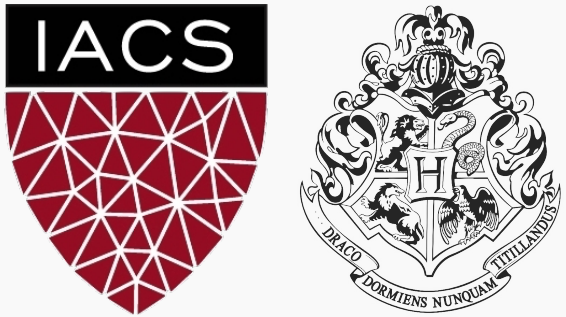
Lecture #15: Regression Trees & Random Forests

CS 109A, STAT 121A, AC 209A: Data Science

Pavlos Protopapas Kevin Rader



Lecture Outline

Review

Decision Trees for Regression

Bagging

Random Forests

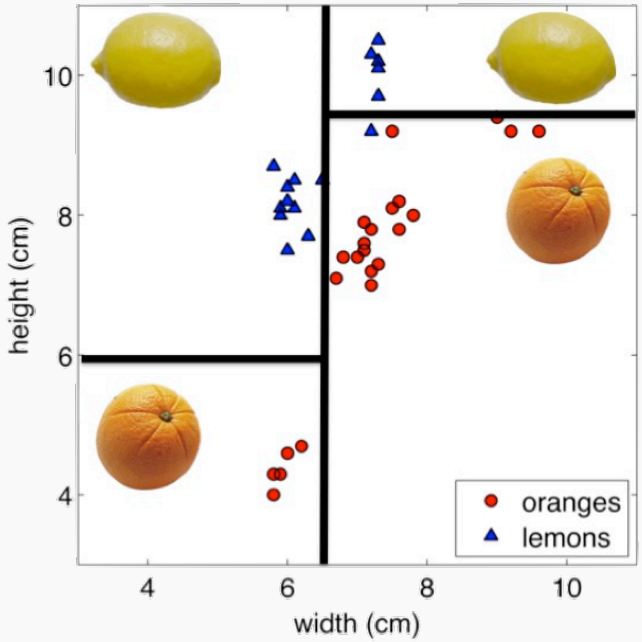
Review

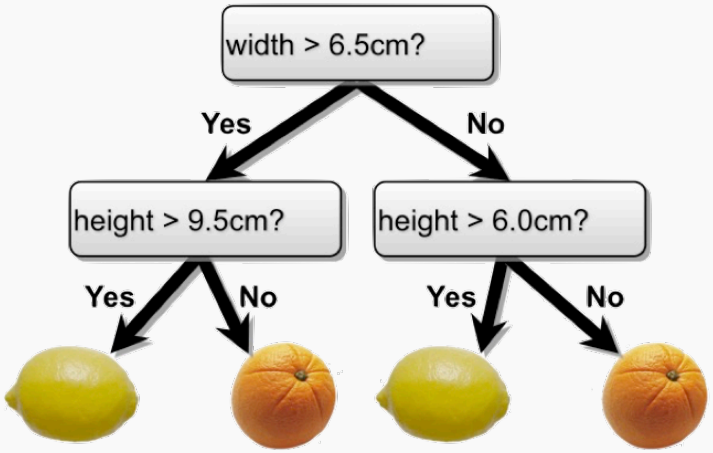
Decision Trees

A ***decision tree model*** is an interpretable model in which the final output is based on a series of comparisons of the values of predictors against threshold values.

Graphically, decision trees can be represented by a flow chart.

Geometrically, the model partitions the feature space wherein each region is assigned a response variable value based on the training points contained in the region.





Learning Algorithm

To learn a decision tree model, we take a greedy approach:

1. Start with an empty decision tree (undivided feature space)
2. Choose the ‘optimal’ predictor on which to split and choose the ‘optimal’ threshold value for splitting by applying a ***splitting criterion***
3. Recurse on on each new node until ***stopping condition*** is met

For classification, we label each region in the model with the label of the class to which the plurality of the points within the region belong.

Decision Trees for Regression

Adaptations for Regression

With just two modifications, we can use a decision tree model for regression:

- The three splitting criteria we've examined each promoted splits that were pure - new regions increasingly specialized in a single class.

