

The background features abstract, overlapping green geometric shapes, primarily triangles and polygons, in various shades of green, creating a modern and dynamic visual effect.

# Chapter 12

# Learning with Trees

Sections 12.2.4-12.4, pp. 255-265

# Section 12.2.4

Implementation of the Decision Tree

## Implementation of the Decision Tree

- If we run out of data or features, pick the class with the most occurrences
- If we only have 1 class left in our data, pick that class
- Calculate information gain for each feature as compared to rest
- Pick feature with maximum information gain
- Loop through all possible values and all datapoints in data
- If our datapoint agrees with the rules, keep it
- Remove the best feature from the remaining data
- Pass on the remaining data, classes and features to next recursive iteration
- Make the tree

```
def make_tree(data, classes, featureNames):  
    # Various initialisations suppressed  
    default = classes[np.argmax(frequency)]  
    if nData==0 or nFeatures == 0:  
        # Have reached an empty branch  
        return default  
    elif classes.count(classes[0]) == nData:  
        # Only 1 class remains  
        return classes[0]  
    else:  
        # Choose which feature is best  
        gain = np.zeros(nFeatures)  
        for feature in range(nFeatures):  
            g = calc_info_gain(data, classes, feature)  
            gain[feature] = totalEntropy - g  
        bestFeature = np.argmax(gain)  
        tree = {featureNames[bestFeature]:{}}  
        # Find the possible feature values  
        for value in values:  
            # Find the datapoints with each feature value  
            for datapoint in data:  
                if datapoint[bestFeature]==value:  
                    if bestFeature==0:  
                        datapoint = datapoint[1:]  
                        newNames = featureNames[1:]  
                    elif bestFeature==nFeatures:  
                        datapoint = datapoint[:-1]  
                        newNames = featureNames[:-1]  
                    else:  
                        datapoint = datapoint[:bestFeature]  
                        datapoint.extend(datapoint[bestFeature+1:])  
                        newNames = featureNames[:bestFeature]  
                        newNames.extend(featureNames[bestFeature+1:])  
                    newData.append(datapoint)  
                    newClasses.append(classes[index])  
                    index += 1  
            # Now recurse to the next level  
            subtree = make_tree(newData, newClasses, newNames)  
            # And on returning, add the subtree on to the tree  
            tree[featureNames[bestFeature]][value] = subtree  
        return tree
```

# Implementation of the Decision Tree

- ▶ Training to testing set generalizability?
  - ▶ Inductive bias
    - ▶ Minimising the amount of information left over to be passed to next node
    - ▶ Maximising entropy means producing an equal a split as possible between classes in dataset
    - ▶ Tendency towards smaller trees
      - ▶ Occam's Razor
      - ▶ KISS (Keep it simple, stupid)
      - ▶ MDL (Minimum Description Length) *Rissanen 1989*
  - ▶ Dataset Noise
    - ▶ Class selection at leaf nodes are based on majority population
      - ▶ Only works well if you have much more sample counts than feature counts
    - ▶ Causes overfitting regardless
      - ▶ Early leaf end condition: must have only 1 class left
      - ▶ Continued formulation of nodes when it should be a leaf
  - ▶ Missing Data
    - ▶ Unique benefit: Assume a test sample passes through every edge and sum across all resulting paths taken

# Implementation of the Decision Tree

- ▶ Issue: All features must be used in tree construction
  - ▶ Overfitting risks
  - ▶ Solution?
    - ▶ Maximum tree size (max levels)
    - ▶ Early stopping via validation set, rate of improvement
    - ▶ Pruning
- ▶ Pruning
  - ▶ Naïve pruning
    - ▶ Replace nodes with most common class in that sub-tree
  - ▶ C4.5 (rule post-pruning)
    - ▶ Convert to flat set of if-then rules
    - ▶ Remove preconditions if it improves accuracy
      - ▶ Preconditions are if rules between the root if condition and final if condition in a path
    - ▶ Sort remaining rules in order of “estimated accuracy”
      - ▶ Something about lower CI-95% of observed accuracy minus  $1.96 * S.D$

# Section 12.2.6

Computational Complexity

# Computational Complexity

- ▶ Construction
  - ▶  $\mathcal{O}(N \log N)$
- ▶ Balanced Binary Tree Prediction
  - ▶  $\mathcal{O}(\log N)$
- ▶ Unbalanced Binary Tree Prediction
  - ▶ Actually very complicated!
  - ▶ Max possible complexity:
    - ▶  $\mathcal{O}(N)$
    - ▶ Not very useful though

# Section 12.2.5

Dealing with Continuous Variables



# Dealing with Continuous Variables

- ▶ Discretization
  - ▶ Convert into a categorical variable
    - ▶ Randomly choose split points
      - ▶ Take each point as a unique variable in category
      - ▶ Calculate entropy as usual and pick best split point
  - ▶ Much more computationally expensive
- ▶ Mentions multivariate trees
  - ▶ Choose split planes on  $>1$  dimensions
    - ▶ Non-orthogonal split planes
      - ▶ Univariate trees are actually very bad, hill-climbing learned, LDA?

# Section 12.3.1

CART (Classification and Regression Trees)

Gini Impurity

# Gini Impurity

- ▶ Variation of entropy information measure
- ▶ Maximise purity, minimise impurity
  - ▶ Ability of each leaf node to separate a set of samples into sets of the same class

$$G_k = 1 - \sum_{i=1}^c N(i)^2$$

- ▶ Where  $N(i)$  is the fraction of datapoints belonging to class  $i$  in a node.
- ▶ Equivalent to expected error rate if prediction was based purely on the class distribution.
- ▶ Variation: Weighted Gini Impurity
  - ▶ Useful for future topic on boosting in random forests

$$G_i = \sum_{j \neq i} \lambda_{ij} N(i) N(j)$$

# Section 12.3.2

## Regression in Trees

# Regression in Trees

- ▶ Use sum-of-squares error
- ▶ Output value is just average of all datapoints in leaf node
- ▶ For each feature
  - ▶ Choose a split point that minimises sum-of-squares error
  - ▶ Select feature who's split point provides the most minimisation
  - ▶ Back to normal decision tree construction