Predict House Prices for King County

p1 RF

maycd

Contents

```
house <- read.csv("data.csv", stringsAsFactors = TRUE)
```

Stratified sampling

```
set.seed(123)
split_strat <- rsample::initial_split(house, prop = 0.8, strata = 'price')
house_train <- rsample::training(split_strat)
house_test <- rsample::testing(split_strat)
```

```
write.csv(house_train, file = "house_train.csv")
write.csv(house_test, file = "house_test.csv")
rm(list = ls())
```

```
house_train <- read.csv("house_train.csv", stringsAsFactors = TRUE)
dim(house_train)  # dataset: house_train, response: price
```

```
## [1] 17289    20
```

```
# number of features
n_features <- length(setdiff(names(house_train), "price"))
```

RF

```
# train a default random forest model
house_rf1 <- ranger(
  price ~ .,
  data = house_train,
  mtry = floor(n_features / 3),
  respect.unordered.factors = "order",
  seed = 123
)

# get OOB RMSE
(default_rmse <- sqrt(house_rf1$prediction.error))
```

```
## [1] 0.0792796
```

```
house_rf1
```

```
## Ranger result
##
## Call:
##  ranger(price ~ ., data = house_train, mtry = floor(n_features/3),      respect.unordered.factors =
##
## Type:                             Regression
## Number of trees:                  500
## Sample size:                      17289
## Number of independent variables:  19
## Mtry:                             6
## Target node size:                 5
## Variable importance mode:         none
## Splitrule:                        variance
## OOB prediction error (MSE):       0.006285256
## R squared (OOB):                  0.8794334
```

```
# create hyperparameter grid
hyper_grid <- expand.grid(
  mtry = floor(n_features * c(.25, .33, .35, .4)),
  min.node.size = c(1, 3, 5, 10),
  replace = c(TRUE, FALSE),
  sample.fraction = c(.5, .63, .8),
  rmse = NA
)

tic()
# execute full cartesian grid search
for(i in seq_len(nrow(hyper_grid))) {
  # fit model for ith hyperparameter combination
  fit <- ranger(
    formula = price ~ .,
    data = house_train,
    num.trees = n_features * 10,
    mtry = hyper_grid$mtry[i],
    min.node.size = hyper_grid$min.node.size[i],
    replace = hyper_grid$replace[i],
    sample.fraction = hyper_grid$sample.fraction[i],
    verbose = FALSE,
    seed = 123,
    respect.unordered.factors = 'order',
    )
  #export OOB error
  hyper_grid$rmse[i] <- sqrt(fit$prediction.error)
}
toc()
```

```
## 566.82 sec elapsed
```

```
# assess top 10 models
hyper_grid %>%
  arrange(rmse) %>%
  mutate(perc_gain = (default_rmse - rmse) / default_rmse * 100) %>%
  head(10)
```

```
##    mtry min.node.size replace sample.fraction       rmse   perc_gain
## 1     7             5   FALSE            0.63 0.07912199  0.19881405
## 2     7            10   FALSE            0.63 0.07918942  0.11375367
## 3     7             3    TRUE            0.80 0.07919466  0.10714669
## 4     7            10    TRUE            0.80 0.07921723  0.07867266
## 5     7             1    TRUE            0.80 0.07928971 -0.01274560
## 6     7             5    TRUE            0.80 0.07932566 -0.05809381
## 7     7            10   FALSE            0.80 0.07933988 -0.07602535
## 8     7             1   FALSE            0.50 0.07935378 -0.09356203
## 9     7             5   FALSE            0.50 0.07940313 -0.15580745
## 10    7             3    TRUE            0.63 0.07942087 -0.17818088
```

```
best <- which.min(hyper_grid$rmse)
house_rf <- ranger(
    formula = price ~ .,
    data = house_train,
    num.trees = n_features * 10,
    mtry = hyper_grid$mtry[best],
    importance = "impurity",  # importance
    min.node.size = hyper_grid$min.node.size[best],
    replace = hyper_grid$replace[best],
    sample.fraction = hyper_grid$sample.fraction[best],
    verbose = FALSE,
    seed = 123,
    respect.unordered.factors = 'order'
  )
house_rf
```

```
## Ranger result
##
## Call:
##  ranger(formula = price ~ ., data = house_train, num.trees = n_features *      10, mtry = hyper_grid
##
## Type:                             Regression
## Number of trees:                  190
## Sample size:                      17289
## Number of independent variables:  19
## Mtry:                             7
## Target node size:                 5
## Variable importance mode:         impurity
## Splitrule:                        variance
## OOB prediction error (MSE):       0.006260289
## R squared (OOB):                  0.8799123
```

```
# get OOB RMSE
(house_rf_rmse <- sqrt(house_rf$prediction.error))
```

```
## [1] 0.07912199
```

```
save(house_rf, file = "house_rf.rda")
```

p2 GBM

maydc

Contents

```r
house_train <- read.csv("house_train.csv", stringsAsFactors = TRUE)
dim(house_train)  # dataset: house_train, response: price
```

```
## [1] 17289    20
```

```r
# number of features
n_features <- length(setdiff(names(house_train), "price"))
```

GBM

```r
tic()
# run a basic GBM model
set.seed(123)
house_gbm1 <- gbm(
  formula = price ~ .,
  data = house_train,
  distribution = "gaussian",  # SSE loss function
  n.trees = 4000,  # start with sufficiently large n.trees
  shrinkage = 0.1,
  interaction.depth = 3,
  n.minobsinnode = 10,
  cv.folds = 10
)
# find index for number trees with minimum CV error
best <- which.min(house_gbm1$cv.error)

# get MSE and compute RMSE
sqrt(house_gbm1$cv.error[best])
```

```
## [1] 0.0742174
```

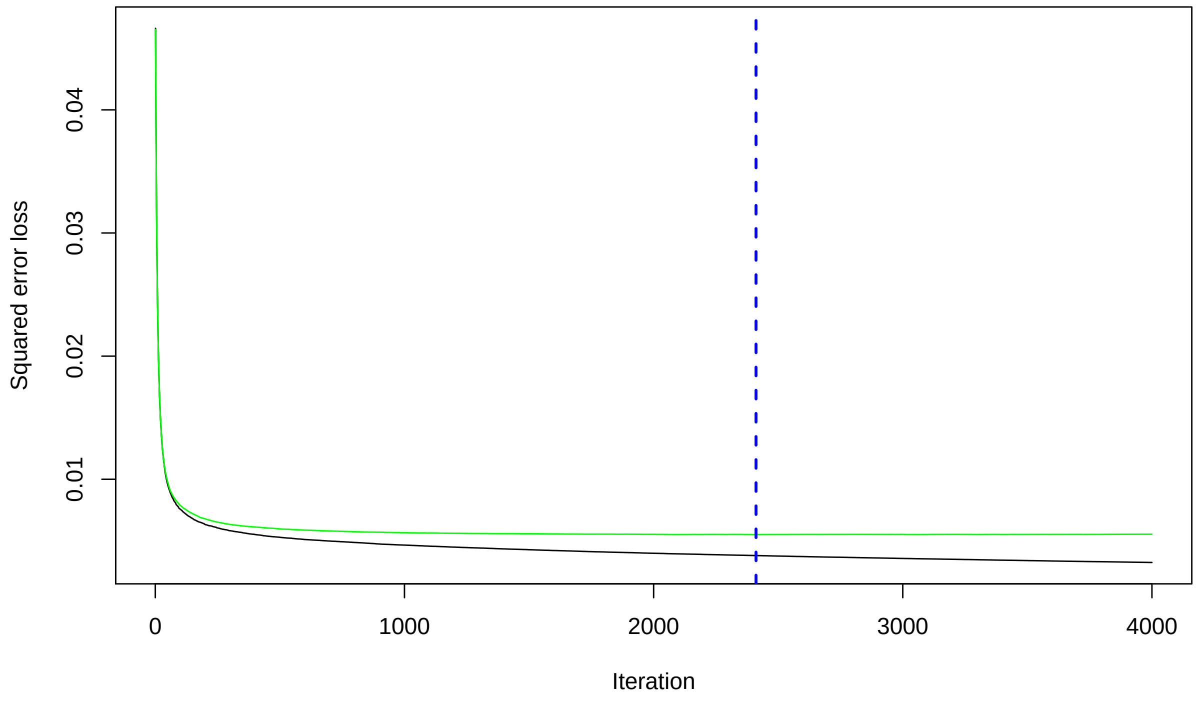
```r
toc()
```

```
## 306.08 sec elapsed
```

```r
house_gbm1
```

```
## gbm(formula = price ~ ., distribution = "gaussian", data = house_train,
##     n.trees = 4000, interaction.depth = 3, n.minobsinnode = 10,
##     shrinkage = 0.1, cv.folds = 10)
## A gradient boosted model with gaussian loss function.
## 4000 iterations were performed.
## The best cross-validation iteration was 2411.
## There were 19 predictors of which 18 had non-zero influence.
```

```r
# plot error curve
gbm.perf(house_gbm1, method = "cv")
```



```
## [1] 2411
```

```r
tic()
# create grid search
hyper_grid <- expand.grid(
  learning_rate = c(0.1, 0.05, 0.01),
  rmse = NA,
  trees = NA,
  time = NA
)
# execute grid search
for(i in seq_len(nrow(hyper_grid))) {

  # fit gbm
  set.seed(123)
  train_time <- system.time({
    m <- gbm(
      formula = price ~ .,
      data = house_train,
      distribution = "gaussian",
      n.trees = 4000,
      shrinkage = hyper_grid$learning_rate[i],
      interaction.depth = 3,
      n.minobsinnode = 10,
      cv.folds = 10
   )
  })

  # add SSE, trees, and training time to results
  hyper_grid$rmse[i]  <- sqrt(min(m$cv.error))
  hyper_grid$trees[i] <- which.min(m$cv.error)
  hyper_grid$time[i]  <- train_time[["elapsed"]]

}

# results
arrange(hyper_grid, rmse)
```

```
##   learning_rate       rmse trees   time
## 1          0.05 0.07382686  4000 189.15
## 2          0.10 0.07421740  2411 272.75
## 3          0.01 0.07759592  4000 190.32
```

```
best <- which.min(hyper_grid$rmse)
toc()
```

```
## 652.72 sec elapsed
```

```
tic()
# search grid
hyper_grid <- expand.grid(
  n.trees = 2500,  # reduce to near optimal n.trees
  shrinkage = hyper_grid$learning_rate[best],
  interaction.depth = c(3, 5, 7),
  n.minobsinnode = c(5, 10, 15)
)

# create model fit function
model_fit <- function(n.trees, shrinkage, interaction.depth, n.minobsinnode) {
  set.seed(123)
  m <- gbm(
    formula = price ~ .,
    data = house_train,
    distribution = "gaussian",
    n.trees = n.trees,
    shrinkage = shrinkage,
    interaction.depth = interaction.depth,
    n.minobsinnode = n.minobsinnode,
    cv.folds = 10
  )
  # compute RMSE
  sqrt(min(m$cv.error))
}
```

```r
# perform search grid with functional programming
hyper_grid$rmse <- purrr::pmap_dbl(
  hyper_grid,
  ~ model_fit(
    n.trees = ..1,
    shrinkage = ..2,
    interaction.depth = ..3,
    n.minobsinnode = ..4
    )
)

# results
arrange(hyper_grid, rmse)
```

```
##   n.trees shrinkage interaction.depth n.minobsinnode       rmse
## 1    2500      0.05                 7              5 0.07293927
## 2    2500      0.05                 7             10 0.07314956
## 3    2500      0.05                 7             15 0.07321427
## 4    2500      0.05                 5              5 0.07342068
## 5    2500      0.05                 5             10 0.07343202
## 6    2500      0.05                 5             15 0.07352400
## 7    2500      0.05                 3             10 0.07441080
## 8    2500      0.05                 3             15 0.07441427
## 9    2500      0.05                 3              5 0.07453070
```

```r
best <- which.min(hyper_grid$rmse)
toc()
```

```
## 2471.02 sec elapsed
```

```r
set.seed(123)
house_gbm <- gbm(
  formula = price ~ .,
  data = house_train,
  distribution = "gaussian",  # SSE loss function
  n.trees = hyper_grid$n.trees[best],
  shrinkage = hyper_grid$shrinkage[best],
  interaction.depth = hyper_grid$interaction.depth[best],
  n.minobsinnode = hyper_grid$n.minobsinnode[best],
  cv.folds = 10
)
house_gbm
```

```
## gbm(formula = price ~ ., distribution = "gaussian", data = house_train,
##     n.trees = hyper_grid$n.trees[best], interaction.depth = hyper_grid$interaction.depth[best],
##     n.minobsinnode = hyper_grid$n.minobsinnode[best], shrinkage = hyper_grid$shrinkage[best],
##     cv.folds = 10)
## A gradient boosted model with gaussian loss function.
## 2500 iterations were performed.
## The best cross-validation iteration was 2148.
## There were 19 predictors of which 18 had non-zero influence.
```

```r
# find index for number trees with minimum CV error
best <- which.min(house_gbm$cv.error)

# get MSE and compute RMSE
```

```r
(house_gbm_rmse <- sqrt(house_gbm$cv.error[best]))
```

```
## [1] 0.07293927
```

```r
save(house_gbm, file = "house_gbm.rda")
```

p3 XGB

maycd

Contents

```
house_train <- read.csv("house_train.csv", stringsAsFactors = TRUE)
dim(house_train)  # dataset: house_train, response: price
```

```
## [1] 17289    20
```

```
# number of features
n_features <- length(setdiff(names(house_train), "price"))
```

