Data Analytics Course at IIFT

Dr. Tanujit Chakraborty

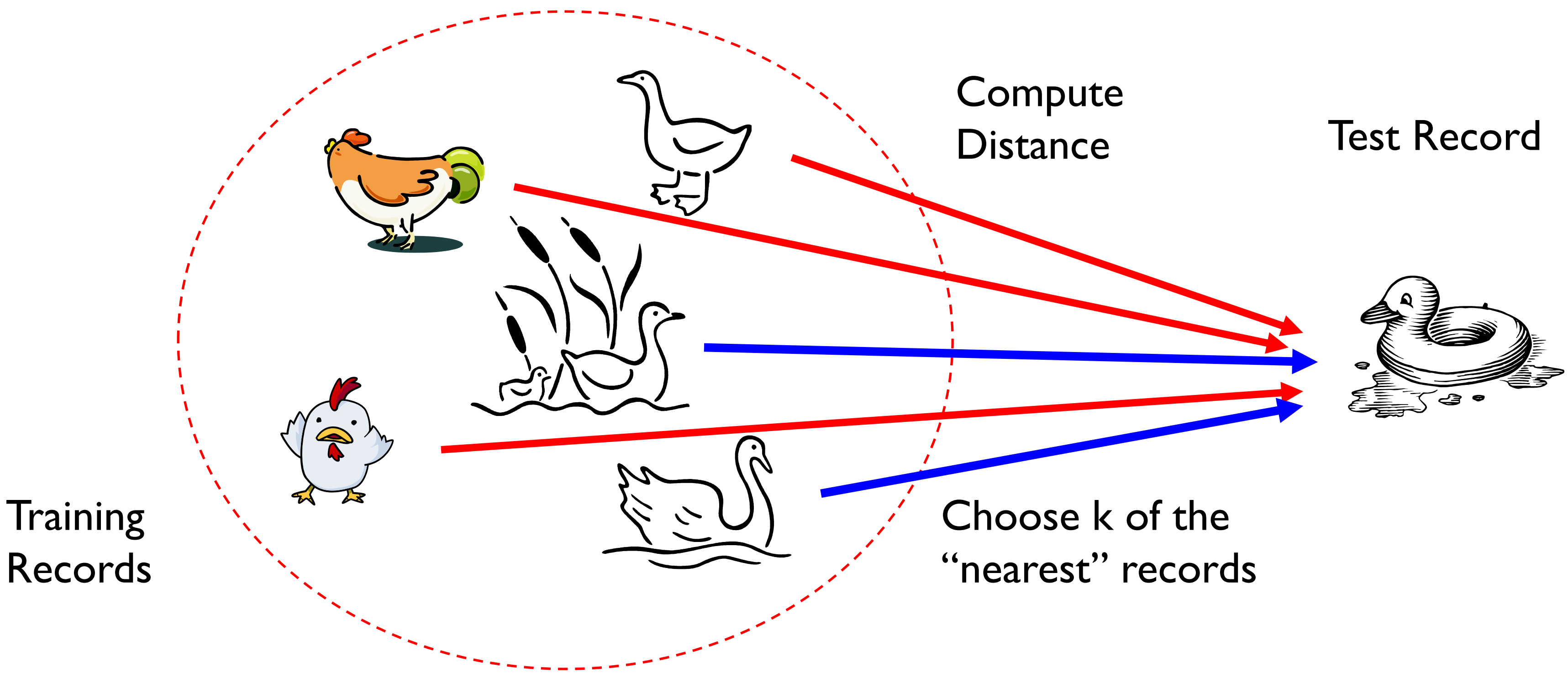
Session - 12

K-Nearest Neighbor Model

Decision Trees

Nearest Neighbor Classifiers

- Basic idea:
  - If it walks like a duck, quacks like a duck, then it's probably a duck



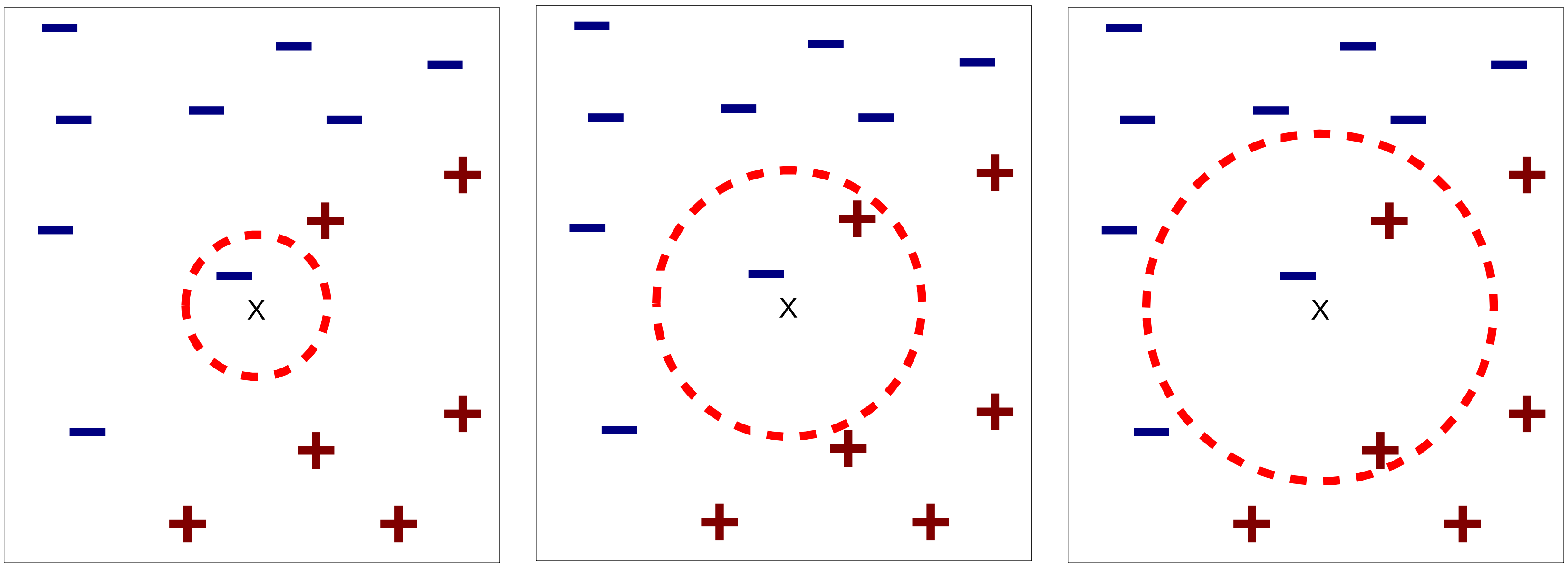
Basic Idea

- *k*-NN classification rule is to assign to a test sample the majority category label of its *k* nearest training samples
- In practice, *k* is usually chosen to be odd, so as to avoid ties
- The $k = 1$ rule is generally called the nearest-neighbor classification rule

Basic Idea

- kNN does not build model from the training data.
- To classify a test instance *d*, define *k*-neighborhood *P* as *k* nearest neighbors of *d*
- Count number *n* of training instances in *P* that belong to class c_j
- Estimate $\Pr(c_j|d)$ as n/k
- No training is needed. Classification time is linear in training set size for each test case.

Definition of Nearest Neighbor



(a) 1-nearest neighbor (b) 2-nearest neighbor (c) 3-nearest neighbor

K-nearest neighbors of a record x are data points that have the k smallest distance to x

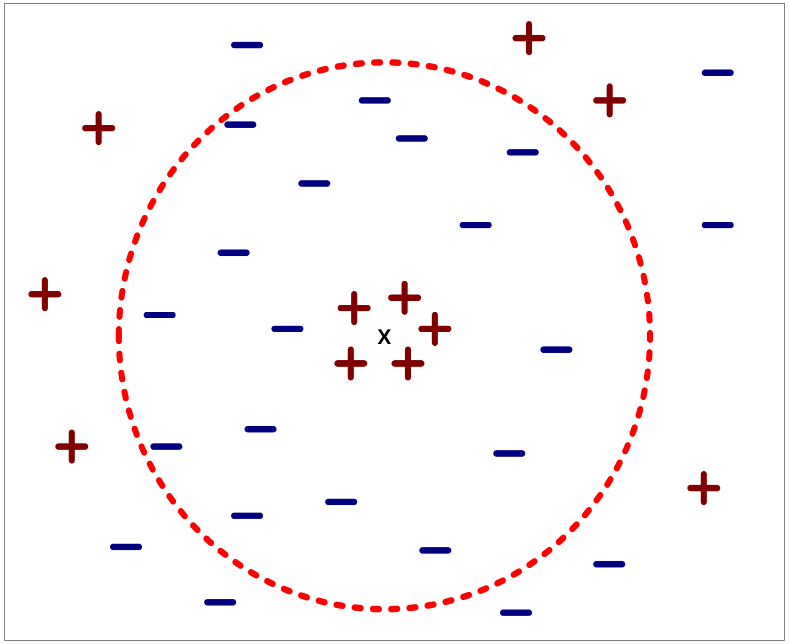
Nearest-Neighbor Classifiers: Issues

- The value of *k*, the number of nearest neighbors to retrieve
- Choice of Distance Metric to compute distance between records
- Computational complexity
  - Size of training set
  - Dimension of data

Value of K

- Choosing the value of k:
  - If k is too small, sensitive to noise points
  - If k is too large, neighborhood may include points from other classes

Rule of thumb:
K = sqrt(N)
N: number of training points



Distance Metrics

Minkowsky:

$$D(\boldsymbol{x},\boldsymbol{y}) = \left(\sum_{i=1}^{m} |x_i - y_i|^r \right)^{1/r}$$

Euclidean:

$$D(\boldsymbol{x},\boldsymbol{y}) = \sqrt{\sum_{i=1}^{m} (x_i - y_i)^2}$$

Manhattan / city-block:

$$D(\boldsymbol{x},\boldsymbol{y}) = \sum_{i=1}^{m} |x_i - y_i|$$

Camberra:

$$D(\boldsymbol{x},\boldsymbol{y}) = \sum_{i=1}^{m} \frac{|x_i - y_i|}{|x_i + y_i|}$$

Chebychev:

$$D(\boldsymbol{x},\boldsymbol{y}) = \max_{i=1}^{m} |x_i - y_i|$$

Quadratic:

$$D(\boldsymbol{x},\boldsymbol{y}) = (\boldsymbol{x}-\boldsymbol{y})^T Q(\boldsymbol{x}-\boldsymbol{y}) = \sum_{j=1}^{m} \left(\sum_{i=1}^{m} (x_i - y_i) q_{ji} \right) (x_j - y_j)$$

Q is a problem-specific positive definite $m \times m$ weight matrix

Mahalanobis:

$$D(\boldsymbol{x},\boldsymbol{y}) = [\det V]^{1/m} (\boldsymbol{x}-\boldsymbol{y})^{\mathrm{T}} V^{-1} (\boldsymbol{x}-\boldsymbol{y})$$

V is the covariance matrix of $A_1..A_m$, and A_j is the vector of values for attribute j occuring in the training set instances $1..n$.

Correlation:

$$D(\boldsymbol{x},\boldsymbol{y}) = \frac{\sum_{i=1}^{m} (x_i - \overline{x_i})(y_i - \overline{y_i})}{\sqrt{\sum_{i=1}^{m} (x_i - \overline{x_i})^2 \sum_{i=1}^{m} (y_i - \overline{y_i})^2}}$$

$\overline{x_i} = \overline{y_i}$ and is the average value for attribute i occuring in the training set.

Chi-square:

$$D(\boldsymbol{x},\boldsymbol{y}) = \sum_{i=1}^{m} \frac{1}{sum_i} \left(\frac{x_i}{size_{\boldsymbol{x}}} - \frac{y_i}{size_{\boldsymbol{y}}} \right)^2$$

sum_i is the sum of all values for attribute i occuring in the training set, and $size_{\boldsymbol{x}}$ is the sum of all values in the vector $\boldsymbol{x}$.

Kendall's Rank Correlation:

$$D(\boldsymbol{x},\boldsymbol{y}) = 1 - \frac{2}{n(n-1)} \sum_{i=1}^{m} \sum_{j=1}^{i-1} \operatorname{sign}(x_i - x_j) \operatorname{sign}(y_i - y_j)$$

sign(x)=-1, 0 or 1 if $x < 0$, $x = 0$, or $x > 0$, respectively.

Figure 1. Equations of selected distance functions. ($\boldsymbol{x}$ and $\boldsymbol{y}$ are vectors of m attribute values).

