ADVANCED TREES

Applied Analytics: Frameworks and Methods 1

Outline

- Evaluation of Trees
- Model Hyperparameters
- Cross-validation
- Bagging
- Random Forests
- Boosting

Advantages and Disadvantages of Trees

- Trees are very easy to explain to people. In fact, they are even easier to explain than linear regression!
- Some people believe that decision trees more closely mirror human decision-making than do regression and classification approaches.
- Trees can be displayed graphically, and are easily interpreted even by a non-expert (especially if they are small).
- Trees can easily handle qualitative predictors without the need to create dummy variables.
- Trees are prone to overfitting train data
- Unfortunately, trees generally do not have the same level of predictive accuracy as some of the other prediction and classification approaches.

Making Trees better

- Disadvantage: Trees are prone to overfitting train data
- Solution: Use cross-validation to optimize complexity. This involves picking good model hyperparameters.

- Disadvantage: Trees generally do not have the same level of predictive accuracy as some of the other prediction and classification approaches.
- Solution: Predictive performance of trees can be substantially improved by aggregating many decision trees.

Model Hyperparameters

- Parameters are estimated during model training. On the other hand, hyperparameters control the learning process. For e.g., in a regression tree "minbucket" controls the size of the tree by setting a minimum for the number of observations an interior node must have.
- Specifically, the implementation of Trees in library(rpart) has a number of hyperparameters.
  - *rpart.control(minsplit = 20, minbucket = round(minsplit/3), cp = 0.01,*
    maxcompete = 4, maxsurrogate = 5, usesurrogate = 2, xval = 10,
    surrogatestyle = 0, maxdepth = 30, ...)
- Default values for model hyperparameters provide a good starting point for modeling. However, tuning these hyperparameters can greatly improve model performance.

Process of Tuning Model Hyperparameters

1. Identifying hyperparameter(s) to tune
2. Specifying range of hyperparameter values to evaluate. This is also known as grid search.
3. Train a model on each combination of hyperparameters
4. Measure performance
5. Choose hyperparameter values for model with best performance

Process of Tuning Model Hyperparameters

1. Identify hyperparameter(s) to tune
   – *One or more hyperparameters may be selected. For a regression tree, one could use the complexity parameter, cp, which controls the size of the tree.*
2. Specifying range of hyperparameter values to evaluate. This is also known as grid search.
3. Train a model on each combination of hyperparameters
4. Measure performance
5. Choose hyperparameter values for model with best performance

Process of Tuning Model Hyperparameters

1. Identify hyperparameter(s) to tune
2. Specifying range of hyperparameter values to evaluate. This is also known as grid search.
   - *For a regression tree, say 41 values of cp from 0 to 0.16 in intervals of 0.004.*
   - *Values to evaluate may be informed by experience. Alternatively, one can use a trial and error approach. E.g., begin with a broad grid search (e.g., intervals of 0.01) and narrow it down later to focus on a specific range (e.g., intervals of 0.001).*
3. Train a model on each combination of hyperparameters
4. Measure performance
5. Choose hyperparameter values for model with best performance

Process of Tuning Model Hyperparameters

1. Identify hyperparameter(s) to tune
2. Specifying range of hyperparameter values to evaluate. This is also known as grid search.
3. Train a model on each combination of hyperparameters
4. Measure performance
5. Choose hyperparameter values for model with best performance

Process of Tuning Model Hyperparameters

1. Identify hyperparameter(s) to tune
2. Specifying range of hyperparameter values to evaluate. This is also known as grid search.
3. Train a model on each combination of hyperparameters
4. Measure performance
   - *Training error is a poor guide for test error*
   - *Test sample can only be used for evaluating the final model, not a series of models with different combinations of hyperparameters*
   - *Solution is to use a resampling procedure called cross-validation*
5. Choose hyperparameter values for model with best performance

Process of Tuning Model Hyperparameters

1. Identify hyperparameter(s) to tune
2. Specifying range of hyperparameter values to evaluate. This is also known as grid search.
3. Train a model on each combination of hyperparameters
4. Measure performance
5. Choose hyperparameter values for model with best performance
   – *Select the hyperparameter values for the model with lowest cross-validation error*

