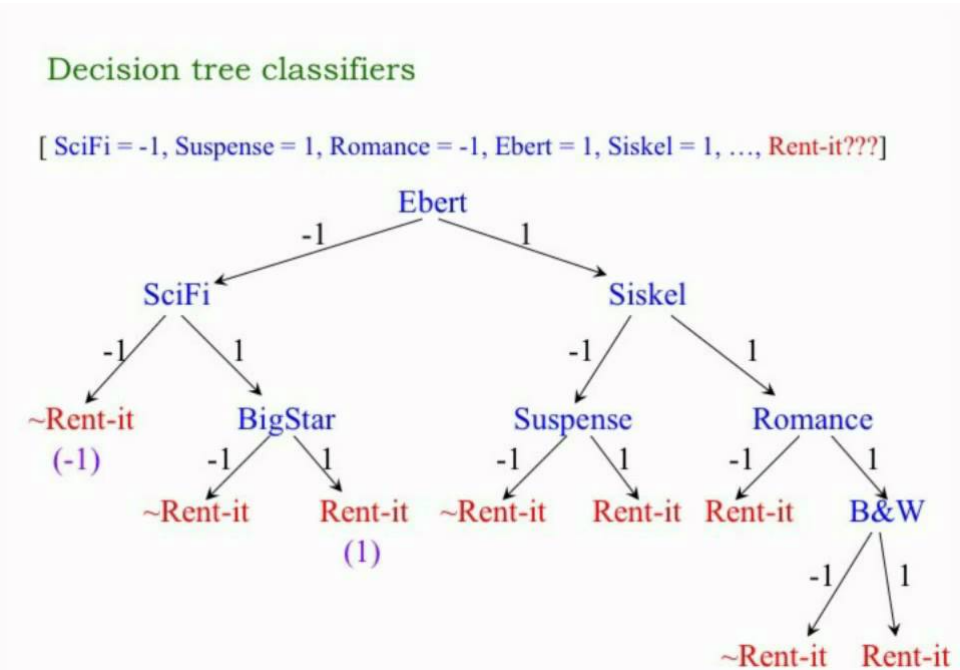
CS 467, Introduction to Machine Learning

University of Southern California

M. R. Rajati, PhD

Tree-Based Methods



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

- Tree-based methods are simple and useful for interpretation.
- However they typically are not competitive with the best supervised learning approaches in terms of prediction accuracy.

Pros and Cons

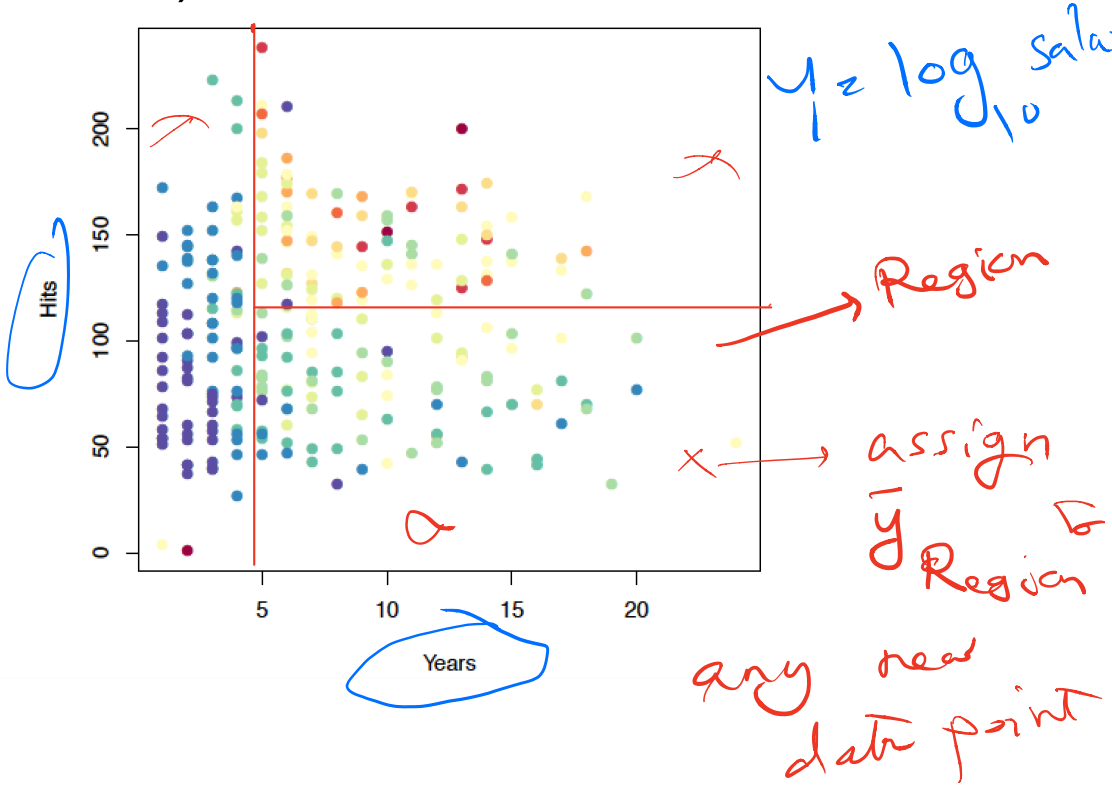
- Hence we also discuss *bagging*, *random forests*, and *boosting*. These methods grow multiple trees which are then combined to yield a single consensus prediction.
- Combining a large number of trees can often result in dramatic improvements in prediction accuracy, at the expense of some loss of interpretation.

The Basics of Decision Trees

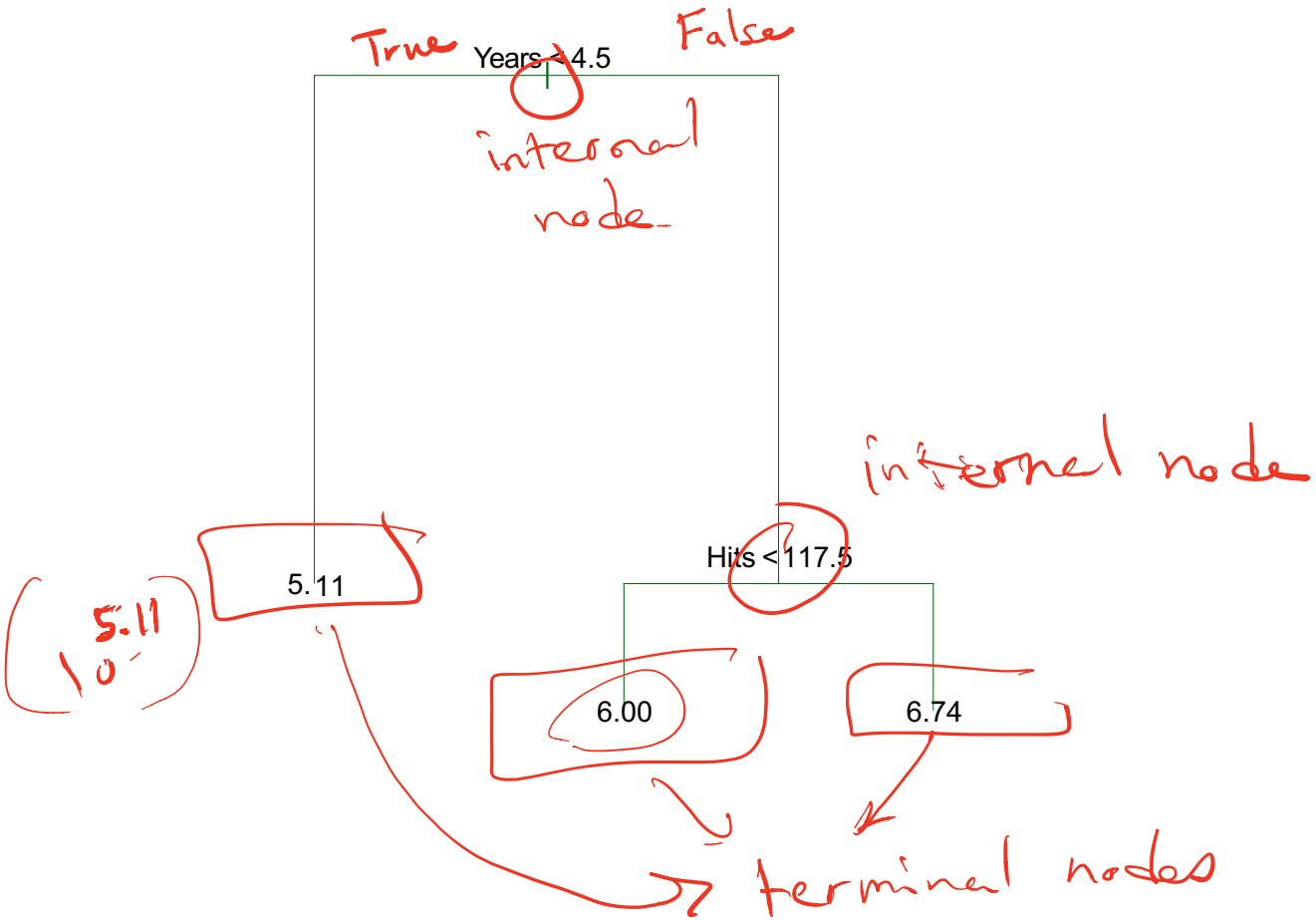
- Decision trees can be applied to both regression and classification problems.
- We first consider regression problems, and then move on to classification.

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.

Details of previous figure

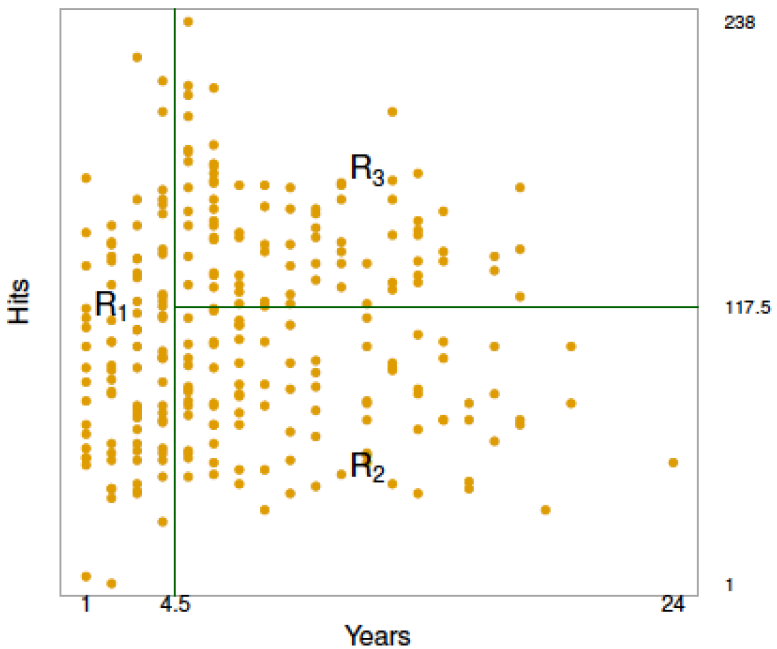
- At a given internal node, the label (of the form $X_j < t_k$) indicates the left-hand branch emanating from that split, and the right-hand branch corresponds to $X_j \geq t_k$. For instance, the split at the top of the tree results in two large branches. The left-hand branch corresponds to Years<4.5, and the right-hand branch corresponds to Years $\geq$ 4.5.