Distance Measure: Scale Effects

- Different features may have different measurement scales
  - E.g., patient weight in kg (range [50,200]) vs. blood protein values in ng/dL (range [-3,3])
- Consequences
  - Patient weight will have a much greater influence on the distance between samples
  - May bias the performance of the classifier

Standardization

- Transform raw feature values into z-scores

$$z_{ij} = \frac{x_{ij} - m_j}{s_j}$$

  -
  - x_{ij} is the value for the i^{th} sample and j^{th} feature
  - m_j is the average of all x_{ij} for feature j
  - s_j is the standard deviation of all x_{ij} over all input samples
- Range and scale of z-scores should be similar (providing distributions of raw feature values are alike)

Decision Trees

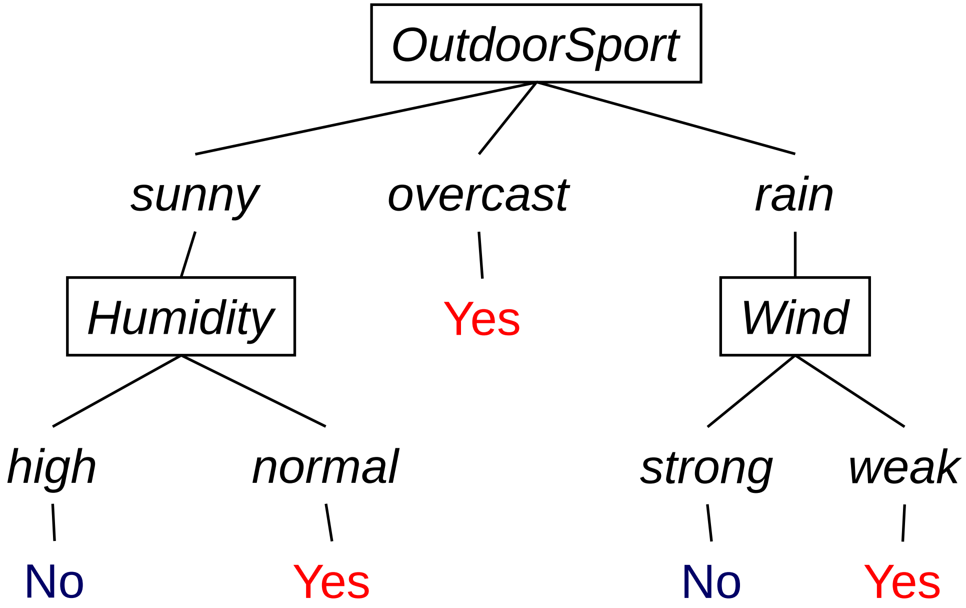
Training Examples

Day	Outlook	Temp	Humidity	Wind	Tennis?
D1	Sunny	Hot	High	Weak	*No*
D2	Sunny	Hot	High	Strong	*No*
D3	Overcast	Hot	High	Weak	*Yes*
D4	Rain	Mild	High	Weak	*Yes*
D5	Rain	Cool	Normal	Weak	*Yes*
D6	Rain	Cool	Normal	Strong	*No*
D7	Overcast	Cool	Normal	Strong	*Yes*
D8	Sunny	Mild	High	Weak	*No*
D9	Sunny	Cool	Normal	Weak	*Yes*
D10	Rain	Mild	Normal	Weak	*Yes*
D11	Sunny	Mild	Normal	Strong	*Yes*
D12	Overcast	Mild	High	Strong	*Yes*
D13	Overcast	Hot	Normal	Weak	*Yes*
D14	Rain	Mild	High	Strong	*No*

Representation of Concepts

Decision trees: <u>disjunction of conjunction of attributes</u>

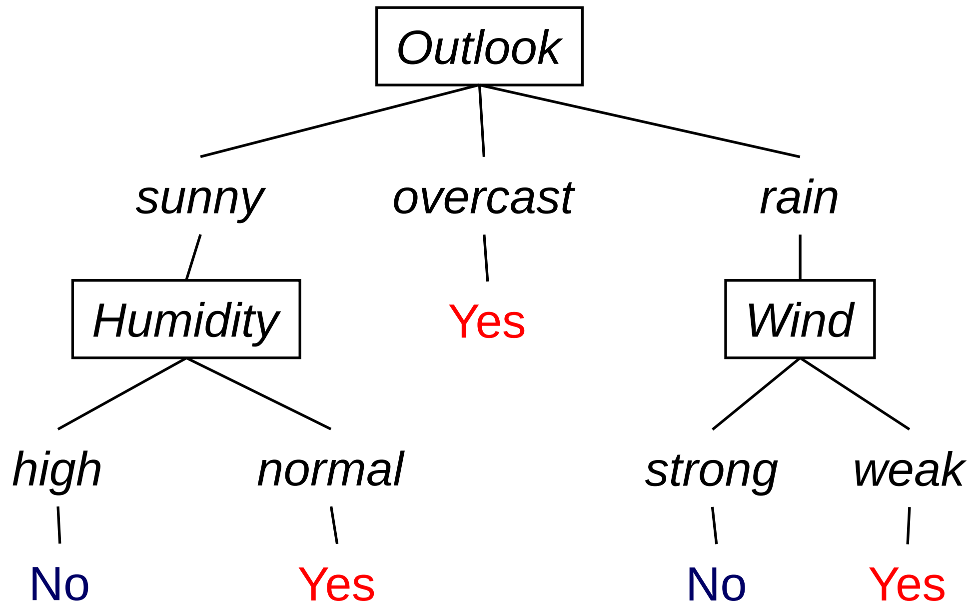
- (Sunny AND Normal) OR (Overcast) OR (Rain AND Weak)
- More powerful representation
- Larger hypothesis space H
- Can be represented as a tree
- Common form of decision making in humans



Decision Trees

- Decision tree to represent learned target functions
  - Each internal node <u>tests</u> an attribute
  - Each branch corresponds to <u>attribute value</u>
  - Each leaf node assigns a classification

- Can be represented by logical formulas



Representation in decision trees

- Example of representing rule in DT's:

  if outlook = sunny AND humidity = normal

  OR

  if outlook = overcast

  OR

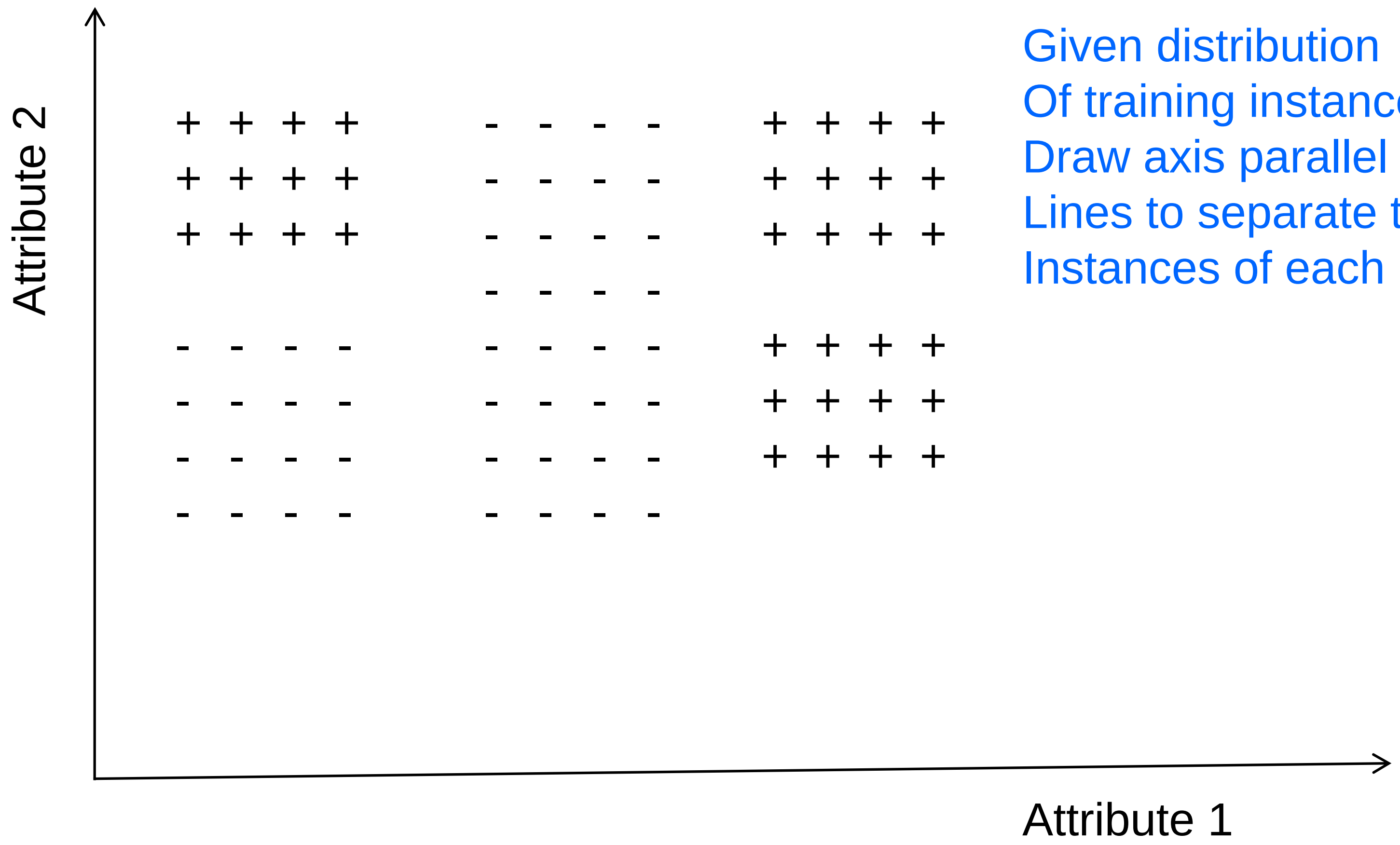
  if outlook = rain AND wind = weak

  then playtennis

Applications of Decision Trees

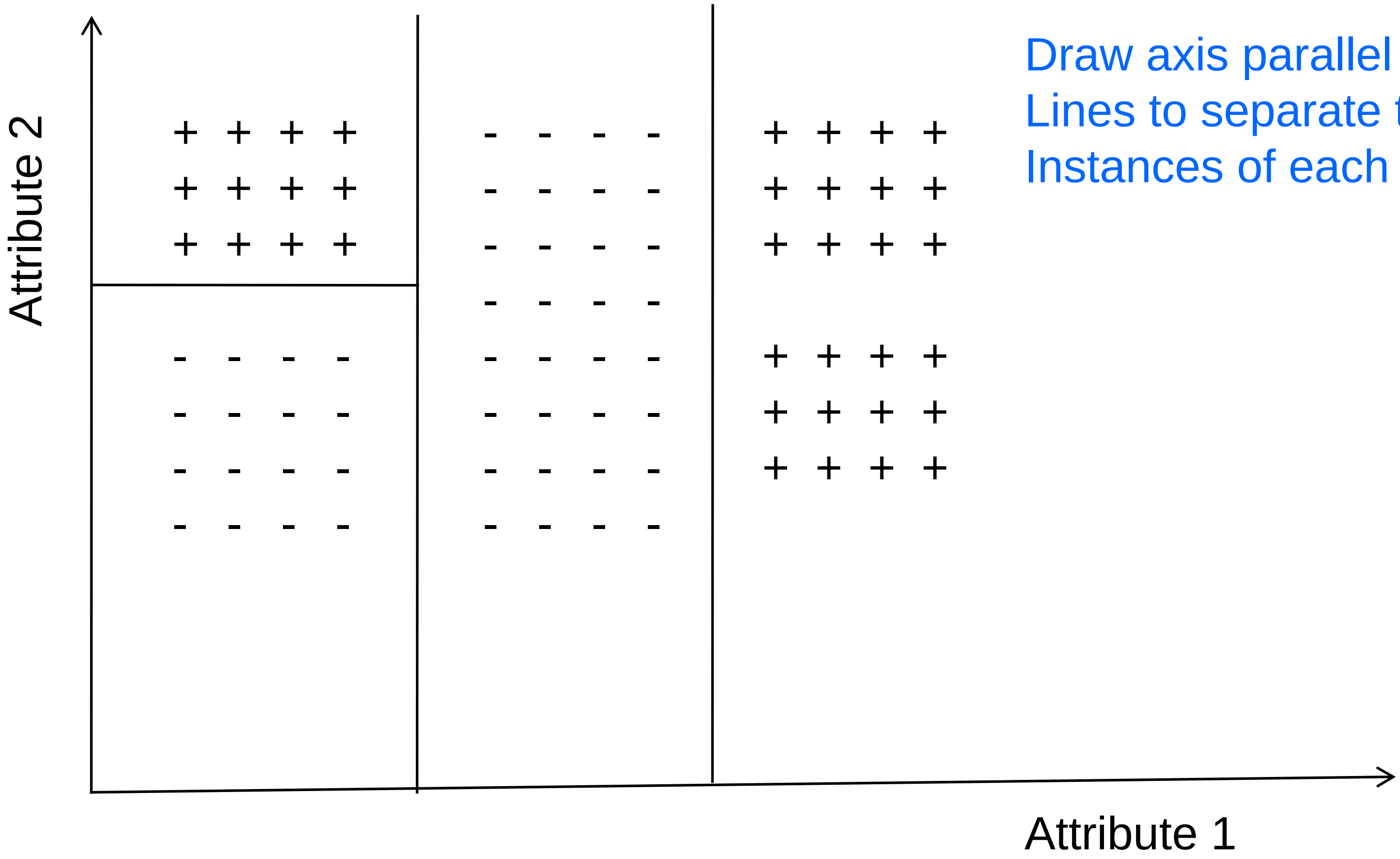
- Instances describable by a fixed set of attributes and their values
- Target function is discrete valued
  - 2-valued
  - N-valued
  - But can approximate continuous functions
- Disjunctive hypothesis space
- Possibly noisy training data
  - Errors, missing values, …
- Examples:
  - Equipment or medical diagnosis
  - Credit risk analysis
  - Calendar scheduling preferences

