Dataset

We are using the UC Irvine Mushrooms dataset.

Methods

Inductive Learning

Given target function `f`, the training inputs are tuples `(x, f(x))`. The goal is to learn a function `h` that approximates `f` as much as possible, using the training data.

Drawbacks

- ignores prior knowledge
- requires examples to train on

Learning Decision Trees

The goal is to find a small tree consistent with example inputs. We can do this by recursively choosing the most significant attribute as the root of each subtree.

```
function DTL(examples, attributes, defaultValue) {
    if (examples.isEmpty()) return defaultValue;
    if (examples.allClassificationsMatch()) return examples[0].classification;
    if (attributes.isEmpty()) return mode(examples);

    // a good choose attribute function will reduce uncertainty as much
    // as possible
    let best = ChooseAttribute(attributes, examples);
    let tree = new DecisionTree({rootAttr: best});

    for (const val_i of best) {
        // examples_i is the set of examples with the attribute best matching
val_i
        let examples_i = examples.filter((ex) -> ex.attribute(best) == val_i);
        let subtree = DTL(examples_i, attributes - best, mode(examples));
        tree.addBranch({label: val_i, subtree: subtree});
    }
}
```

The above function leaves just one question: how to choose the best attribute. Per the comment, we want to reduce uncertainty. That means we want to reduce entropy as much as possible. Entropy is defined as

$$I(P(v_1), P(v_2), \dots P(v_n)) = -\sum_{i=1} P(v_i) \log_2 P(v_i)$$

We also need the entropy across all possible branches for the attribute, so we weight them according to their probabilities:

$$AvgEntropy = \sum_{i=1}^{n} \left(\frac{p_i + n_i}{p + n} \times I\left(\frac{p_i}{p_i + p_n}, \frac{n_i}{p_i + p_n} \right) \right)$$

where p_i is the number of times we choose "yes" after taking branch i, and n_i is the number of times we choose "no" after taking branch i. p and n are the total across all branches.

Performance Evaluation