Decision Trees

Divide and Conquer strategy for classification.

- **Tree.** Decision node w/ test function \( f(x) \)
- **Leaf node**

Each \( f(x) \) defines a discriminant in \( d \)-dim input space

Hierarchical structure

- 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.
- E.g., attributes - color, size
- \( f(x) : x_j > w_o \) - threshold value

Binary split:
- \( L_m = \{ x \mid x_j \geq w_o \} \)
- \( R_m = \{ x \mid x_j < w_o \} \)

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. instance 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
$$\hat{p}(C_i|m) = \frac{N^i_m}{N_m}.$$

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

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

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
$$\hat{p}(C_i|m,j) = \frac{N^i_{m,j}}{N_{m,j}}.$$

Impurity after split $I'_m = -\sum_{j=1}^n \frac{N_{m,j}}{N_m} \sum_{i=1}^K \frac{N^i_{m,j}}{N_{m,j}} \hat{p}^i_j \log \hat{p}^i_j$. 
Classificatio & 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 leaves risks overfitting the training data.

In practice, use a threshold \( \theta \) to:

- Don't split any node with impurity < \( \theta \).

For leaf nodes, output the posterior probability \( P(C_i|y) \) - or output MAP \( \hat{C}_i = \arg\max P(C_i|y) \).
Regressin Trees

Sudier 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.

\( b_m(x) \) \( \begin{cases} 1 & \text{if } x \in X_m \\ 0 & \text{otherwise} \end{cases} \)

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

\[
E_m = \frac{1}{N_m} \sum_t \left( r^t - g_m \right)^2 b_m(x^t)
\]

\( N_m = |X_m| \sum_t b_m(x^t) \)

Select \( g_m = \frac{\sum_t b_m(x^t) r^t}{\sum_t b_m(x^t)} \)

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

(Created a piecewise approximations with discontinuities at leaf boundaries.

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

\( X_{mj} \) subset of \( X_m \) taking branch j; \( U_j = X_m - X_{mj} \)

\( b_{mj}(x) \) \( \begin{cases} 1 & \text{if } x \in X_{mj} \\ 0 & \text{otherwise} \end{cases} \)

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

\[
E_m' = \frac{1}{N_{mj}} \sum_t \left( r^t - g_{mj} \right)^2 b_{mj}(x^t)
\]
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 fiddle sub-trees that cause overfitting.

To do so - obtaining pruning set of instances.

For each sub-tree, replace it by a leaf node labeled with training instances covered by sub-tree.

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 - but post-pruning usually gives more accurate trees.
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-SET.

\[
\begin{align*}
X_1 &: \text{Age} \\
X_2 &: \text{Yes or no job} \\
X_3 &: \text{Gender} \\
X_4 &: \text{Job Type}
\end{align*}
\]

\[
\begin{align*}
K_1 &: \text{IF (age > 38.5) AND (yes-or-no-job < 25) THEN } y = 0.8 \\
K_2 &: \text{IF (age > 38.5) AND (yes-or-no-job < 25) THEN } y = 0.6 \\
K_3 &: \text{IF (age < 38.5) AND (job-type = A) THEN } y = 0.7 \\
K_4 &: \text{AND (job-type = B) THEN } y = 0.7 \\
K_5 &: \text{AND (job-type = C) THEN } y = 0.2
\end{align*}
\]

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 pseudo code)

Rules are added to explain positive examples. Rule R to rule R'.

Change in gain:

\[ \text{Gain}(R'/R) = s \cdot \left( \frac{\log N_+}{N_+} - \frac{\log N_+}{N} \right) \]

\( N \) instances covered by \( R \), \( N_+ \) no. true pos for \( N \); \( N_+ \) 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 \( \text{run} = \frac{p-n}{p+n} \)

\( p \) — no. true 1

\( n \) — no. false

prune 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

\[ f(x) = \omega^T x + \omega_0 > 0 \]

At location on the sphere area

\[ f(x) = \| x - s \|_2 \leq d \]

Algorithm proceed as before.