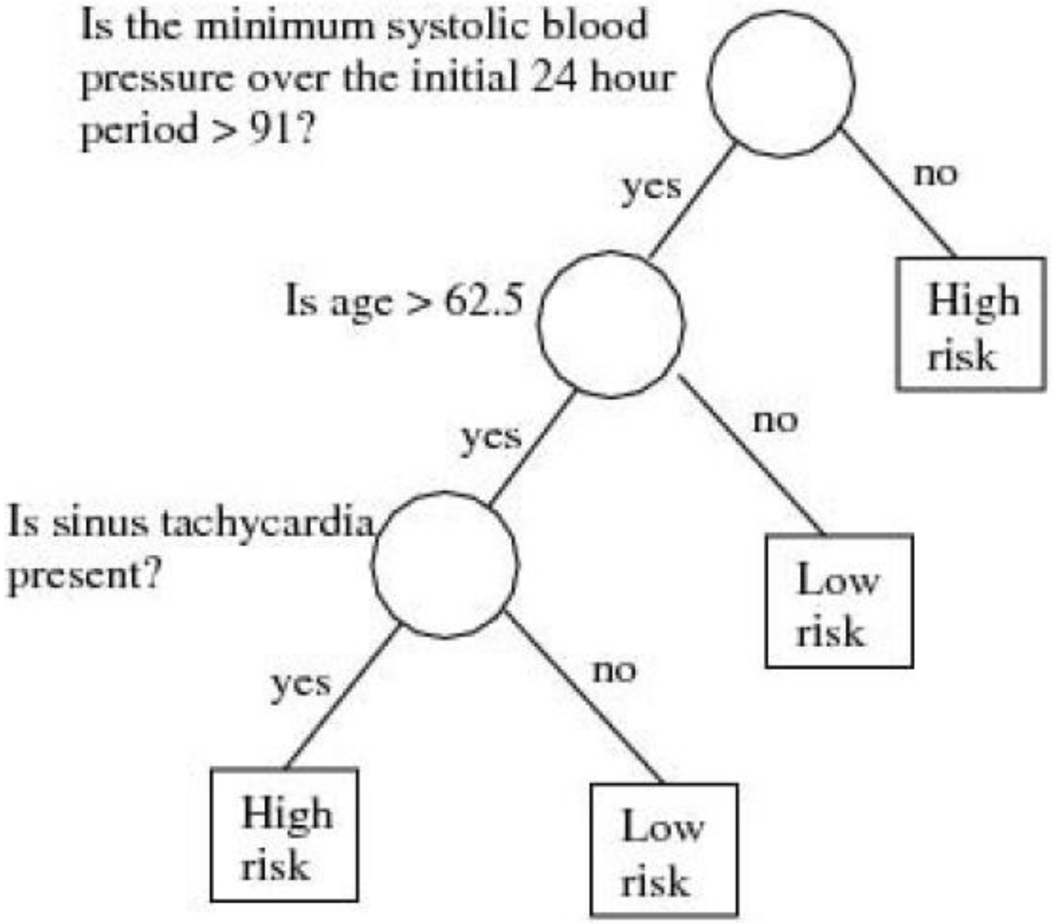
Topic 8. Classification and Regression Trees (CART)

Tree-based Methods

- Tree-based methods involve segmenting the predictor space into a number of simple regions.
- In each region, we typically use the mean or the mode of the training observations in the region to make prediction.
- Region-wise constant, and completely nonparametric
- A medical example
  - Predict high risk patients who will not survive at least 30 days on the basis of the initial 24-hour data.
  - 19 variables are measured during the first 24 hours. These include blood pressure, age, etc.

An Example of Classification Tree


Is the minimum systolic blood pressure over the initial 24 hour period > 91?
yes
no
High risk
Is age > 62.5
yes
no
Is sinus tachycardia present?
Low risk
yes
no
High risk
Low risk

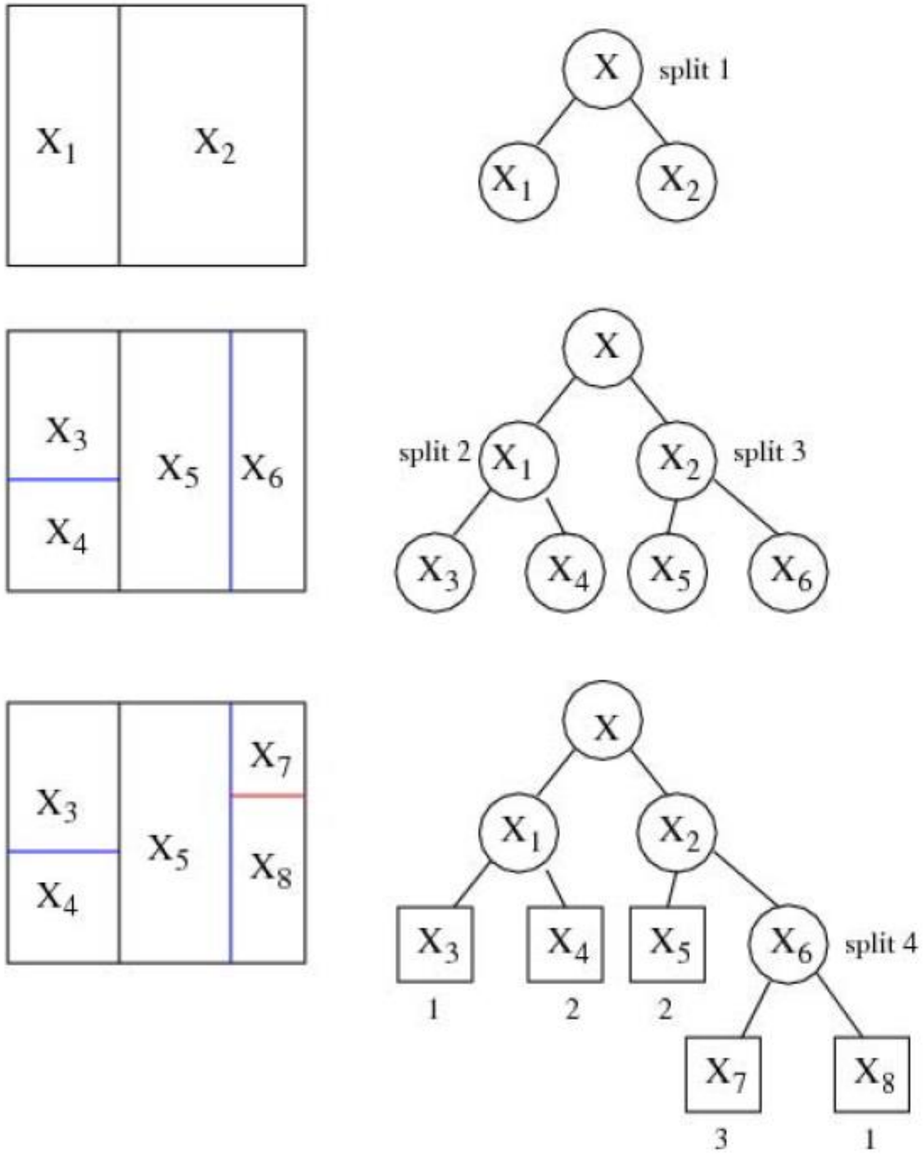
Some Notations

- We start from classification tree.
- $X = (X_1, X_2, \dots, X_p)^T \in \psi$, some of which may be categorical.
- Trees are constructed by repeated splits of subsets of ψ into two descendant subsets, beginning with ψ itself.
- Terminology: node, root node, terminal node (leaf node), parent node, child node, sibling node.
- Two child nodes form a partition of the region occupied by their parent node.
- Every leaf node is assigned with a class.

