Bagging

Algorithm class: Non-parametric

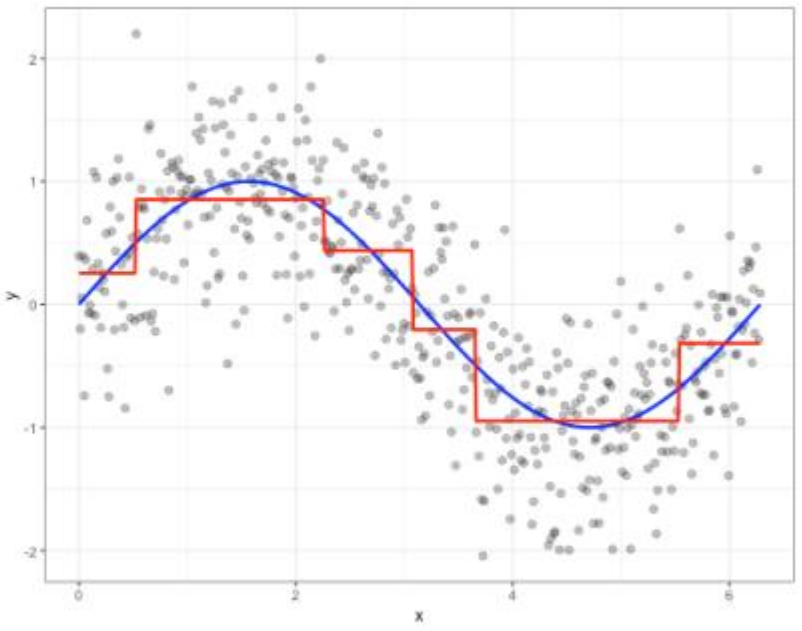
Mechanism: Average predictions of many trees

Applicable: Both classification and regression problem

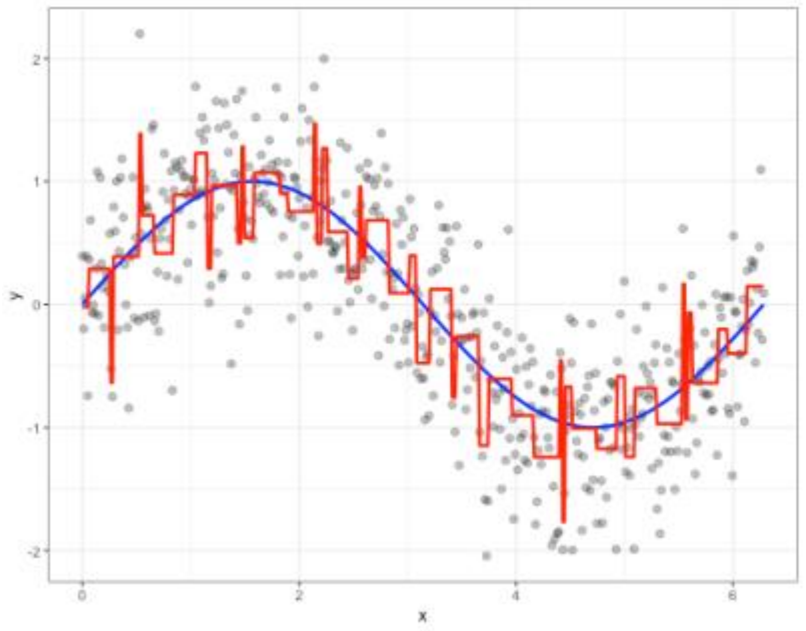
By model averaging, Bagging helps reduce variance and minimize overfitting

