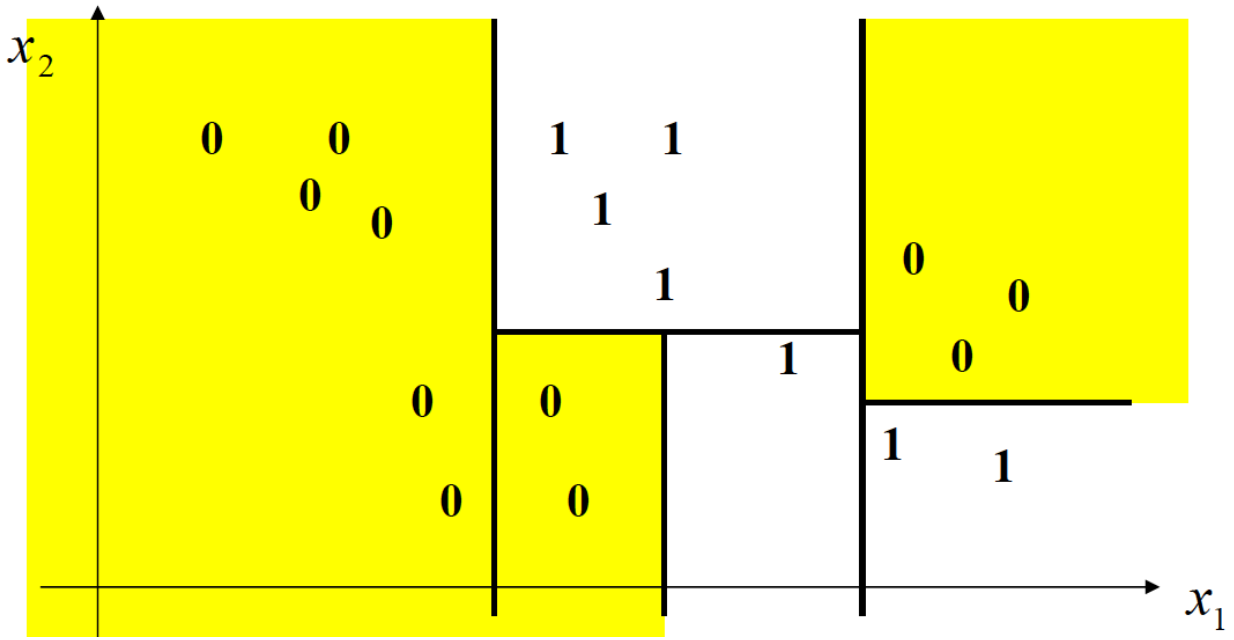
CS559 Lecture 20

Decision Tree and Random Forest

Decision trees

- An alternative approach to classification:
  - **Partition the input space to regions**
  - **Regress or classify independently in every region**



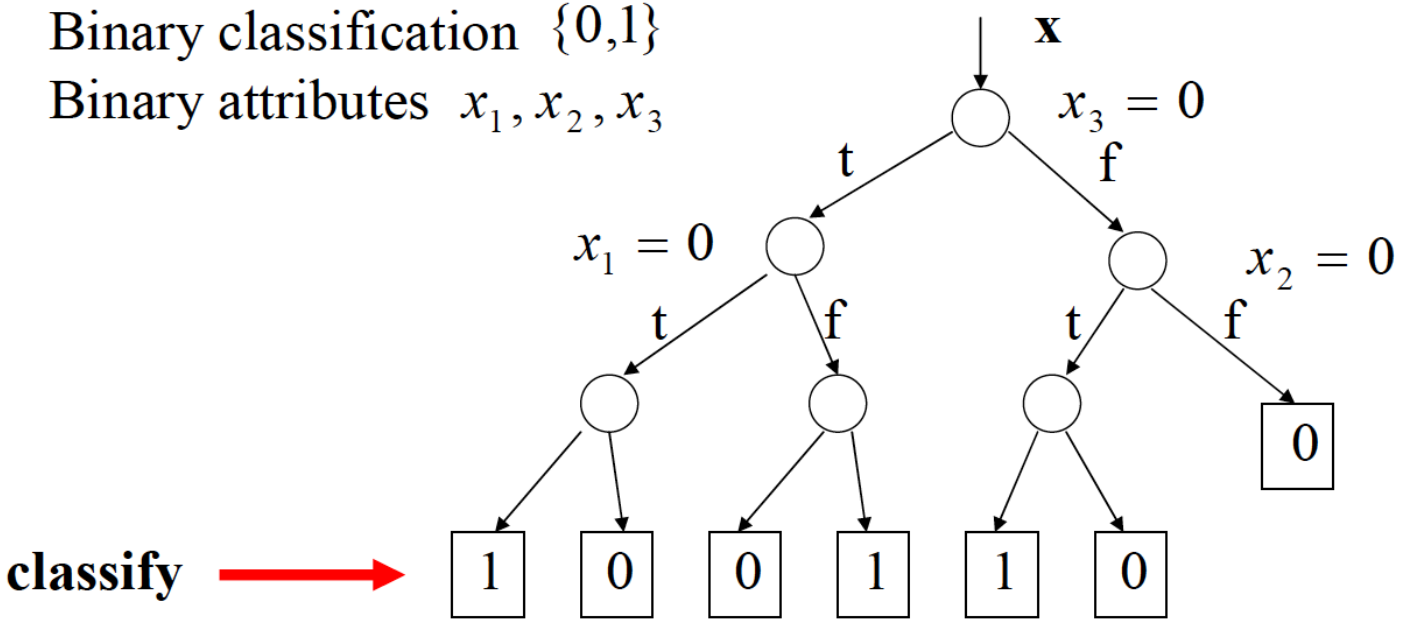
Decision trees

- The partitioning idea is used in the **decision tree model**:
  - Split the space recursively according to inputs in **x**
  - Regress or classify at the bottom of the tree

Example:

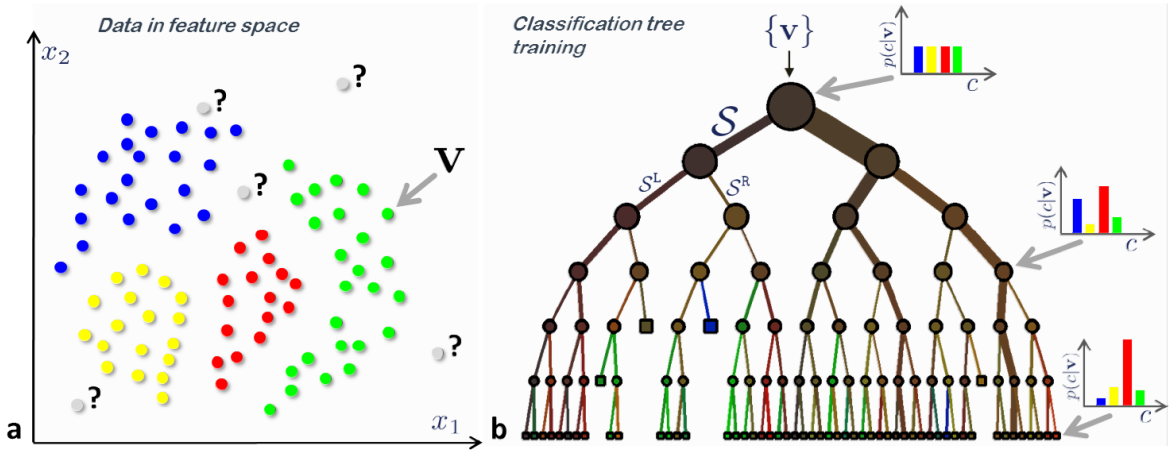
Binary classification $\{0,1\}$

Binary attributes x_1, x_2, x_3



Basic Notation

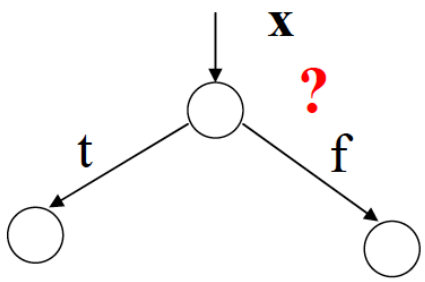
- Input data is represented as a collection of points in the d-dimensional space
- A decision tree is a hierarchical structure of connected nodes.
- Training a decision tree involves sending all training data {v} into the tree and optimizing the parameters of the split nodes to optimize a energy function.



Decision trees

How to construct the decision tree?

- **Top-bottom algorithm**:
  - Find the best split condition (quantified based on the impurity measure)
  - Stops when no improvement possible



- **Impurity measure**:
  - Measures how well are the two classes separated
  - Ideally we would like to separate all 0s and 1

- Splits of **finite vs. continuous value attributes**

  Continuous value attributes conditions: $x_3 \leq 0.5$

Impurity measure

Let $|D|$ - Total number of data entries

$|D_i|$ - Number of data entries classified as i

$p_i = \frac{|D_i|}{|D|}$ - ratio of instances classified as i

- **Impurity measure** defines how well the classes are separated
- In general the impurity measure should satisfy:
  - Largest when data are split evenly for attribute values

$$p_i = \frac{1}{\text{number of classes}}$$

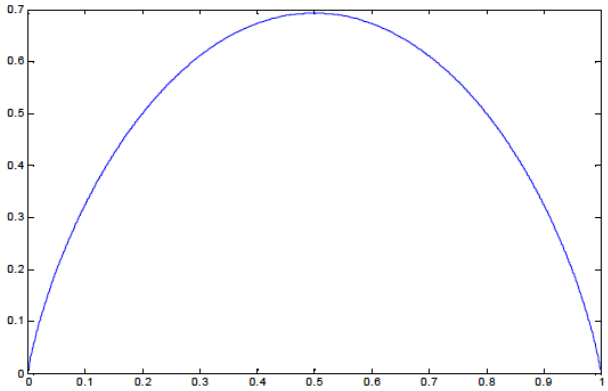
  - Should be 0 when all data belong to the same class

Impurity measures

- There are various impurity measures used in the literature
  - **Entropy based measure (Quinlan, C4.5)**

$$I(D) = Entropy\ (D) = -\sum_{i=1}^{k} p_i \log\ p_i$$

Example for k=2



  - **Gini measure (Breiman, CART)**

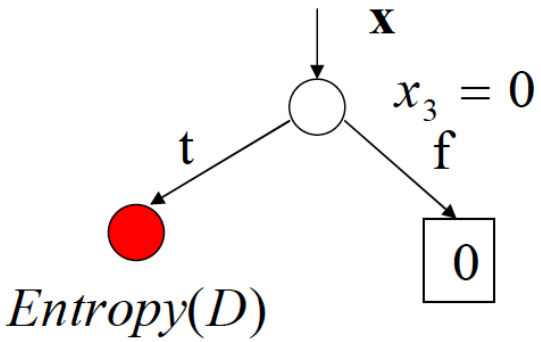
$$I(D) = Gini\ (D) = 1 - \sum_{i=1}^{k} p_i^{\ 2}$$

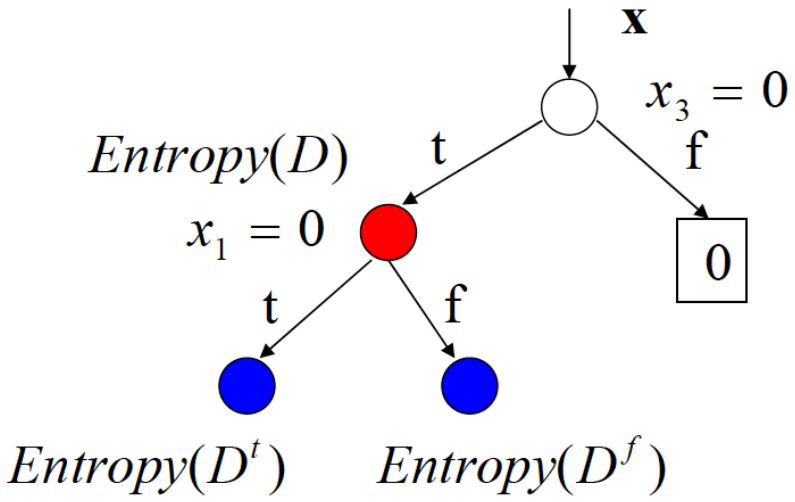
Impurity measures

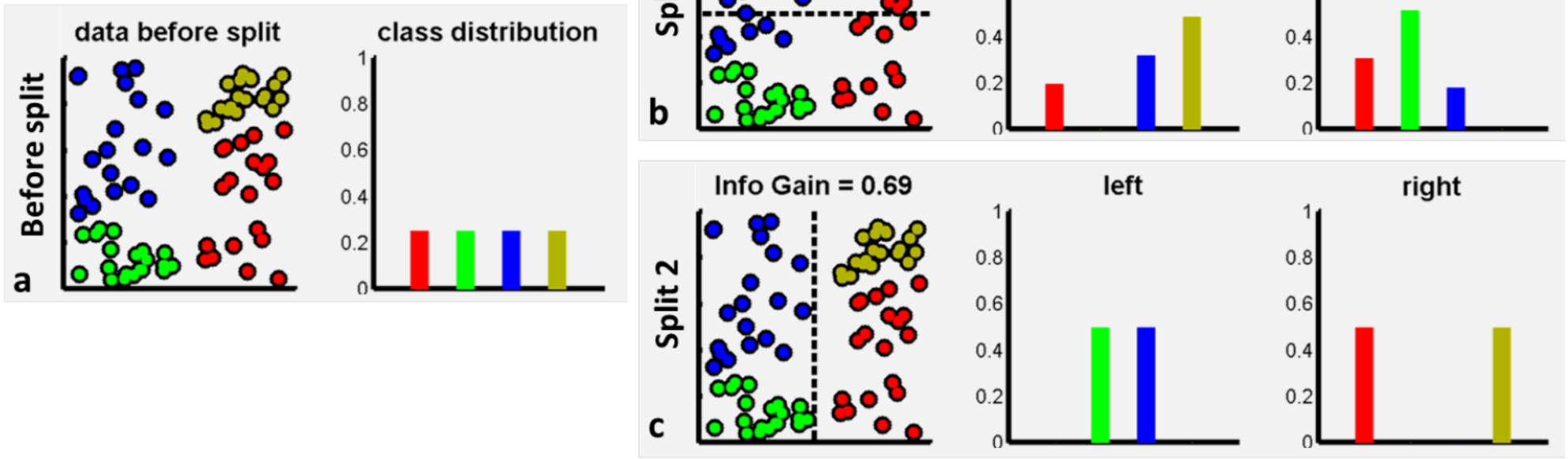
- **Gain due to split** – expected reduction in the impurity measure (entropy example)

$$Gain\ (D, A) = Entropy\ (D) - \sum_{v \in Values\ (A)} \frac{|\ D^v\ |}{|\ D\ |} Entropy\ (D^v)$$