Some Notations (Cont.)

- A node will have only one parent node except for the root node with none.
- A node is denoted by t.
- Its left child node is denoted by t_L and right by t_R.
- The collection of all the nodes is denoted by T.
- The collection of all the leaf nodes is denoted by $\tilde{T}$.
- A split is denoted by s, and the set of splits is denoted by S.

Tree Building



Key Elements of A Tree

- The construction of a tree involves

  1. The decisions when to declare a node terminal or to continue splitting it
  2. The selection of the splits
  3. The assignment of each terminal node to a class

- In particular

  1. A set of binary questions used as splits
  2. A goodness of split criterion $\phi(s,t)$ that can be evaluated for any split s of any node t
  3. A stop-splitting rule
  4. A rule for assigning every terminal node to a class

1. Standard Set of Questions

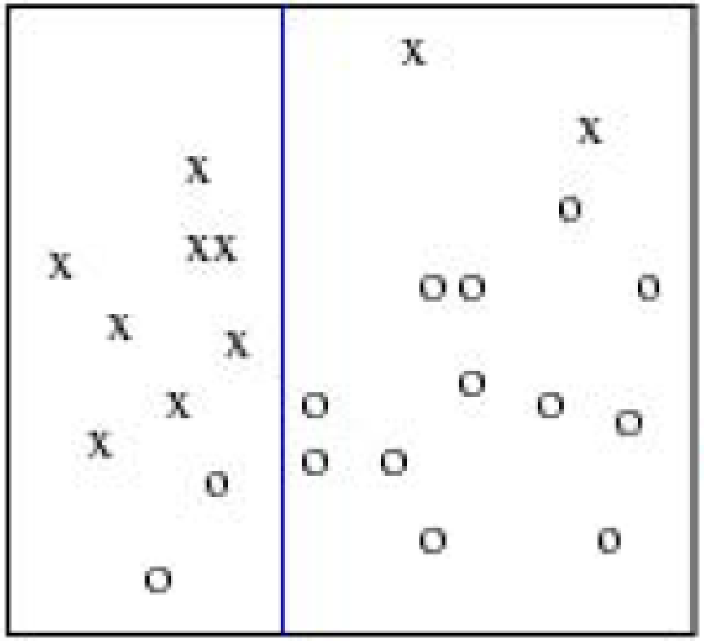
- Each split depends on the value of only one variable.
- For each variable X_j, S includes all questions of the form

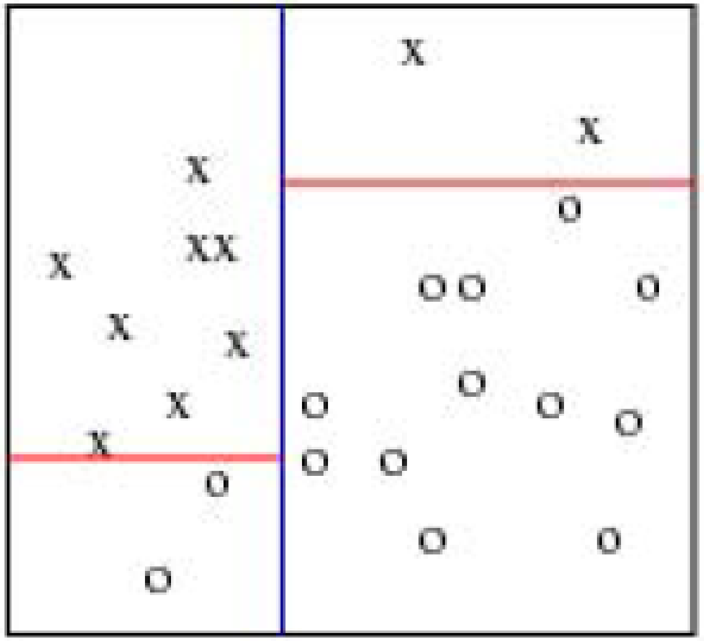
$$\{\text{Is } X_j \in A?\}$$

  - If X_j is continuous or ordinal, $A = (-\infty, c]$.
  - If X_j is categorical, A is a subset of categories.
- The splits for all p variables constitute the standard set of questions.

2. Goodness of Splits

- The goodness of split is measured by an **impurity** function defined for each node.
- Intuitively, we want each leaf node to be “pure”, that is, one class dominates.





The Impurity Function

- Define $p_{tk} = P(Y = k|t) = |t|^{-1} \sum_{i \in t} I(y_i = k)$, possible impurity functions $G(t)$ are
  - Misclassification error: $1 - \max_k p_{tk}$
  - Gini index: $\sum_k p_{tk}(1 - p_{tk}) = 1 - \sum_k p_{tk}^2$
  - Cross entropy: $-\sum_k p_{tk} \log(p_{tk})$

- Goodness of split: $\phi(s, t) = G(t) - p_L G(t_L) - p_R G(t_R)$, where p_L and p_R are proportions of the samples in node t that go to the left child t_L and the right child t_R, respectively.

3. Stopping Criteria

- A simple criterion: stop splitting a node t when

$$\max_{s \in S} \frac{|t|}{n} \phi(s, t) < \beta$$

where β is a pre-specified threshold.

- The idea is to stop splitting when the node is sufficiently pure or sufficiently small.

- The above stopping criteria may not work well; other more sophisticated stopping rules are available.

