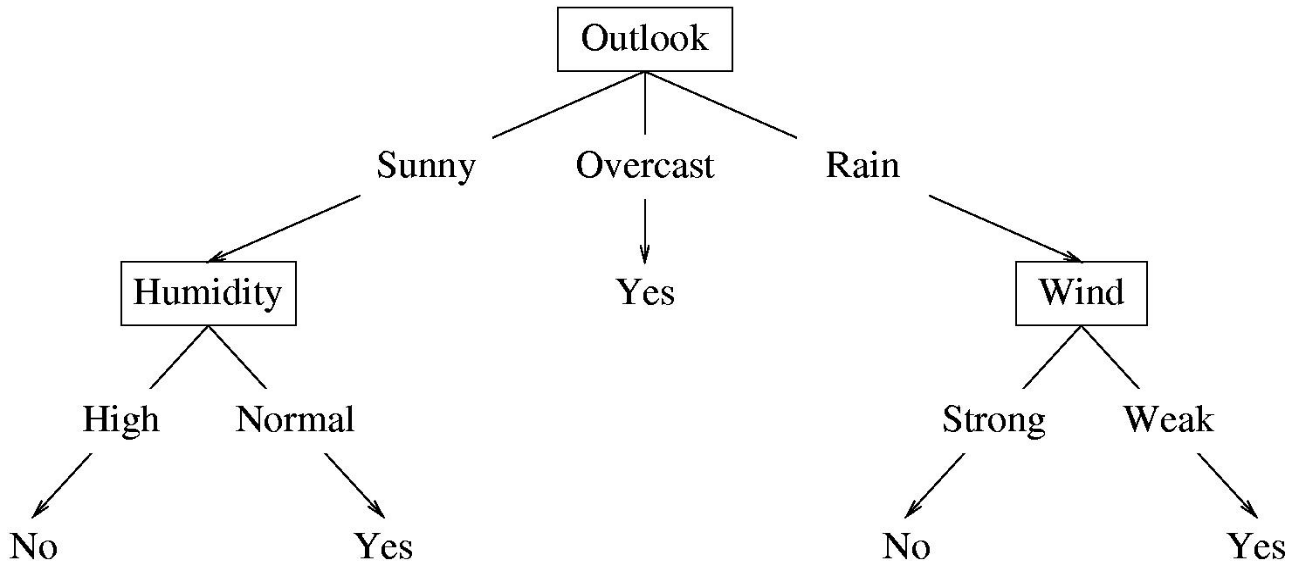
Decision Trees & Overfitting

Summary of Decision Trees (so far)



- Decision tree induction → choose the best attribute
  - Choose split via information gain
  - Build tree greedily, recursing on children of split
  - Stop when we achieve homogeny
    - i.e., when all instance in a child have the same class

Summary of Decision Trees (so far)

Information Gain: Mutual information of attribute A and the class variable of data set X

$$InfoGain(X, A) = H(X) - H(X \mid A)$$

$$= H(X) - \sum_{v \in values(A)} \underbrace{\frac{|\{x \in X \mid x_A = v\}|}{|X|}}_{\text{fraction of instances with value } v \text{ in attribute } A} \times \underbrace{H(\{x \in X \mid x_A = v\})}_{\text{entropy of those instances}}$$

Entropy:

$$H(X) = - \sum_{c \in Classes} \underbrace{\underbrace{\frac{|\{x \in X \mid class(x) = c\}|}{|X|}} \log_2 \underbrace{\frac{|\{x \in X \mid class(x) = c\}|}{|X|}}}_{\text{fraction of instances of class } c}$$

Attributes with Many Values

- Problem
  - If attribute has many values, InfoGain() will select it
  - e.g., imagine using **`date = Jan_28_2011`** as an attribute