XGBoost

```
xgb_prep <- recipe(price ~ ., data = house_train) %>%
  step_integer(all_nominal()) %>%
  prep(training = house_train, retain = TRUE) %>%
  juice()

X <- as.matrix(xgb_prep[setdiff(names(xgb_prep), "price")])
Y <- xgb_prep$price
```

```
# before tuning
tic()
set.seed(123)
house_xgb1 <- xgb.cv(
  data = X,
  label = Y,
  nrounds = 2500,
  objective = "reg:squarederror",
  early_stopping_rounds = 5,
  nfold = 10,
  params = list(
    eta = 0.01,
    max_depth = 7,
    min_child_weight = 2,
    subsample = 0.8,
    colsample_bytree = 0.9),
  verbose = 0
)

# minimum test CV RMSE
```

```r
min(house_xgb1$evaluation_log$test_rmse_mean)
```

```
## [1] 0.072148
```

```r
toc()
```

```
## 364.36 sec elapsed
```

```r
tic()
# hyperparameter grid
hyper_grid <- expand.grid(
  eta = 0.01,
  max_depth = 7,
  min_child_weight = 2,
  subsample = 0.8,
  colsample_bytree = 0.9,
  gamma = c(0, 1, 10),
  lambda = c(0, 1e-3, 1e-2, 0.1, 1, 100),
  alpha = c(0, 1e-2, 0.05, 0.1, 1, 100),
  rmse = 0,          # a place to dump RMSE results
  trees = 0          # a place to dump required number of trees
)

# grid search
for(i in seq_len(nrow(hyper_grid))) {
  set.seed(123)
  m <- xgb.cv(
    data = X,
    label = Y,
    nrounds = 3000,
    objective = "reg:squarederror",
    early_stopping_rounds = 10,
    nfold = 10,
    verbose = 0,
    params = list(
      eta = hyper_grid$eta[i],
      max_depth = hyper_grid$max_depth[i],
      min_child_weight = hyper_grid$min_child_weight[i],
      subsample = hyper_grid$subsample[i],
      colsample_bytree = hyper_grid$colsample_bytree[i],
      gamma = hyper_grid$gamma[i],
      lambda = hyper_grid$lambda[i],
      alpha = hyper_grid$alpha[i]
    )
  )
  hyper_grid$rmse[i] <- min(m$evaluation_log$test_rmse_mean)
  hyper_grid$trees[i] <- m$best_iteration
}

# results
hyper_grid %>%
  filter(rmse > 0) %>%
  arrange(rmse) %>%
  glimpse()
```

```
## Rows: 108
## Columns: 10
## $ eta              <dbl> 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01,~
## $ max_depth        <dbl> 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7,~
## $ min_child_weight <dbl> 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,~
## $ subsample        <dbl> 0.8, 0.8, 0.8, 0.8, 0.8, 0.8, 0.8, 0.8, 0.8, 0.8, 0.8~
## $ colsample_bytree <dbl> 0.9, 0.9, 0.9, 0.9, 0.9, 0.9, 0.9, 0.9, 0.9, 0.9, 0.9~
## $ gamma            <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,~
## $ lambda           <dbl> 1e-01, 1e+00, 1e+00, 0e+00, 1e-03, 1e+00, 1e-02, 1e+0~
## $ alpha            <dbl> 1.00, 0.10, 1.00, 1.00, 1.00, 0.01, 1.00, 0.00, 0.05,~
## $ rmse             <dbl> 0.0718948, 0.0719072, 0.0719247, 0.0719401, 0.0719519~
## $ trees            <dbl> 2529, 1990, 2263, 2208, 2317, 2105, 2263, 1990, 1796,~
```

```r
best <- which.min(hyper_grid$rmse) # rmse does not decrease
toc()
```

```
## 28912.48 sec elapsed
```

```r
# after tuning
tic()
set.seed(123)
house_xgb <- xgb.cv(
  data = X,
  label = Y,
  nrounds = 2500,
  objective = "reg:squarederror",
  early_stopping_rounds = 5,
  nfold = 10,
  verbose = 0,
  params = list(
    eta = 0.01,
    max_depth = 7,
    min_child_weight = 2,
    subsample = 0.8,
    colsample_bytree = 0.9,
    gamma = hyper_grid$gamma[best],
    lambda = hyper_grid$lambda[best],
    alpha = hyper_grid$alpha[best]
  )
)

# minimum test CV RMSE
min(house_xgb$evaluation_log$test_rmse_mean)
```

```
## [1] 0.0719843
```

```r
toc()
```

```
## 453.75 sec elapsed
```

```r
save(house_xgb, file = "house_xgb.rda")
```

```r
# final model
tic()
set.seed(123)
house_xgb_final <- xgboost(
  data = X,
  label = Y,
  nrounds = 2500,
  objective = "reg:squarederror",
  early_stopping_rounds = 5,
  verbose = 0,
  params = list(
    eta = 0.01,
    max_depth = 7,
    min_child_weight = 2,
    subsample = 0.8,
    colsample_bytree = 0.9,
    gamma = hyper_grid$gamma[best],
    lambda = hyper_grid$lambda[best],
    alpha = hyper_grid$alpha[best]
  )
)
house_xgb_final
```

```
## ##### xgb.Booster
## raw: 15.9 Mb
## call:
##   xgb.train(params = params, data = dtrain, nrounds = nrounds,
##     watchlist = watchlist, verbose = verbose, print_every_n = print_every_n,
##     early_stopping_rounds = early_stopping_rounds, maximize = maximize,
##     save_period = save_period, save_name = save_name, xgb_model = xgb_model,
##     callbacks = callbacks, objective = "reg:squarederror")
## params (as set within xgb.train):
##   eta = "0.01", max_depth = "7", min_child_weight = "2", subsample = "0.8", colsample_bytree = "0.9"
## xgb.attributes:
##   best_iteration, best_msg, best_ntreelimit, best_score, niter
## callbacks:
##   cb.evaluation.log()
##   cb.early.stop(stopping_rounds = early_stopping_rounds, maximize = maximize,
##     verbose = verbose)
## # of features: 19
## niter: 2500
## best_iteration : 2500
## best_ntreelimit : 2500
## best_score : 0.0539
## best_msg : [2500]    train-rmse:0.053900
## nfeatures : 19
## evaluation_log:
##     iter train_rmse
##        1   5.119303
##        2   5.068162
## ---
##     2499   0.053903
##     2500   0.053900
```

```r
toc()
```

```
## 56.05 sec elapsed
```

```r
save(house_xgb_final, file = "house_xgb_final.rda")
```

p4 Comparison

maycd

Contents

As the last part of project, this document "p4 Comparison" will chiefly discuss results of the models. (All objects loaded into this document are outcomes from the previous three documents executed in sequence "p1 RF", "p2 GBM", and "p3 Xgboost". Because we did not specify `row.names=FALSE` when writing data into csv in the document "p1 RF", train and test data both have a duplicate row containing rownames. However, the issue does not affect our tree-based models.)

```
# KingCountyHouses package
# Description:
# housing data in Washington State from 2014-05-02 to 2015-05-27. There are 21,613 data points with 19
# Source:
# https://geodacenter.github.io/data-and-lab//KingCounty-HouseSales2015/
```

```
house_train <- read.csv("house_train.csv", stringsAsFactors = TRUE)
dim(house_train)  # dataset: house_train, response: price
```

```
## [1] 17289    20
```

```
# number of features
n_features <- length(setdiff(names(house_train), "price"))
```

RF

Hyperparameters

In document "p1 RF", we have performed tasks (1) set seed as 123, (2) performed stratified sampling, and (3) performed random forest on the training data.

