Alpaydin
Ch 9.

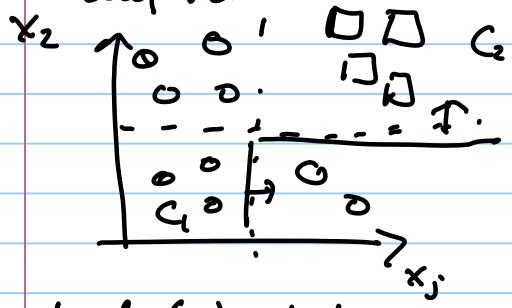
Decision Trees

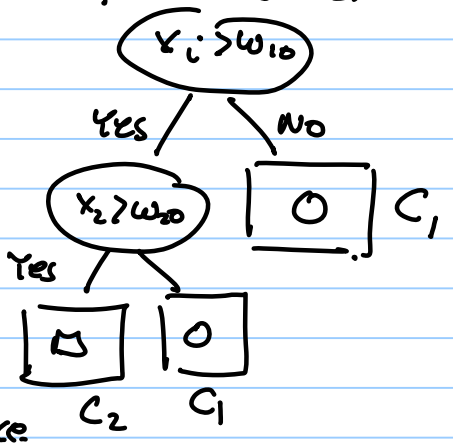
Note Title 3/30/2008

Divide and Conquer strategy for classification.

Tree. Decision Node w test function $f_m(x)$

Leaf Node





Each $f_m(\underline{x})$ defines a discriminant in d-dim input space.

Hierarchical structure → Best case: can find one of N regions with $\log N$ questions

Decision Trees are also interpretable, because they can be converted to a set of IF-THEN rules.

Univariate Tree

test uses only one of the input dimensions.

→ e.g. attributes — color, size

$f_m(\underline{x}) : x_j \geq w_{m0}$ ← threshold value.

binary split:

$$L_m = \{\underline{x} \mid x_j \geq w_{m0}\}$$
$$R_m = \{\underline{x} \mid x_j < w_{m0}\}$$

Finding best tree is NP-complete, so use local heuristic search procedures.

Classification Trees

Goodness of a split is quantified by an impurity measure.

Pure – if all instances after split belong to the same class.

Node m, N_m is no. instances reaching m (i.e. N for root node).

N^i_m of N_m belongs to class i, $\sum_i N^i_m = N_m$

The estimate for probability of class C_i is

$$\hat{P}(C_i | \underline{x}, m) \equiv p^i_m = N^i_m / N_m .$$

Node m is pure if $p^i_m = 0$ or 1, for all i.

Impurity Measure: $I_m = -\sum_{i=1}^{K} p^i_m \log p^i_m$.
entropy.

Other measures: Gini index $\phi(p, 1-p) = 2p(1-p)$

If node m is not pure, then the node should be split to decrease impurity

test reach j.

$$\hat{P}(C_i | \underline{x}, m_j) \equiv p^i_{m_j} = N^i_{mj} / N_{mj}$$

Impurity after split $I'_m = -\sum_{j=1}^{n} \frac{N_{mj}}{N_m} \sum_{i=1}^{K} p^i_{mj} \log p^i_{mj}$.

(3)

Classification & Regression Trees (CART)

CART For all attributes and for all possible splits – calculate the impurity and choose the one with maximal purity.

Then repeat recursively and in parallel for all branches that are not pure.

Continue till all branches are pure.

Danger – growing the tree until we have pure leafs risks overfitting the training data.

In practice, use a threshold θ_ℓ

Don't split any node with impurity $< \theta_\ell$.

For leaf Nodes ^{m} – output the posterior probability $p(C_i | m)$ – or output MAP $\hat{C}_i = \text{ARGMAX}\, p(C_i | m)$

Regression Trees

Similar to classification tree, except we replace the impurity measure by one more suitable for regression.

node m, $\mathcal{X}_m$ subset of $\mathcal{X}$ reaching node m.

define $b_m(\underline{x}) = \begin{cases} 1 & \text{if } \underline{x} \in \mathcal{X}_m : \underline{x} \text{ reaches node } m \\ 0 & \text{otherwise} \end{cases}$

g_m is the estimated value in node m.

$$E_m = \frac{1}{N_m} \sum_t (r^t - g_m)^2 b_m(\underline{x}^t)$$

$$N_m = |\mathcal{X}_m| = \sum_t b_m(\underline{x}^t)$$

<u>Select</u> $$g_m = \frac{\sum_t b_m(\underline{x}^t) r^t}{\sum_t b_m(\underline{x}^t)}$$

If E_m is below threshold, then create a leaf node and store the g_m value.