Details of previous figure

- 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 | Years< 4.5}, R_2 ={X | Years ≥ 4.5, Hits<117.5}, and R_3 ={X | Years ≥ 4.5, Hits ≥ 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.

Terminology for Trees

- 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 factor in determining Salary, and players with less experience earn lower salaries than more experienced players.
- Given that a player is less experienced, the number of Hits that he made in the previous year seems to play little role in his Salary.

Interpretation of Results

- For players with an experience of 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.

Interpretation of Results

- Surely an over-simplification, but compared to a regression model, it is easy to display, interpret and explain

Details of the tree-building process

1. We divide the predictor space — that is, the set of possible values for $X_1, X_2, \ldots, X_p$ — into J distinct and non-overlapping regions, $R_1, R_2, \ldots, R_J$.
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.

More details of the tree-building process

- The goal is to find boxes $R_1, \ldots, R_J$ that minimize the RSS, given by

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

RSS for region J

- 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

- 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.

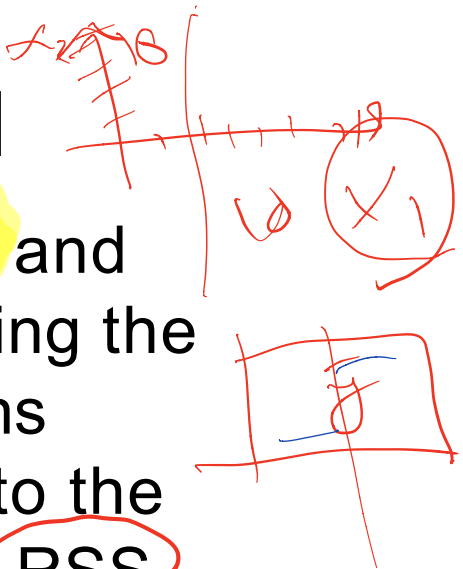
More details of the tree-building process

- The approach is *top-down* because it begins at the top of the tree and then successively splits the predictor space; each split is indicated via two new branches further down on the tree.

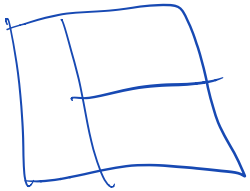
More details of the tree-building process

- 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.

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.

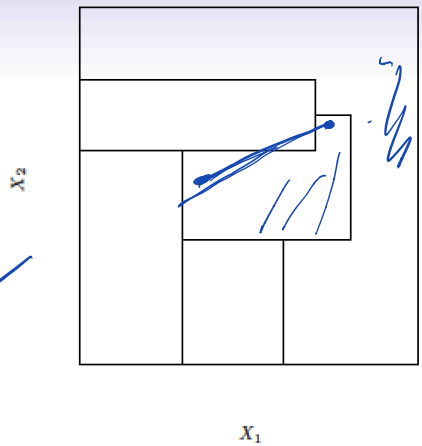


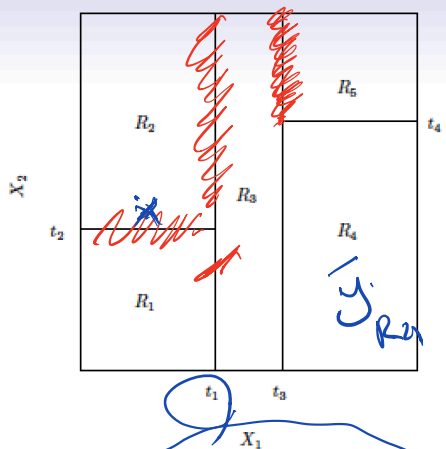
Details— Continued

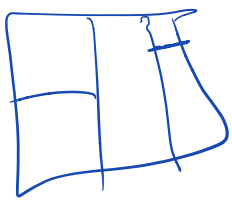
- However, this time, instead of splitting the entire predictor space, we split one of the two previously identified regions. We now have three regions.
- Again, we look to split one of these three regions further, so as to minimize the RSS. The process continues until a stopping criterion is reached; for instance, we may continue until no region contains more than five observations.

Predictions

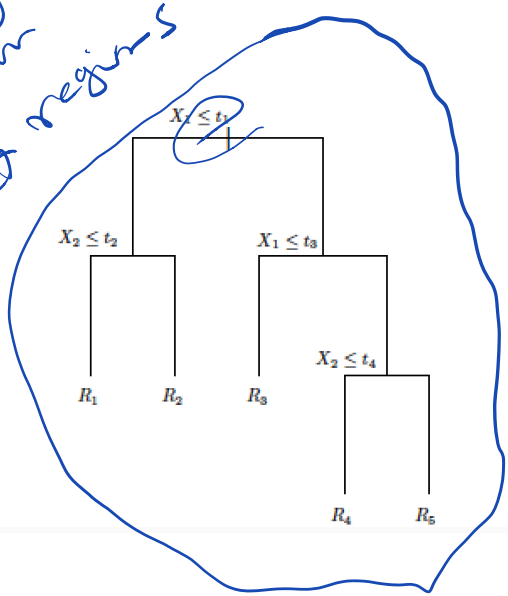
- We predict the response for a given test observation using the mean of the training observations in the region to which that test observation belongs.
- A five-region example of this approach is shown in the next slide.

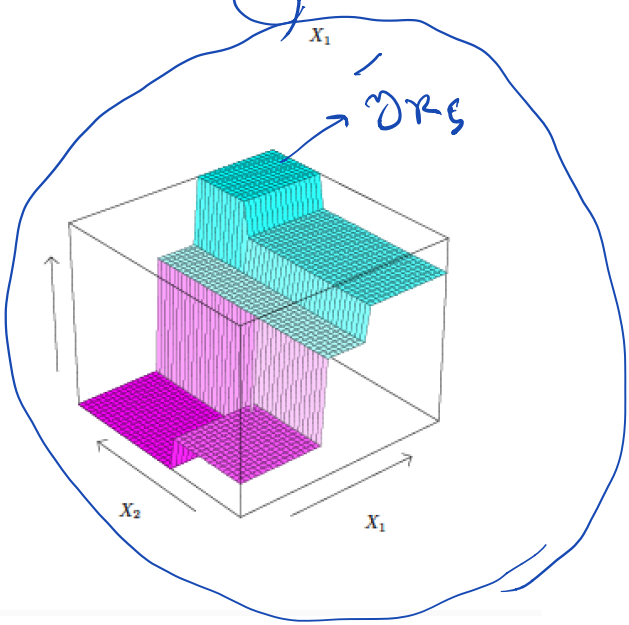






Recursive Binary splitting can only result in convex regions





Problems with the regression surface:

1 - Discontinuity

2 - Being constant over each region

↳ Remedy: Build a (simple) model for each region. Example: Linear regression

⇒ A Model Tree

---

1. Blur the boundaries of the regions by defining "degrees of membership" in each region.

Aristotelian Set Theory/Logic

Membership is 0-1

$x \in A$

$y \notin A$

To assign degrees of membership to members, we use "Fuzzy Set Theory."

(Rule-Based Fuzzy System)

Time

Lotfi A. Zadeh, Prof. @
EECS, UC Berkeley

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?*
- A smaller tree with fewer splits (that is, fewer regions $R_1, \ldots, R_J$) might lead to lower variance and better interpretation at the cost of a little bias.

Pruning a tree

- One possible alternative to the process described above is to grow the tree only so long as the decrease in the RSS due to each split exceeds some (high) threshold.

Pruning a tree

- 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.

Pruning a tree— continued

- A better strategy is to grow a very large tree T_0, and then *prune* it back in order to obtain a *subtree*
- *Cost complexity pruning* — also known as *weakest link pruning* — is used to do this

Pruning a tree— continued

- 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

$$\sum_{m=1}^{|T|} \sum_{i:\, x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha |T|$$

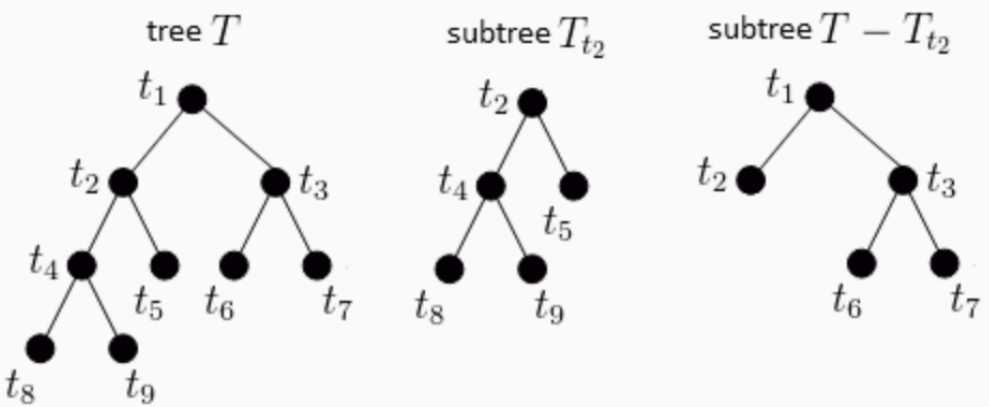
is as small as possible. Here |T| indicates the number of terminal nodes of the tree T, R_m is the rectangle (i.e. the subset of predictor space) corresponding to the m^{th} terminal node, and $\hat{y}_{R_m}$ is the mean of the training observations in R_m.

Pruning a tree— continued

$$\sum_{m=1}^{|T|} \sum_{i:\, x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha |T|$$

Pruning Subtrees

Subtrees:



Choosing the best subtree

- The tuning parameter α controls a trade-off between the subtree's complexity and its fit to the training data.
- We select an optimal value $\hat{\alpha}$ using cross-validation.
- We then return to the full data set and obtain the subtree corresponding to $\hat{\alpha}$.

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.

candidate α's

(Ex: $\alpha \in \{10^{-3}, 10^{-2}, 10^{-1}, 10^{0}, \cdots, 10^{3}\}$

Summary: tree algorithm

3. Use K-fold cross-validation to choose α. For each $k = 1, \ldots, K$:
   3.1 Repeat Steps 1 and 2 on the $\frac{K-1}{K}$ th fraction of the training data, excluding the kth fold.
   3.2 Evaluate the mean squared prediction error on the data in the left-out kth 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.