The best tuning hyperparameters are: mtry = 7 (a random subset of 7 variables for split) min.node.size = 5 (at least 5 observations in a leaf) replace = FALSE (sampling without replacement) sample.fraction = 0.63 (0.63 of observations to sample)

We find them by Cartesian full grid search: we start with a RF model with default hyperparameters, create a hyperparameter tuning grid containing different values of mtry, min.node.size, replace, and sample.fraction, execute Cartesian full grid search in a for loop of one model for each combination of hyperparameters in the grid, measure their RMSE. The best tuning hyperparameters are in the one with the least cross-validated RMSE on the training data, which is 0.07912 here.

```
load(file = "house_rf.rda")
```

```
house_rf
```

```
## Ranger result
##
## Call:
##  ranger(formula = price ~ ., data = house_train, num.trees = n_features *      10, mtry = hyper_grid$
##
## Type:                             Regression
## Number of trees:                  190
## Sample size:                      17289
## Number of independent variables:  19
## Mtry:                             7
## Target node size:                 5
## Variable importance mode:         impurity
## Splitrule:                        variance
## OOB prediction error (MSE):       0.006260289
## R squared (OOB):                  0.8799123
```

After tuning, model RMSE decreases by

```
paste(round((1 - 0.07912199/0.0792796) * 100, 4), "%", sep = "")
```

```
## [1] "0.1988%"
```

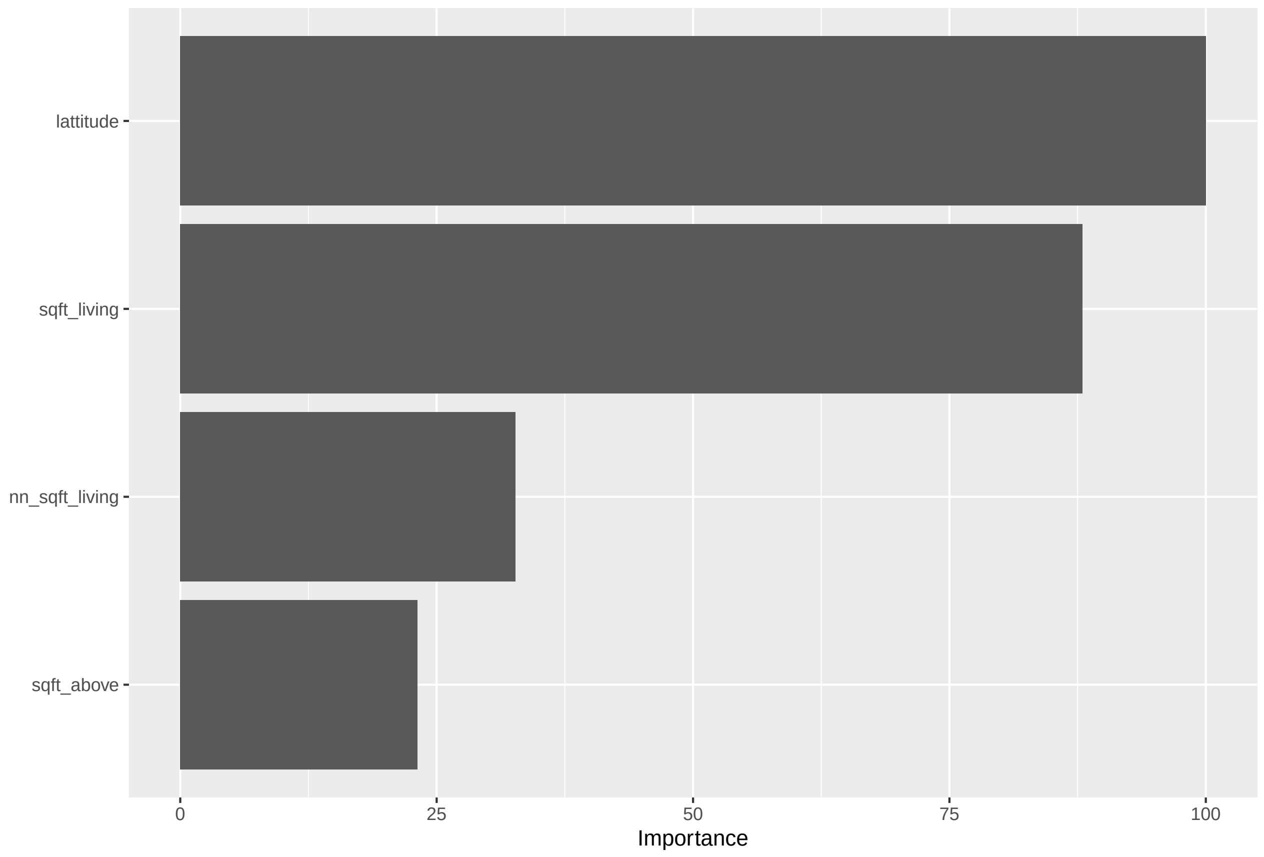
Variable importance

`lattitude`, `sqft_living`, `nn_sqft_living`, and `sqft_above` are the four most important predictors.

```
vi_scores <- vi(house_rf)
head(vi_scores, 4)
```

```
## # A tibble: 4 x 2
##   Variable       Importance
##   <chr>               <dbl>
## 1 lattitude           173.
## 2 sqft_living         152.
## 3 nn_sqft_living       56.5
## 4 sqft_above           40.0
```

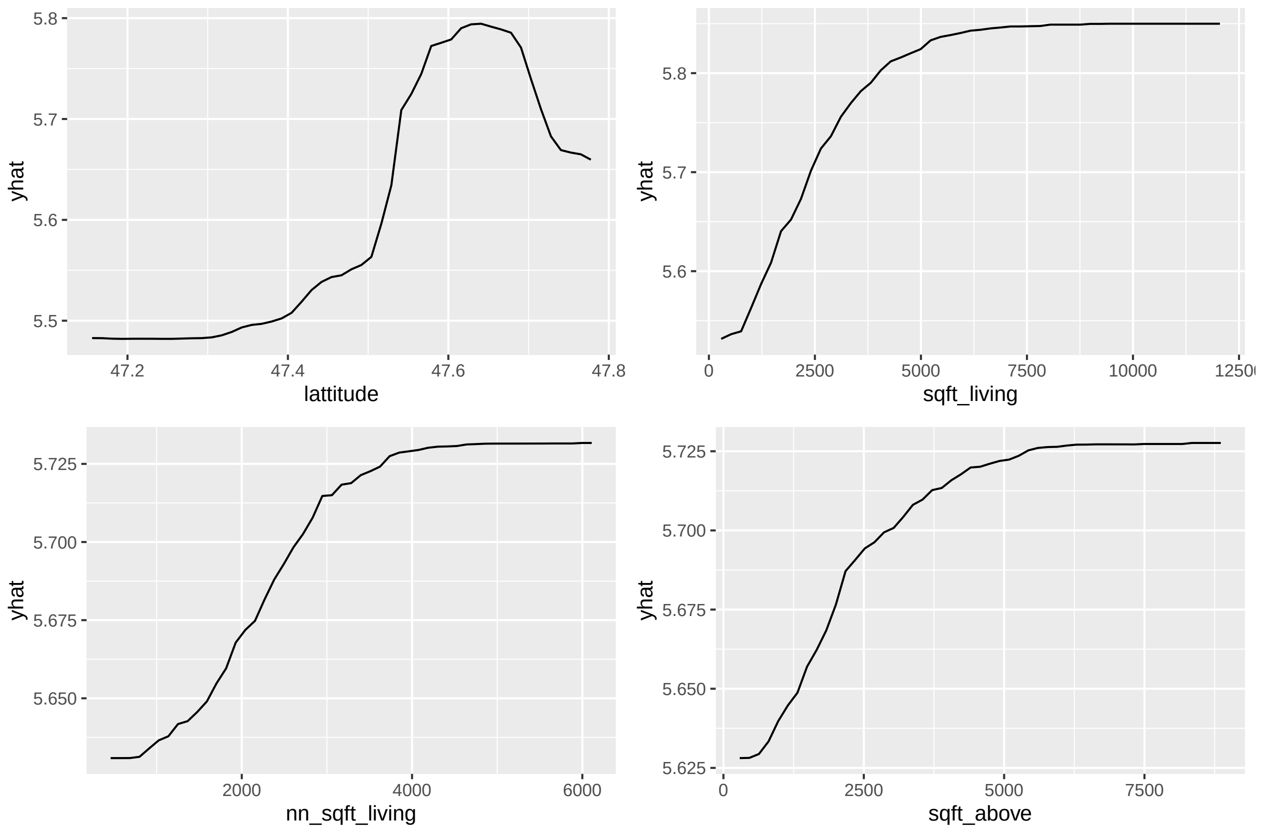
```
vip(house_rf, num_features = 4, scale = TRUE)
```