(Creates a piecewise approximation with discontinuities at leaf boundaries.)

If E_m is above threshold, split the node so as to decrease the errors of the child nodes.

$\mathcal{X}_{mj}$ subset of $\mathcal{X}_m$ taking branch j; $\bigcup_{j=1}^{n} \mathcal{X}_{mj} = \mathcal{X}_m$

$b_{mj}(\underline{x}) = \begin{cases} 1 & \text{if } \underline{x} \in \mathcal{X}_{mj} : \underline{x} \text{ reaches } m \text{ and take branch } j \\ 0 & \text{otherwise} \end{cases}$

$$g_{mj} = \frac{\sum_t b_{mj}(x^t) r^t}{\sum_t b_{mj}(x^t)}$$

$$E_m' = \frac{1}{N_m} \sum_j \sum_t (r^t - g_{mj})^2 b_{mj}(\underline{x}^t)$$ — error after split.

(5)

Pruning

pre-pruning – stopping tree construction before we have reached pure leaf nodes.

post-pruning – try to find and remove unnecessary sub-trees.

Grow tree until all nodes are pure.
Then find subtrees that cause overfitting.

To do so – obtaining pruning set of instances.
For each subtree,
replace it by a leaf node labeled with training instances covered by subtree.
If leaf node does not perform worse than the subset on the pruning set, then prune the subtree and keep the leaf node.

pre-pruning is faster – but post-pruning usually gives more accurate trees

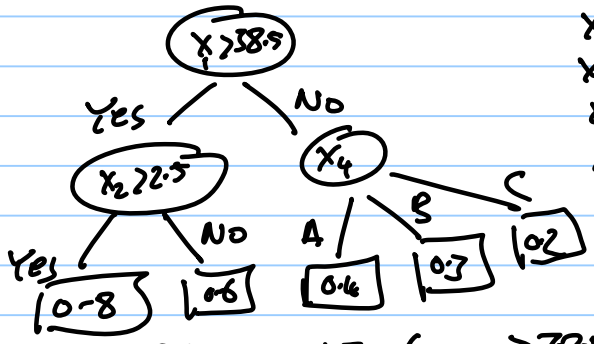
(6)

Rule Extraction from Trees

Interpretability

Each path from root to leaf corresponds to a conjunction of tests.

These paths can be written down as a set of IF-THEN rules — called a RULE-BASE



X_1 : Age
X_2 : Years in job
X_3 : Gender
X_4 : Job Type

R1 : IF (age > 38.5) AND (years-on-job > 2.5) THEN $y = 0.8$

R2 : IF (age > 38.5) AND (years-in-job < 2.5) THEN $y = 0.6$
R3 : IF (age ≤ 38.5) AND (job-type = A) THEN $y = 0.4$
R4 : AND (job-type = B) THEN $y = 0.7$
R5 : AND (job-type = C) THEN $y = 0.2$

Can validate the rules (by an expert)

Can also calculate the percentage of training data covered by the rule (e.g. rule support).

Several different paths may have the same leaf values. This gives disjunction of conjunctions.

(7) # Learning Rules from Data.

Rule induction - works similarly to tree induction but does depth-first search and generates one path of a tree.

Example: RIPPER (see Alpaydin p 188 for pseudo code)

Rules are added to explain positive examples

Rule R to rule R'.

Change in gain:

$$Gain(R', R) = s \cdot \left(\log \frac{N'_+}{N'} - \log \frac{N_+}{N} \right)$$

N instances covered by R N_+ no. true positives

N' & N'_+ for R.

Conditions are added to a rule until it covers no negative example.

Can prune back using a pruning set to find rule that maximizes the rule value metric $rum = \frac{p-n}{p+n}$

p - no. true / n - no. false positives on pruning set

(8) Propositional Rules & First-Order Rules

IF Father(y,x) AND Female(y)
THEN Daughter(x,y)

Inductive Logic Programming
learning first order rules is similar to learning propositional rules

Multivariate Trees
All input variables can be used to split
$f_m(\underline{x}) : \underline{w}_m^T \underline{x} + w_{m0} > 0$

Alternatives are the sphere nodes
$f_m(\underline{x}) : \|\underline{x} - \underline{s}_m\| \leq d_m$

Algorithm proceeds as before.