The problem with single trees

Single shallow trees are poor predictors



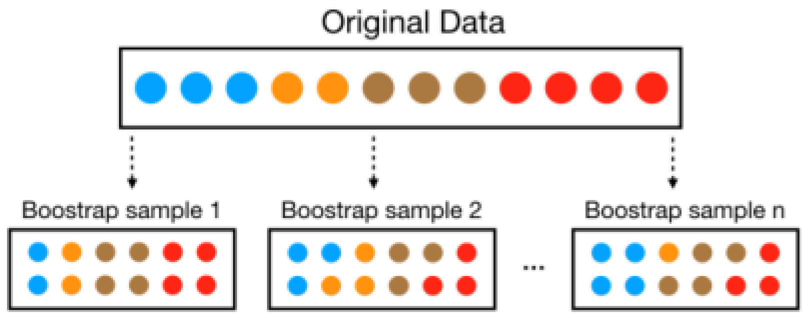
Single deep trees have high variance



Bagging uses this high variance to our advantage

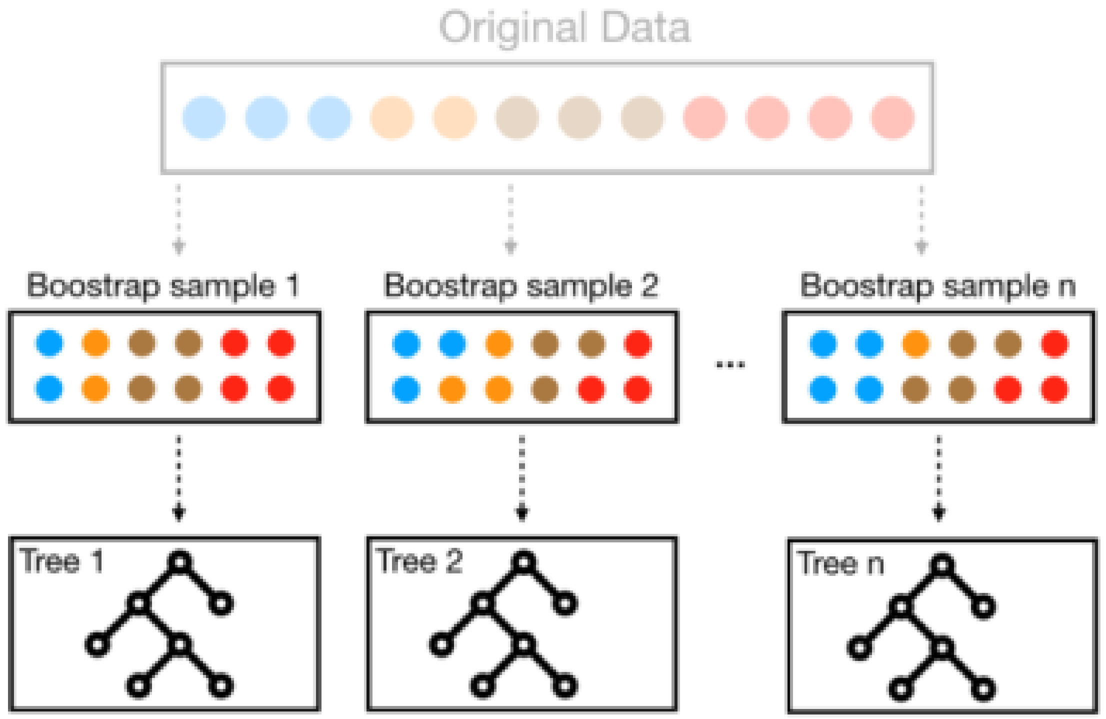
Bootstrap Aggregating – wisdom of the crowd

1. Create bootstrap samples from training data
2.
3.



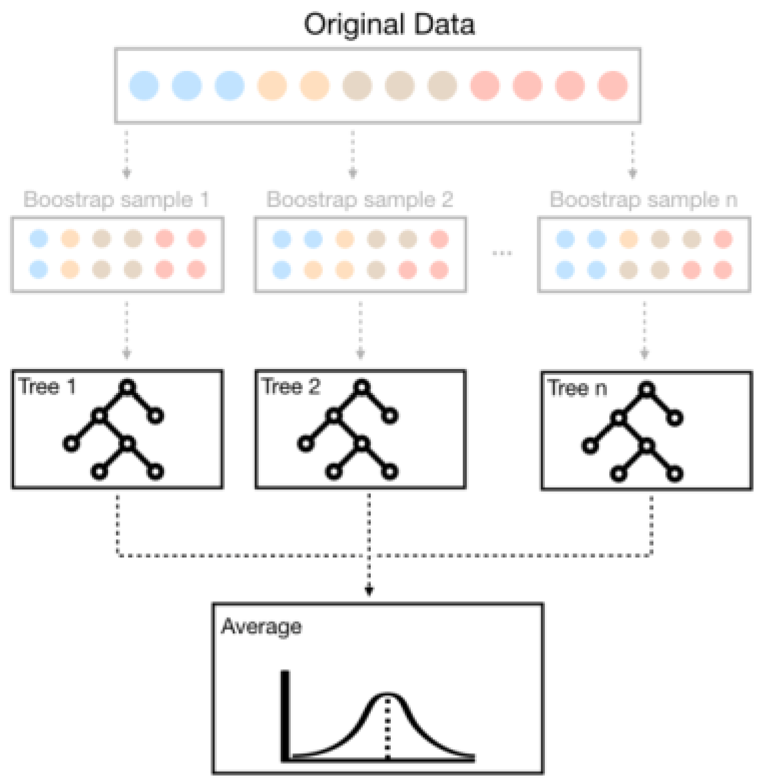
Bootstrap Aggregating – wisdom of the crowd

1. Create bootstrap samples from training data
2. Fit an overgrown tree to each resampled dataset
3.



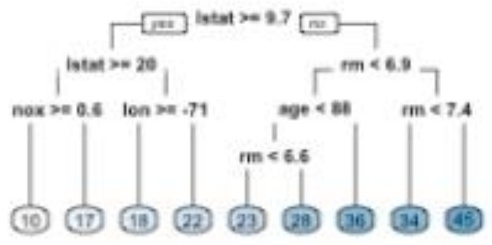
Bootstrap Aggregating – wisdom of the crowd