PDP plots

```
tic()
p1 <- partial(house_rf, pred.var = vi_scores[[1, 1]]) %>% autoplot()
p2 <- partial(house_rf, pred.var = vi_scores[[2, 1]]) %>% autoplot()
p3 <- partial(house_rf, pred.var = vi_scores[[3, 1]]) %>% autoplot()
p4 <- partial(house_rf, pred.var = vi_scores[[4, 1]]) %>% autoplot()
grid.arrange(p1, p2, p3, p4, ncol = 2)
```

```
## Warning: Use of `object[[1L]]` is discouraged. Use `.data[[1L]]` instead.

## Warning: Use of `object[["yhat"]]` is discouraged. Use `.data[["yhat"]]`
## instead.

## Warning: Use of `object[[1L]]` is discouraged. Use `.data[[1L]]` instead.

## Warning: Use of `object[["yhat"]]` is discouraged. Use `.data[["yhat"]]`
## instead.

## Warning: Use of `object[[1L]]` is discouraged. Use `.data[[1L]]` instead.

## Warning: Use of `object[["yhat"]]` is discouraged. Use `.data[["yhat"]]`
## instead.

## Warning: Use of `object[[1L]]` is discouraged. Use `.data[[1L]]` instead.
## Warning: Use of `object[["yhat"]]` is discouraged. Use `.data[["yhat"]]`
## instead.
```



```
toc()
```

```
## 177.027 sec elapsed
```

The marginal effect of latitude is that the mean of predicted prices soars to almost $580,000 drastically when latitude is near 47.6 to 47.7 degree north, holding other variables constant at the average levels. This left-skewed line suggests that houses in the northern region of King County in Washington State are sold at much higher sale prices than those of the southern region and extreme northern region.

A possible explanation is that there are various infrastructures including large parks, commercial buildings, churches, and schools round the latitude from local maps.

The three predictors related to living size share a similar trend of positive correlation with prices. The effect gradually declines and the prices level off.

The marginal effect of living size is that the mean of predicted prices raises steadily at $8 per square foot as living size increases. When living size exceeds 7,500 square feet, increase in living size does not influence prediction of house prices.

The mean of predicted prices experiences the steepest increase at a rate of $9.75 per square feet when size of living space of 15 neighbors is between 2000 to 4000 square feet.

When the house possesses a small living size above the ground of within 2500 square feet, the mean of predicted prices is mostly affected, with the slope of a $3 increase in price per square feet increase in size.

Basic GBM

Hyperparameters

In document "p2 GBM", we have performed task (4) performed basic GBM algorithm on the training data.

The best tuning hyperparameters are: shrinkage = 0.05 (learning rate) interaction.depth = 7 (7 splits on a tree) n.minobsinnode = 5 (at least 5 observations in a leaf)

We find them by alternative optimization with grid search: Step 1: we run a basic GBM model Step 2: tune shrinkage in grid search first, choose the optimal shrinkage by RMSE Step 3: then tune the combination of interaction.depth and n.minobsinnode with shrinkage fixed at the new value, choose the optimal combination of interaction.depth and n.minobsinnode RMSE. Step 4: Usually, we repeat the step 2 and 3 until no significant improvement. Here we omit Step 4 because of cost-effectiveness consideration. The best tuning hyperparameters are in the one with the least cross-validated RMSE on the training data, which is 0.07294 here.

```
load(file = "house_gbm.rda")
```

```
house_gbm$shrinkage
```

```
## [1] 0.05
```

```
house_gbm$interaction.depth
```

```
## [1] 7
```

```
house_gbm$n.minobsinnode
```

```
## [1] 5
```

After tuning, model RMSE decreases by

```
paste(round((1 - 0.07293927/0.0742174) * 100, 4), "%", sep = "")
```

```
## [1] "1.7221%"
```

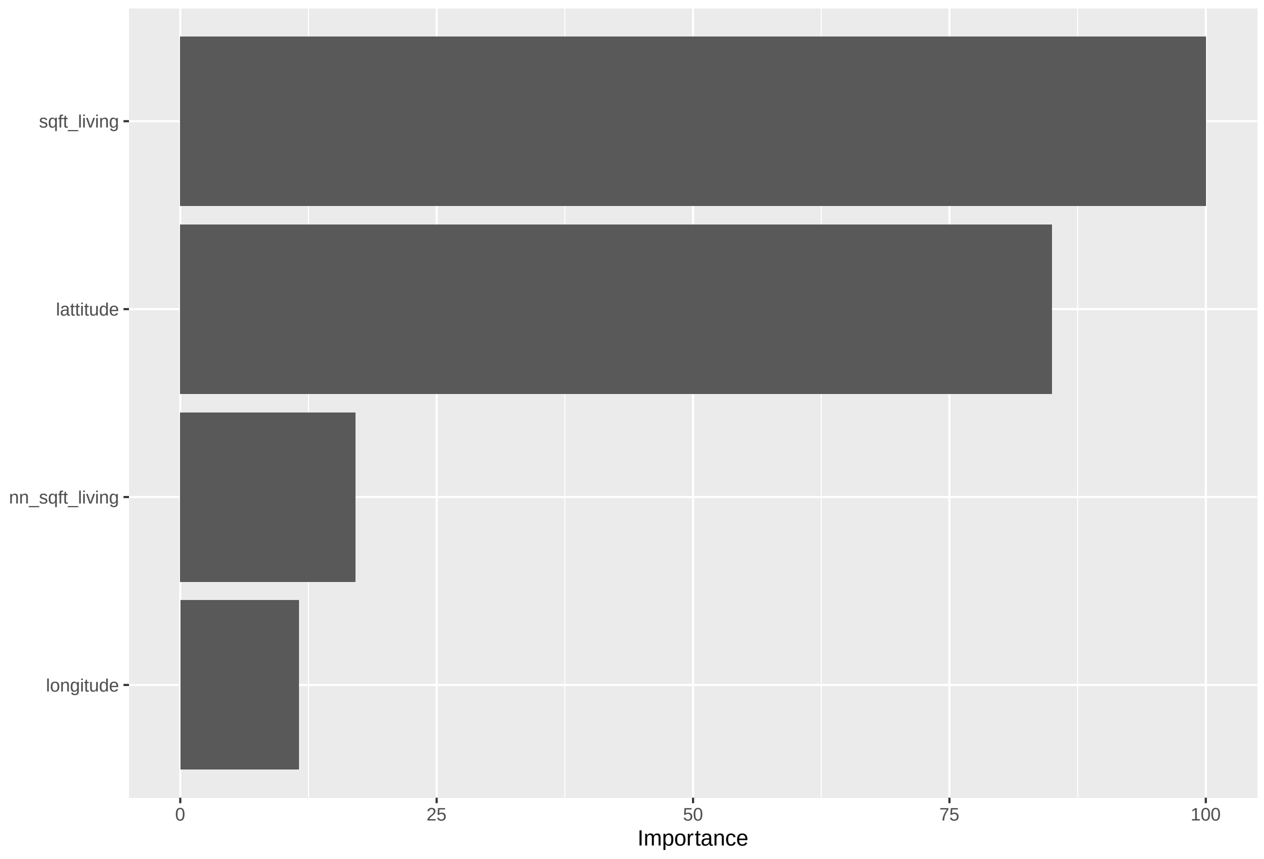
Variable importance

`sqft_living`, `lattitude`, `nn_sqft_living`, and `longitude` are the four most important predictors.

```
vi_scores <- vi(house_gbm)
head(vi_scores, 4)
```

```
## # A tibble: 4 x 2
##   Variable       Importance
##   <fct>               <dbl>
## 1 sqft_living         39.0
## 2 lattitude           33.2
## 3 nn_sqft_living       6.67
## 4 longitude            4.51
```

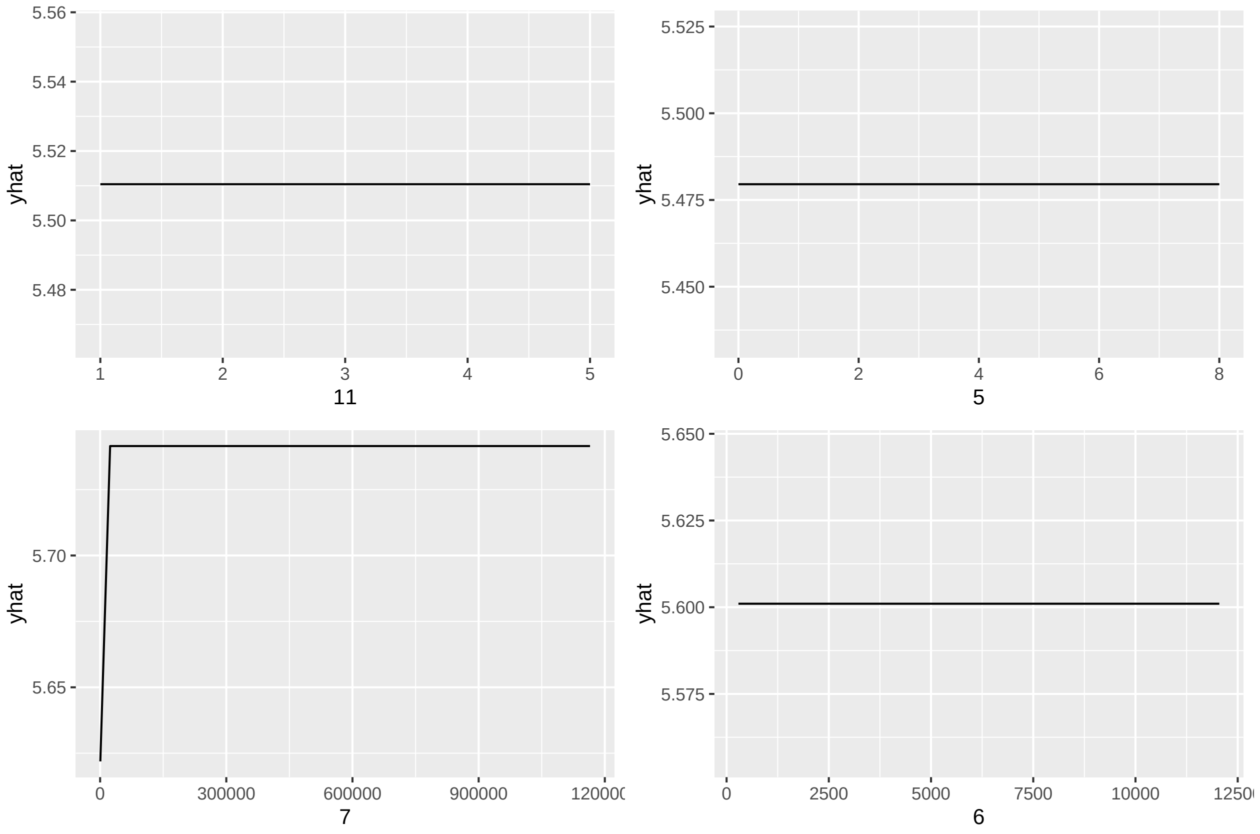
```
vip(house_gbm, num_features = 4, scale = TRUE)
```



PDP plots

```
tic()
p1 <- partial(house_gbm, pred.var = vi_scores[[1, 1]], n.trees = 100) %>%
autoplot()
p2 <- partial(house_gbm, pred.var = vi_scores[[2, 1]], n.trees = 100) %>%
autoplot()
p3 <- partial(house_gbm, pred.var = vi_scores[[3, 1]], n.trees = 100) %>%
autoplot()
p4 <- partial(house_gbm, pred.var = vi_scores[[4, 1]], n.trees = 100) %>%
autoplot()
grid.arrange(p1, p2, p3, p4, ncol = 2)
```

```
## Warning: Use of `object[[1L]]` is discouraged. Use `.data[[1L]]` instead.

## Warning: Use of `object[["yhat"]]` is discouraged. Use `.data[["yhat"]]`
## instead.

## Warning: Use of `object[[1L]]` is discouraged. Use `.data[[1L]]` instead.

## Warning: Use of `object[["yhat"]]` is discouraged. Use `.data[["yhat"]]`
## instead.

## Warning: Use of `object[[1L]]` is discouraged. Use `.data[[1L]]` instead.

## Warning: Use of `object[["yhat"]]` is discouraged. Use `.data[["yhat"]]`
## instead.

## Warning: Use of `object[[1L]]` is discouraged. Use `.data[[1L]]` instead.
## Warning: Use of `object[["yhat"]]` is discouraged. Use `.data[["yhat"]]`
## instead.
```



```r
toc()
```

```
## 0.441 sec elapsed
```

The marginal effect of latitude predicted by GBM is similar to that of RF. A possible explanation for the phenomena of high prices at a certain latitude is that there are various infrastructures including large parks, commercial buildings, churches, and schools around the latitude from local maps.

The two predictors related to living size share a similar trend of positive correlation with prices. The effect gradually declines and the prices level off.

