MATH 154 - HW9 - bagging & Random Forests

Ra-Zakee Muhammad

due: Thursday, November 11, 2021

Amber helped me with 4c, 4d, 4e, 4f ### summary This assignment extends ideas about classification and regression trees. First bagging is used to improve the variance of trees. Random Forests are an extension of bagging using a prediction which is an average over many trees which have been built on a subset of predictor variables.

As you do the assignment, pay attention to the steps and how well the model does (on the training data) at each step. The point of this homework is to help differentiate between the different tree based analyses we've been covering.

requisites

Read relevant sections of *An Introduction to Statistical Learning*, https://web.stanford.edu/~hastie/ISLR2/ISLRv2_website.pdf. bagging and Random Forests (section 8.2) [no boosting or BART].

assignment

1. **Pod Q** Describe one thing you learned from someone in your pod this week (it could be: content, logistical help, background material, R information, something fun, etc.) 1-3 sentences.

Sakeet Likes solving rubiks cubes.

2. For which of the following would you need to feature engineer for use in a tree or forest model? For any that need feature engineering, explain what steps are needed (either words or code is fine).

a. Explanatory / predictor variables with extremely skewed values

b. Response / outcome variable with extremely skewed values (e.g., think NBA salaries)

If the outcome variable has extremely skewed values then the resulting model may be inacurate for both the clustered values and the outlier values. Inorder to solve this problem the step_log would be useful in the recipe on the response variable.

c. A factor variable coded as numeric (e.g., think zip code) yes factorize

d. A factor variable with many missing values yes levelize

e. Binary factor variable coded as 0 / 1

f. Identifying numeric value (e.g., bar code associated with each individual penguin) yes id variable

3. Problem 7 in section 8.4 of *An Introduction to Statistical Learning.*

In the lab (see Section 8.3.3), we applied Random Forests to the Boston data (predict `medv` using the rest of the variables) using `mtry=6`; and using `trees=25` and `trees=500`. Create a plot displaying the test error resulting from Random Forests on this data set for a more comprehensive range of values for `mtry` and `ntree` (in the interest of time, you might try only `trees` in seq(1,401, by=50) and `mtry` in 1:10. Describe the results obtained. [n.b., as mentioned in class, we'll be using CV to train random forests, but other software might use OOB observations.]

The problem uses the `Boston` data which is available in the **MASS** package.

Hint 1: use the **tidymodels** syntax we've covered in class (slides, notes, etc.). Don't use the base R syntax in the text. Use `ranger` as the engine (`randomForest` will work, but it will be slower).

Hint 2: you'll probably want to set a seed early on (or often?) so that you can report on the output you need.

Hint 3: in the console, type `?Boston` to understand the dataset

```
library(MASS)
data(Boston)
```

```
set.seed(47)
Bos_split <- initial_split(Boston, prop = 0.75)
Bos_train <- training(Bos_split)
Bos_test <- testing(Bos_split)
```

```
Boston_rf_recipe <-
  recipe(medv ~ . ,
         data = Bos_train)
summary(Boston_rf_recipe)
```

```
## # A tibble: 14 x 4
##    variable type    role      source
##    <chr>    <chr>   <chr>     <chr>
##  1 crim     numeric predictor original
##  2 zn       numeric predictor original
##  3 indus    numeric predictor original
##  4 chas     numeric predictor original
##  5 nox      numeric predictor original
##  6 rm       numeric predictor original
##  7 age      numeric predictor original
##  8 dis      numeric predictor original
##  9 rad      numeric predictor original
## 10 tax      numeric predictor original
## 11 ptratio  numeric predictor original
## 12 black    numeric predictor original
## 13 lstat    numeric predictor original
## 14 medv     numeric outcome   original
```

```
Boston_rf <- rand_forest(mtry=tune(),
                         trees= tune()) %>%
  set_engine("ranger") %>%
  set_mode("regression")
Boston_rf
```

```
## Random Forest Model Specification (regression)
##
## Main Arguments:
##   mtry = tune()
##   trees = tune()
##
## Computational engine: ranger
```

```
Boston_rf_wflow <- workflow() %>%
  add_model(Boston_rf) %>%
  add_recipe(Boston_rf_recipe)
Boston_rf_wflow
```

```
## == Workflow ====================================================================
## Preprocessor: Recipe
## Model: rand_forest()
##
## -- Preprocessor ----------------------------------------------------------------
## 0 Recipe Steps
##
## -- Model -----------------------------------------------------------------------
## Random Forest Model Specification (regression)
##
## Main Arguments:
##   mtry = tune()
##   trees = tune()
##
## Computational engine: ranger
```

```
set.seed(234)
Boston_folds <- vfold_cv(Bos_train,
                         v = 4)
```

```
Boston_grid <- expand.grid(mtry = 1:10, trees = seq(1,401, by=50) )
Boston_grid
```

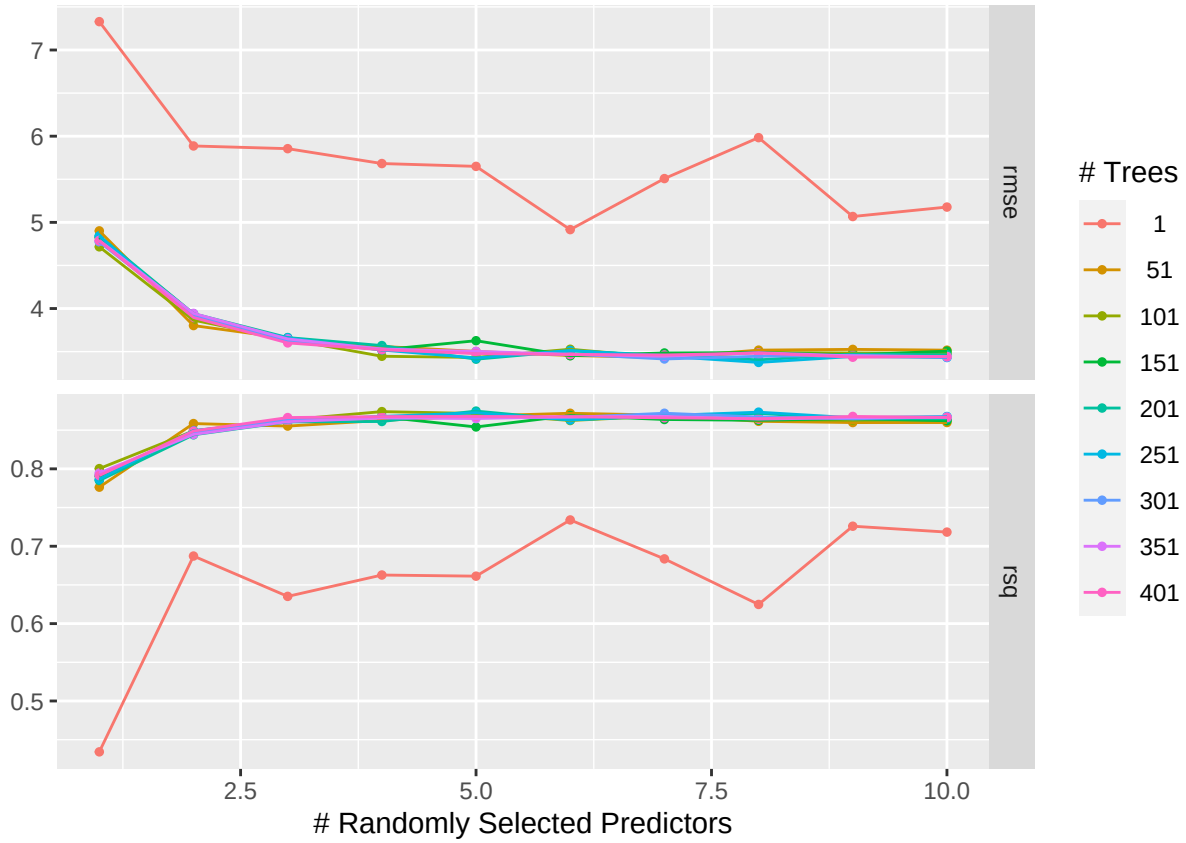
```
##    mtry trees
## 1     1     1
## 2     2     1
## 3     3     1
## 4     4     1
## 5     5     1
## 6     6     1
## 7     7     1
## 8     8     1
## 9     9     1
## 10   10     1
## 11    1    51
## 12    2    51
## 13    3    51
## 14    4    51
## 15    5    51
```

```
## 16    6    51
## 17    7    51
## 18    8    51
## 19    9    51
## 20   10    51
## 21    1   101
## 22    2   101
## 23    3   101
## 24    4   101
## 25    5   101
## 26    6   101
## 27    7   101
## 28    8   101
## 29    9   101
## 30   10   101
## 31    1   151
## 32    2   151
## 33    3   151
## 34    4   151
## 35    5   151
## 36    6   151
## 37    7   151
## 38    8   151
## 39    9   151
## 40   10   151
## 41    1   201
## 42    2   201
## 43    3   201
## 44    4   201
## 45    5   201
## 46    6   201
## 47    7   201
## 48    8   201
## 49    9   201
## 50   10   201
## 51    1   251
## 52    2   251
## 53    3   251
## 54    4   251
## 55    5   251
## 56    6   251
## 57    7   251
## 58    8   251
## 59    9   251
## 60   10   251
## 61    1   301
## 62    2   301
## 63    3   301
## 64    4   301
## 65    5   301
## 66    6   301
## 67    7   301
## 68    8   301
## 69    9   301
```

```
## 70   10   301
## 71    1   351
## 72    2   351
## 73    3   351
## 74    4   351
## 75    5   351
## 76    6   351
## 77    7   351
## 78    8   351
## 79    9   351
## 80   10   351
## 81    1   401
## 82    2   401
## 83    3   401
## 84    4   401
## 85    5   401
## 86    6   401
## 87    7   401
## 88    8   401
## 89    9   401
## 90   10   401
```

Code for the plot could look like this:

```
# tuning
Bos_rf_tuned <- Boston_rf_wflow %>%
  tune_grid(resamples = Boston_folds,
            grid = Boston_grid)

# plot
Bos_rf_tuned %>%
  autoplot()
```



The accuracy is higher when the number of randomly selected predictors (mtry) is higher. Also, the accuracy of the forest above 51 trees doesn't have much different, they all perform fairly well.

4. Problem 8 in section 8.4 of *An Introduction to Statistical Learning*.

In the lab (see section 8.3.1), a classification tree was applied to the `Carseats` data (`Carseats` is in the **ISLR** package) set after converting `Sales` into a qualitative response variable. Now we will seek to predict `Sales` using regression trees and related approaches, treating the response as a quantitative variable.

Hint: try using `?Carseats` to find out more information about the data set.

a. Split the data set into a training set and a test set.

```
set.seed(47)
car_split <- initial_split(Carseats, prop = 0.75)
car_train <- training(car_split)
car_test <- testing(car_split)
```

b. Fit a single regression tree to the training set. Plot the tree, and interpret the results. What **test** MSE do you obtain?

```
car_recipe <-
  recipe(Sales ~ . ,
         data = car_train)
summary(car_recipe)
```

```
## # A tibble: 11 x 4
##    variable    type    role      source
##    <chr>       <chr>   <chr>     <chr>
##  1 CompPrice   numeric predictor original
##  2 Income      numeric predictor original
##  3 Advertising numeric predictor original
##  4 Population  numeric predictor original
##  5 Price       numeric predictor original
##  6 ShelveLoc   nominal predictor original
##  7 Age         numeric predictor original
##  8 Education   numeric predictor original
##  9 Urban       nominal predictor original
## 10 US          nominal predictor original
## 11 Sales       numeric outcome   original
```

#model

```
car_rf <- decision_tree() %>%
  set_engine("rpart") %>%
  set_mode("regression")
car_rf
```

```
## Decision Tree Model Specification (regression)
##
## Computational engine: rpart
```

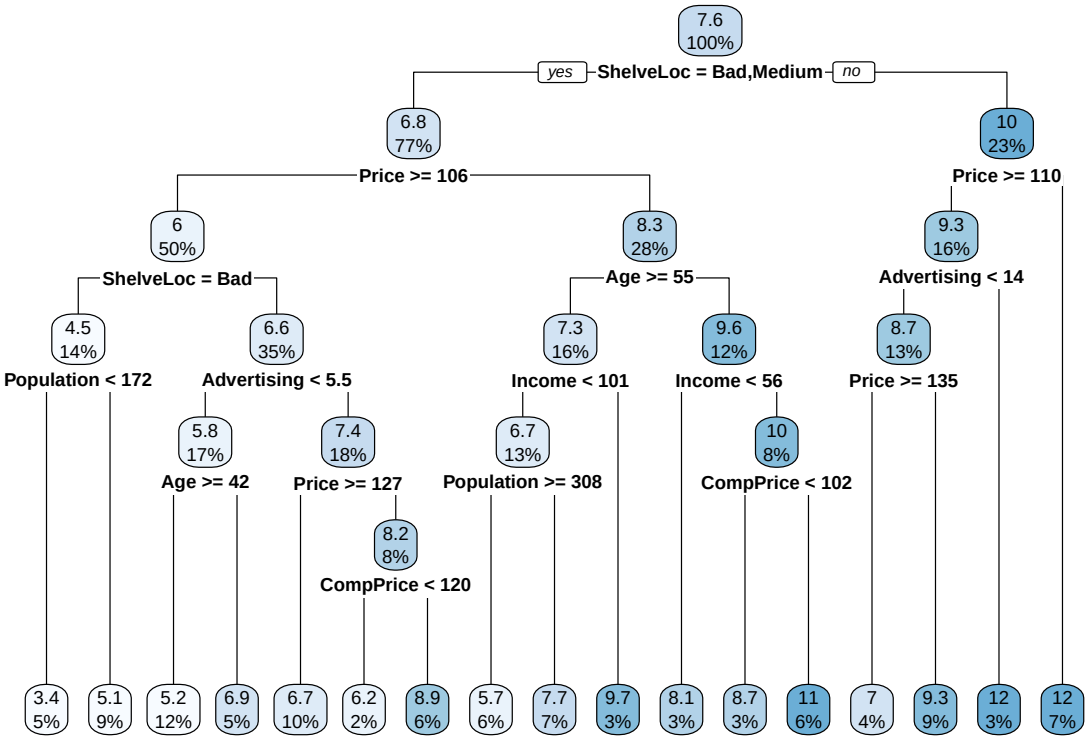
#workflow

```
car_rf_wflow <- workflow() %>%
  add_model(car_rf) %>%
  add_recipe(car_recipe)
car_rf_wflow
```

```
## == Workflow ====================================================================
## Preprocessor: Recipe
## Model: decision_tree()
##
## -- Preprocessor ----------------------------------------------------------------
## 0 Recipe Steps
##
## -- Model -----------------------------------------------------------------------
## Decision Tree Model Specification (regression)
##
## Computational engine: rpart
```

```
fit_obj <- car_rf_wflow %>% fit(data = car_train)
```

```
library(rpart.plot)
plot_fit <- fit_obj %>%
  extract_fit_parsnip()

rpart.plot(
plot_fit$fit,
roundint = FALSE)
```



Code for calculating the test MSE could look like this:

```
fit_obj %>%
  predict(new_data = car_test) %>%
  cbind(car_test) %>%
  summarize(mse = mean((.pred - Sales)^2))
```

```
##      mse
## 1 4.7096
```

The MSE is 4.7096. As the first split variable, ShelveLoc is an important variable that distinguishes the data. Price and age are also important.

c. Use cross-validation in order to determine the optimal level of tree complexity. Does pruning the tree improve the **test** MSE?

```
set.seed(234)
car_folds <- vfold_cv(car_train,
                      v = 4)
```

```
cart_grid <- expand.grid(
  cost_complexity = c(0, 10^(seq(-5,-1,1))))
```

```
car_cart_tune <-
  decision_tree(cost_complexity = tune()) %>%
  set_engine("rpart") %>%
  set_mode("regression")


car_cart_wflow_tune <- workflow() %>%
  add_model(car_cart_tune) %>%
  add_recipe(car_recipe)
```

```
car_tuned <- car_cart_wflow_tune %>%
  tune_grid(resamples = car_folds,
            grid = cart_grid)
```

```
car_best <- finalize_model( car_cart_tune, select_best(car_tuned, "rmse"))
```

```
car_cart_tune_2 <-
  decision_tree(cost_complexity = 0) %>%
  set_engine("rpart") %>%
  set_mode("regression")

car_cart_wflow_tune_2 <- workflow() %>%
  add_model(car_cart_tune_2) %>%
  add_recipe(car_recipe)
```

```
tuned_fit_obj_2 <- car_cart_wflow_tune_2 %>% fit(data = car_train)
```

```
tuned_fit_obj_2 %>%
  predict(new_data = car_test) %>%
  cbind(car_test) %>%
  summarize(mse = mean((.pred - Sales)^2))
```

```
##      mse
## 1 4.7582
```

Tuning by cost complexity doesn't improve the test MSE. The test MSE is now 4.76, which is higher than before.

d. Use the bagging approach (no need to CV, just use default value for `trees`) in order to analyze this data. What test MSE do you obtain? Create a `vip()` plot to determine which variables are most important. [note: in the `train` function use `importance="permutation"` so that you'll have the variable importance information on the next step.]