- Alternative approach: use GainRatio() instead

$$GainRatio(X, A) = \frac{InfoGain(X, A)}{SplitInformation(X, A)}$$

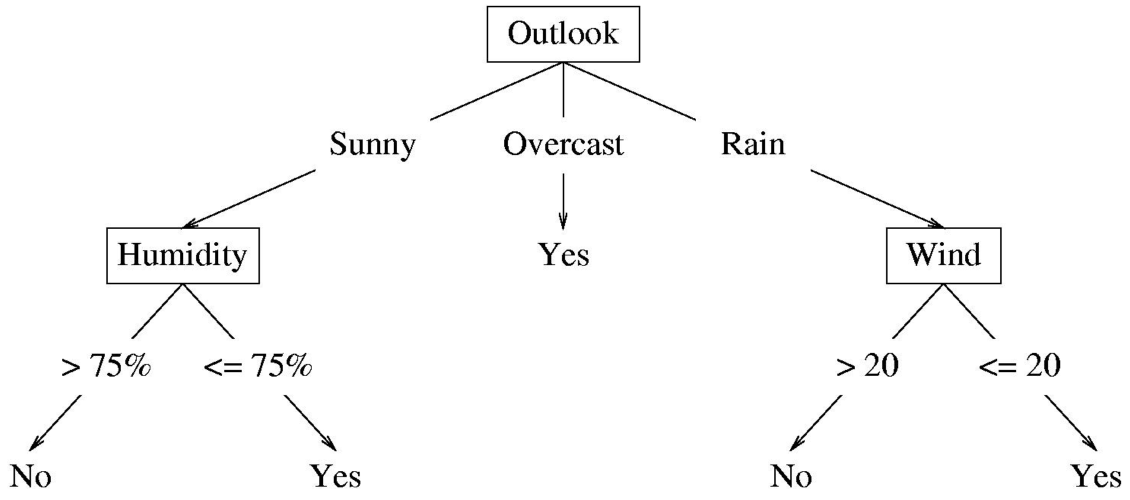
$$SplitInformation(X, A) = -\sum_{v \in values(A)} \frac{|X_v|}{|X|} \log_2 \frac{|X_v|}{|X|}$$

where X_v is a subset of X for which A has value v

Extensions of ID3

- Using gain ratios
- Real-valued data
- Noisy data and overfitting
- Generation of rules
- Setting parameters
- Cross-validation for experimental validation of performance
- C4.5 is an extension of ID3 that accounts for unavailable values, continuous attribute value ranges, pruning of decision trees, rule derivation, and so on

Real-Valued Features



- Change to binary splits by choosing a threshold
- One method:
  - Sort instances by value, identify adjacencies with different classes

Temperature:	40	48	60	72	80	90
PlayTennis:	No	No	Yes	Yes	Yes	No

candidate splits

  - Choose among splits by InfoGain()

Unknown Attribute Values

What if some examples are missing values of *A*?

Use training example anyway, sort through tree:

- If node *n* tests *A*, assign most common value of *A* among other examples sorted to node *n*
- Assign most common value of *A* among other examples with same class label
- Assign probability p_i to each possible value v_i of *A.*

  Assign fraction p_i of example to each descendent of tree

Classify new examples in same fashion

Noisy Data

- Many kinds of "noise" can occur in the examples:
  - Two examples have same attribute/value pairs, but different classifications
  - Some values of attributes are incorrect because of errors in the data acquisition process or the preprocessing phase
  - The instance was labeled incorrectly (+ instead of -)

- Also, some attributes are irrelevant to the decision-making process
  - e.g., color of a die is irrelevant to its outcome