  For classification, purity of the regions is a good indicator the performance of the model.

  For regression, we want to select a splitting criterion that promotes splits that improves the predictive accuracy of the model as measured by, say, the MSE.

- For regression with output in $\mathbb{R}$, we want to label each region in the model with a real number - typically the average of the output values of the training points contained in the region.

Learning Regression Trees

The learning algorithms for decision trees in regression tasks is:

1. Start with an empty decision tree (undivided feature space)
2. Choose a predictor j on which to split and choose a threshold value t_j for splitting such that the weighted average MSE of the new regions as smallest possible:

$$\text{argmin}_{j,t_j} \frac{N_1}{N}\text{MSE}(R_1) + \frac{N_2}{N}\text{MSE}(R_2)$$

   or equivalently,

$$\text{argmin}_{j,t_j} \frac{N_1}{N}\text{Var}[y|x \in R_1] + \frac{N_2}{N}\text{Var}[y|x \in R_2]$$

   where N_i is the number of training points in R_i and N is the number of points in R.

3. Recurse on on each new node until ***stopping condition*** is met

Stopping Conditions

Most of the stopping conditions, like maximum depth or minimum number of points in region, we saw last time can still be applied.

In the place of purity gain, we can instead compute accuracy gain for splitting a region R

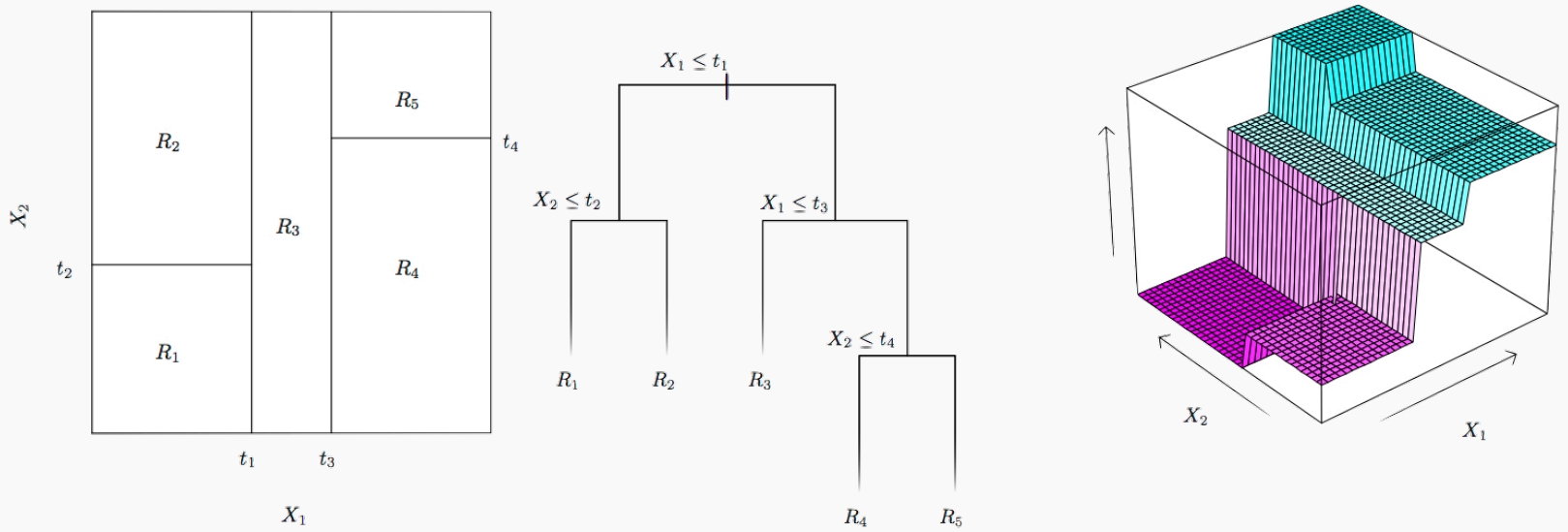
$$\text{Gain}(R) = \Delta(R) = MSE(R) - \frac{N_1}{N} MSE(R_1) - \frac{N_2}{N} MSE(R_2)$$

and stop the tree when the gain is less than some pre-defined threshold.

