

Tree-based Methods

- Here we describe *tree-based* methods for regression and classification.
- These involve *stratifying* or *segmenting* the predictor space into a number of simple regions.
- Since the set of splitting rules used to segment the predictor space can be summarized in a tree, these types of approaches are known as *decision-tree* methods.

Pros and Cons

ensemble learning models:

- combine several learning models (e.g., trees) into one predictive model (e.g., random forests)
- can often achieve improved predictive performance at the cost of model interpretability

- Tree-based methods are simple interpretation.
- However they typically are not competitive with the best supervised learning approaches in terms of prediction accuracy.
- Hence we also discuss *bagging*, *random forests*, and *boosting*. These methods grow multiple trees which are then combined to yield a single consensus prediction.
- Combining a large number of trees can often result in dramatic improvements in prediction accuracy, at the expense of some loss interpretation.

The Basics of Decision Trees



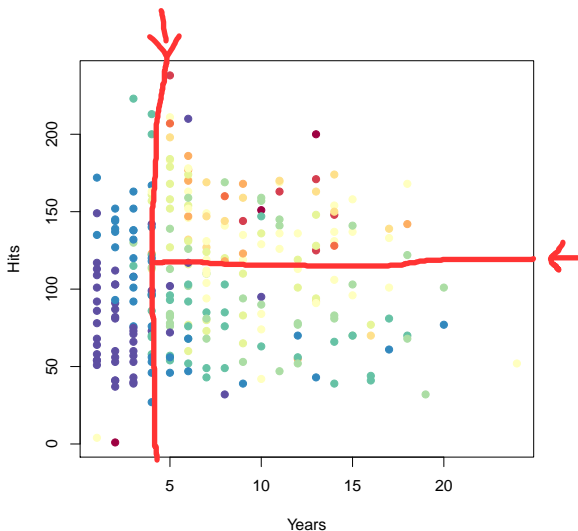
regression trees
(regression problems)

classification trees
(classification problems)

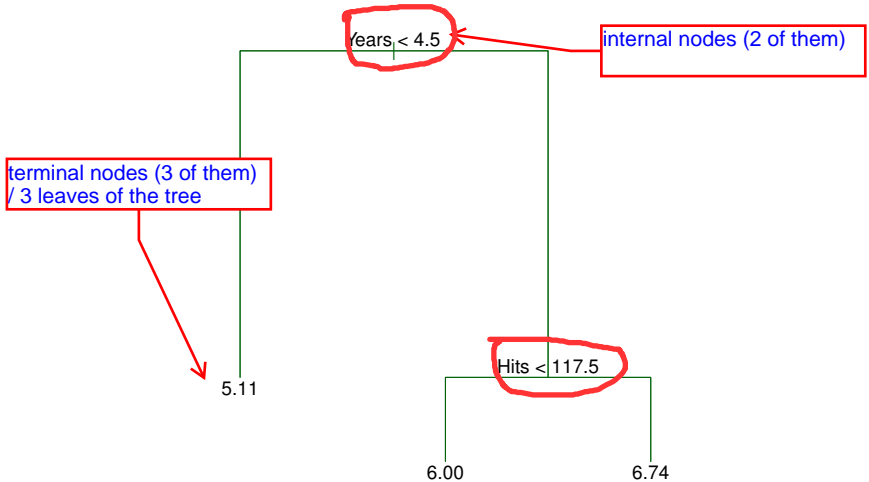
- Decision trees can be applied to both regression and classification problems.
- We first consider regression problems, and then move on to classification.

Baseball salary data: how would you stratify it?

Salary is color-coded from low (blue, green) to high (yellow, red)



Decision tree for these data

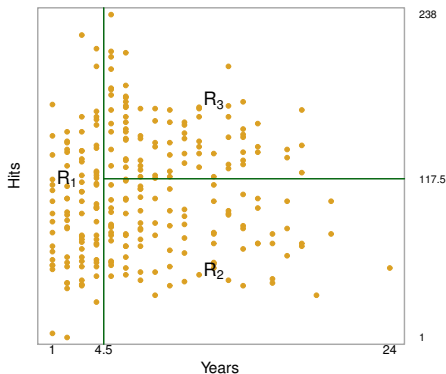


Details of previous figure

- For the Hitters data, a regression tree for predicting the log salary of a baseball player, based on the number of years that he has played in the major leagues and the number of hits that he made in the previous year.
- At a given internal node, the label (of the form $X_j < t_k$) indicates the left-hand branch emanating from that split, and the right-hand branch corresponds to $X_j \geq t_k$. For instance, the split at the top of the tree results in two large branches. The left-hand branch corresponds to **Years**<4.5, and the right-hand branch corresponds to **Years**>=4.5.
- The tree has two internal nodes and three terminal nodes, or leaves. The number in each leaf is the mean of the response for the observations that fall there.

