Lecture 5: Decision Trees (Part II)

- Dealing with noise in the data
  - Overfitting
  - Pruning
- Dealing with missing attribute values
- Dealing with attributes with multiple values
- Integrating costs into node choice
- Decision trees for regression

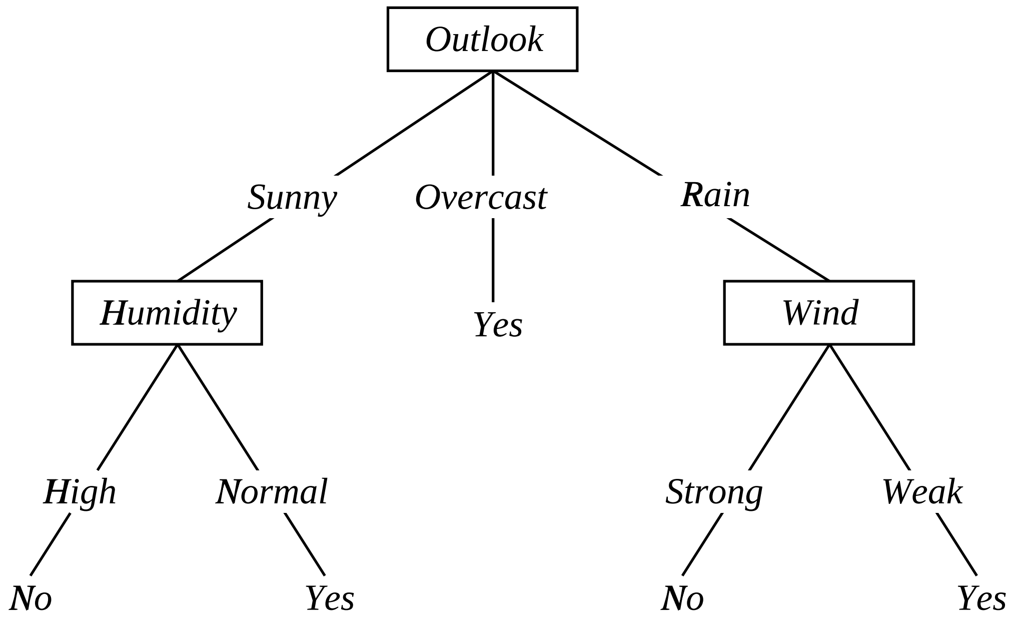
Dealing with noise in the training data

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

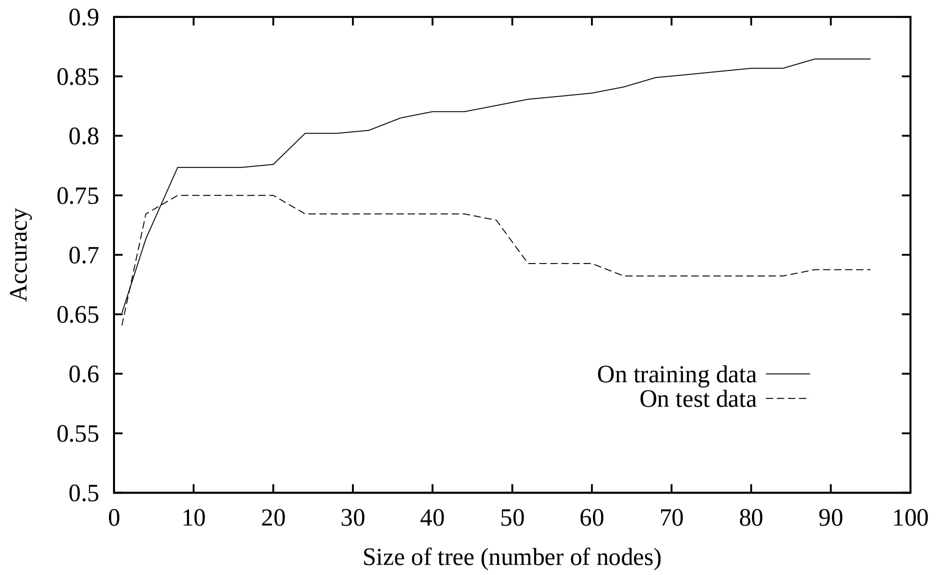
- Training data: $error_{train}(h)$
- Entire distribution $\mathcal{D}$ of data: $error_{\mathcal{D}}(h)$

