Decision Trees
(Part II: Pruning the tree)
nanopoulos@ismll.de
**Underfitting and Overfitting**

2000 points in two classes (1000 per class)
Swap 150 points between the classes
1000 training/1000 test
Swap additional 200 in training set

---

**Underfitting and Overfitting**

Underfitting: when model is too simple, both training and test errors are large
Overfitting due to Noise

(a) Decision tree with 11 leaf nodes. (b) Decision tree with 24 leaf nodes.

Decision boundary is distorted by noise point.
<table>
<thead>
<tr>
<th>Name</th>
<th>Body Temperature</th>
<th>Gives Birth</th>
<th>Four-legged</th>
<th>Hibernates</th>
<th>Class Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>porcupine</td>
<td>warm-blooded</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>cat</td>
<td>warm-blooded</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>bat</td>
<td>warm-blooded</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no^</td>
</tr>
<tr>
<td>whale</td>
<td>warm-blooded</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no^</td>
</tr>
<tr>
<td>salamander</td>
<td>cold-blooded</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>komodo dragon</td>
<td>cold-blooded</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>python</td>
<td>cold-blooded</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>salmon</td>
<td>cold-blooded</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>eagle</td>
<td>warm-blooded</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>guppy</td>
<td>cold-blooded</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
</tbody>
</table>

Body Temperature

- Warm-blooded
- Cold-blooded

Gives Birth

- Yes
- No

Four-legged

- Yes
- No

Class Label

- Mammals
- Non-mammals

Non-mammals

- Mammals
- Non-mammals
Overfitting due to Insufficient Examples

Lack of data points in the lower half of the diagram makes it difficult to predict correctly the class labels of that region.

- Insufficient number of training records in the region causes the decision tree to predict the test examples using other training records that are irrelevant to the classification task.

<table>
<thead>
<tr>
<th>Name</th>
<th>Body Temperature</th>
<th>Gives Birth</th>
<th>Four-legged</th>
<th>Hibernates</th>
<th>Class Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>salamander</td>
<td>cold-blooded</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>guppy</td>
<td>cold-blooded</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>eagle</td>
<td>warm-blooded</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>poorwill</td>
<td>warm-blooded</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>platypus</td>
<td>warm-blooded</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>
Notes on Overfitting

Overfitting results in decision trees that are more complex than necessary

Training error no longer provides a good estimate of how well the tree will perform on previously unseen records

Need new ways for estimating errors
Estimating Generalization Errors

Re-substitution errors: error on training (Σ e(t))
Generalization errors: error on testing (Σ e'(t))

Methods for estimating generalization errors:

**Optimistic approach:** e'(t) = e(t)

**Pessimistic approach:**
- For each leaf node: e' (t) = (e(t) + 0.5)
- Total errors: e'(T) = e(T) + N × 0.5 (N: number of leaf nodes)
- For a tree with 30 leaf nodes and 10 errors on training (out of 1000 instances):
  - Training error = 10/1000 = 1%
  - Generalization error = (10 + 30×0.5)/1000 = 2.5%

**Reduced error pruning (REP):**
- uses validation data set to estimate generalization error

---

Occam’s Razor

Given two models of similar generalization errors, one should prefer the simpler model over the more complex model

For complex models, there is a greater chance that it was fitted accidentally by errors in data

Therefore, one should include model complexity when evaluating a model
How to Address Overfitting

Pre-Pruning (Early Stopping Rule)
Stop the algorithm before it becomes a fully-grown tree

Typical stopping conditions for a node:
- Stop if all instances belong to the same class
- Stop if all the attribute values are the same

More restrictive conditions:
- Stop if number of instances is less than some user-specified threshold
- Stop if class distribution of instances are independent of the available features (e.g., using $\chi^2$ test)
- Stop if expanding the current node does not improve impurity measures (e.g., Gini or information gain).

How to Address Overfitting...

Post-pruning
Grow decision tree to its entirety
Trim the nodes of the decision tree in a bottom-up fashion
If generalization error improves after trimming, replace sub-tree by a leaf node.
Class label of leaf node is determined from majority class of instances in the sub-tree
Can use MDL for post-pruning
Example of Post-Pruning

Class = Yes | 20
Class = No | 10
Error = 10/30

Training Error (Before splitting) = 10/30
Pessimistic error = (10 + 0.5)/30 = 10.5/30
Training Error (After splitting) = 9/30
Pessimistic error (After splitting)
= (9 + 4 × 0.5)/30 = 11/30
PRUNE!

A?
A1
A2
A3
A4

Class = Yes | 8
Class = No | 4

Class = Yes | 3
Class = No | 4

Class = Yes | 4
Class = No | 1

Class = Yes | 5
Class = No | 1

Reduced Error Pruning

Quinlan 1978
Mingers 1978
Esposito et al. 1996
Elomaa & Kaariainen 2001
Partitioning Data in Tree Induction

Estimating accuracy of a tree on new data: “Test Set”
Some post pruning methods need an independent data set: “Pruning Set”

All available data

Growing Set

Pruning Set

Training Set

Test Set

To evaluate the classification technique, experiment with repeated random splits of data

Typical Proportions

All available data

Growing Set

Pruning Set

Problem with using “Pruning Set”: less data for “Growing Set”
Reduced Error Pruning (REP)

Use pruning set to estimate accuracy of sub-trees and accuracy at individual nodes
Let $T$ be a sub-tree rooted at node $v$

Define:

Gain from pruning at $v = \#\text{misclassification in } T - \#\text{misclassification at } v$

Repeat: prune at node with largest gain until only negative gain nodes remain

“Bottom-up restriction”: $T$ can only be pruned if it does not contain a sub-tree with lower error than $T$

REP example

\[
E(T_{v_2}) = 3, \ E(v_2) = 2, \ E(T_{v_3}) = 1, \ E(v_3) = 3
\]
Real implementations

<table>
<thead>
<tr>
<th>name</th>
<th>CHAID</th>
<th>CART</th>
<th>ID3</th>
<th>C4.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>measure</td>
<td>( \chi^2 )</td>
<td>Gini index, entropy index</td>
<td>information gain</td>
<td>information gain ratio</td>
</tr>
<tr>
<td>splits</td>
<td>all</td>
<td>binary nominal, binary qualitative, ordinal qualitative</td>
<td>complete</td>
<td>complete, binary nominal, binary quantitative</td>
</tr>
<tr>
<td>stopping criterion</td>
<td>( \chi^2 ) independence test</td>
<td>minimum number of cases/node</td>
<td>( \chi^2 ) independence test</td>
<td>lower-bound on selection measure</td>
</tr>
<tr>
<td>pruning technique</td>
<td>none</td>
<td>error complexity pruning</td>
<td>pessimistic error pruning</td>
<td>pessimistic error pruning, error based pruning</td>
</tr>
</tbody>
</table>

Example: C4.5

Simple depth-first construction.
Uses Information Gain
Sorts Continuous Attributes at each node.
Needs entire data to fit in memory.
Unsuitable for Large Datasets.
Needs out-of-core sorting.

You can download the software from:

http://www.cse.unsw.edu.au/~quinlan/c4.5r8.tar.gz