



# Ensemble Methods, Random Forests and AdaBoost

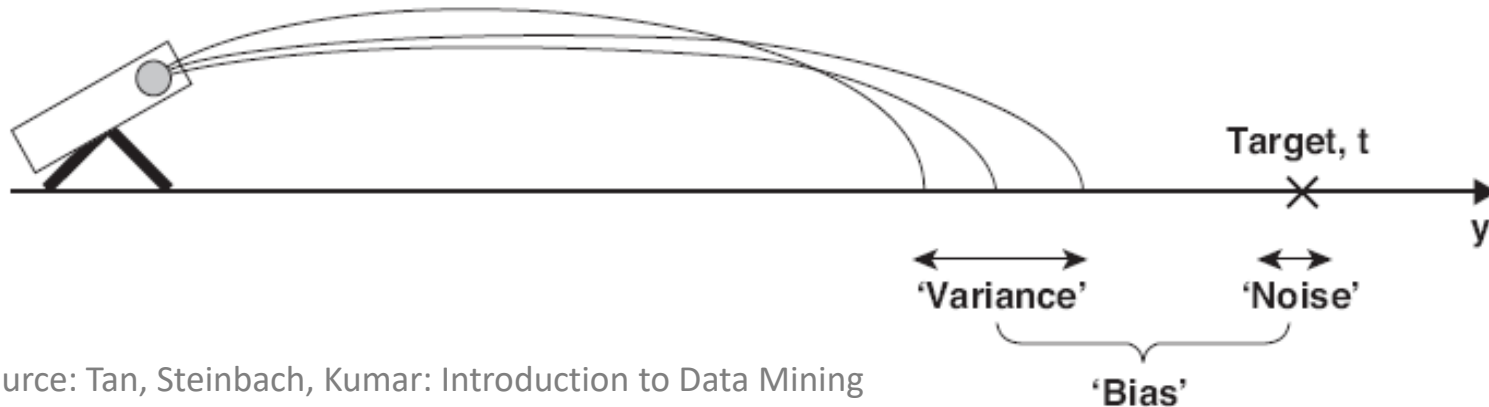
Prof. Dr. Stephan Trahasch  
Offenburg University of Applied Sciences

# Outline

---

- Ensemble Methods
- Random Forests
- AdaBoost
- Stacking
- Summary

# Variance, Bias and Noise



Variance, bias and noise are different components of the error

$$err = Bias_{\alpha} + Variance_f + Noise$$

In this example:

Bias: depending on firing angle  $\alpha$  (systematic error)

Variance: depending on the applied force  $f$  (data)

Noise: Fuzziness of the target (noise)

# Bias and Variance Decomposition

## Bias

the part of the error that is caused by the model

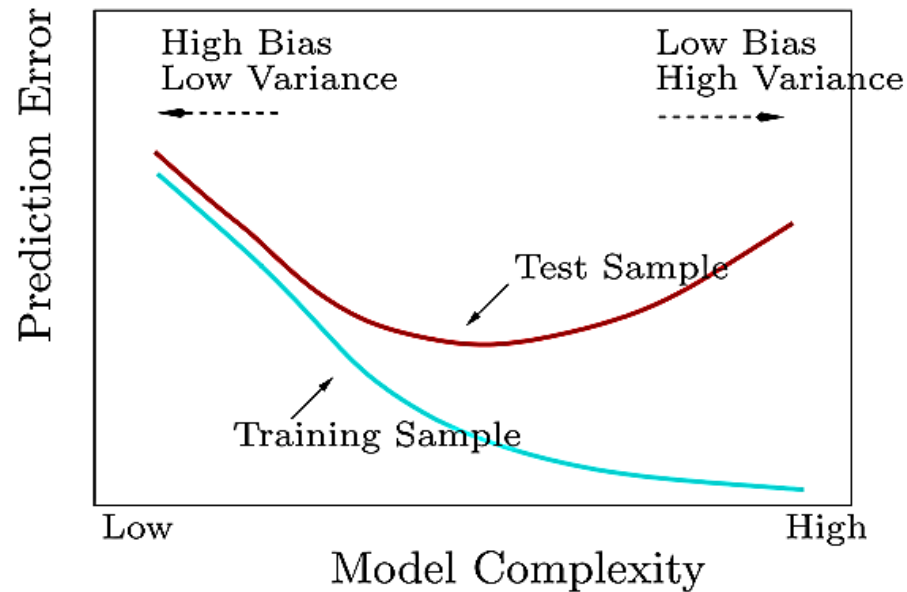
## Variance

the part of the error that is caused by the data sample

## Bias-Variance Trade-off

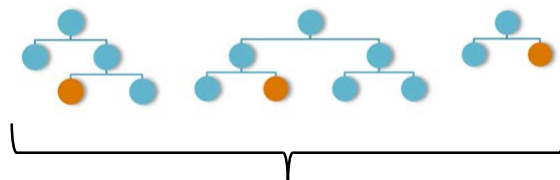
- Algorithms that can easily adapt to any given decision boundary are very sensitive to small variations in the data and vice versa
- Models with a low bias often have a high variance
  - e.g., nearest neighbor, unpruned decision trees
- Models with a low variance often have a high bias
  - e.g., decision stump, linear model

# Bias-Variance Trade-off



## Random Forest

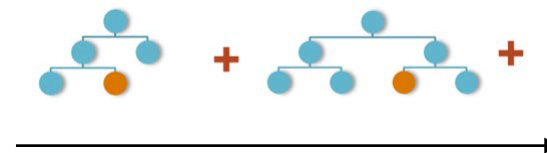
Variance ↓



Voting

## Boosting

Bias ↓



Weighted Sum

# Ensemble Classifiers

## Idea

Combine the predictions of multiple classifiers

## Objectives

- Reduce variance: results are less dependent on peculiarities of a single training set
- Reduce bias: a combination of multiple classifiers may learn a more expressive concept class than a single classifier

## Important

formation of an ensemble of diverse classifiers from a single training set

# Why does ensembles work?

Suppose there are 25 base classifiers

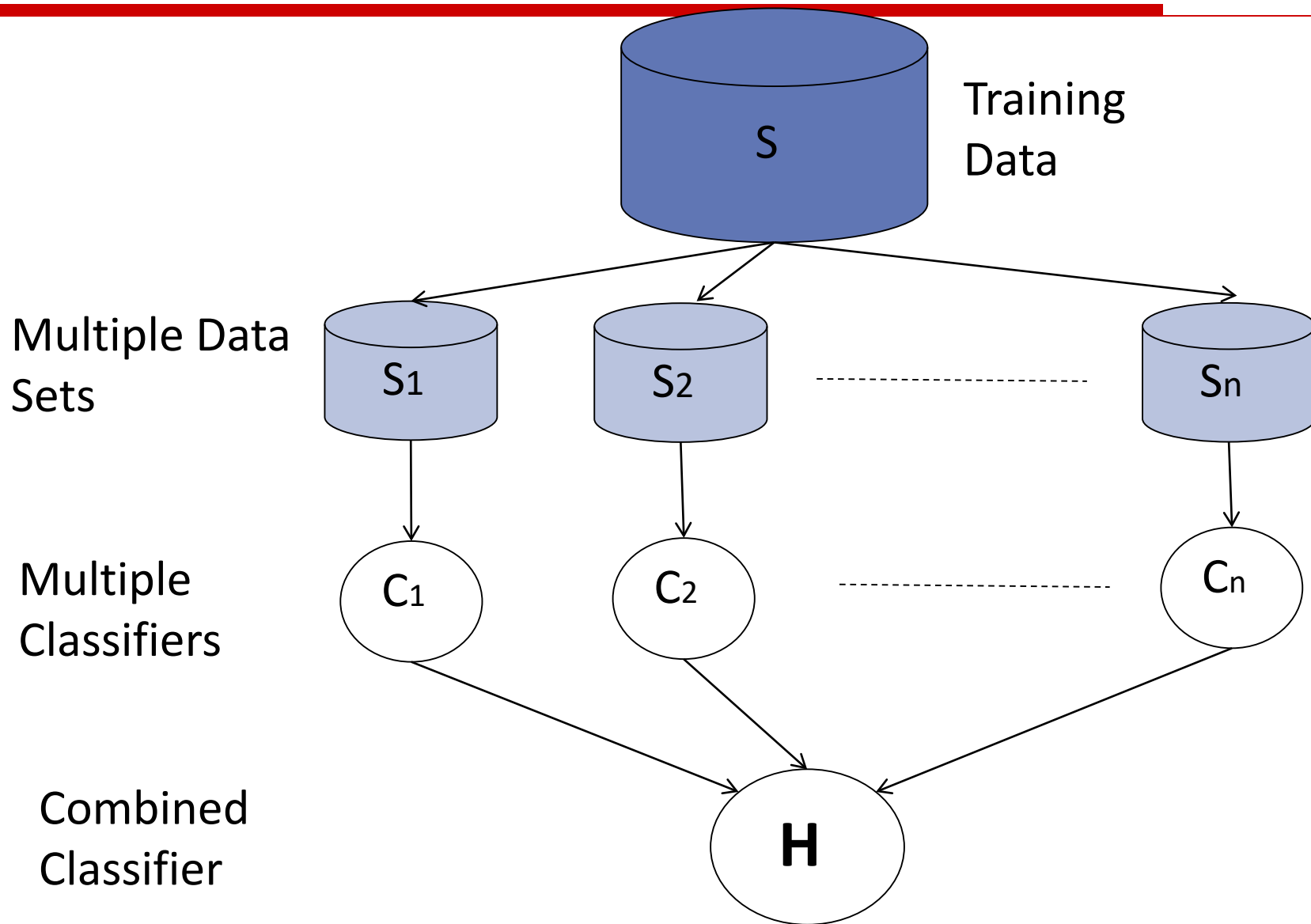
- Each classifier has error rate,  $\varepsilon = 0.35$
- Assume classifiers are independent  
i.e., probability that a classifier makes a mistake does not depend on whether other classifiers made a mistake

Probability that the ensemble classifier makes a wrong prediction

- The ensemble makes a wrong prediction if the majority of the classifiers makes a wrong prediction
- The probability that 13 or more classifiers err is

$$\sum_{i=13}^{25} \binom{25}{i} \varepsilon^i (1 - \varepsilon)^{25-i} \cong 0.06 < \varepsilon$$

# Basic Idea of Meta Learning





# Ensemble Methods

- Bootstrapping
- Bagging (Breiman 1994)
- Boosting (Schapire 1989)
- Adaboost (Schapire 1995)
- Random Forest (Breiman 2001)

Prediction of classes by combining several models that have been trained.

# Problem

- Biases in data samples may mislead classifiers
  - overfitting problem
  - model is overfit to single noise points
  
- If we had different samples
  - e.g., data sets collected at different times, in different places, ...
  - ...and trained a single model on each of those data sets...
  - only one model would overfit to each noise point
  - voting could help address these issues
  
- But usually, we only have one dataset!