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

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

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 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

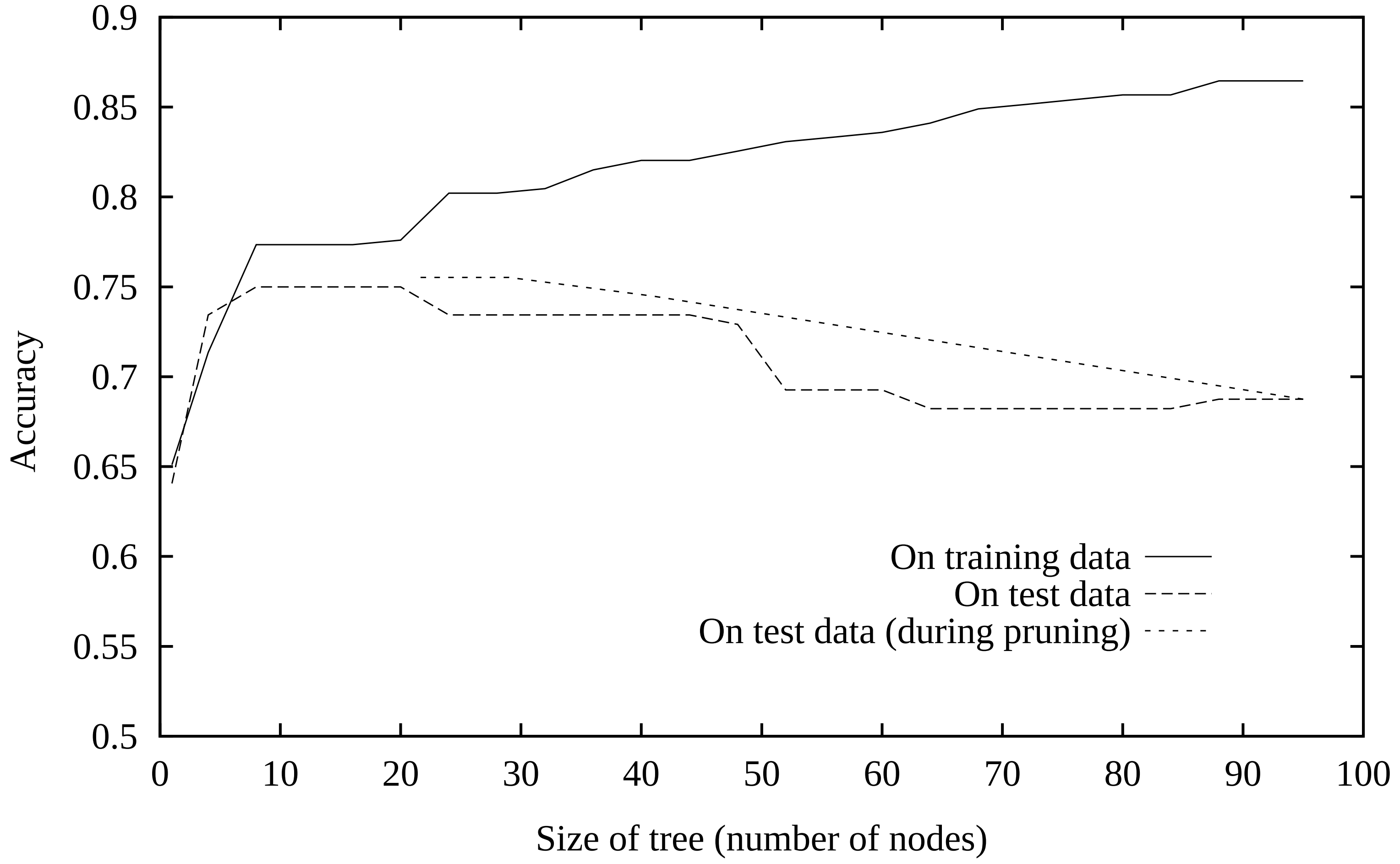
$$size(tree) + size(misclassifications(tree))$$

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

Attributes with multiple values

If an attribute splits the data perfectly, it will always be preferred by information gain.

E.g. a unique ID for each data point!

But that has very poor generalization performance.

You would think pruning can help, but what can you do with a tree that just has one node?