1. Create bootstrap samples from training data
2. Fit an overgrown tree to each resampled dataset
3. Average predictions

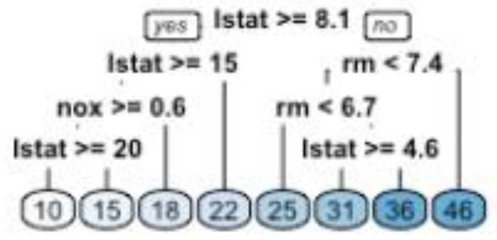


However, there is a problem

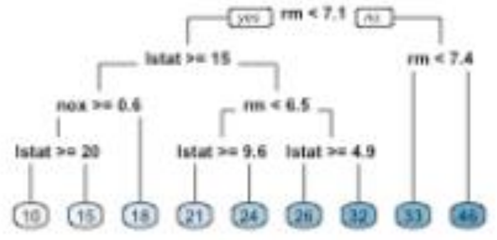
Decision Tree 1



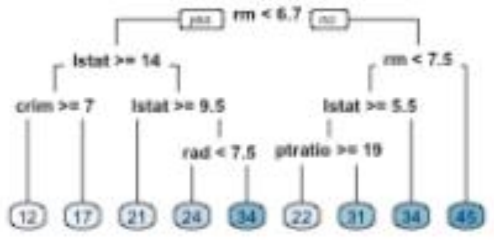
Decision Tree 2



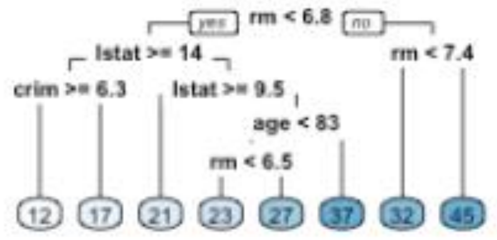
Decision Tree 3



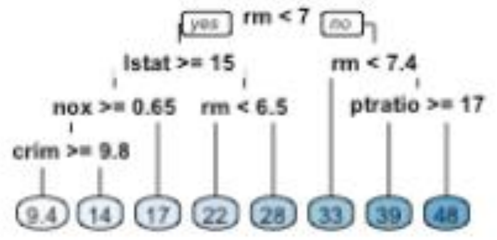
Decision Tree 4



Decision Tree 5



Decision Tree 6



which prevents bagging from optimally reducing variance of the predictive values

Random Forest

Algorithm class: Non-parametric

Mechanism: Average predictions of many trees (de-correlated)

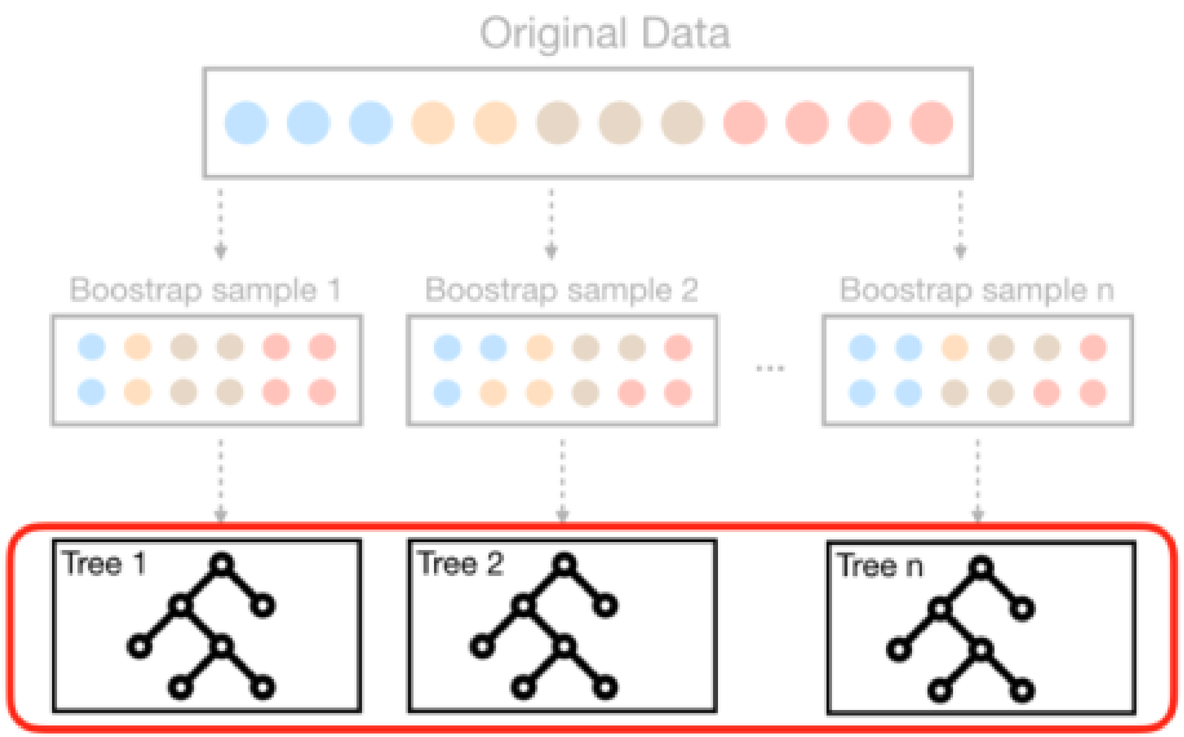
Applicable: Both classification and regression problem