Results

- Overall, the tree stratifies or segments the players into three regions of predictor space: $R_1 = \{X \mid \text{Years} < 4.5\}$, $R_2 = \{X \mid \text{Years} \geq 4.5, \text{Hits} < 117.5\}$, and $R_3 = \{X \mid \text{Years} \geq 4.5, \text{Hits} \geq 117.5\}$.



Terminology for Trees

- In keeping with the *tree* analogy, the regions R_1 , R_2 , and R_3 are known as *terminal nodes*
- Decision trees are typically drawn *upside down*, in the sense that the leaves are at the bottom of the tree.
- The points along the tree where the predictor space is split are referred to as *internal nodes*
- In the hitters tree, the two internal nodes are indicated by the text **Years**<4.5 and **Hits**<117.5.

Interpretation of Results

- **Years** is the most important variable, and players with less experience are paid less than more experienced players.
- Given that a player is less experienced, the **Salary** that he made in the previous year seems to play little role in his **Salary**.
- But among players who have been in the major leagues for five or more years, the number of **Hits** made in the previous year does affect **Salary**, and players who made more **Hits** last year tend to have higher salaries.
- Surely an over-simplification, but compared to a regression model, it is easy to display, interpret and explain

Advantages of trees:

- very interpretable: allows for easy inference & explanation
- easy to visualize

Disadvantage:

- over-simplified, may not give good predictions

Details of the tree-building process

1. We divide the predictor space — that is, the set of possible values for X_1, X_2, \dots, X_p — into J distinct and non-overlapping regions, R_1, R_2, \dots, R_J .
2. For every observation that falls into the region R_j , we make the same prediction, which is simply the mean of the response values for the training observations in R_j .

More details of the tree-building process

- In theory, the regions could have any shape. However, we choose to divide the predictor space into high-dimensional rectangles, or *boxes*, for simplicity and for ease of interpretation of the resulting predictive model.
- The goal is to find boxes R_1, \dots, R_J that minimize the RSS, given by

$$\sum_{j=1}^J \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2,$$

where \hat{y}_{R_j} is the mean response for the training observations within the j th box.

More details of the tree-building process

very computationally
expensive, similar to the
best subset selection

- Unfortunately, it is computationally infeasible to consider every possible partition of the feature space into J boxes.
- For this reason, we take a *top-down*, *greedy* approach that is known as recursive binary splitting.
- The approach is *top-down* because it begins at the top of the tree and then successively splits. Each split is indicated via two new nodes on the tree.
- It is *greedy* because at each step of the tree-building process, the *best* split is made at that particular step, rather than looking ahead and picking a split that will lead to a better tree in some future step.

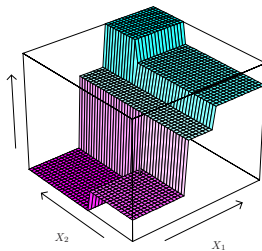
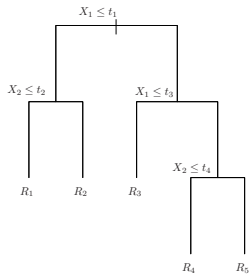
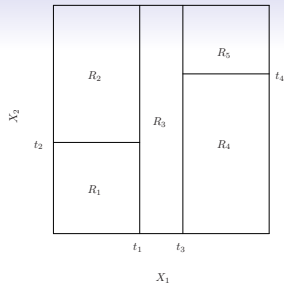
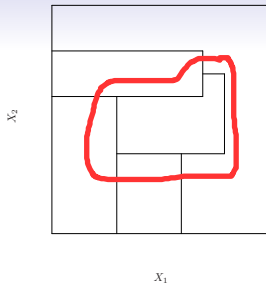
similar to forward selection: we may not get the best split overall, but computationally much faster (do not look forward)

Details— Continued

- We first select the predictor X_j and the cutpoint s such that splitting the predictor space into the regions $\{X|X_j < s\}$ and $\{X|X_j \geq s\}$ leads to the greatest possible reduction in RSS.
- Next, we repeat the process, looking for the best predictor and best cutpoint in order to split the data further so as to minimize the RSS within each of the resulting regions.
- However, this time, instead of splitting the entire predictor space, we split one of the two previously identified regions. We now have three regions.
- Again, we look to split one of these three regions further, so as to minimize the RSS. The process continues until a stopping criterion is reached; for instance, we may continue until no region contains more than five observations.