Recall: if mtry = number of variables, the Random Forest function will produce *exactly* the bagged model.

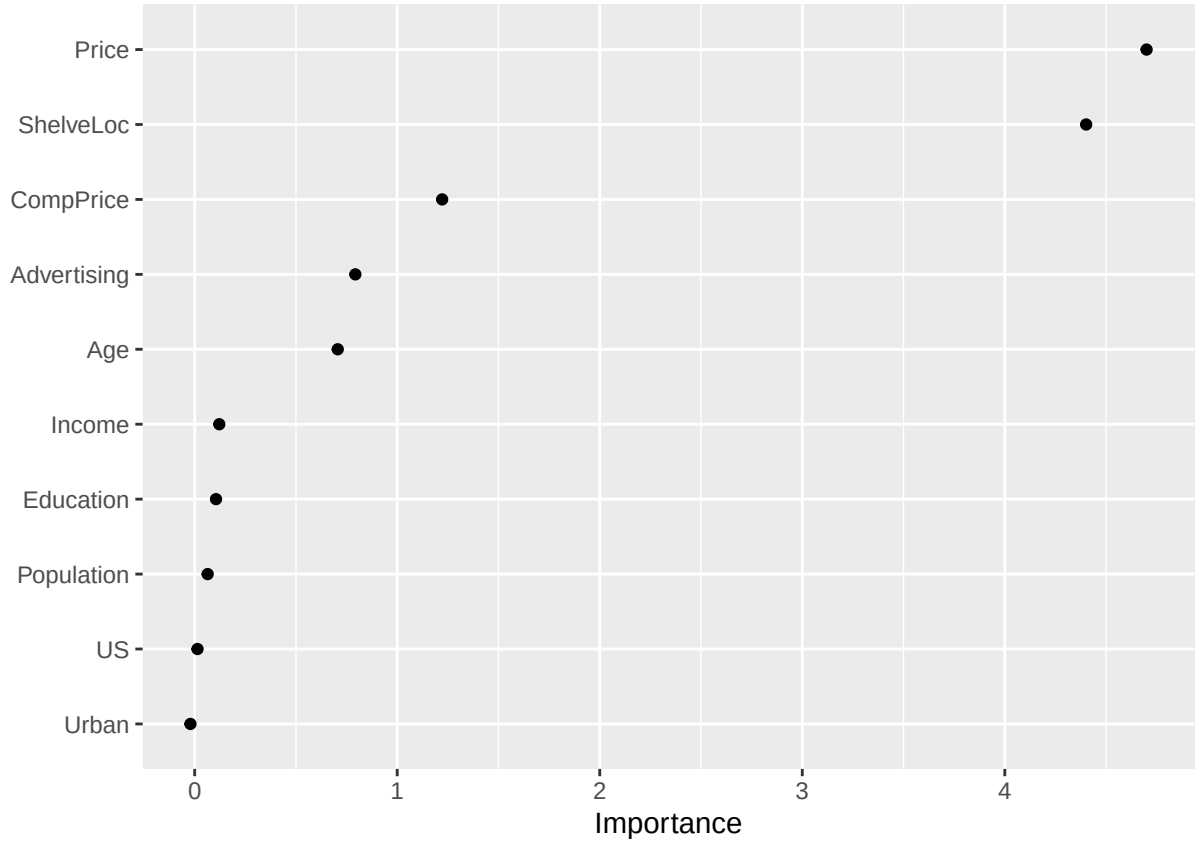
Not necessary, but if you wanted to see how the OOB error compares to the training or test error, you can get the OOB error directly by printing the `fit()` object.

```
set.seed(47)
Cars_mtry <- length(names(Carseats)) - 1
```