CROSS-VALIDATION

Optimal Complexity

- Model is built on training data. The error of this model is almost always going to be less than that on a new sample. So, there are two methods of estimating error on new data.
  - *Apply a penalty on the training error rate to estimate the test error rate. E.g., AIC, BIC, Mallow's Cp and Adjusted R2.*
  - *Cross-validation*

Cross-validation

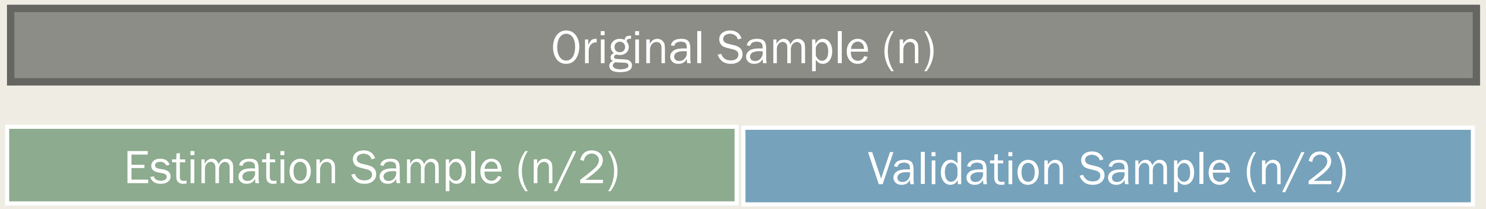
- Cross-validation is part of a general technique called resampling. Bootstrapping, another resampling approach will be examined later
- Resampling methods involve repeatedly drawing samples from a training set and refitting a model of interest on each sample in order to obtain more information about the fitted model
  - *Model Assessment: estimate test error rates*
  - *Model Selection: select the appropriate level of model flexibility*
- Resampling method in general are computationally expensive! But these days we have powerful computers.
- Cross-validation methods
    - Validation Set Approach
    - K-fold Cross-Validation
    - Leave One Out Cross-Validation (LOOCV)

Validation Set Approach

1. Randomly split the data into a estimation set and a validation set.
2. Estimate the model using the estimation sample.
3. Apply the estimated model to the validation sample
4. Error in the validation-set provides an estimate of error in a new sample.

The Validation Set Approach

1. Randomly split sample into two halves: estimation and validation samples
2. Build model on estimation sample. Assess performance on validation sample.



Evaluation of Validation Set Approach

- Simple approach.
- Not computationally intensive
- The validation estimate of the test error can be highly variable, depending on precisely which observations are included in the training set and which observations are included in the validation set.
- In the validation approach, only a subset of the observations — those that are included in the training set rather than in the validation set — are used to fit the model. This suggests that the validation set error may tend to overestimate the test error for the model fit on the entire data set.

Illustration: Validation Set Approach

- Use Auto data in ISLR package
- Run 8 regressions of horsepower to predict mpg, each with a different degree of horsepower
  - *mpg ~ horsepower*
  - *mpg ~ horsepower^2*
  - *mpg~ horsepower^3 ……*
- Plot polynomial degree of model vs. test-MSE