Predictions

- We predict the response for a given test observation using the mean of the training observations in the region to which that test observation belongs.
- A five-region example of this approach is shown in the next slide.



Details of previous figure

Top Left: A partition of two-dimensional feature space that could not result from recursive binary splitting.

Top Right: The output of recursive binary splitting on a two-dimensional example.

Bottom Left: A tree corresponding to the partition in the top right panel.

Bottom Right: A perspective plot of the prediction surface corresponding to that tree.

Pruning a tree

- The process described above may produce good predictions on the training set, but is likely to *overfit* the data, leading to poor test set performance. *Why?*

Pruning a tree

a tree with large J may yield too complex of a predictor
- this results in high variance of the fitted model (but low bias) - we are making a lot of decisions from the training data

- The process described above may work on the training set, but is likely to lead to poor test set performance. *Why?*
- A smaller tree with fewer splits (that is, fewer regions R_1, \dots, R_J) might lead to lower variance and better interpretation at the cost of a little bias.

recursive binary splitting is too greedy: stopping in the middle of the procedure may not give us a good fitted tree (too shortsighted)

- alternative to the process described above is to stop only so long as the decrease in the RSS exceeds some (high) threshold.
- This strategy will result in smaller trees, but is too *short-sighted*: a seemingly worthless split early on in the tree might be followed by a very good split — that is, a split that leads to a large reduction in RSS later on.

one way: consider all possible subtrees of the fully grown tree T_0 (from recursive binary splitting), then perform cross validation:

- problem: very computationally expensive to find all subtrees & perform CV (especially when T_0 is large)
- similar to the computational problem for best subset selection

— cont
very la
n a *sub*
so known as *weakest link*

solution: consider only a small set of "good" subtrees

- adopt a similar formulation as LASSO for variable selection

- we consider a sequence of trees indexed by a nonnegative tuning parameter α . For each value of α there corresponds a subtree $T \subset T_0$ such that

$|T|$

$|T|$: measure of complexity of fitted model (cardinality)

tuning parameter α plays the same role as λ in penalized regression

- penalizing $|T|$: the number of terminal nodes (or leaves) in the subtree, the number of partitions
- α controls trade-off between tree complexity and its fit to training data:
 - $\alpha=0$: the optimal tree is the fully grown tree T_0 (high variance, low bias)
 - α large: the optimal tree becomes the null tree with only one node, we are predicting the mean

$$R_m)^2 + \alpha|T|$$

indicates the number of
is the rectangle (i.e. the
ponding to the m th
ean of the training

Choosing the best subtree

As we increase λ from 0, the tree gets pruned in a "nested" fashion

- obtaining the full subtree solution path (sequence of solutions as λ increases) is computationally easy

- allow for efficient tuning of λ via cross-validation

- - similar to the efficient computation of the solution path for Lasso/ridge regression.

trade-off between the
training data.

- We select an optimal value $\hat{\lambda}$ using cross-validation.
- We then return to the full data set and obtain the subtree corresponding to $\hat{\lambda}$.