Decision Trees



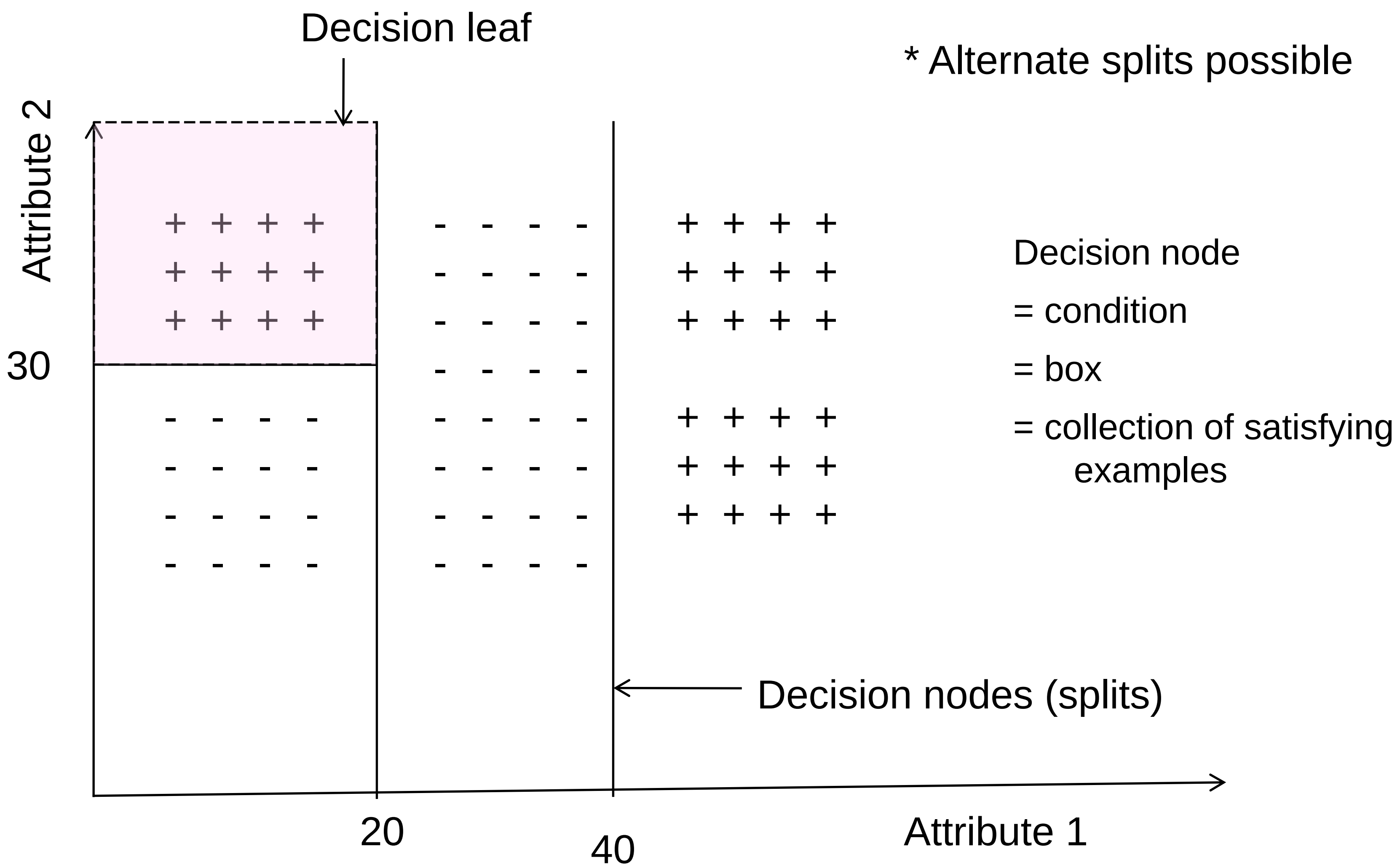
Given distribution
Of training instances
Draw axis parallel
Lines to separate the
Instances of each class

Decision Tree Structure



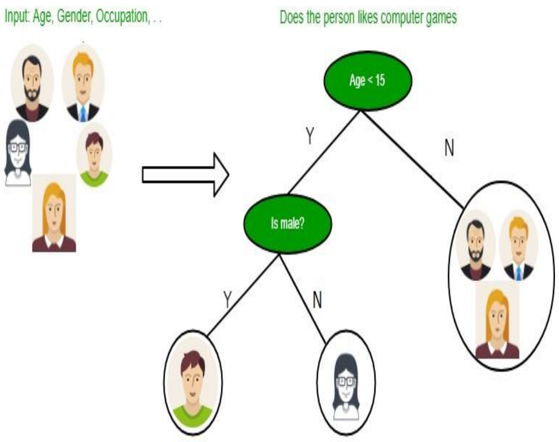
Draw axis parallel
Lines to separate the
Instances of each class