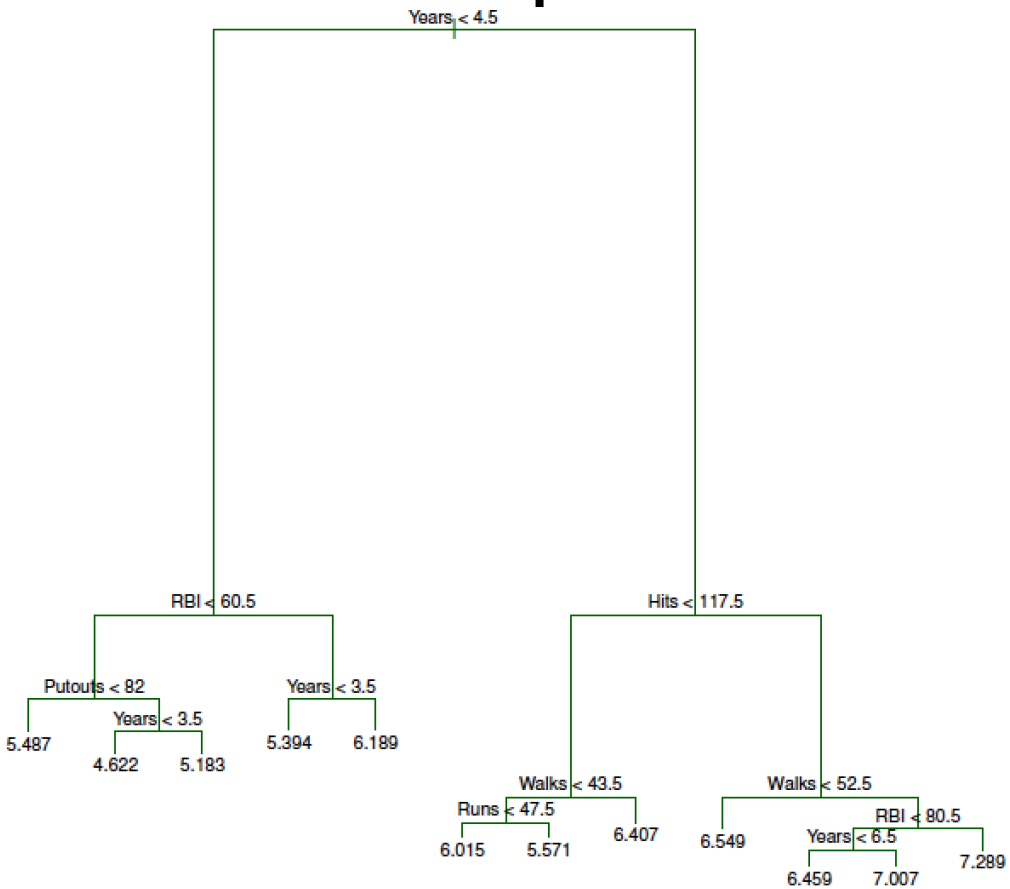
Baseball example continued

- First, we randomly divided the data set in half, yielding 132 observations in the training set and 131 observations in the test set.

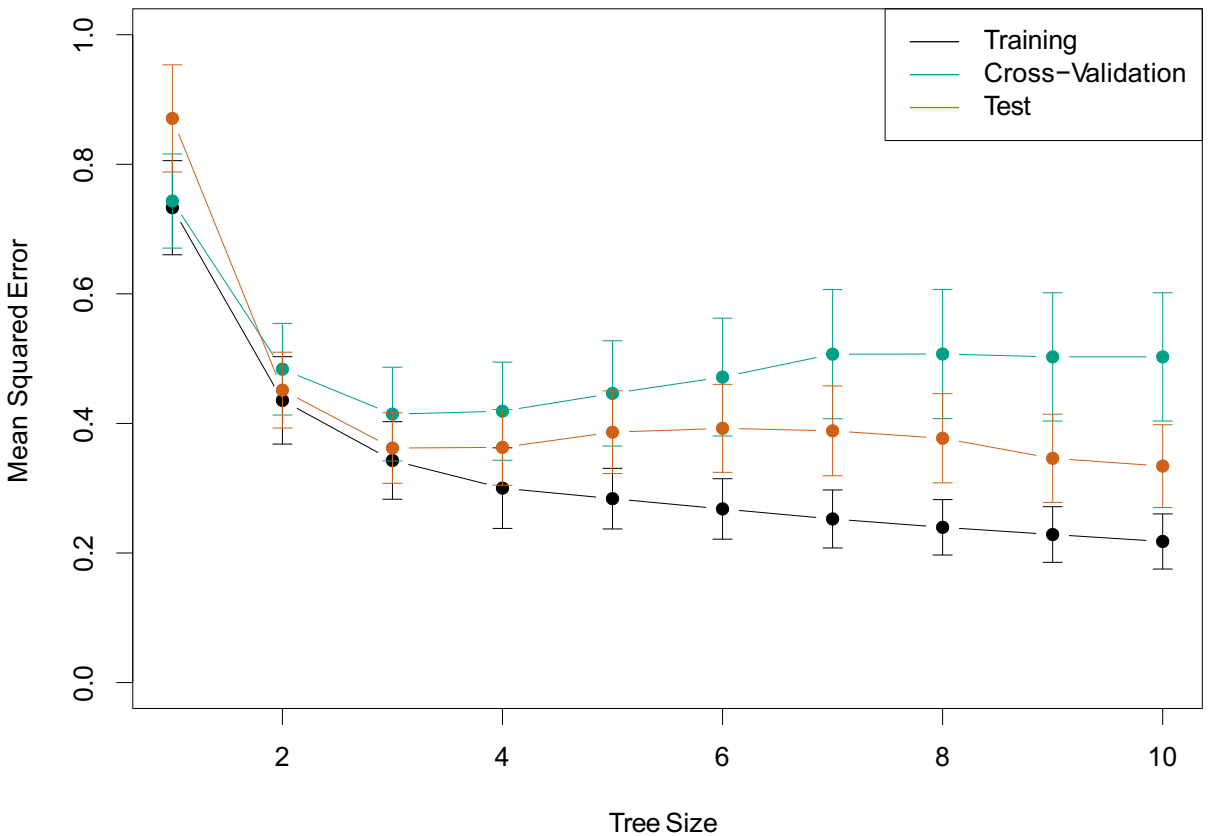
Baseball example continued

- 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

- 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

Details of classification trees

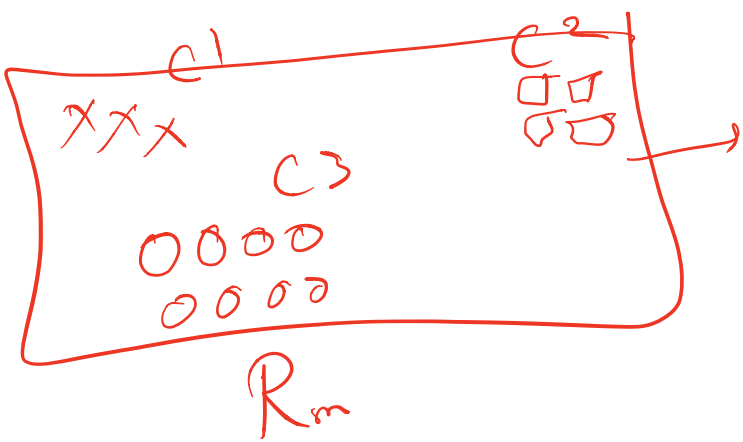
- A natural alternative to RSS is the *classification error rate*, which is simply the fraction of the training observations in that region that do not belong to the most common class:

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

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

Example

A simple example:



Prediction: C3

$\hat{p}_{1m} = \frac{3}{15}$

$\hat{p}_{2m} = \frac{4}{15}$

$\hat{p}_{3m} = \frac{8}{15}$

E

= misclassification rate = $\frac{3+4}{15} = 1 - \frac{8}{15}$

$= 1 - \max_k \hat{p}_{mk}$

Details of classification trees

- 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

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

over all possible classes

a measure of total variance across the *K* classes. The Gini index takes on a *small value* if all of the $\hat{p}_{mk}$'s are close to zero or one.

$\hat{p}_{mk}$ close to 1 or zero is ideal

Gini index and Deviance

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

- 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.
- The Tree Algorithm in this case is called ID3

Gini index and Deviance

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

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

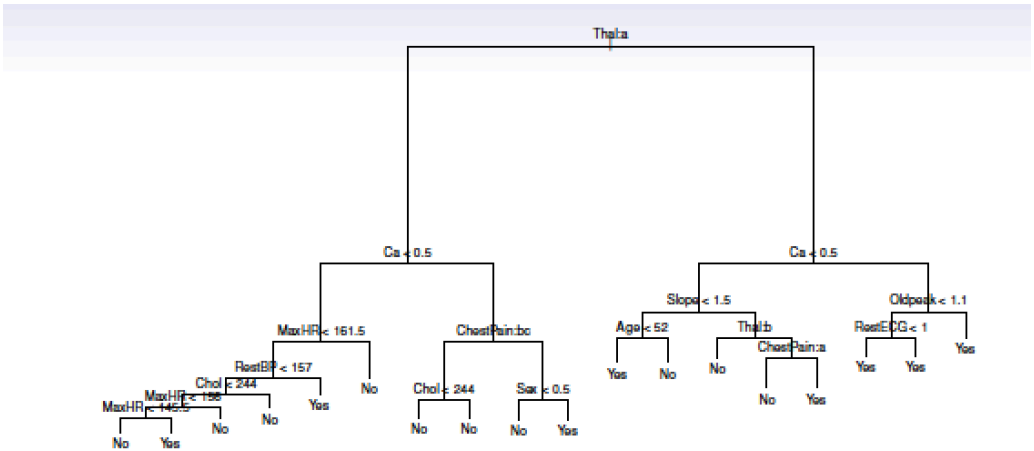
  - The tree algorithm in this case is called C4.5
  - It turns out that the Gini index and the cross-entropy are very similar numerically.