4. Class Assignment Rule

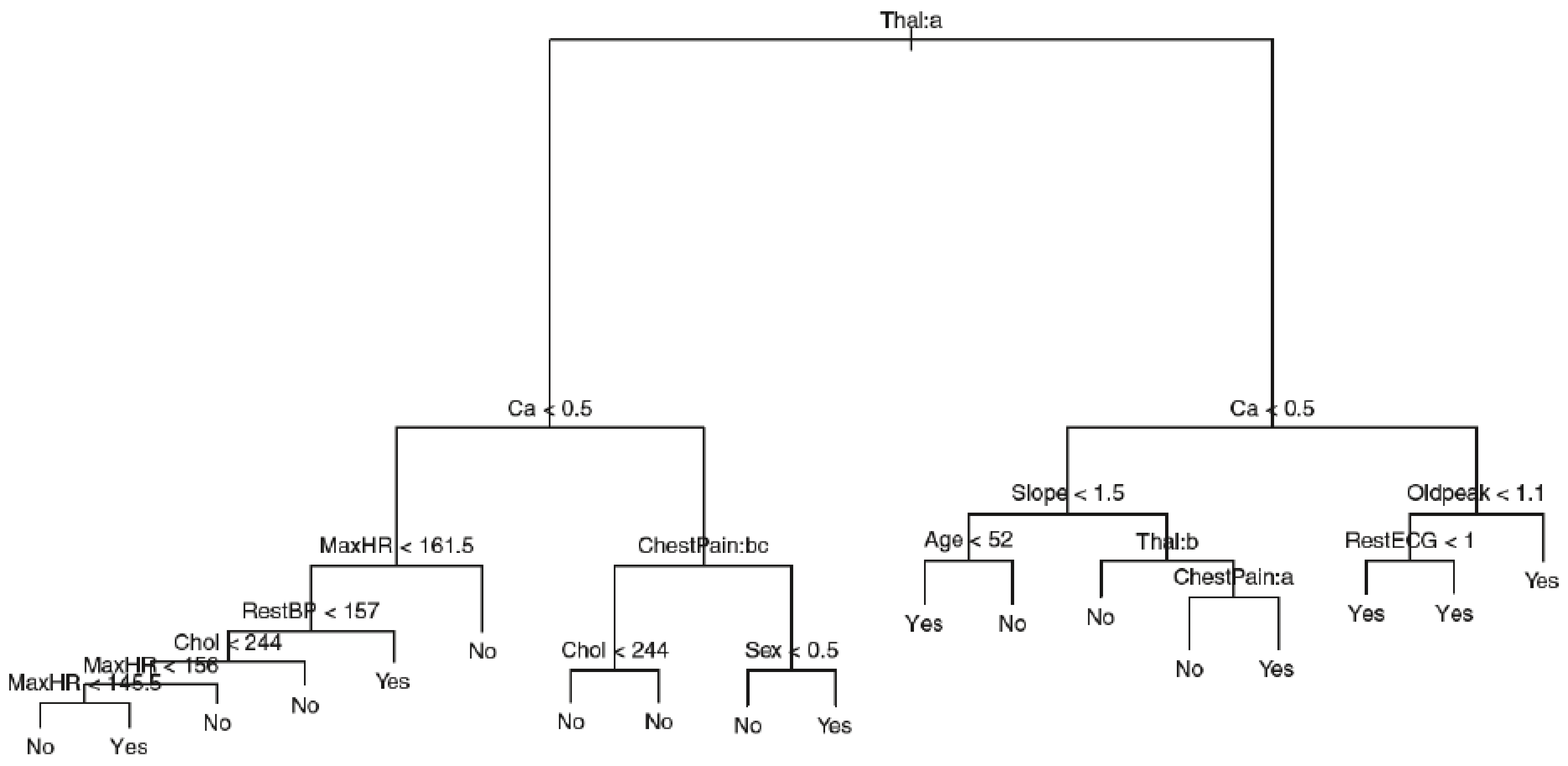
- A class assignment rule assigns a class $k = \{1, \ldots, K\}$ to every terminal node $t \in \tilde{T}$.

- The class assigned to node $t \in \tilde{T}$ is denoted by $\kappa(t)$.

- For 0-1 loss, the class assignment rule is

$$\kappa(t) = \underset{k}{\operatorname{argmax}}\, \hat{p}_{tk}$$

Example of Classification Tree



Right-sized Trees

- Denote the misclassification error of a tree T by $R(T)$, then

$$R(T) = \sum_{t \in \tilde{T}} p(t)G(t)$$

  where $p(t)$ is the proportion of observations in node t, and $G(t)$ is defined by the misclassification error.

- $R(T)$ is biased downward, as

$$p(t)G(t) \geq p(t_L)G(t_L) + p(t_R)G(t_R)$$

  Thus the larger a tree is, the smaller its misclassification error (**overfitting!**)

Pruning

- Grow a very large tree T_{max}
  1. Until all terminal nodes are pure, containing only one class
  2. When the number of data in each terminal node is no greater than a certain threshold
  3. As long as the tree is sufficiently large

- A "selective" pruning procedure is needed.
  1. The pruning is optimal in a certain sense
  2. The search for different ways of pruning should be of manageable computational load

Minimal Cost Pruning

- For any subtree T of T_{max}, define its cost function as

$$R_\lambda(T) = R(T) + \lambda\left|\tilde{T}\right|$$

where λ is a tuning parameter.