$|\ D^v\ |$ - a partition of D with the value of attribute A = v






Before split
data before split
class distribution
1
0.8
0.6
0.4
0.2
0
a
Split 1
Info Gain = 0.40
top
bottom
b
Split 2
Info Gain = 0.69
left
right
c

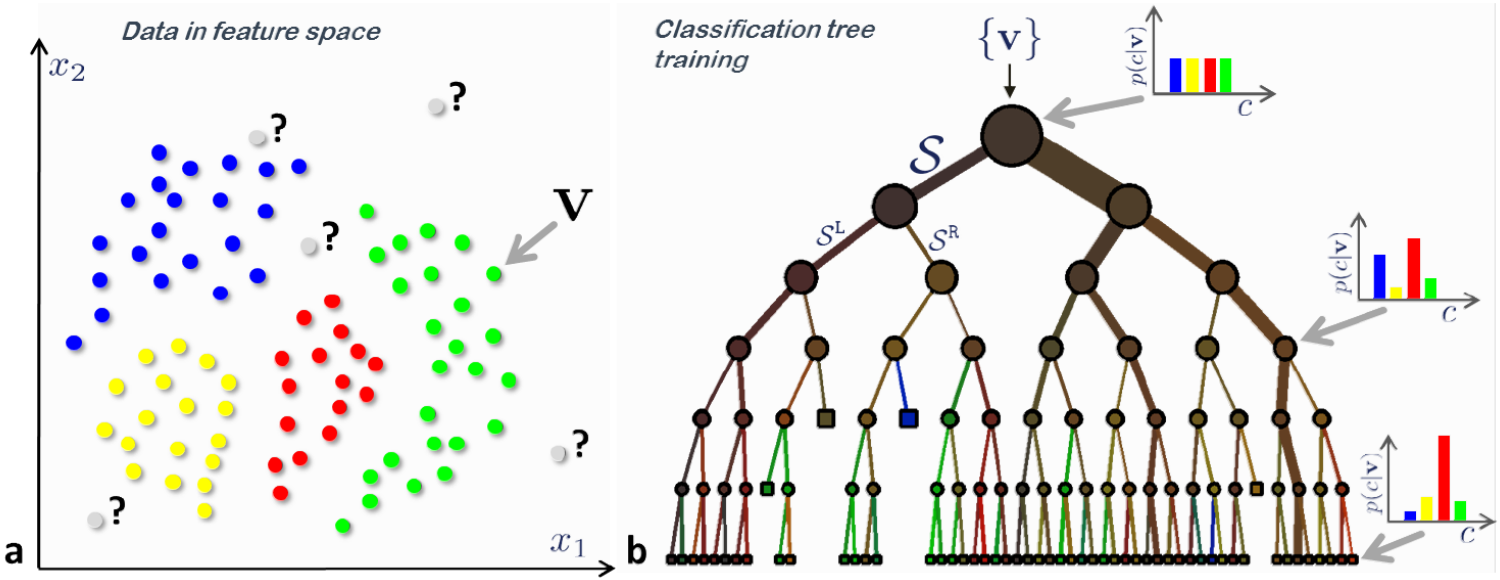
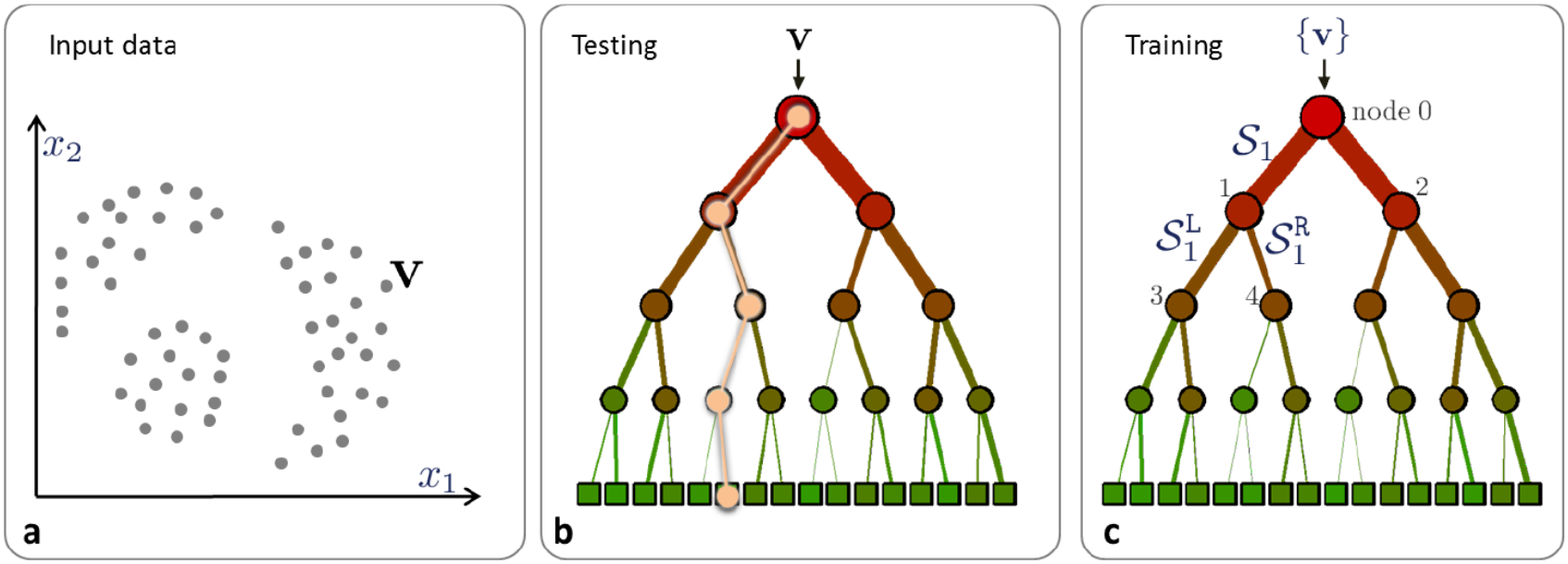


Fig. 3.1: **Classification: training data and tree training. (a)** Input data points. The ground-truth label of training points is denoted with different colours. Grey circles indicate unlabelled, previously unseen test data. **(b)** A binary classification tree. During training a set of labelled training points $\{\mathbf{v}\}$ is used to optimize the parameters of the tree. In a classification tree the entropy of the class distributions associated with different nodes decreases (the confidence increases) when going from the root towards the leaves.