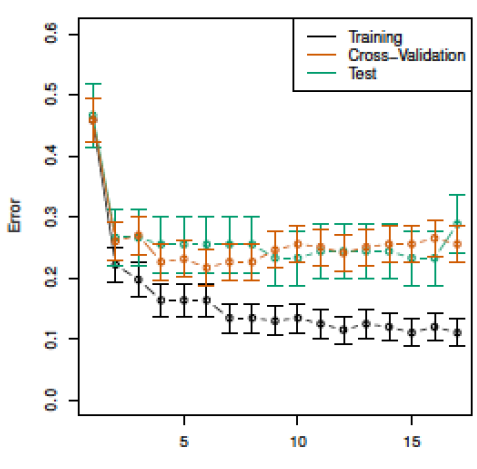
Example: heart data

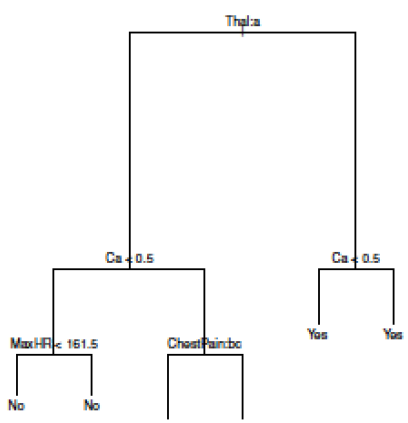
- These data contain a binary outcome HD for 303 patients who presented with chest pain.
- An outcome value of Yes indicates the presence of heart disease based on an angiographic test, while No means no heart disease.

Example: heart data

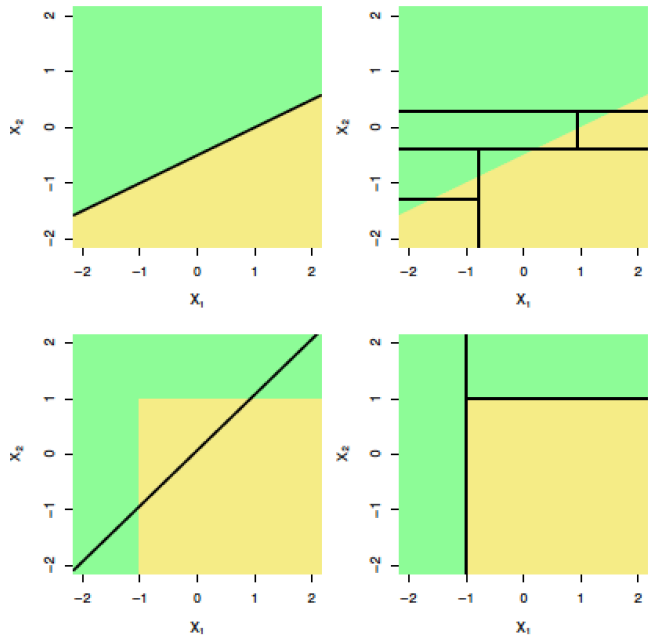
- There are 13 predictors including Age, Sex, Chol (a cholesterol measurement), and other heart and lung function measurements.
- Cross-validation yields a tree with six terminal nodes. See next figure.


Thal:a
Ca < 0.5
Ca < 0.5
MaxHR < 161.5
ChestPain:bc
Slope < 1.5
Oldpeak < 1.1
Age < 52
Thal:b
RestECG < 1
RestBP < 157
Chol < 244
ChestPain:a
MaxHR < 156
MaxHR < 145.5
Chol < 244
Sex < 0.5
No
Yes


Training
Cross–Validation
Test
Error
Tree Size
0.0
0.1
0.2
0.3
0.4
0.5
0.6
5
10
15


Thal:a
Ca < 0.5
Ca < 0.5
MaxHR < 161.5
ChestPain:bc
No
Yes

Trees Versus Linear Models



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 sometimes even easier to explain than linear regression!

Advantages and Disadvantages of Trees

- Some people believe that decision trees more closely mirror human decision-making than do the regression and classification approaches seen in previous chapters.

Advantages and Disadvantages of Trees

- 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.

Advantages and Disadvantages of Trees

- 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.

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

Bagging

- *Bootstrap aggregation*, or *bagging*, is a general-purpose (wrapper) 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.
- It is an *ensemble method*.

Bagging

- Recall that given a set of n independent observations $Z_1, \ldots, Z_n$, 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— continued

- 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

$$\hat{f}_{\text{bag}}(x) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}^{*b}(x).$$

This is called *bagging*.

Bagging classification trees

- The above prescription can be 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.

Out-of-Bag Error Estimation

- It turns out that there is a very straightforward way to estimate the test error of a bagged model.

Out-of-Bag Error Estimation

- Recall that the key to bagging is that trees are repeatedly fit to bootstrapped subsets of the observations.
- One can show that on average, each bagged tree makes use of around two-thirds of the observations.

Out-of-Bag Error Estimation

- The remaining one-third of the observations not used to fit a given bagged tree are called the *out-of-bag* (OOB) observations.

Out-of-Bag Error Estimation

- We can predict the response for the i^{th} observation using each of the trees in which that observation was OOB. This 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.

Random Forests

- *Random forests* provide an improvement over bagged trees by way of a small tweak that *decorrelates* 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.

Random Forests

- 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.

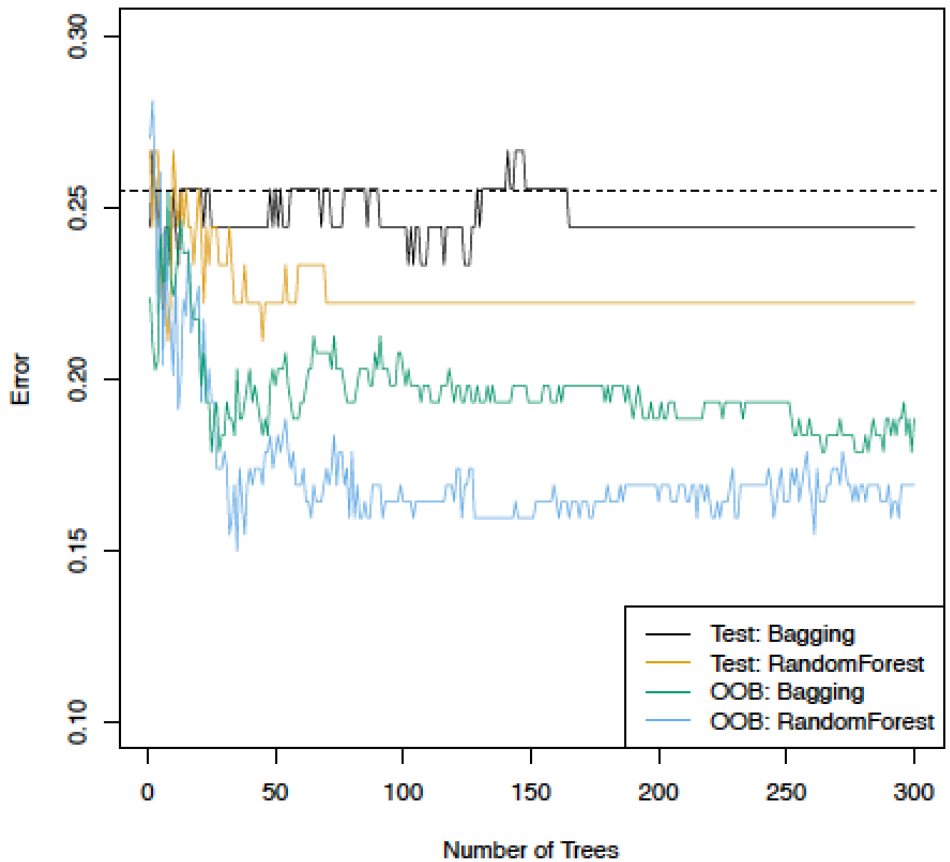
Random Forests

- A fresh selection of m predictors is taken at each split, and typically we choose $m \approx p^{1/2}$ — 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).

Random Forests: An Alternative Version

- Alternatively, one can build *B* decision trees, each of which on a random subset of *m* features.
- Instead of a bootstrap sample from the training set, one can even use a subsample with replacement from the training set, which is equivalent to a bootstrap sample with a size different from the training set.

Bagging the heart data



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 $m = p^{1/2}$.
- 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

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.

Example: gene expression data

- 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.

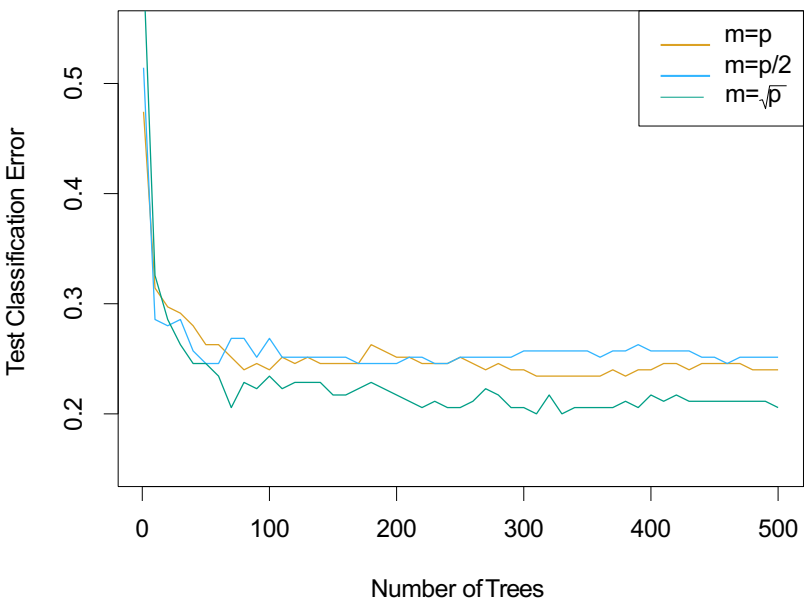
Example: gene expression data

- 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.

Example: gene expression data

- 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



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.

Details of previous figure

- Random forests ($m < p$) lead to a slight improvement over bagging ($m = p$). A single classification tree has an error rate of 45.7%.

