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 $\mathbf{x}$ for classification or to model $y \in \mathbb{R}$ as a function of $\mathbf{x}$ for regression.

- 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 a 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*).

e.g. Draw $\log_2 p$ and $-\log_2 p$ for probability $p \in (0, 1]$:

- $P(x) = 1 \implies I(x) =$ _____________
- $P(x) = \frac{1}{2} \implies I(x) =$ _____________
- $P(x) = \epsilon$, where ϵ is small $\implies I(x) =$ _____________

The *entropy* of a random variable X with possible outcomes $x_1, \ldots, x_n$, a measure of uncertainty of X, is the expected value (weighted average) information content in *bits* given by its outcome:

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

e.g.

- For a fair coin flip X with outcomes 0 and 1 (tails and heads) whose probabilities are $\frac{1}{2}$ and $\frac{1}{2}$, $H(X) =$ _____________
- For an unfair coin flip $X_{\text{usually heads}}$ with outcomes 0 and 1 whose probabilities are $\frac{1}{100}$ and $\frac{99}{100}$, $H(X_{\text{usually heads}}) =$ _____________
- 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), $H(X_{\text{always heads}}) =$ _____________

  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 $\frac{1}{4}$, $H(Y) =$ _____________
- For the outcome Z of two fair coin flips with outcomes $(0, 0)$, $(0, 1)$, $(1, 0)$ and $(1, 1)$, whose probabilities are all $\frac{1}{4}$, $H(Z) =$ _____________

Learning a Classification Tree with the ID3 algorithm[1]

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

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

the average of the y values in S (for classification, it is also the proportion of 1 values).[2]

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

$$\begin{aligned} H(S) &= \sum_{y\in\{0,1\}} P(y)\,[-\log_2 P(y)] \\ &= P(0)\,[-\log_2 P(0)] + P(1)\,[-\log_2 P(1)] \\ &= [1 - P(1)]\,(-\log_2 [1 - P(1)]) + P(1)\,[-\log_2 P(1)] \\ &= -f_{ID3}(S) \log_2 f_{ID3}(S) - [1 - f_{ID3}(S)] \log_2 [1 - f_{ID3}(S)] \end{aligned}$$

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 $S_- = \{(\mathbf{x}, y) \in S|x^{(j)} \le t\}$ and its complement $S_+ = \{(\mathbf{x}, y) \in S|x^{(j)} > t\}$ that make two new child nodes; choose the best pair (j, t).[3]

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 ϵ.
- The tree has reached some maximum depth d.

ϵ 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^S_{ID3}.
[3] Burkov uses $<$ in S_- and $\ge$ in S_+. I use $\le$ in S_- and $>$ in S_+ to match scikit-learn.

A note on context (optional)

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

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

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

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

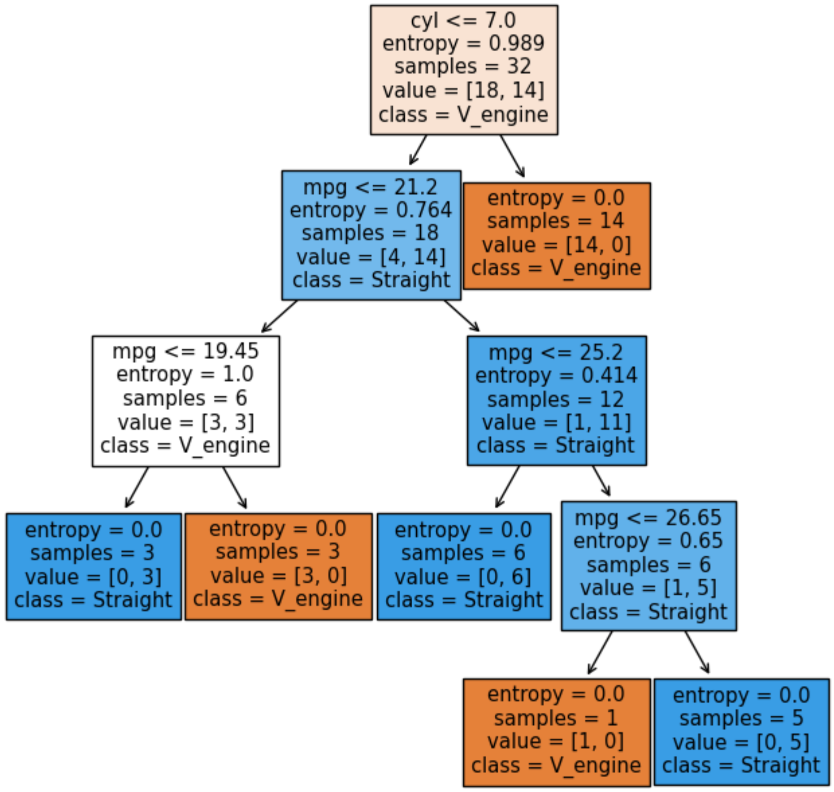
- accepts continuous and discrete features
- handles incomplete examples
- addresses overfitting by bottom-up *pruning*

Python

- `from sklearn.tree import DecisionTreeClassifier`:
  - `clf = DecisionTreeClassifier(criterion='entropy', max_depth=None, min_impurity_decrease=0)`
    (d = `max_depth`, ϵ = `min_impurity_decrease`)
  - `clf.fit(X, y)` fits the classifier to array $\mathtt{X}_{N\times D}$ and $\mathtt{y}_{N\times 1}$
  - `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:

- User guide: `https://scikit-learn.org/stable/modules/tree.html`
- Reference manual:
  `https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html`
- Examples are at the bottom of the manual page.



Decision Tree Regression

If $y \in \mathbb{R}$, we can use a decision tree for regression. The prediction at a node containing S is $\hat{y}_S = \bar{y}_S = \frac{1}{|S|} \sum_{(\mathbf{x},y)\in S} y$. 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:

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

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:
  `https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html`