Testing

- During testing, a split node applies a test to the input data v and sends it to the appropriate child

- The process is repeated until a leaf node is reached (beige path)



Decision tree learning

- **Greedy learning algorithm:**

  Repeat until no or small improvement in the purity

  – Find the attribute with the highest gain

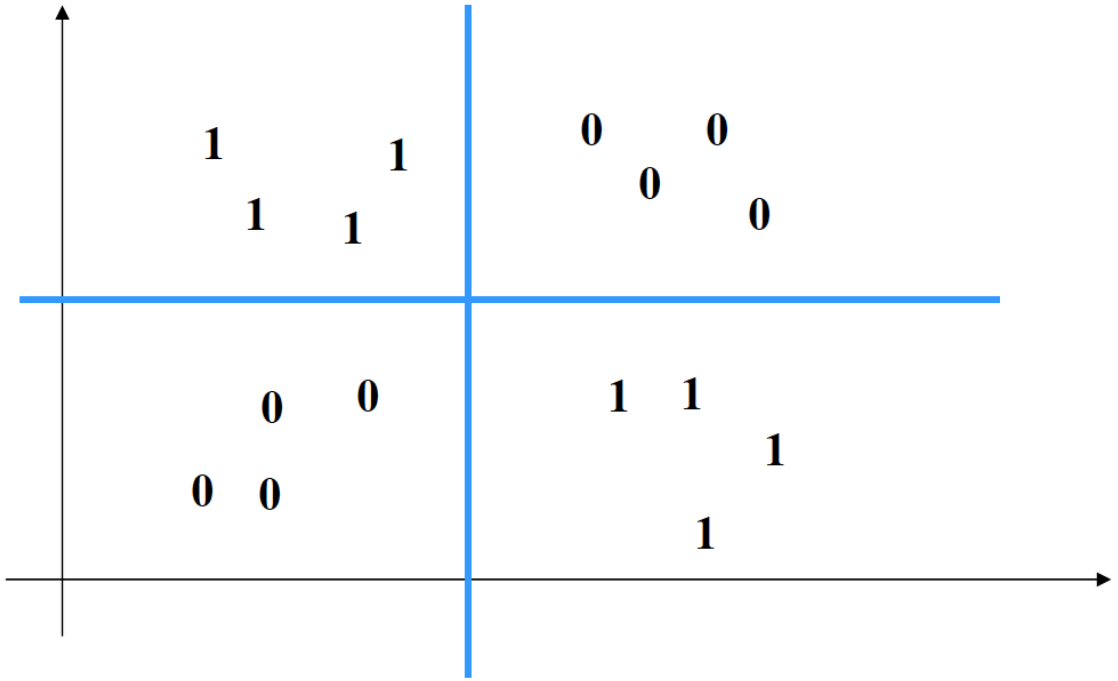
  – Add the attribute to the tree and split the set accordingly

- Builds the tree in the top-down fashion
  – Gradually expands the leaves of the partially built tree
- The method is greedy
  – It looks at a single attribute and gain in each step
  – May fail when the combination of attributes is needed to improve the purity (parity functions)

Decision tree learning

- **Limitations of greedy methods**

  Cases in which a combination of two or more attributes improves the impurity



Decision tree learning

By reducing the impurity measure we can grow **very large trees**

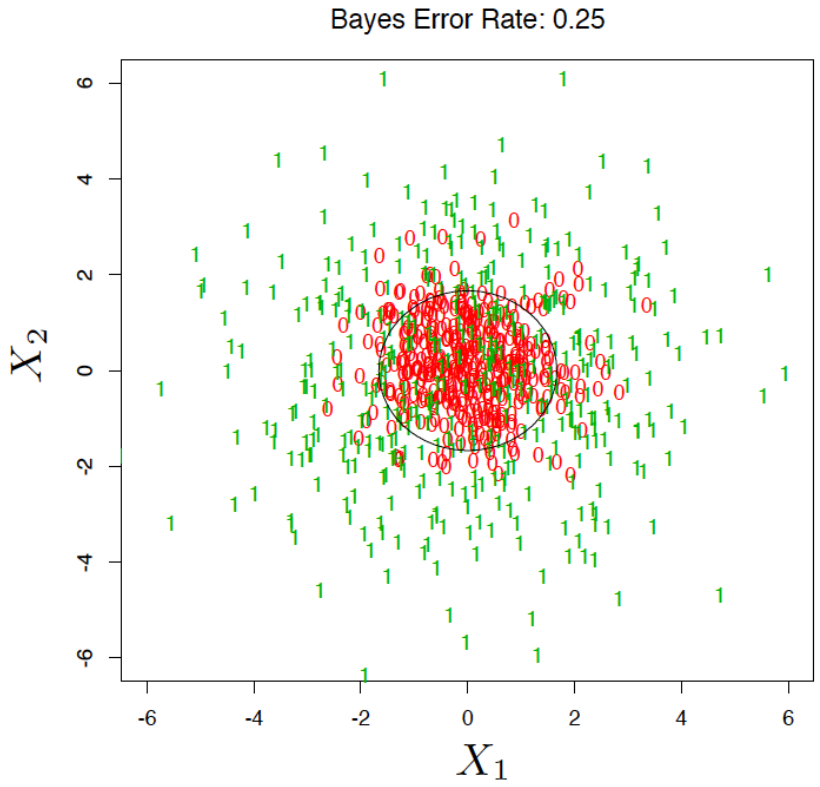
Problem: Overfitting

- We may split and classify very well the training set, but we may do worse in terms of the generalization error

Solutions to the overfitting problem:

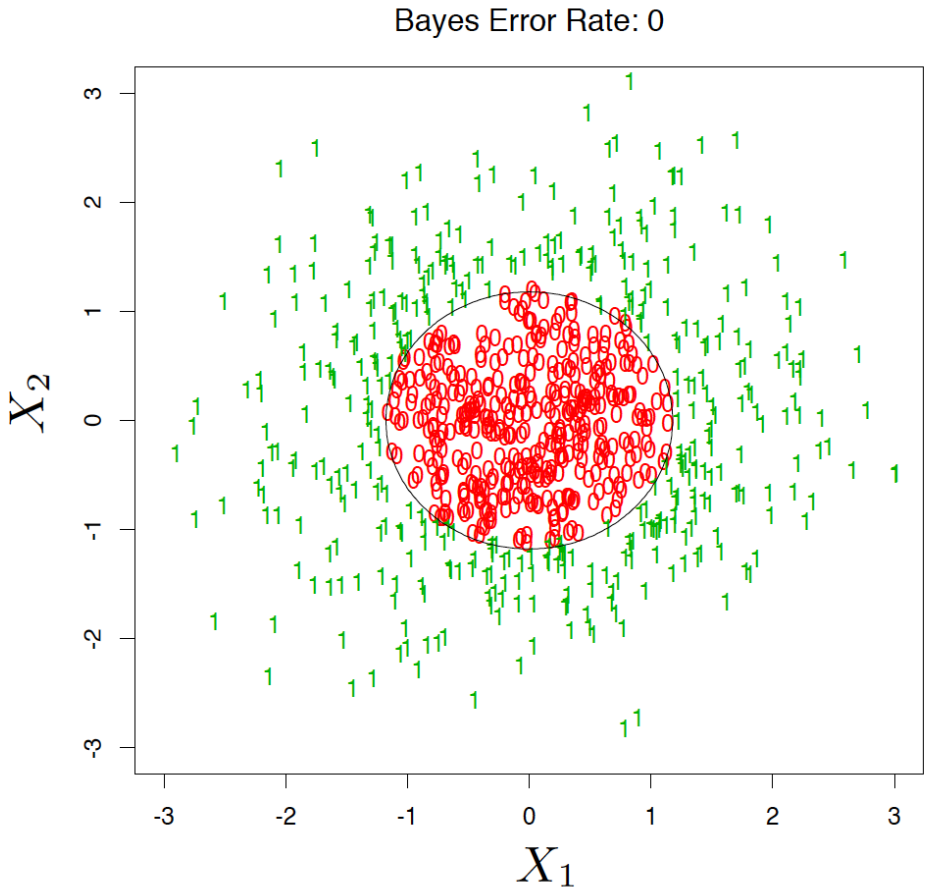
- **Solution 1.**
  - Prune branches of the tree built in the first phase
  - Use validation set to test for the overfit
- **Solution 2**.
  - Test for the overfit in the tree building phase
  - Stop building the tree when performance on the validation set deteriorates