Based on Slide from M. desJardins & T. Finin

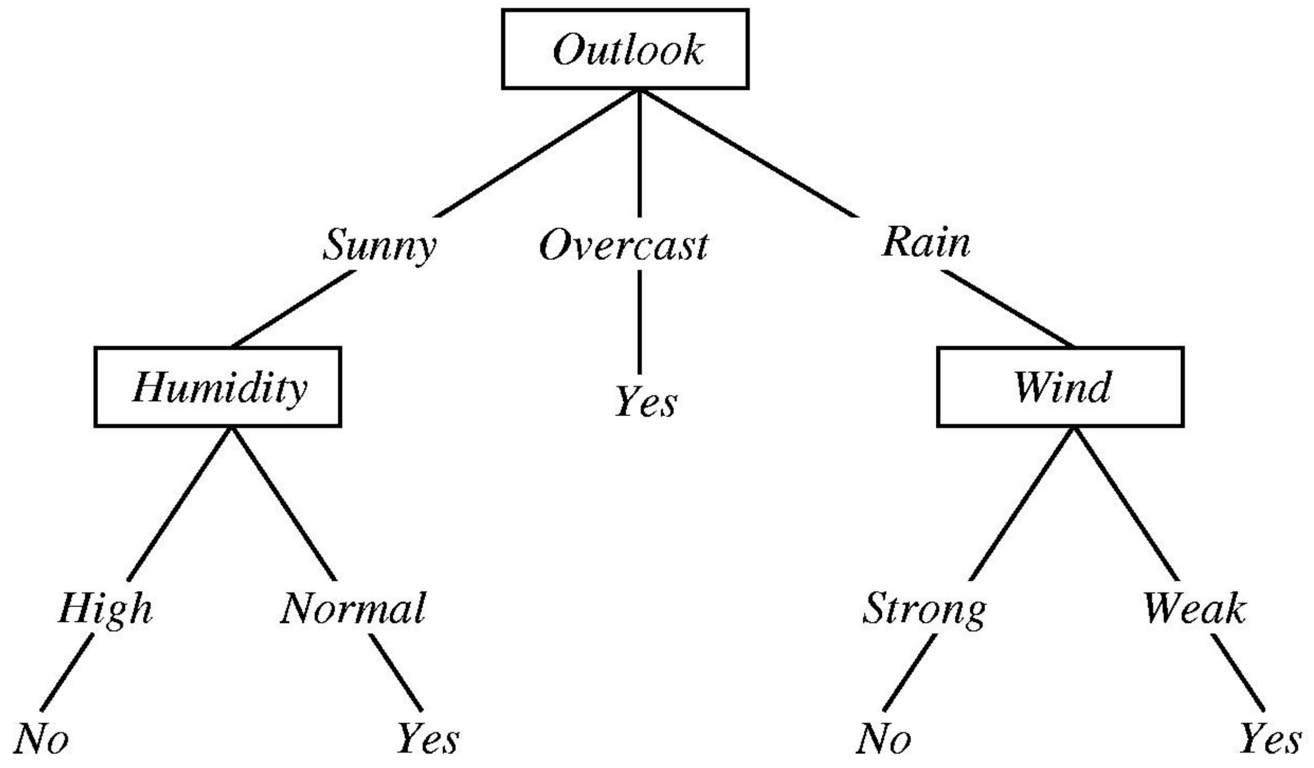
Overfitting

- Irrelevant attributes can result in *overfitting* the training example data
  - If hypothesis space has many dimensions (large number of attributes), we may find **meaningless regularity** in the data that is irrelevant to the true, important, distinguishing features

- If we have too little training data, even a reasonable hypothesis space will 'overfit'

Overfitting in Decision Trees

Consider adding a noisy training example to the following tree:



What would be the effect of adding:

<outlook=sunny, temperature=hot, humidity=normal, wind=strong, playTennis=No> ?

Based on Slide by Pedro Domingos

Overfitting

Consider error of hypothesis h over

- training data: $error_{train}(h)$
- entire distribution $\mathcal{D}$ of data: $error_{\mathcal{D}}(h)$

Hypothesis $h \in H$ **overfits** training data if there is an alternative hypothesis $h' \in H$ such that

