Decision Trees

Divide and Conquer strategy for classification.

Each $f_n(x)$ defines a discriminant in $d$-dim input space.

Hierarchical structure $\rightarrow$ Real 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 dimension.

$\rightarrow$ e.g. attribute = color, size

$\text{fn}(x) : x_j > w_0$ (threshold value)

Binary split: $L_m = \{ x | x_j > w_0 \}$

$R_m = \{ x | x_j < w_0 \}$

Finding best tree is P/P$^c$-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 \( i \), \( \frac{\sum_i N^i_m}{N_m} = N^i_m \).

The estimate for probability of class \( C_i \) is:

\[
\hat{p}(C_i | X, m) = \frac{N^i_m}{N_m}
\]

Node \( m \) is pure if \( \hat{p}^m = 0 \) or \( 1 \), for all \( i \).

Impurity Measure: \( I_m = -\sum_{i=1}^{k} \hat{p}^m \log \hat{p}^m \). 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:

\[
\hat{p}(C_i | X, m_j) = \frac{N^i_{m_j}}{N_{m_j}}
\]

Impurity after split:

\[
I_m' = -\sum_{j=1}^{N_m} \frac{N_{m_j}}{N_m} \sum_{i=1}^{k} \frac{N^i_{m_j}}{N_{m_j}} \hat{p}^i \log \hat{p}^i.
\]
(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 until all branches are pure.

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

In practice, use a threshold $\theta$.

Don’t split any node with impurity $< \theta$.

For leaf nodes – output the posterior probability $P(C_i | m)$ – or output MAP $\hat{C}_i = \arg\max P(C_i | m)$. 
Regression Trees

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

Node \( m \), \( X_m \) subset of \( X \) reaching node \( m \).

Define \( b_m(x) = \begin{cases} \frac{1}{|X_m|} & \text{if } x \in X_m; \text{ } x \text{ real node} \\ 0 & \text{otherwise} \end{cases} \)

\( g_m \) is the estimate value in node \( m \).

\[
E_m = \frac{1}{N_m} \sum_{t} (r_t - g_m)^2 b_m(x^t)
\]

\( N_m = \sum_{t} b_m(x^t) \)

Select \( g_m = \frac{1}{E_m} \sum_{t} b_m(x^t) r_t \)

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

(Create a piecewise approximation with)

discontinuation at leaf boundaries.

If \( E_m \) is above threshold, split the node so as to decrease the error of the child nodes.

\( X_{m,j} \) subset of \( X_m \); taking branch \( j \); \( X_{m,j} = X \)

\[
b_{m,j}(x^t) = \begin{cases} \frac{1}{|X_{m,j}|} & \text{if } x \in X_{m,j}; \text{ } x \text{ real node and take branch } j \\ 0 & \text{otherwise} \end{cases}
\]

\( g_{m,j} = \frac{1}{E_{m,j}} \sum_{t} b_{m,j}(x^t) r_t \)

\[
E_{m,j} = \frac{1}{N_{m,j}} \sum_{t} (r_t - g_{m,j})^2 b_{m,j}(x^t)
\]

- error after split.
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 sub-trees 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 sub-tree and keep the leaf node.

Pre-pruning is faster - back post-pruning usually gives more accurate 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-SET.

\[
\begin{align*}
X_1 &: \text{Age} \\
X_2 &: \text{Yes on job} \\
X_3 &: \text{Gender} \\
X_4 &: \text{Job Type = B} \\
\end{align*}
\]

\[
\begin{array}{c}
X > 38.5 \\
\text{Yes} \\
\text{No}
\end{array}
\]

\[
\begin{array}{c}
X_2 > 2.5 \\
\text{Yes} \\
\text{No}
\end{array}
\]

\[
\begin{array}{c}
0.9 \\
\text{Yes} \\
0.7 \\
\text{No}
\end{array}
\]

\[
\begin{array}{c}
0.6 \\
\text{Yes} \\
0.2 \\
\text{No}
\end{array}
\]

\[
\begin{align*}
K_1 &: (\text{age} > 38.5) \land (\text{yes on job} > 2.5) \quad \text{THEN } y = 0.8 \\
K_2 &: (\text{age} > 38.5) \land (\text{yes on job} < 2.5) \quad \text{THEN } y = 0.6 \\
K_3 &: (\text{age} \leq 38.5) \land (\text{job type} = A) \quad \text{THEN } y = 0.4 \\
K_4 &: \quad \land (\text{job type} = B) \quad \text{THEN } y = 0.7 \\
K_5 &: \quad \land (\text{job type} = C) \quad \text{THEN } y = 0.2
\end{align*}
\]

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 value.
This gives disjunction of conjunctions.
Rule induction - works similarly to tree induction but does depth-first search and generates one path at a time.

Example: RIPPER (see Alpaydin p. 188 for process)

Rules are added to explain positive examples

Rule R to rule R'.

Change in gain:

\[ G_{\text{gain}}(R; R') = s \cdot \left( \log \frac{N'}{N'} \right) - \left( \log \frac{N+N'}{N+N'} \right) \]

\( N \) instances covered by \( R \)  \( N' \) no. true positives

\( N' \) 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

\[ \text{rm} = \frac{p-n}{p+n} \]

\( p \) no. true pos

\( n \) no. false pos

\( p-n \) positive on pruning set
Proposed 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

$\text{fm} (x_1): \omega^T x + \omega_0 > 0$

At each non-leaf sphere

$\text{fm} (x_1): \|x - s\| \leq d_m$

Algorithm proceed as before.