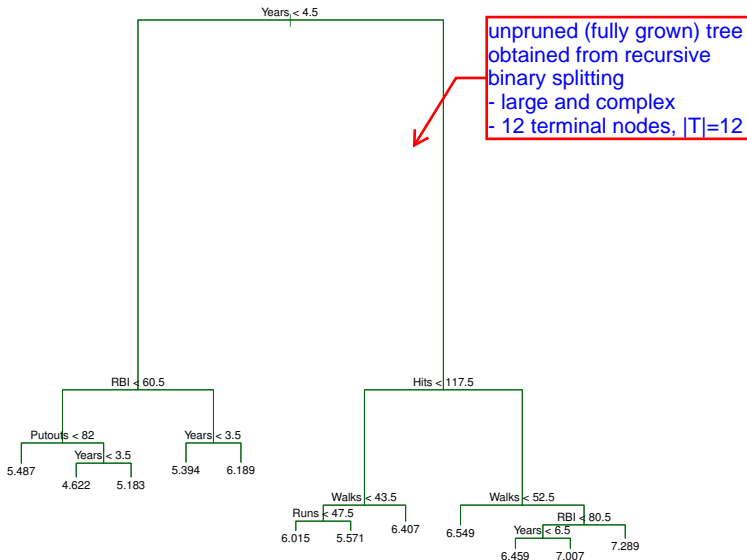
Summary: tree algorithm

1. Use recursive binary splitting to grow a large tree on the training data, stopping only when each terminal node has fewer than some minimum number of observations.
2. Apply cost complexity pruning to the large tree in order to obtain a sequence of best subtrees, as a function of α .
3. Use K-fold cross-validation to choose α . For each $k = 1, \dots, K$:
 - 3.1 Repeat Steps 1 and 2 on the $\frac{K-1}{K}$ th fraction of the training data, excluding the k th fold.
 - 3.2 Evaluate the mean squared prediction error on the data in the left-out k th fold, as a function of α .Average the results, and pick α to minimize the average error.
4. Return the subtree from Step 2 that corresponds to the chosen value of α .

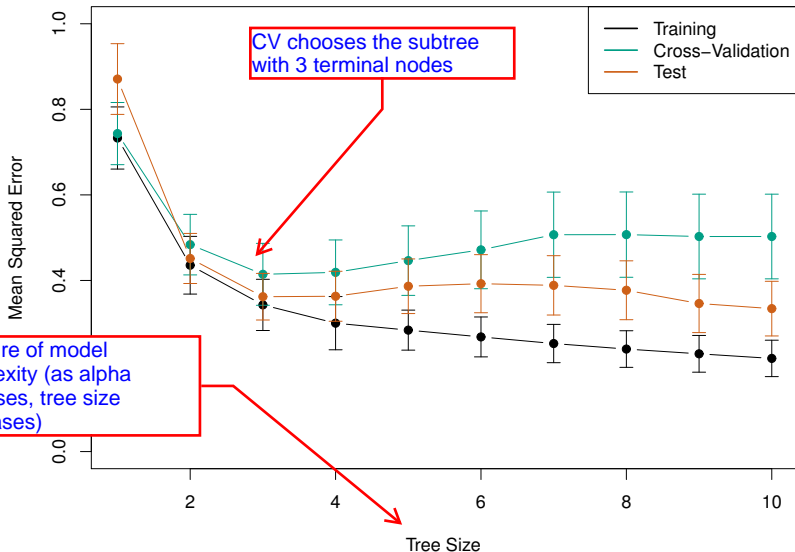
Baseball example continued

- First, we randomly divided the data set in half, yielding 132 observations in the training set and 131 observations in the test set.
- We then built a large regression tree on the training data and varied α in in order to create subtrees with different numbers of terminal nodes.
- Finally, we performed six-fold cross-validation in order to estimate the cross-validated MSE of the trees as a function of α .

Baseball example continued



Baseball example continued



Classification Trees

this is essentially the Bayes-optimal classifier (majority votes)

- Very similar to a regression tree, except that it is used to predict a qualitative response rather than a quantitative one.
- For a classification tree, we predict that each observation belongs to the *most commonly occurring class* of training observations in the region to which it belongs.

Details of classification trees

- Just as in the regression setting, we use recursive binary splitting to grow a classification tree.
- In the classification setting, RSS cannot be used as a criterion for making the binary splits
- A natural alternative to RSS is the *classification error rate*. this is simply the fraction of the training observations in that region that do not belong to the most common class:

$$E = 1 - \max_k(\hat{p}_{mk}).$$

Here \hat{p}_{mk} represents the proportion of training observations in the m th region that are from the k th class.

- However classification error is not sufficiently sensitive for tree-growing, and in practice two other measures are preferable.

Gini index and Deviance

- The *Gini index* is defined by

a measure of "total variance" across the K classes within a partition:

- recall the variance of Bernoulli (p) = $p(1-p)$
- provides a measure of node purity: the smaller G is, the more pure the node

$$G = \sum_{k=1}^K \hat{p}_{mk}(1 - \hat{p}_{mk}),$$

total variance across the K classes. The Gini index is small if all of the \hat{p}_{mk} 's are close to zero or one.