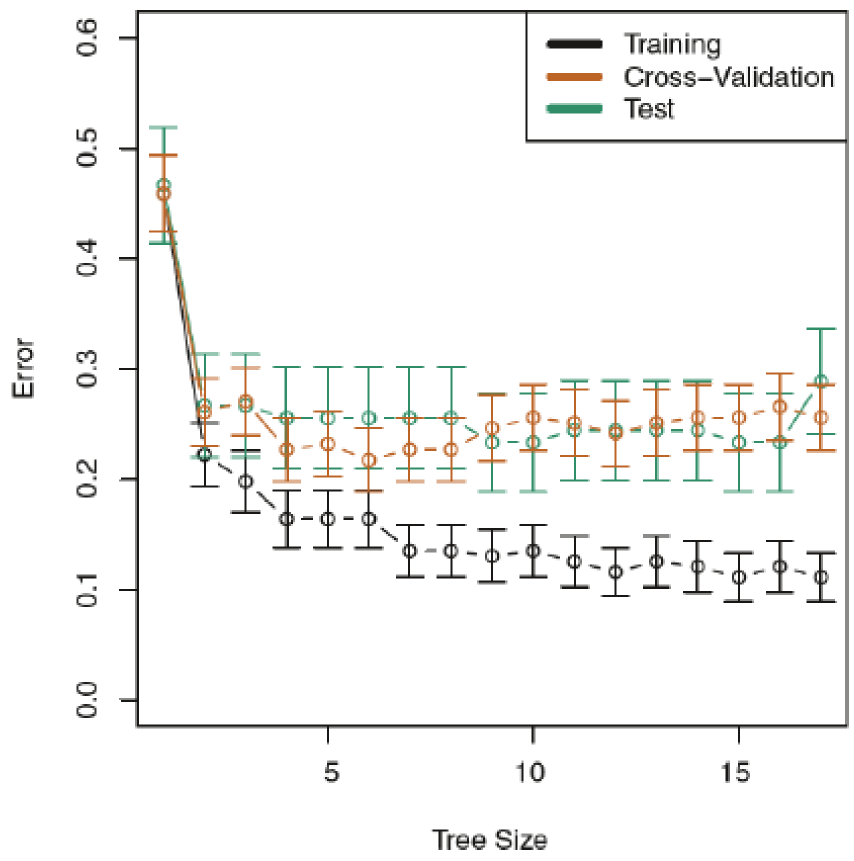
- For each λ, solve

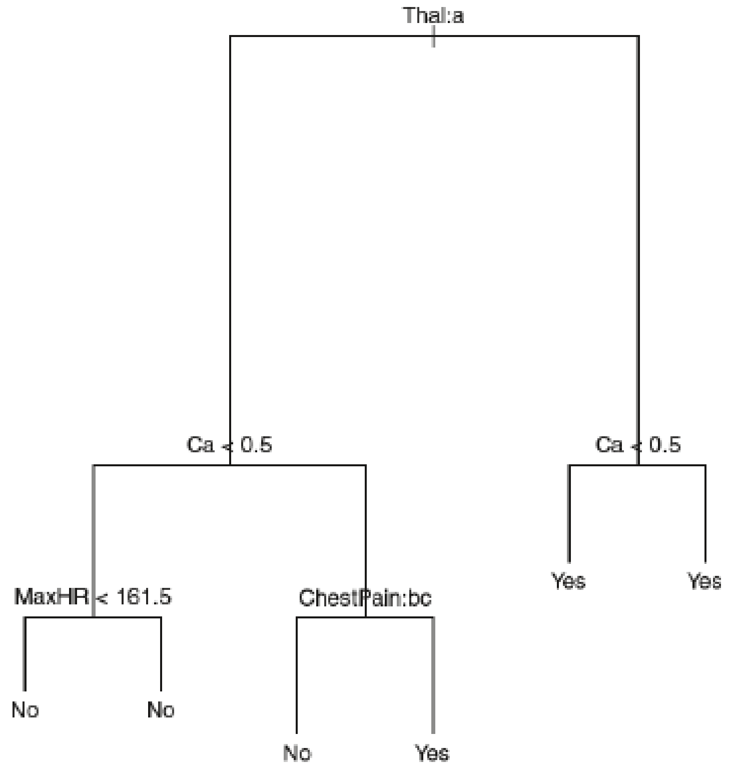
$$T(\lambda) = \underset{T}{\operatorname{argmin}}\, R_\lambda(T)$$

To search for the pruning branch, many techniques have been proposed, such as the weakest-link cutting.

- Optimal λ can be determined by cross validation.

Example of A Pruned Tree





Extension to Regression Tree

- S remains the same
- Impurity function: $G(t) = \sum_{i \in t} (y_i - \text{avg}(y|t))^2$
- Goodness of fit: $\phi(s, t) = G(t) - G(t_L) - G(t_R)$
- Stopping rule remains the same
- Response assignment rule:

$$\kappa(t) = \text{avg}(y|t)$$

Advantages of Tree-based Methods

- Easy to interpret
- Handle both categorical and ordered variables in a simple and natural way
- Automatic stepwise variable selection and complexity reduction
- Provide an estimate of conditional class probability
- Very nice and intuitive graphical display

Issues of Tree-based Methods

- Categorical variable with q classes produces $2^{q-1} - 1$ possible splits.
  - Simplification is possible for binary classification with Gini index or cross entropy, and regression with squared error loss.
  - But it is unclear for multiclass classification.
- Splits based on combined variables
- Lack of smoothness
- **High variance, constructed tree is very sensitive to sample**

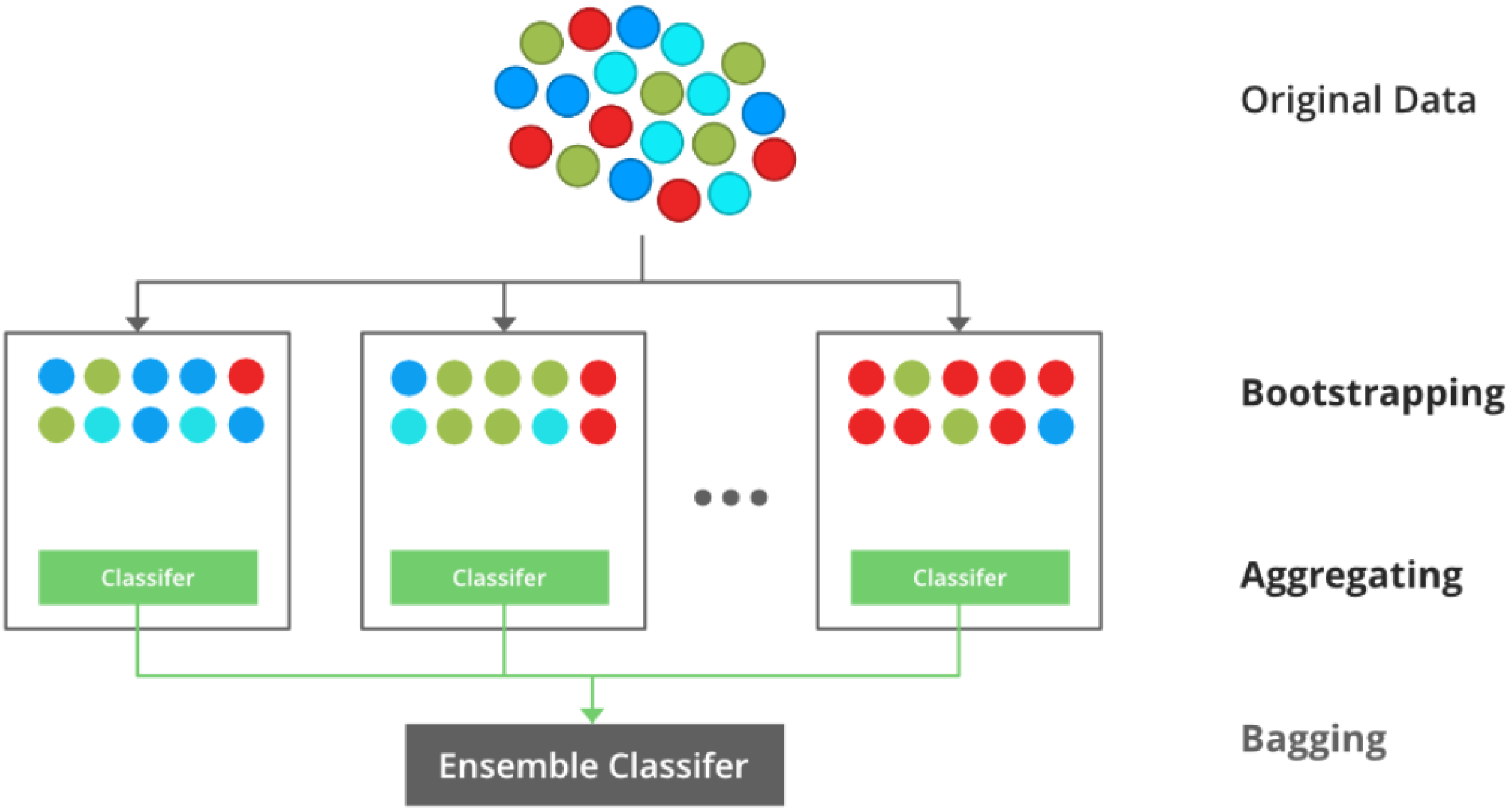
Bagging (Bootstrap Aggregation)

- **Bagging** is a technique to reduce the variance of an estimated prediction function.

