NAIVE BAYES CLASSIFIERS

Class prior

The distribution $P(y)$ is easy to estimate from training data:

$$P(y) = \frac{\#\text{observations in class } y}{\#\text{observations}}$$

Class-conditional distributions

The class conditionals $p(x|y)$ usually require a modeling assumption. Under a given model:

- Separate the training data into classes.
- Estimate $p(x|y)$ on class y by maximum likelihood.

TREE CLASSIFIERS

Idea

- Recall: Classifiers classify according to location in $\mathbb{R}^d$
- Linear classifiers: Divide space into two halfspaces
- What if we are less sophisticated and divide space only along axes? We could classify e.g. according to

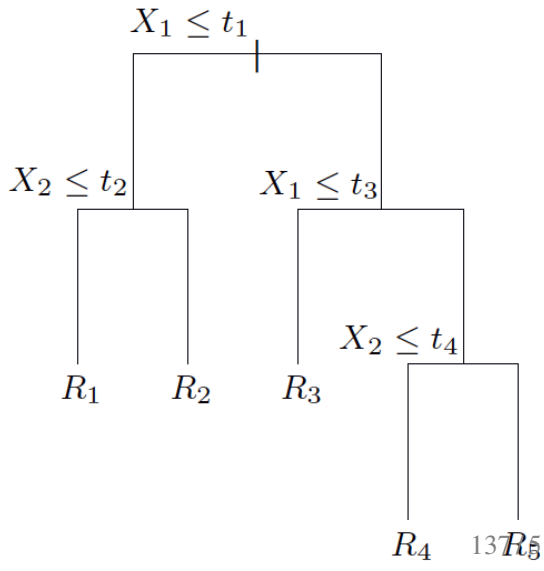
$$\mathbf{x} \in \begin{cases} \text{Class +} & \text{if } x_3 > 0.5 \\ \text{Class -} & \text{if } x_3 \leq 0.5 \end{cases}$$

- This decision would correspond to an affine hyperplane perpendicular to the x_3-axis, with offset 0.5.

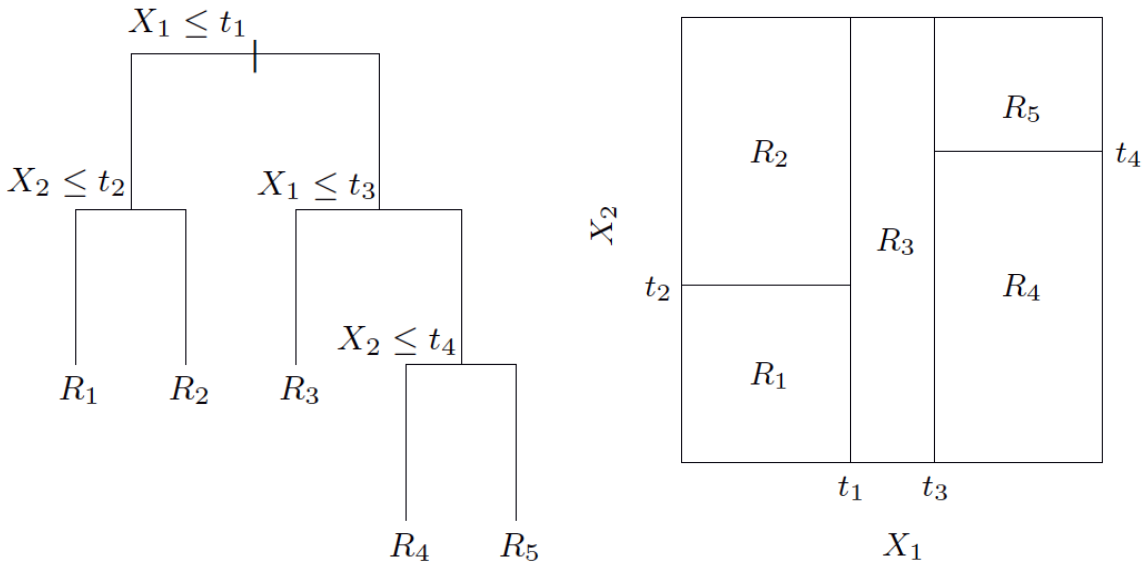
Tree classifier

A **tree classifier** is a binary tree in which

- Each inner node is a rule of the form $x_i > t_i$.
- The threshold values t_i are the parameters which specify the tree.
- Each leaf is a class label.



