Decision Trees

UW ECE 657A - Core Topic

Mark Crowley

Lecture Outline

Lecture Outline

Motivation

- Metric Classification: k-nearest neighbours, SVM, logistic regression, neural networks
- What if the data features are not numeric (discrete or continuous) values with a well defined order?
- How to perform classification of data for `color`$\in$\{'red', 'blue','orange'\}, `DNA`$\in$\{A,G,C,T\}
- How to discriminate data with such labels, How to learn a flexible model that can predict such labels without just learning a mapping. How to make it generalizable?

Idea: A Tree of Twenty Questions

Claim: *Any* finite pattern can be represented by a finite series of yes/no questions.

Lecture Outline

Decision Tree Definition

A **Decision Tree** is defined as:

- Directed graph $G =< N, E >$
- Each **node** $n \in N$ represents a choice on a particular variable/feature F amongst B subsets of the values of F.
- **Edges** (links, branches) connect a node to child (descendent) nodes which model a split on another feature (or the same feature with different split subsets)
- **Leaf** nodes are nodes with no children which have a category label assigned to them.
- The leaves form a *partition* of the data.
- A decision tree categorizes a data point according to the label of the leaf node that data point reaches by following the rules defined in the internal nodes.

Properties of Decision Trees

- A decision tree recursively partitions the input space, defining a local model in each resulting region of the input space represented by a leaf node.
- If all the dimensions are numerical we can envision what the decision tree is doing as dividing up space into rectangles.
- Decision trees allow us to combine Numerical and non-numerical data dimensions into a single learning process.

The Challenge : How to Split Nodes

- The question at each node splits the data along that dimension
- Number of split choices is the **branching factor** of the tree.
- We can represent any multiple branch split with a series of binary splits.
- Using a higher branching factor raises the risk of overfitting.
- Finding the optimal partitioning of the data is *NP-complete* so we usually use a greedy approach to find an approximate solution.

Lecture Outline

The CART framework

A generalization of decision trees is the *Classification and Regresssion Tree (CART)* framework. Questions we need to answer:

1. Should properties be binary only?
2. Which feature will be tested at each node?
3. When should a node become a leaf?
4. Can a tree be "too large", how could we make it smaller?
5. If a leaf is impure, what label do we assign?
6. How do we deal with missing data?

Magically answer all those questions

Note: no magic allowed

- Once you have satisfactory answers to those questions, then you have enough specification to implement and run an algorithm.
- For example, a very basic algorithm `fitTree` is described in *(Murphy 2012)*.

Basic `fitTree` Algorithm

This is a way to look at the general meta-algorithm for using trees to build classifiers.

Algorithm 1: `fitTree`$(node, D, depth)$

```
node.prediction = mean(y_i : i ∈ D) OR label distribution
j, t, D_L, D_R = split(D)
if not worthSplitting(depth, cost(D), D_L, D_R) then
    return node
else
    node.testfeature = j
    node.testvalue = t
    node.left = fitTree(node, D_L, depth+1)
    node.right = fitTree(node, D_R, depth+1)
    return node
```

From (Murphy, 2012).

Lecture Outline

Evaluating Your Choice

Cost as Node Impurity

Basic principle, we want the simplest model at each node, i.e. Occam's razor.
A leaf node is **pure** if all the datapoints associated with it are in have the same label/category.
(We'll say the *impurity* $cost(D) = 0$ in this case)

- Misclassification Impurity - intuitive but generally not as good as others
- Entropy Impurity - most popular
- Gini Impurity - useful for training trees

Misclassification Impurity

$$cost(D) = 1 - \max_j P(\omega_j)$$

Measures the *minimum* probability that a datapoint would be miscalssified for this node.
Very peaked at uniform probability of all labels.
But isn't smooth, derivative is discontuous so breaks some gradient search methods.

Entropy Impurity

$$cost(D) = -\sum_{c} P(\omega_c) \log_2 P(\omega_c)$$

where:

- $P(\omega_c)$ is the fraction (or probability) of points in that node having label ω_c

$$P(\omega_c) = \frac{1}{|D|} \sum_{i \in D} \mathcal{I}(y_i = c)$$

- If all points have same label then $cost(D) = 0$
- The maximum of $cost(D)$ will be when points have uniform likelihood of all labels

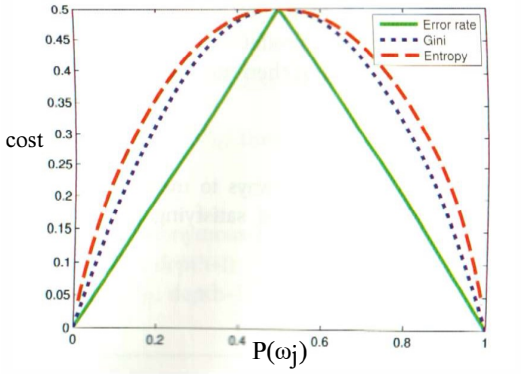
Gini Impurity

Also called Gini Index

$$cost(D) = \sum_{i \neq j} P(\omega_i)P(\omega_j) = \frac{1}{2}\left[1 - \sum_j P^2(\omega_j)\right]$$

- This is the expected error rate at node D if the label is selected randomly from the distribution present in that node.
- More strongly peaked at uniform probability than entropy impurity

Behaviour of Impurity Models



- For the binary two-class case. Entropy and Gini impurity measures very similar.
- Both are more sensitive to change in class probability than misclassifiaction rate.