Expressiveness of Decision Trees

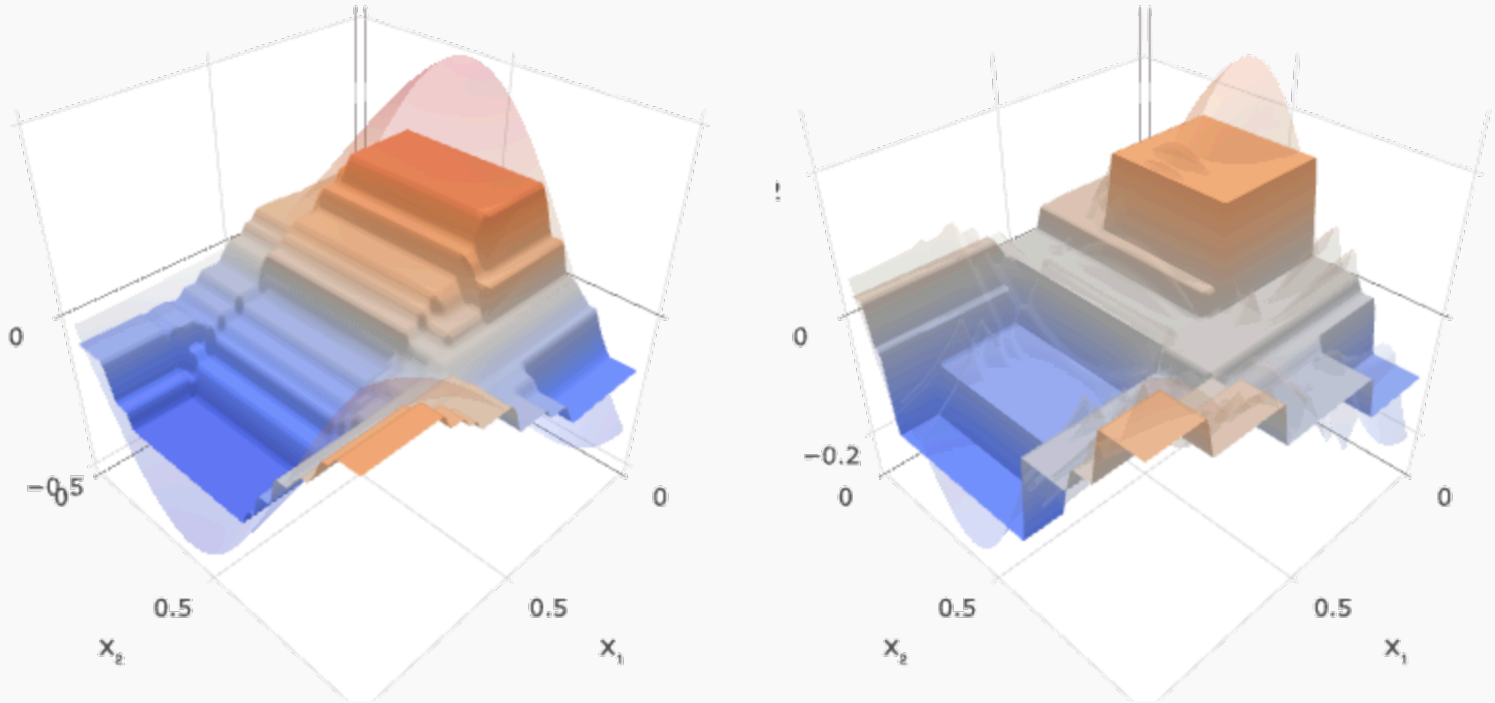
We've seen that classification trees approximate boundaries in the feature space that separate classes.

Regression trees, on the other hand, define ***simple functions*** or step functions, functions that are defined on partitions of the feature space and are constant over each part.



Expressiveness of Decision Trees

For a fine enough partition of the feature space, these functions can approximate complex non-linear functions.



Bagging

Limitations of Decision Tree Models

Decision trees models are highly interpretable and fast to train, using our greedy learning algorithm.