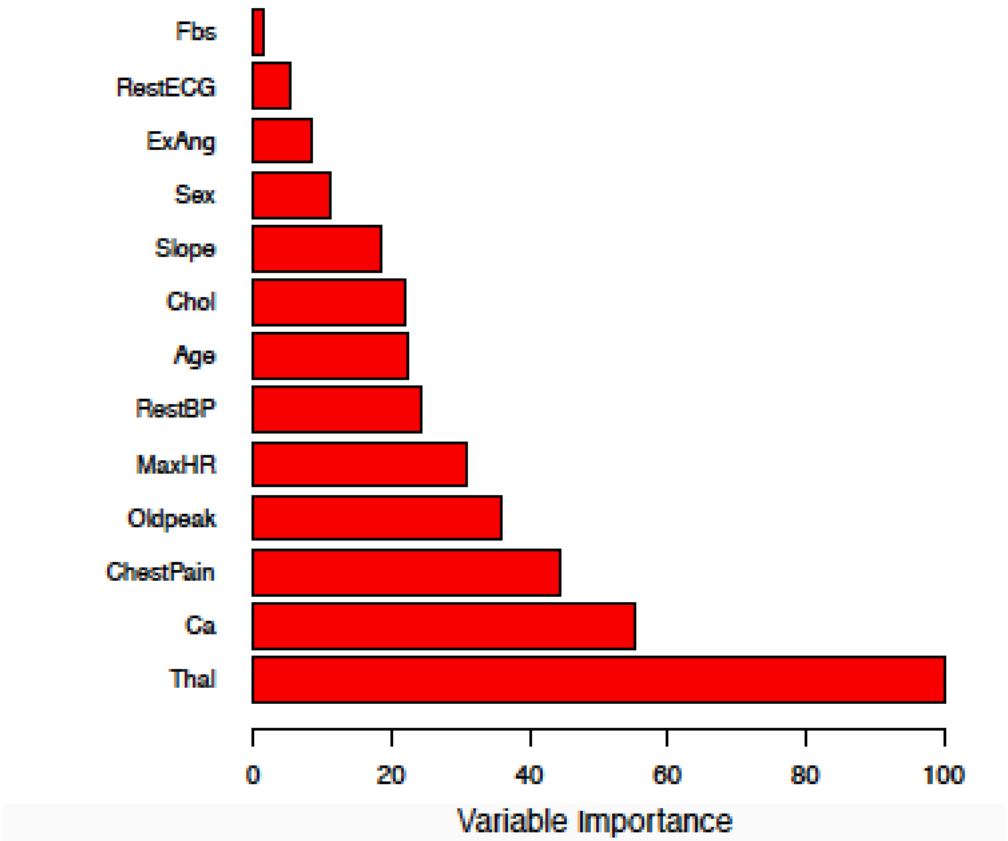
Variable importance measure

- 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.

Variable importance measure

- 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.

Variable importance measure



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

Summary

- 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.

Appendix
More on Ensemble Methods

Stacking

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.

Boosting

- 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.

Boosting

- 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 algorithm for regression trees

1. Set $\hat{f}(x) = 0$ and $r_i = y_i$ for all i in the training set.
2. For $b = 1, 2, \ldots, B$, repeat:
    1. Fit a tree $\hat{f}^b$ with d splits ($d + 1$ terminal nodes) to the training data (X, r).
    2. Update $\hat{f}$ by adding in a shrunken version of the new tree:

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

    3. Update the residuals,

$$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i).$$

Boosting algorithm for regression trees

Output the boosted model,

$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x)$$

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*.

What is the idea behind this procedure?

- 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.

What is the idea behind this procedure?

- Each of these trees can be rather small, with just a few terminal nodes, determined by the parameter *d* in the algorithm.

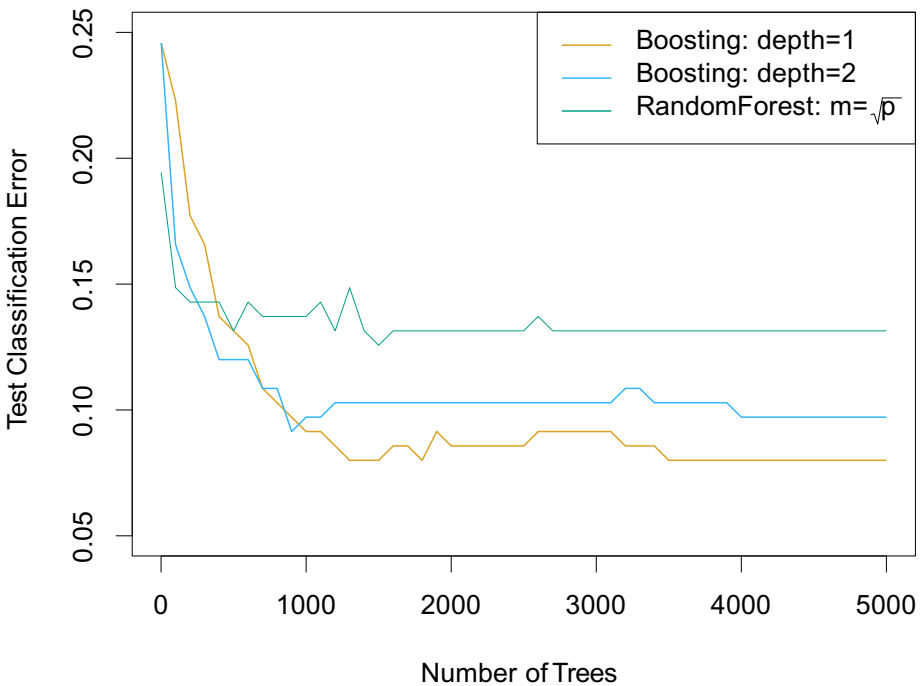
What is the idea behind this procedure?

- 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 (see appendix), nor does the text book.
- More details in *Elements of Statistical Learning, chapter 10 or Appendix.*
- The R package gbm (gradient boosted models) handles a variety of regression and classification problems.

Gene expression data continued



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*.

Details of previous figure

- 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.

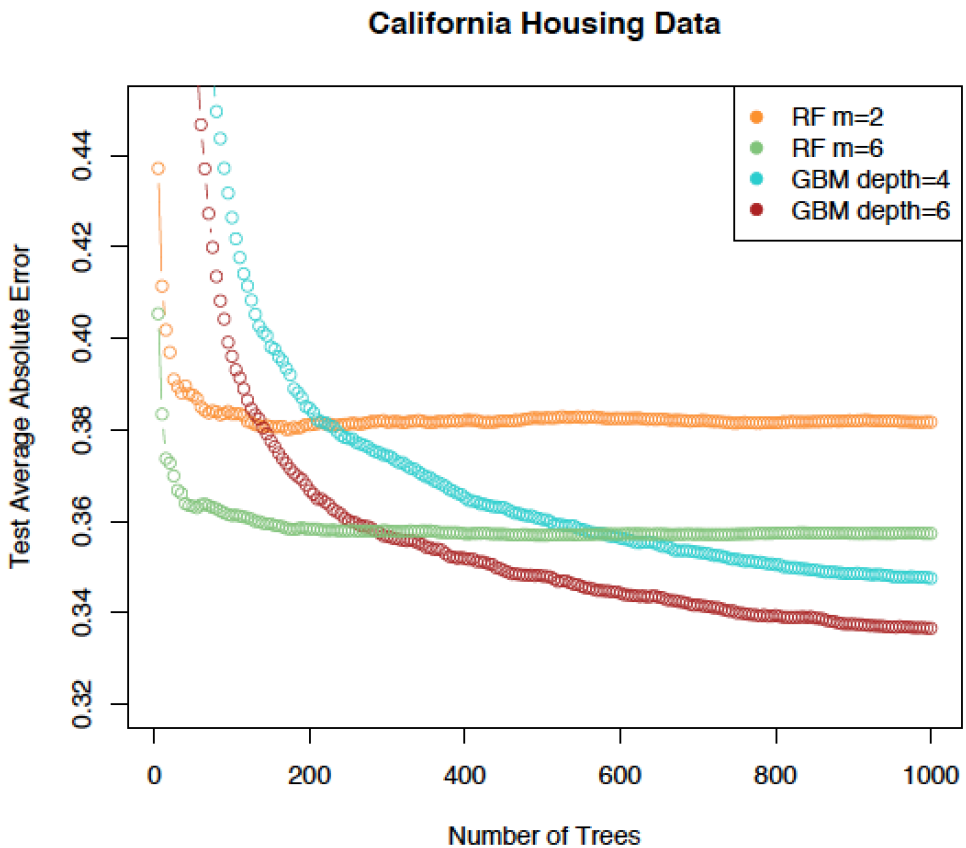
Tuning parameters for boosting

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.

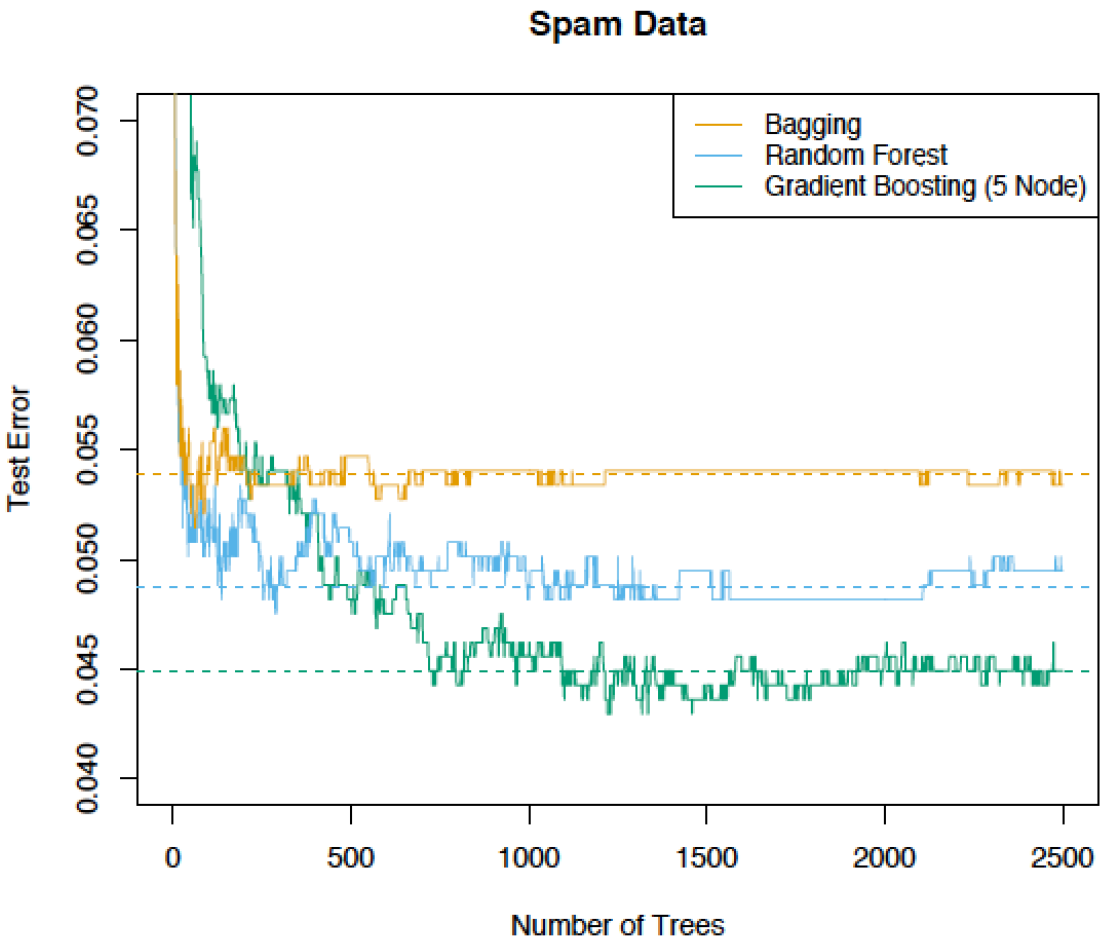
Tuning parameters for boosting

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



Another classification example



Appendix

Extra Trees

Extra Trees

- Extremely Randomized Trees were proposed by Geurts et al. in 2006.
- The idea is to make the trees even weaker, but to compensate that with the large number of estimators in the ensemble.