Random Forest: generalization of Bagging, performance typically better

Idea

Split variable randomization

- Follow a similar bagging process but …



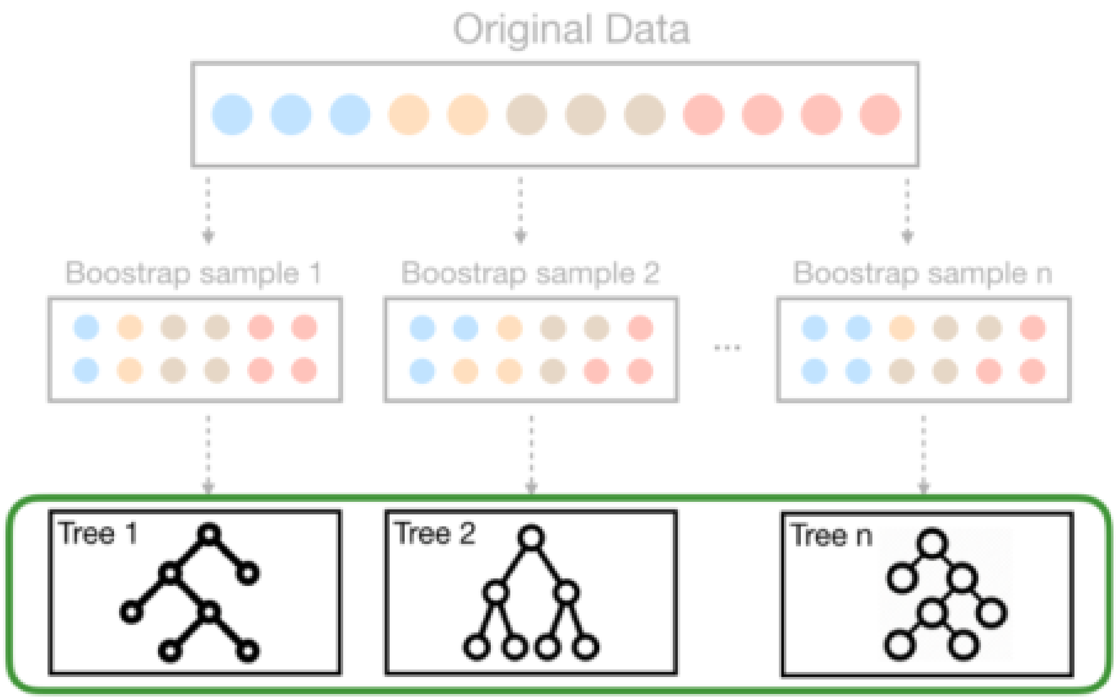
Trees produced by bagging

Idea

Split variable randomization

- Follow a similar bagging process but …
- Each time **a split is to be performed**,

- regression trees: m = p/3
- classification trees: m=$\sqrt{p}$
- m is commonly referred to as mtry



Trees produced by RF

Random Forest

Essentially

- Bagging introduces randomness into **rows of the data**

- Random forest introduces randomness into ________________________________
  - This provides a more diverse set of trees that almost always lowers the prediction error

Out of bag (OOB) Performance

- For large enough N, an original data point has a 63% probability of ending up in any bootstrap sample
  - i.e. 37% of the observations NOT used in for a particular tree

- These OOB observations: model performance assessment

- Let’s now look at how to implement RF

Implementation of Random Forest

- Simple way: ranger, full grid search
- More advanced: h2o, random grid search & early stopping rules

Ames Housing Example (RF), with ranger package

Direct implementation of RF, no tuning

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

# train a default random forest model
ames_rf1 <- ranger(
    Sale_Price ~ .,
 ···data = ames_train,
    mtry = floor(n_features / 3),
    respect.unordered.factors = "order",
    seed = 123
)