Illustration: Validation Set Approach
Test-set Error vs. Degree for one split

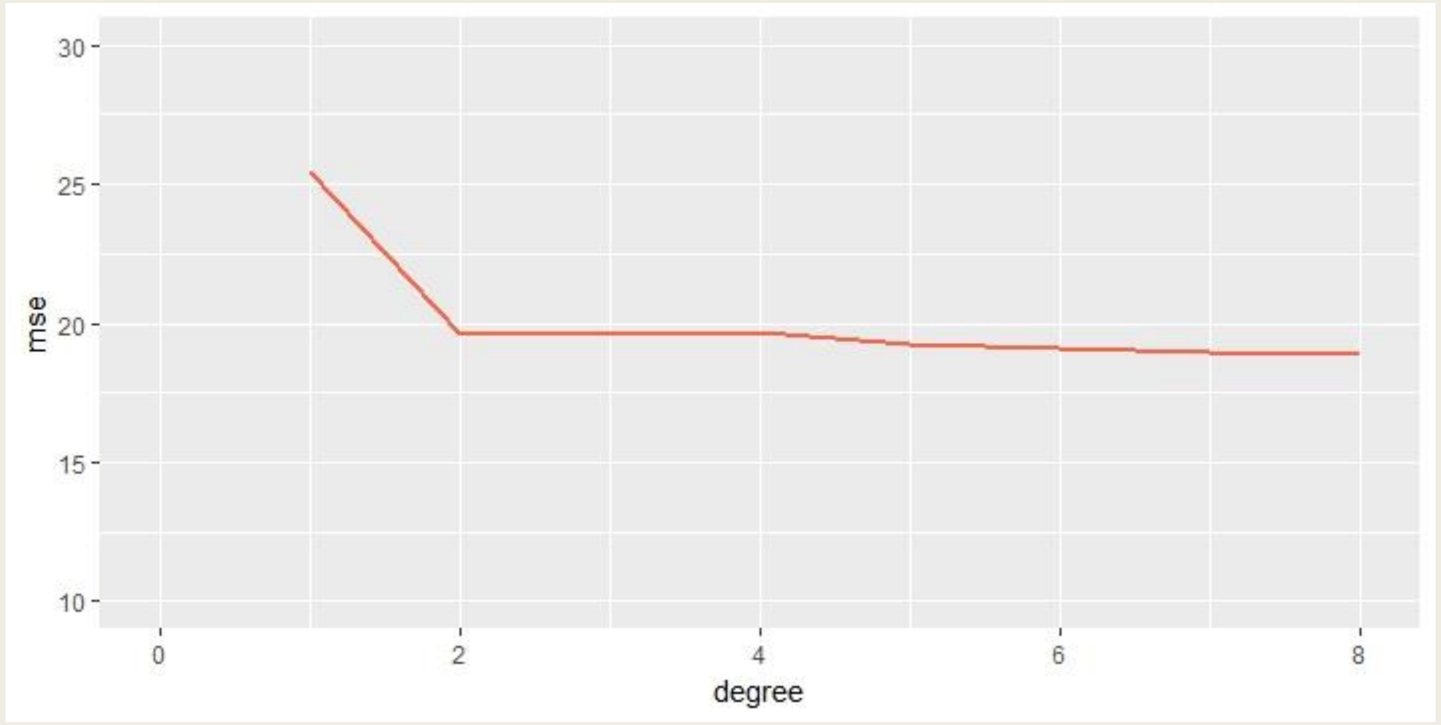
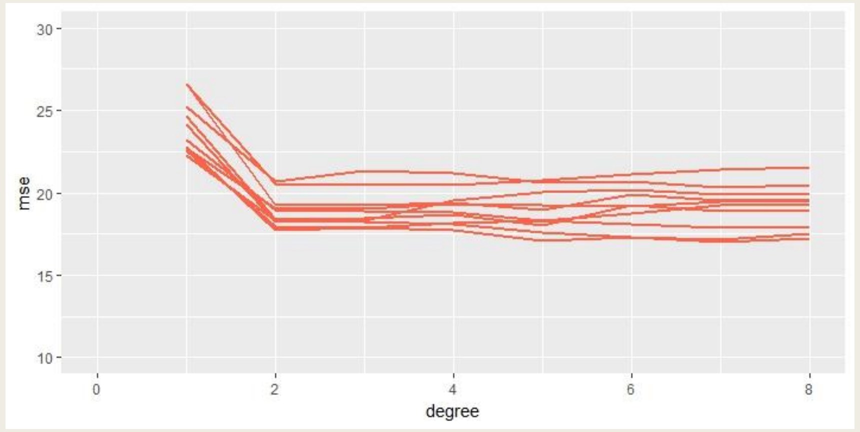


Illustration: Validation Set Approach
Repeat for Ten Different Splits

- Every train-test split is bound to be different. Here, we split, train model, estimate test-set mse, and plot results 10 times and 20 times.
- Each line on following charts indicates result of a different train-test split.