TREES



- Each leaf of the tree corresponds to a region R_m of $\mathbb{R}^d$.
- Classes $k \in \{1, \ldots, K\}$ (not restricted to two classes).
- Training: Each node is assigned class to which most points in R_m belong,

$$k(m) := \arg\max_k \#\{x_i \in R_m \text{ with } y_i = k\}$$

Overall algorithm

- At each step: Current tree leaves define regions $R_1, \ldots, R_M$.
- For each R_m, find the best split.
- Continue splitting regions until tree has depth D (input parameter).

Step of training algorithm

At each step: Current tree leaves define regions $R_1, \ldots, R_M$.

For each region R_m:

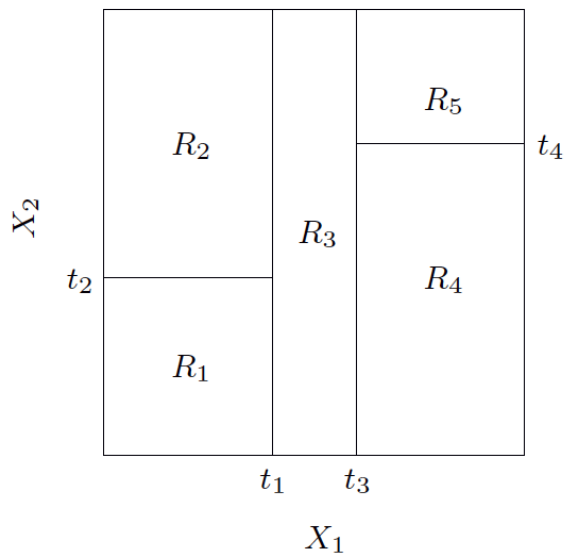
1. For each axis j: Compute best splitting point t_j as

$$t_j := \arg\min Q(R_m, t_j)$$

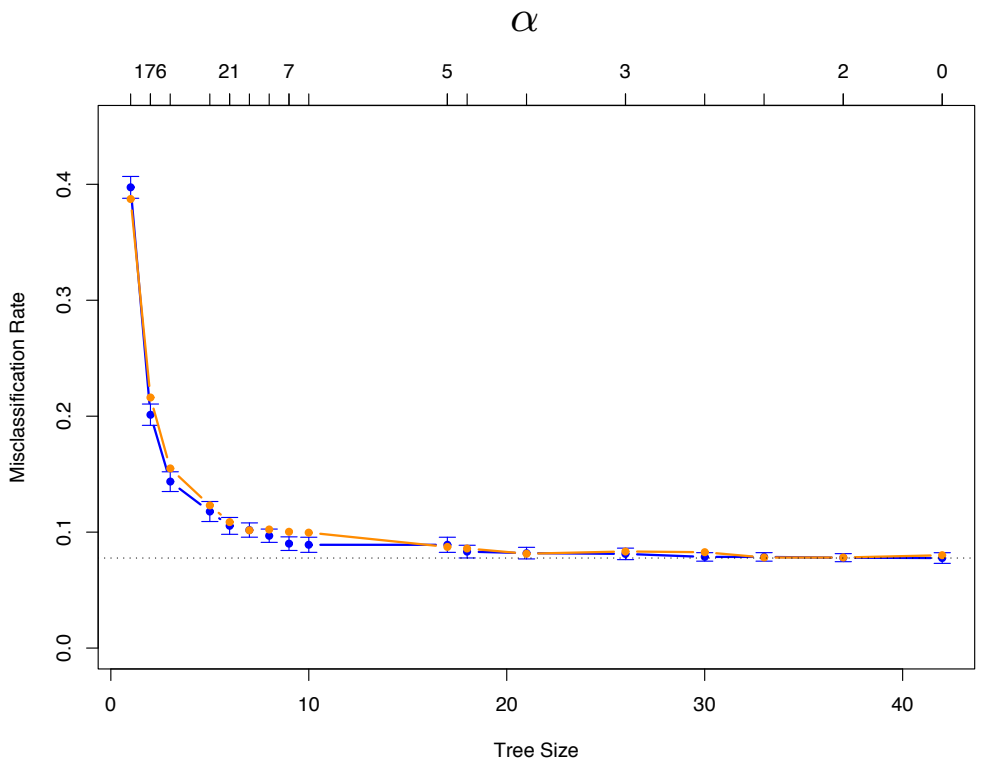
2. Select best splitting axis:

$$j := \arg\min_j Q(R_m, t_j)$$

3. Split R_m along axis j at t_j



Influence of Tree Size



Tree Size

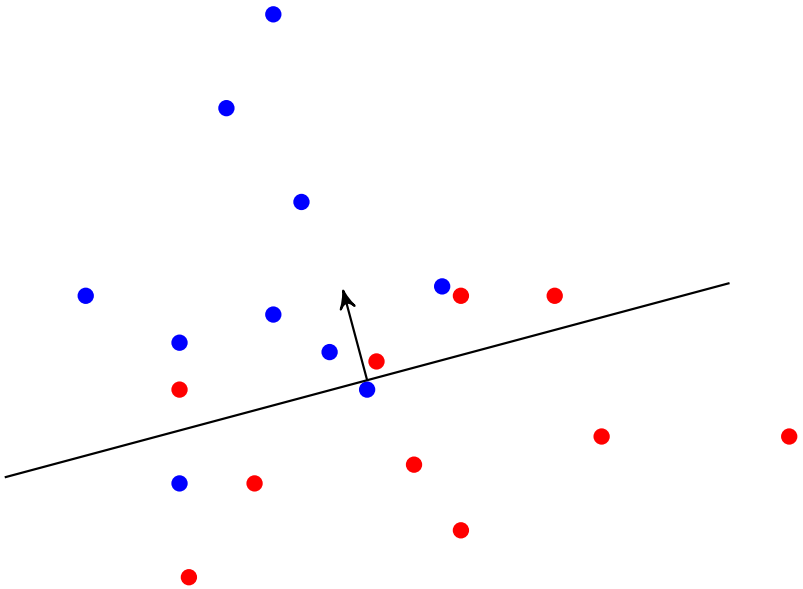
- Tree of height D defines 2^D regions.
- D too small: Insufficient accuracy. D too large: Overfitting.
- D can be determined by cross validation or more sophisticated methods ("complexity pruning" etc), which we will not discuss here.

Decision Stumps

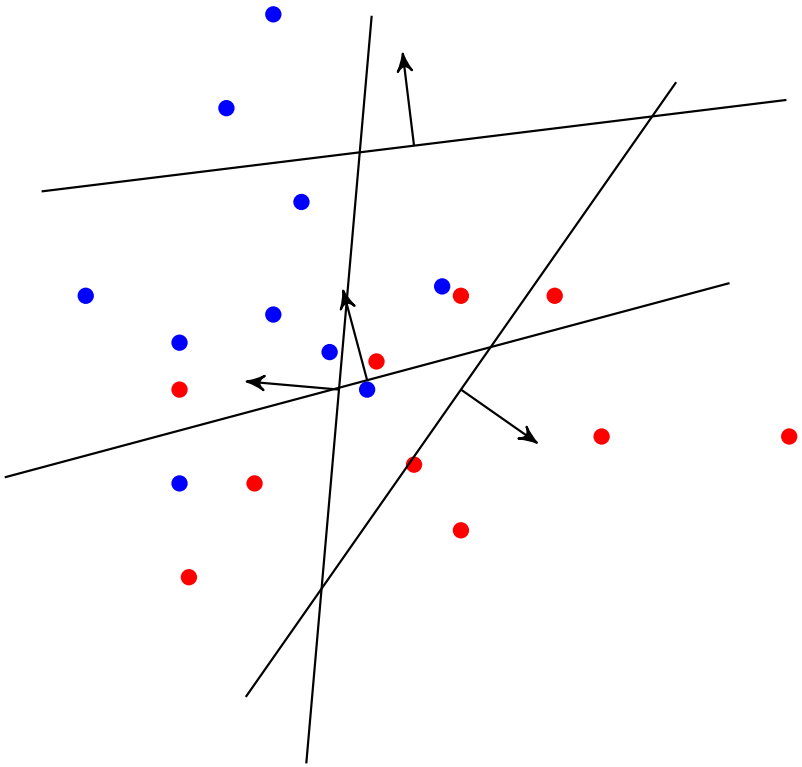
- The simplest possible tree classifier is a tree of depth 1. Such a classifier is called a **decision stump**.
- A decision stump is parameterized by a pair (j, t_j) of an axis j and a splitting point t_j.
- Splits $\mathbb{R}^d$ into two regions.
- Decision boundary is an affine hyperplane which is perpendicular to axis j and intersects the axis at t_j.
- Often used in Boosting algorithms and other ensemble methods.

