1 Module 14: Decision Trees

Contents

1 Module 14: Decision Trees **1**
1.1 Basics 1
1.2 Entropy 2
1.3 Overfitting 3
1.4 Bagging 3

1.1 Basics

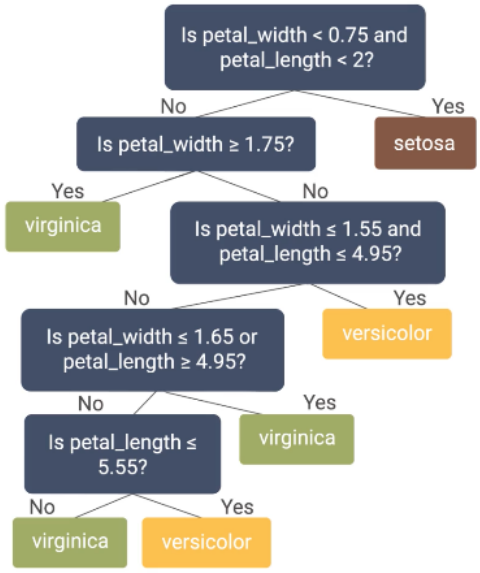
Decision trees benefit from...

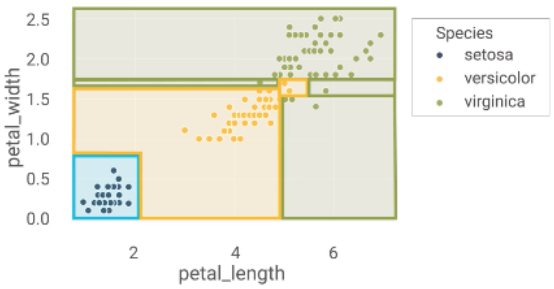
- being very interpretable
- require much less training data
- capable of intaking categorical data
- handle collinearity well

Composition
Root node: beginning of tree
Internal nodes: can be split further (2 or more branches)
Leaf nodes: can no longer be split
Impure nodes: leaf nodes which do not encapsulate a single class (have multiple classes w/i)

Algorithm

- Determine root node
- Calculate class entropy
- Calculate entropy for each attribute after split
- Calculate information gain from each split
- Perform splits until decision tree is complete
- Assess accuracy of decision tree






```
from sklearn import tree

model = tree.DecisionTreeClassifier(criterion = 'entropy')
model.fit(X, y)

# Visualize tree thru sklearn
tree.plot_tree( model, feature_names = list, class_names = list, rounded =
↪  True, filled = True )

# Viz thru Graphviz
import graphviz
data = tree.export_graphviz( model, out_file = None, feature_names = list,
↪  class_names = list, rounded = True, filled = True )
graph = graphviz.Source(data)
graph.render( format = 'png', filename = 'iris_tree' )
```

1.2 Entropy

$$S = -\sum_c p_c \log_2 p_c$$

for proportion of each class c: p_c

Entropy cases:

$-1\log_2 1 = 0$: All data in node is part of same class

$-0.5\log_2 0.5 - 0.5\log_2 0.5 = 1$: Data is evenly split between two classes

$3 \times -0.33\log_2 0.33 = 1.58$: Evenly split between three classes
... $C \times -\frac{1}{C}\log_2\frac{1}{C} = \log_2 C$: Evenly split between C classes

Weighted Entropy: entropy × fraction of samples in that node, s.t. weighted entropy decreases at each level

We use weighted entropy to decide which split to use – we want the highest change in weighted entropy.

1.3 Overfitting

For decision trees, more features ≠ overfitting

To avoid overfitting, restrict decision tree complexity. **Prevent (unnecessary) growth**

- Disallow splitting after a sample threshold i.e. ¡1% using min_sample_split
- Cap node depth using max_depth
- Do not create splits where weighted entropy is too small i.e. $\delta WS < 0.01$

Pruning: Allow tree to grow & cut branches afterward

- Use validation set to prune – run model with & without branch

1.4 Bagging

Especially with "black box" estimators, one can draw "random" subsets of the dataset, fit them individually, and return an aggregate (either by voting or averaging) final prediction which can reduce variance substantially.

```
from sklearn.ensemble import BaggingRegressor

BaggingRegressor(estimator = e.g. DecisionTreeClassifier,
    n_estimators:int (default = 10) # Number of base estimators in ensemble
    max_samples:int/float, max_features:int/float
    # Number of samples/features to draw from X
    bootstrap:bool # For samples, whether they are drawn with replacement;
    # Bagging is with replacement (default)
    # Pasting is drawing random subsets of the samples
    bootstrap_features:bool #Same principle with features
)
```