Two solutions:

1. Use another criterion that is more fair
2. Ensure that all attributes have the same number of values

A better criterion: Gain ratio

For a set of instances S and an attribute A with v possible values

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

where

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

So for an attribute that splits the data into many partitions mostly uniformly, *SplitInformation* will be high

Problem: It can actually become too high!

Solution: First use *Gain*, then use *GainRatio* for attributes with *Gain* above average

Other such metrics are also used.

Ensuring the same number of values

If an attribute A has $v > 2$ possible values, $Val_1..Val_v$, replace it by v Boolean attributes, $A_k, k = 1..v$, where:

$$A_k = \begin{cases} 1 & \text{if } A = Val_k \\ 0 & \text{otherwise} \end{cases} \quad , \; \forall k = 1..v$$

This is called *1-of-v encoding*

Used more generally to encode learning data (e.g. in neural networks)

Missing values during classification

- “Most likely” value based on all the data that reaches the current node. “Most likely” means the most frequent attribute value
- Assign all possible values with some probability. Usually we just count the occurrences of the different attribute values in the instances that have reached the same node. We will predict all the possible class labels with the appropriate probabilities too.

Missing values during tree construction

1. Introduce an “unknown” value
2. Modify gain ratio to take into account the probability of an attribute being known:

$$Gain(S, A)P(A)$$

where P(A) is the fraction of the instances that reached the node, in which the value was known

Costs of attributes

Include cost in the metric, e.g.

$$\frac{Gain^2(S, A)}{Cost(A)}$$

Mostly a problem in specific domains (e.g. medicine).

Multiple metrics have been studied and proposed, without a consensus.

Decision trees for regression

Regression problem: given a set of instances $x_1^i..x_m^i, y^i, i = 1..n$, where x_k^i are attribute-value pair and y^i is a real number, find a function $f : X_1 \times \ldots \times X_m \rightarrow \Re$ that approximates the training points well.

Usually, by “approximate well” we mean minimize the mean squared error:

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (y^i - f(x_1^i, \ldots x_m^i))^2$$

How can we use decision trees for regression?

Main idea: construct a *piece-wise constant approximation!*

Basic CART algorithm (Breiman et. al, 1984)

Given a set of labeled training instances $x_1^i..x_m^i, y^i, i = 1..n$, where each label y^i is a real number:

1. Compute the average of all the labels: $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y^i$
2. Compute the mean squared error of the instances:

$$\frac{1}{n}\sum_{i=1}^{n}(y^i - \bar{y})^2$$

3. If the error is below a desired threshold, create a leaf with the label $\bar{y}$ (why?)
4. Otherwise pick the *best* attribute to split the data
5. Add a node that tests the attribute
6. Split the training set according to the value of the attribute
7. Recurse on each subset of the training data

Choosing the best attribute

The same principle as in classification: we want the attribute that minimizes the error in each partition.

The error in this case is the sum of the mean square errors from each partition.

If $v \in V$ are all the possible values of an attribute, and the corresponding partitions have N_v examples, then we want to minimize:

$$\sum_{v \in V} \frac{1}{N_v} \sum_{i=1}^{N_v} (y^i - \bar{y}^v)^2$$

Pruning in CART

The program looks for a tree that minimizes a cost function with two components:

- The mean squared error on the training data
- The size of the tree

This is called *cost-complexity pruning*.

Summary

- Decision trees are logical representations, and can represent any hypothesis
- The construction algorithm works top-down and is greedy with respect to the information gain metric
- This means that the decision trees obtained are not guaranteed to be “optimal” in any sense
- However, the algorithm has good accuracy in practice, is very fast, and produces classifiers that are easy to interpret.
- General mechanisms exist for dealing with problems in real data sets (real-valued attributes, attributes with multiple values, missing data, etc.)

Summary (continued)

- Like all machine learning algorithms, decision trees are prone to overfitting (i.e. capturing the regularities of the training set).
- In decision trees, overfitting causes too many nodes to be created
- Pruning methods avoid overfitting by regulating the number of nodes (typically by deleting nodes)
- **Because all learning algorithms overfit, it is essential to evaluate the algorithm on a separate test set, that has not been used during training!** Most of the time we do this using cross-validation.
- Decision trees can be used for regression tasks as well.