Lecture 4: Decision Trees

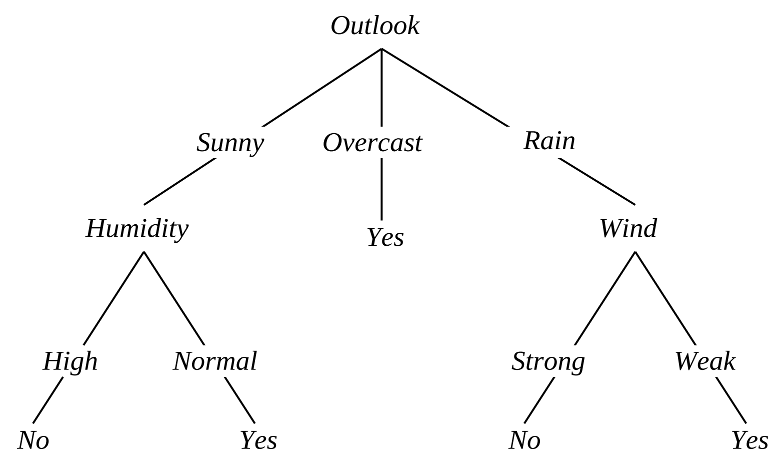
- What is a decision tree?
- Constructing decision trees
- Dealing with noise

Decision tree example (1)

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 tree example (2)



A decision tree is:

a set of nodes, where each node tests the value of an attribute and branches on all possible values

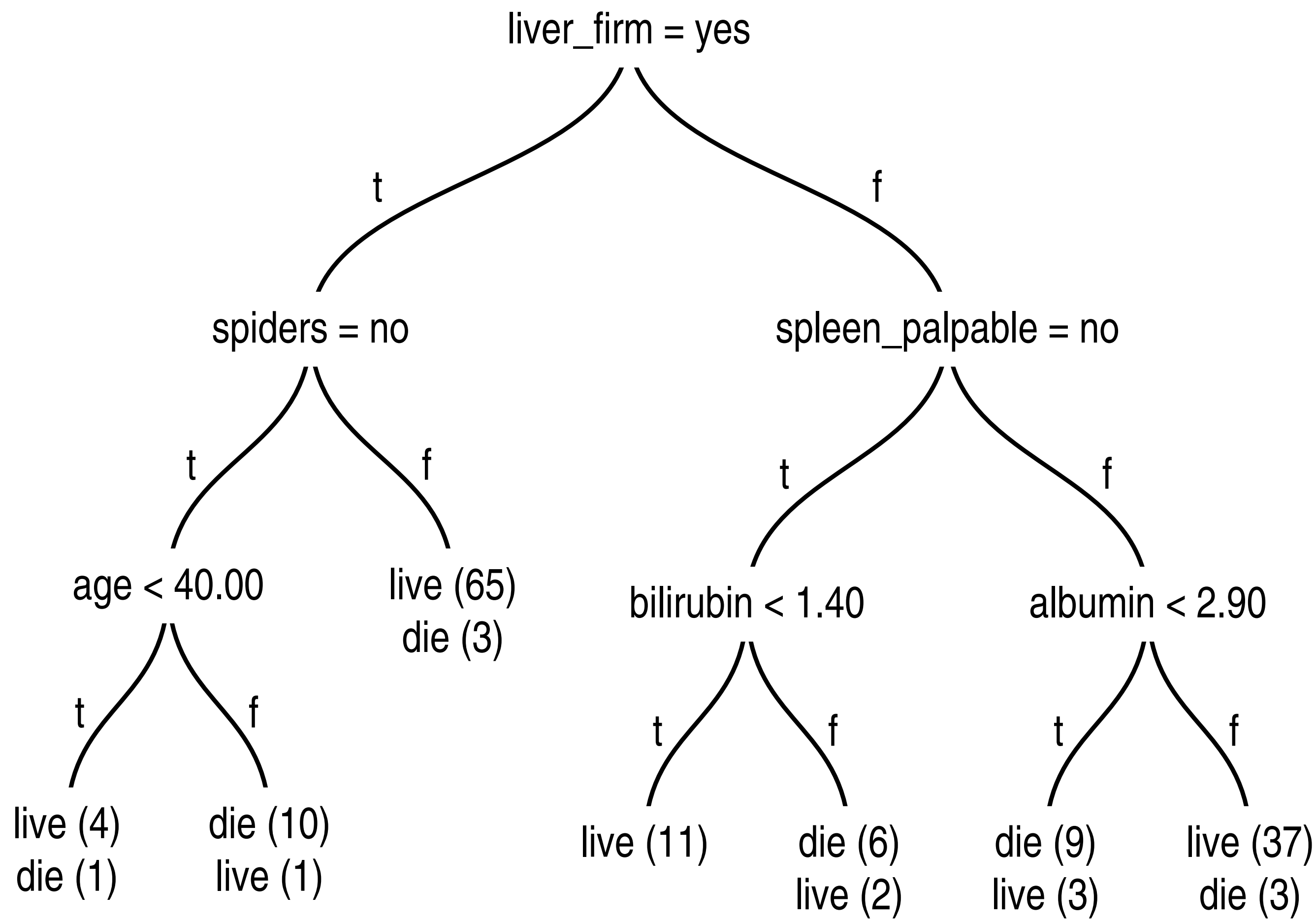
a set of leaves, where each leaf gives a class value