- For this reason the Gini index is referred to as a measure of node *purity* — a small value indicates that a node contains predominantly observations from a single class.

a node is more "pure" if it consists of mostly observations from one class

Gini index and Deviance

- The *Gini index* is defined by

$$G = \sum_{k=1}^K \hat{p}_{mk}(1 - \hat{p}_{mk}),$$

measure of "randomness"
for a random variable
- smaller D: less "random",
all probabilities are
concentrated on 0 or 1
- larger D: more "random",
many observations are
from many classes
- goal is to minimize to
achieve greater node purity

total variance act
in a small value if

Two steps for fitting
classification trees:
- Tree growing (to fully
grown tree T0) using Gini
Index / Deviance
- Tree pruning using
misclassification error

on the Gini index

— a small value indicates that a node contains
predominantly observations from a single class.

- An alternative to the Gini index is *cross-entropy*, given by

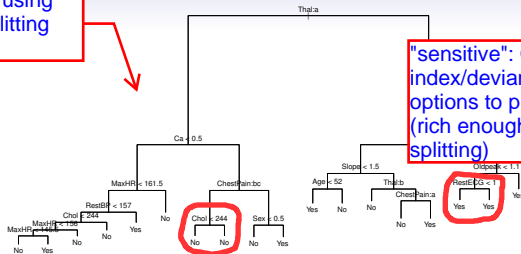
$$D = - \sum_{k=1}^K \hat{p}_{mk} \log \hat{p}_{mk}.$$

- It turns out that the Gini index and the cross-entropy are
very similar numerically.

Example: heart data

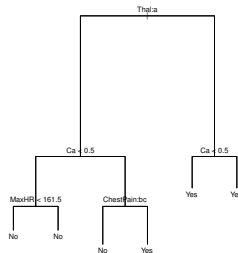
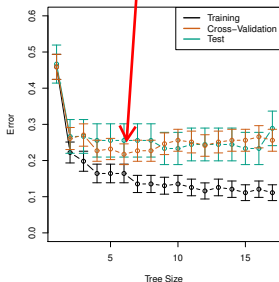
- These data contain a binary outcome **HD** for 303 patients who presented with chest pain.
- An outcome value of **Yes** indicates the presence of heart disease based on an angiographic test, while **No** means no heart disease.
- There are 13 predictors including **Age**, **Sex**, **Chol** (a cholesterol measurement), and other heart and lung function measurements.
- Cross-validation yields a tree with six terminal nodes. See next figure.

fully grown tree T0 using recursive binary splitting with Gini index



"sensitive": Gini index/deviance gives more options to prune the tree (rich enough ways for splitting)

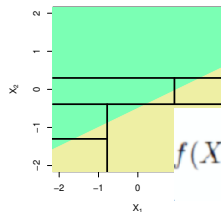
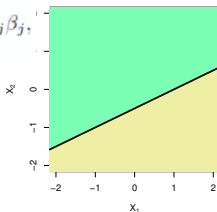
CV on α : optimal tree size = 6 (terminal nodes)



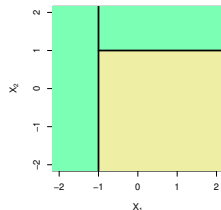
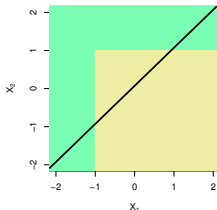
Trees Versus Line

when should we use regression
(classification) trees over linear
(logistic) regression?

$$f(X) = \beta_0 + \sum_{j=1}^p X_j \beta_j,$$



$$f(X) = \sum_{m=1}^M c_m \cdot 1_{(X \in R_m)}$$



Top Row: True linear boundary; Bottom row: true non-linear boundary.

Left column: linear model; Right column: tree-based model

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 the regression and classification approaches seen in previous chapters.
- ▲ 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.
- ▼ Unfortunately, trees generally do not have the same level of predictive accuracy as some of the other regression and classification approaches seen in this book.

ensemble learning

However, by aggregating many decision trees, the predictive performance of trees can be substantially improved. We introduce these concepts next.

Bagging

Problem with fully grown decision trees:
- high variance of the fitted model
=> high test errors for prediction

- *Bootstrap aggregation*, or *bagging*, is a general-purpose procedure for reducing the variance of a statistical learning method; we introduce it here because it is particularly useful and frequently used in the context of decision trees.
- Recall that given a set of n independent observations Z_1, \dots, Z_n , each with variance σ^2 , the variance of the mean \bar{Z} of the observations is given by σ^2/n .
- In other words, *averaging a set of observations reduces variance*. Of course, this is not practical because we generally do not have access to multiple training sets.