# get OOB RMSE
(default_rmse <- sqrt(ames_rf1$prediction.error))
# 25488.39
```

For regression tree

Baseline RF model, RMSE ≈ 25,500

Next, we will look at how to tune hyperpara. to improve the model

Random Forest

Tuning Hyperparameters

Random forests provide good "out-of-the-box" performance but there are a few hyperpara. we can tune to increase performance.

# Trees Mtry	Typically have the largest impact on predictive accuracy
Min node size/Max depth (Tree Complexity) Sampling scheme	Some impact on predictive accuracy, but can increase computational efficiency

Random Forest

Tuning Hyperparameters: # Trees

Need to be sufficiently large: stabilize error rate

Rule of thumb: start with 10p trees and adjust as necessary

More trees provide robust and stable error estimates and variable importance measures

Computation time increases linearly with the number of trees

Random Forest

Tuning Hyperparameters: mtry (#split vars)

Balance low tree correlation and reasonable predictive strength

Rule of thumb default:
Regression default:
Classification default:

Start with 5 values evenly spaced from 2 to p, including the default rule-of-thumb value

Few relevant predictors: Should we ↑ or ↓ mtry?

Random Forest

Tuning Hyperparameters: Min node size/Max depth (Tree Complexity)

Control the complexity of individual trees

Rule of thumb:
Regression default: 5
Classification default: 1
Start with 3 values (1,5,10)

If run time is a concern, can ↓run time substantially by ↑node size (tradeoff between runtime and model accuracy)

Random Forest

Tuning Hyperparameters: Sampling scheme

1. Sample size (default: 100%)
2. Sample with replacement / without replacement
(default: with replacement)

Rationale:
↓ Sample size reduces between-tree correlation
(Smaller sample size produces more random trees)

Rule of thumb:
3-4 values of sample sizes ranging from 25-100%
Try both sampling with/without replacement

Ames Housing Example (RF), with ranger package (cont'd)

Tuning Strategy Illustration

```
# The code below searches across 120 combinations of hyperparameter settings.

# create hyperparameter grid
hyper_grid <- expand.grid(
    mtry = floor(n_features * c(.05, .15, .25, .333, .4)),
    min.node.size = c(1, 3, 5, 10),
    replace = c(TRUE, FALSE),
    sample.fraction = c(.5, .63, .8),
    rmse = NA
)
```

mtry

Min node size

Sample scheme

Note: expand.grid returns a dataframe with columns mtry, min.node.size, replace, sample.fraction, rmse (values to be filled)

Ames Housing Example (RF), with ranger package (cont'd)

Tuning Strategy Illustration

#trees
mtry
Node size
Sample scheme

```r
# execute full cartesian grid search
for(i in seq_len(nrow(hyper_grid))) {
    # fit model for ith hyperparameter combination
    fit <- ranger(
        formula         = Sale_Price ~ .,
        data            = ames_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)
}

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

Fills rmse in hyper_grid (created by expand.grid)

Ames Housing Example (RF), with ranger package (cont'd)

Tuning Strategy Illustration

RMSE slightly improvement over baseline model

Observations

1. Default mtry = 26 (#features/3) nearly sufficient

2. Smaller node size performs better (deeper tree)

3. Sample <100% and sample without replacement consistently performs better

- **Probably due to data having a lot of high-cardinality & imbalanced categorial features**

%improvement of RMSE w.r.t. baseline model

```
   mtry min.node.size replace sample.fraction     rmse perc_gain
1    26             1   FALSE             0.8 24713.06  3.041873
2    26             3   FALSE             0.8 24847.98  2.512570
3    20             3   FALSE             0.8 24917.05  2.241554
4    20             1   FALSE             0.8 24929.10  2.194284
5    32             5   FALSE             0.8 24940.14  2.150967
6    32             1   FALSE             0.8 24978.78  1.999392
7    32             3   FALSE             0.8 24990.83  1.952085
8    26             5   FALSE             0.8 25004.10  1.900044
9    20             5   FALSE             0.8 25028.46  1.804464
10   12             1   FALSE             0.8 25029.93  1.798693
```

Ames Housing Example (RF), with h2o package

Benefits of h2o package:

- Random grid search
  - Full Cartesian hyperpara. search can be computationally expensive
  - Randomly jump from one random para. combination to another
- Can specify early stopping rules
  - E.g. #models trained >= threshold, certain runtime elapses

```r
# convert training data to h2o object
train_h2o <- as.h2o(ames_train)

# set the response column to Sale_Price
response <- "Sale_Price"

# set the predictor names
predictors <- setdiff(colnames(ames_train), response)
```

Ames Housing Example (RF), with h2o package

Baseline h2o RF

- Syntax and result very similar to the baseline ranger RF

```
# The following fits a default random forest model with h2o
# Runtime: 1 minute on i7 CPU
h2o_rf1 <- h2o.randomForest(
    x = predictors,
    y = response,
    training_frame = train_h2o,
    ntrees = n_features * 10,
    seed = 123
)

h2o_rf1

#H2ORegressionMetrics: drf
#** Reported on training data. **
#    ** Metrics reported on Out-Of-Bag training samples **
#
#    MSE:  626755219
#RMSE:  25035.03
#MAE:  15238.9
#RMSLE:  0.1415424
#Mean Residual Deviance :  626755219
```