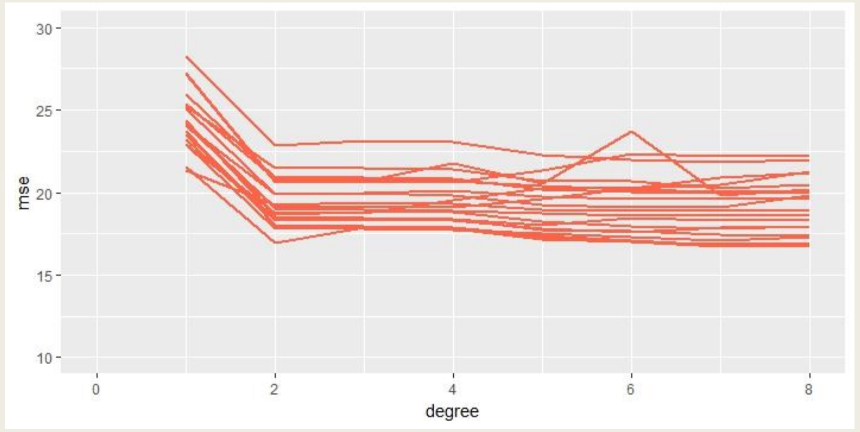
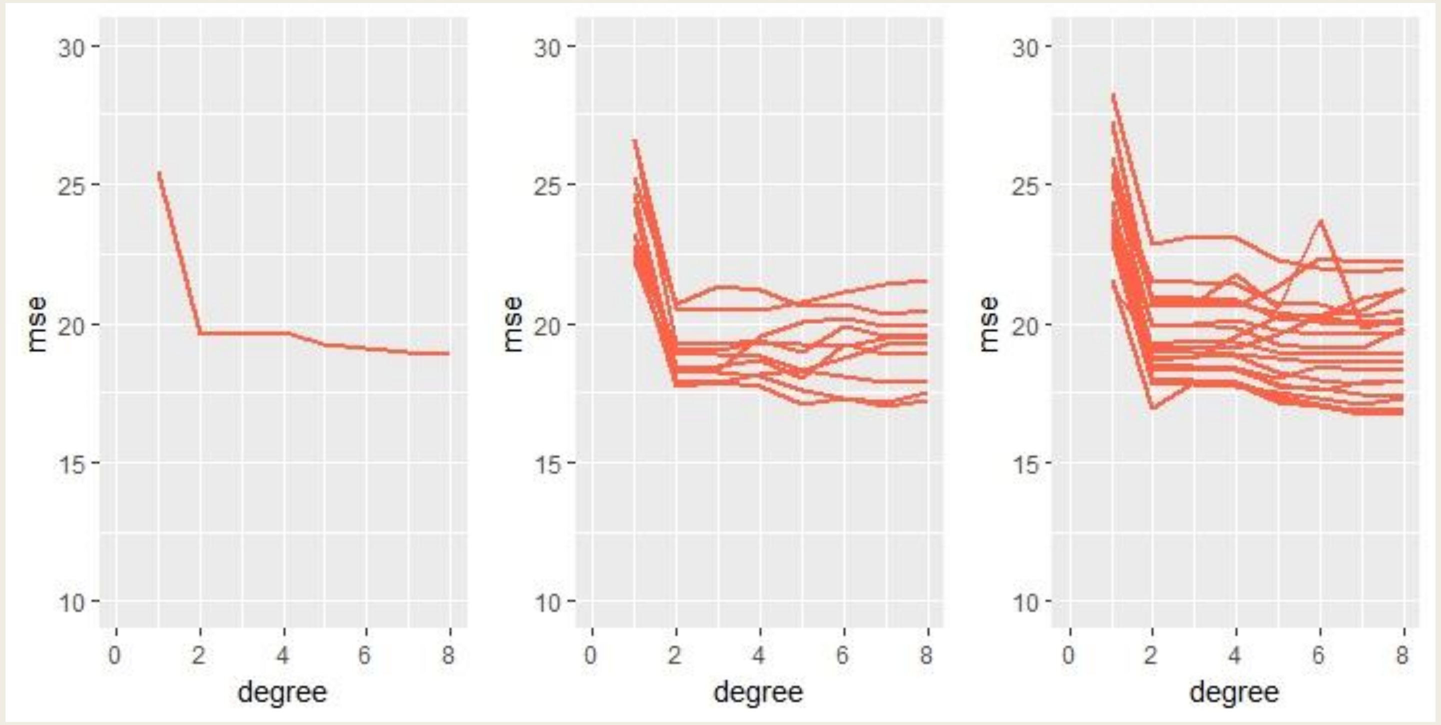