Bagging—

Bootstrapping: construct a new training dataset by sampling the original training data with replacement

- Instead, we can bootstrap, by taking repeated samples from the (single) training data set.
- In this approach we generate B different bootstrapped training data sets. We then train our method on the b th bootstrapped training set in order to get $\hat{f}^{*b}(x)$, the prediction at a point x . We then average all the predictions to obtain

$$\hat{f}_{\text{bag}}(x) =$$

This is called *bagging*.

Bagging: ensemble learning method that improves stability & predictive performance, by reducing the variance of an individual learner

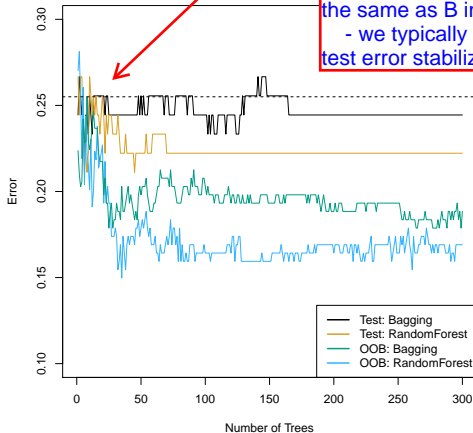
- each individual tree is fully grown & unpruned
- => high variance but low bias
- averaging these trees helps significantly reduce variance while keeping bias at a similar level
- => better predictive performance

Bagging classification trees

- The above prescription applied to regression trees
- For classification trees: for each test observation, we record the class predicted by each of the B trees, and take a *majority vote*: the overall prediction is the most commonly occurring class among the B predictions.

Bagging the

- dotted line: test error for a single unpruned decision tree
- bagging: solid black line
- observations:
 - bagging gives us slightly lower prediction errors than a fully grown decision tree
 - a large bootstrap samples B will not overfit the model for bagging
 - why? the fitted predictor averages over individual trees (variance decreases, bias stays the same as B increases)
 - we typically increase # of trees (B) until the test error stabilizes



Details of previous figure

Bagging and random forest results for the Heart data.

- The test error (black and orange) is shown as a function of B , the number of bootstrapped training sets used.
- Random forests were applied with $m = \sqrt{p}$.
- The dashed line indicates the test error resulting from a single classification tree.
- The green and blue traces show the OOB error, which in this case is considerably lower

Out-of-Bag Error Esti

OOB error estimation: a computationally efficient way of estimating test error on a bagged model (compared to CV)

- It turns out that there is a very straightforward way to estimate the test error of a bagged model.
- Recall that the key to bagging is that trees are repeatedly fit to bootstrapped subsets of the observations. HW4 Q3
show that on average, each bagged tree makes use of around two-thirds of the observations.
- The remaining one-third of the observations not used to fit a given bagged tree are referred to as the *out of bag* (OOB) observations.
- We can predict the response for the i th observation using each of the trees in which that observation was not used. OOB test error is much easier to compute, since the test error estimate comes as a by-product of the fitted model
If we fit B bagged trees, each tree will yield around $B/3$ predictions for the i th observation, which we average.
- This estimate is essentially the LOO cross-validation error for bagging, if B is large.

Recall earlier motivation for bagging:

- Z_1, \dots, Z_B each with variance σ^2

Random forests allow for only a random subset of predictors to be split, at each split of the bootstrapped trees

- why does this decorrelate the trees?

- suppose one predictor (x_1) is highly influential, and others less so

- then most of the bootstrapped trees in bagging will have x_1 as its top split \Rightarrow trees will be highly correlated

- with the split restriction for random forests, we force the bootstrapped trees to consider other splits besides $x_1 \Rightarrow$ decorrelates the trees & maximizes the benefit of variance reduction

- *Random forests*

by way of a small example, reduces the variance

- As in bagging, bootstrapped trees

- But when building a tree is considered

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 (4 out of the 13 for the Heart data).