Why are decision stumps useful?

A *randomly* chosen hyperplane classifier has an *expected* error of 0.5 (i.e. 50%).

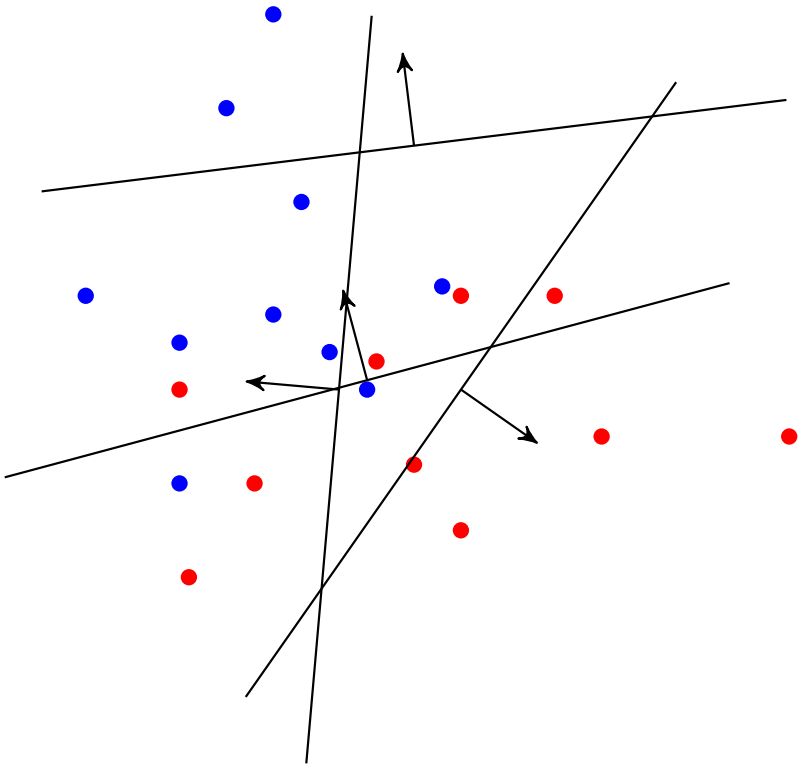


A *randomly* chosen hyperplane classifier has an *expected* error of 0.5 (i.e. 50%).



ENSEMBLES

A *randomly* chosen hyperplane classifier has an *expected* error of 0.5 (i.e. 50%).



- Many random hyperplanes combined by majority vote: Still 0.5.
- A single classifier slightly better than random: $0.5 + \varepsilon$.
- What if we use m such classifiers and take a majority vote?

Decision by majority vote

- m individuals (or classifiers) take a vote. m is an odd number.
- They decide between two choices; one is correct, one is wrong.
- After everyone has voted, a decision is made by simple majority.

Note: For two-class classifiers $f_1, \ldots, f_m$ (with output ± 1):

$$\text{majority vote} \;= \operatorname{sgn}\left(\sum_{j=1}^{m} f_j\right)$$

Assumptions

Before we discuss ensembles, we try to convince ourselves that voting can be beneficial. We make some simplifying assumptions:

- Each individual makes the right choice with probability $p \in [0, 1]$.
- The votes are *independent*, i.e. stochastically independent when regarded as random outcomes.