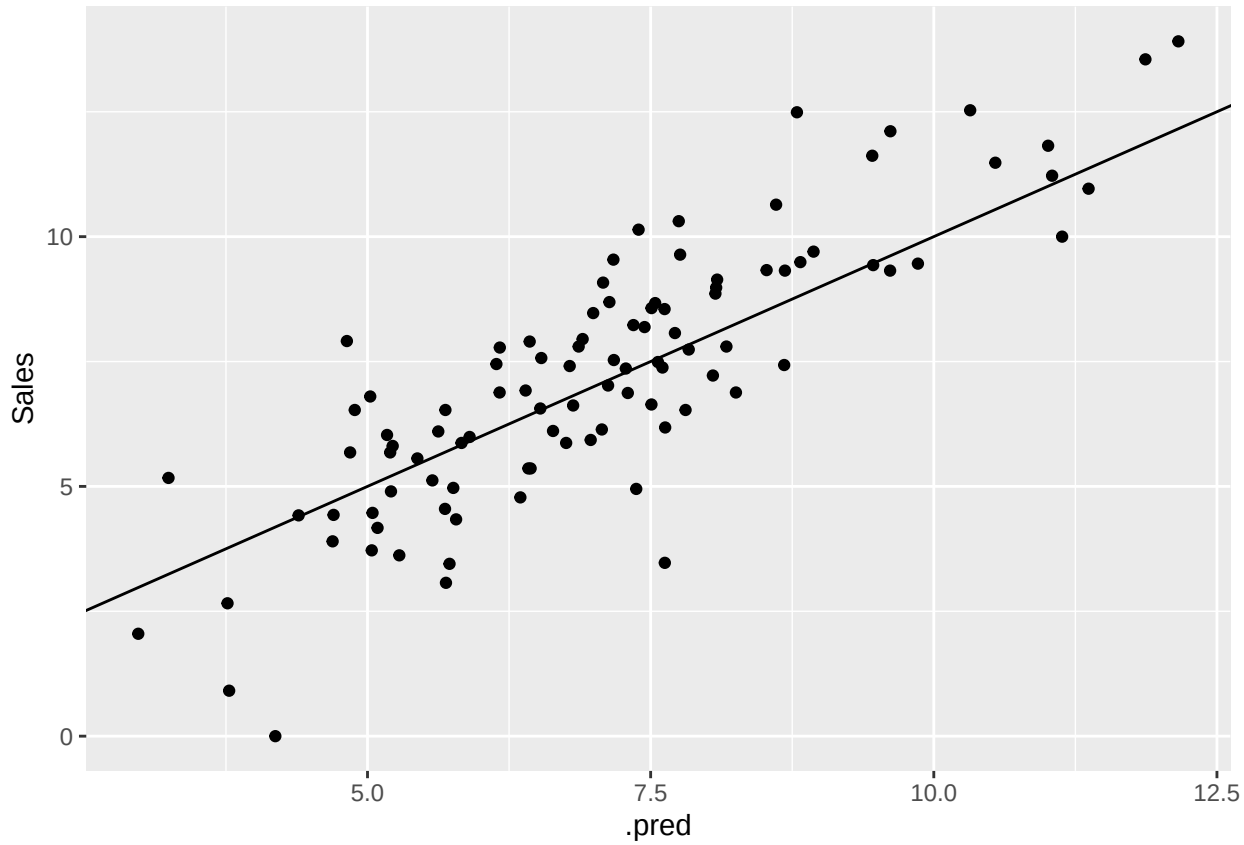
```
Car_model <- rand_forest(mtry = Cars_mtry) %>%
set_engine("ranger", importance = "permutation") %>%
set_mode("regression")


# workflow
Car_wflow <- workflow() %>%
add_model(Car_model) %>%
add_recipe(car_recipe)


# fit
Car_bag <- Car_wflow %>%
fit(data = car_train)


# test MSE
Car_bag %>%
predict(new_data = car_test) %>%
cbind(car_test) %>%
summarize(test_mse = mean((.pred - Sales)^2))
```

```
##   test_mse
## 1   2.0733
```

```
library(vip)
Car_bag %>%
extract_fit_parsnip() %>%
vip(geom = "point")
```




```
Car_bag %>%
predict(new_data = car_test) %>%
cbind(car_test) %>%
ggplot(aes(x = .pred, y = Sales)) +
geom_point() +
geom_abline()
```



I obtain a test MSE of 2.07. This is less than half of the MSE of just one tree!

e. Use Random Forests (no need to CV, just use default value for `trees` and `mtry`) to analyze this data. What **test** MSE do you obtain? Create a `vip()` plot to determine which variables are most important. Describe the effect of using `mtry` (as opposed to the bagged forest), the number of variables considered at each split, on the error rate obtained.