Suppose we get a new instance:

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

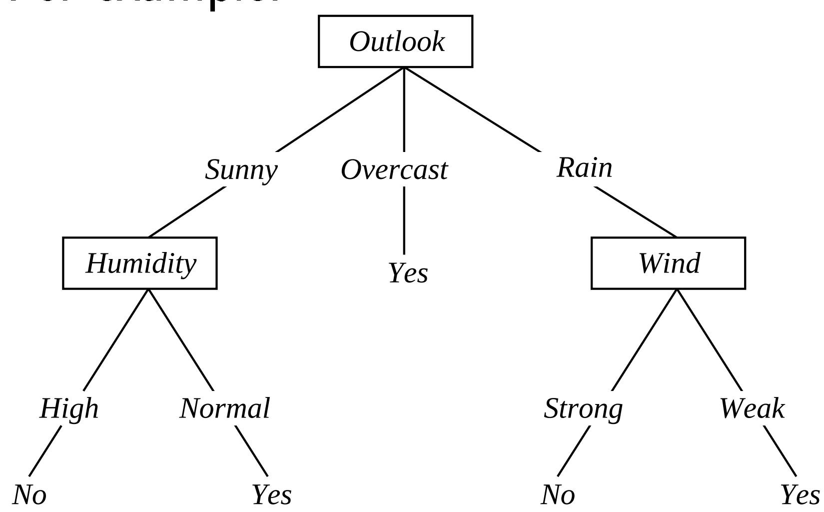
How do we classify it?

Real example: the "hepatitis" task



Decision trees as logical representations

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



corresponds to:

(Outlook=Sunny $\wedge$ Humidity=Normal)
$\vee$ (Outlook=Overcast) $\vee$ (Outlook=Rain $\wedge$ 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?

- Classification problems: instances come 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
- Credit risk analysis
- Learning to fly
- Scene analysis and image segmentation

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

Decision tree construction

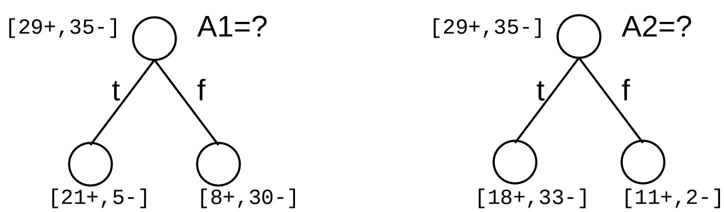
Given a set of labelled training instances:

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 attribues, 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

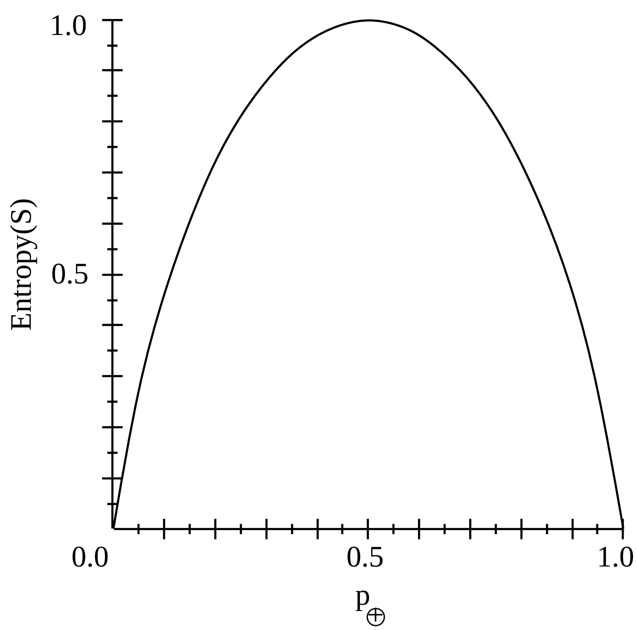
Entropy

Consider:

S - a sample of training examples
p_+ is the proportion of positive examples in S
p_- is the proportion of negative examples in S

Entropy measures the impurity of S:

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



Why this formula?