Choosing a Split Value

- Given a partial tree, deciding on node D, chosen the feature to split on
- **Question:** What value should you split on?

$$\Delta cost(D) = cost(D) - \left(\frac{|D_L|}{|D|} cost(D_L) + \frac{|D_R|}{|D|} cost(D_R) \right)$$

where:

- D_L and D_R are the left/right descendent nodes and by splitting on feature T at value s.
- $|D|$ means the number of nodes in the set

The best split is the one that maximizes $\Delta cost(D)$
For entropy impurity this amounts to maximizing the **information gain** of the split.

When to Stop Splitting?

When should we stop splitting?

- **Simplest approach:** keep splitting until each leaf has only datapoints in the same class or has only a single datapoint. $\rightarrow$ Tends to overfit for classification/regression.
- **Minimize Cross-Validated Error:** measure the decrease in error a new node would cause against a validation set, or on many random cross-validation sets. Then stop splitting when the gain is minimized sufficient.
- **Threshold:** Stop splitting if proposed split provides $< \epsilon$ reduction in impurity, or if leaf has less than 5 datapoints.
- **Minimize Global Criterion:** ...

Minimize A Global Criterion

Define a global criterion. Keep splitting until it reaches a minimum value.

$$\alpha \cdot size + \sum_{d \in leafnodes(N)} cost(d)$$

where:

- *size* - number of nodes, or edges
- α - some positive constant

Properties:

- This acts as a **regularizer** that discourages larger trees.
- If *entropy impurity* is being used this is equivalent to minimizing the **minimum description length**, the number of bits used to represent the model.
- The sum of leaf impurities is a *measure of uncertainty* in the training data given the current model.

Pruning the Tree

Horizon Effect: what if you stop splitting at a node but later features would have led to better classification?

- Can't see past horizon unless you explore it.
- Stop too early and the model has low predictive power.
- Stop too late and it overfits.
- So why not fully grow the tree first then decide what to remove?

Basic Pruning Algorithm

1. Grow a full tree : until leaf node impurity hits a minimum or very small number of datapoints in leaf.
2. Find a pair of sibling leaf nodes i and j which have not yet been examined
3. Calculate how much impurity of parent node k would go up if i and j were eliminated then remove branches that would cause the smallest increase in error.

Benefits of Pruning

PROs

- All the data can be used for training unlike the threshold cross validation approaches
- Avoids the horizon effect

CONs

- More expensive than computing when to stop early

Another way to think about CART

CART can be see as an adaptive basis-function (kernal) model. The basis function defines regions as hyper-rectangles and weights specify the regression response or the matching label proportions for each region.

$$f(x) = E[y|x] = \sum_{m=1}^{M} w_m \mathcal{I}(\mathrm{x} \in R_m) = \sum_{m=1}^{M} w_m \phi(\mathrm{x}; \mathrm{v}_m)$$

where:

- R_m is the m^{th} regions partitioned by the tree
- v_m encodes the choice of feature and the threshold value to split on
- $\mathcal{I}$ is an indicator function that is 1 iff datapoint x falls within that hyper-rectangular region

From (Murphy, 2012)

Decision Tree Algorithm: ID3

Intended for use with nominal, unordered data.

- On each iteration, it iterates through every unused feature and calculates the entropy for that attribute.
- It then selects the one with the smallest entropy and the dataset is split on that feature
- Creates a branch for *each value* of the current feature (not binary!)
- The algorithm continues to recurse on each subset, considering only features never selected before.
- Stopping:
  - every element in the subset belongs to the same class
  - no more feautres left, if leaf not pure then the *most common* class label is used
- ID3 is only prefferred if computational costs are a big issue.

Decision Tree Algorithm: C4.5

Improvement on ID3, most popular in use (see WEKA data mining tool).

- Performs pruning on the tree after it's grown
- Missing Data: what if some datapoints don't have a value for feature f? C4.5 adds a '?' and doesn't use that feature for entropy calculations, algorithm otherwise stays the same

Pros and Cons of Decision Trees

PRO:

- Simple to implement, Lots of flexibility in impurity measures, training algorithms
- Resulting model is easy to interpret as logical rules
- Can be seen as an automated kernal learning method, similar to Neural Networks
- Universal expressive power (also like neural networks)
- Handle nominal, discrete and continous data
- They perform automated variable selection

CON:

- Very easy to overfit the data, create a complete mapping
- Tend to be *unstable*, small changes in input lead to very different trees
- Fairly slow to train (not as bad as Neural networks though)
- Don't perform as well as more modern methods (but...)