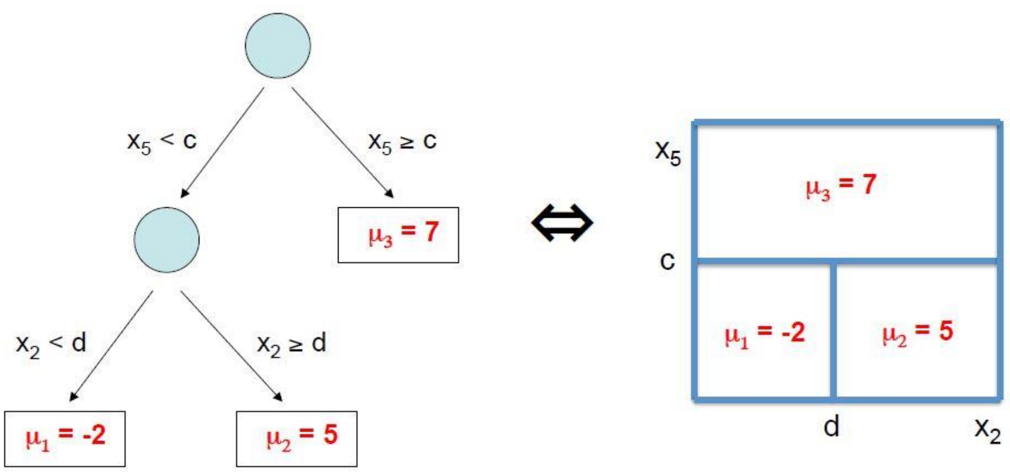
Decision Tree Structure



Decision Tree Construction

- Find the best structure
- Given a training data set

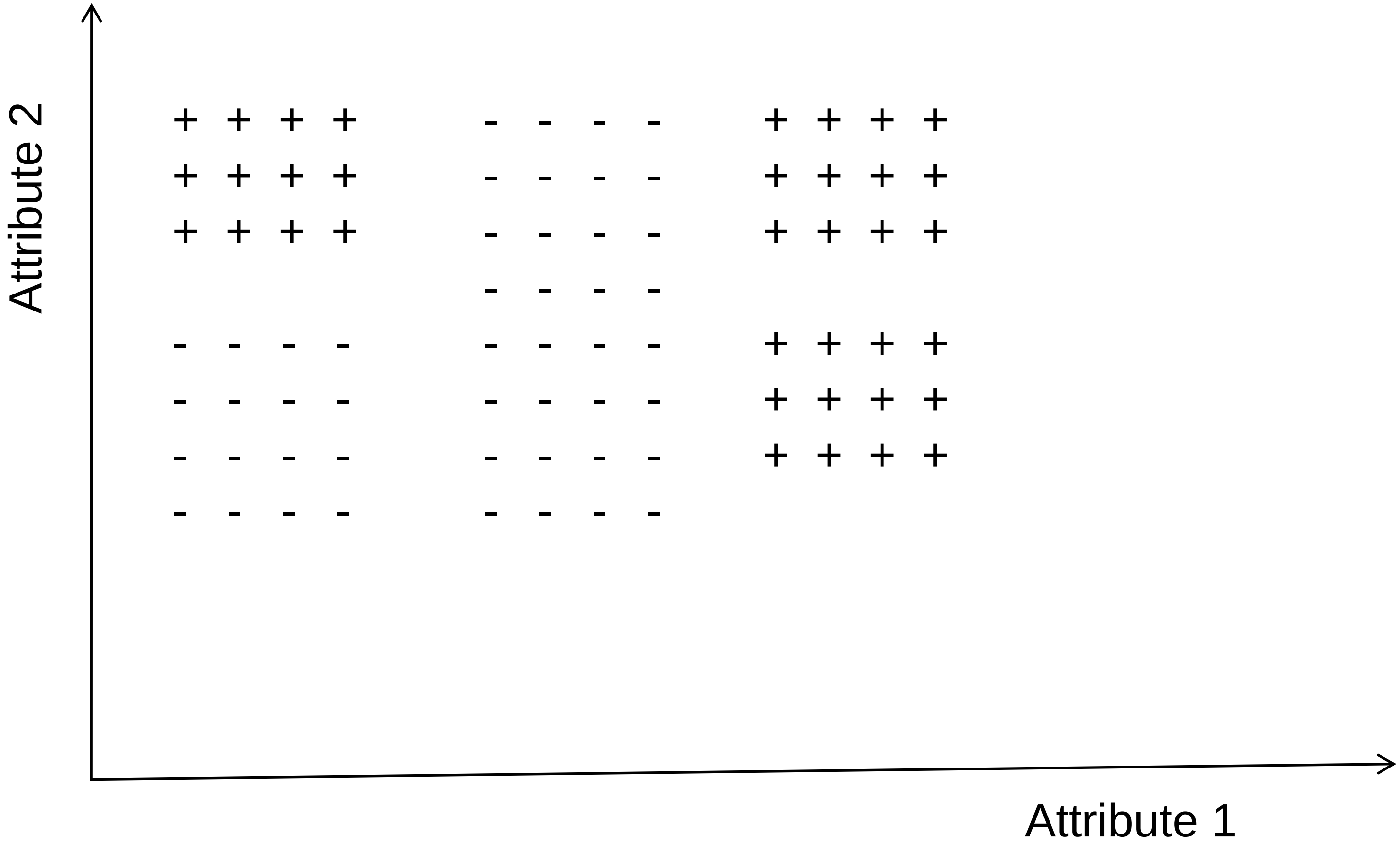




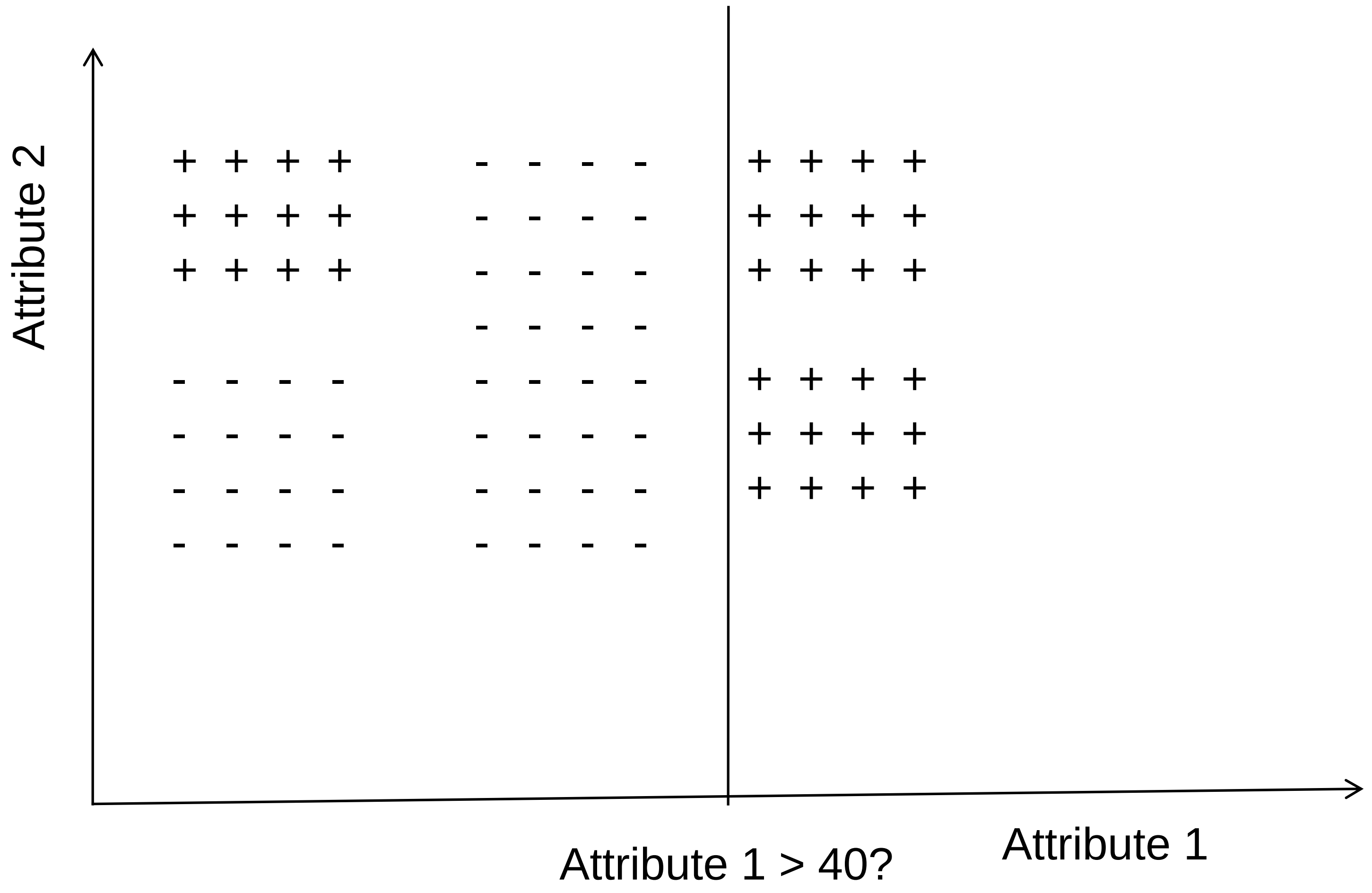
Top-Down Construction

- Start with empty tree
- Main loop:
  1. Split the "best" decision attribute (*A*) for next node
  2. Assign *A* as decision attribute for node
  3. For each value of *A*, create new descendant of node
  4. Sort training examples to leaf nodes
  5. If training examples perfectly classified, STOP,
     Else iterate over new leaf nodes
- Grow tree just deep enough for perfect classification
  - If possible (or can approximate at chosen depth)
- Which attribute is best?

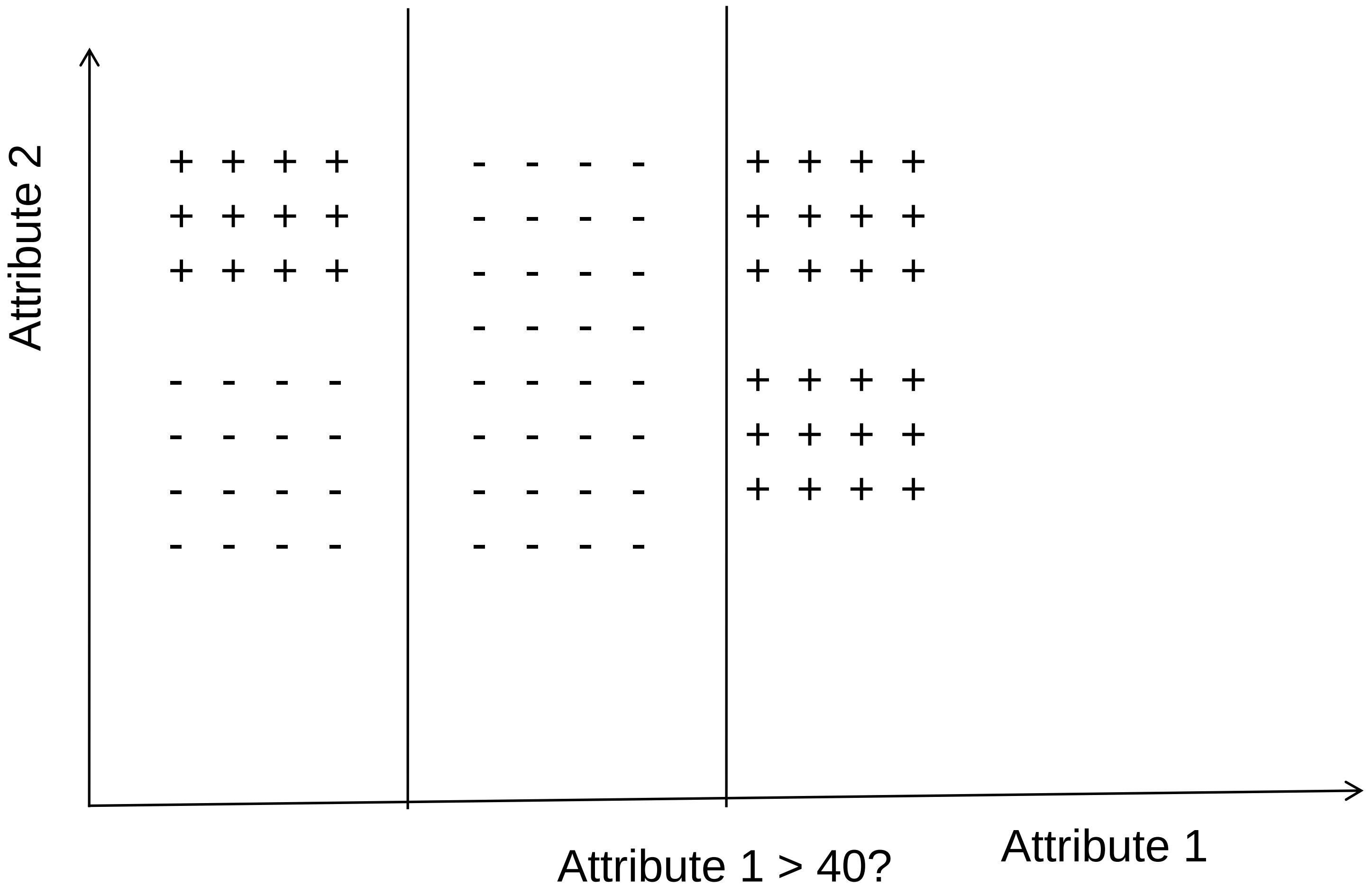
Best attribute to split?



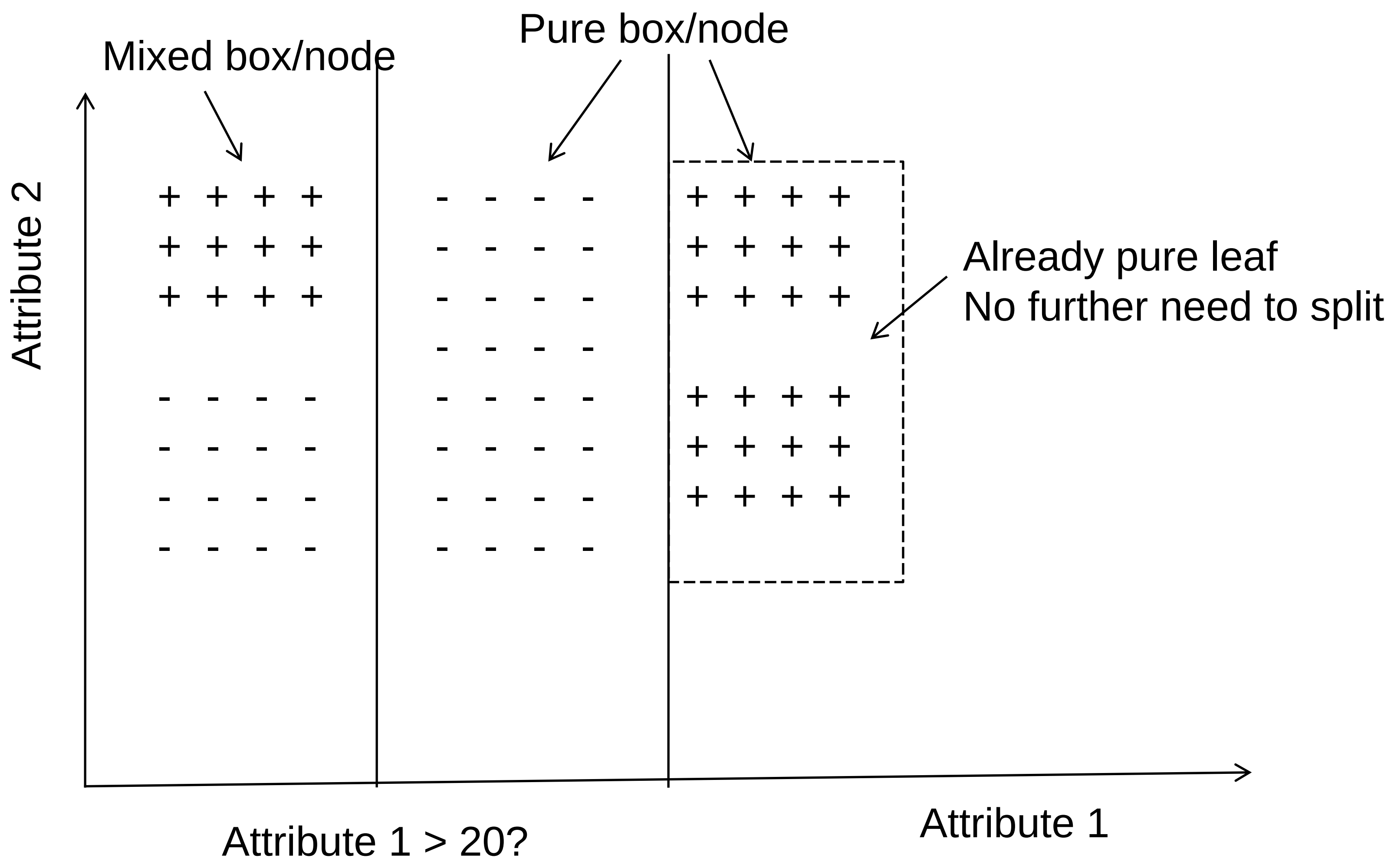
Best attribute to split?



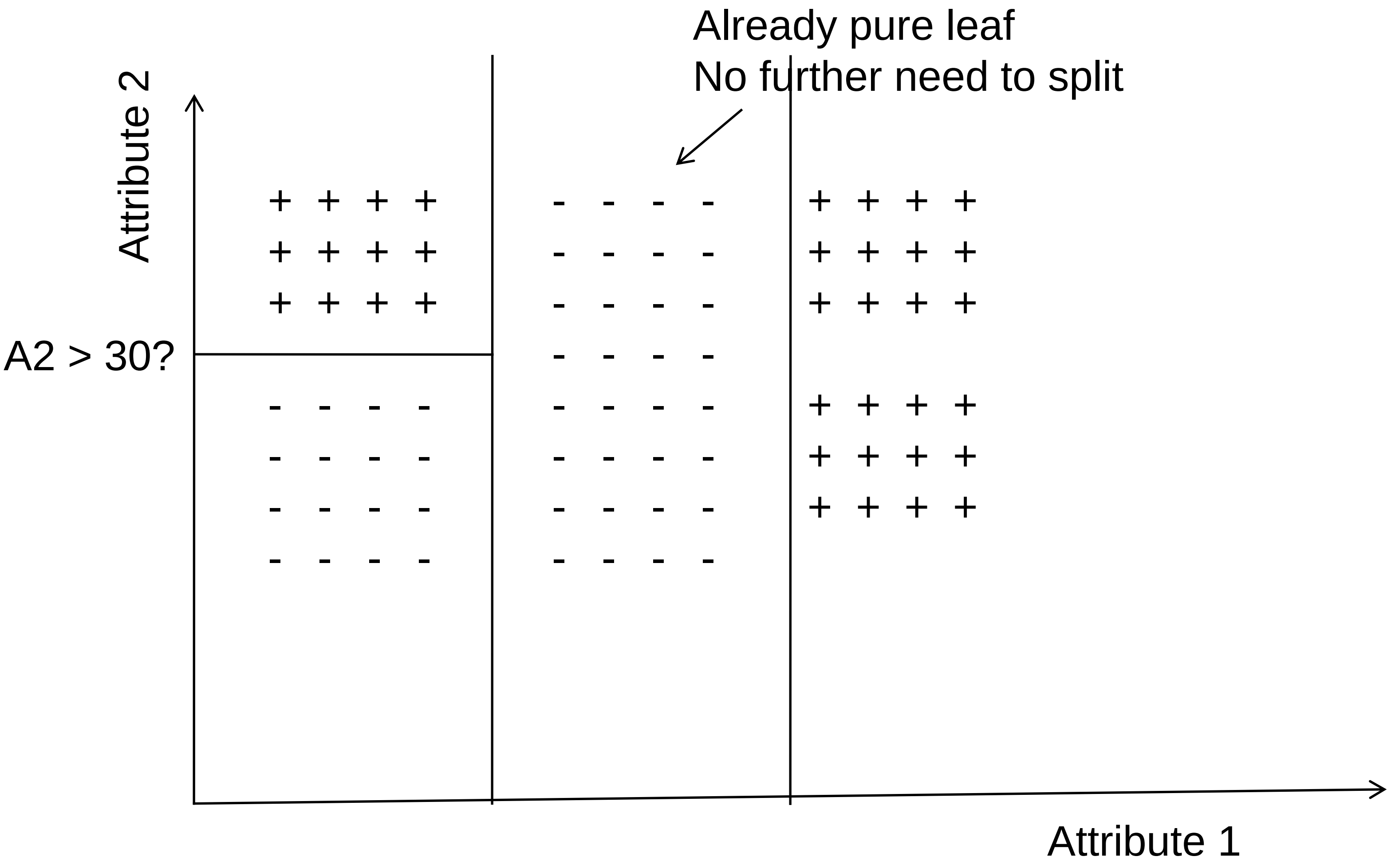
Best attribute to split?