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.

Now suppose that the number can be in one of two subsets P and N and I am willing to tell you where to look. What is the expected number of questions to ask?

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

Why this formula? (2)

Now how much information is there in this case, compared with not knowing anything?

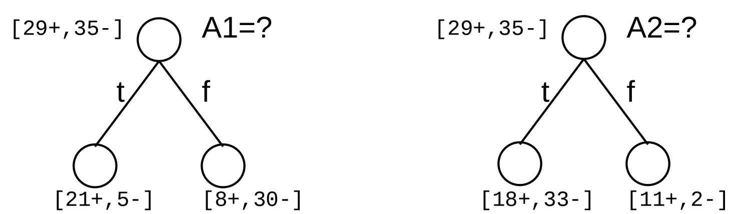
$$p_P \log_2 |P| + p_N \log_2 |N| - (p_P + p_N) \log_2 |S|$$

If you compute it it comes to the entropy formula

Information Gain

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

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



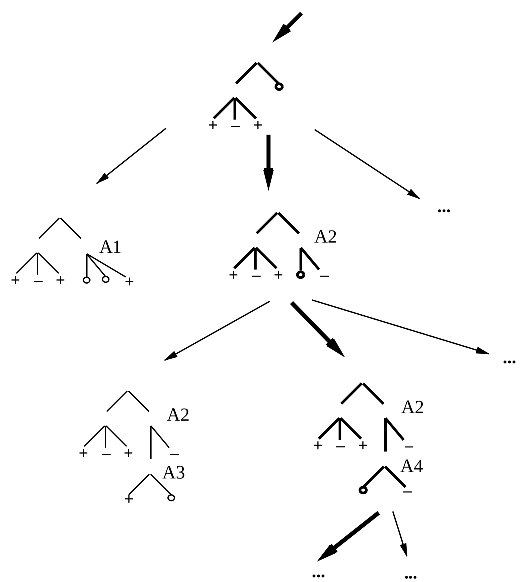
$$Entropy(S) = -\frac{29}{64}\log_2\frac{29}{64} - \frac{35}{64}\log_2\frac{35}{64}$$

$$Gain(S,A1) = Entropy(S) - \frac{26}{64}Entropy(S1(A1)) - \frac{38}{64}Entropy(S2(A1))$$

$$Gain(S,A2) = Entropy(S) - \frac{51}{64}Entropy(S1(A2)) - \frac{13}{64}Entropy(S2(A2))$$

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 (becase 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 absolute 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
- *Preference 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 hypothezses than long hypotheses
- So if we find one that fits the data, it is less unlikely to be a conincidence

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 top this question can be given using the universal distribution (more about this later).

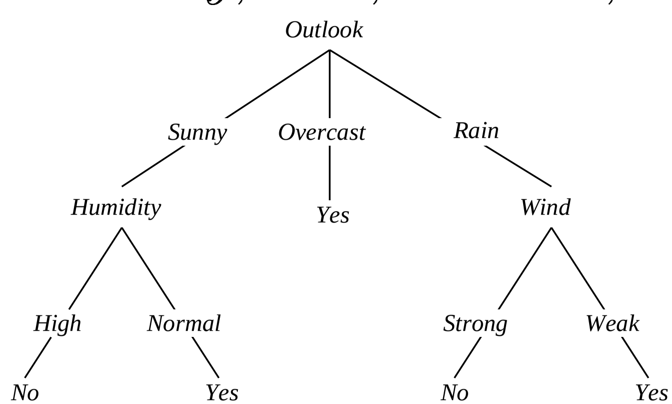
Dealing with noise in the training data

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?

$$Sunny,\ Hot,\ Normal,\ Strong,\ PlayTennis = No$$



The tree grows unnecessarily!

