3 Fundamental Algorithms (part 3 of 5): Decision Trees

A decision tree is a directed acyclic graph like a flowchart used to decide \( y = 0 \) or \( y = 1 \) from \( x \).

- To use a tree, at each node, if the value of some feature \( j \) is less than a threshold, the left branch is followed; otherwise the right branch is followed. (See the figure below.)
- To build a tree, at each node, we choose the feature and threshold on which to split by minimizing the cost associated with the split (below).

Decision Tree Classification

Information Content and Entropy

The information content, also known as self-information and surprisal, of an outcome \( x \) of a random variable \( X \) is \( I(x) = \log_2 \frac{1}{P(x)} = -\log_2 P(x) \), where \( P(x) = P(X = x) \) quantifies the level of surprise at seeing \( x \) (in bits).

\[
I(x) = \begin{cases} 
\log_2 \frac{1}{P(x)} & \text{for } P(x) \neq 0, 1 \\
\epsilon & \text{for } P(x) = 0, 1
\end{cases}
\]

\( \epsilon \) is small

\( H(X) = \text{expected value of } I(X) = \sum_{i=1}^{n} P(x_i) \left[ -\log_2 P(x_i) \right] \)

- For a fair coin flip \( X \) with outcomes 0 and 1 (tails and heads) whose probabilities are \( \frac{1}{2} \) and \( \frac{1}{2} \), \( H(X) = \frac{1}{2} \log_2 2 + \frac{1}{2} \log_2 2 = 1 \)
- For an unfair coin flip \( X_{\text{usually heads}} \) with outcomes 0 and 1 whose probabilities are \( \frac{1}{100} \) and \( \frac{99}{100} \), \( H(X_{\text{usually heads}}) = \frac{1}{100} \log_2 100 + \frac{99}{100} \log_2 99 \approx 1.5 \)
- For an unfair coin flip \( X_{\text{always heads}} \) with outcomes 0 and 1 whose probabilities are 0 and 1 (so both sides are heads), \( H(X_{\text{always heads}}) = 0 \)

Note: Equiprobable outcomes maximize entropy, while a constant variable minimizes entropy.

- For the sum \( Y \) of two fair coin flips with outcomes 0, 1, and 2 whose probabilities are \( \frac{1}{4}, \frac{1}{2}, \) and \( \frac{1}{4} \), \( H(Y) = \frac{1}{4} \log_2 4 + \frac{1}{2} \log_2 2 + \frac{1}{4} \log_2 4 = 1 \)
- For the outcome \( Z \) of two fair coin flips with outcomes (0,0), (0,1), (1,0) and (1,1), whose probabilities are all \( \frac{1}{4} \), \( H(Z) = 1 \)
Learning a Classification Tree with the ID3 algorithm

Let $S$ be a set of training examples $\{(x, y)\}$, where each $y \in \{0, 1\}$. The start node contains $S$ and yields a constant model for $P(y = 1| x)$:

$$f_{ID3}(S) = \frac{1}{|S|} \sum_{(x, y) \in S} y,$$

the proportion of 1 values in $S$.

The entropy of a set of examples $S$ is the entropy of a random draw from $S$:

$$H(S) = \sum_{y \in \{0,1\}} P(y) [- \log_2 P(y)]$$

$$= P(0) [- \log_2 P(0)] + P(1) [- \log_2 P(1)]$$

$$= [1 - P(1)] (- \log_2 [1 - P(1)]) + P(1) [- \log_2 P(1)]$$

$$= -f_{ID3}(S) \log_2 f_{ID3}(S) - [1 - f_{ID3}(S)] \log_2 [1 - f_{ID3}(S)]$$

e.g. Confirm the entropy of a few nodes from the tree below.

To branch from a node containing $S$, consider all features $j = 1, \ldots, D$ and thresholds $t$ that partition $S$ into two subsets $S_-= \{(x, y) \in S | x^{(j)} \leq t\}$ and its complement $S_+= \{(x, y) \in S | x^{(j)} > t\}$ that make two new child nodes; choose the best pair $(j, t)$.