Toy Classification Problem



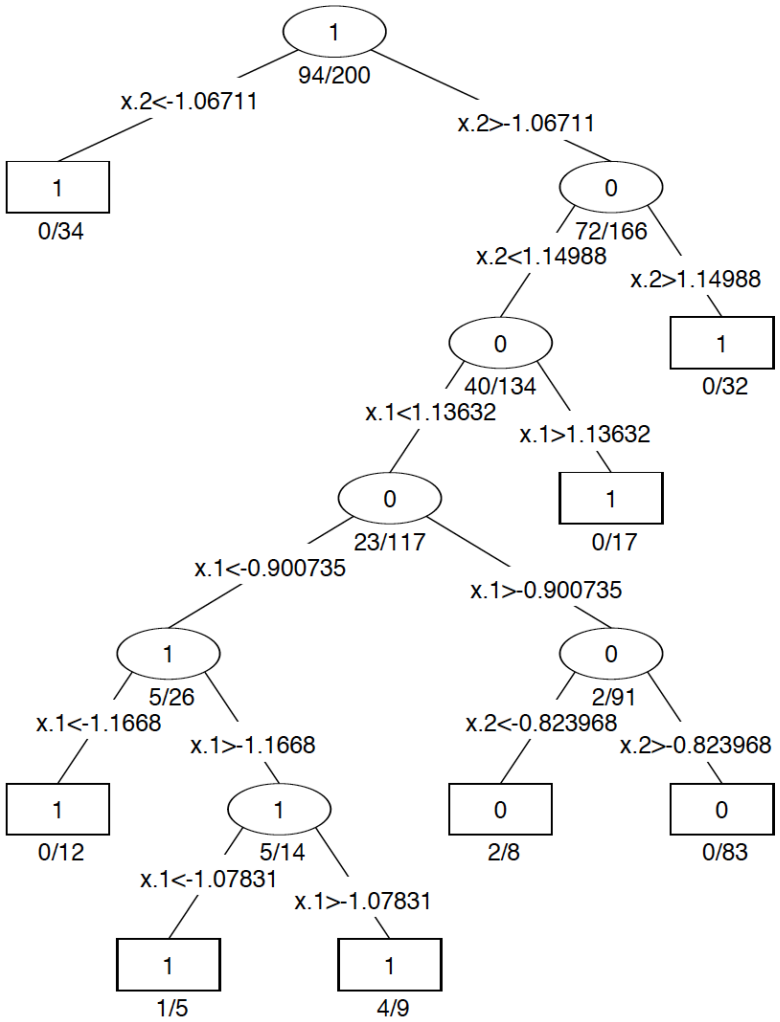
- Data X and Y, with Y taking values $+1$ or -1.
- Here $X = (X_1, X_2)$
- The black boundary is the Bayes Decision Boundary - the best one can do.
- Goal: Given N training pairs (X_i, Y_i) produce a classifier $\hat{C}(X) \in \{-1, 1\}$
- Also estimate the probability of the class labels $P(Y = +1|X)$.

Toy Example - No Noise

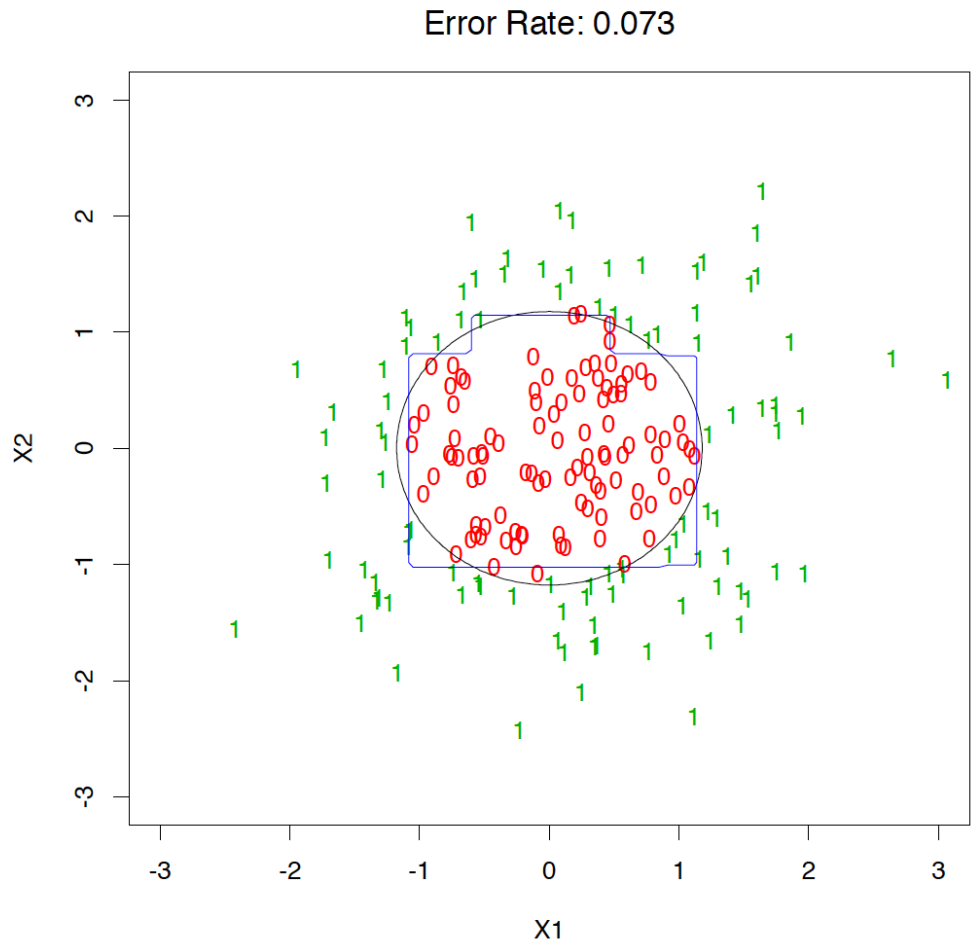


- Deterministic problem; noise comes from sampling distribution of X.
- Use a training sample of size 200.
- Here Bayes Error is 0%.