1. Draw bootstrap samples $(x_1^{*b}, y_1^{*b}), \dots, (x_n^{*b}, y_n^{*b})$; $b = 1, \dots, B$
2. For each bootstrap sample, fit the tree model $\hat{f}^{*b}(x)$.
3. The bagging estimate is

$$\hat{f}_{bag}(x) = \begin{cases} \frac{1}{B}\sum_{b=1}^{B} \hat{f}^{*b}(x) & \text{for regression/classification} \\ mode\left\{\hat{f}^{*1}(x), \dots, \hat{f}^{*B}(x)\right\} & \text{for classification} \end{cases}$$

Idea of Bagging



Why It Works: Some Intuitions

- Pretend we draw bootstrap sample from the true distribution Γ (so **independent**)

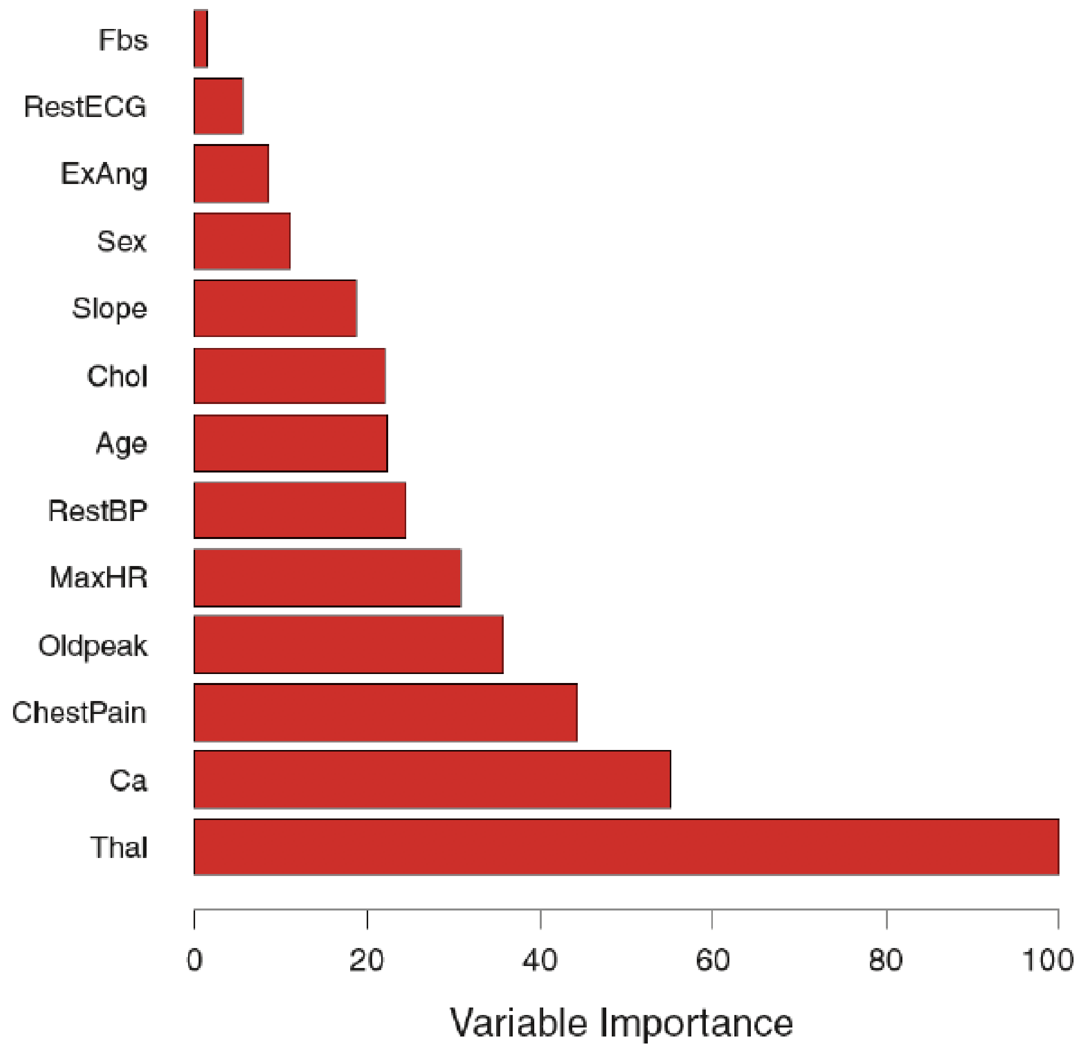
$$E\left(Y-\hat{f}(X)\right)^2 \geq E\left(Y-\hat{f}_{bag}(X)\right)^2$$

as $\hat{f}_{bag}(X) = E_{\Gamma}\left(\hat{f}(x)\right)$.

- Key statistical idea is **averaging reduces variance**.

- “Wisdom of crowds”: the collective knowledge of a diverse and independent body of people exceeds the knowledge of any single individual.