Extra Trees

More specifically, not only the features and samples shown to the tree are randomized, but also the growing of individual trees is random.

Extra Trees

- In particular the split point i.e., the thresholds of comparisons is randomized. This makes training each tree faster.
- Implemented as sklearn.ensemble.ExtraTreesClassifier.

Boosting (Appendix)

Boosting for Classification

Boosting iteratively learns weak classifiers; a weak classifier is one whose error rate is only slightly better than random guessing.

Boosting (Appendix)

Boosting for Classification

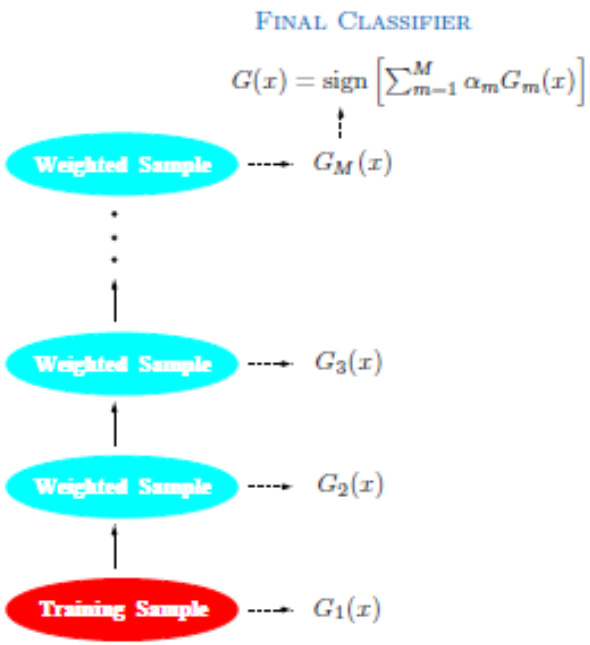
The purpose of boosting is to sequentially apply the weak classification algorithm to repeatedly modified versions of the data, thereby producing a sequence of weak classifiers.

The predictions from all of them are then combined through a weighted majority vote to produce the final prediction.

Boosting (cont.)

Boosting for Classification (cont'd)

Thus, the final result is the weighted sum of the results of weak classifiers.

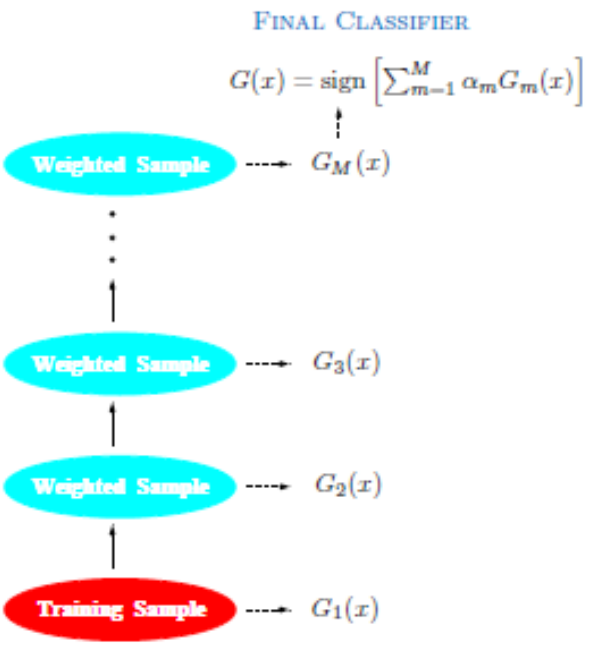


Boosting (cont.)

Boosting for Classification (cont'd)

The alphas in the final classifie are computed by the boosting algorithm, which weight the contribution to give higher influence to more accurate classifiers in the sequence.

Weights are modified at each boosting step!



Boosting (cont.)

Up-weight data that are difficult; incorrectly classified in the previous round. **Down-weight data** that are easy; correctly classified in the previous round.

Boosting (cont.)

There are many different boosting algorithms for classification:

AdaBoost, LPBoost, BrownBoost, LogitBoost, Gradient Boosting, etc.

Boosting (cont.)

Example of Boosting Algorithm (AdaBoost):

Algorithm 10.1 *AdaBoost.M1.*

1. Initialize the observation weights $w_i = 1/N,\ i = 1, 2, \ldots, N$.
2. For $m = 1$ to M:
   (a) Fit a classifier $G_m(x)$ to the training data using weights w_i.
   (b) Compute
   $$\text{err}_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}.$$
   (c) Compute $\alpha_m = \log((1 - \text{err}_m)/\text{err}_m)$.
   (d) Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))],\ i = 1, 2, \ldots, N$.
3. Output $G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$.

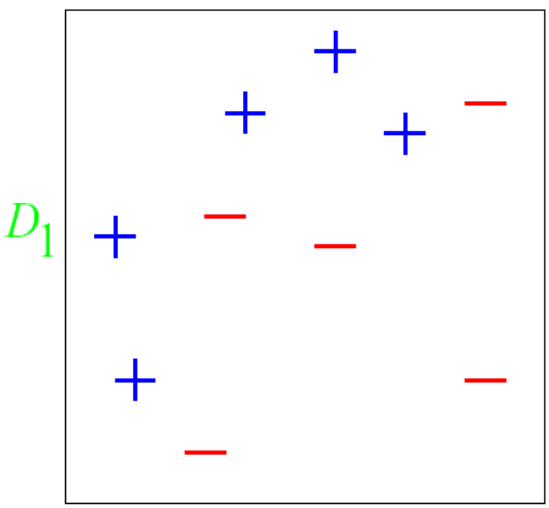
Boosting Intuition

- We adaptively weigh each data case.
- Data cases which are wrongly classified get high weight (the algorithm will focus on them)
- Each boosting round learns a new (simple) classifier on the weighed dataset.

Boosting Intuition

- These classifiers are weighed to combine them into a single powerful classifier.
- Classifiers that that obtain low training error rate have high weight.
- We stop by using monitoring a hold out set (cross-validation).

An Illustrative Example



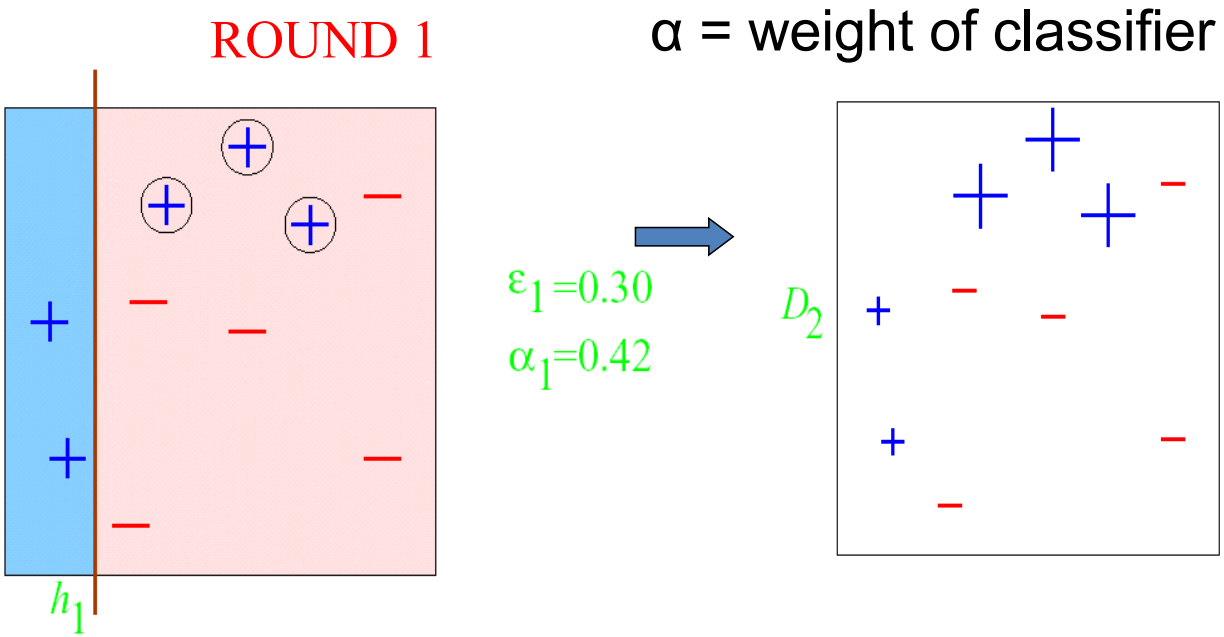
Original training set: equal weights to all training samples