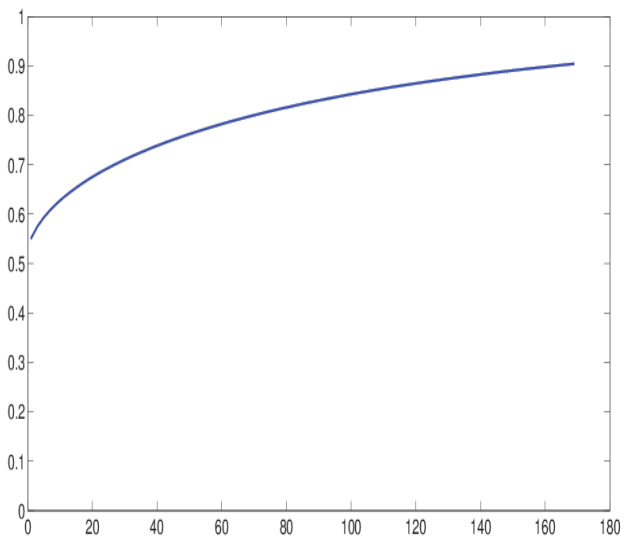
Condorcet's rule

If the individual votes are independent, the answer is

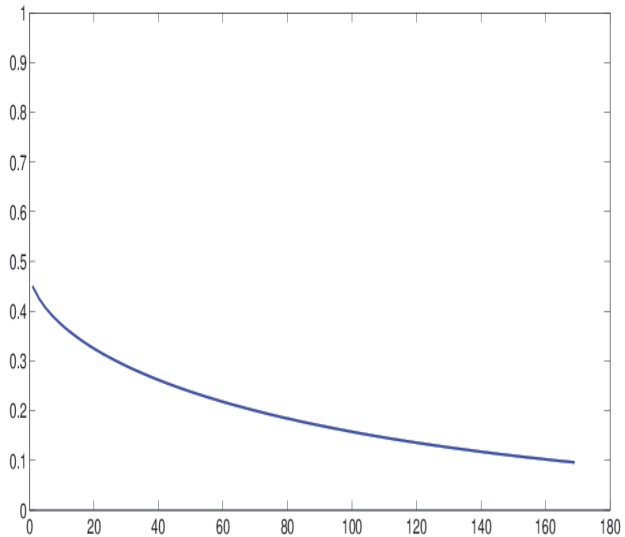
$$\Pr\{\text{ majority makes correct decision }\} = \sum_{j=\frac{m+1}{2}}^{m} \frac{m!}{j!(m-j)!} p^j (1-p)^{m-j}$$

This formula is known as **Condorcet's jury theorem**.

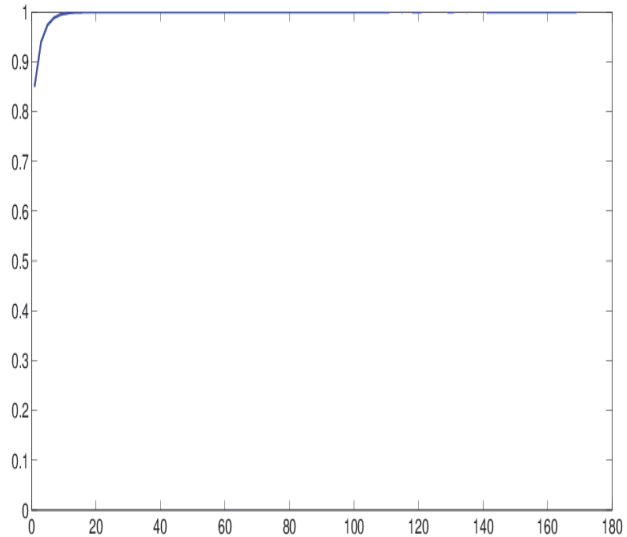
Probability as function of the number of votes



$p = 0.55$



$p = 0.45$



$p = 0.85$

Terminology

- An **ensemble method** makes a prediction by combining the predictions of many classifiers into a single vote.
- The individual classifiers are usually required to perform only slightly better than random. For two classes, this means slightly more than 50% of the data are classified correctly. Such a classifier is called a **weak learner**.