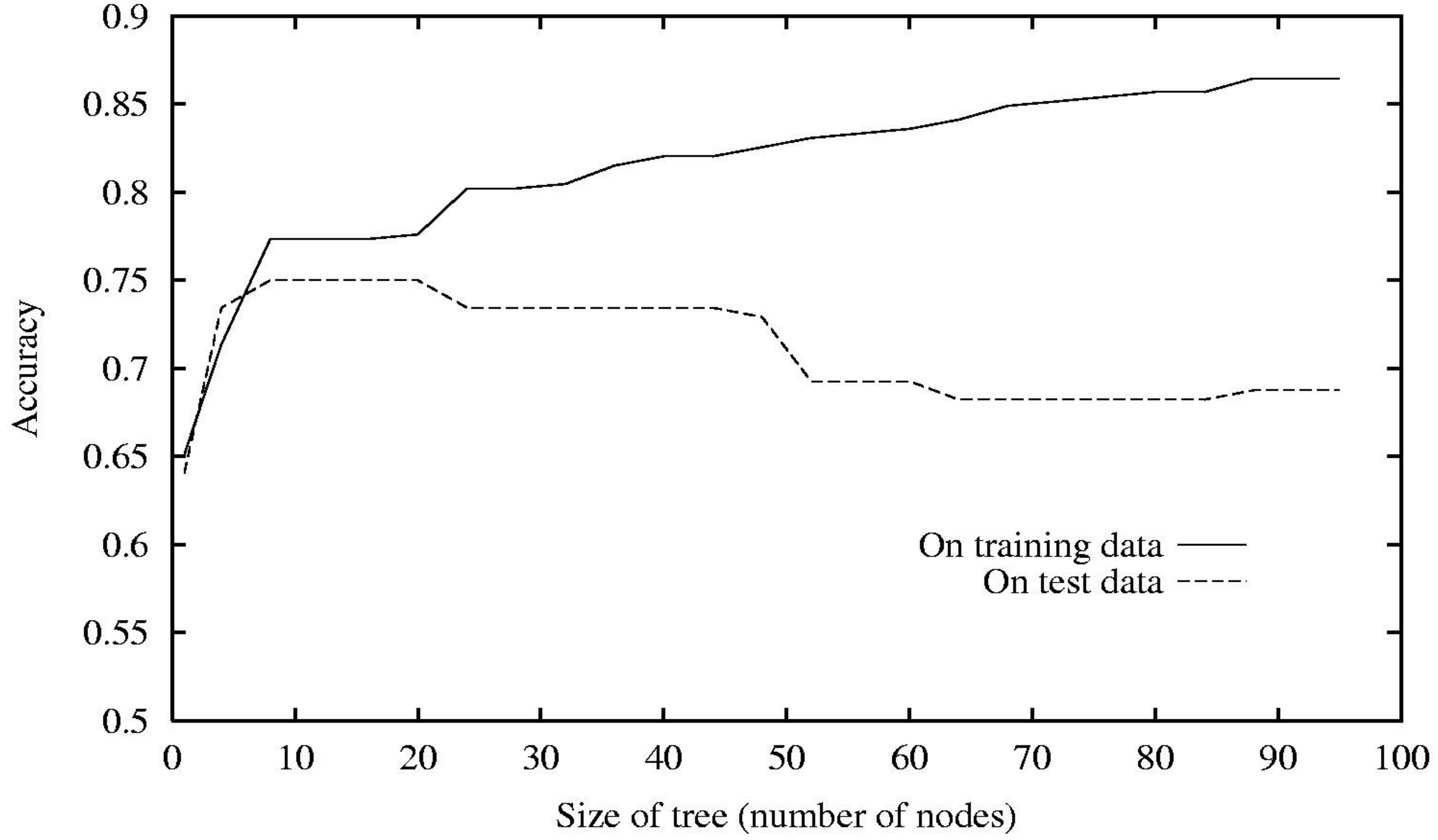
$$error_{train}(h) < error_{train}(h')$$

and

$$error_{\mathcal{D}}(h) > error_{\mathcal{D}}(h')$$

Slide by Pedro Domingos

Overfitting in Decision Tree Learning



Slide by Pedro Domingos

Avoiding Overfitting

How can we avoid overfitting?

- Stop growing when data split is not statistically significant
- Acquire more training data
- Remove irrelevant attributes (manual process – not always possible)
- Grow full tree, then post-prune

How to select "best" tree:

- Measure performance over training data
- Measure performance over separate validation data set
- Add complexity penalty to performance measure

