Basic Models for Supervised Learning

Many learning algorithms can be seen as deriving from:

- decision trees
- linear (and non-linear) classifiers
- Bayesian classifiers

Learning Decision Trees

- Representation is a decision tree.
- Bias is towards simple decision trees.
- Search through the space of decision trees, from simple decision trees to more complex ones.

Decision trees

A (binary) decision tree (for a particular output feature) is a tree where:

- Each nonleaf node is labeled with an test (function of input features).
- The arcs out of a node labeled with values for the test.
- The leaves of the tree are labeled with point prediction of the output feature.

Example Classification Data

Training Examples:

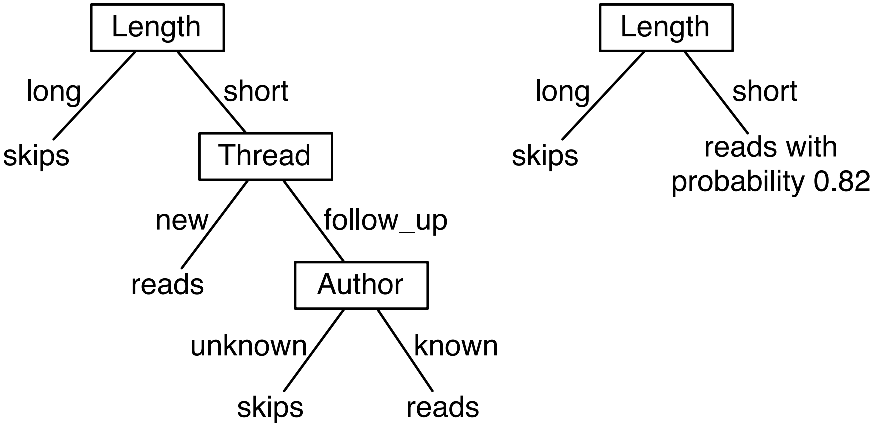
	Action	Author	Thread	Length	Where
e1	skips	known	new	long	home
e2	reads	unknown	new	short	work
e3	skips	unknown	old	long	work
e4	skips	known	old	long	home
e5	reads	known	new	short	home
e6	skips	known	old	long	work

New Examples:

e7	???	known	new	short	work
e8	???	unknown	new	short	work

We want to classify new examples on feature *Action* based on the examples' *Author*, *Thread*, *Length*, and *Where*.

Example Decision Trees



Equivalent Logic Program

$$skips \leftarrow long.$$

$$reads \leftarrow short \wedge new.$$

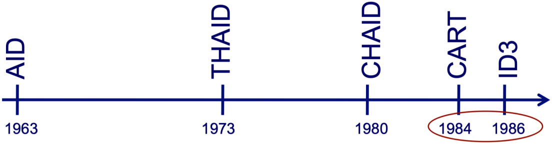
$$reads \leftarrow short \wedge follow\_up \wedge known.$$

$$skips \leftarrow short \wedge follow\_up \wedge unknown.$$

Issues in decision-tree learning

- Given some training examples, which decision tree should be generated?
- A decision tree can represent any discrete function of the input features.
- You need a bias. Example, prefer the smallest tree. Least depth? Fewest nodes? Which trees are the best predictors of unseen data?
- How should you go about building a decision tree? The space of decision trees is too big for systematic search for the smallest decision tree.

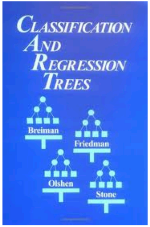
History of decision tree learning



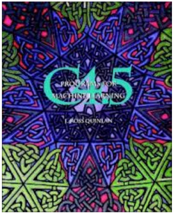
many DT variants have been developed since CART and ID3

dates of seminal publications: work on these 2 was contemporaneous

CART developed by Leo Breiman, Jerome Friedman, Charles Olshen, R.A. Stone



ID3, C4.5, C5.0 developed by Ross Quinlan



Searching for a Good Decision Tree

- The input is a set of input features, a target feature and, a set of training examples.
- Either:
  - Stop and return the a value for the target feature or a distribution over target feature values
  - Choose a test (e.g. an input feature) to split on.
    For each value of the test, build a subtree for those examples with this value for the test.

Choices in implementing the algorithm

- When to stop:

Choices in implementing the algorithm

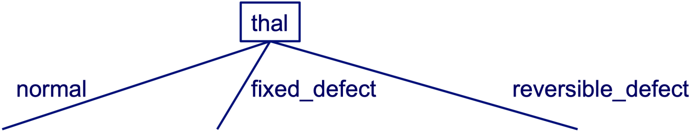
- When to stop:
  - no more input features
  - all examples are classified the same
  - too few examples to make an informative split

Top-down decision tree learning

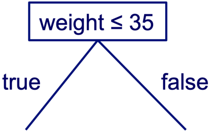
```
MakeSubtree(set of training instances D)
  C = DetermineCandidateSplits(D)
  if stopping criteria met
    make a leaf node N
    determine class label/probabilities for N
  else
    make an internal node N
    S = FindBestSplit(D, C)
    for each outcome k of S
      D_k = subset of instances that have outcome k
      k^th child of N = MakeSubtree(D_k)
  return subtree rooted at N
```