# Bootstrap – also called 0.632 Bootstrap

Given a record with n instances (observations).

Bootstrap sample: draw a sample of the same length from the original dataset with replacement.

Original Data	1	2	3	4	5	6	7	8	9	10
Bootstrap 1	7	8	10	8	2	5	10	10	5	9
Bootstrap 2	1	4	9	1	2	3	2	7	3	2
Bootstrap 3	1	8	5	10	5	5	9	6	3	7
...										

For each set of size n, the probability that a given example appears

$$\text{in it is } \Pr(x \in B_i) = 1 - \left(1 - \frac{1}{n}\right)^n \xrightarrow{n \rightarrow \infty} 0.6322$$

On average, less than 2/3 of the examples appear in any single bootstrap sample.

# Estimate the error rate at Bootstrap

- Error estimation from the test data is very pessimistic
- Training was done on only ~63% of all instances
- To compensate this, we combine the test-set error rate with the resubstitution error on the instances in the training set.
- The error rate is offset against the resubstitution error:

$$\text{error} = 0.632 * \text{error}_{\text{test instances}} + 0.368 * \text{error}_{\text{training instances}}$$

- The resubstitution error (on the training data) is given less weight than the error on the test data.
- The process is repeated several times and the mean value of the error rates is calculated.
- Works well for very small datasets.

## Models may differ when learned on different data samples

- Idea of bagging:
  - create samples by picking examples with replacement
  - learn a model on each sample
  - combine models
- Samples
  - differ in the subset of examples
  - replacement randomly re-weights instances
- Uses usually the same base learner of  $B$  bootstrap versions of the original dataset and calculates mean value from the predictions of the models
- Reduces the variability of the prediction
- Works better with predictions that show small bias but have a higher variance. E. g. decision trees

# Bagging: Basic approach

```
1. for m = 1 to t          // t ... number of iterations
  a) draw with replacement a bootstrap sample  $D_m$  of data
  b) learn a classifier  $C_m$  from  $D_m$ 
2. for each test example
  a) try all classifiers  $C_m$ 
  b) predict the class that receives the highest number
     of votes
```

- variations are possible  
e.g., size of subset, sampling w/o replacement, etc.
- many related variants
  - sampling of features, not instances
  - learn a set of classifiers with different algorithms

# Variant of Bagging: Randomization

- Randomize the learning algorithm instead of the input data
- Some algorithms already have a random component  
e.g. initial weights in neural net
- Most algorithms can be randomized, e.g. greedy algorithms:
  - Pick from the N best options at random instead of always picking the best options
  - E.g.: test selection in decision trees or rule learning
- Can be combined with bagging

# Boosting

## ■ Idea of boosting

- train a set of classifiers, one after another
- later classifiers focus on examples that were misclassified by earlier classifiers
- weight the predictions of the classifiers with their error

## ■ Realization

- perform multiple iterations
- Each time using different example weights
- weight update between iterations
  - increase the weight of incorrectly classified examples
  - so they become more important in the next iterations  
(misclassification errors for these examples count more heavily)
- combine results of all iterations
  - weighted by their respective error measures



## Combines bagging and random attribute subset selection

- Build the tree from a bootstrap sample
- Instead of choosing the best split among all attributes, select the best split among a random subset of  $k$  attributes
- is equal to bagging when  $k$  equals the number of attributes
- There is a bias/variance tradeoff with  $k$ :
  - The smaller  $k$ , the greater the reduction of variance but also
  - the higher the increase of bias

# Outline

---

- Ensemble Methods
- Random Forests
- AdaBoost
- Stacking
- Summary

# Motivation

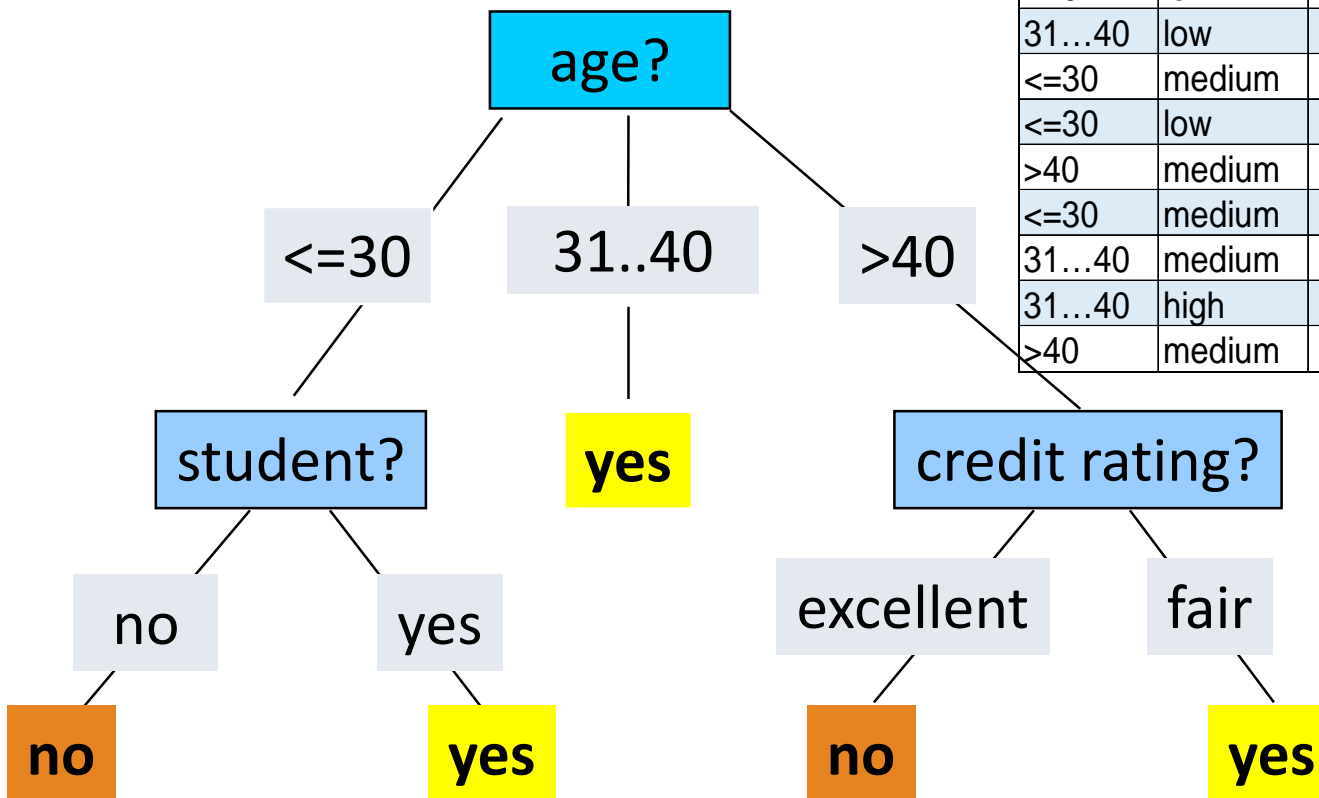


Fig. 3.15: Classification forests in Microsoft Kinect for Xbox

# Decision Tree Induction: An Example

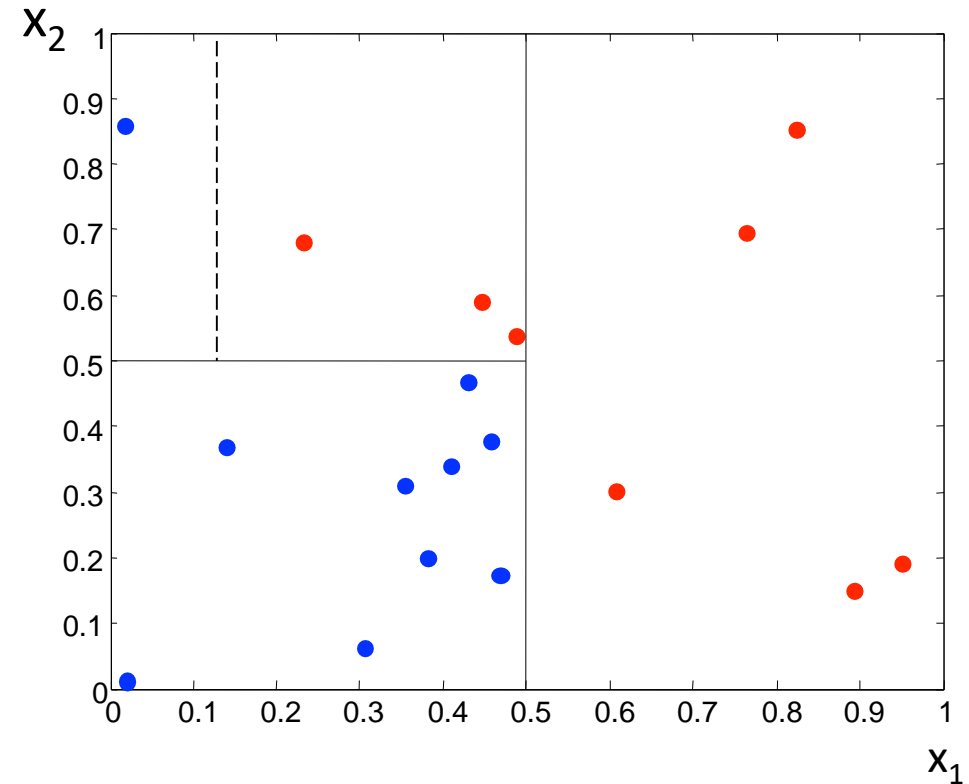
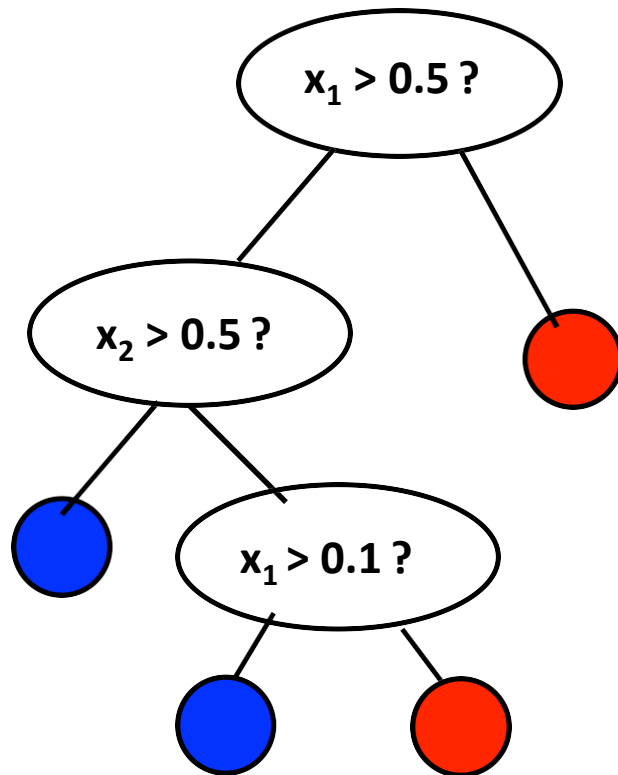
Training data set: Buys\_computer