Which split to make next?



Which split to make next?

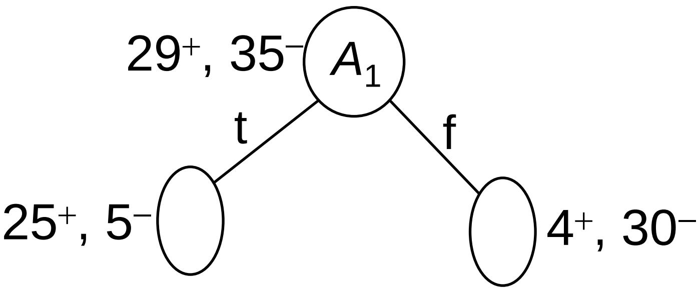


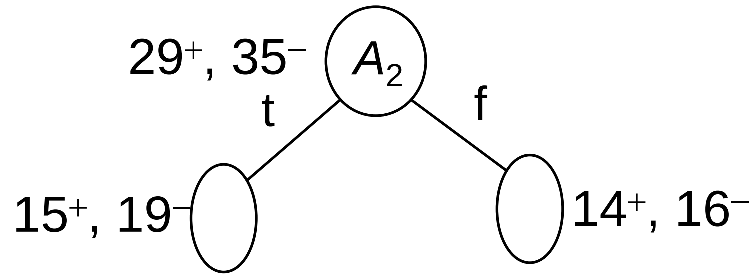
Principle of Decision Tree Construction

- Finally we want to form pure leaves
  - Correct classification
- Greedy approach to reach correct classification
  1. Initially treat the entire data set as a single box
  2. For each box choose the spilt that reduces its impurity (in terms of class labels) by the maximum amount
  3. Split the box having highest reduction in impurity
  4. Continue to Step 2
  5. Stop when all boxes are pure

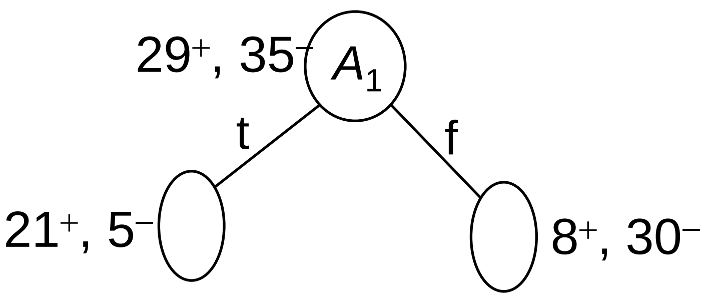
Choosing Best Attribute?

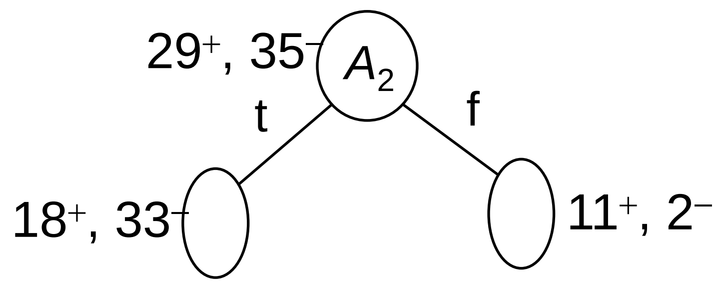
- Consider 64 examples, 29^+ and 35^-
- Which one is better?





- Which is better?





Entropy

- A measure for
  - uncertainty
  - purity
  - information content
- Information theory: optimal length code assigns $(-\log_2 p)$ bits to message having probability p
- S is a sample of training examples
  - p_+ is the proportion of positive examples in S
  - p_- is the proportion of negative examples in S
- Entropy of S: average optimal number of bits to encode information about certainty/uncertainty about S

$$Entropy(S) = p_+(-\log_2 p_+) + p_-(-\log_2 p_-) = -p_+\log_2 p_+ - p_-\log_2 p_-$$

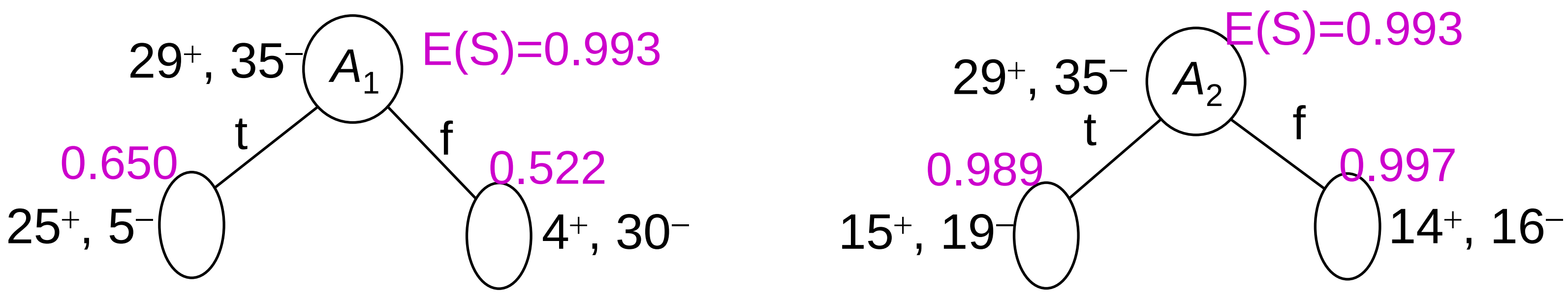
- Can be generalized to more than two values

Entropy

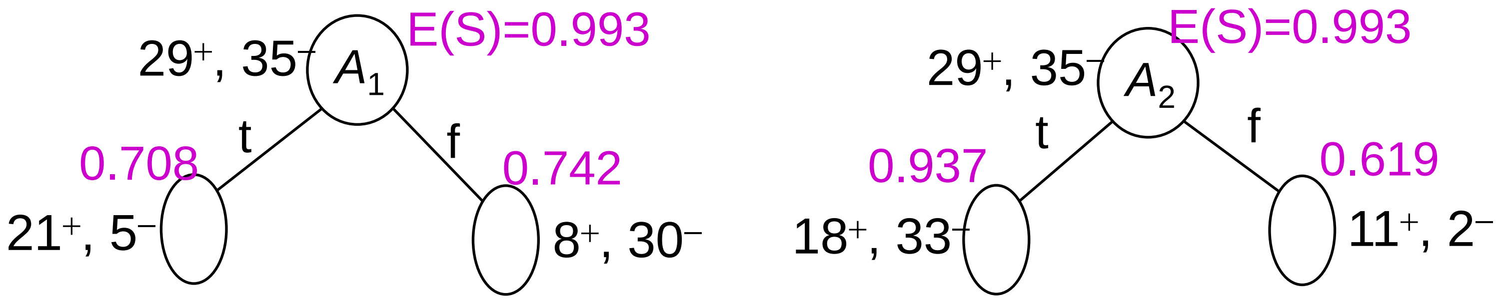
- Entropy can also be viewed as measuring
  - purity of S,
  - uncertainty in S,
  - information in S, …
- E.g.: values of entropy for p+=1, p+=0, p+=.5
- Easy generalization to more than binary values
  - Sum over pi *(-$\log_2$ pi) , i=1,n
    - i is + or – for binary
    - i varies from 1 to n in the general case

Choosing Best Attribute?

- Consider 64 examples ($29^+, 35^-$) and compute entropies:
- Which one is better?



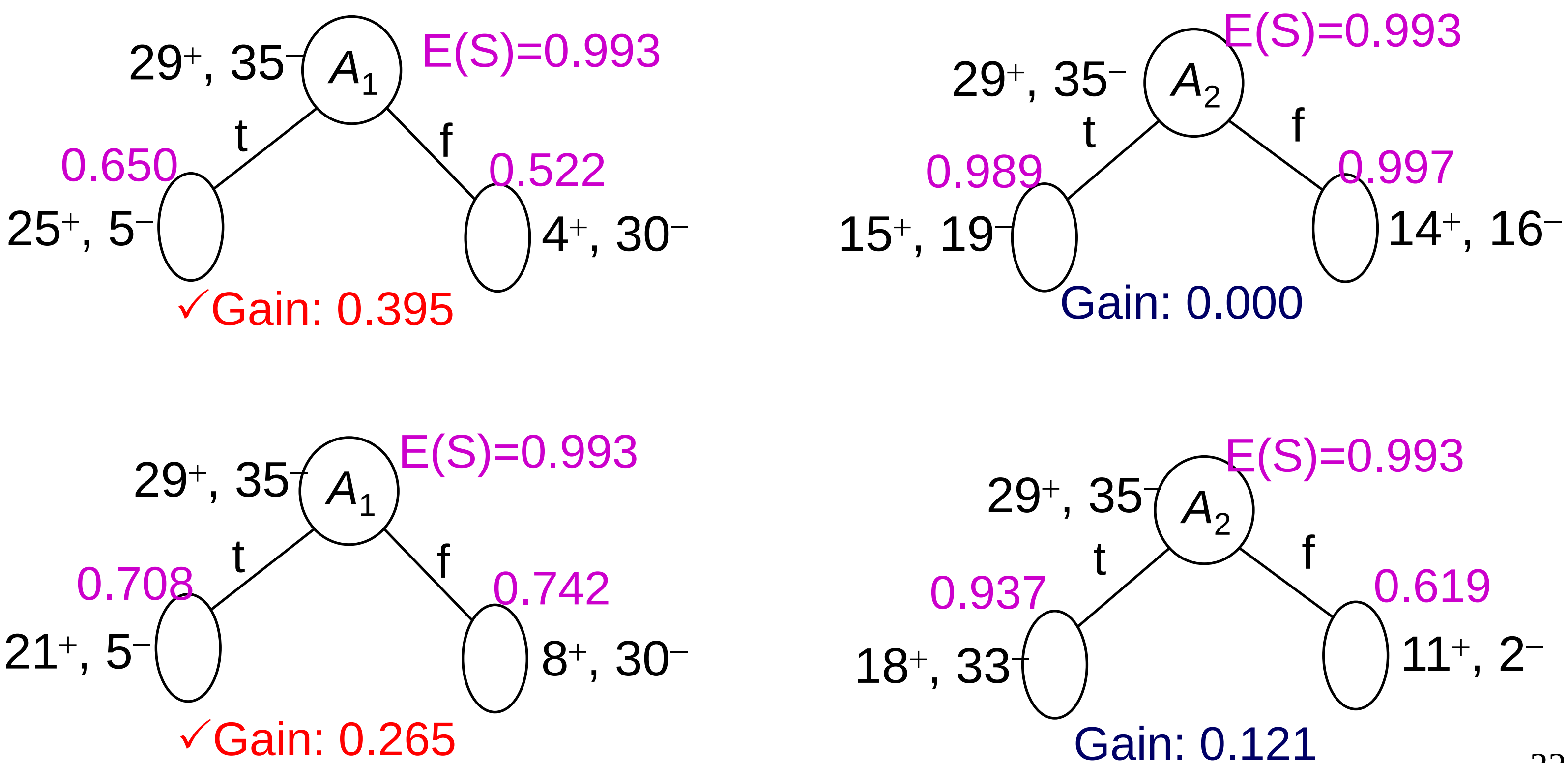
- Which is better?