Illustration: Validation Set Approach



K-Fold Cross-Validation

- Widely used approach for estimating test error.
- Idea is to randomly divide the data into k equal-sized parts. We leave out part k, fit the model to the other $k - 1$ parts (combined), and then obtain predictions for the left-out kth part.
- This is done in turn for each part $k = 1, 2, \ldots K$, and then the results are combined.

K-Fold Cross-Validation

- Divide data into k equal sized parts (k=5 in figure below)
- Use four folds to predict the fifth one.

Train Sample				
Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
Validation	Estimation	Estimation	Estimation	Estimation
Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
Estimation	Validation	Estimation	Estimation	Estimation
Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
Estimation	Estimation	Validation	Estimation	Estimation
Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
Estimation	Estimation	Estimation	Validation	Estimation
Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
Estimation	Estimation	Estimation	Estimation	Validation

Computation

- Let the K parts be C_1, C_2, . . . C_K, where C_k denotes the indices of the observations in part k. There are n_k observations in part k: if N is a multiple of K, then n_k = n/K.
- Compute
  - $CV_k = \sum_{k=1}^{k} \frac{n_k}{n} MSE_k$
- When k = n, we get n-fold or leave-one-out cross validation

Evaluation of k-fold cross Validation

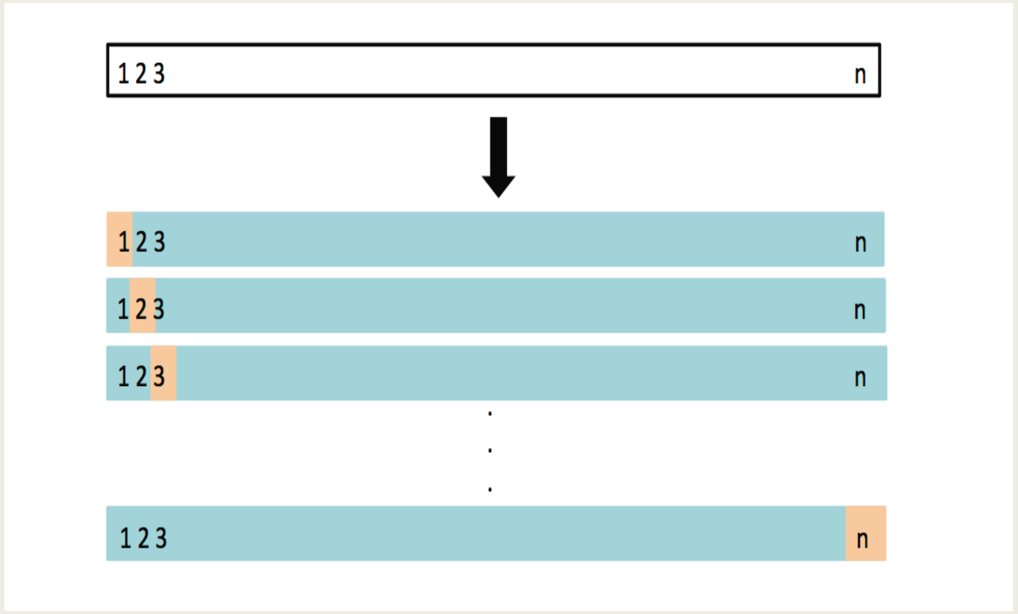
- Less bias than Validation because model is trained on a larger sample (for 5-fold, 80% compared to 50% for Validation)
- Less variance because test statistic is averaged k times.
- Computationally more intensive than Validation but less than LOOCV
- Works best for k=5 or k=10