Resulting tree:

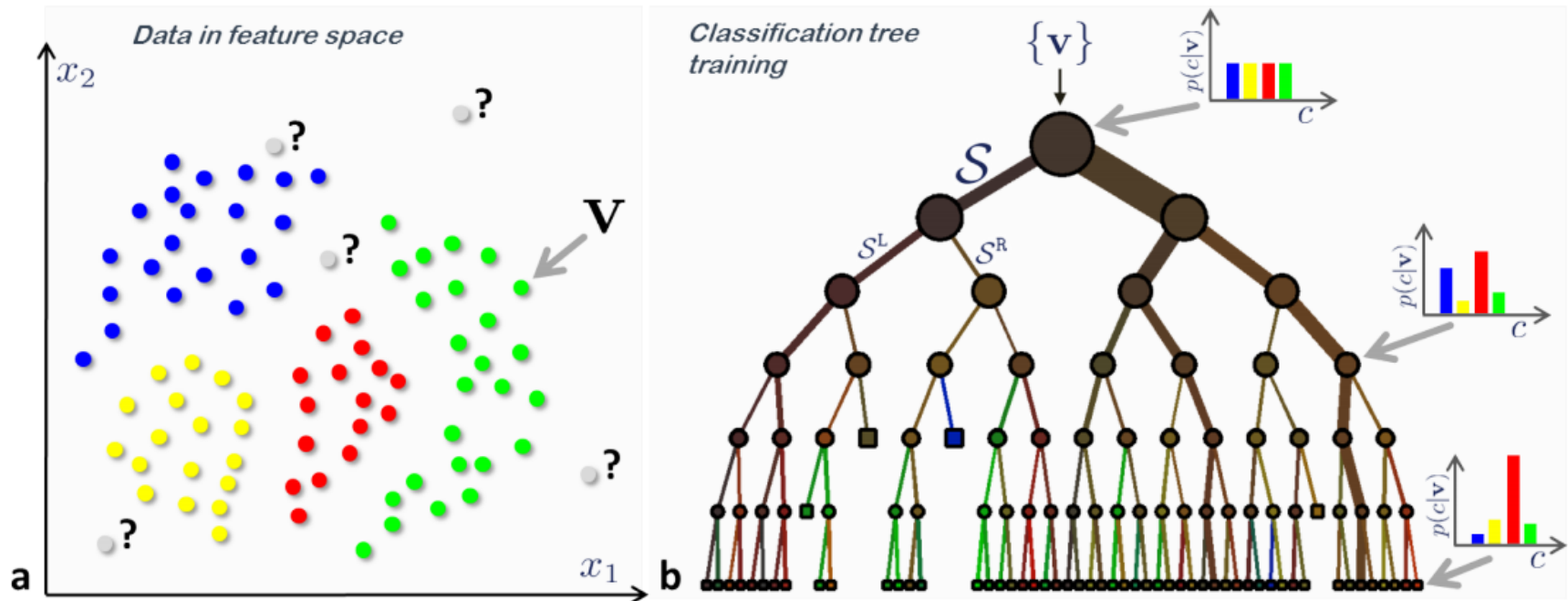


age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31...40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31...40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31...40	medium	no	excellent	yes
31...40	high	yes	fair	yes
>40	medium	no	excellent	no

# Decision Tree splits input data into cases



# Decision Tree splits input data into cases



Criminisi, A., Shotton, J., and Konukoglu, E., "Decision Forests for Classification, Regression, Density Estimation, Manifold Learning and Semi-Supervised Learning," MSR-TR-2011-114, 2011.

## ID3 Algorithm – Iterative Dichotomizer 3

**Input:** Example set  $S$

**Output:** Decision Tree  $DT$

If all examples in  $S$  belong to the same class  $c$   
return a new leaf and label it with  $c$

Else

- i. Select an attribute  $A$  according to some heuristic function
- ii. Generate a new node  $DT$  with  $A$  as test
- iii. For each Value  $v_i$  of  $A$ 
  - a) Let  $S_i$  = all examples in  $S$  with  $A = v_i$
  - b) Use ID3 to construct a decision tree  $DT_i$  for example set  $S_i$
  - c) Generate an edge that connects  $DT$  and  $DT_i$

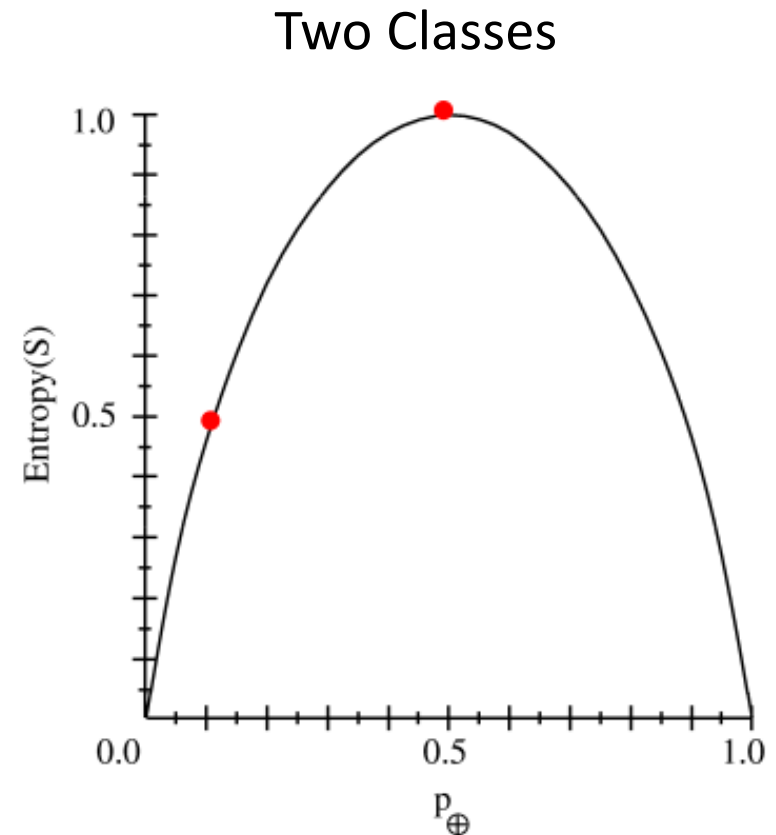
# Entropy

Let  $v_i$  a possible value of a particular attribute A and let  $p_j$  be the fraction of data points belonging to the class  $j \in \{1, \dots, k\}$  for attribute value  $v_i$

$$\text{Info}(v_i) = H(v_i) = -\sum_{j=1}^k p_j \log_2 p_j$$

Low values:

High values:





## Information Gain as Split Criteria (ID3/C4.5)

Let  $p_i$  be the probability that an arbitrary tuple in  $T$  belongs to class  $C_i$ , estimated by  $|C_{i,T}|/|T|$

$$\text{Info}(T) = H(T) = - \sum_{j=1}^k p_j \log_2 p_j$$

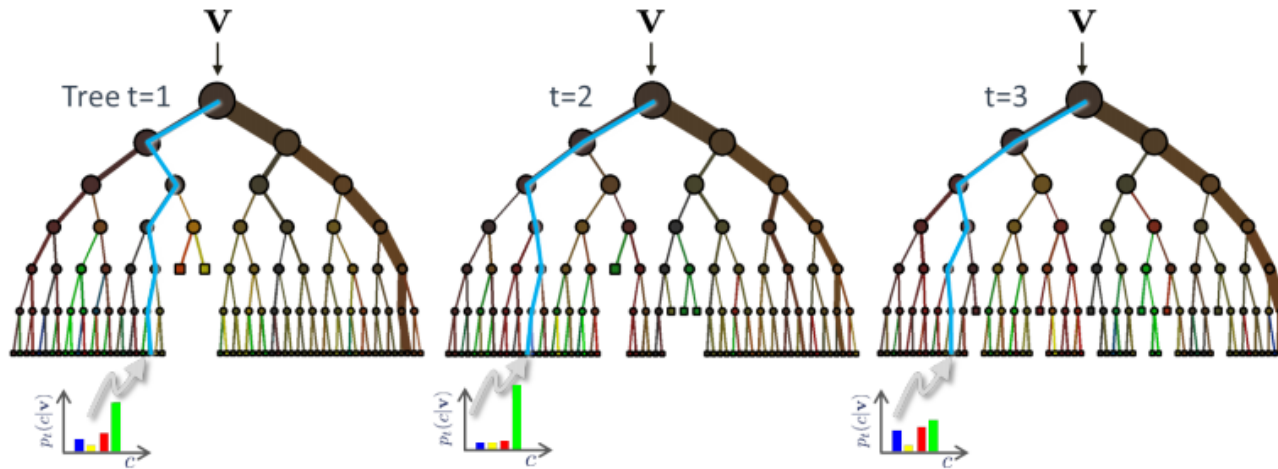
The overall entropy for a  $r$ -way split of set  $T$  into sets  $T_1, \dots, T_r$  by attribute  $A$  is computed as the weighted average:

$$\text{Info}(T, A) = \text{Entropy-Split}(T \Rightarrow T_1 \dots T_r) = \sum_{i=1}^r \frac{|T_i|}{|T|} \cdot H(T_i)$$

Information Gain is computed as difference of the entropy before split and after split.

$$\text{InfoGain}(T, A) = \text{Info}(T) - \text{Info}(T, A) = H(T) - \sum_{i=1}^r \frac{|T_i|}{|T|} \cdot H(T_i)$$

# Random Forest



- Random Forest consists of many "Trees", hence "Forest".
- Random "because the trees are randomly generated"
- It is not the whole dataset, but bootstrap learning samples are used
- At each node only random selection of attributes
- Assigning a value to each record based on the decision of each tree → majority

# Random Forest for Classification or Regression

1. For  $b = 1$  to  $B$ 
  - a) Draw a bootstrap sample  $\mathbf{Z}^*$  of size  $N$  from the training data.
  - b) Grow a random forest tree  $T_b$  to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size  $n_{min}$  is reached.
    - i. Select  $m$  variables at random from the  $p$  variables.
    - ii. Pick the best variable/split-point among the  $m$ .
    - iii. Split the node into two daughter nodes.
2. Output the ensemble of trees  $\{T_b\}_1^B$

## Classification:

Let  $\hat{C}_b(x)$  be the class prediction of the  $b$ th random-forest tree.