Strategy

- We have seen above that if the weak learners are random and independent, the prediction accuracy of the majority vote will increase with the number of weak learners.
- Since the weak learners all have to be trained on the training data, producing random, independent weak learners is difficult.
- Different ensemble methods (e.g. Boosting, Bagging, etc) use different strategies to train and combine weak learners that behave relatively independently.

Methods We Will Discuss

Boosting

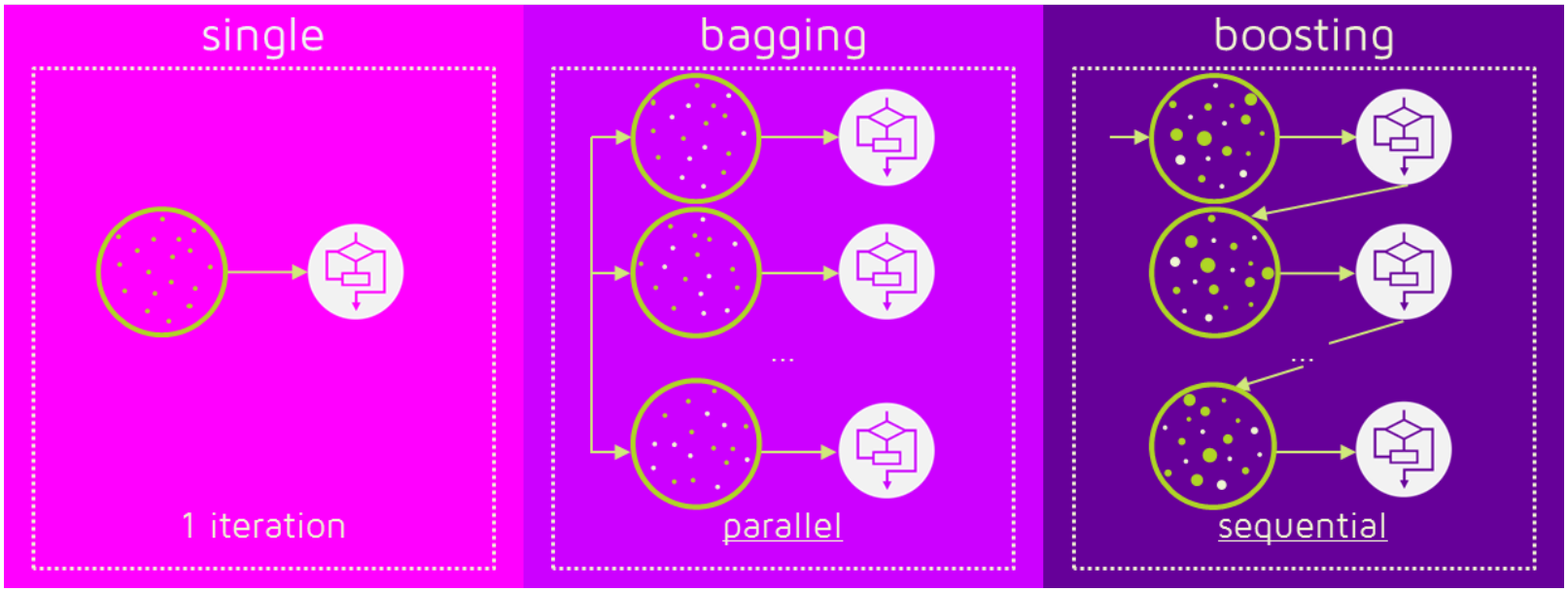
- After training each weak learner, data is modified using weights.
- Deterministic algorithm.

Bagging

Each weak learner is trained on a random subset of the data.