Overfitting

Consider error of hypothesis h over

Training data: $error_{train}(h)$

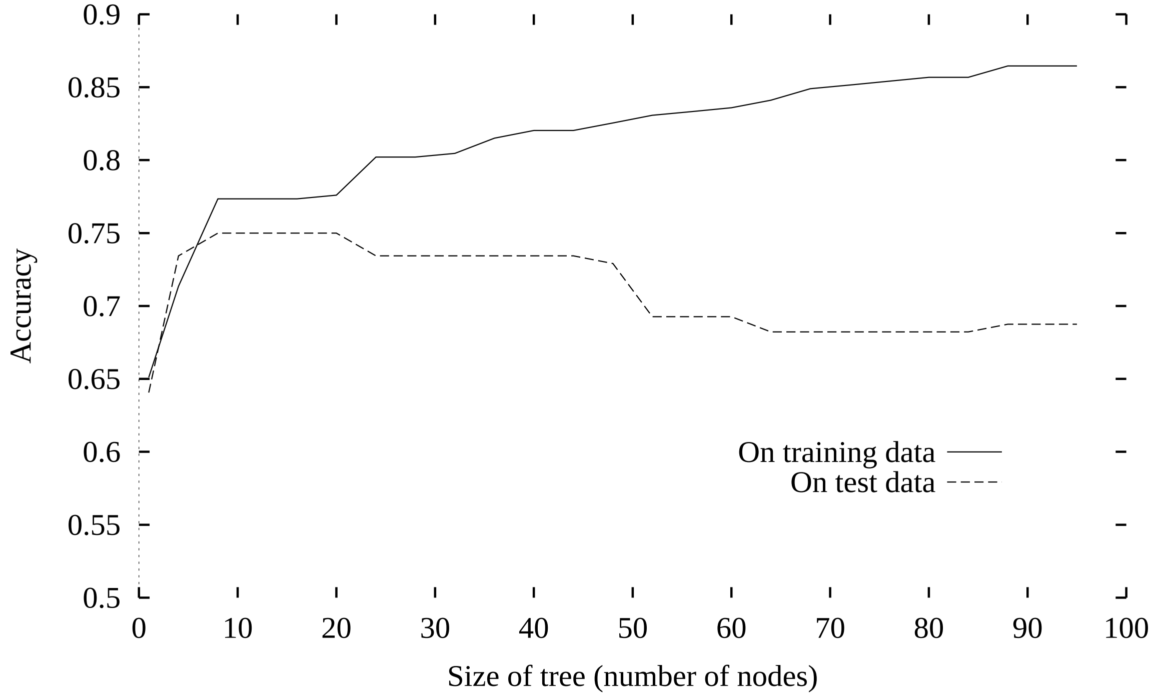
Entire distribution $\mathcal{D}$ of data: $error_{\mathcal{D}}(h)$

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

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

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 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

How to select the "best" tree:

1. Measure performance over training data only
2. Measure performance over 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.

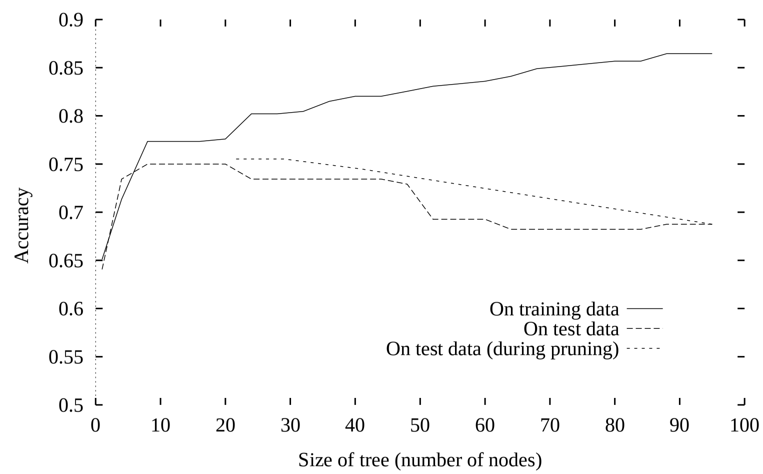
Example: Reduced-Error Pruning

Split data into *training* and *validation* 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 one that most improves *validation* set accuracy

Produces smallest version of most accurate subtree



Example: Rule post-pruning

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

Currently the most frequently used method (e.g. C4.5)

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

How do we evaluate the accuracy of a decision tree

A general approach, that we will use for other classifiers as well, is k-**fold cross-validation**

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 $TestError_i$.

Most often $k = 10$.

More about cross-validation

If for any reason we need a validation set, 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 data is limited, an alternative method is *leave-one-out cross-validation*, where we justkeep 1 example for testing.

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