The AI-ML and Data Science Club of IITH

Decision Trees

Contents

Compiled by : Srinith

1 What is Decision Tree?

Decision Tree is one of the **supervised learning**[1] algorithms of Machine Learning which is used for regression and classification ,which create yes/no questions and continually split the data set until you isolate all data points belonging to each class . At the end all the leaf nodes dont split the data further , belong to a particular class.

2 How Decision Tree works?

A decision tree consists of two types of nodes: decision nodes and leaf nodes.Decision nodes are the nodes where decisions are taken based on the input variables. Leaf nodes represent the final output of the model. Decision tree uses a flow chart type of structure. The algorithm follows the following steps ,

- It starts with the root node , then selects the best feature and best threshold which classifies our target
- Then we look at the best feature in the remaining features and continue branching , until we find a perfect classification , i.e, no need for further branching from that nodes, which are leaf nodes.
- So , recursively we are creating sub trees until there is no need to branch the nodes.

We consider all the features and thresholds for the splits , but consider the ones which give least cost value (will be seen further).At each step we are trying to minimise some cost value , so we could say that Decision tree is going to follow Greedy , Top-Down , Recursive Partitioning.

3 How to choose Splits?

The next question we would like to ask is how to choose the best splits. We can use the concept of Information Gain to do that. Information gain , is a measure of the reduction in entropy (uncertainty) achieved by splitting the data on a particular feature. Entropy(E) tells us the amount of uncertainty or randomness in the data or the purity of that node i.e.

$$E = -\sum_{i=1}^{k} p(y=i) \log_k p(y=i)$$

where , $p(y=i) = \dfrac{\text{number of data points of } i^{th}\text{class}}{\text{total number of data points}}$.
The parameter information gain is caluclated as the difference between the entropy of the original set and the weighted average entropy of the subsets generated by the split. The higher the information gain, the more informative the attribute or feature is for the classification task.

$$IG = E(parent) - \sum_{i=1}^{k} w_i E(child_i)$$

Here we have considered entropy to kind of find the impurity of a node , we can also use the concept of Gini index instead of entropy . The Gini index is a measure of impurity or diversity

[1]Machines are trained with the data that is **tagged** with the correct output, which is used to predict the output

in a set of data. It calculates the probability of misclassifying a random sample element based on the distribution of classes in the set. The Gini index ranges from 0 (perfectly pure, i.e., all elements belong to the same class) to 1 (perfectly impure, i.e., the elements are evenly distributed across all classes).
Let there be N_j input samples in total for leaf j which are classified into m categories. Let n_{ji}, i belongs to [m] denote the number of samples classified into the ith category. The Gini impurity index of a leaf is given by

$$GI = 1 - \sum_{i=1}^{m} p_{ji}^2$$

Then the information gain function is modified as ,

$$IG = GI(parent) - \sum_{i=1}^{k} w_i GI(child_i)$$

4 Relation between Entropy and Gini Index

The Gini index and entropy are mathematically related and can be expressed in terms of each other. The Gini index (G) can be written as:

$$G = 2H - 1$$

where H is the normalized entropy of the distribution, which is given by:

$$H = -\sum \frac{[p(i)\log(p(i))]}{\log(n)}$$

where p(i) is the proportion of the population holding the ith value, and n is the total number of values in the distribution.

The Gini index is related to the normalized entropy through a linear transformation, and the normalized entropy can be written in terms of the Gini index and the entropy of the individual values in the distribution.

5 Decision Trees illustration

Let us take that we want to predict if a customer will buy a product or not based on two parameters his age and income. And let us take that in the dataset we have 50 percent of customers have bought the product and 50 percent didn't.
So , initially the entropy of datset is

$$E = -\frac{1}{2} * \log_2 \frac{1}{2} - \frac{1}{2} * \log_2 \frac{1}{2} = 1.$$

In case 1 , let us take that we split initially based on age i.e , age $\leq$20 and age>20 , and let the cut be as in table 1 and 2

the entropy and information gain are ,

$$E(age \leq 20) = -0.6 * log2(0.6) - 0.4 * log2(0.4) = 0.971$$
$$E(age > 20) = -0.4 * log2(0.4) - 0.6 * log2(0.6) = 0.971$$
$$\text{weighted entropy} = (2/5) * E(Age \leq 20) + (3/5) * E(Age > 20) = 0.971$$
$$\text{information gain} = \text{entropy(before) - weighted entropy(after)} = 0.029$$

Suppose we split on other parameter say income and then also we find the information gain , the more the information gain the more it can be used as a condition to split.

Age ≤ 20	Purchase
Yes	30
No	20

Table 1

Age > 20	Purchase
Yes	20
No	30

Table 2

6 Questions

Subjective :

1. What is the maximum number of nodes a binary decision tree can have if the input vector has L features?
2. What is the Time complexity of DT classfier?
3. List Down the possible domains where decision trees could be used.
4. Is Feature scaling required with decision trees ?
5. How to prevent overfitting in decision tree learning ?

7 Decision Trees with python

Few Decision Tree Algorithm functions written only using `numpy` (used iris dataset from the UCI Machine Learning Repository)

```
import numpy as np
```

```
def entropy(y):
  _, counts = np.unique(y, return_counts=True)
  p = counts / len(y)
  return -np.sum(p * np.log2(p))

  # Function to calculate information gain of a split
  def information_gain(X, y, feature, threshold):
      split_mask = X[:, feature] < threshold
      left_labels, right_labels = y[split_mask], y[~split_mask]
      parent_entropy = entropy(y)
      left_entropy = entropy(left_labels)
      right_entropy = entropy(right_labels)
      left_weight = len(left_labels) / len(y)
      right_weight = len(right_labels) / len(y)
      return parent_entropy - (left_weight * left_entropy) - (
         right_weight * right_entropy)
```

```
    # Building the tree recursively
    def build_tree(X, y, depth, max_depth):
        n_samples, n_features = X.shape
        n_labels = len(np.unique(y))

        # Check if node is pure or max depth is reached
        if n_labels == 1 or depth >= max_depth:
            return {'class': np.bincount(y).argmax()}

        # Find best split
        best_feature, best_threshold, best_info_gain = None, None, -1
        for feature in range(n_features):
            thresholds = np.unique(X[:, feature])
            for threshold in thresholds:
                info_gain = information_gain(X, y, feature, threshold)
                if info_gain > best_info_gain:
                    best_feature = feature
                    best_threshold = threshold
                    best_info_gain = info_gain

        # Split data and recursively build subtrees
        split_mask = X[:, best_feature] < best_threshold
        left_tree = build_tree(X[split_mask], y[split_mask], depth=
           depth+1, max_depth=max_depth)
        right_tree = build_tree(X[~split_mask], y[~split_mask], depth=
           depth+1, max_depth=max_depth)

        # Return decision node
        return {'feature': iris.columns[best_feature], 'threshold':
           best_threshold,
                'left': left_tree, 'right': right_tree}
```