LOOCV

1. Randomly divide the train sample into two halves:
   1. *Estimation Sample: n-1*
   2. *Validation Sample: 1*
2. Fit the model on estimation sample
3. Test model on validation sample
4. Repeat Steps 1-3, n times

$$CV_{(n)} = \frac{1}{n}\sum_{i=1}^{n} MSE_i .$$

5. MSE is average of n models.



Evaluation of LOOCV

- Less bias because almost the entire sample is used for training the model
- Generally less variance than Validation because of n repeats.
- LOOCV is computationally intensive
  - *mostly because algorithms use the same approach as k-fold cross validation instead of using a computational shortcut. See Elements of Statistical Learning, Chapter 7 for details*
- LOOCV sometimes useful, but typically doesn’t shake up the data enough. The estimates from each fold are highly correlated and hence their average can have high variance.
- A better choice is k = 5 or 10.

See

AdvancedTrees_Wages.html and
AdvancedRegressionTrees_Credit.html

for a demonstration in R

ENSEMBLE MODELS

Ensemble Models

- Ensemble models take on a wisdom of the crowds approach by combining predictions from a number of models.
- Two heads are better than one, Many heads are even better.
- Two categories of tree-based ensemble methods
  - *Bagging and Forests: Grow multiple trees by using bootstrapped samples and average predictions.*
  - *Boosting: Grow a series of trees sequentially, each tree using information from the previously grown trees*

Trees

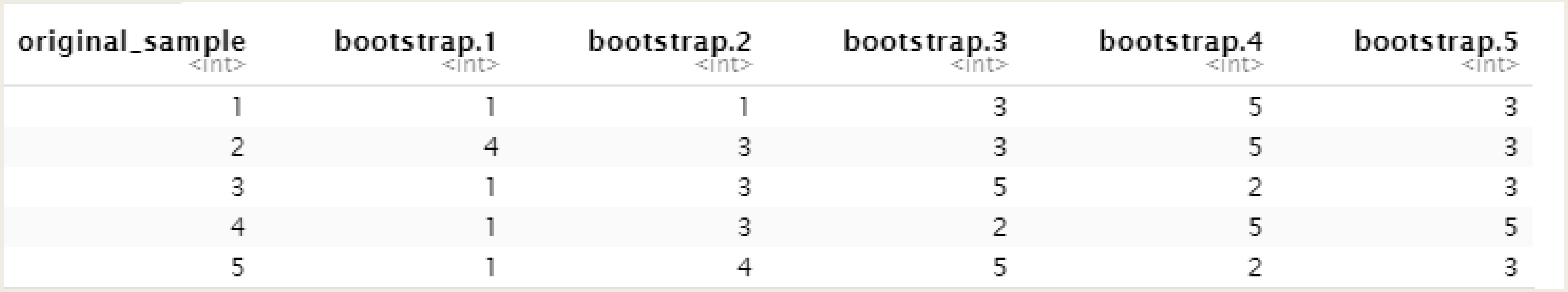
- Trees are easy to interpret but…
  - *they generally underperform regression based methods*
  - *And they tend to overfit the data*
- Three ensemble models to improve trees are
  - *Bagging*
  - *Random Forests*
  - *Boosting*