Then  $\hat{C}_{rf}^B(x) = \text{majority vote}\{\hat{C}_b(x)\}_1^B$

**Regression:**  $\hat{f}_{rf}^B(x) = \frac{1}{B} \sum_{b=1}^B T_b(x)$

# Why does Random Forest work?

$$\begin{aligned}
 \text{Var} \left( \frac{1}{B} \sum_{i=1}^B T_i(x) \right) &= \frac{1}{B^2} \sum_{i=1}^B \sum_{j=1}^B \text{Cov}(T_i(x), T_j(x)) \\
 &= \frac{1}{B^2} \sum_{i=1}^B \left( \sum_{j \neq i}^B \text{Cov}(T_i(x), T_j(x)) + \text{Var}(T_i(x)) \right) \\
 &= \frac{1}{B^2} \sum_{i=1}^B \left( (B-1)\sigma^2 \cdot \rho + \sigma^2 \right) \\
 &= \frac{B(B-1)\rho\sigma^2 + B\sigma^2}{B^2} \\
 &= \frac{(B-1)\rho\sigma^2}{B} + \frac{\sigma^2}{B} \\
 &= \rho\sigma^2 - \frac{\rho\sigma^2}{B} + \frac{\sigma^2}{B} \\
 &= \boxed{\rho\sigma^2} + \boxed{\sigma^2 \frac{1-\rho}{B}}
 \end{aligned}$$

Diagram annotations: A red box labeled  $i=j$  points to the  $\text{Var}(T_i(x))$  term in the second line. Red arrows point from the  $\text{Cov}(T_i(x), T_j(x))$  term in the second line to the  $(B-1)\sigma^2 \cdot \rho$  term in the third line, and from the  $\text{Var}(T_i(x))$  term to the  $\sigma^2$  term in the third line.

Hint:

$$\begin{aligned}
 \text{Var} \left( \sum_{i=1}^n X_i \right) &= \sum_{i,j=1}^n \text{Cov}(X_i, X_j) \\
 &= \sum_{i=1}^n \text{Var}(X_i) + \sum_{i,j=1, i \neq j}^n \text{Cov}(X_i, X_j) \\
 &= \sum_{i=1}^n \text{Var}(X_i) + 2 \sum_{i=1}^{n-1} \sum_{j=i+1}^n \text{Cov}(X_i, X_j).
 \end{aligned}$$

De-correlation  
results in higher  
accuracy!

Before each split, select  $m < p$   
of the attributes randomly as  
candidates for split.

### Spam Data

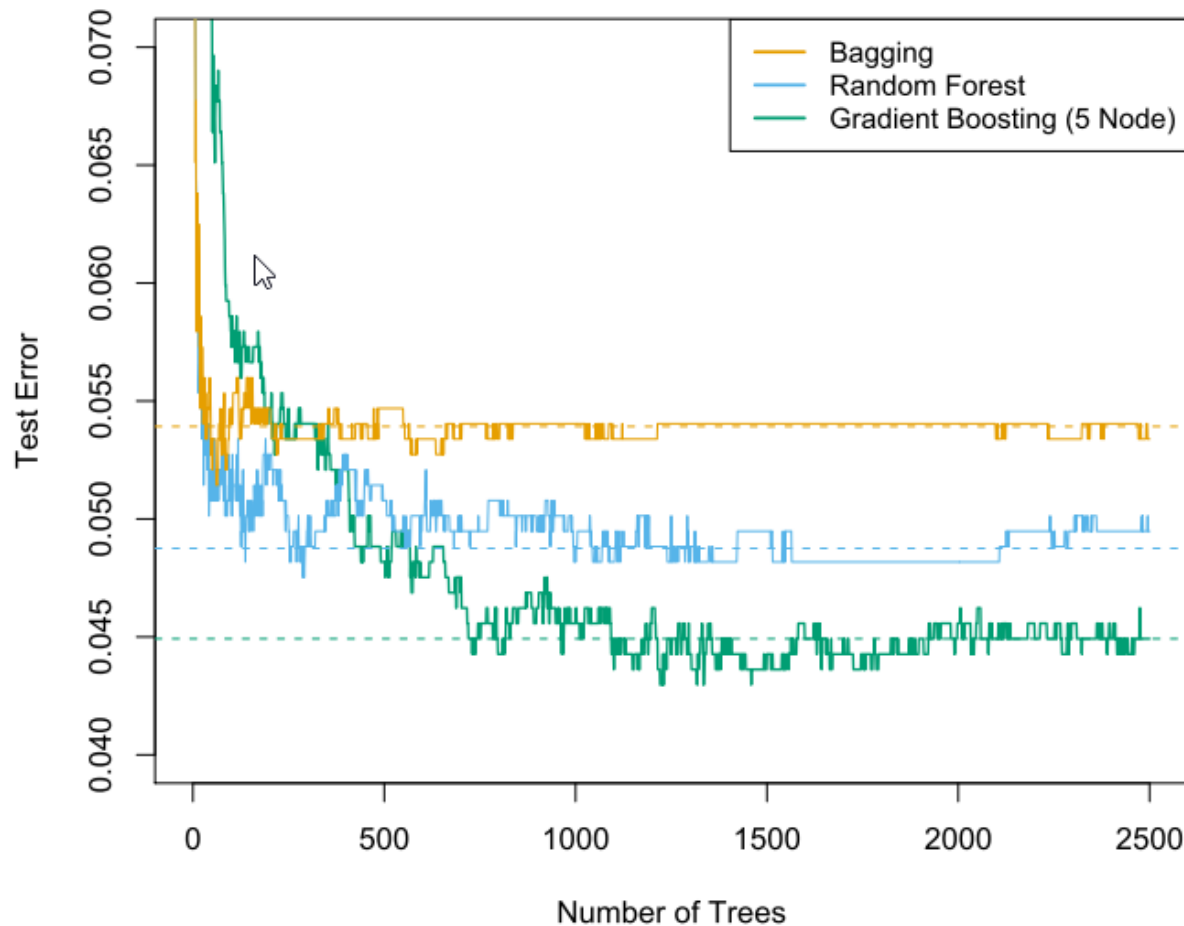


FIGURE 15.1. Bagging, random forest, and gradient boosting, applied to the spam data. For boosting, 5-node trees were used, and the number of trees were chosen by 10-fold cross-validation (2500 trees). Each “step” in the figure corresponds to a change in a single misclassification (in a test set of 1536).

Source: ESL\_II

## Bootstrap: Out-Of-Bag (OOB)

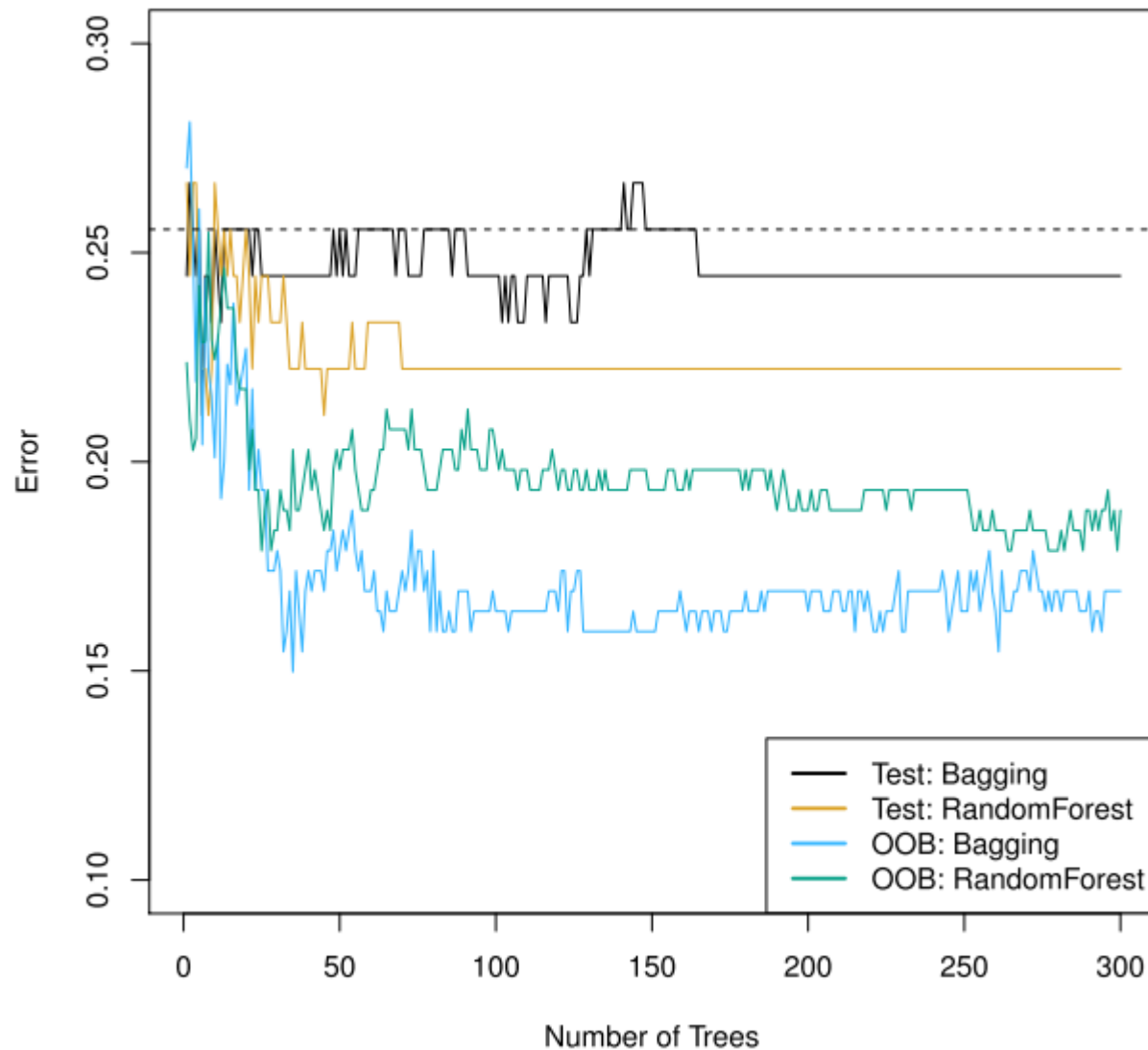
Observations that are not included in the bootstrap sample are called Out-Of-Bag (OOB).

Out-of-Bag error rate can be calculated for any observation over whole forest and corresponds to leave-one-out error rate.

Unlike most models, cross-validation is performed while Random Forest is being adapted.

→ no cross-validation necessary

# Bagging-Error and Out-Of-Bag-Error

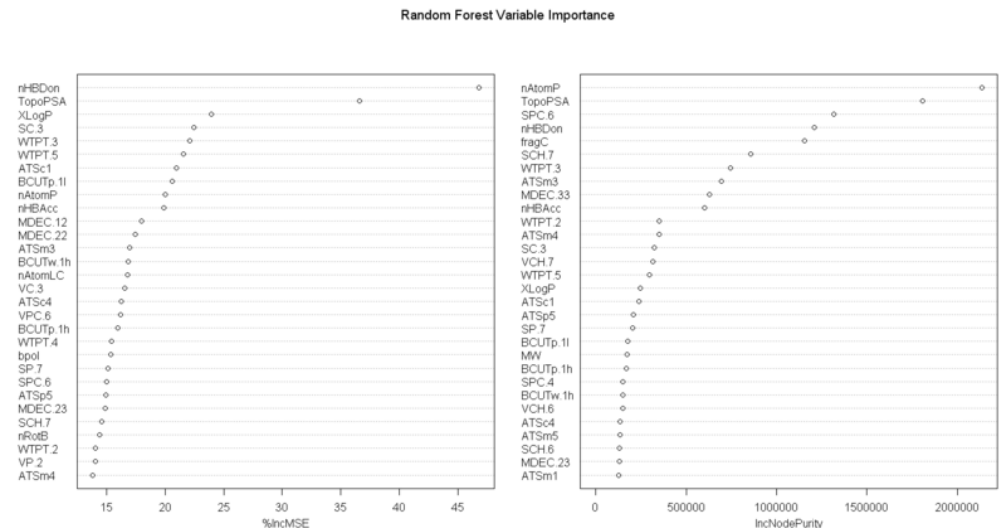
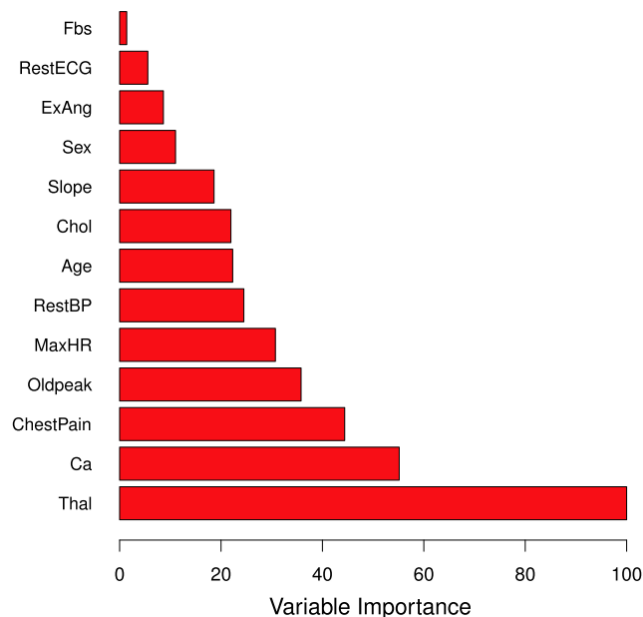


# Ranking of features

For each split, you can calculate the improvement in the split criterion for split characteristics.

The influence of a characteristic is calculated as the sum of the improvements in the split criterion over the whole forest

→ feature ranking





# Advantages of Random Forest

- Suitable for high-dimensional data where the number of attributes exceeds the number of objects observed
- yet robust against overfitting
- Quantifies the importance of individual variables
- Good performance compared to classics such as support vector machines and neural networks
- Straightforward estimation of the error rate
- Consistent estimation of target values
- Problem with a single tree:  
has high variance → wrong decision in "high" nodes runs through subsequent nodes → many trees reduce this effect

# Differences to an standard Decision Tree

- Each tree is trained with a “bootstrap resample of data”.  
“Bootstrap resample of data set with N samples: Make new data set by drawing with replacement N samples; i.e., some samples will probably occur multiple times in new data set”
- For each split, randomly select m attributes from the total p attributes ( $m = \sqrt{p}$ )
- No Pruning

# Outline

---

- Ensemble Methods
- Random Forests
- AdaBoost
- Stacking
- Summary

# Boosting

## ■ Ensembles

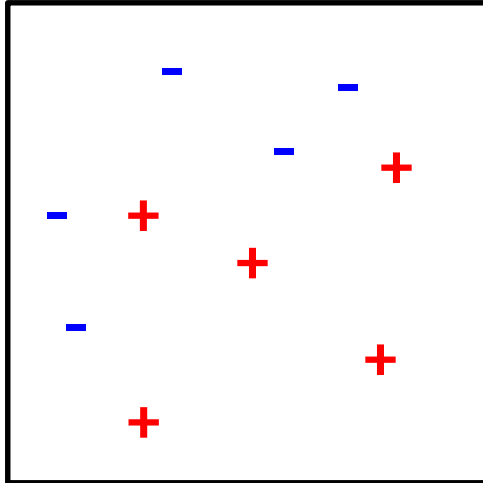
- Weighted combinations of classifiers
- “Committee” decisions
- Equal weights (majority vote) in a simple case
- Might want to weight unevenly – up-weight good experts

## ■ **Boosting**

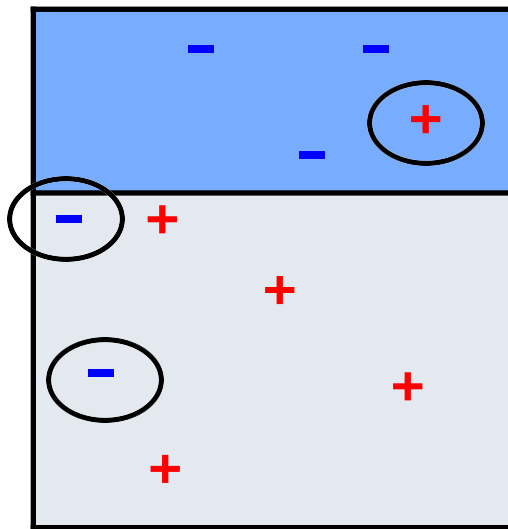
- Focus new experts on examples that others get wrong
- Train experts sequentially
- Errors of early experts indicate the “hard” examples
- Focus later classifiers on getting these examples right
- Combine the whole set in the end
- Convert many “weak” learners into a complex classifier

# Example: Classes +1 , -1

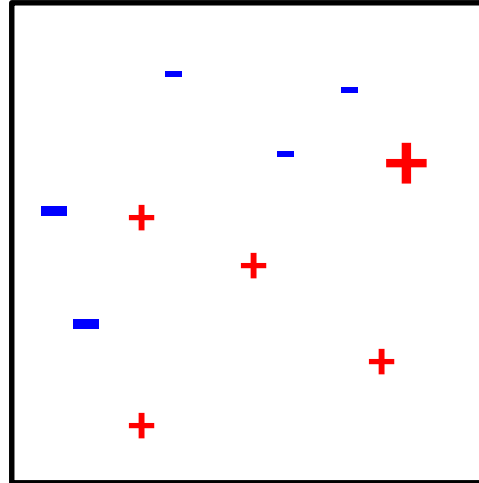
Original data set,  $D_1$



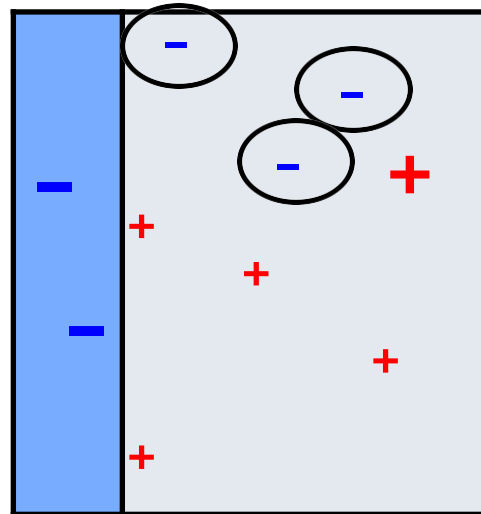
Trained classifier



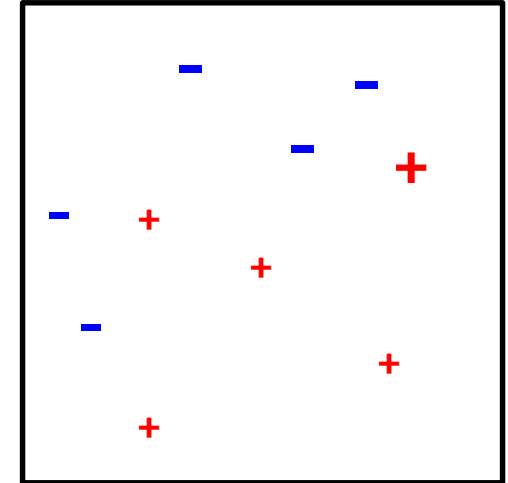
Update weights,  $D_2$



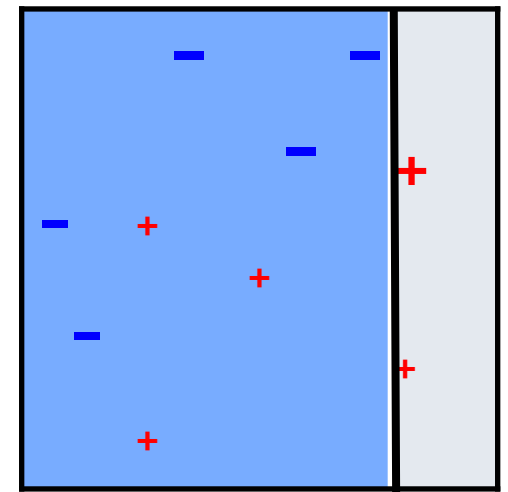
Trained classifier

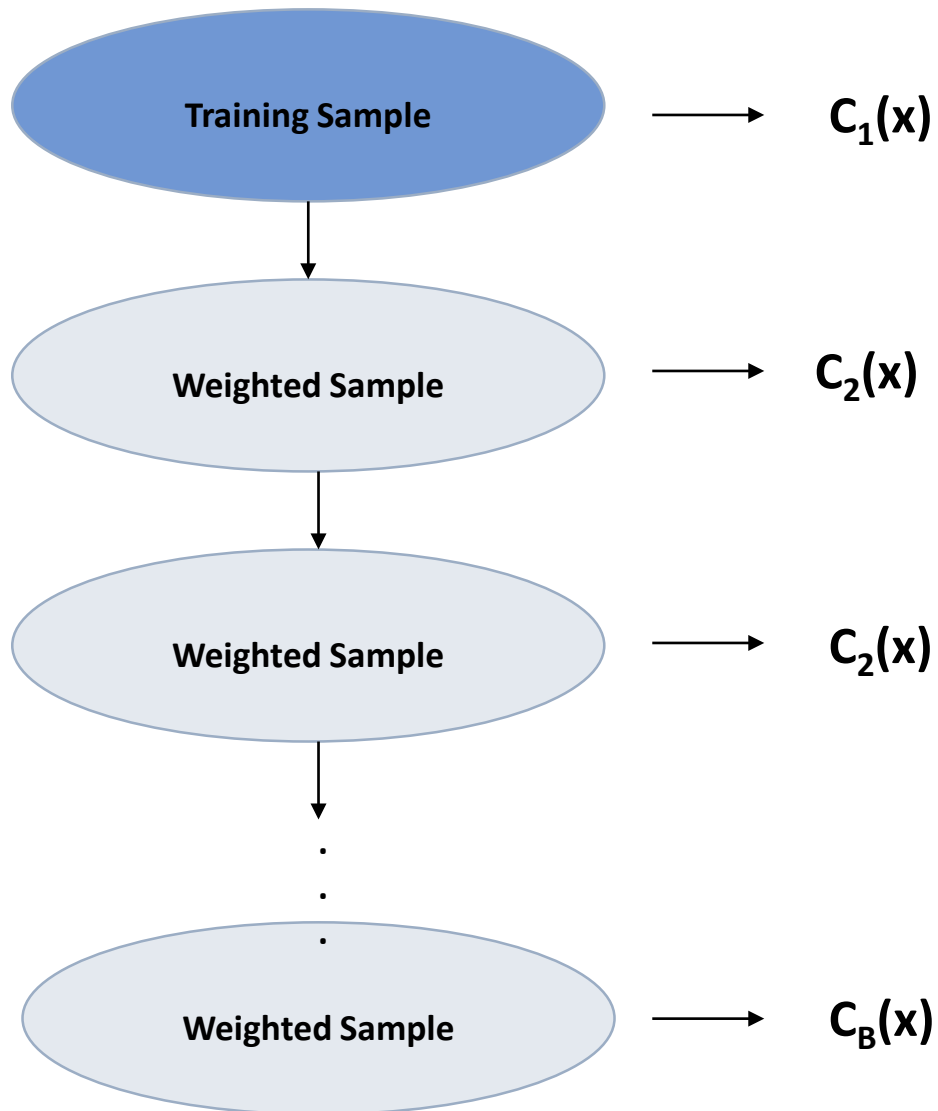


Update weights,  $D_3$



Trained classifier





Final hypothesis

$$C(x) = \text{sign} \left[ \sum_{m=1}^B \alpha_m C_m(x) \right]$$

# Algorithm AdaBoost.M1 - Adaptive Boosting

1. Initialize example weights  $w_i = \frac{1}{N}$  ( $i = 1, \dots, N$ )
2. for  $m = 1$  to  $B$  //  $B$  ... number of iterations
  - a) learn a classifier  $C_m$  using the current example weights
  - b) compute a weighted error estimate

$$err_m = \frac{\sum w_i \text{ of all incorrectly classified } e_i}{\sum_{i=1}^N w_i}$$

- c) compute a classifier weight  $\alpha_m = \frac{1}{2} \ln\left(\frac{1-err_m}{err_m}\right)$
- d) for all **correctly** classified examples  $e_i$ :  $w_i \leftarrow w_i e^{-\alpha_m}$
- e) for all **incorrectly** classified examples  $e_i$ :  $w_i \leftarrow w_i e^{+\alpha_m}$
- f) normalize the weights  $w_i$  so that they sum to 1

# Boosting – Algorithm AdaBoost.M1

...

3. for each test example

a) try all classifiers  $C_m$

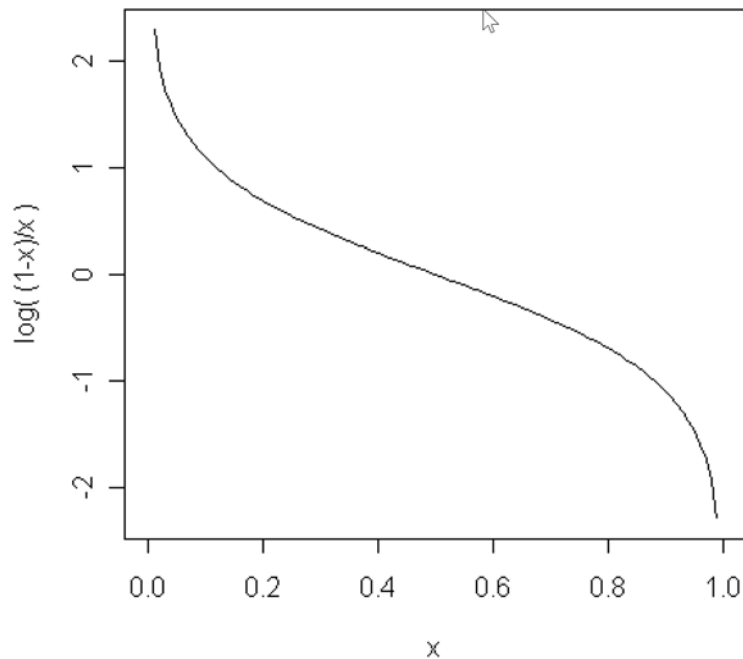
b) predict the class that receives the highest sum of weights  $\alpha_m$



# Illustration of the Weights

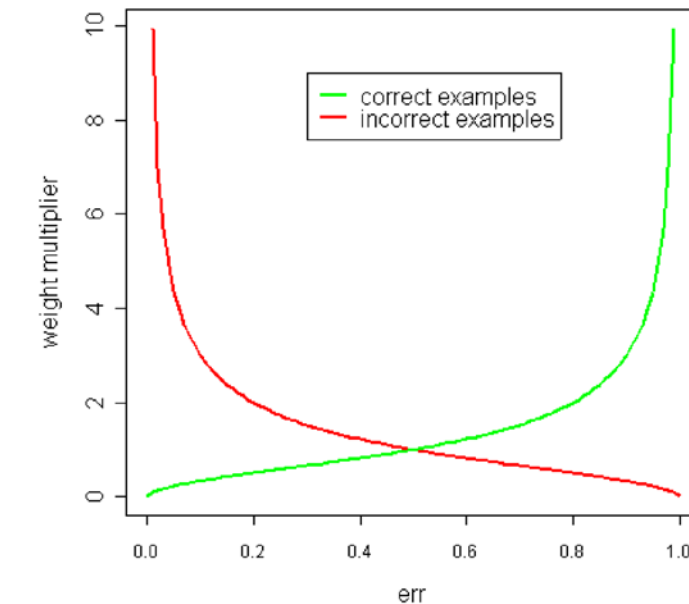
## Classifier Weights $\alpha_m$

differences near 0 or 1  
are emphasized



## Example Weights

multiplier for correct and  
incorrect examples, depending  
on error



# Boosting – Error rate example

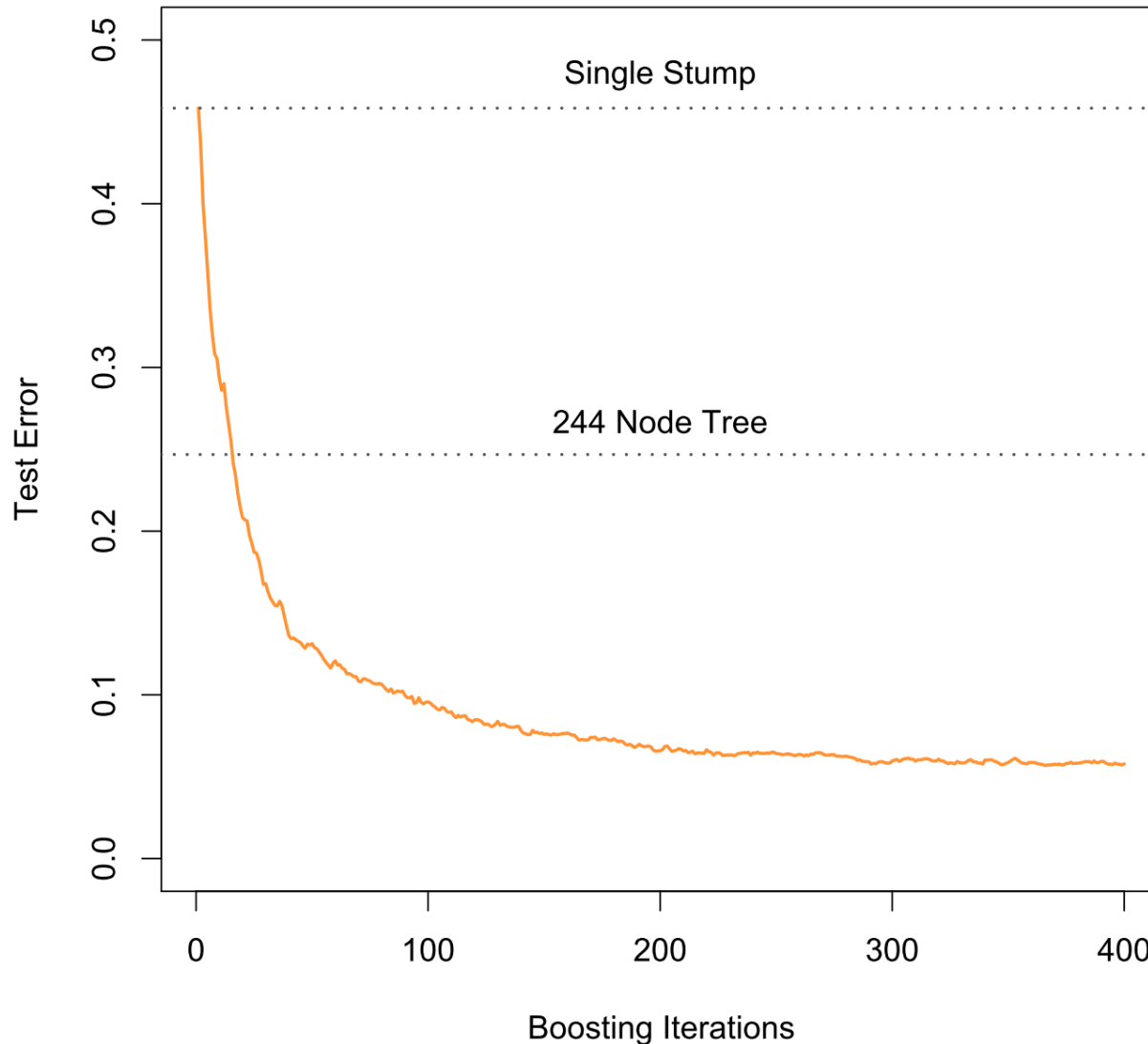


FIGURE 10.2. Simulated data (10.2): test error rate for boosting with stumps, as a function of the number of iterations. Also shown are the test error rate for a single stump, and a 244-node classification tree.

Hastie, T., Tibshirani, R., & Friedman, J. H. (2017). The elements of statistical learning: Data mining, inference, and prediction (12th ed). Springer series in statistics. New York: Springer.

# Boosting – Error rate example

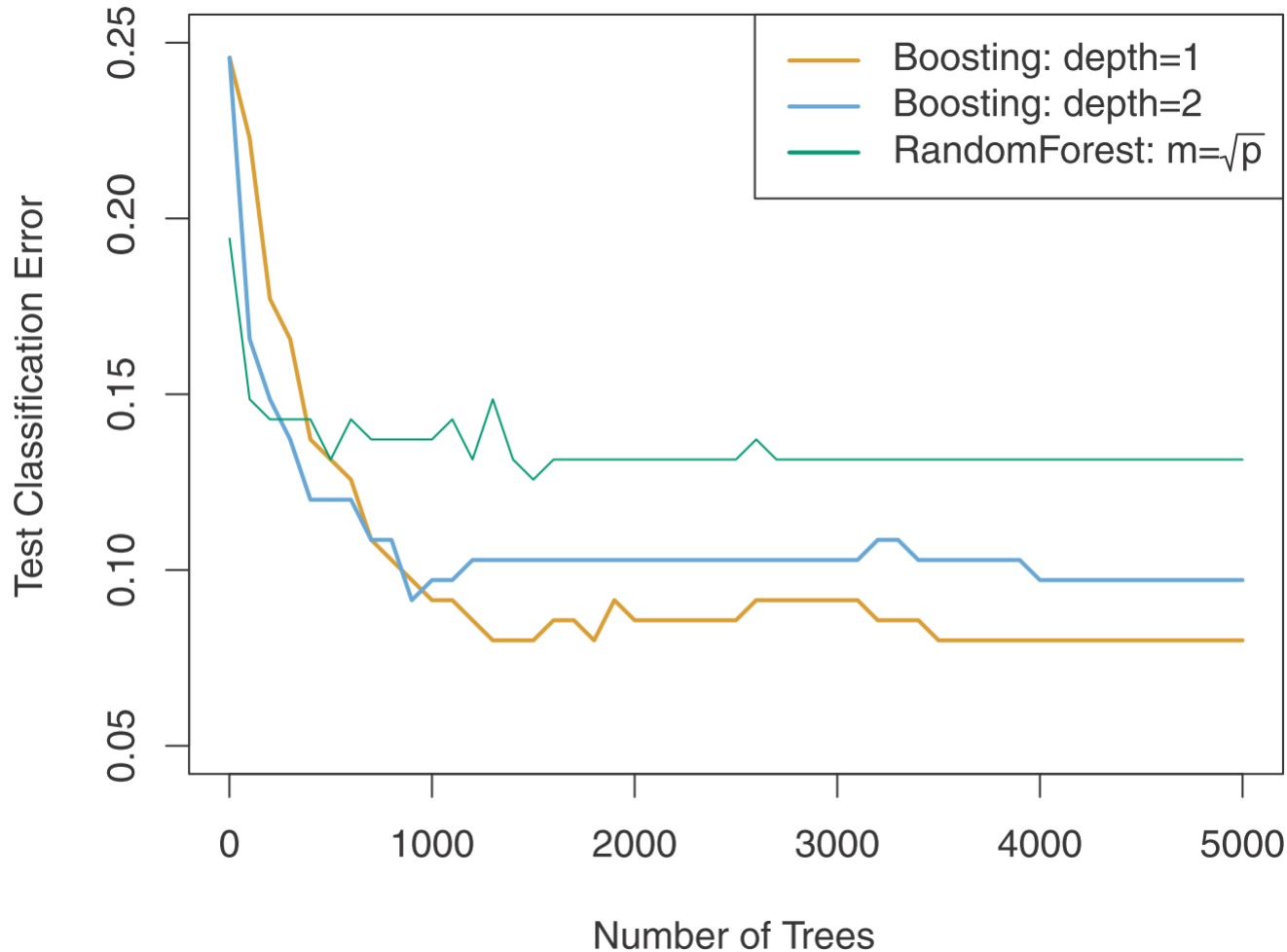
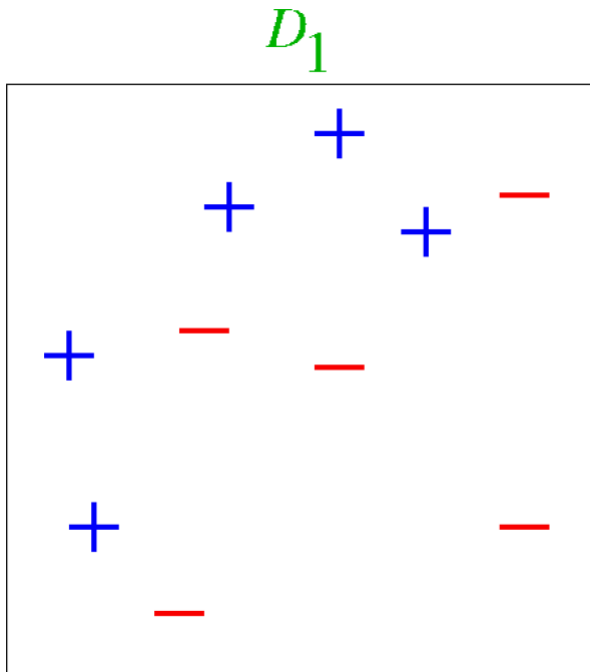


FIGURE 8.11. Results from performing boosting and random forests on the 15-class gene expression data set in order to predict cancer versus normal. The test error is displayed as a function of the number of trees. For the two boosted models,  $\lambda = 0.01$ . Depth-1 trees slightly outperform depth-2 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 %.

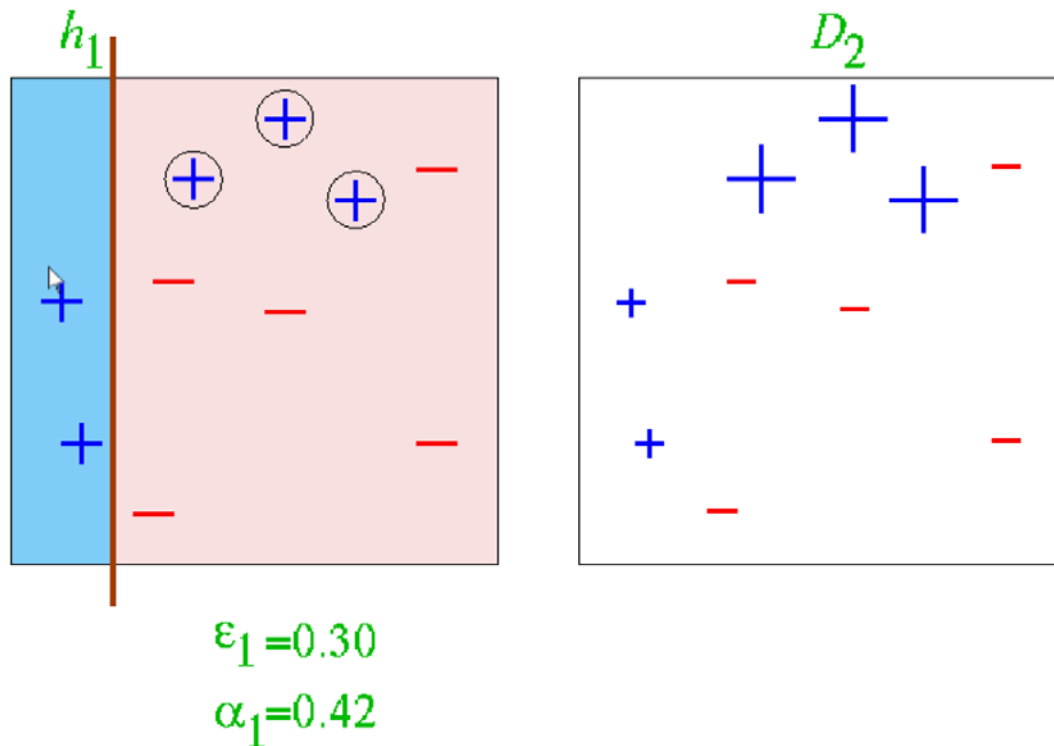
James, G., Witten, D., Hastie, T., & Tibshirani, R. (2014). An introduction to statistical learning: With applications in R (Corrected at 4th print). Springer texts in statistics. New York: Springer.

# Toy Example

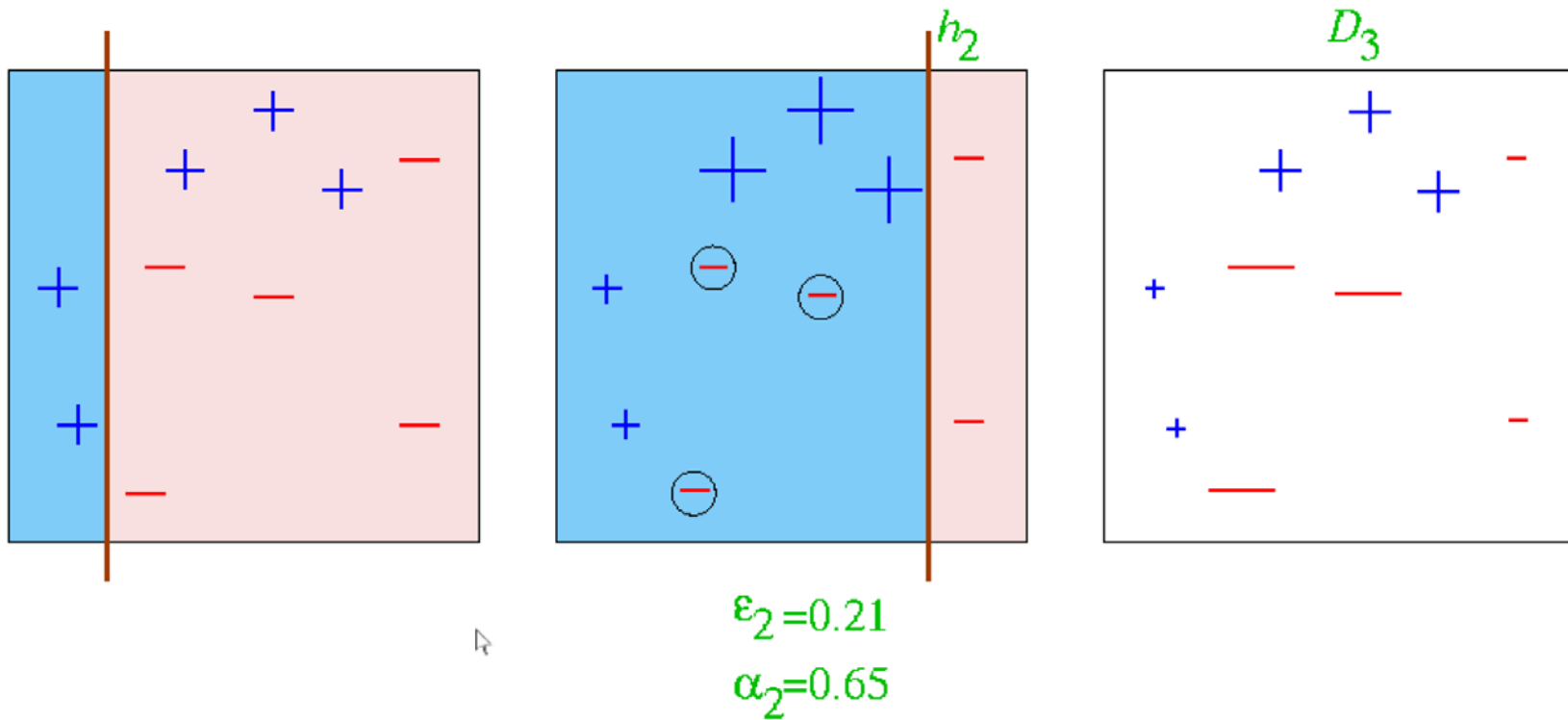


Taken from Verma & Thrun, Slides to CALD Course CMU 15-781, Machine Learning, Fall 2000)