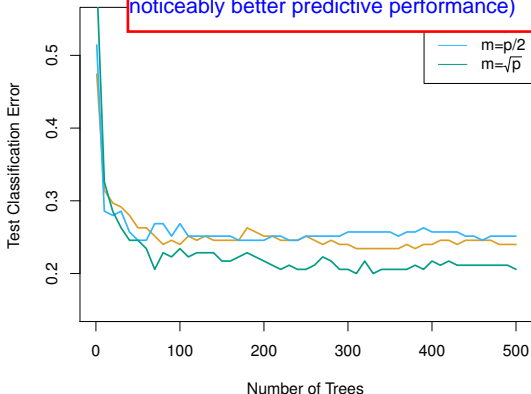
Example: gene expression data

- We applied random forests to a high-dimensional biological data set consisting of expression measurements of 4,718 genes measured on tissue samples from 349 patients.
- There are around 20,000 genes in humans, and individual genes have different levels of activity, or expression, in particular cells, tissues, and biological conditions.
- Each of the patient samples has a qualitative label with 15 different levels: either normal or one of 14 different types of cancer.
- We use random forests to predict cancer type based on the 500 genes that have the largest variance in the training set.
- We randomly divided the observations into a training and a test set, and applied random forests to the training set for three different values of the number of splitting variables m .

Results: gene expression data

Observations:

- $m=p$: bagging (orange)
- $m=p/2$: a restriction of 50% predictors for each split
- $m = \sqrt{p}$: restriction of 4.5% predictors (results in noticeably better predictive performance)



Details of previous figure

- Results from random forests for the fifteen-class gene expression data set with $p = 500$ predictors.
- The test error is displayed as a function of the number of trees. Each colored line corresponds to a different value of m , the number of predictors available for splitting at each interior tree node.
- Random forests ($m < p$) lead to a slight improvement over bagging ($m = p$). A single classification tree has an error rate of 45.7%.

Boosting

- Like bagging, boosting is a general approach that can be applied to many statistical learning methods for regression or classification. We only discuss boosting for decision trees.
- 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.

Boosting algo

1. Set $\hat{f}(x) = 0$ and r_i

2. For $b = 1, 2, \dots, B$,

2.1 Fit a tree \hat{f}^b with
training data (X, r_i)

2.2 Update \hat{f} by add

Intuition behind boosting:

- prevent overfitting of large decision trees by learning slowly & iteratively

what do we mean by "slowly"?

- fit small trees (i.e., with d splits, where d small) to slowly improve regions where the predictor is poor
- applying a small learning rate (λ) to prevent overfitting to a single tree

what do we mean by "iteratively"?

- we want to sequentially fit trees that adaptively reduces the residuals

$$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x).$$

2.3 Update the res

3. Output the boost

Algorithm parameters:

- B: # of decision trees (# of iterations)
 - tuned via cross-validation
- λ : learning rate (typically small, e.g., 0.01)
 - encourages slow & steady learning
 - smaller λ => larger B to get a good fit
- d: interaction depth (# of splits allowed in your tree)
 - $d = 1$: each tree is a stump consisting of one split
 - in this case, the fitted boosted predictor is an additive model (interpretable)
 - more generally, d controls the interaction order of the boosted model

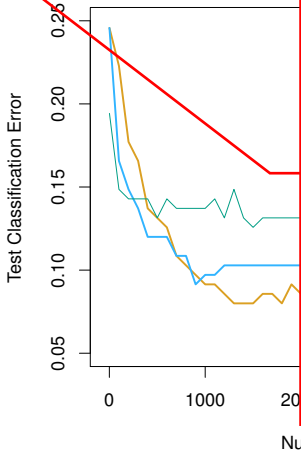
What is the idea behind this procedure?

- 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 \hat{f} 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.

Boosting for classification

- Boosting for classification is similar in spirit to boosting for regression, but is a bit more complex. We will not go into detail here, nor do we in the text book.
- Students can learn about the details in *Elements of Statistical Learning, chapter 10*.
- The R package `gbm` (gradient boosted models) handles a variety of regression and classification problems.

Gene expression data continued



Observations:

- Boosting with depth 1 or 2 performs better than RFs
- Boosting with depth 1 (which uses stump trees) perform better than depth 2
- Depth 1 boosting = more interpretable (you can gauge the effect of each gene separately)
- this also suggests that there may be little interactions between genes for this particular disease (gene-gene interaction = epistasis)

Differences between boosting and bagging / RFs:

- Bagging / RFs: individual trees that are fully grown and not pruned (low bias, high variance). we rely on ensemble averaging to reduce variance.
- Boosting: addresses overfitting by growing the predictor slowly (i.e., with small learning rate & small trees) and iteratively (by fitting trees sequentially on residuals).