BAG

Bootstrap AGgregation

Bootstrapping Process

- Draw samples from original data with replacement
- Repeat multiple times

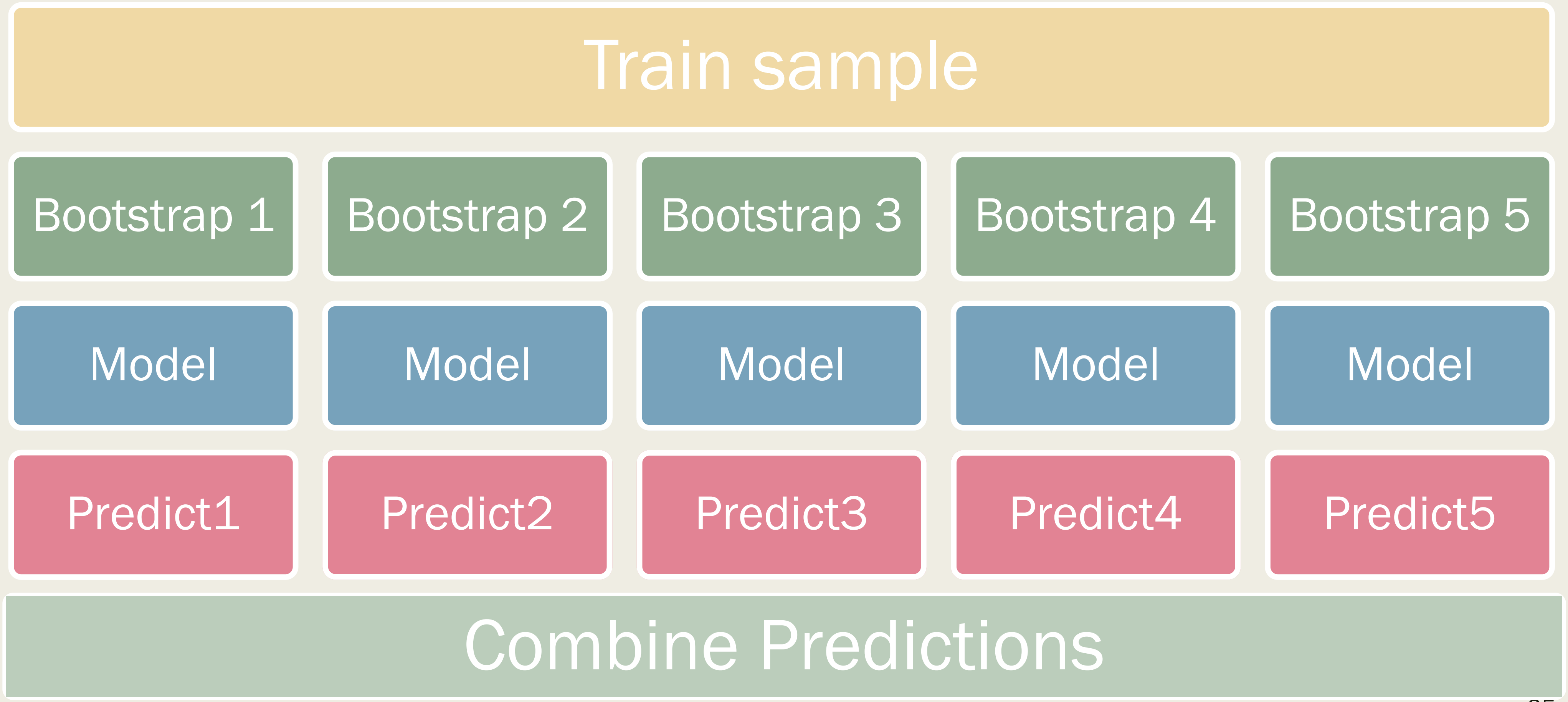


original_sample <int>	bootstrap.1 <int>	bootstrap.2 <int>	bootstrap.3 <int>	bootstrap.4 <int>	bootstrap.5 <int>
1	1	1	3	5	3
2	4	3	3	5	3
3	1	3	5	2	3
4	1	3	2	5	5
5	1	4	5	2	3

Bagging

- Bagging involves three steps
  - *Generating a large number of bootstrapped samples from train sample*
  - *Train the model on each sample*
  - *Combining models by averaging (metric outcome) or majority vote (non-metric outcome)*

Bagging Process



Bagging

- Trees constructed in bagging are not pruned, so they tend to be very large trees. But, that is okay, because we are going to average a large number of trees.
- Averaging predictions reduces variance while leaving bias unchanged

Popular R Packages

- library(ipred)
- library(randomForest) (after a small tweak)