Variable Importance

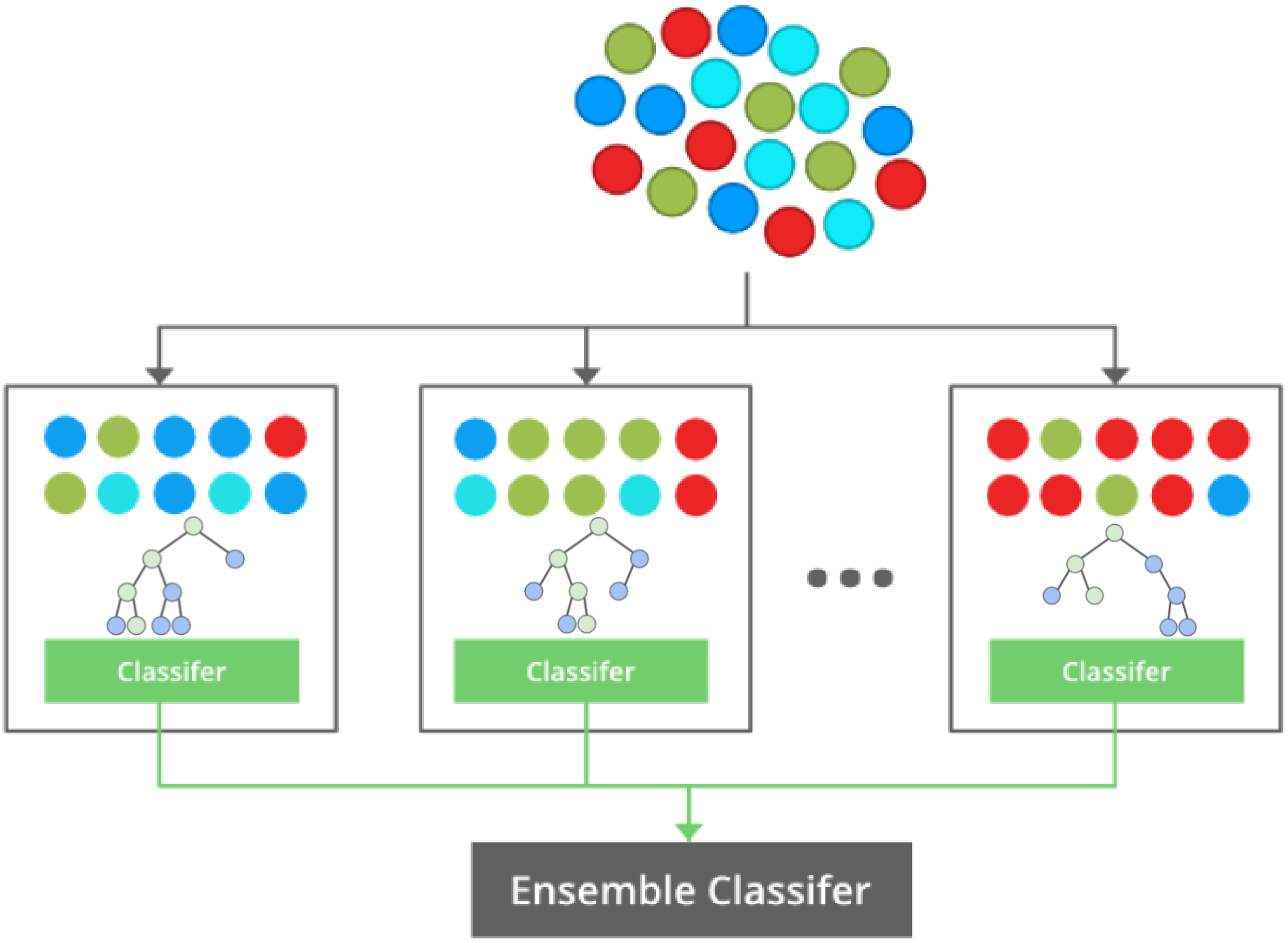


Random Forest

- Random forest is similar to bagging, but different as it averages de-correlated trees.

1. Draw bootstrap samples $(x_1^{*b}, y_1^{*b}), \dots, (x_n^{*b}, y_n^{*b})$; $b = 1, \dots, B$
2. For each bootstrap sample, grow a tree by repeating:
   a. Randomly select a subset of the p predictors ($\sqrt{p}$ for classification)
   b. Pick the best split among the chosen subset
   c. Split the node
3. The random forest estimate is constructed similarly as in bagging (average for regression, and majority vote for classification).

Idea of Random Forest



Why It Works: Some Intuition

- In bagging, the bootstrap trees are correlated, and correlation limits the benefit of averaging.

- Averaging B variables with variance σ^2 and correlation ρ yields variance

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

- In random forest, the correlation is reduced by selection of variables in growing trees.

- The price it has to pay is the increment of bias, which, fortunately, is usually small.

Boosting

- Boosting also involves growing multiple trees, but sequentially and adaptively.
- Each tree is grown using information from previously grown trees.
- Boosting does not involve bootstrap sampling: each tree is fit on a modified version of the original dataset.
- Iteratively learning weak classifiers
- Final result is the weighted sum of the results of each weak classifiers.

Idea behind Boosting

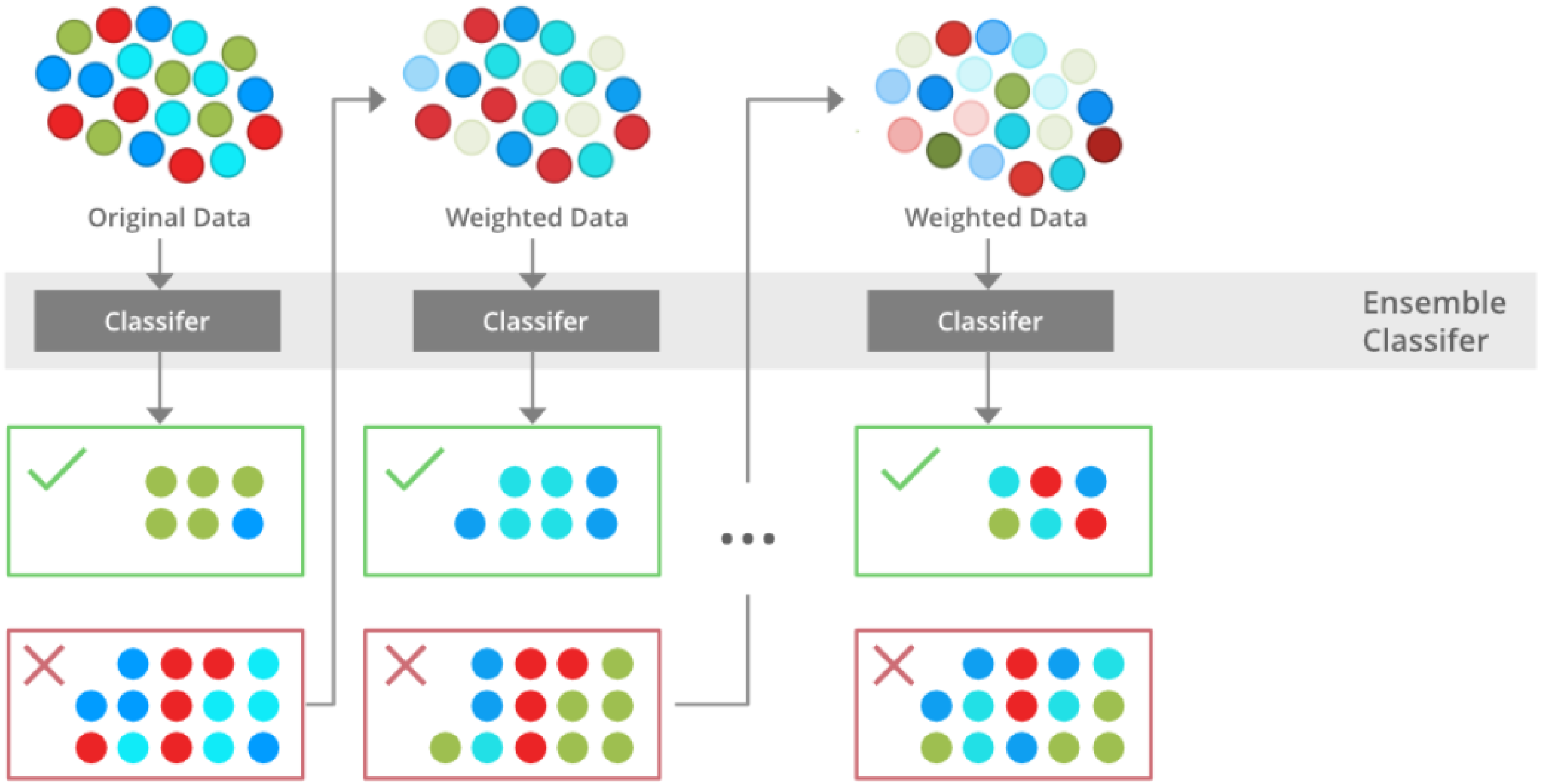
- Fitting a single large tree may cause overfitting.
- Boosting ***learns slowly***: in each step, fit a very small tree to the residuals, so that improve $\hat{f}$ in areas where it does not perform well.
- In general, statistical learning approaches that learn slowly tend to perform well.