Random forests

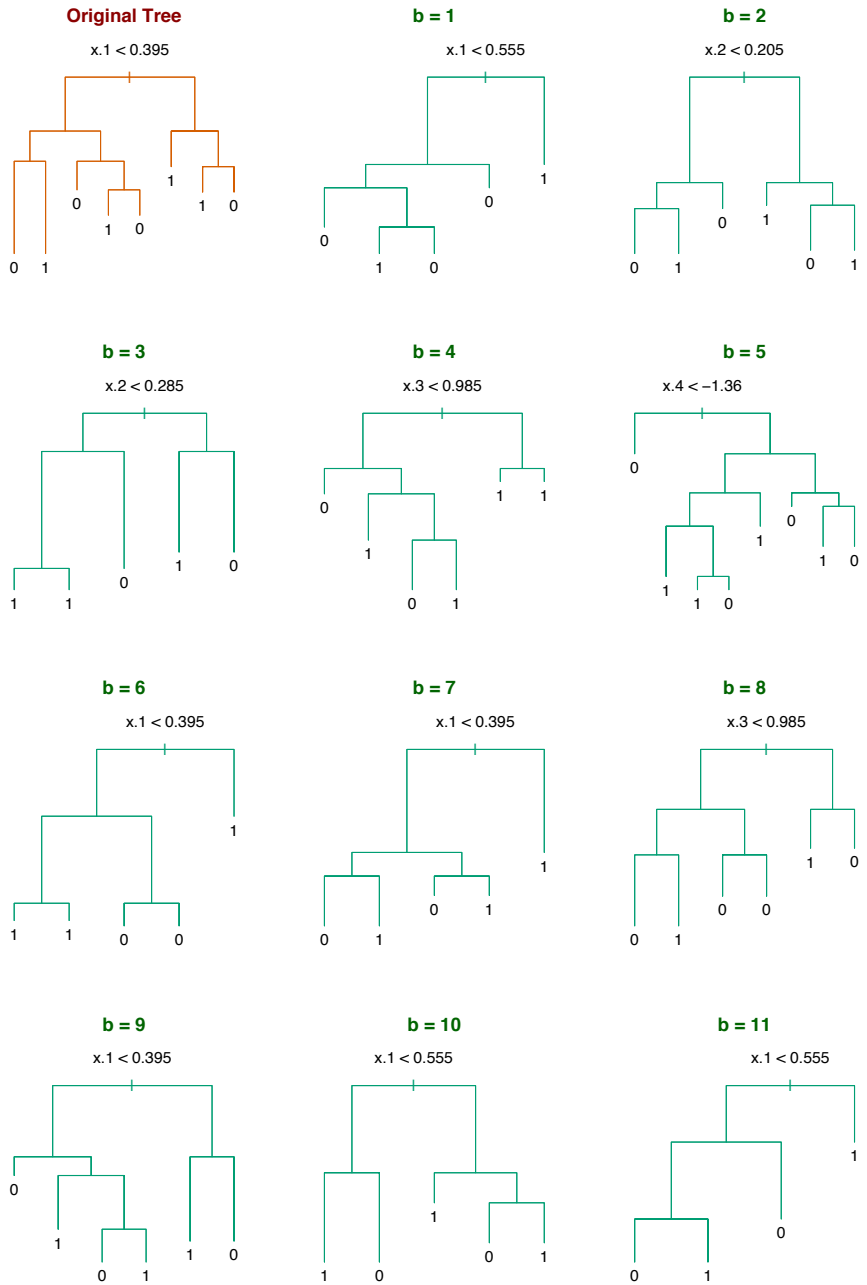
- Bagging with tree classifiers as weak learners.
- Uses an additional step to remove dimensions in $\mathbb{R}^d$ that carry little information.


single
1 iteration
bagging
...
parallel
boosting
...
sequential

Bagging and Random Forests

Example: Bagging Trees

- Two classes, each with Gaussian distribution in $\mathbb{R}^5$.
- Note the variance between bootstrapped trees.



Bagging vs. Boosting

- Bagging works particularly well for trees, since trees have high variance.
- Boosting typically outperforms bagging with trees.
- The main culprit is usually dependence: Boosting is better at reducing correlation between the trees than bagging is.

Random Forests

Modification of bagging with trees designed to further reduce correlation.

- Tree training optimizes each split over all dimensions.
- Random forests choose a different subset of dimensions *at each split*.
- Optimal split is chosen within the subset.
- The subset is chosen at random out of all dimensions $\{1, \ldots, d\}$.