Information Gain

- *Gain*(*S*,*A*): reduction in entropy after choosing attr. *A*

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



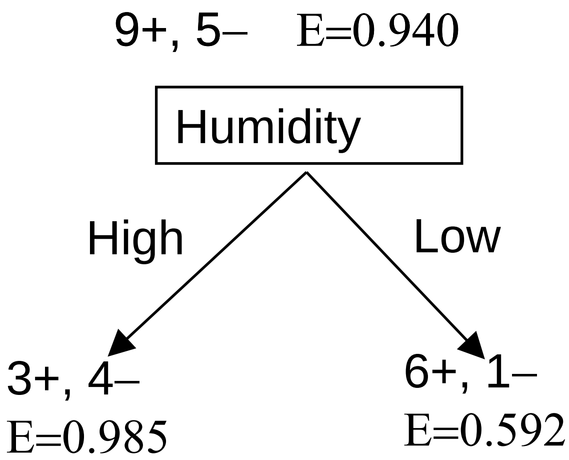
Gain function

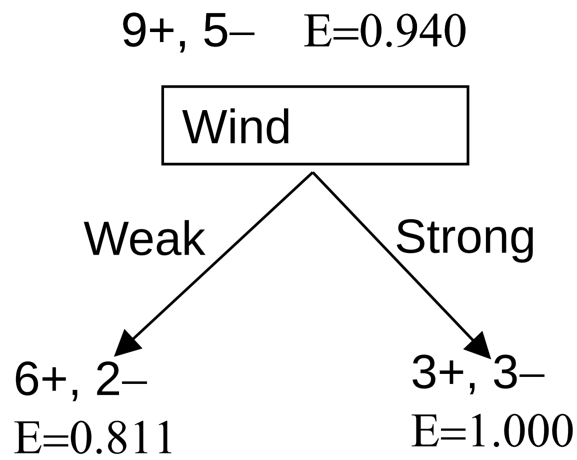
- Gain is measure of how much can
  - Reduce uncertainty
    - Value lies between 0,1
    - What is significance of
      - gain of 0?
        - example where have 50/50 split of +/- both before *and* after discriminating on attributes values
      - gain of 1?
        - Example of going from "perfect uncertainty" to perfect certainty after splitting example with predictive attribute
  - Find "patterns" in TE's relating to attribute values
    - Move to locally minimal representation of TE's

Training Examples

Day	Outlook	Temp	Humidity	Wind	Tennis?
D1	Sunny	Hot	High	Weak	*No*
D2	Sunny	Hot	High	Strong	*No*
D3	Overcast	Hot	High	Weak	*Yes*
D4	Rain	Mild	High	Weak	*Yes*
D5	Rain	Cool	Normal	Weak	*Yes*
D6	Rain	Cool	Normal	Strong	*No*
D7	Overcast	Cool	Normal	Strong	*Yes*
D8	Sunny	Mild	High	Weak	*No*
D9	Sunny	Cool	Normal	Weak	*Yes*
D10	Rain	Mild	Normal	Weak	*Yes*
D11	Sunny	Mild	Normal	Strong	*Yes*
D12	Overcast	Mild	High	Strong	*Yes*
D13	Overcast	Hot	Normal	Weak	*Yes*
D14	Rain	Mild	High	Strong	*No*