Reduced-Error Pruning

Split data into *training* and *validation* sets

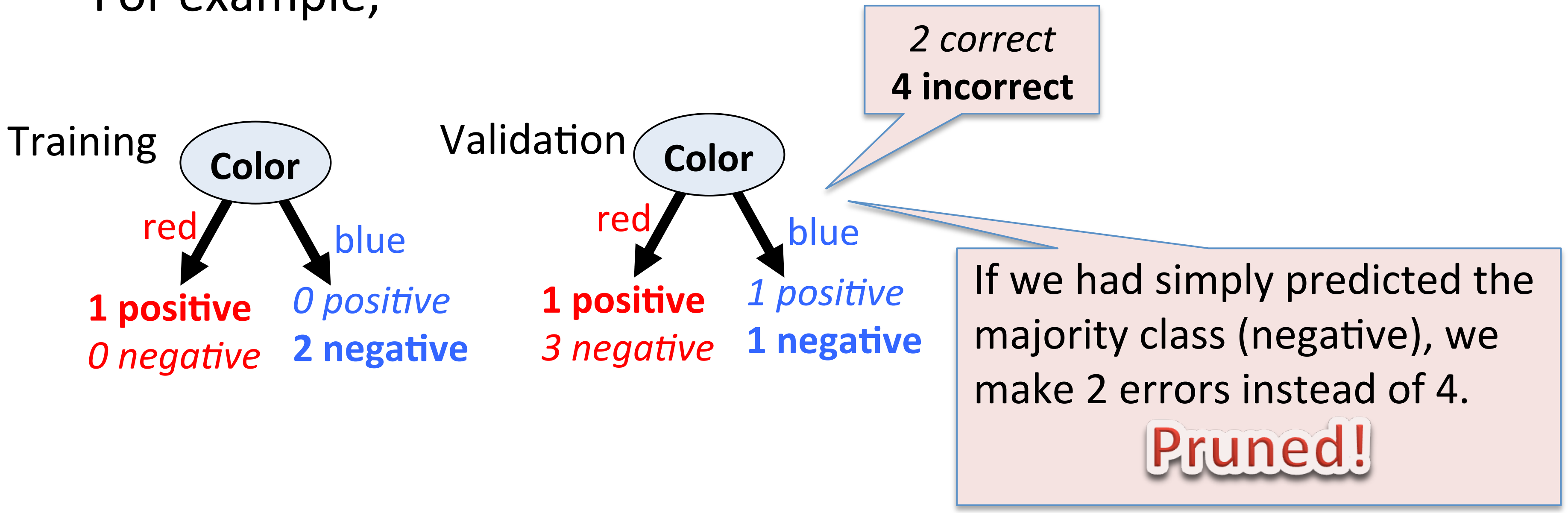
Grow tree based on *training set*

Do until further pruning is harmful:

1. Evaluate impact on validation set of pruning each possible node (plus those below it)
2. Greedily remove the node that most improves *validation set* accuracy