For ID3, the best subset pair is the one that minimizes the weighted average entropy of the split:

$$H(S_-, S_+) = \frac{|S_-|}{|S|} H(S_-) + \frac{|S_+|}{|S|} H(S_+)$$

We stop at a leaf if any of these are true:

- All examples in the leaf are classified correctly by the constant model.
- We cannot find a feature upon which to split.
- The split reduces entropy less than some $\epsilon$.
- The tree has reached some maximum depth $d$.

$\epsilon$ and $d$ are hyperparameters that we set experimentally.

---

1. There are several other decision tree algorithms; some can handle $y_i \in \mathbb{Z}$ and categorical $y_i$.
2. Burkov’s notation for $f_{ID3}(S)$ is $f_{ID3}^S$.
3. Burkov uses $<$ in $S_-$ and $\geq$ in $S_+$. I use $\leq$ in $S_-$ and $>$ in $S_+$ to match scikit-learn.
A note on context

Burkov asserts that this algorithm approximately maximizes the average log-likelihood:

\[
\frac{1}{N} \sum_{i=1}^{N} \left[ y_i \ln f_{ID3}(x_i) + (1 - y_i) \ln (1 - f_{ID3}(x_i)) \right].
\]

- In logistic regression \( f_{w^*,b^*} \) was the optimal solution for its parametric model.
- ID3 approximates a solution by building a nonparametric model \( f_{ID3}(x) = P(y = 1|x) \).
  It does not look ahead when branching, so it finds only a local maximum.

The most widely-used decision tree uses \( C4.5 \), an extension of ID3, which

- accepts continuous and discrete features
- handles incomplete examples
- addresses overfitting by bottom-up pruning

Python

- from sklearn.tree import DecisionTreeClassifier:
  - clf = DecisionTreeClassifier(criterion='entropy', max_depth=None,
    min_impurity_decrease=0)
    \((d = \text{max\_depth}, \epsilon = \text{min\_impurity\_decrease})\)
  - clf.fit(X, y) fits the classifier to array \( X_{N \times D} \) and \( y_{N \times 1} \)
  - clf.predict_proba(X)[:, 1] gives probabilities \( \{P(y_i = 1)\} \) \((\[:, 0\] gives \( \{P(y_i = 0)\})\)
  - clf.predict(X) uses a decision threshold to give predictions \( \{\hat{y}_i\} \) for examples in \( X \)
  - clf.score(X, y) gives the average accuracy on \( X \) with respect to \( y \)

- from sklearn import tree:
  - tree.plot_tree(clf, feature_names=None, filled=True) makes a graph of the tree

- from sklearn.tree import export_text:
  - print(export_text(clf, feature_names=None)) prints the tree as plain text

To learn more:

- Examples are at the bottom of the manual page.
Decision Tree Regression

If $y_i \in \mathbb{R}$, we can use a decision tree for regression. The prediction at a node containing $S$ is

$$\bar{y}_S = \frac{1}{|S|} \sum_{(x,y) \in S} y.$$  

To branch from a node, consider all features $j = 1, \ldots, D$ and thresholds $t$ that partition $S$ into $S_-$ and $S_+$ as before. The best subset pair is the one that minimizes the squared error associated with the split:

$$\sum_{(x,y) \in S_-} (y - \bar{y}_S_-)^2 + \sum_{(x,y) \in S_+} (y - \bar{y}_S_+)^2$$

where $\bar{y}_S_-$ and $\bar{y}_S_+$ are the means of the $y$ values in $\bar{y}_S_-$ and $\bar{y}_S_+$, respectively.

Python

- from sklearn.tree import DecisionTreeRegressor:
  - model = DecisionTreeRegressor(criterion='squared_error', max_depth=None)
  - model.score(X, y) gives $R^2$ (proportion of variability in $y$ accounted for by $X$)

To learn more:

- Reference manual: