Lecture 4: Decision Trees

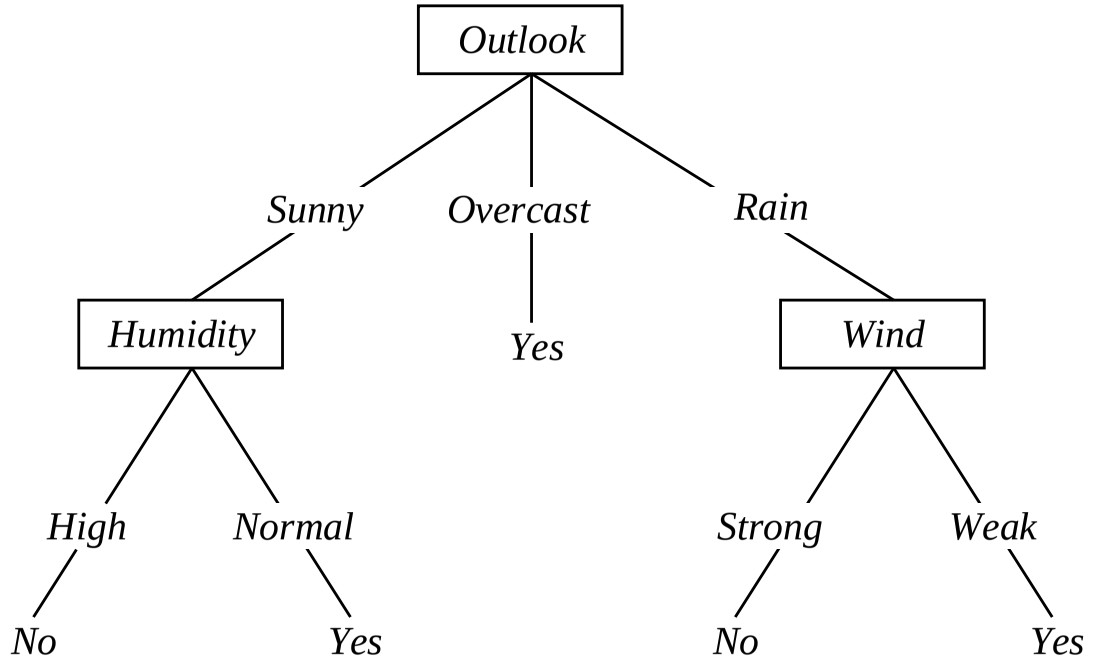
- What is a decision tree?
- Constructing decision trees
- Dealing with noise

Classification problem example

Day	*Outlook*	*Temperature*	*Humidity*	*Wind*	*PlayTennis*
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*

Discover a “rule” for the PlayTennis predicate!

Decision trees



A decision tree consists of:

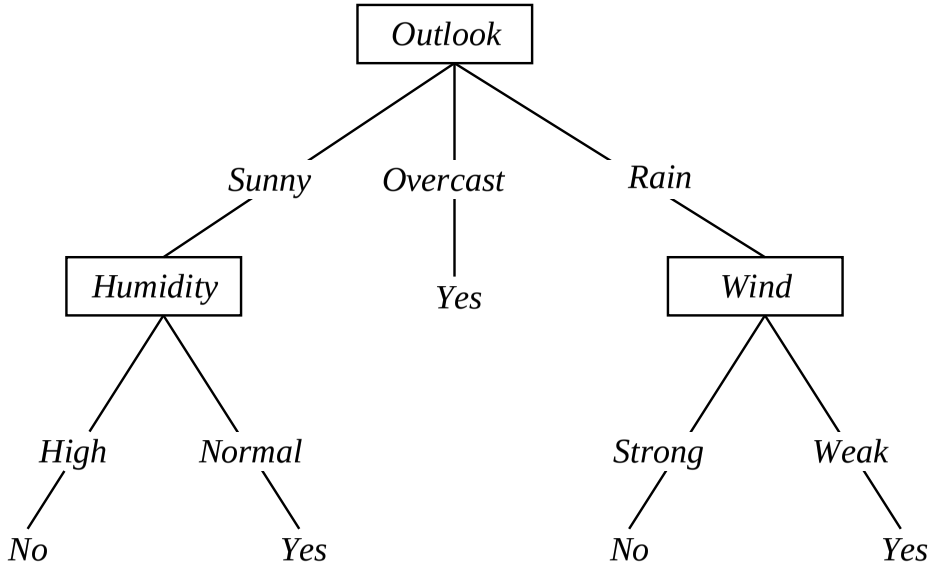
- a set of nodes, where each node tests the value of an attribute and branches on all possible values
- a set of leaves, where each leaf gives a class value

Using decision trees for classification

Suppose we get a new instance:

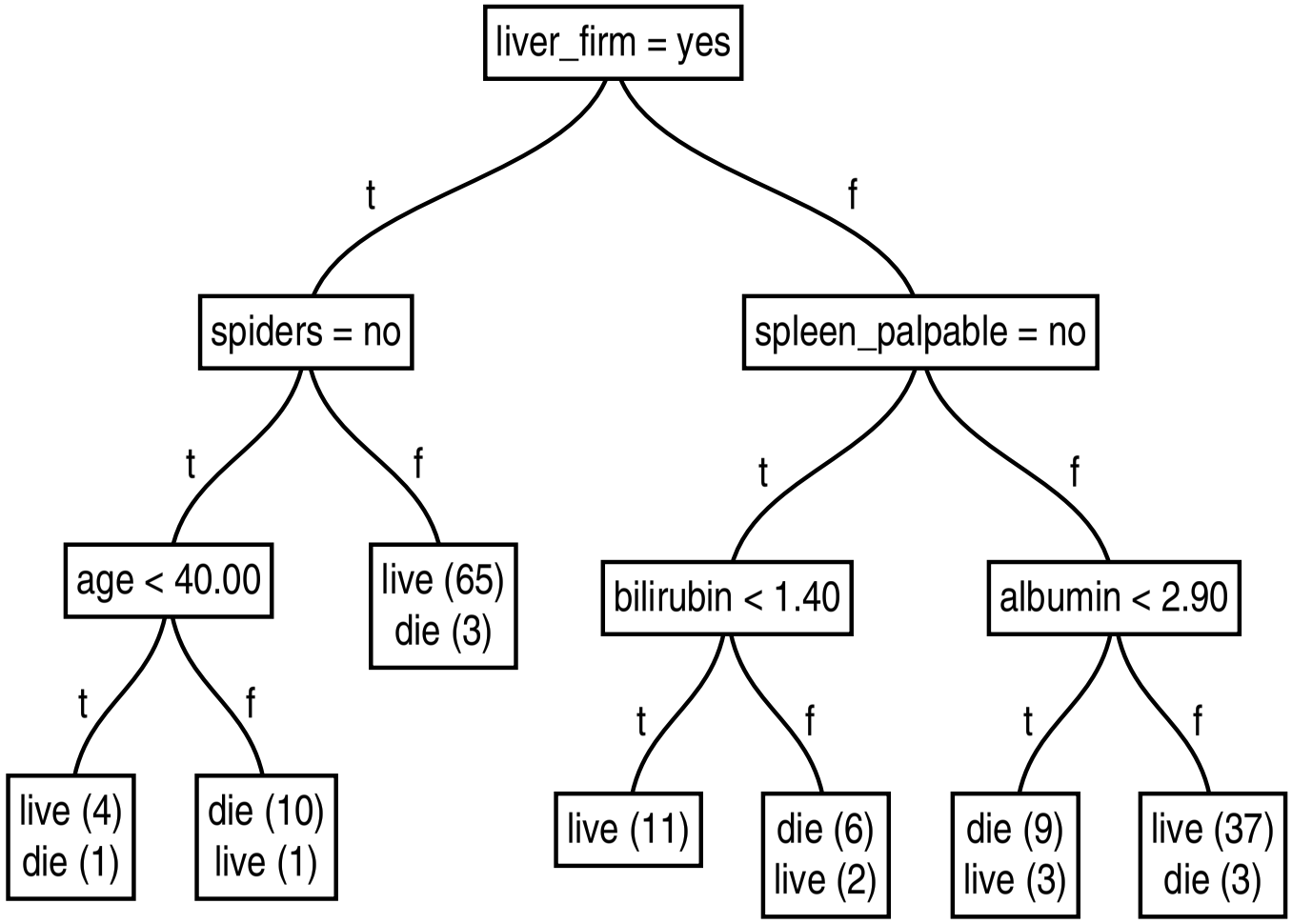
Outlook =Sunny, Temperature=Hot,Humidity=High,Wind=Strong

How do we classify it?



- At every node, test the corresponding attribute
- Send the instance down the appropriate branch of the tree
- If at a leaf, output the corresponding classification

Real example: the “hepatitis” task



Good things about decision trees

- Provide a general representation of classification rules
- Easy to understand!
- Fast learning algorithms (e.g. C4.5, CART)
- Robust to noise (attribute and classification noise, missing values)
- Good accuracy

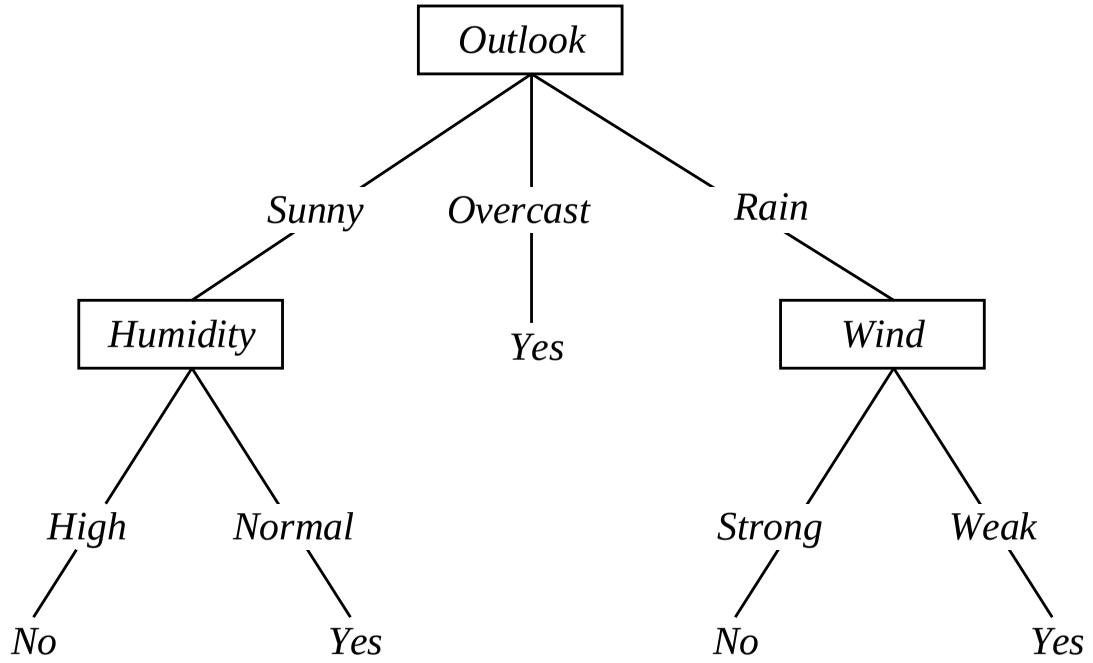
Decision trees are widely used in large, realistic classification problems, e.g.:

- Star classification
- Medical diagnosis
- Industrial applications

Often incorporated in data mining software (e.g. SGI Mineset).

Decision trees as logical representations

Each decision tree has an equivalent representation in propositional logic. For example:



corresponds to:

(Outlook=Sunny ∧ Humidity=Normal)

∨ (Outlook=Overcast) ∨ (Outlook=Rain ∧ Wind=Weak)

What is easy/hard for decision trees to represent ?

How would we represent:

- $\wedge$, $\vee$, XOR
- $(A \wedge B) \vee (C \wedge D)$
- *M* of *N*

Natural to represent disjunctions, hard to represent functions like parity, XOR (need exponential-size trees).

Sometimes duplication occurs (same subtree on various paths).

When would one use a decision tree?

- Data is represented as attribute-value pairs
- Target function is discrete valued
- Disjunctive hypothesis may be required
- Possibly noisy training data, missing values
- Need to construct a classifier fast
- Need an understandable classifier

Existing applications include:

- Equipment/medical diagnosis
- Learning to fly
- Scene analysis and image segmentation

Standard algorithm developed in the ’80s, now commercially available packages (C4.5). Quite successful in practice

Top-down induction of decision trees

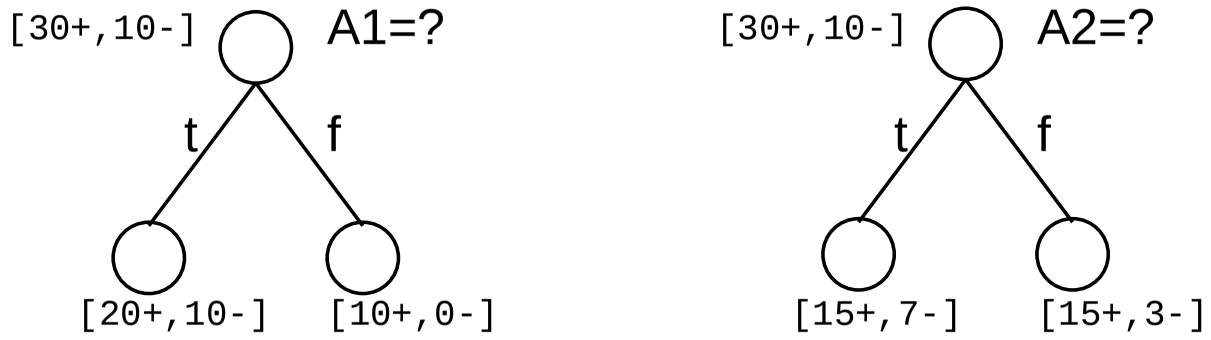
Given a set of labeled training instances:

1. If all the training instances have the same class, create a leaf with that class label and exit.
2. Pick the *best* attribute to split the data on
3. Add a node that tests the attribute
4. Split the training set according to the value of the attribute
5. Recurse on each subset of the training data

This is the ID3 algorithm (Quinlan, 1983) and is at the core of C4.5

Which attribute is best?

Consider we have 29 positive examples, 35 negative ones, and we are considering two attributes, that would give the following splits of instances:



Intuitively, we would like an attribute that *separates* the training instances as well as possible

We need a mathematical measure for the *purity* of a set of instances

A bit of information theory

Suppose you want to guess if a number is in a set S, and you can ask yes/no questions.

What is the best questioning strategy?

Pick the “middle” of S and ask if the number is less than that, then pick the middle of the remaining range etc.

You need $\log_2 |S|$ questions.

A bit of information theory (2)

Now suppose that the number can be in one of two subsets P and N, with probability p_P and p_N respectively, and you are told if the number is in P or N. What is the expected number of questions to ask?

$$p_P \log_2 |P| + p_N \log_2 |N|,$$

where

$$p_P = \frac{|P|}{|S|} \text{ and } p_N = \frac{|N|}{|S|}$$

A bit of information theory (3)

How much information do you gain by knowing if the number is in P or N?

$$
\begin{aligned}
I &= \log_2 |S| - p_P \log_2 |P| - p_N \log_2 |N| \\
&= (p_P + p_N) \log_2 |S| - p_P \log_2 |P| - p_N \log_2 |N| \\
&= -p_P \log_2 \frac{|P|}{|S|} - p_N \log_2 \frac{|N|}{|S|} \\
&= -p_P \log_2 p_P - p_N \log_2 p_N
\end{aligned}
$$

This is the **entropy function**

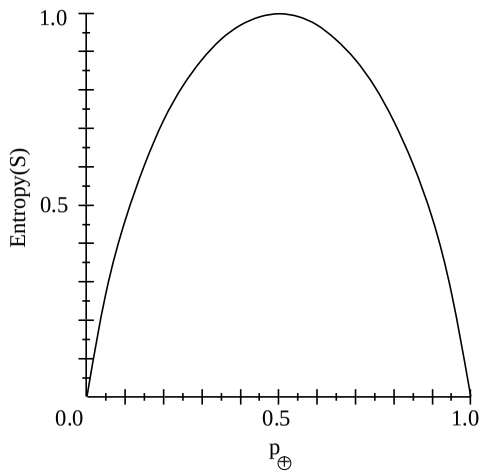
Entropy

Consider:

- S - a sample of training examples
- $p_{\oplus}$ is the proportion of positive examples in S
- $p_{\ominus}$ is the proportion of negative examples in S

Entropy measures the impurity of S:

$$Entropy(S) \equiv -p_{\oplus} \log_2 p_{\oplus} - p_{\ominus} \log_2 p_{\ominus}$$

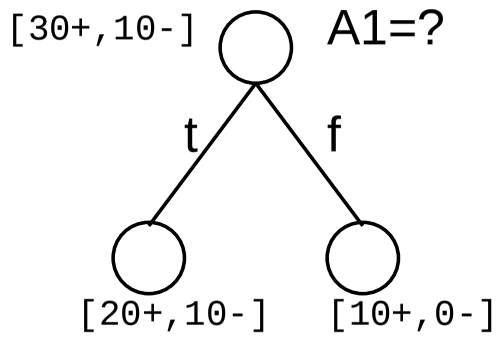


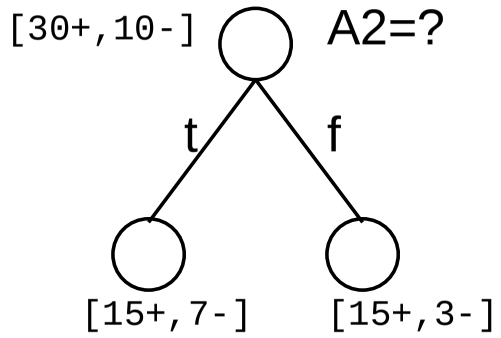
Information Gain

We will use entropy to determine what is the *best* attribute

Gain(S,A) = expected reduction in entropy due to sorting on attribute *A*

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





Check that in this case, A1 wins.

Decision tree construction as search

- State space: all possible trees
- Actions: which attribute to test
- Goal: tree consistent with the training data
- Depth-first search, no backtracking
- Heuristic: information gain (or other variations)
- Can get stuck in a local minimum, but is fairly robust (because of the heuristic)

Inductive bias of decision tree construction

- The hypothesis space is complete! We can represent any Boolean function of the attributes
- So there is *no representational bias*
- Outputs a single hypothesis: the “shortest” tree, as anticipated by the information gain
- Because there is no backtracking, it is subject to local minima
- But because the search choices are statistically based, it is robust to noise in the data
- *Algorithmic bias: prefer shorter (smaller) trees; prefer trees that place attributes with high information gain close to the root*

Occam's Razor: Why prefer short hypotheses?

Pro:

- There are fewer short hypotheses than long hypotheses
- So if we find one that fits the data, it is less unlikely to be a coincidence

Con:

- There are many ways to define short hypotheses (e.g. all trees with prime numbers of nodes)
- So what is so special about the size of the hypotheses?

A formal answer to this question can be given using the universal distribution: the probability of a hypothesis is $2^{-K(s)}$, where $K(s)$ is the length of the shortest program that can write down s.

For details, see Kirchherr and Li (1997).

Attributes with continuous values

Example:

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

A decision tree needs to perform tests on these attributes as well

What kind of test do we want?

Value of the attribute less than a cut point!

What cut points should we consider?

We need to consider only cut points where the class label changes!

Using decision trees for real data

- How do we estimate classifier error
- How to deal with noise in the data
- How to deal with missing attributes
- How to incorporate attribute costs

Example: CRX data, UCI Repository

```
|  This file concerns credit card applications.  All attribute names
|  and values have been changed to meaningless symbols to protect
|  confidentiality of the data.

+, -.   | classes

A1:     b,a.
A2:     continuous.
A3:     continuous.
A4:     u, y, l, t.
A5:     g, p, gg.
A6:     c, d, cc, i, j, k, m, r, q, w, x, e, aa, ff.
A7:     v, h, bb, j, n, z, dd, ff, o.
A8:     continuous.
A9:     t,f.
A10:    t,f.
A11:    continuous.
A12:    t,f.
A13:    g, p, s.
A14:    continuous.
A15:    continuous.
```

Estimating the accuracy of a classifier

We want to estimate the *true error* of the classifier

- **Resubstitution:** Build the classifier using all the training data, and test it using the same data

  Too optimistic! (why?)
- **Test sample estimation:** Divide the data into a *training set* and a *test set*.

  Wastes data
- **Cross-validation:** General method for determining accuracy.

k-fold cross-validation procedure

1. Split the training data into k partitions (folds), ensuring that the class distribution is roughly the same in each partition
2. Repeat k times:
   (a) Take one fold to be the test set
   (b) Take the remaining $k-1$ folds to form the training set
   (c) We train the decision tree on the training set, then measure $TrainingError_i$ and $TestError_i$
3. Report the average of $TrainingError_i$ **and** the average of $TestError_i$.

Magic number: $k = 10$.

More about cross-validation

- *Leave-one-out cross-validation:* special case in which the test is performed on just one instance.
  Used especially if data is scarce.

- If for any reason we need a validation set (for the algorithm itself), that will be kept separate from the training and test sets
  E.g. One fold is for testing, one for validation and the remaining $k-2$ for training

- If we are comparing different algorithms *test them on the SAME folds!*

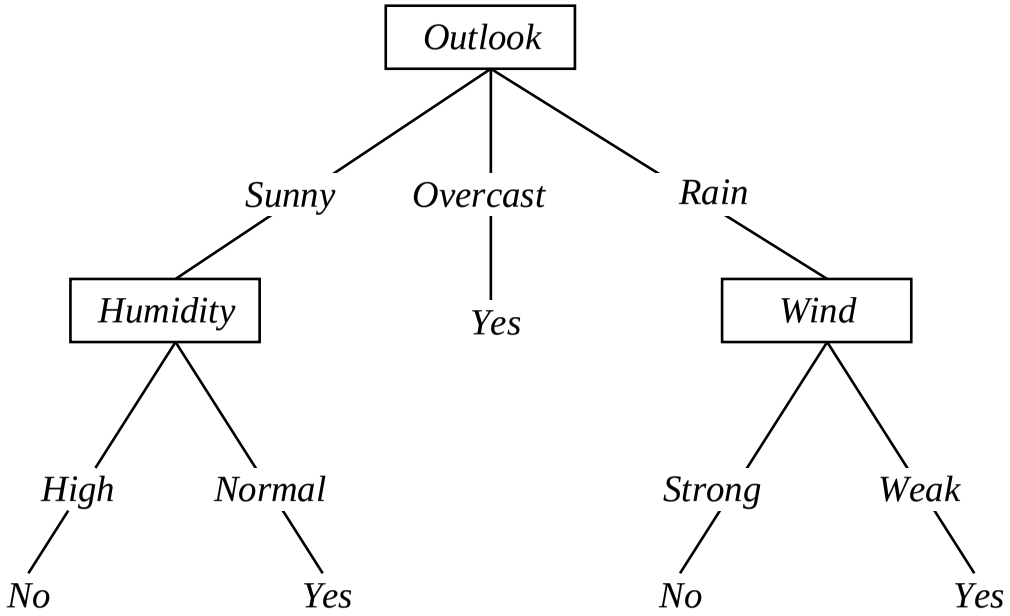
Dealing with noise in the training data

Noise is inevitable!

- Values of attributes can be misrecorded
- Values of attributes may be missing
- The class label can be misrecorded

What happens when adding a noisy example?

Example: The effect of noise



Suppose we add to the data a noisy example:

Sunny, Hot, Normal, Strong, PlayTennis=No

The tree grows unnecessarily!

Overfitting

Consider error of hypothesis h over

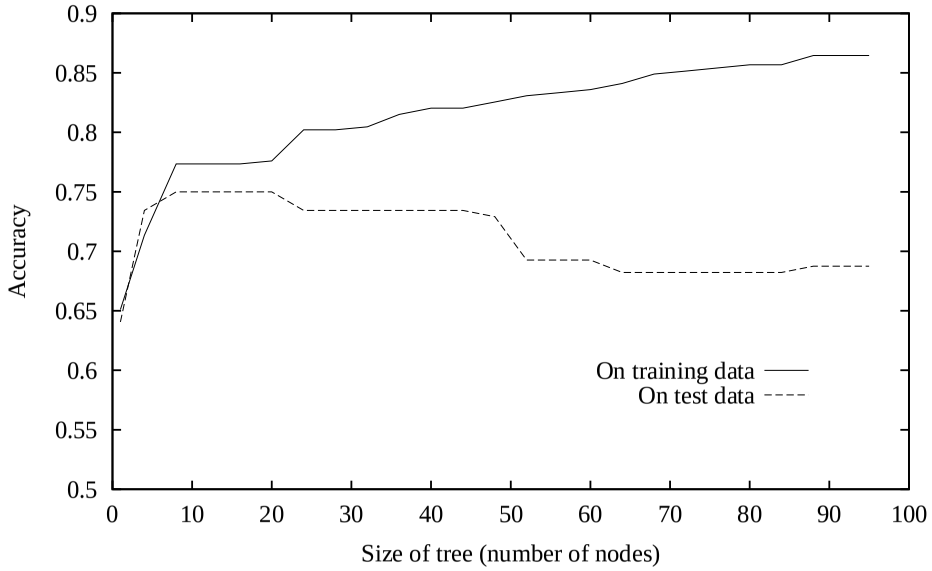
- Training data: $error_{train}(h)$
- Entire distribution $\mathcal{D}$ of data: $error_{\mathcal{D}}(h)$

Hypothesis h **overfits** training data if there is an alternative hypothesis h' such that

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

This is a general problem for all supervised learning methods

Overfitting in decision trees



As the tree grows, the accuracy degrades, because the algorithm is finding *irrelevant* attributes.

Do not believe anyone's results unless they report them on separate training and test sets!

Avoiding overfitting

1. Stop growing the tree when further splitting the data does not yield a statistically significant improvement
2. Grow a full tree, then *prune* the tree, by eliminating nodes

The second approach has been more successful in practice

In both cases, the leaves of the tree will now be impure:

- The leaf can be assigned the class label of the majority of the instances which reached the leaf
- Alternatively, one can use probability estimates of the class membership, based on instance counts.

How to select the “best” tree

1. Measure performance over training data only
2. Measure performance over a separate validation data set
3. Minimum description length principle: minimize

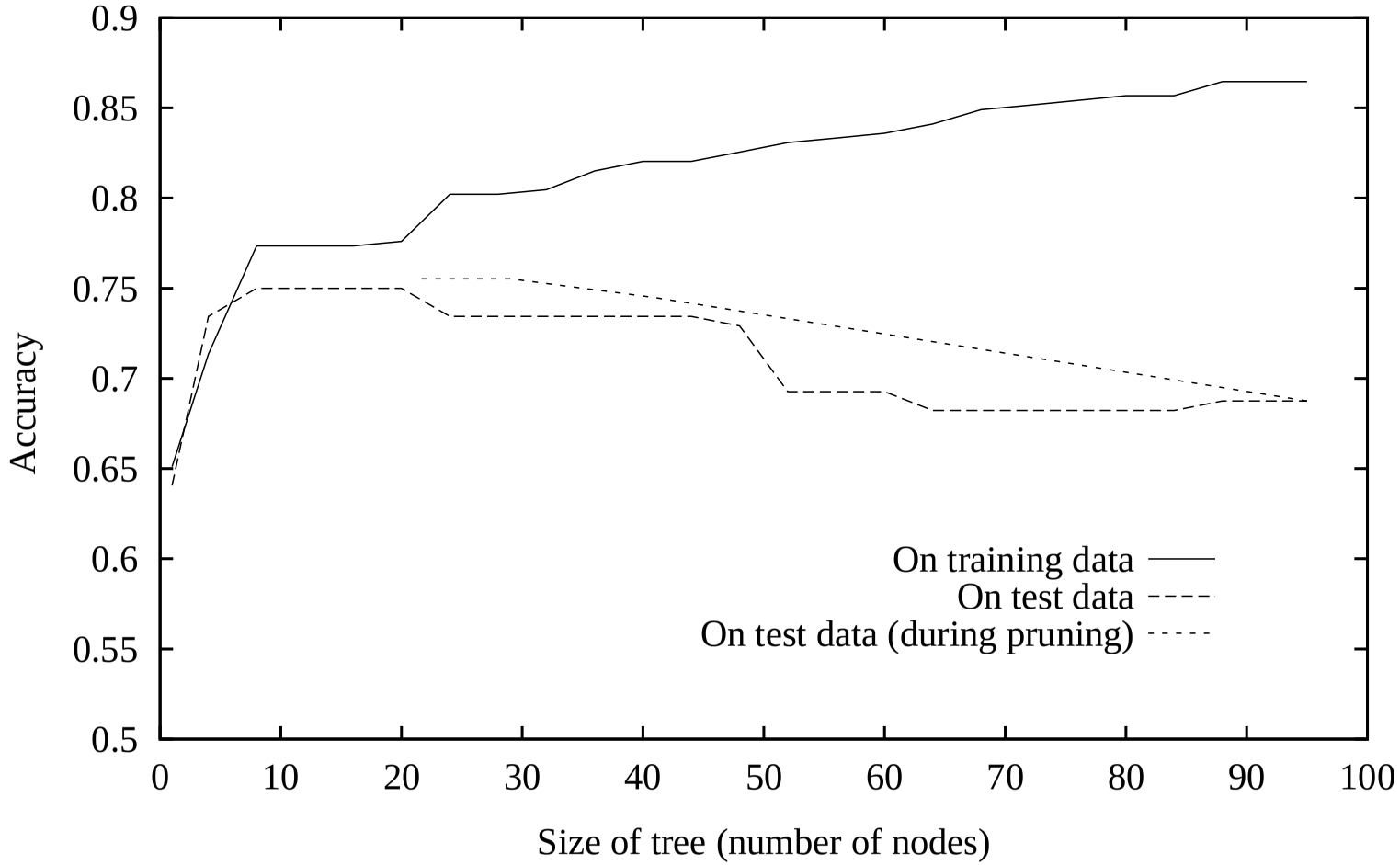
$$size(tree) + size(misclassifications(tree))$$

The second one (training and validation set) is the most common.

Example: Reduced-error pruning

1. Split data into a *training set* and a *validation set*
2. Grow a large tree (e.g. until each leaf is pure)
3. For each node:
   (a) Evaluate the validation set accuracy of pruning the subtree rooted at the node
   (b) Greedily remove the node that most improves validation set accuracy, with its corresponding subtree
   (c) Replace the removed node by a leaf with the majority class of the corresponding examples.
4. Stop when pruning starts hurting the accuracy on the validation set.

Example: Effect of reduced-error pruning



Example: Rule post-pruning in C4.5

1. Convert the decision tree to rules
2. Prune each rule independently of the others, by removing preconditions such that the accuracy is improved
3. Sort final rules in order of estimated accuracy

C4.5 builds a pessimistic estimate of the estimate from the accuracy on the training set.

Advantages:

- Can prune attributes higher up in the tree *differently on different paths*
- There is no need to reorganize the tree if pruning an attribute that is higher up
- Most of the time people want rules anyway, for readability