Classification Tree



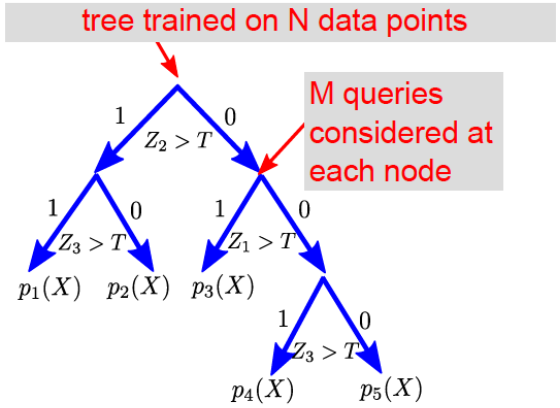
Decision Boundary: Tree



When the nested spheres are in 10-dimensions, Classification Trees produces a rather noisy and inaccurate rule $\hat{C}(X)$, with error rates around 30%.

Problems with decision trees/classification trees

- **Quality: Unclear when to terminate building the tree**
  - too deep and we overfit (doesn't generalise well to unseen data)
  - too shallow and we underfit (poor performance again)
- **Computational efficiency: Can be expensive to train**
  - for large numbers of queries: impossible to search over all of them
  - for large numbers of data-points: entropy computations slow



Model Averaging

Classification trees can be simple, but often produce noisy (bushy) or weak (stunted) classifiers.

- Bagging (Breiman, 1996): Fit many large trees to bootstrap-resampled versions of the training data, and classify by majority vote.
- Boosting (Freund & Shapire, 1996): Fit many large or small trees to reweighted versions of the training data. Classify by weighted majority vote.
- Random Forests (Breiman 1999): Fancier version of bagging.

In general Boosting $\succ$ Random Forests $\succ$ Bagging $\succ$ Single Tree.

Bagging

Bagging or bootstrap aggregation averages a given procedure over many samples, to reduce its variance — a poor man's Bayes. See

pp 246.

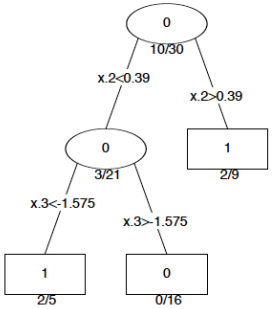
Suppose $C(\mathcal{S}, x)$ is a classifier, such as a tree, based on our training data $\mathcal{S}$, producing a predicted class label at input point x.

To bag C, we draw bootstrap samples $\mathcal{S}^{*1}, \ldots \mathcal{S}^{*B}$ each of size N with replacement from the training data. Then

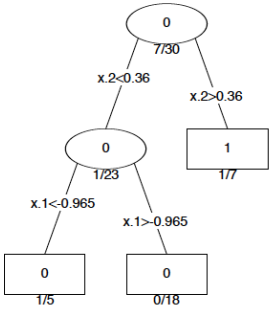
$$\hat{C}_{bag}(x) = \text{Majority Vote } \{C(\mathcal{S}^{*b}, x)\}_{b=1}^{B}.$$

Bagging can dramatically reduce the variance of unstable procedures (like trees), leading to improved prediction. However any simple structure in C (e.g a tree) is lost.

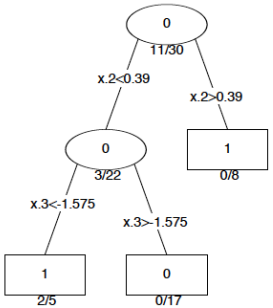
Original Tree



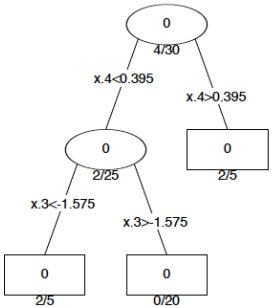
Bootstrap Tree 1



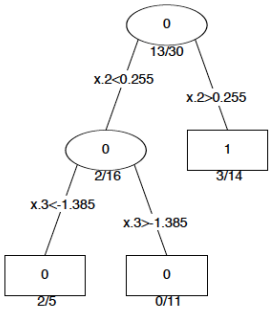
Bootstrap Tree 2



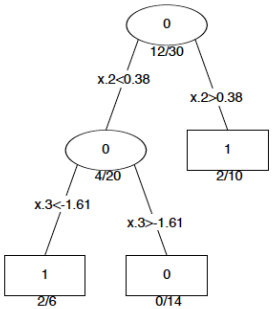
Bootstrap Tree 3



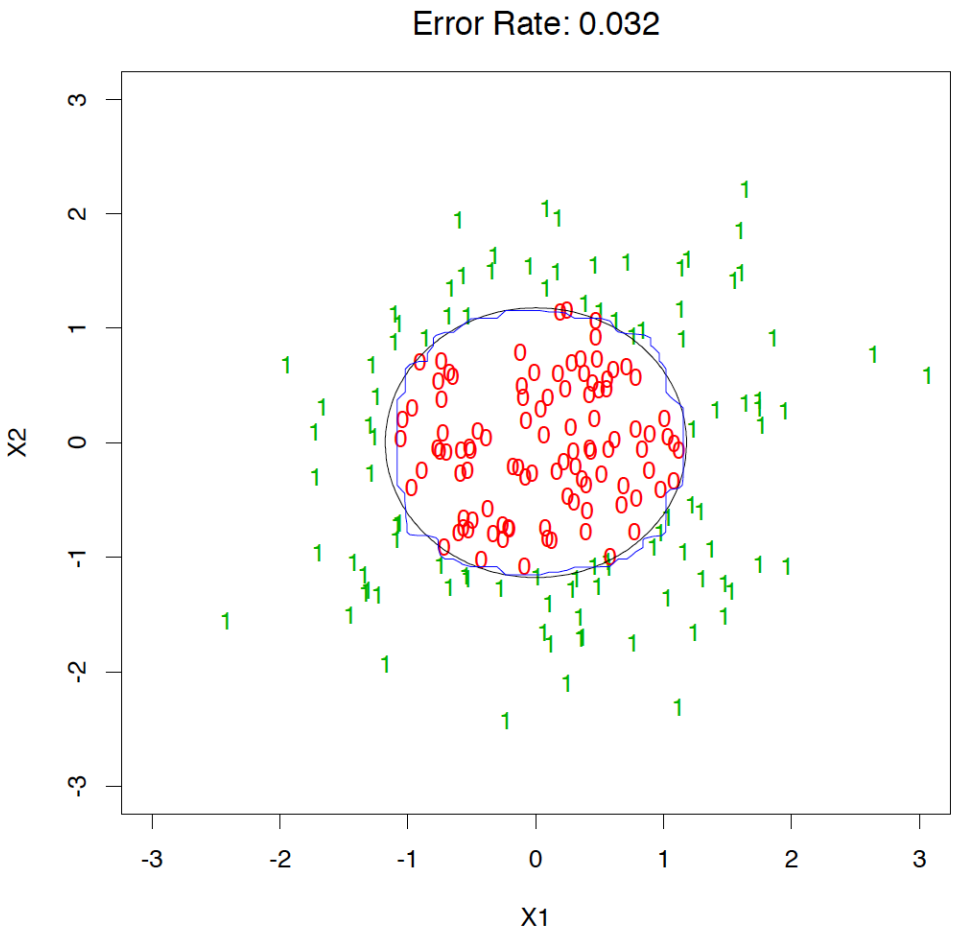
Bootstrap Tree 4



Bootstrap Tree 5



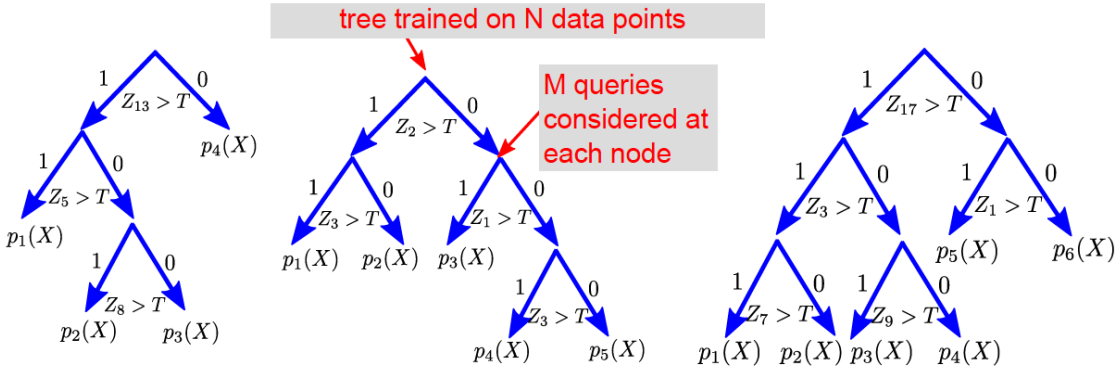
Decision Boundary: Bagging



Bagging averages many trees, and produces smoother decision boundaries.

Random Forests

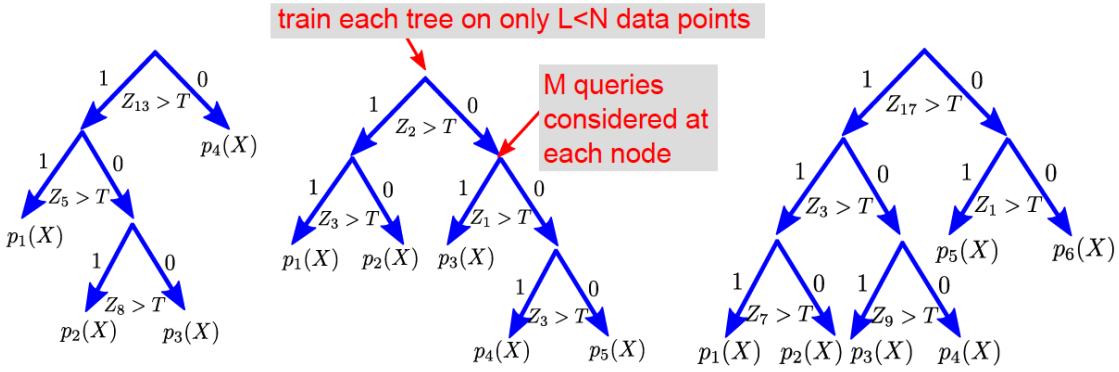
ensemble method – averages over (diverse) classification trees (a **forest**)



Random Forests

ensemble method – averages over (diverse) classification trees (a **forest**)

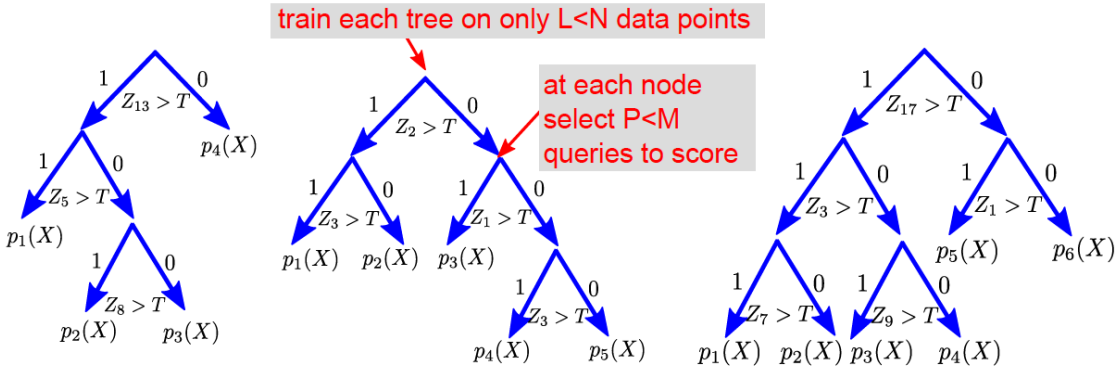
- for each tree draw L samples of the original data



Random Forests

ensemble method – averages over (diverse) classification trees (a **forest**)

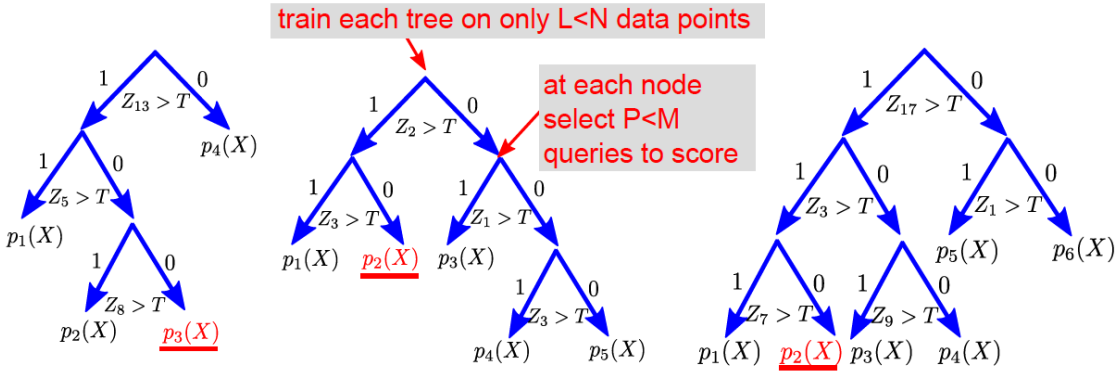
- for each tree draw L samples of the original data
- at each node randomly sample P queries and choose the best among them



Random Forests

ensemble method – averages over (diverse) classification trees (a **forest**)

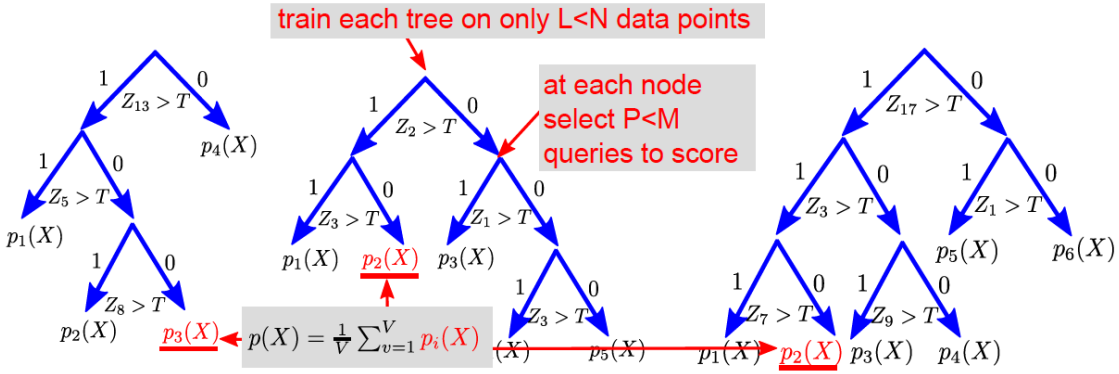
- for each tree draw L samples of the original data
- at each node randomly sample P queries and choose the best among them
- aggregate across trees (majority vote or average $\Longrightarrow$ mixture model)



Random Forests

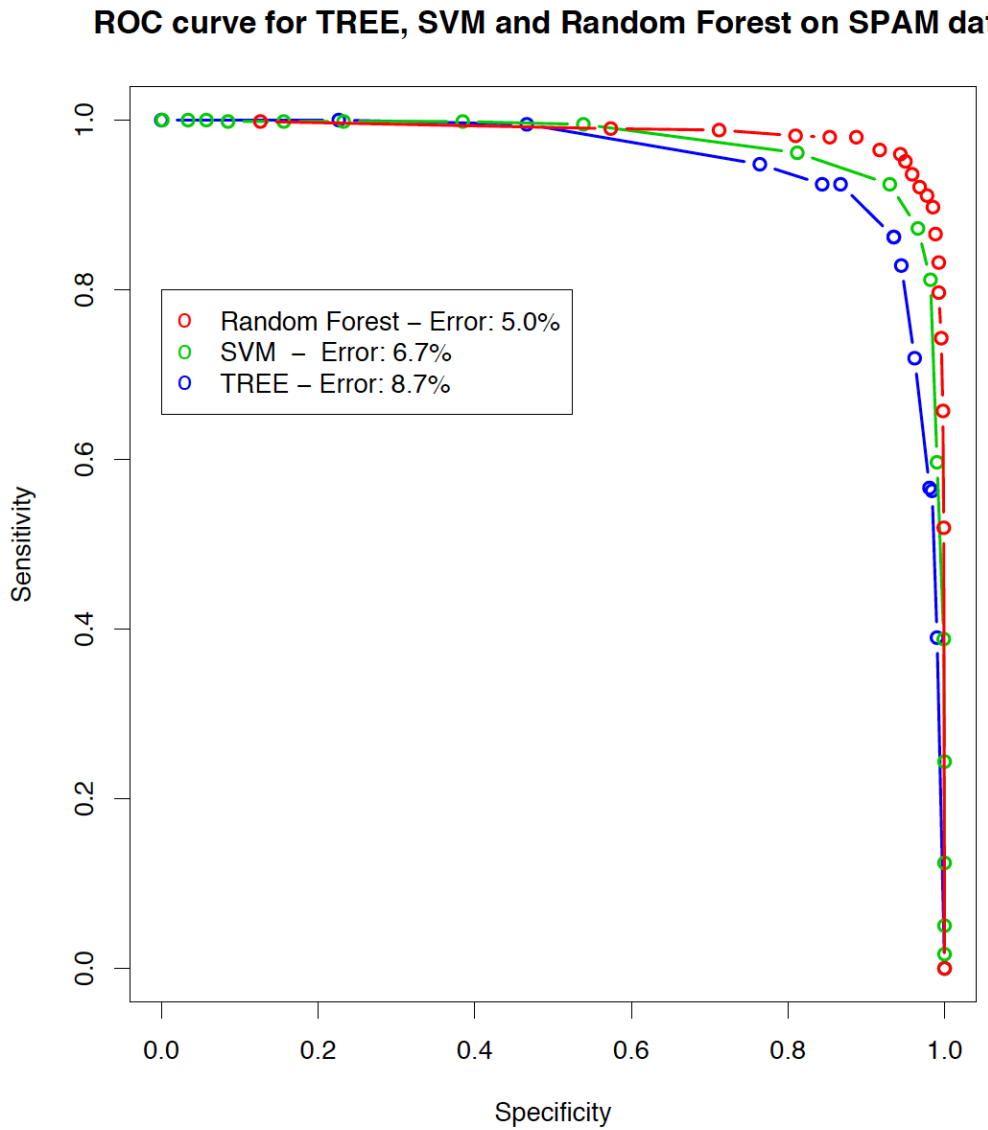
ensemble method – averages over (diverse) classification trees (a **forest**)

- for each tree draw L samples of the original data
- at each node randomly sample P queries and choose the best among them
- aggregate across trees (majority vote or average $\Longrightarrow$ mixture model)
- avoids over-fitting and computationally efficient



Random forests

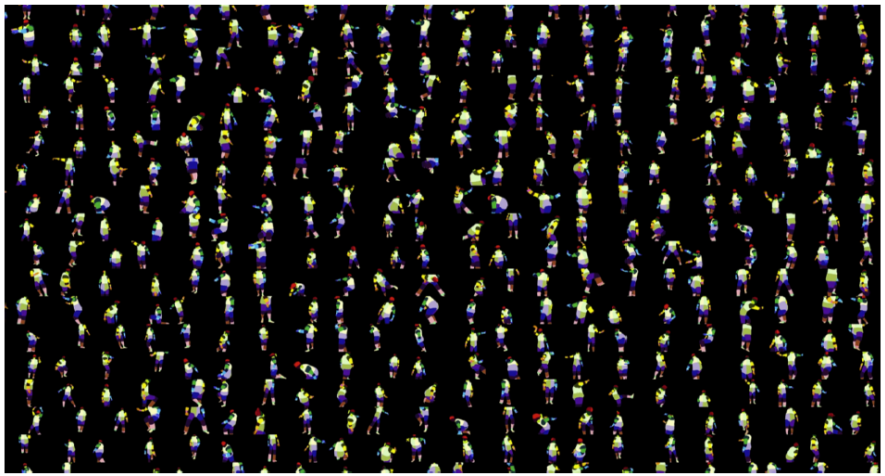
- refinement of bagged trees; quite popular
- at each tree split, a random sample of m features is drawn, and only those m features are considered for splitting. Typically $m = \sqrt{p}$ or $\log_2 p$, where p is the number of features
- For each tree grown on a bootstrap sample, the error rate for observations left out of the bootstrap sample is monitored. This is called the "out-of-bag" error rate.
- random forests tries to improve on bagging by "de-correlating" the trees. Each tree has the same expectation.



TREE, SVM and RF

Random Forest dominates both other methods on the SPAM data — 5.0% error. Used 500 trees with default settings for `randomForest` package in `R`.

Random forests: body part classification in the kinect



1 million test images, 1 day using a 1000 core cluster

Test set misclassification error (%)

Data set	**Forest**	**Single tree**
Breast cancer	2.9	5.9
Ionosphere	5.5	11.2
Diabetes	24.2	25.3
Glass	22.0	30.4
Soybean	5.7	8.6
Letters	3.4	12.4
Satellite	8.6	14.8
Shuttle $\times 10^3$	7.0	62.0
DNA	3.9	6.2
Digit	6.2	17.1

Demo

http://cs.stanford.edu/people/karpathy/svmjs/demo/demoforest.html

Summary of Random Forests

- random forests are a very popular tool for **classification** in computer vision
- based on **decision trees**: classifiers constructed greedily using the conditional entropy
- the extension hinges on two ideas:
  - building an **ensemble of trees** by **training on subsets of data**
  - considering a **reduced number of possible queries** at each node
- there are extensions for making **continuous predicitons** (regression, squared error scoring)