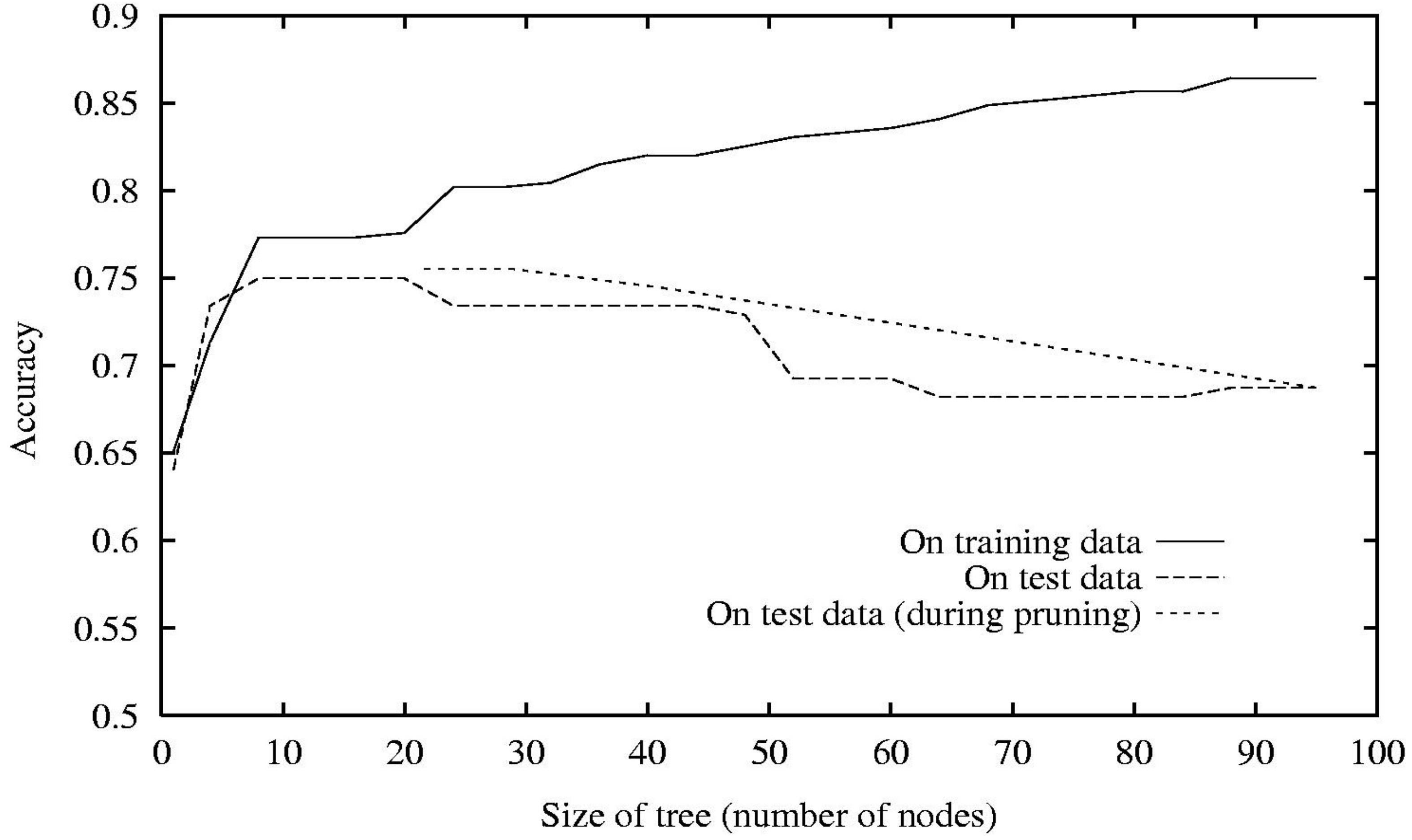
Pruning Decision Trees

- Pruning of the decision tree is done by replacing a whole subtree by a leaf node.
- The replacement takes place if a decision rule