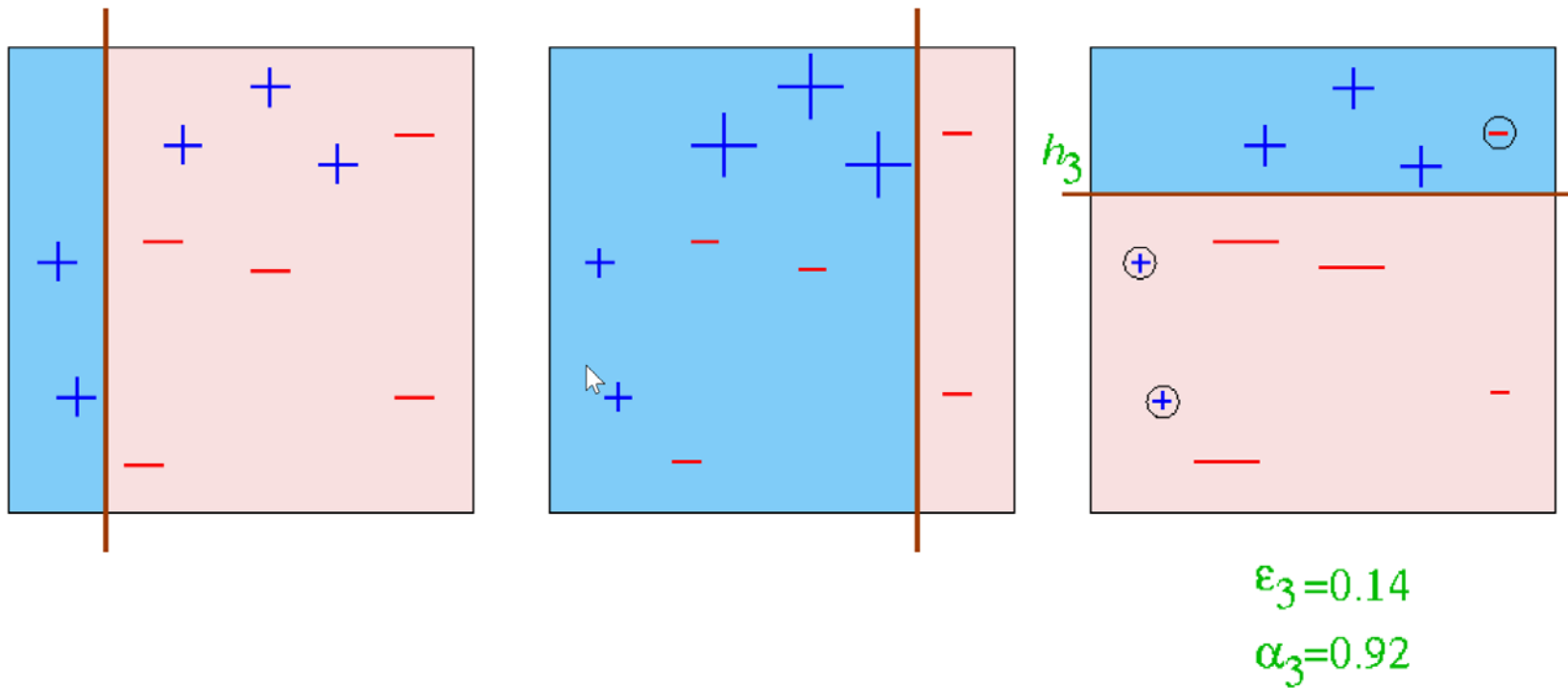
# Round 1



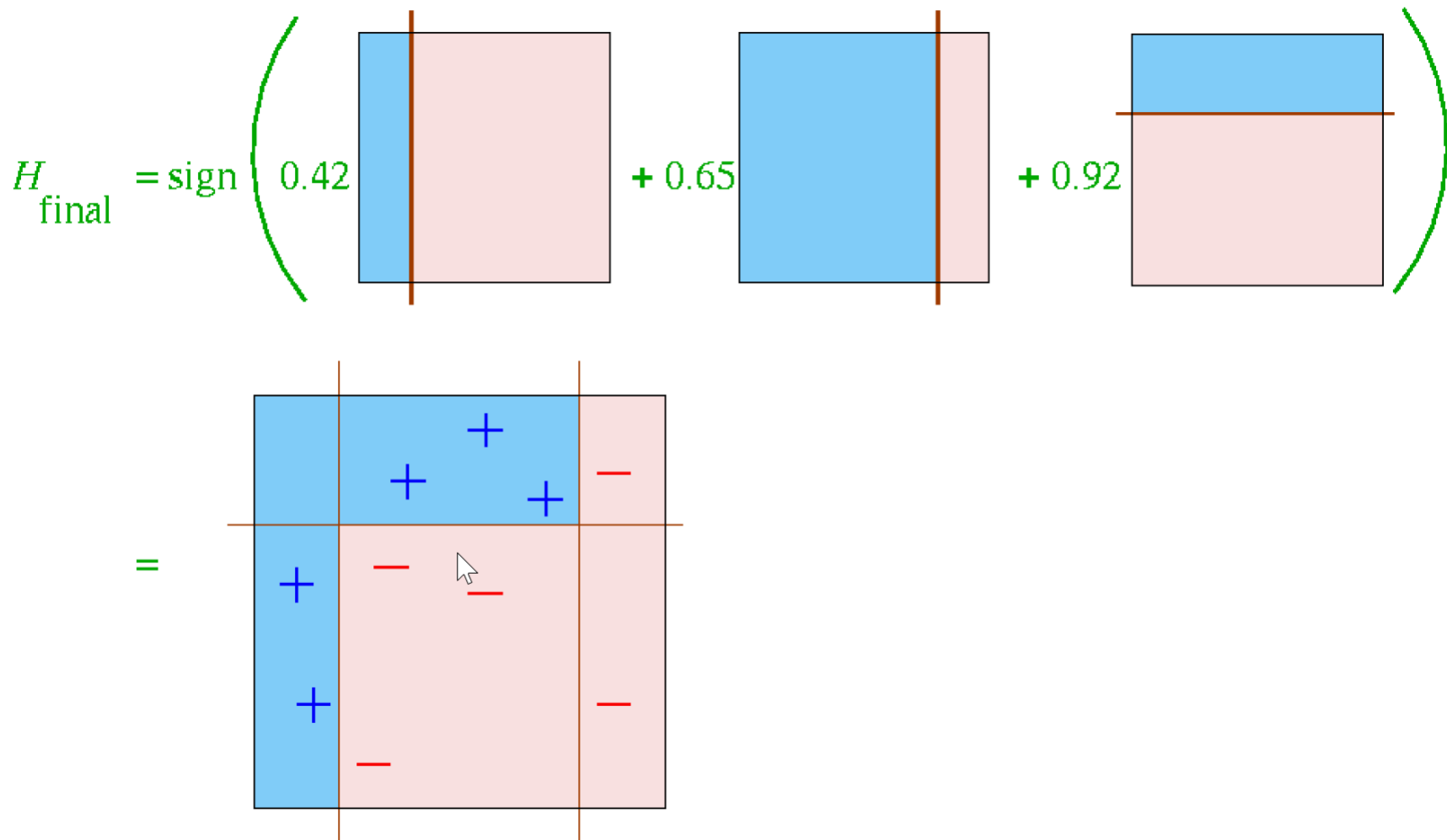
# Round 2



# Round 3



# Final Hypothesis





# Boosting

- Each of the trees can be rather small, with just a few terminal nodes.
- They learn slowly.
- In general, statistical learning approaches that learn slowly tend to perform well.
- Unlike bagging and random forests, boosting can overfit if the number of trees is too large, although this overfitting tends to occur slowly if at all. We use cross-validation to select the number of trees.
- In boosting, unlike in bagging, the construction of each tree depends strongly on the trees that have already been grown.

# Outline

---

- Ensemble Methods
- Random Forests
- AdaBoost
- Stacking
- Summary

# Combining Predictions

- voting
  - each ensemble member votes for one of the classes
  - predict the class with the highest number of vote (e.g., bagging)
- weighted voting
  - make a weighted sum of the votes of the ensemble members
  - weights typically depend
    - on the classifiers confidence in its prediction (e.g., the estimated probability of the predicted class)
    - on error estimates of the classifier (e.g., boosting)
- stacking
  - Why not use a classifier for making the final decision?
  - training material are the class labels of the training data and the (cross-validated) predictions of the ensemble members

## Learn a function that combines the predictions of the individual classifiers

- Train  $n$  different classifiers  $C_1 \dots C_n$  The base classifiers.
- Obtain predictions of the classifiers for the training examples
- Form a new data set (the meta data)
  - i.Classes  
the same as the original dataset
  - ii.Attributes  
one attribute for each base classifier  
value is the prediction of this classifier on the example
- Train a separate classifier  $M$  (the meta classifier)  
This is better done with cross-validation!

# Example Using a stacked classifier

Attributes			Class
$x_{11}$	...	$x_{1n_a}$	$t$
$x_{21}$	...	$x_{2n_a}$	$f$
...	...	...	...
$x_{n_e 1}$	...	$x_{n_e n_a}$	$t$

training set

$C_1$	$C_2$	...	$C_{n_c}$
$t$	$t$	...	$f$
$f$	$t$	...	$t$
...	...	...	...
$f$	$f$	...	$t$

predictions of the  
classifiers

$C_1$	$C_2$	...	$C_{n_c}$	Class
$t$	$t$	...	$f$	$t$
$f$	$t$	...	$t$	$f$
...	...	...	...	...
$f$	$f$	...	$t$	$t$

training set for stacking

Try each of the  
classifiers  $C_1 \dots C_n$

Form a feature vector  
consisting of their  
predictions

Submit these feature  
vectors to the meta  
classifier M

# Outline

---

- Ensemble Methods
- Random Forests
- AdaBoost
- Stacking
- Summary

# Summary

---