However, in order to capture a complex decision boundary (or to approximate a complex function), we need to use a large tree (since each time we can only make axis aligned splits).

We've seen that large trees have high variance and are prone to overfitting.

For these reasons, in practice, decision tree models often underperforms when compared with other classification or regression methods.

Bagging

One way to adjust for the high variance of the output of an experiment is to perform the experiment multiple times and then average the results.

The same idea can be applied to high variance models:

1. **(Bootstrap)** we generate multiple samples of training data, via bootstrapping. We train a full decision tree on each sample of data.

2. **(Aggregate)** for a given input, we output the averaged outputs of all the models for that input.

   For classification, we return the class that is outputted by the plurality of the models.

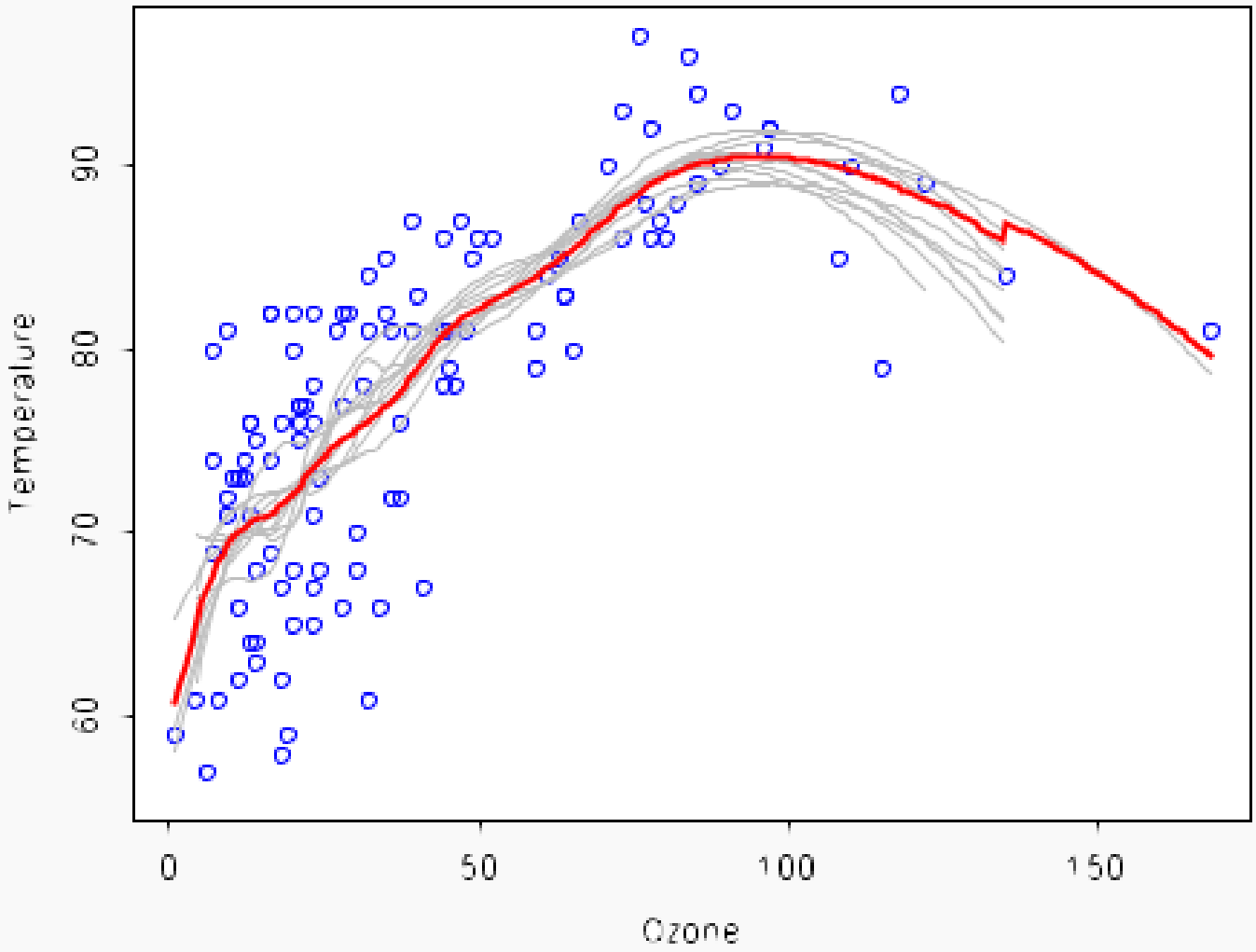
This method is called ***Bagging*** (Breiman, 1996), short for, of course, Bootstrap Aggregating.