establishes that the expected error rate in the subtree is greater than in the single leaf.
- For example,



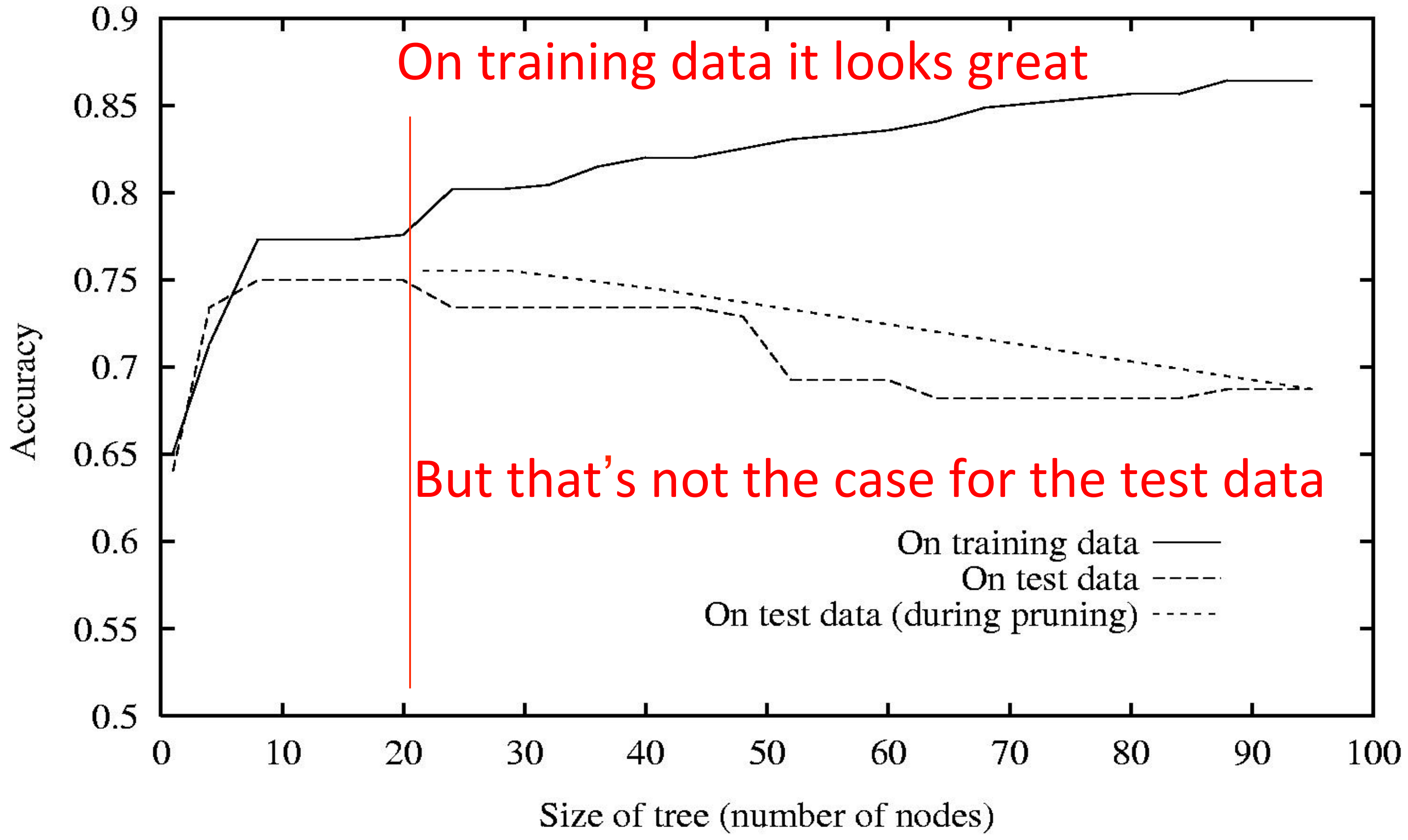
Based on Example from M. desJardins & T. Finin