Taken from "**A Tutorial on Boosting" by** Yoav Freund and Rob Schapire

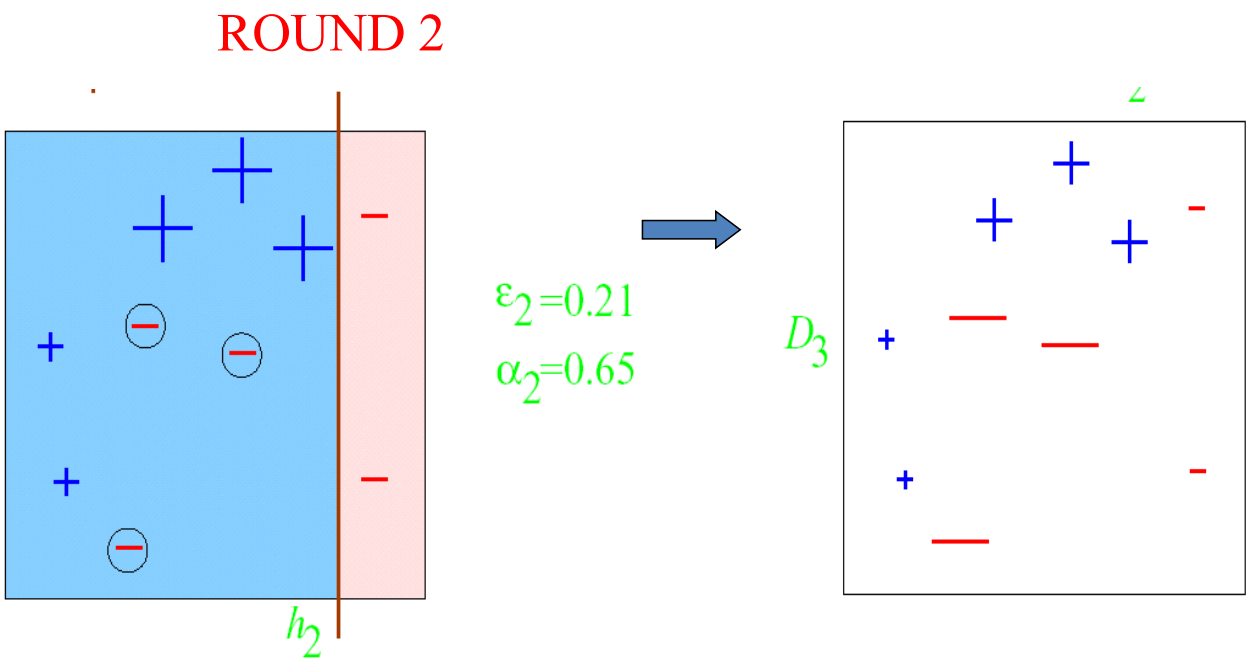
AdaBoost example

ε = error rate of classifier

α = weight of classifier

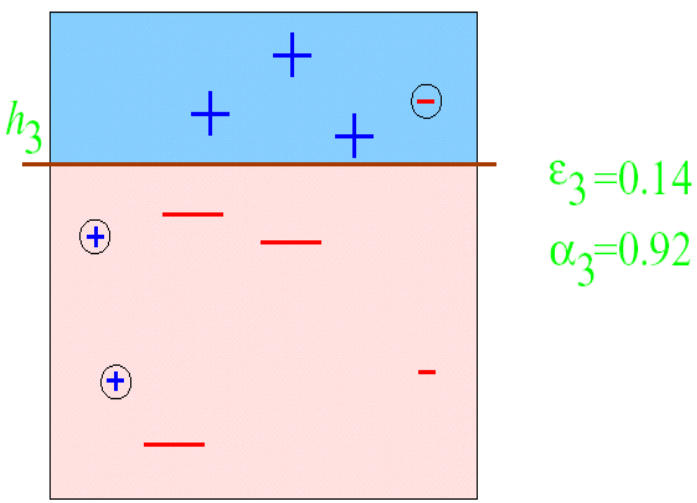


AdaBoost example



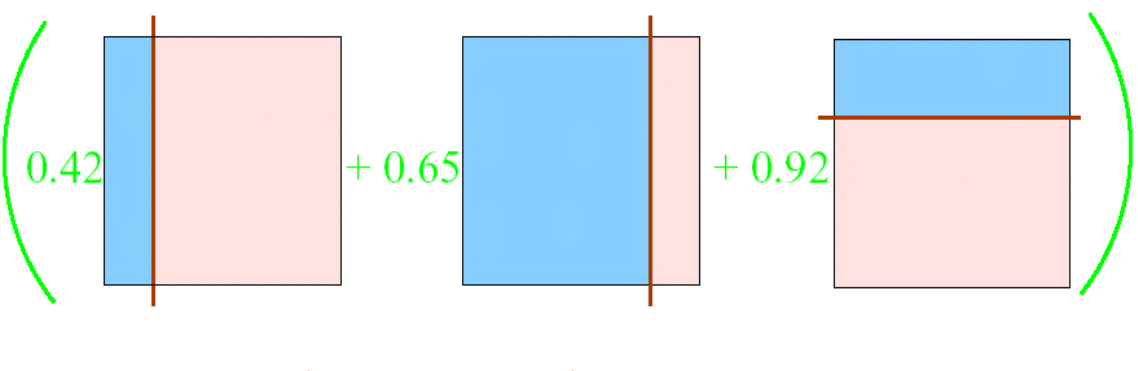
AdaBoost example

ROUND 3



AdaBoost example

$$H_{\text{final}} = \text{sign}\left(0.42 \; + 0.65 \; + 0.92 \right)$$



Boosting (cont.)

Boosting is remarkably resistant to overfitting, and it is fast and simple.

In fact, it can often continue to improve even when the training error has gone to zero.

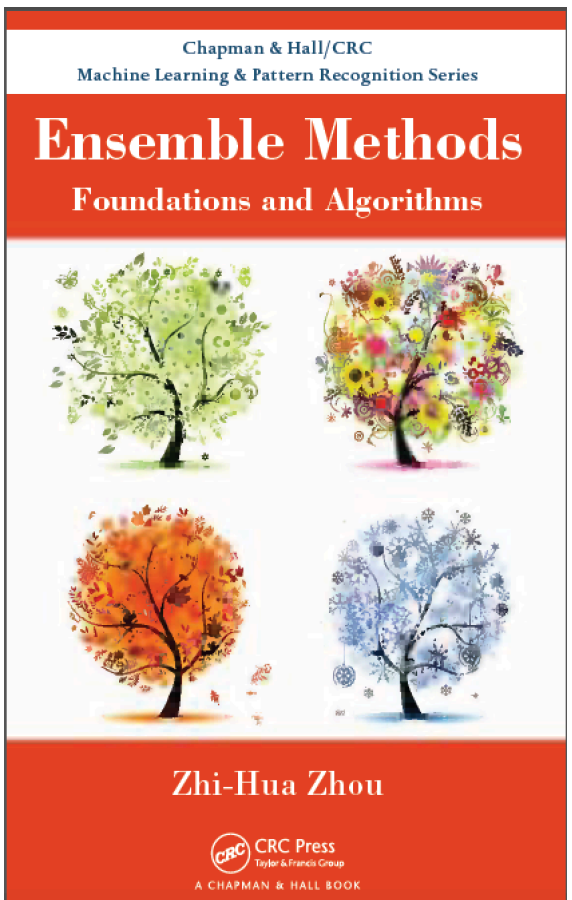
Boosting (cont.)

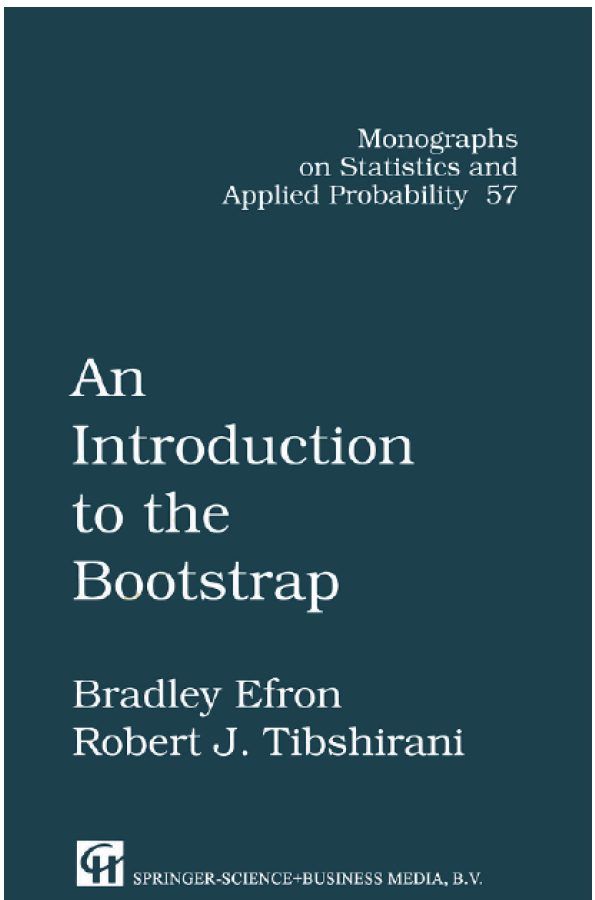
It improves the performance of many kinds of machine learning algorithms.

Boosting does not work when:

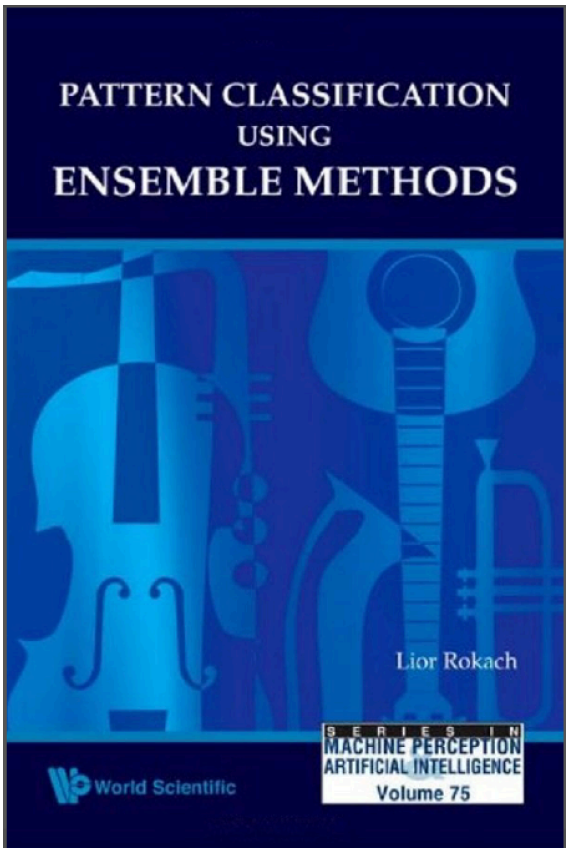
- Not enough data, base learner is too weak or too strong, and/or susceptible to noisy data.

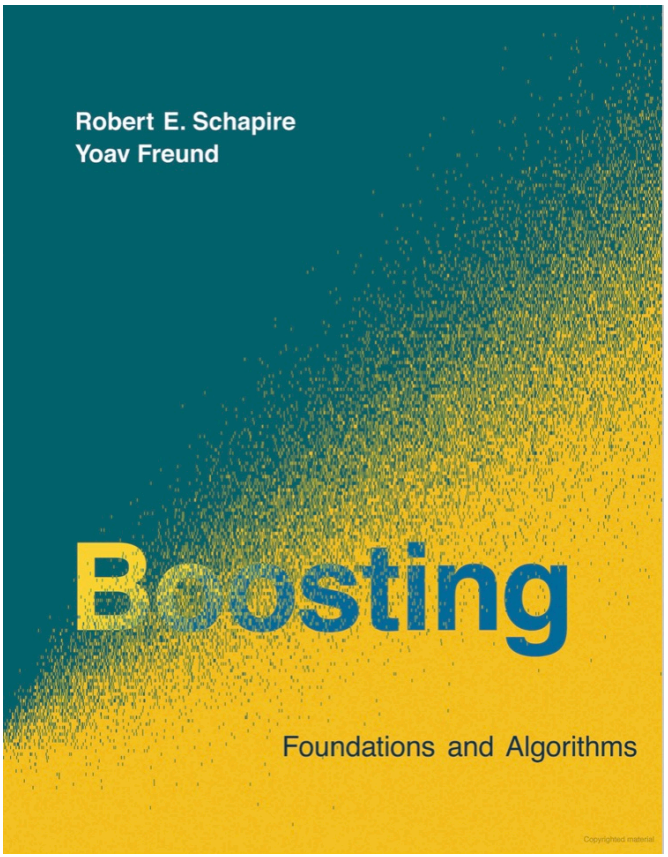
More


Chapman & Hall/CRC
Machine Learning & Pattern Recognition Series
Ensemble Methods
Foundations and Algorithms
Zhi-Hua Zhou
CRC Press
Taylor & Francis Group
A CHAPMAN & HALL BOOK


Monographs
on Statistics and
Applied Probability 57
An
Introduction
to the
Bootstrap
Bradley Efron
Robert J. Tibshirani
SPRINGER-SCIENCE+BUSINESS MEDIA, B.V.