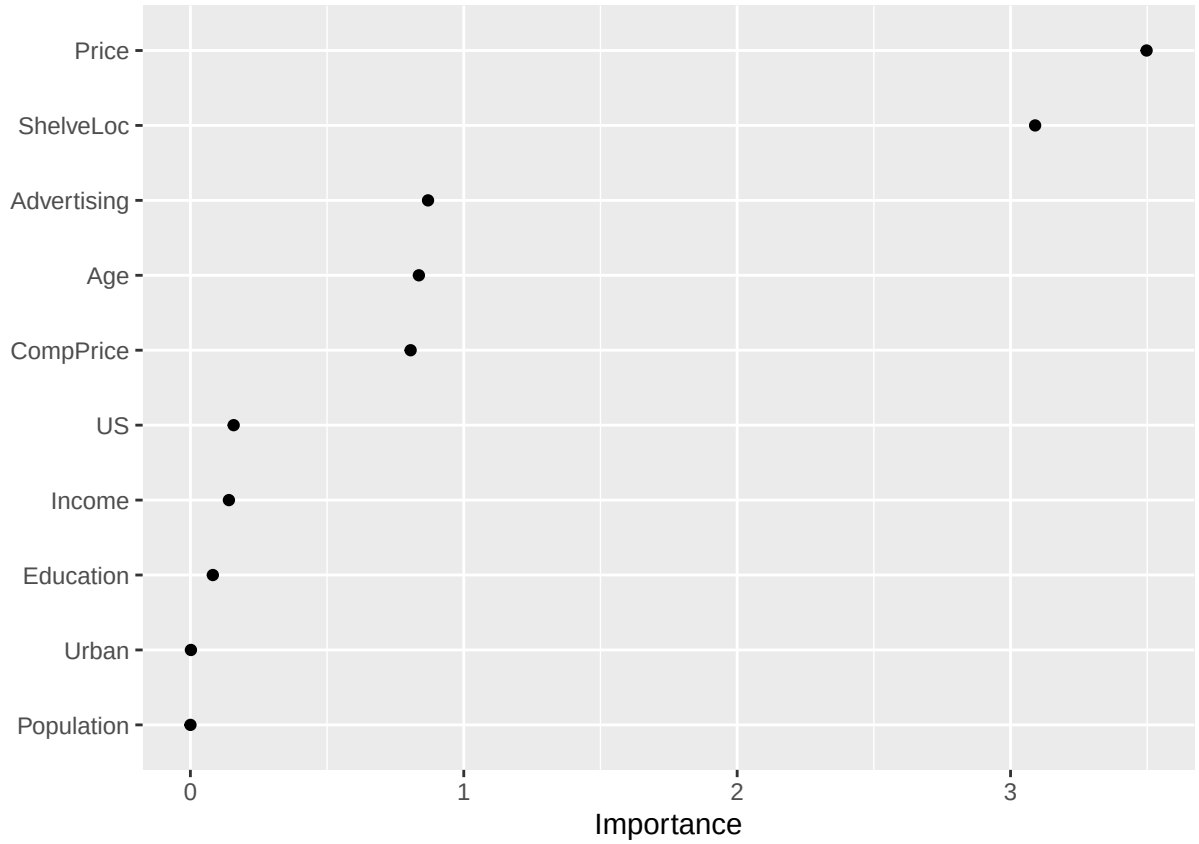
Note: the default `mtry` value is the (rounded down) square root of the number of variables. $\sqrt{10} = 3.16$, so our model will choose `mtry = 3`.

```
Cars_mtry <- 3
Car_model <- rand_forest(mtry = Cars_mtry) %>%
set_engine("ranger", importance = "permutation") %>%
set_mode("regression")

# # workflow
# Car_wflow <- workflow() %>%
# add_model(Car_model) %>%
# add_recipe(Car_recipe)
# fit
Car_3 <- Car_wflow %>%
fit(data = car_train)
# test MSE
Car_3 %>%
predict(new_data = car_test) %>%
```