The marginal effect of living size is that the mean of predicted prices raises steadily as living size increases. When living size exceeds 6,000 square feet, house prices remain \$600,000, which is \$10,000 greater than the forecast in RF.

The mean of predicted prices experiences the steepest increase at a rate of \$9.75 per square feet when the size of the living space of the nearest 15 neighbors is between 2000 to 4000 square feet.

When the house possesses a small living size above the ground of within 2500 square feet, the mean of predicted prices is mostly affected, with the slope of a \$3 increase in price per square feet increase in size.

The marginal effect of longitude is that the mean of predicted prices decreases by steps of \$1,200 or \$2,500 as longitude increases. From local maps, the houses are gathered in blocks, thus the change in house prices is not smooth. It reveals that proximity to the coastline in the west and away from mountains in the east lead to high house prices.

Xgboost

Hyperparameters

In document "p3 XGB", we have performed task (5) performed Xgboost algorithm on the training data.

The best tuning hyperparameters are: gamma = 0 (minimum loss reduction required to split) lambda = 0.1 (Ridge L2 regularization on leaf weights) alpha = 1 (Lasso L1 regularization on leaf weights)

Like in RF, we find them by Cartesian full grid search: we start with a RF model with default hyperparameters, create a hyperparameter tuning grid containing different values of gamma, lambda, and alpha, execute Cartesian full grid search in a for loop of one model for each combination of hyperparameters in the grid, measure their RMSE. The best tuning hyperparameters are in the one with the least cross-validated RMSE on the training data, which is 0.07198 here.

```
xgb_prep <- recipe(price ~ ., data = house_train) %>%
  step_integer(all_nominal()) %>%
  prep(training = house_train, retain = TRUE) %>%
  juice()

X <- as.matrix(xgb_prep[setdiff(names(xgb_prep), "price")])
Y <- xgb_prep$price
```

```
load(file = "house_xgb.rda")
```

```
house_xgb$params
```

```
## $eta
## [1] 0.01
##
## $max_depth
## [1] 7
##
## $min_child_weight
## [1] 2
##
## $subsample
## [1] 0.8
##
## $colsample_bytree
## [1] 0.9
##
## $gamma
## [1] 0
##
## $lambda
## [1] 0.1
##
## $alpha
## [1] 1
##
## $objective
## [1] "reg:squarederror"
##
## $silent
## [1] 1
```

After tuning, model RMSE decreases by

```r
paste(round((1 - 0.0719843/0.072148) * 100, 4), "%", sep = "")
```

```
## [1] "0.2269%"
```

```r
load(file = "house_xgb_final.rda")
```

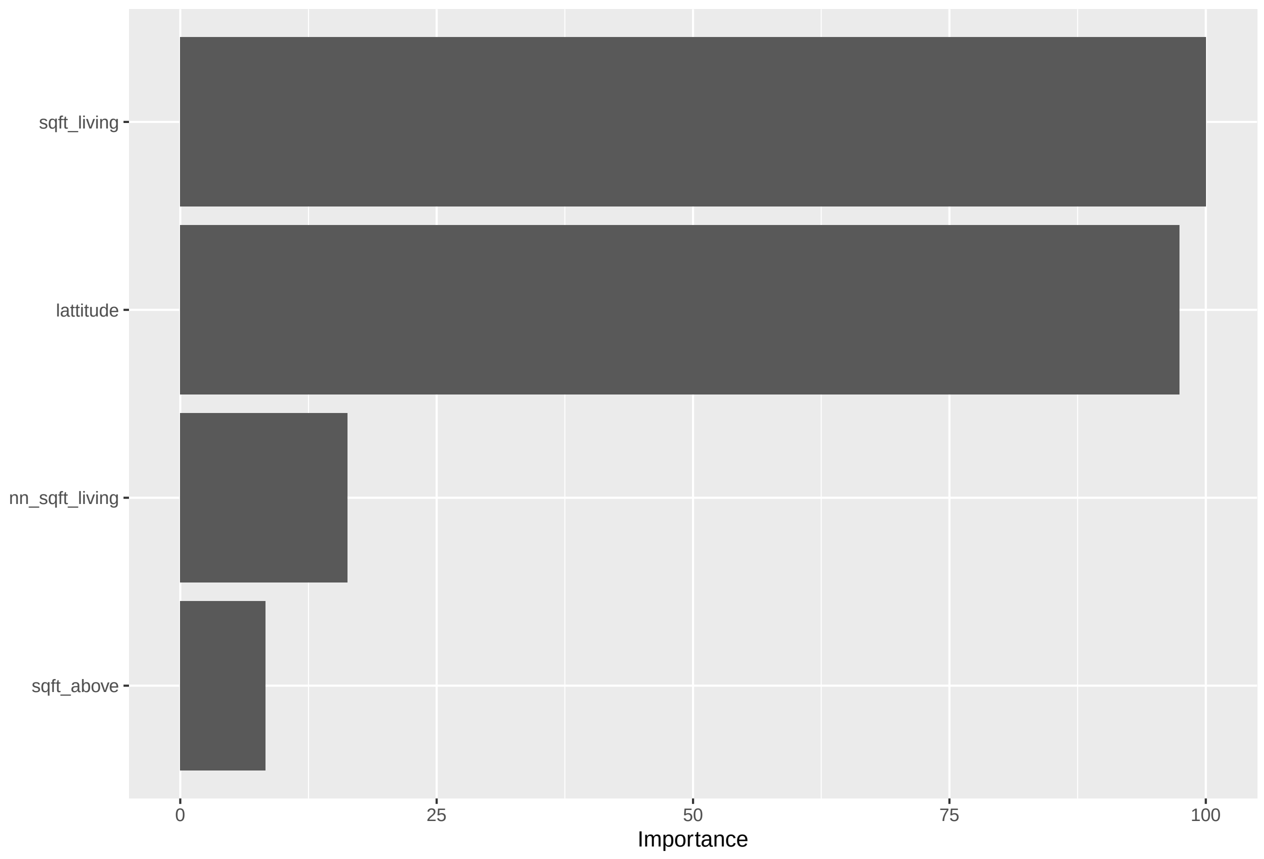
Variable importance

`sqft_living`, `lattitude`, `nn_sqft_living`, and `sqft_above` are the four most important predictors.

```r
vi_scores <- vi(house_xgb_final)
head(vi_scores, 4)
```

```
## # A tibble: 4 x 2
##   Variable       Importance
##   <chr>               <dbl>
## 1 sqft_living        0.400
## 2 lattitude          0.390
## 3 nn_sqft_living     0.0652
## 4 sqft_above         0.0332
```