Similar to baseline RF using ranger

Ames Housing Example (RF), with h2o package

h2o RF with Random Grid Search + Early Stopping Rule (Optional)

```
# To execute a grid search in h2o we need our hyperparameter grid to be a list.
# For example, the following code searches a larger grid space than before with
# a total of 240 hyperparameter combinations.

# We then create a random grid search strategy that will stop if none of the
# last 10 models have managed to have a 0.1% improvement in MSE compared to
# the best model before that.
# If we continue to find improvements then we cut the grid search off after 300 seconds
```

Recall in ranger,
we build the hyperpara. grid using the following syntax

```
hyper_grid <- expand.grid(
    mtry = floor(n_features * c(.05, .15, .25, .333, .4)),
    min.node.size = c(1, 3, 5, 10),
    replace = c(TRUE, FALSE),
    sample.fraction = c(.5, .63, .8),
    rmse = NA
)
```

Ames Housing Example (RF), with h2o package

h2o RF with Random Grid Search + Early Stopping Rule (Optional)

```r
# To execute a grid search in h2o we need our hyperparameter grid to be a list.
# For example, the following code searches a larger grid space than before with
# a total of 240 hyperparameter combinations.

# We then create a random grid search strategy that will stop if none of the
# last 10 models have managed to have a 0.1% improvement in MSE compared to
# the best model before that.
# If we continue to find improvements then we cut the grid search off after 300 seconds

# hyperparameter grid
hyper_grid <- list(
    mtries = floor(n_features * c(.05, .15, .25, .333, .4)),
    min_rows = c(1, 3, 5, 10),
    max_depth = c(10, 20, 30),
    sample_rate = c(.55, .632, .70, .80)
)
# random grid search strategy
search_criteria <- list(
    strategy = "RandomDiscrete",
    stopping_metric = "mse",
    stopping_tolerance = 0.001,   # stop if improvement is < 0.1%
    stopping_rounds = 10,         # over the last 10 models
    max_runtime_secs = 60*5      # or stop search after 5 min.
)
```

In h2o, we use a list

Ames Housing Example (RF), with h2o package

h2o RF with Random Grid Search + Early Stopping Rule (Optional)

```r
# To execute a grid search in h2o we need our hyperparameter grid to be a list.
# For example, the following code searches a larger grid space than before with
# a total of 240 hyperparameter combinations.

# We then create a random grid search strategy that will stop if none of the
# last 10 models have managed to have a 0.1% improvement in MSE compared to
# the best model before that.
# If we continue to find improvements then we cut the grid search off after 300 seconds

# hyperparameter grid
hyper_grid <- list(
    mtries = floor(n_features * c(.05, .15, .25, .333, .4)),
    min_rows = c(1, 3, 5, 10),
    max_depth = c(10, 20, 30),
    sample_rate = c(.55, .632, .70, .80)
)

# random grid search strategy
search_criteria <- list(
    strategy = "RandomDiscrete",
    stopping_metric = "mse",
    stopping_tolerance = 0.001,   # stop if improvement is < 0.1%
    stopping_rounds = 10,         # over the last 10 models
    max_runtime_secs = 60*5      # or stop search after 5 min.
)
```

In h2o, we use a list

Min node size

Random grid-search strategy: "RandomDiscrete"

- **Randomly jump from one hyperpara. combination to another**

Early stopping criteria for grid-search

- **Stop if the last 10 RF models do NOT improve RMSE by 0.1%**
- **Stop if run time > 5 mins**

Feature Interpretation

For RF: 2 approaches for variable importance

At this point, do not need to know the details, just know there are 2 measures

Impurity (Same as CART)

- Based on the average total reduction in MSE

Permutation (Applicable for All ML models, will talk about it in more details)

- Permute a feature to a random value, see how it affects MSE

Feature Interpretation

E.g. using ranger

```
# re-run model with impurity-based variable importance
rf_impurity <- ranger(
    formula = Sale_Price ~ .,
    data = ames_train,
    num.trees = 2000,
    mtry = 32,
    min.node.size = 1,
    sample.fraction = .80,
    replace = FALSE,
    importance = "impurity",
    respect.unordered.factors = "order",
    verbose = FALSE,
    seed  = 123
)
```