Effect of Reduced-Error Pruning



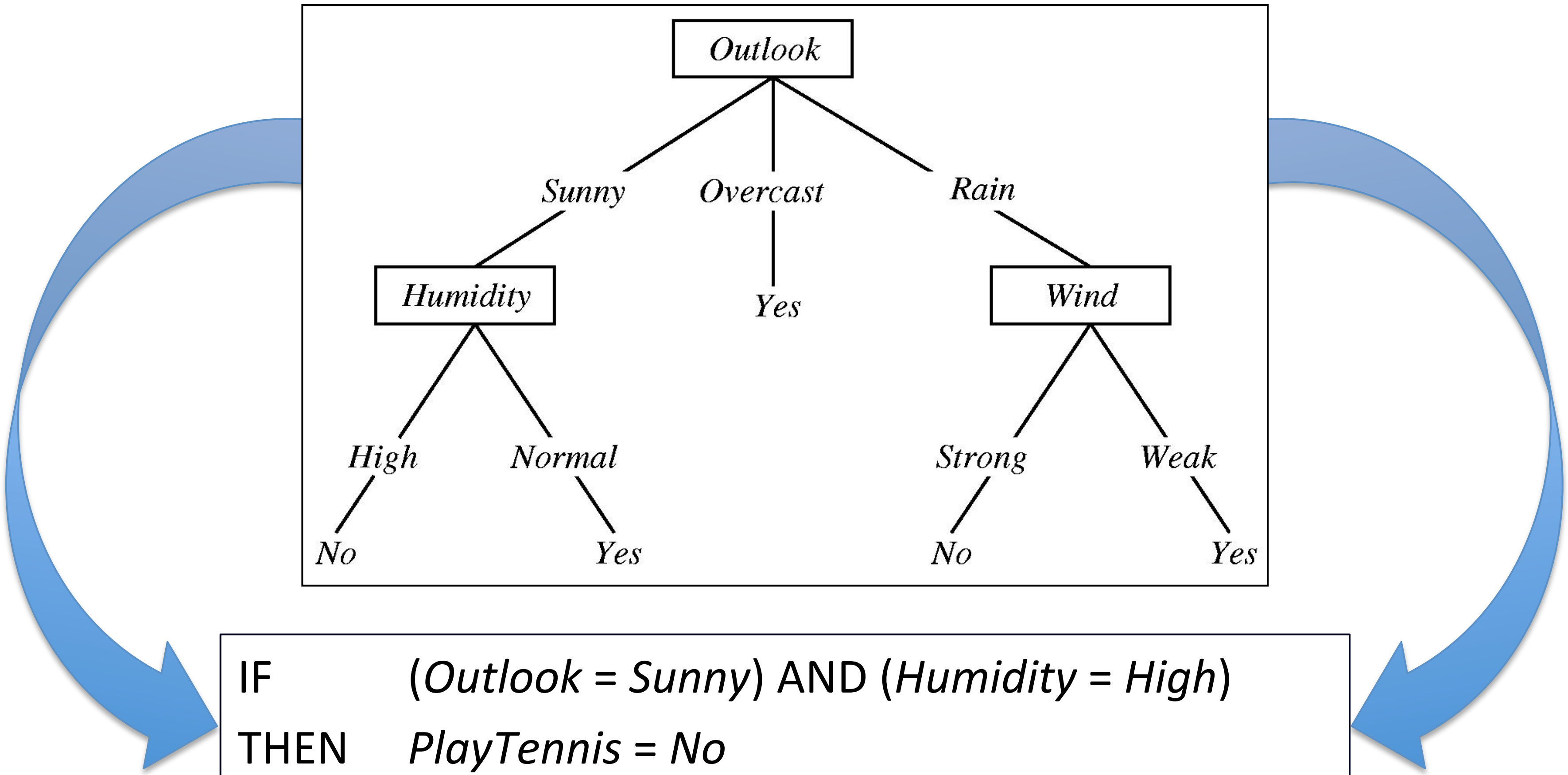
Based on Slide by Pedro Domingos

Effect of Reduced-Error Pruning



The tree is pruned back to the red line where it gives more accurate results on the test data

Converting a Tree to Rules



IF (*Outlook* = *Sunny*) AND (*Humidity* = *High*)
THEN *PlayTennis* = *No*

IF (*Outlook* = *Sunny*) AND (*Humidity* = *Normal*)
THEN *PlayTennis* = *Yes*

. . .

Converting Decision Trees to Rules

- It is easy to derive rules from a decision tree: write a rule for each path from the root to a leaf

  (*Outlook* = *Sunny*) AND (*Humidity* = *High*) → *PlayTennis* = *No*

- To simplify the resulting rule set:
  - Let LHS be the left-hand side of a rule
  - LHS' obtained from LHS by eliminating some conditions
  - Replace LHS by LHS' in this rule if the subsets of the training set satisfying LHS and LHS' are equal
  - A rule may be eliminated by using meta-conditions such as "if no other rule applies"

Rule Post-Pruning

1. Convert tree to equivalent set of rules
2. Prune each rule independently of others
3. Sort final rules into desired sequence for use

Perhaps most frequently used method (e.g., C4.5)

Scaling Up

- ID3, C4.5, etc.: assumes that data fits in memory (OK for up to hundreds of thousands of examples)

- SPRINT, SLIQ: multiple sequential scans of data (OK for up to millions of examples)

- VFDT: at most one sequential scan (OK for up to billions of examples)

Slide by Pedro Domingos

Comparison of Learning Methods

Characteristic	Neural Nets	SVM	Trees	MARS	k-NN, Kernels
Natural handling of data of "mixed" type	▼	▼	▲	▲	▼
Handling of missing values	▼	▼	▲	▲	▲
Robustness to outliers in input space	▼	▼	▲	▼	▲
Insensitive to monotone transformations of inputs	▼	▼	▲	▼	▼
Computational scalability (large N)	▼	▼	▲	▲	▼
Ability to deal with irrelevant inputs	▼	▼	▲	▲	▼
Ability to extract linear combinations of features	▲	▲	▼	▼	◆
Interpretability	▼	▼	◆	▲	▼
Predictive power	▲	▲	▼	◆	▲

[Table 10.3 from Hastie, et al. Elements of Statistical Learning, 2nd Edition]

Summary: Decision Tree Learning

- Representation: decision trees
- Bias: prefer small decision trees
- Search algorithm: greedy
- Heuristic function: information gain or information content or others
- Overfitting / pruning

Slide by Pedro Domingos

Summary: Decision Tree Learning

- Widely used in practice
- Strengths include
  - Fast and simple to implement
  - Can convert to rules
  - Handles noisy data
- Weaknesses include
  - Univariate splits/partitioning using only one attribute at a time --- limits types of possible trees
  - Large decision trees may be hard to understand
  - Requires fixed-length feature vectors
  - Non-incremental (i.e., batch method)
  - Sacrifices predictive power