See

AdvancedTrees_Wages.html and
AdvancedRegressionTrees_Credit.html

for a demonstration in R

RANDOM FORESTS

Random Forests

- Random forests provide an improvement over bagged trees by way of a small tweak that de-correlates the trees. This reduces the variance when we average the trees.
- As in bagging, we build a number of decision trees on bootstrapped training samples.
- But when building these decision trees, each time a split in a tree is considered, a random selection of m predictors is chosen as split candidates from the full set of p predictors. The split is allowed to use only one of those m predictors.
- A fresh selection of m predictors is taken at each split, and typically we choose $m \approx \sqrt{p}$ — that is, the number of predictors considered at each split is approximately equal to the square root of the total number of predictors
- Notice that random forest with $m = p$ is same as bagging!!

Why consider a random sample of predictors?

- Suppose that we have a very strong predictor in the data set along with a number of other moderately strong predictor, then in the collection of bagged trees, most or all of them will use the very strong predictor for the first split!
- All bagged trees will look similar. Hence all the predictions from the bagged trees will be highly correlated
- Averaging many highly correlated quantities does not lead to a large variance reduction, and thus random forests “de-correlates” the bagged trees leading to more reduction in variance

Popular R Packages

- library(randomForest)
- library(ranger)
- library(Rborist)

See

AdvancedTrees_Wages.html and
AdvancedRegressionTrees_Credit.html

for a demonstration in R

BOOSTING

Boosting

- Like bagging, boosting is a general approach that can be applied to many statistical learning methods for regression or classification.
- Recall that bagging involves creating multiple copies of the original training data set using the bootstrap, fitting a separate decision tree to each copy, and then combining all of the trees in order to create a single predictive model.
- Notably, each tree is built on a bootstrap data set, independent of the other trees.
- Boosting works in a similar way, except that the trees are grown sequentially: each tree is grown using information from previously grown trees.

Approach

- Unlike fitting a single large decision tree to the data, which amounts to fitting the data hard and potentially overfitting, the boosting approach instead learns slowly.
- Given the current model, we fit a decision tree to the residuals from the model. We then add this new decision tree into the fitted function in order to update the residuals.
- Each of these trees can be rather small, with just a few terminal nodes, determined by the parameter d in the algorithm.
- By fitting small trees to the residuals, we slowly improve ˆf in areas where it does not perform well. The shrinkage parameter λ slows the process down even further, allowing more and different shaped trees to attack the residuals.

Tuning Parameters

- Choice of hyperparameters is critical in establishing a boosting model. Here are some of the important ones in the Gradient Boosting Machine.

1. n.trees: The number of trees B. Unlike bagging and random forests, boosting can overfit if B is too large, although this overfitting tends to occur slowly if at all. We use cross-validation to select B.
2. shrinkage: The shrinkage parameter λ, a small positive number. This controls the rate at which boosting learns. Typical values are 0.01 or 0.001, and the right choice can depend on the problem. Very small λ can require using a very large value of B in order to achieve good performance.
3. interaction.depth: The number of splits d in each tree, which controls the complexity of the boosted ensemble. Often d = 1 works well, in which case each tree is a stump, consisting of a single split and resulting in an additive model. More generally d is the interaction depth, and controls the interaction order of the boosted model, since d splits can involve at most d variables.

Boosting

- Tend to perform better than state-of-the-art prediction models
- Some boosting algorithms (e.g., gbm and lightgbm) can optimize a user-defined cost function not just minimize a standard loss function

- Prone to overfitting. Tuning is extremely important
- Can be extremely slow
- Sensitive to extreme values

Popular R Packages

- library(gbm)
- library(xgboost)
- library(lightgbm)

See

AdvancedTrees_Wages.html and AdvancedRegressionTrees_Credit.html

for a demonstration in R

In closing

- Predictive performance of trees can be greatly improved by
  - *tuning hyperparameters, and*
  - *modeling techniques that aggregate many trees*

Summary

In this module, we

- Evaluated Trees
- Discussed Model Hyperparameters
- Examined Cross-Validation
- Used Bagging, Random Forest and Boosting models