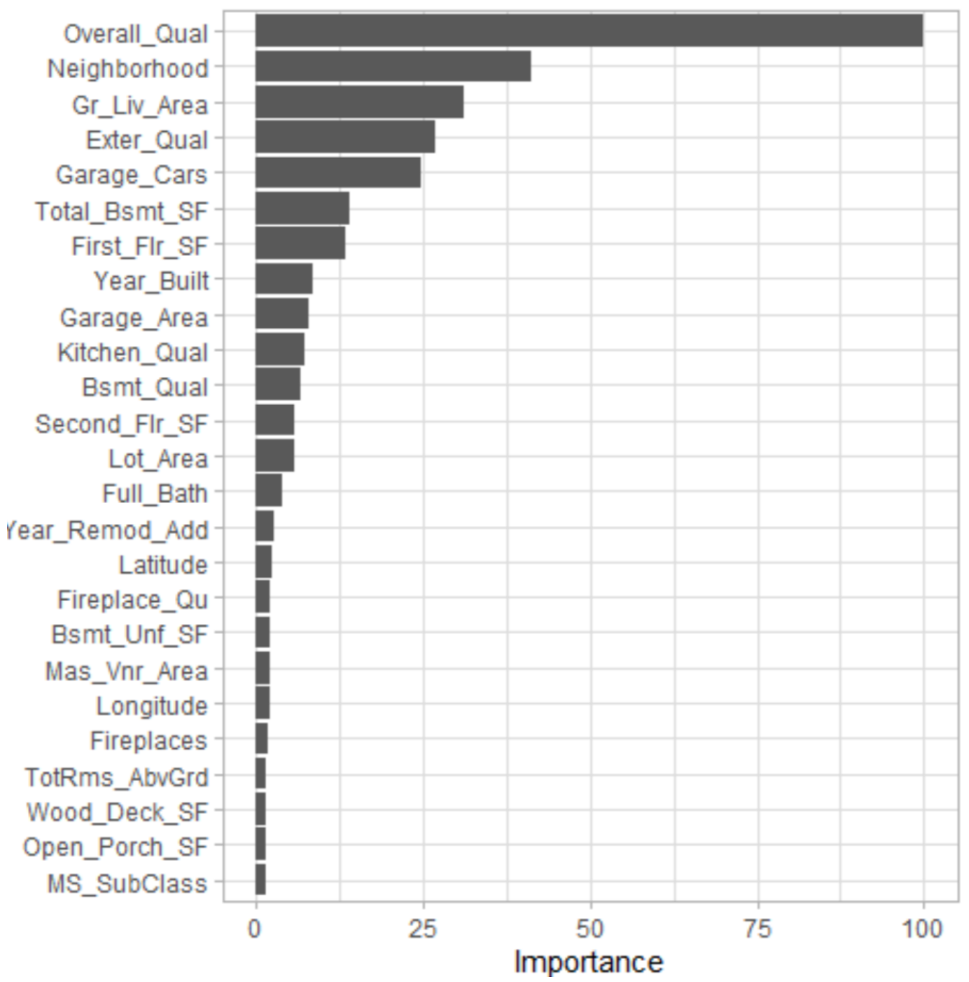
```
# re-run model with permutation-based variable importance
rf_permutation <- ranger(
    formula = Sale_Price ~ .,
    data = ames_train,
    num.trees = 2000,
    mtry = 32,
    min.node.size = 1,
    sample.fraction = .80,
    replace = FALSE,
    importance = "permutation",
    respect.unordered.factors = "order",
    verbose = FALSE,
    seed  = 123
)
```

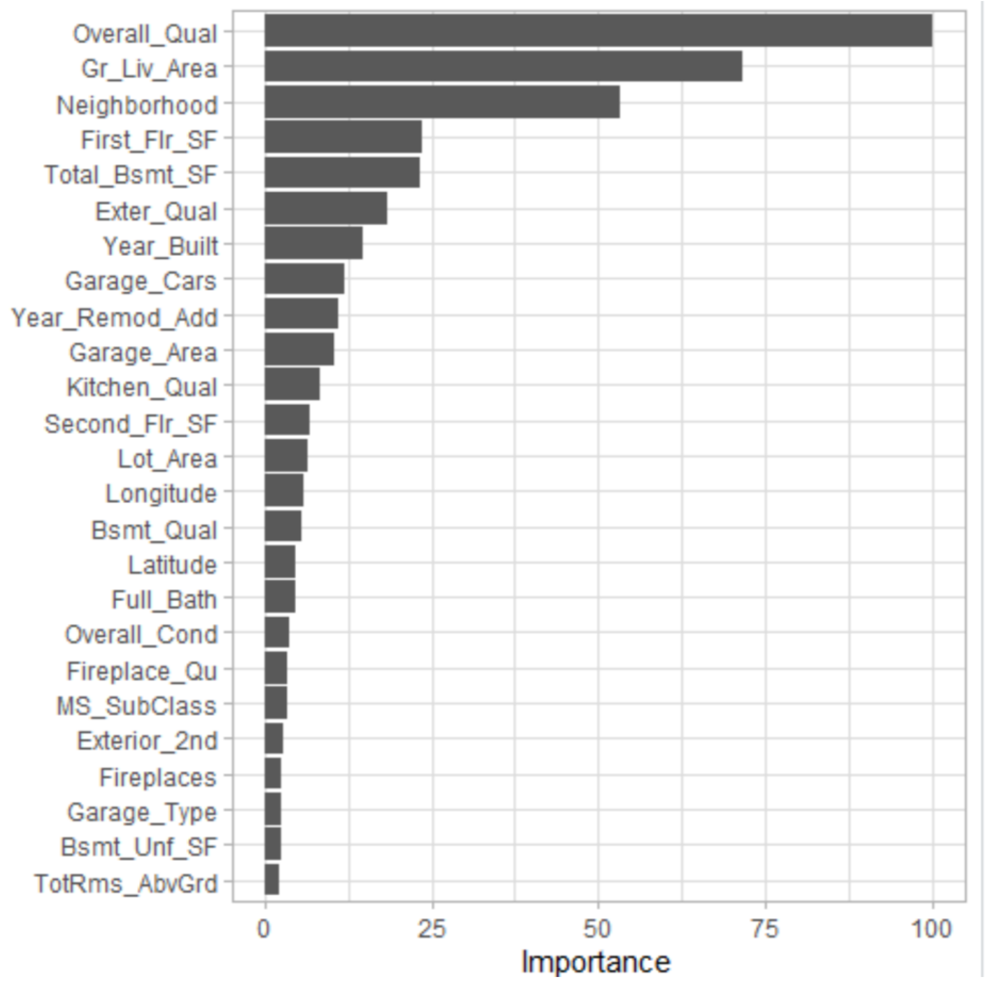
```
p1 <- vip::vip(rf_impurity, num_features = 25, scale = TRUE)
p2 <- vip::vip(rf_permutation, num_features = 25, scale = TRUE)

gridExtra::grid.arrange(p1, p2, nrow = 1)
```