More


PATTERN CLASSIFICATION
USING
ENSEMBLE METHODS
Lior Rokach
SERIES IN
MACHINE PERCEPTION
ARTIFICIAL INTELLIGENCE
Volume 75
World Scientific

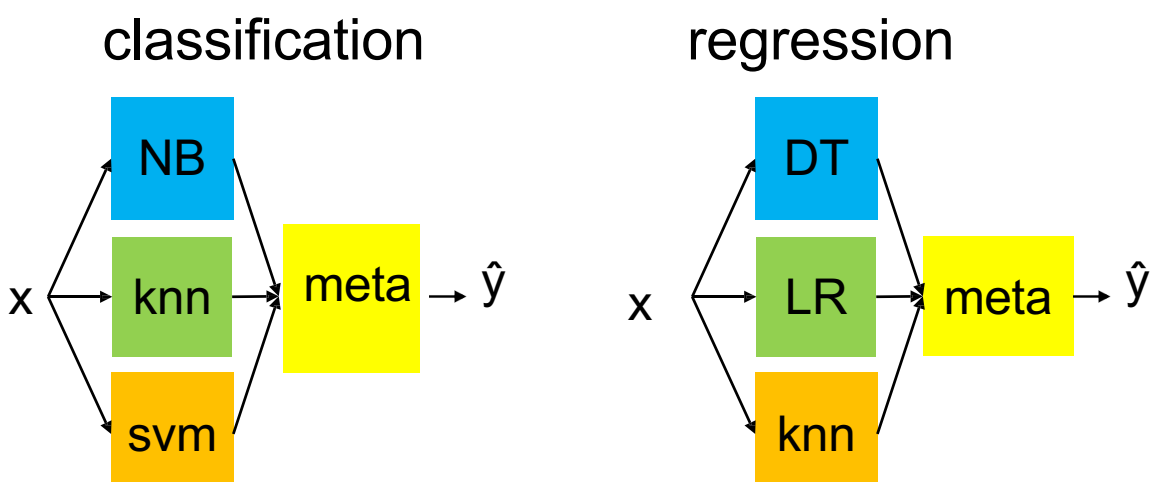

Robert E. Schapire
Yoav Freund
Boosting
Foundations and Algorithms

Appendix
More on Ensemble Methods

Stacking

Stacking

- Basic idea: use the output of multiple classifiers as input to a meta-model (meta-learner).
- We ‘stack’ the meta-model on top of the base models



Stacking – a naïve approach

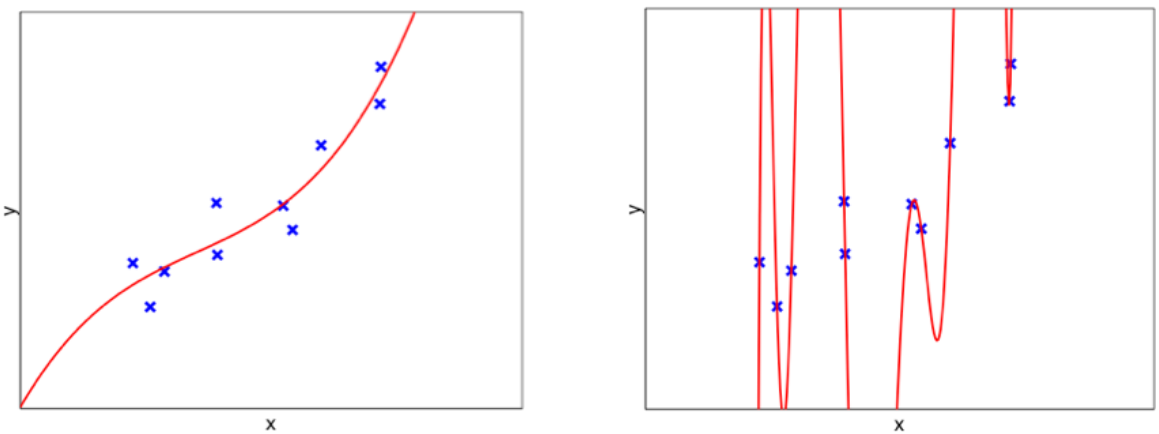
- Let's consider the regression case
- Base model predictions are $f_1(x)$,. .., $f_L(x)$
- Meta learner could be a linear regression model

$$f_{\text{meta}}(\mathbf{x}) = \sum_{i=1}^{L} w_i f_i(\mathbf{x})$$

- If we could choose w_i to minimize true error, the stacked model would always be at least as good as any base model!
  - Worst case we set all w_i to 0 except that of the best base model
- But what if we minimize the train error instead?

Stacking – a naïve approach

- Consider the following base models



- What weights would minimize train set error?
- Does that yield good generalization error for the meta model?

Stacking – a naïve approach

- Naïve implementation of stacking prefers over-fitted models
- Underlying problem: the outputs of the base models have been adapted to the labels.

Stacking – a naïve approach

- Thus, inputs of the meta model are not representative of the inputs it will get at test-time.
- To avoid preference for overfitted models, inputs to the meta- model should not have seen the labels for the data points themselves

Stacking – second attempt

- Now, we can train the meta-model on the data in the base model outputs paired with the target label
- Any base-model output is now a good indication of test-time behavior
- If the meta-model has free parameters itself, we can cross-validate using the same folds

Stacking – second attempt

- Usually, the meta-model is relatively simple (e.g. linear regression or logistic regression)
- Empirical Recommendation: Do not have your base-learners as the meta-learner. For example, if you use logistic regression as the base-learner, use some other classifier (e.g. SVM) as the meta-learner.

Testing the stacked model

- To test the stacked model, again we set aside a test set from the very beginning
- Have several versions of the base models from cross-validation!

Testing the stacked model

- Two approaches:
  - Retrain the base models on the whole dataset
    - Possible disadvantage: slightly different input to meta-model
  - Use an average of the trained base models
    - Possible disadvantage: time cost
- Then feed the base model predictions into the trained meta- model

Comparison to model selection

- If we force meta-learner to use just one base model (with weight 1) and set all other weights to 0, this is equivalent to selecting the best model with cross-validation
- More expressive meta-models (e.g. linear / logistic regression) can leverage the relative strength of multiple models

Comparison to model selection

- A very complex meta-model (e.g. decision tree) could again easily overfit
- Could use cross-validation on the meta-level to ensure good generalization properties

Effectiveness of stacking

- Stacking generally improves performance, but not by much
- Additional cost of training and evaluating multiple models

Effectiveness of stacking

- If interpretability or speed are important consideration, stacking might not help you much.
- In competitions where a small gain is important and time cost is not so much of an issue, it is usually effective!
- Quite useful in collaborative approaches where everyone can integrate their own model in overall system

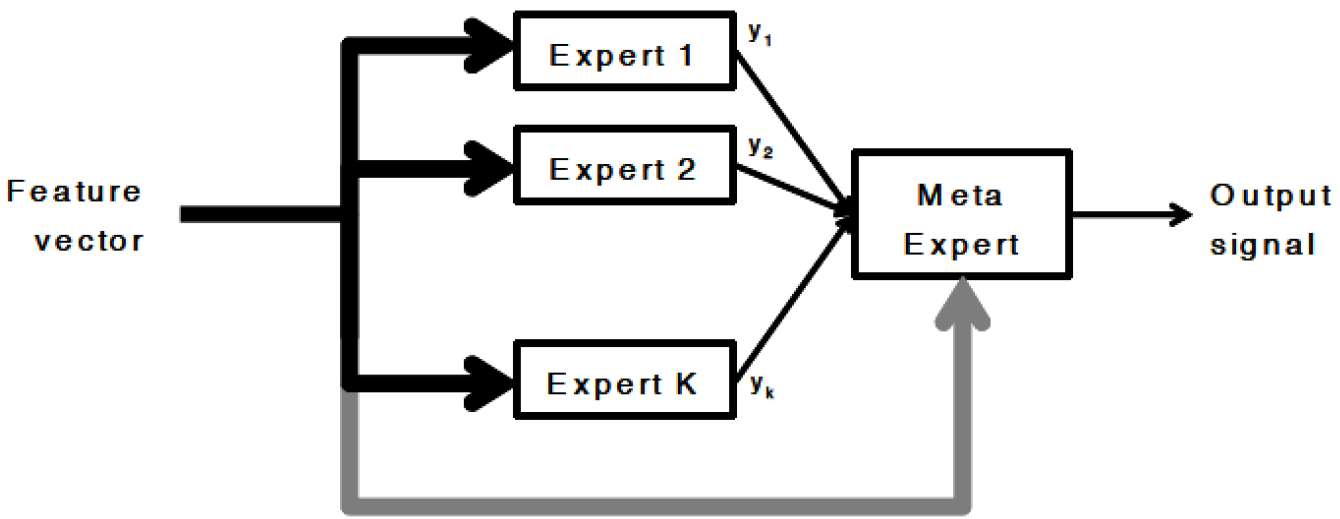
Adaptive Meta-Learners

- The meta-learner is a function that depends on the input feature vector
- Thus, the ensemble implements a function that is local to each region in feature space
- This divide-and-conquer approach leads to modular ensembles where relatively simple classifiers specialize in different parts of I/O space

Adaptive Meta-Learners

- In contrast with static-combiner ensembles, the individual experts here do not need to perform well for all inputs, only in their region of expertise
- Representative examples of this approach are Mixture of Experts (ME) and Hierarchical ME [Jacobs et al., 1991; Jordan and Jacobs, 1994]

Adaptive Meta-Learners



Mixture of Experts

- **ME is the classical adaptive ensemble method**
- A gating network is used to partition feature space into different regions, with one expert in the ensemble being responsible for generating the correct output within that region [Jacobs et al., 1991]

Mixture of Experts

- The experts in the ensemble and the gating network are trained simultaneously.
- ME can be extended to a multi-level hierarchical structure, where each component is itself a ME.