Boosting for Classification

Algorithm:

1. Initialise the dataset and assign equal weight to each of the data point.
2. Provide this as input to the model and identify the wrongly classified data points.
3. Increase the weight of the wrongly classified data points and decrease the weights of correctly classified data points. And then normalize the weights of all data points.
4. Repeat to Step 2 for a few steps
5. Final output: weighted average of all models according to their performance.

Boosting for Classification



Boosting for Regression Tree

1. Initialize $f_0(\mathbf{x}) = \bar{y}$.

2. Compute $g_m(x_i) = y_i - f_{m-1}(\mathbf{x}_i) = r_i$.

3. Fit a regression tree h_m to the training data $(\mathbf{x}_i, r_i)$.

4. Update $f_m(\mathbf{x}) = f_{m-1}(\mathbf{x}) + \alpha h_m(\mathbf{x})$, and iterate

Remark: This boils down to the standard approach of iteratively fitting the residuals.

Boosting Algorithms

- There are many boosting algorithms: **adaboost** (adaptive boosting) is the first (1996).
- Examples of other boosting algorithms
  - LogitBoost (2000)
  - L2Boost (2002)
  - Gradient boosting (Gradient descent + Boosting)
- Focus on binary classification and may be extended to multiclass case.

Adaboost.M1 for Binary Classification Trees

1. Initialize $w_{1,i} = \frac{1}{n}; i = 1, \ldots, n$
2. For $m = 1$ to M:
   a. Fit a weak classifier $h_m(\mathbf{x}): R^p \rightarrow \{-1, 1\}$ to the training data with weights $w_{1,i}$.
   b. Compute weighted misclassification error:
   $$e_m = \sum_{i=1}^{n} w_{m,i}\, I(y_i \neq h_m(\mathbf{x}_i))$$
   c. Compute $\alpha_m = \frac{1}{2}\log((1 - e_m)/e_m)$.
   d. Update $w_{m+1,i} = w_{m,i} exp(-y_i \alpha_m h_m(\mathbf{x}_i))/Z_m$, where $Z_m = \sum_i w_{m,i} \exp(-y_i \alpha_m h_m(\mathbf{x}_i))$.
3. Output $f(x) = \frac{\sum_m \alpha_m h_m(\mathbf{x})}{\sum_m \alpha_m}$ and $G(\mathbf{x}) = \operatorname{sign}(f(\mathbf{x}))$.

Weak Classifier and Weights

- The key parameters in Adaboost.M1 are the weights $\{w_{m,i}\}$ which are adaptively updated in each iteration m:
  - The weight $w_{m,i}$ increases weights on misclassified points and decreases weights on correctly classified points.
- $\{\alpha_m\}$ sum up all the contributions from each $h_m(\mathrm{x}_i)$:
  - The weight α_m is larger for the weak classifier h_m with better performance (i.e., with smaller e_m).
- Why Adaboost takes this form?
  - It requires $\alpha_m > 0$ or equivalently $e_m < 1/2$, i.e., the weak classifier needs to be better than "random guessing".
- The weak classifiers are often set as shallow decision tree.

Recursive Idea

- The final classifier takes the additive form

$$f(x) = \sum_m \alpha_m h_m(\mathbf{x})$$

which can be written in a recursive form

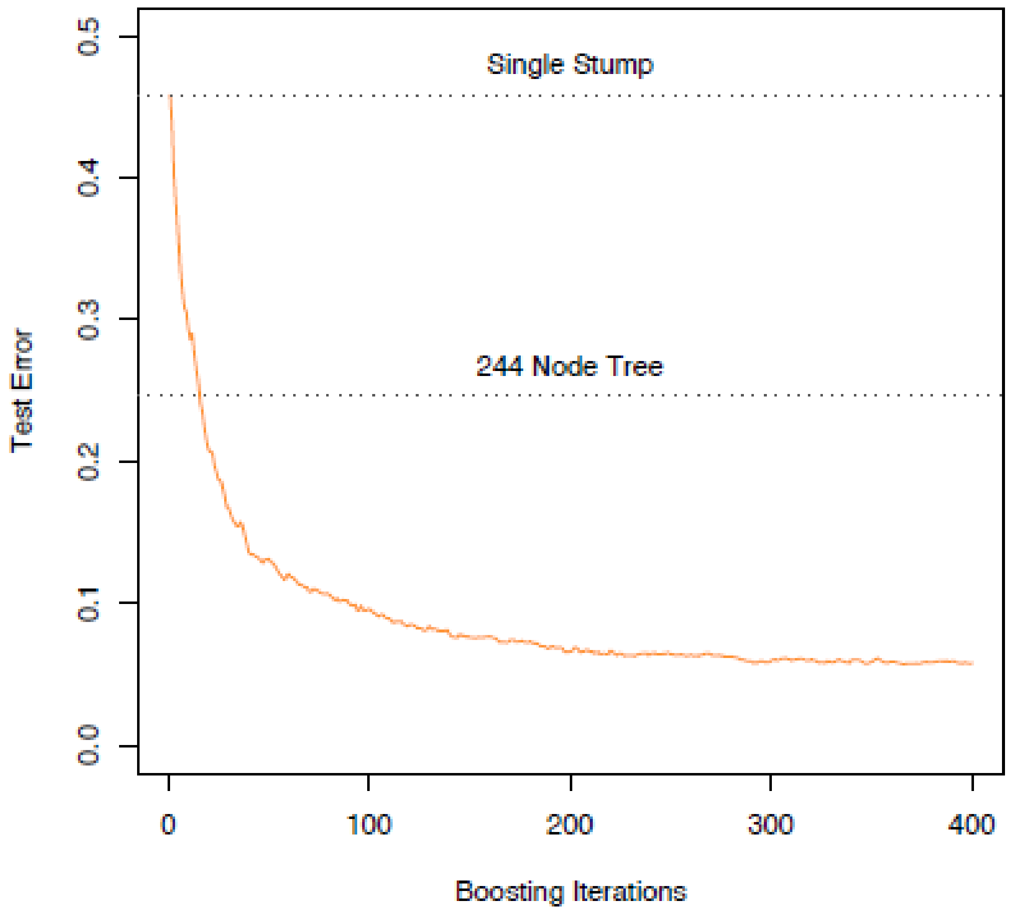
$$f_m(x) = f_{m-1}(x) + \alpha_m h_m(\mathbf{x}) \qquad m = 1,2, \dots, M$$

which is a (forward stagewise) additive model.

- Boosting is a way of fitting an additive expansion in a set of elementary “basis” functions.

