x 6]> <tibble [0 x 3]>
2 <split [240/60]> Fold2 <tibble [60 x 6]> <tibble [0 x 3]>
3 <split [240/60]> Fold3 <tibble [60 x 6]> <tibble [0 x 3]>
4 <split [240/60]> Fold4 <tibble [60 x 6]> <tibble [0 x 3]>
5 <split [240/60]> Fold5 <tibble [60 x 6]> <tibble [0 x 3]>
```

```r
gb_fit %>%
  collect_metrics() %>%
  print(width = Inf) %>%
  filter(.metric == "roc_auc") %>%
```

```r
  mutate(tree_depth = as.factor(tree_depth)) %>%
  ggplot(mapping = aes(x = learn_rate, y = mean, color = tree_depth)) +
  geom_point() +
  geom_line() +
  labs(x = "Learning Rate", y = "CV AUC") +
  scale_x_log10()
```

```
# A tibble: 60 x 8
   tree_depth learn_rate .metric  .estimator  mean     n std_err
        <int>      <dbl> <chr>    <chr>      <dbl> <int>   <dbl>
 1          1  0.00001   accuracy binary     0.717     5  0.0183
 2          1  0.00001   roc_auc  binary     0.671     5  0.0181
 3          2  0.00001   accuracy binary     0.713     5  0.0309
 4          2  0.00001   roc_auc  binary     0.729     5  0.0203
 5          3  0.00001   accuracy binary     0.737     5  0.0295
 6          3  0.00001   roc_auc  binary     0.773     5  0.0258
 7          1  0.0000599 accuracy binary     0.717     5  0.0183
 8          1  0.0000599 roc_auc  binary     0.671     5  0.0181
 9          2  0.0000599 accuracy binary     0.743     5  0.0310
10          2  0.0000599 roc_auc  binary     0.736     5  0.0187
   .config
   <chr>
 1 Preprocessor1_Model01
 2 Preprocessor1_Model01
 3 Preprocessor1_Model02
 4 Preprocessor1_Model02
 5 Preprocessor1_Model03
 6 Preprocessor1_Model03
 7 Preprocessor1_Model04
 8 Preprocessor1_Model04
 9 Preprocessor1_Model05
10 Preprocessor1_Model05
# i 50 more rows
```




```
gb_fit %>%
  show_best(metric = "roc_auc")
```

```
# A tibble: 5 x 8
  tree_depth learn_rate .metric .estimator  mean     n std_err .config
       <int>      <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
1          1     0.0774 roc_auc binary     0.928     5  0.0138 Preprocessor1_Mo~
2          3     0.0129 roc_auc binary     0.919     5  0.0161 Preprocessor1_Mo~
3          2     0.0129 roc_auc binary     0.915     5  0.0173 Preprocessor1_Mo~
4          1     0.464  roc_auc binary     0.912     5  0.0191 Preprocessor1_Mo~
5          3     0.0774 roc_auc binary     0.908     5  0.0182 Preprocessor1_Mo~
```

```
#select the best model
best_gb <- gb_fit %>%
  select_best(metric = "roc_auc")
best_gb
```

```
# A tibble: 1 x 3
  tree_depth learn_rate .config
       <int>      <dbl> <chr>
1          1     0.0774 Preprocessor1_Model16
```

Finalize the model

```r
#Final model
final_wf <- gb_wf %>%
  finalize_workflow(best_gb)
final_wf
```

```
== Workflow ====================================================================
Preprocessor: Recipe
Model: boost_tree()

-- Preprocessor ----------------------------------------------------------------
2 Recipe Steps

* step_dummy()
* step_zv()

-- Model -----------------------------------------------------------------------
Boosted Tree Model Specification (classification)

Main Arguments:
  trees = 1000
  tree_depth = 1
  learn_rate = 0.0774263682681127

Computational engine: xgboost
```

```r
final_fit <-
  final_wf %>%
  last_fit(data_split)
final_fit
```

```
# Resampling results
# Manual resampling
# A tibble: 1 x 6
  splits            id               .metrics .notes   .predictions .workflow
  <list>            <chr>            <list>   <list>   <list>       <list>
1 <split [300/100]> train/test split <tibble> <tibble> <tibble>     <workflow>
```

```r
final_fit %>%
  collect_metrics()
```

```
# A tibble: 3 x 4
  .metric     .estimator .estimate .config
  <chr>       <chr>          <dbl> <chr>
1 accuracy    binary         0.82  Preprocessor1_Model1
2 roc_auc     binary         0.907 Preprocessor1_Model1
```

```
3 brier_class binary         0.127 Preprocessor1_Model1
```

Conclusion

We choose the boosting method as the final model for classifying Sales in the Carseats data set, because it has the highest accuracy and roc_auc.