Determine the Root Attribute





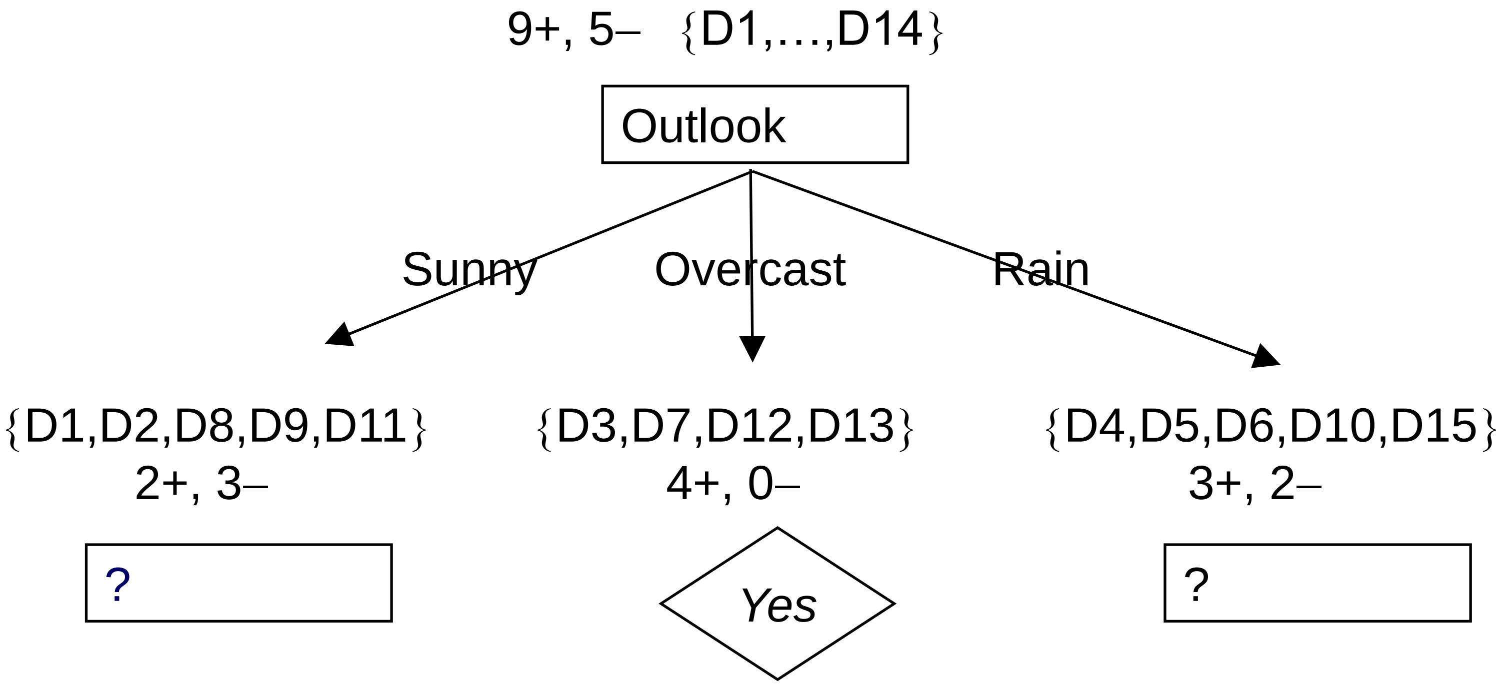
Gain (S, Humidity) = 0.151

Gain (S, Wind) = 0.048

Gain (S, Outlook) = 0.246

Gain (S, Temp) = 0.029

Sort the Training Examples



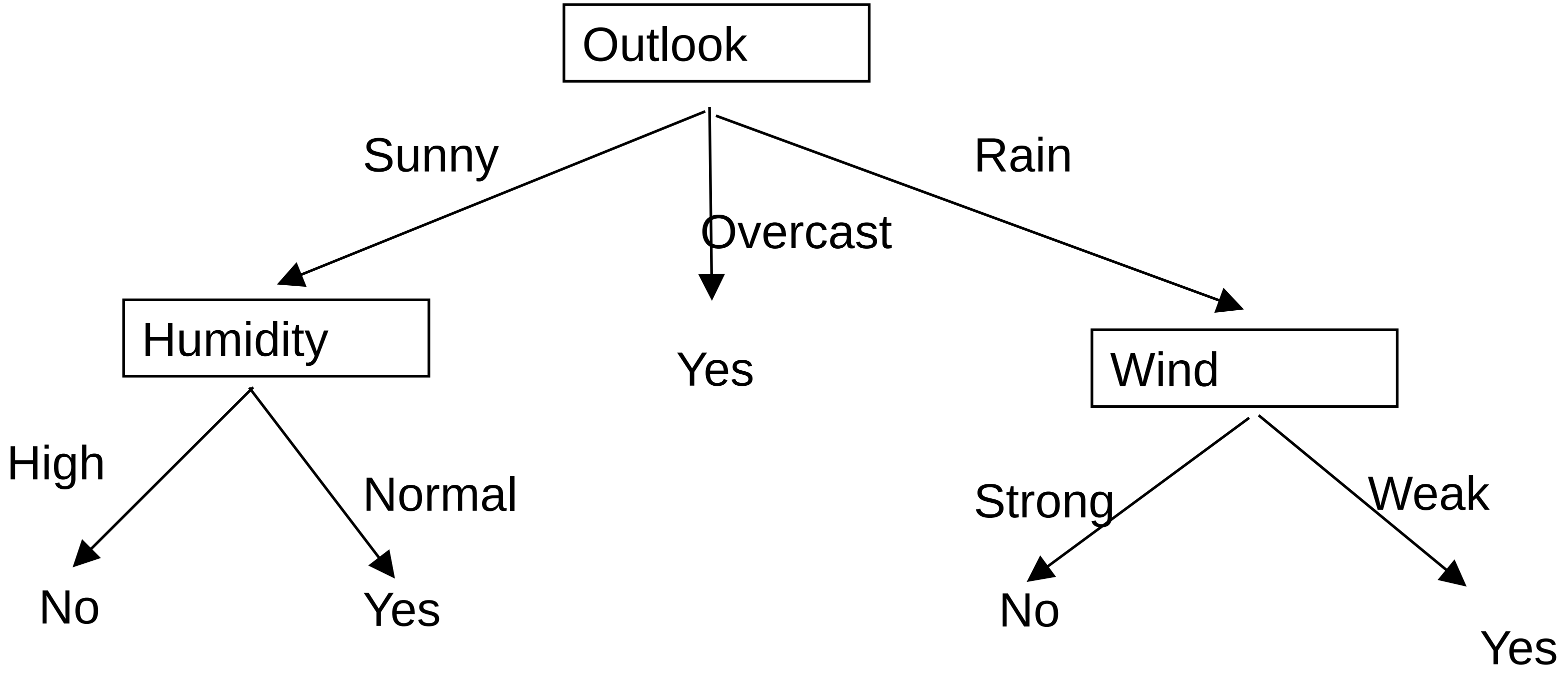
S_{sunny}= {D1,D2,D8,D9,D11}

Gain (S_{sunny}, Humidity) = .970

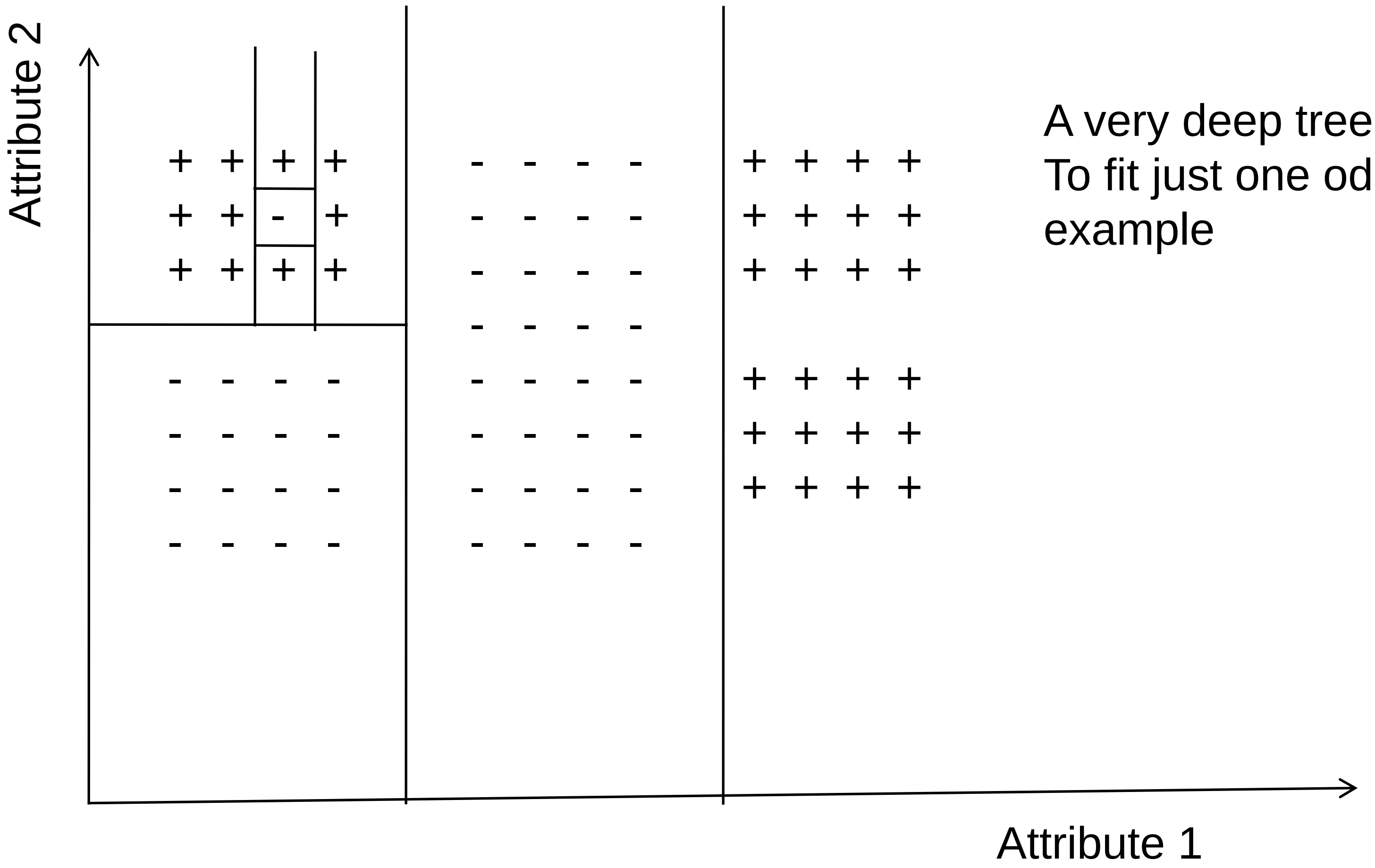
Gain (S_{sunny}, Temp) = .570

Gain (S_{sunny}, Wind) = .019

Final Decision Tree for Example



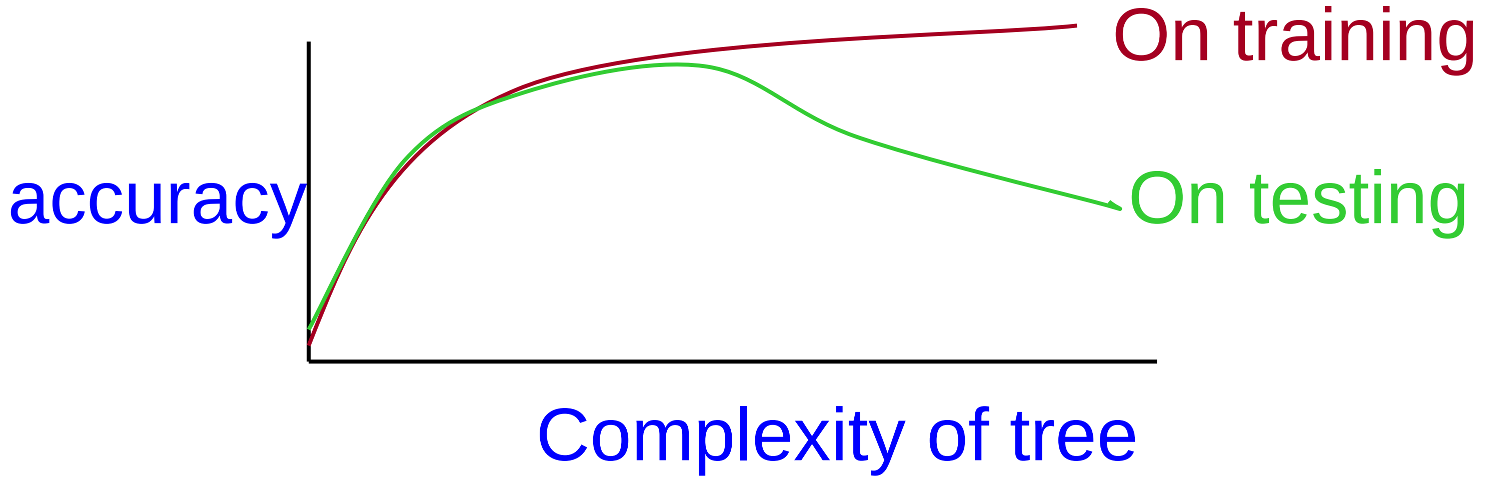
When to stop splitting further?



A very deep tree required
To fit just one odd training
example

Overfitting the Data

- Learning a tree that classifies the training data perfectly may not lead to the tree with the best generalization performance.
  - There may be noise in the training data the tree is fitting
  - The algorithm might be making decisions based on very little data
- A hypothesis *h* is said to overfit the training data if the is another hypothesis, h', such that *h* has smaller error than *h'* on the training data but *h* has larger error on the test data than *h'*.

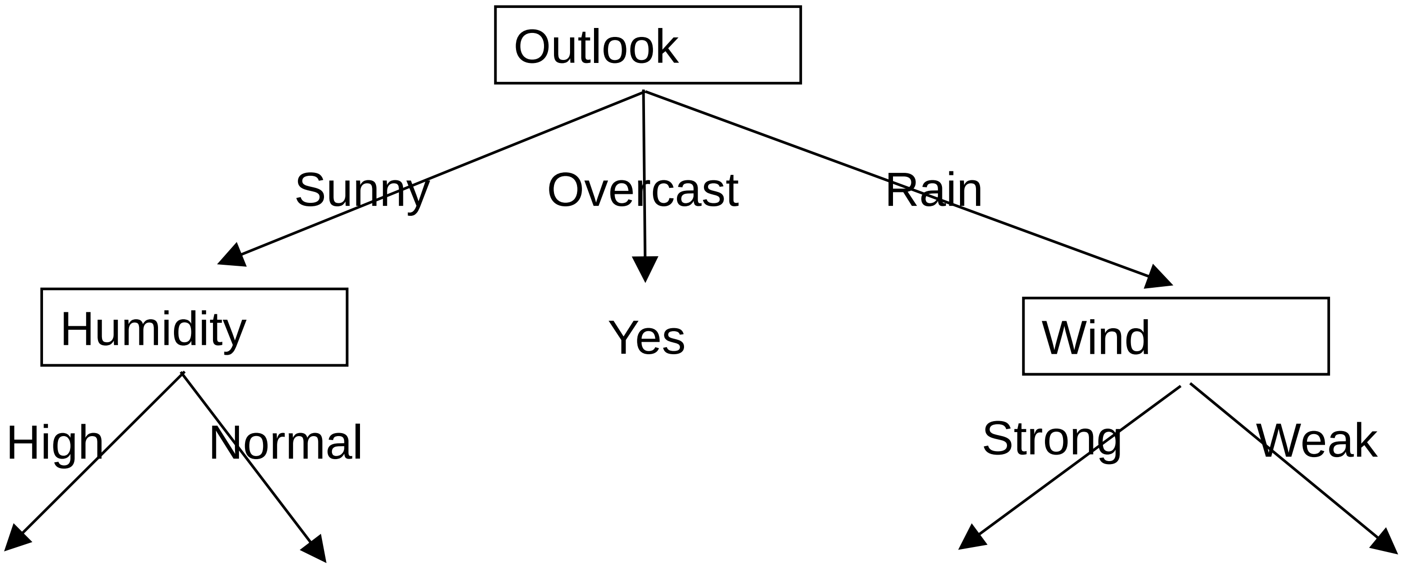


Overfitting in Decision Trees

- Consider adding *noisy* training example (should be +):

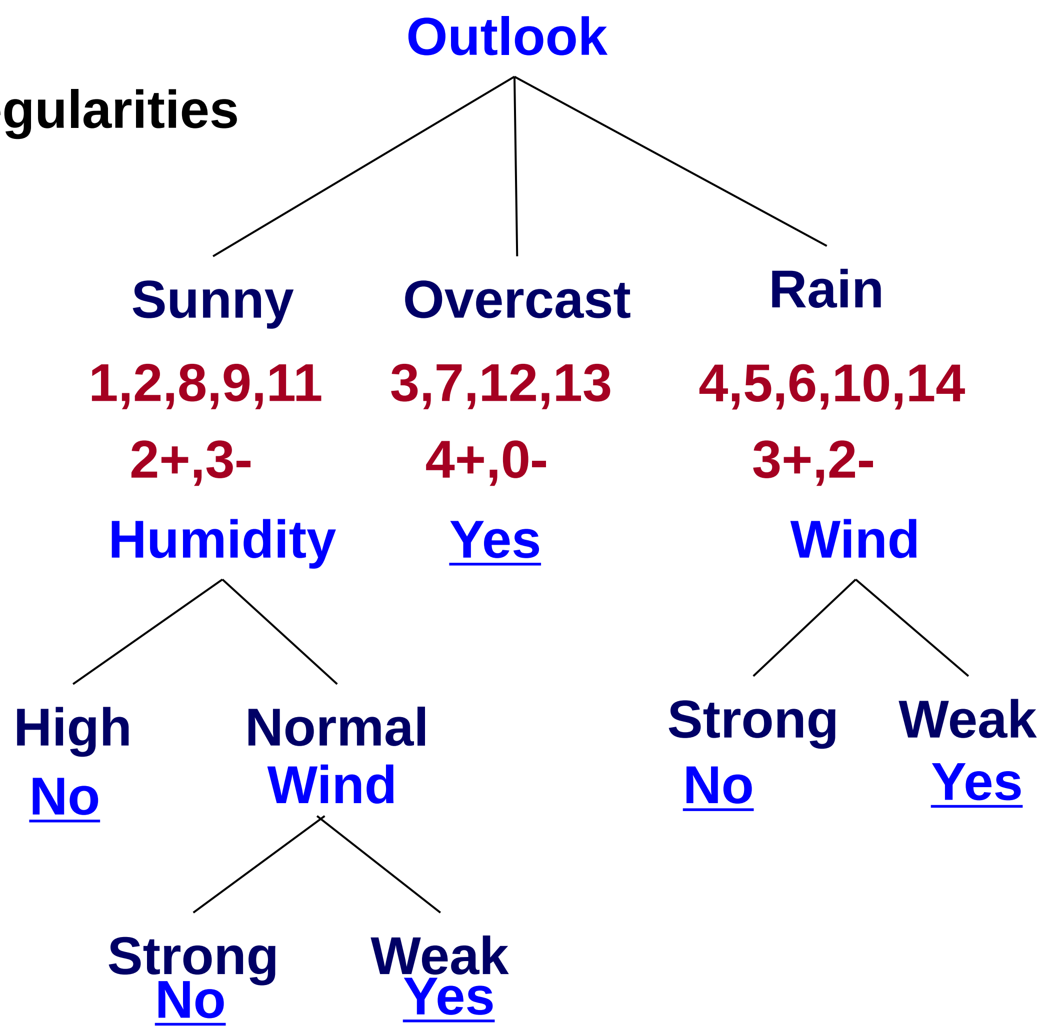
Day	Outlook	Temp	Humidity	Wind	Tennis?
D15	Sunny	Hot	Normal	Strong	*No*

- What effect on earlier tree?



Overfitting - Example

Noise or other coincidental regularities



Avoiding Overfitting

- Two basic approaches
  - Prepruning: Stop growing the tree at some point during construction when it is determined that there is not enough data to make reliable choices.
  - Postpruning: Grow the full tree and then remove nodes that seem not to have sufficient evidence. (more popular)
- Methods for evaluating subtrees to prune:
  - Cross-validation: Reserve hold-out set to evaluate utility (more popular)
  - Statistical testing: Test if the observed regularity can be dismissed as likely to be occur by chance
  - Minimum Description Length: Is the additional complexity of the hypothesis smaller than remembering the exceptions ?

    This is related to the notion of regularization that we will see in other contexts– keep the hypothesis simple.

Reduced-Error Pruning

- A post-pruning, cross validation approach
  - Partition training data into "grow" set and "validation" set.
  - Build a complete tree for the "grow" data
  - Until accuracy on validation set decreases, do:

    For each non-leaf node in the tree
    Temporarily prune the tree below; replace it by majority vote.
    Test the accuracy of the hypothesis on the validation set
    Permanently prune the node with the greatest increase
    in accuracy on the validation test.
- Problem: Uses less data to construct the tree
- Sometimes done at the rules level

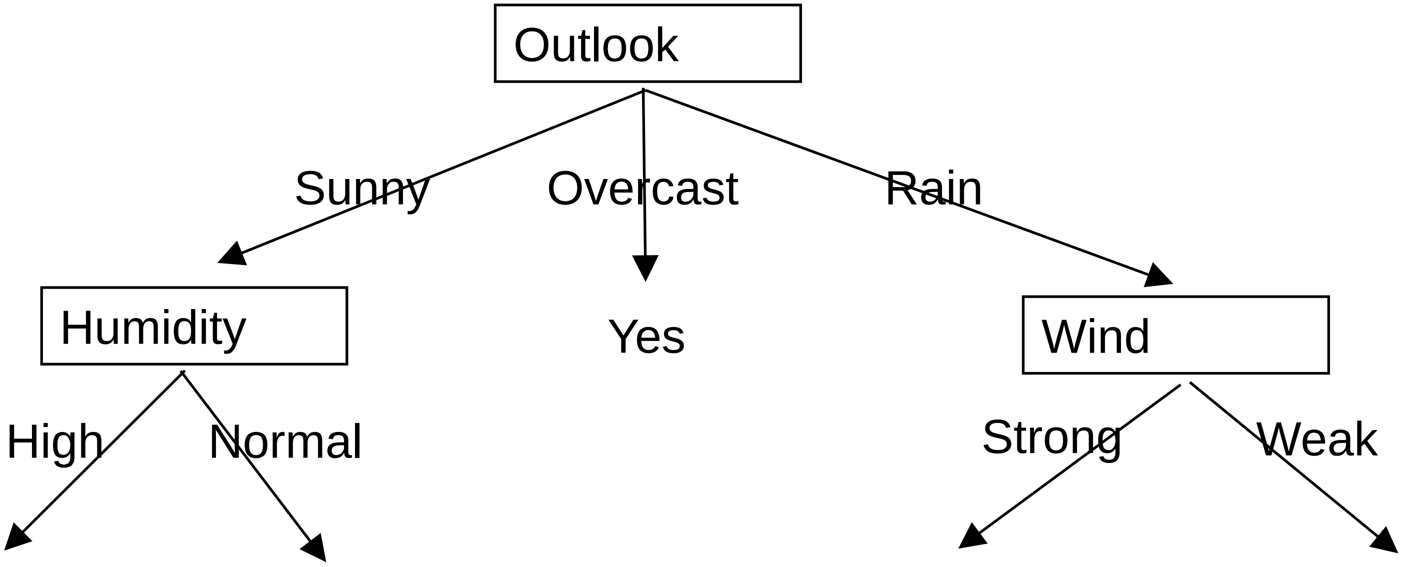
General Strategy: Overfit and Simplify

Rule post-pruning

- Allow tree to grow until best fit (allow overfitting)
- Convert tree to equivalent set of rules
  - One rule per leaf node
  - Prune each rule independently of others
    - Remove various preconditions to improve performance
  - Sort final rules into desired sequence for use