Details of previous figure

- Results from performing boosting and random forests on the fifteen-class gene expression data set in order to predict *cancer* versus *normal*.
- The test error is displayed as a function of the number of trees. For the two boosted models, $\lambda = 0.01$. Depth-1 trees slightly outperform depth-2 trees, and both outperform the random forest, although the standard errors are around 0.02, making none of these differences significant.
- The test error rate for a single tree is 24%.

Tuning parameters for boosting

1. 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 .

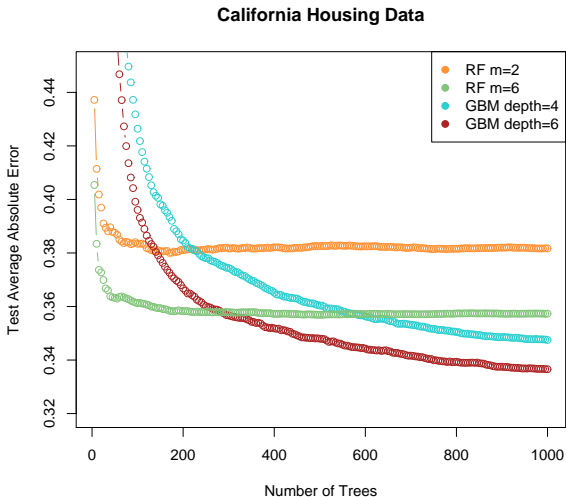
Tuning parameters for boosting

1. 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. 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.

Tuning parameters for boosting

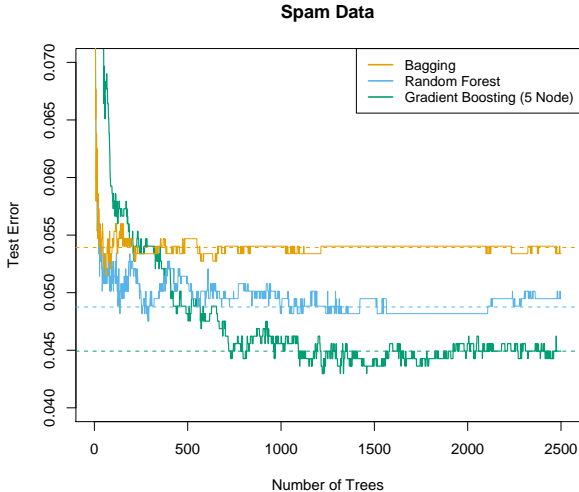
1. The *number of trees* B . Unlike bagging and random forests, boosting can overfit if B is too large, although this overfitting tends to occur slowly if at all. We use cross-validation to select B .
2. The *shrinkage parameter* λ , a small positive number. This controls the rate at which boosting learns. 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. 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.

Another regression example



from *Elements of Statistical Learning*, chapter 15.

Another classification example

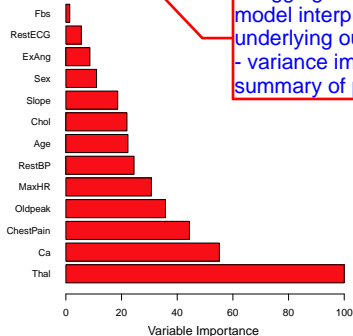


from *Elements of Statistical Learning, chapter 15.*

Variable importance measure

- For bagged/RF regression trees, we record the total amount that the RSS is decreased due to splits over a given predictor, averaged over all B trees. A large value indicates an important predictor.
- Similarly, for bagged/RF classification trees, we add up the total amount that the Gini index is decreased by splits over a given predictor, averaged over all B trees.

- bagging / RFs improve predictive power at the expensive of model interpretability (we can no longer see the "decision" underlying our predictor - "black-box")
- variance importance allow us to quantify an overall summary of predictor importance in bagging / RFs



Variable importance plot
for the **Heart** data

Summary

- Decision trees are simple and interpretable models for regression and classification
- However they are often not competitive with other methods in terms of prediction accuracy
- Bagging, random forests and boosting are good methods for improving the prediction accuracy of trees. They work by growing many trees on the training data and then combining the predictions of the resulting ensemble of trees.
- The latter two methods— random forests and boosting—are among the state-of-the-art methods for supervised learning. However their results can be difficult to interpret.