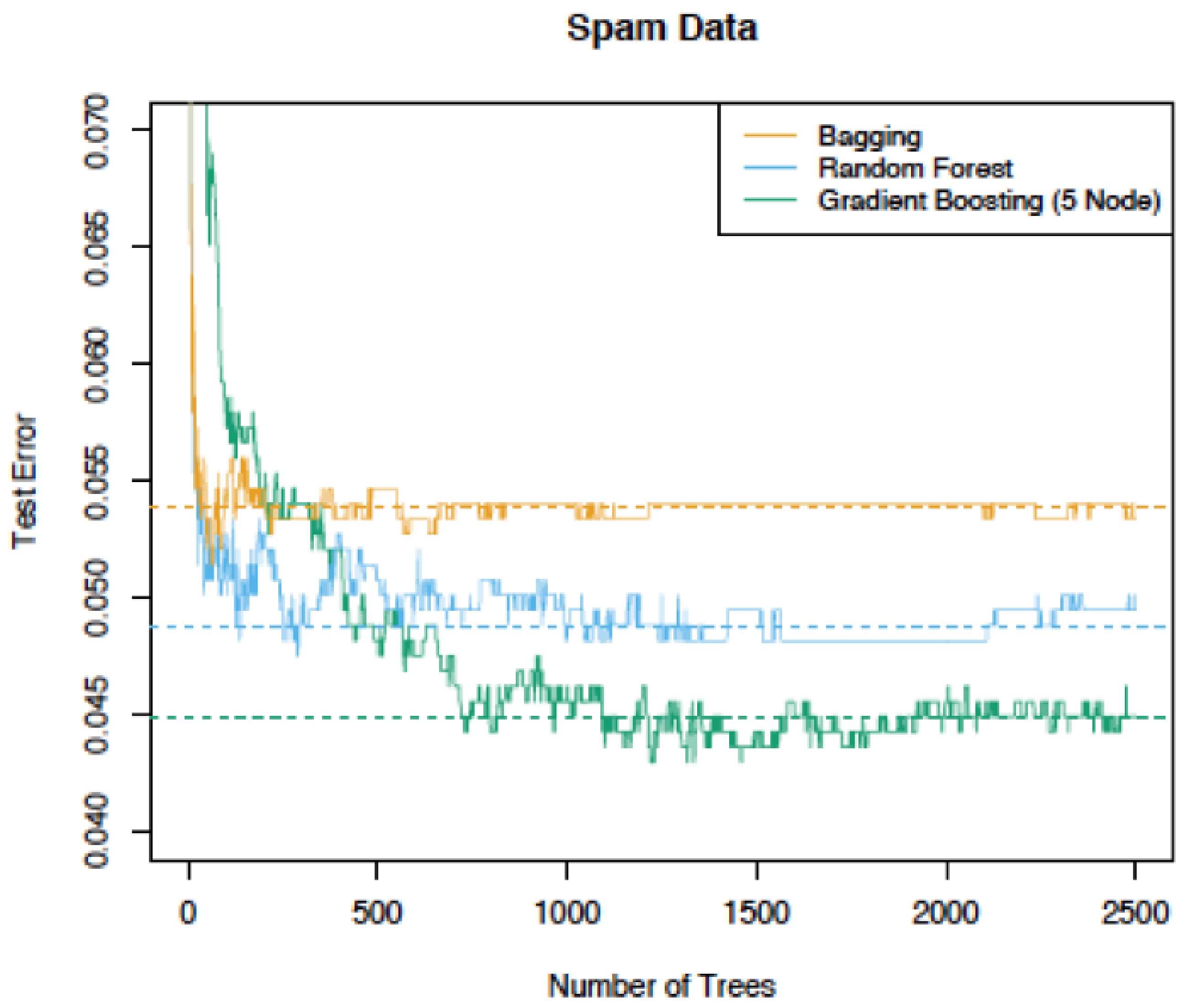
Performance of Adaboost

- A simulated example



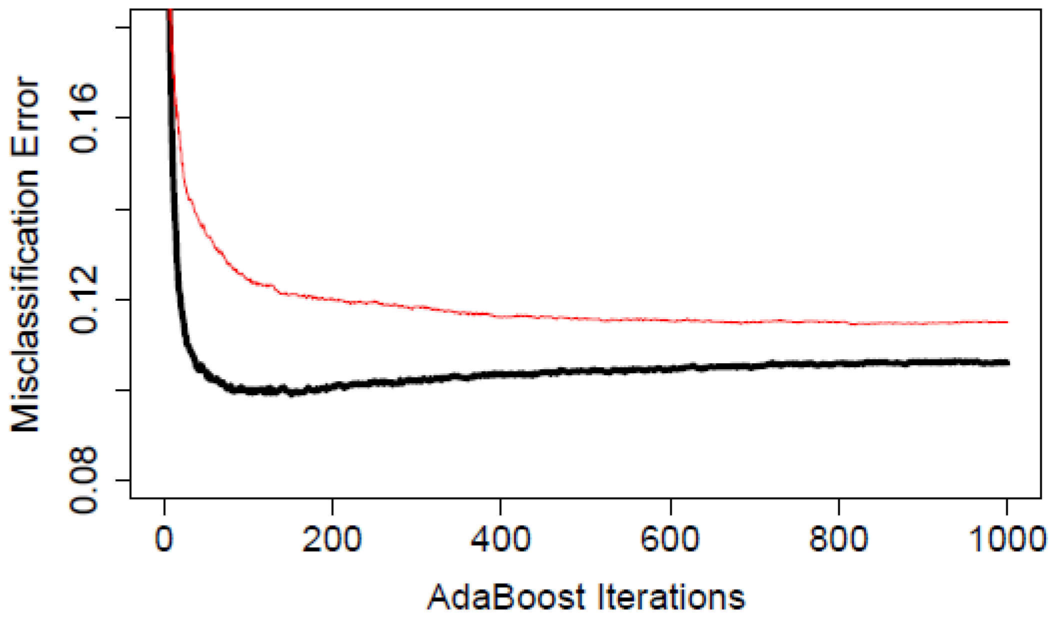
Stump: one-node tree

Spam Detection



Does Boosting Overfit?

- In practice, the test error does not rise much as the number of iterations m increases. Boosting algorithms are relatively immune to overfitting but it overfits slightly after a long time of iterations.



Red line: Adaboost with 8-node trees
Black line: Adaboost with stumps

Regularization

- Early stopping: stop the boosting algorithm at a medium number of iterations

- Shrinkage:

$$f_m(x) = f_{m-1}(x) + v \cdot \alpha_m h_m(\mathbf{x})$$

  where $0 < v < 1$ is a shrinkage factor.

- These two approaches operate in a similar fashion by controlling $\sum_m \alpha_m$: small value of M and small value of v result in small $\sum_m \alpha_m$ and large training error, and vice versa.

Summary of Tree Ensemble Methods

- **Bagging**: trees are grown independently on random samples of the observations. Consequently, the trees tend to be quite similar to each other. Thus, bagging can get caught in local optima and fail to thoroughly explore the model space.

- **Random forests**: trees are once again grown independently on random samples of the observations. However, each split on each tree is performed using a random subset of the features, thereby decorrelating the trees, and leading to a more thorough exploration of model space relative to bagging.

- **Boosting**: only the original data are used. The trees are grown successively, using a “slow” learning approach: each new tree is fit to the signal that is left over from the earlier trees, and shrunken down before it is used.