Bagging

Note that bagging enjoys the benefits of

1. High expressiveness - by using full trees each model is able to approximate complex functions and decision boundaries.
2. Low variance - averaging the prediction of all the models reduces the variance in the final prediction, assuming that we choose a sufficiently large number of trees.

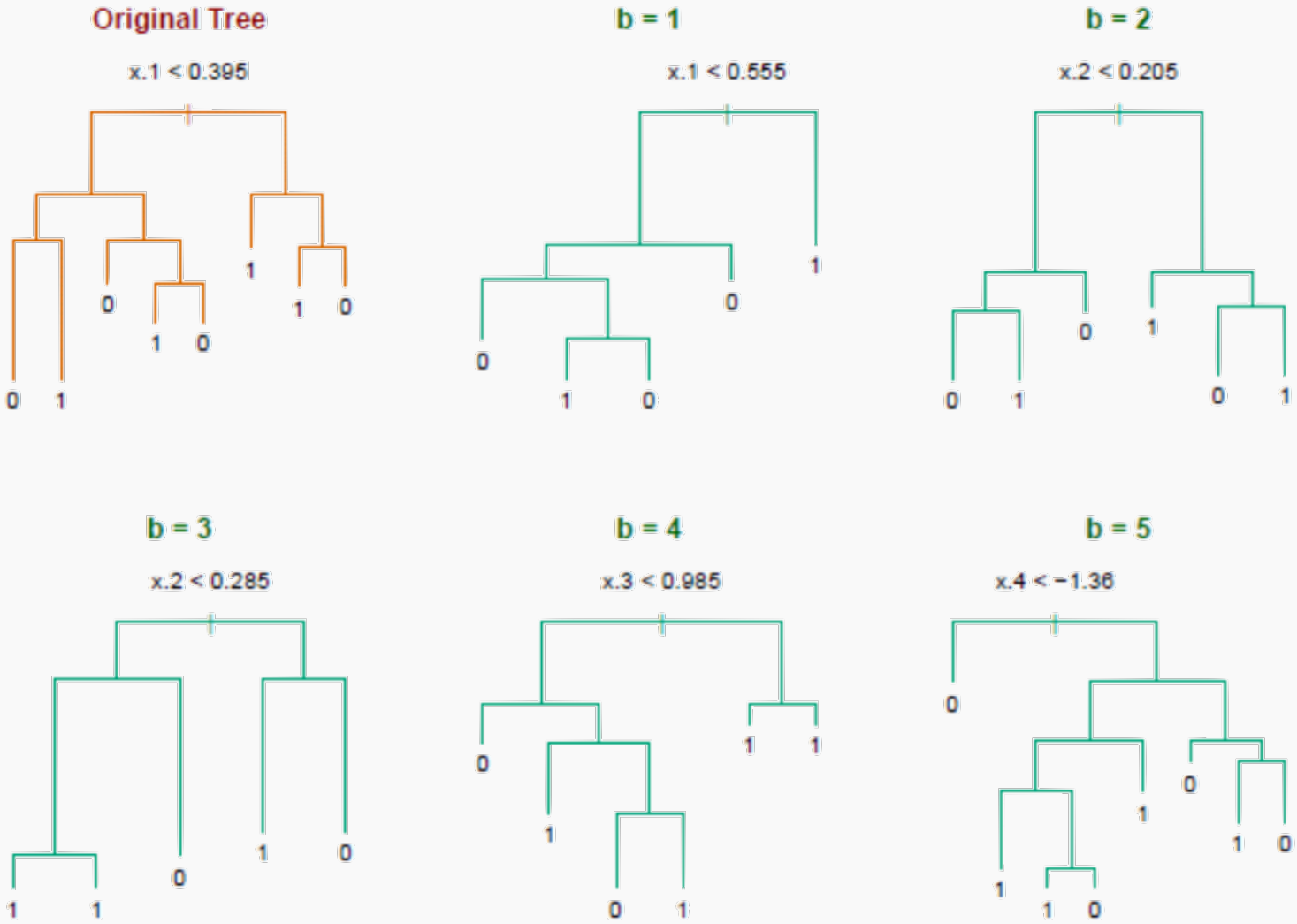
Bagging



Bagging

However, the major drawback of bagging (and other ***ensemble methods*** that we will study) is that the averaged model is no longer easily interpretable - i.e. one can no longer trace the 'logic' of an output through a series of decisions based on predictor values!

Bagging


Original Tree
x.1 < 0.395
0 1 0 1 1 0 1 0
b = 1
x.1 < 0.555
0 1 0 0 1
b = 2
x.2 < 0.205
0 1 0 1 0 1
b = 3
x.2 < 0.285
1 1 0 1 0
b = 4
x.3 < 0.985
0 1 0 1 1
b = 5
x.4 < −1.36
1 1 0 1 0 1 0

Out-of-Bag Error

Bagging is an example of an ***ensemble method***, a method of building a single model by training and aggregating multiple models.

With ensemble methods, we get a new metric for assessing the predictive performance of the model, the ***out-of-bag error***.

Given a training set and an ensemble of modeled each trained on a bootstrap sample, we compute the ***out-of-bag error*** of the averaged model by

1. for each point in the training set, we average the predicted output for this point over the models whose bootstrap training set excludes this point.

   We compute the the error or squared error of this averaged prediction. Call this the point-wise out-of-bag error.

2. we average the point-wise out-of-bag error over the full training set.

Random Forests

Improving on Bagging

In practice, the ensembles of trees in Bagging tend to be highly correlated.

Suppose we have an extremely strong predictor, x_j, in the training set amongst moderate predictors. Then the greedy learning algorithm ensures that most of the models in the ensemble will choose to split on x_j in early iterations.

That is, each tree in the ensemble is identically distributed, with the expected output of the averaged model the same as the expected output of any one of the trees.

Improving on Bagging

Recall, for B number of identically but not independently distributed variables with pairwise correlation ρ and variance σ^2, the variance of their mean is