Example of rule post pruning

- IF (Outlook = Sunny) ^ (Humidity = High)
  - THEN PlayTennis = No
- IF (Outlook = Sunny) ^ (Humidity = Normal)
  - THEN PlayTennis = Yes



Extensions of basic algorithm

- Continuous valued attributes
- Attributes with many values
- TE's with missing data
- Attributes with associated costs
- Other impurity measures
- Regression tree

Continuous Valued Attributes

- Create a discrete attribute from continuous variables
  - E.g., define critical Temperature = 82.5
- Candidate thresholds
  - chosen by gain function
  - can have more than one threshold
  - typically where values change quickly

(48+60)/2 (80+90)/2

Temp	40	48	60	72	80	90
Tennis?	N	N	Y	Y	Y	N

Attributes with Many Values

- Problem:
  - If attribute has many values, *Gain* will select it (why?)
  - E.g. of birthdates attribute
    - 365 possible values
    - Likely to discriminate well on small sample
  - For sample of fixed size n, and attribute with N values, as N -> infinity
    - ni/N -> 0
    - - pi*log pi -> 0 for all i and entropy -> 0
    - Hence gain approaches max value

Attributes with many values

- Problem: *Gain* will select attribute with many values
- One approach: use *GainRatio* instead

$$GainRatio(S, A) = \frac{Gain(S, A)}{SplitInformation(S, A)}$$

Entropy of the partitioning

$$SplitInformation(S, A) = -\sum_{i=1}^{c} \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|}$$

Penalizes higher number of partitions

where S_i is the subset of S for which A has value v_i

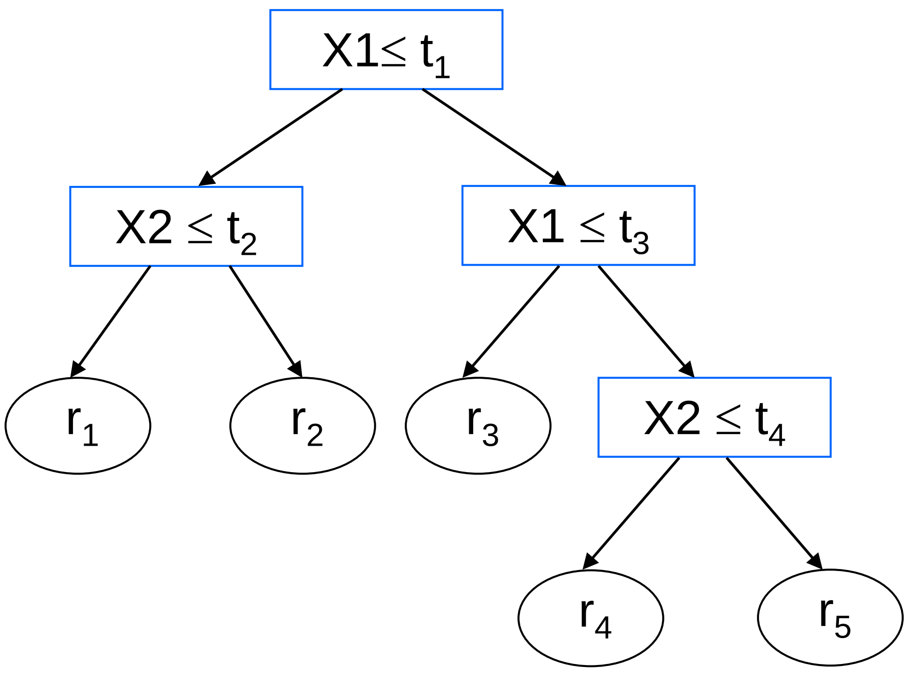
(example of Si/S = 1/N: SplitInformation = log N)

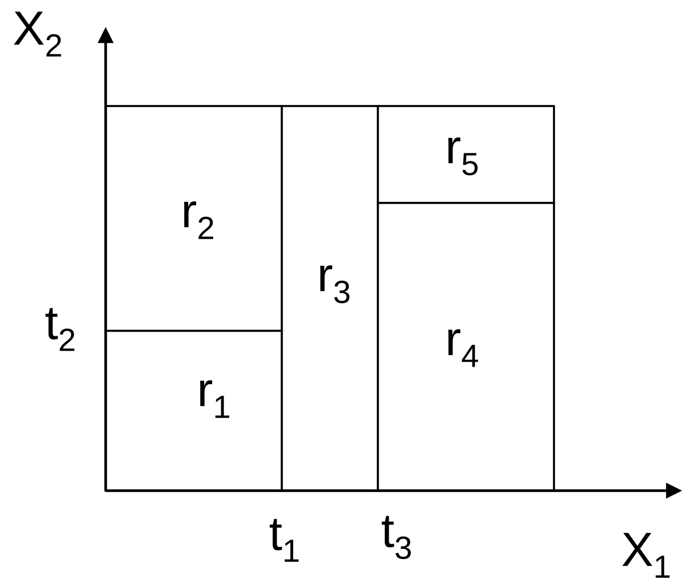
Regression Tree

- Similar to classification
- Use a set of attributes to predict the value (instead of a class label)
- Instead of computing information gain, compute the sum of squared errors
- Partition the attribute space into a set of rectangular subspaces, each with its own predictor
  - The simplest predictor is a constant value

Rectilinear Division

- A regression tree is a piecewise constant function of the input attributes





Growing Regression Trees

- To minimize the square error on the learning sample, the prediction at a leaf is the average output of the learning cases reaching that leaf
- Impurity of a sample is defined by the variance of the output in that sample:

$$I(LS)=\mathrm{var}_{y|LS}\{y\}=\mathrm{E}_{y|LS}\{(y-\mathrm{E}_{y|LS}\{y\})^2\}$$

- The best split is the one that reduces the most variance:

$$\Delta I(LS,A)=\mathrm{var}_{y|LS}\{y\}-\sum_a \frac{|LS_a|}{|LS|}\mathrm{var}_{y|LS_a}\{y\}$$

Regression Tree Pruning

- Exactly the same algorithms apply: pre-pruning and post-pruning.
- In post-pruning, the tree that minimizes the squared error on *VS* is selected.
- In practice, pruning is more important in regression because full trees are much more complex (often all objects have a different output values and hence the full tree has as many leaves as there are objects in the learning sample)

When Are Decision Trees Useful ?

- Advantages
  - Very fast: can handle very large datasets with many attributes
  - Flexible: several attribute types, classification and regression problems, missing values…
  - Interpretability: provide rules and attribute importance
- Disadvantages
  - Instability of the trees (high variance)
  - Not always competitive with other algorithms in terms of accuracy

History of Decision Tree Research

- Hunt and colleagues in Psychology used full search decision trees methods to model human concept learning in the 60's
- Quinlan developed ID3, with the information gain heuristics in the late 70's to learn expert systems from examples
- Breiman, Friedmans and colleagues in statistics developed CART (classification and regression trees simultaneously
- A variety of improvements in the 80's: coping with noise, continuous attributes, missing data, non-axis parallel etc.
- Quinlan's updated algorithm, C4.5 (1993) is commonly used (New:C5)
- Boosting (or Bagging) over DTs is a good general purpose algorithm

Summary

- Decision trees are practical for concept learning
- Basic information measure and gain function for best first search of space of DTs
- ID3 procedure
  - search space is complete
  - Preference for shorter trees
- Overfitting is an important issue with various solutions
- Many variations and extensions possible

Notes on Decision Tree

1. **Optimal Decision Tree:** Finding an optimal decision tree is an NP-complete problem. Hence, decision tree induction algorithms employ a heuristic based approach to search for the best in a large search space. Majority of the algorithms follow a greedy, top-down recursive divide-and-conquer strategy to build decision trees.

2. **Missing data and noise:** Decision tree induction algorithms are quite robust to the data set with missing values and presence of noise. However, proper data pre-processing can be followed to nullify these discrepancies.

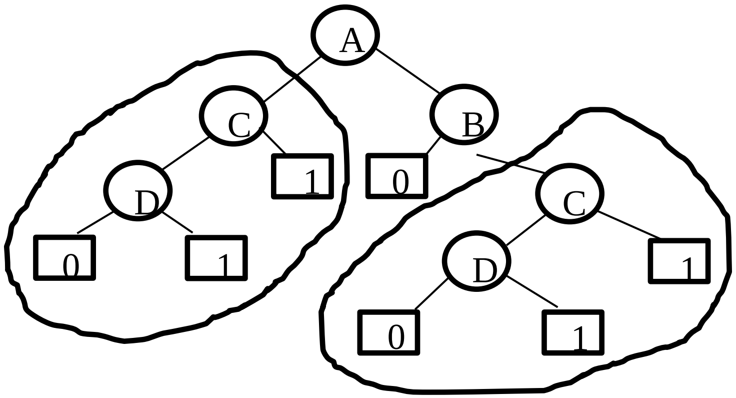
3. **Redundant Attributes:** The presence of redundant attributes does not adversely affect the accuracy of decision trees. It is observed that if an attribute is chosen for splitting, then another attribute which is redundant is unlikely to chosen for splitting.

4. **Computational complexity:** Decision tree induction algorithms are computationally inexpensive, in particular, when the sizes of training sets are large, Moreover, once a decision tree is known, classifying a test record is extremely fast, with a worst-case time complexity of O(d), where d is the maximum depth of the tree.

Notes on Decision Tree

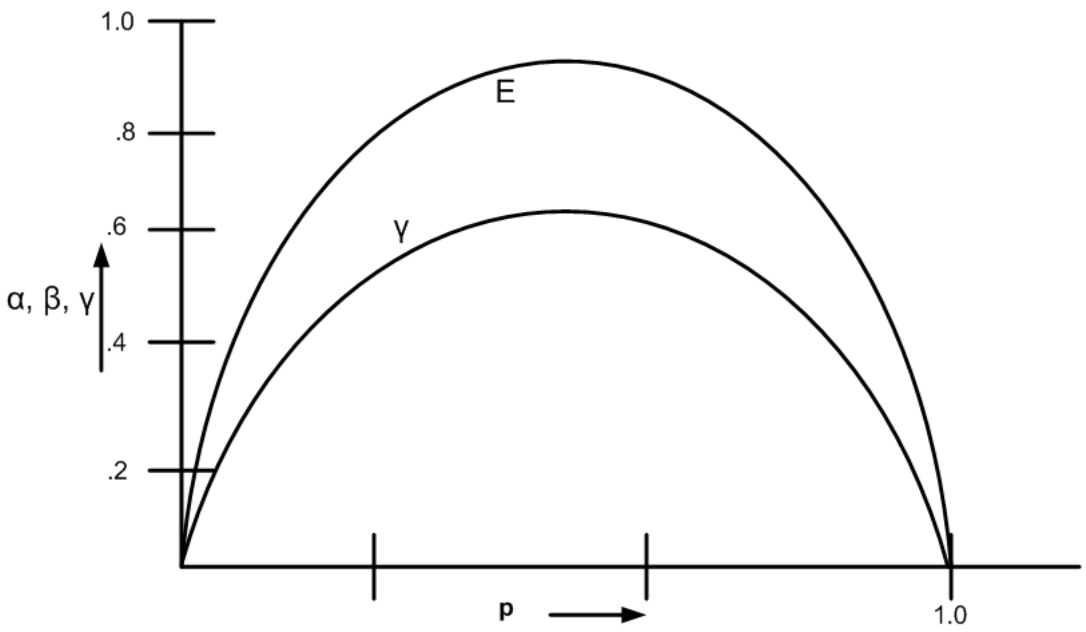
Data Fragmentation Problem: Since the decision tree induction algorithms employ a top-down, recursive partitioning approach, the number of tuples becomes smaller as we traverse down the tree. At a time, the number of tuples may be too small to make a decision about the class representation, such a problem is known as the data fragmentation. To deal with this problem, further splitting can be stopped when the number of records falls below a certain threshold.

Tree Pruning: A sub-tree can replicate two or more times in a decision tree (see figure below). This makes a decision tree unambiguous to classify a test record. To avoid such a sub-tree replication problem, all sub-trees except one can be pruned from the tree.



Notes on Decision Tree

Decision tree equivalence: The different splitting criteria followed in different decision tree induction algorithms have little effect on the performance of the algorithms. This is because the different heuristic measures (such as information gain (α), Gini index (γ) and Gain ratio (β) are quite consistent with each other); also see the figure below.



Software

- In R:
  - Packages tree and rpart
- C4.5:
  - http://www.cse.unwe.edu.au/~quinlan
- Weka
  - http://www.cs.waikato.ac.nz/ml/weka