Candidate splits

- splits on nominal features have one branch per value



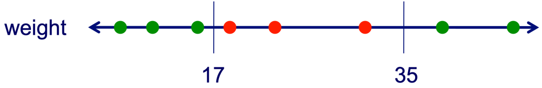
- splits on continuous features use a threshold

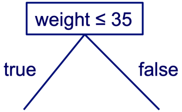


Candidate splits on continuous features

given a set of training instances D and a specific feature F

- sort the values of F in D
- evaluate split thresholds in intervals between instances of different classes





- could use midpoint of each considered interval as the threshold
- C4.5 instead picks the largest value of F in the entire training set that does not exceed the midpoint

Candidate splits on a numeric feature

```
;; For each numeric feature at each node of DT induction
DetermineCandidateNumericSplits(training instances D, feature x_i)
  C = {} ;; initialize set of candidate splits for feature x_i
  S = partition instances in D into sets s_1 ... s_V
       where the instances in each set have the same value for x_i
  let v_j denote the value of x_i for set s_j
  sort the sets in S using v_j as the key for each s_j
  for each pair of adjacent sets s_j, s_{j+1} in sorted S
    if s_j, s_{j+1} contain pair of instances
              with different class labels
      ;; use midpoints for splits
      add candidate split x_i ≤ (v_j + v_{j+1})/2 to C
  return C
```

Finding the best split

- How should we select the best feature to split on at each step?
- Key hypothesis: the simplest tree that classifies the training examples accurately will work well on previously unseen examples

Occams razor

- attributed to 14th century William of Ockham
- Nunquam ponenda est pluralitis sin necesitate
- Entities should not be multiplied beyond necessity
- should proceed to simpler theories until simplicity can be traded for greater explanatory power
- when you have two competing theories that make exactly the same predictions, the simpler one is the better

Occams razor and decision trees

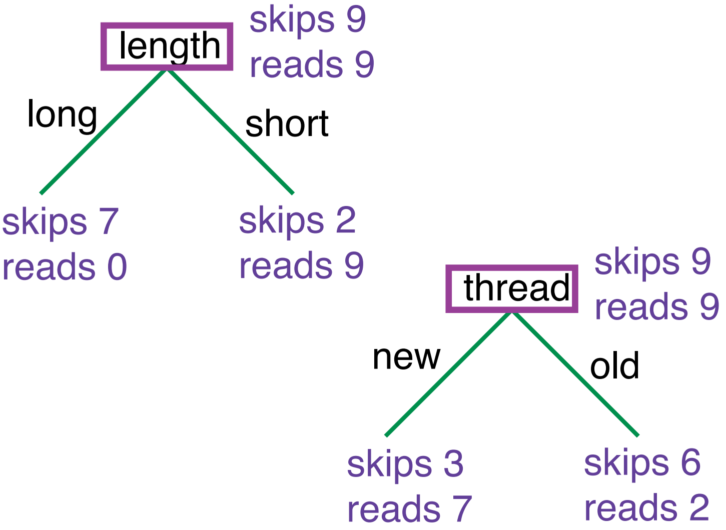
Why is Occams razor a reasonable heuristic for decision tree learning?

- there are fewer short models (i.e. small trees) than long ones
- a short model is unlikely to fit the training data well by chance
- a long model is more likely to fit the training data well coincidentally

Finding the best splits

- Which test to split on isn't defined.
- Can we return the smallest possible decision tree that accurately classifies the training set?
  NO! This is an NP-hard problem
- Instead, well use an information-theoretic heuristic to *greedily* choose splits
- Often we use myopic split: which single split gives smallest error.
- Myopia is a limitation: an important feature may not appear to be informative until used in conjunction with other features;
  - a lookahead search strategy can potentially alleviate this limitation.
- With multi-valued features, the text can be can to split on all values or split values into half. More complex tests are possible.

Example: possible splits



Handling Overfitting

- This algorithm can overfit the data.
  This occurs when noise and correlations in the training set that are not reflected in the data as a whole.
- To handle overfitting:
  - restrict the splitting, and split only when the split is useful.
  - allow unrestricted splitting and prune the resulting tree where it makes unwarranted distinctions.
  - learn multiple trees and average them.

Handling Overfitting

- This algorithm can overfit the data.
  This occurs when noise and correlations in the training set that are not reflected in the data as a whole.
- To handle overfitting:
  - restrict the splitting, and split only when the split is useful.
  - allow unrestricted splitting and prune the resulting tree where it makes unwarranted distinctions.
  - learn multiple trees and average them.

Handling Overfitting

- This algorithm can overfit the data.
  This occurs when noise and correlations in the training set that are not reflected in the data as a whole.
- To handle overfitting:
  - restrict the splitting, and split only when the split is useful.
  - allow unrestricted splitting and prune the resulting tree where it makes unwarranted distinctions.
  - learn multiple trees and average them.

Handling Overfitting

- This algorithm can overfit the data.
  This occurs when noise and correlations in the training set that are not reflected in the data as a whole.
- To handle overfitting:
  - restrict the splitting, and split only when the split is useful.
  - allow unrestricted splitting and prune the resulting tree where it makes unwarranted distinctions.
  - learn multiple trees and average them.