$$\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2.$$

As we increase B, the second term vanishes but the first term remains.

Consequently, variance reduction in bagging is limited by the fact that we are averaging over highly correlated trees.

Random Forests

Random Forest is a modified form of bagging that creates ensembles of independent decision trees.

To de-correlate the trees, we:

1. train each tree on a separate bootstrap sample of the full training set (same as in bagging)
2. for each tree, at each split, we ***randomly*** select a set of J' predictors from the full set of predictors.

   From amongst the J' predictors, we select the optimal predictor and the optimal corresponding threshold for the split.

Tuning Random Forests

Random forest models have multiple hyper-parameters to tune:

1. the number of predictors to randomly select at each split
2. the total number of trees in the ensemble
3. the minimum leaf node size

In theory, each tree in the random forest is full, but in practice this can be computationally expensive (and added redundancies in the model), thus, imposing a minimum node size is not unusual.

Tuning Random Forests

There are standard (default) values for each of random forest hyper-parameters recommended by long time practitioners, but generally these parameters should be tuned through cross validation (making them data and problem dependent).

Using out-of-bag errors, training and cross validation can be done in a single sequence - we cease training once the out-of-bag error stabilizes

Example

[compare RF, Bagging and Tree]
[test for variable importance]

Final Thoughts on Random Forests

- When the number of predictors is large, but the number of relevant predictors is small, random forests can perform poorly.

  In each split, the chances of selected a relevant predictor will be low and hence most trees in the ensemble will be weak models.

Final Thoughts on Random Forests

- Increasing the number of trees in the ensemble generally does not increase the risk of overfitting.

  Again, by decomposing the generalization error in terms of bias and variance, we see that increasing the number of trees produces a model that is at least as robust as a single tree.

  However, if the number of trees is too large, then the trees in the ensemble may become more correlated, increase the variance.