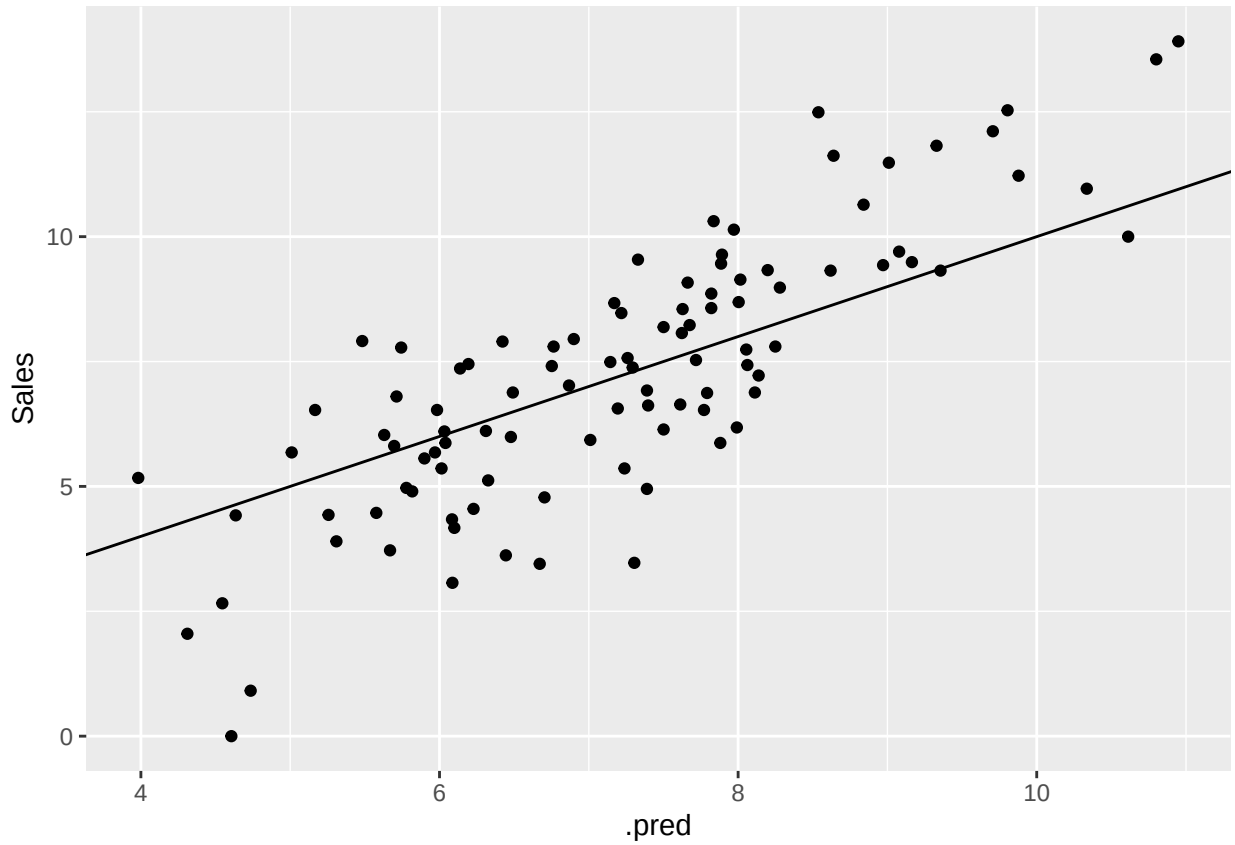
```
cbind(car_test) %>%
summarize(test_mse = mean((.pred - Sales)^2))
```

```
##   test_mse
## 1   2.7013
```

```
Car_3 %>%
extract_fit_parsnip() %>%
vip(geom = "point")
```




```
Car_3 %>%
predict(new_data = car_test) %>%
cbind(car_test) %>%
ggplot(aes(x = .pred, y = Sales)) +
geom_point() +
geom_abline()
```



I obtain a test MSE of 2.70. The effect of limiting mtry is to decorrelate the individual trees. With bagging, the individual trees are all pretty similar, and averaging trees with high correlation results in estimates with high variance. But perhaps in this dataset, bagging trees performs better.

f. In trying to **decide** between CART, a bagged tree, and a regression tree, should you compare the three values of the training MSE or the three values of the test MSE?

We should compare the training MSE.