Feature Interpretation

Typically, similar variables at the top between the two approaches

- Can conclude top 3 important vars: Overall_Qual, Gr_Liv_Area, Neighborhood





Summary

Method	Hyperpara	Unique features	RMSE	Package Demonstrated	
CART	• Tree depth • Node size • cp	Simple to interpret	-	rpart caret method = "rpart"	
Random Forest	# Trees (~10p) Mtry (#split vars, p/3 or $\sqrt{p}$) Node size (Tree Complexity) Sampling scheme • Sample size • Sample with/without replacement	Subsample rows/cols Early Stopping (in adding trees)	~24000	ranger h2o Algorithm = "randomForest"	

End

Ames Housing Example (RF), with h2o package

h2o RF with Random Grid Search + Early Stopping Rule (Optional)

```
# perform grid search
# The following executes the grid search with early stopping turned on.
# The early stopping we specify below in h2o.grid() will stop growing an individual random
# experienced at least a 0.05% improvement in the overall OOB error in the last 10 trees.
# Runtime: 5 minutes

random_grid <- h2o.grid(
    algorithm = "randomForest",
    grid_id = "rf_random_grid",
    x = predictors,
    y = response,
    training_frame = train_h2o,
    hyper_params = hyper_grid,
    ntrees = n_features * 10,
    seed = 123,
    stopping_metric = "RMSE",
    stopping_rounds = 10,
    stopping_tolerance = 0.005,
    search_criteria = search_criteria
)
```

Early stopping criteria for building one RF

- **Stop if the last 10 trees added do NOT improve RMSE by 0.5%**

```
# stop if last 10 trees added
# don't improve RMSE by 0.5%
# https://www.rdocumentation.org/packages/h2o/v
```

Ames Housing Example (RF), with h2o package

h2o RF with Random Grid Search + Early Stopping Rule (Optional)

```r
# collect the results and sort by our model performance metric of choice
random_grid_perf <- h2o.getGrid(
    grid_id = "rf_random_grid",
    sort_by = "mse",
    decreasing = FALSE
)
random_grid_perf
```

Ames Housing Example (RF), with h2o package

h2o RF with Random Grid Search + Early Stopping Rule (Optional)

```
#H2O Grid Details
#================
#
#    Grid ID: rf_random_grid
#Used hyper parameters:
#     -  max_depth
#-  min_rows
#-  mtries
#-  sample_rate
#Number of models: 66
#Number of failed models: 0
#
#Hyper-Parameter Search Summary: ordered by increasing mse
#max_depth min_rows mtries sample_rate                 model_ids                 mse
#1         30      1.0     20         0.8 rf_random_grid_model_57   6.0865378782043E8
#2         20      1.0     20         0.8 rf_random_grid_model_31 6.087217272346667E8
#3         20      1.0     26         0.8 rf_random_grid_model_32 6.179447344528593E8
#4         20      1.0     26         0.7 rf_random_grid_model_66 6.283513049703969E8
#5         30      1.0     32       0.632 rf_random_grid_model_64 6.373143927266866E8
#
#---
```

Note: with early stopping, results may NOT be the same (#models searched in laptops of different speed will be different)

Assessed 66 models, best CV RMSE = 24670

This is near-optimal, and the random grid-search is more efficient