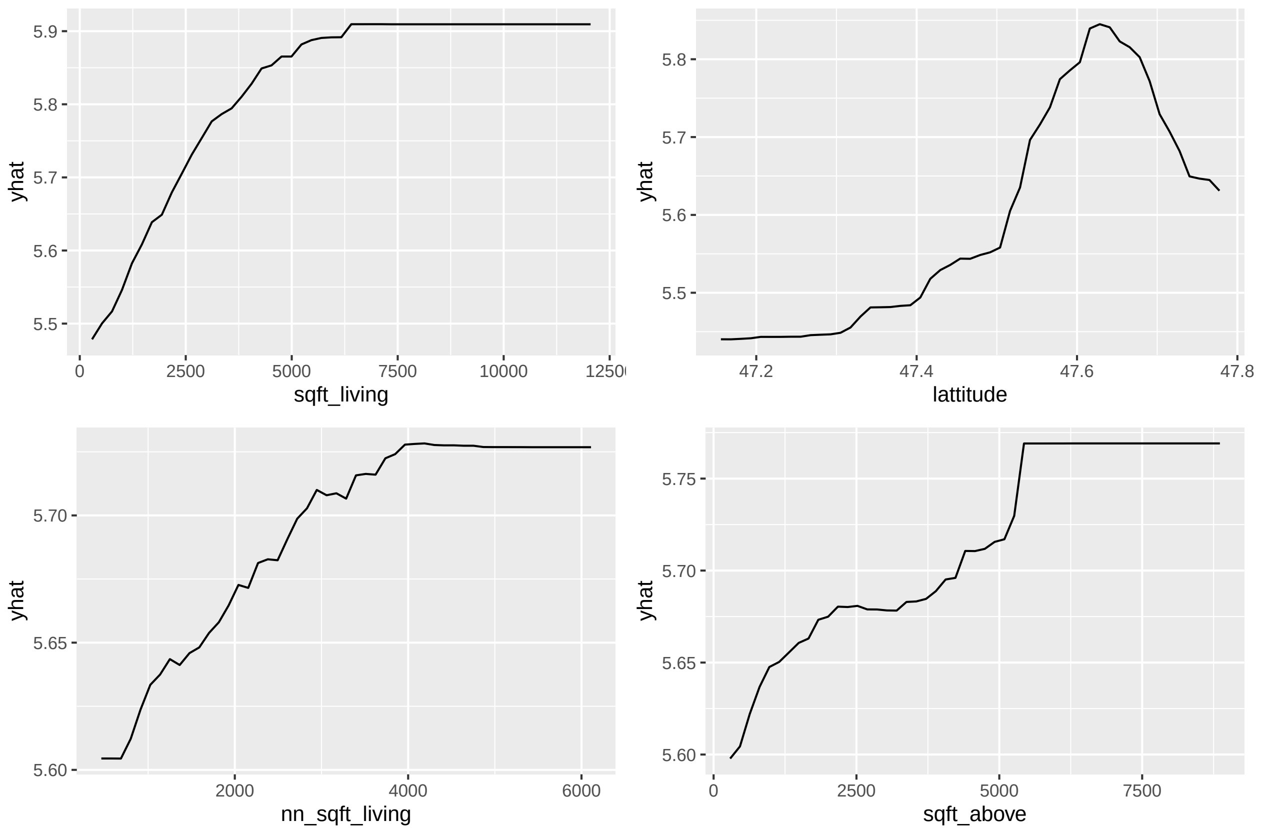
```r
vip(house_xgb_final, num_features = 4, scale = TRUE)
```



PDP plots

```r
tic()
p1 <- partial(house_xgb_final, pred.var = vi_scores[[1, 1]],
              train = X, type = "regression") %>% autoplot()
p2 <- partial(house_xgb_final, pred.var = vi_scores[[2, 1]],
              train = X, type = "regression") %>% autoplot()
p3 <- partial(house_xgb_final, pred.var = vi_scores[[3, 1]],
              train = X, type = "regression") %>% autoplot()
p4 <- partial(house_xgb_final, pred.var = vi_scores[[4, 1]],
              train = X, type = "regression") %>% autoplot()
grid.arrange(p1, p2, p3, p4, ncol = 2)
```

```
## Warning: Use of `object[[1L]]` is discouraged. Use `.data[[1L]]` instead.

## Warning: Use of `object[["yhat"]]` is discouraged. Use `.data[["yhat"]]`
## instead.

## Warning: Use of `object[[1L]]` is discouraged. Use `.data[[1L]]` instead.

## Warning: Use of `object[["yhat"]]` is discouraged. Use `.data[["yhat"]]`
## instead.

## Warning: Use of `object[[1L]]` is discouraged. Use `.data[[1L]]` instead.

## Warning: Use of `object[["yhat"]]` is discouraged. Use `.data[["yhat"]]`
## instead.

## Warning: Use of `object[[1L]]` is discouraged. Use `.data[[1L]]` instead.

## Warning: Use of `object[["yhat"]]` is discouraged. Use `.data[["yhat"]]`
## instead.
```



```r
toc()
```

```
## 185.627 sec elapsed
```

The marginal effect of living space and living space of 15 neighbors appear similar to those in RF.

The marginal effect of latitude seems the same as in RF except that the mean of predicted prices immediately decreases after the summit.

The living space above grade exerts a considerable influence on prices when living space is approximately 1,200 and 5,000 square feet.

Comparison

Variance importance

(6) The important variables in the three models are alike. Similarities include that they all regard `lattitude` and `sqft_living` as of the greatest importance. `nn_sqft_living` ranks the third, but much less important than `lattitude` or `sqft_living`.

The difference of the three models are that Basic GBM values `longitude` compared with `sqft_above` in RF and Xgboost, and that RF values `lattitude` as the top compared with `sqft_living` in Basic GBM and Xgboost.

Final model

(7) Xgboost has the smallest cross-validated RMSE on the training data among RF (0.07912), GBM (0.07294), and Xgboost (0.07198). Hence, we use all training data to refit the Xgboost model `house_xgb_final` in document "p3 XGB" as our final model.

```
house_xgb_final
```

```
## ##### xgb.Booster
## raw: 15.9 Mb
## call:
##   xgb.train(params = params, data = dtrain, nrounds = nrounds,
##     watchlist = watchlist, verbose = verbose, print_every_n = print_every_n,
##     early_stopping_rounds = early_stopping_rounds, maximize = maximize,
##     save_period = save_period, save_name = save_name, xgb_model = xgb_model,
##     callbacks = callbacks, objective = "reg:squarederror")
## params (as set within xgb.train):
##   eta = "0.01", max_depth = "7", min_child_weight = "2", subsample = "0.8", colsample_bytree = "0.9"
## xgb.attributes:
##   best_iteration, best_msg, best_ntreelimit, best_score, niter
## callbacks:
##   cb.evaluation.log()
##   cb.early.stop(stopping_rounds = early_stopping_rounds, maximize = maximize,
##     verbose = verbose)
## # of features: 19
## niter: 2500
## best_iteration : 2500
## best_ntreelimit : 2500
## best_score : 0.0539
## best_msg : [2500]    train-rmse:0.053900
## nfeatures : 19
## evaluation_log:
##     iter train_rmse
##        1   5.119303
##        2   5.068162
## ---
##     2499   0.053903
##     2500   0.053900
```

Prediction on test data

```
house_test <- read.csv("house_test.csv", stringsAsFactors = TRUE)
dim(house_test)  # dataset: house_test, response: price
```

```
## [1] 4324   20
```

```
Y_pred <- predict(house_xgb_final, newdata = data.matrix(house_test[!names(house_test) %in% c("price")]
summary(Y_pred)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   5.148   5.511   5.657   5.669   5.798   6.657
```

RMSE is 0.01328, less than the cross-validated RMSE

```
sqrt(mean(Y_pred - house_test$price))
```

```
## [1] 0.01327817
```