Decision Trees

December 1, 2020

1 Introduction

Suppose you are making plans for winter break, and you would like to assess the health risk of getting together with family for the holidays. You might ask yourself a series of questions to ascertain the risk of contracting and spreading Covid-19. For example, you might ask:

1. Does anyone have symptoms of Covid-19 prior to the gathering?
2. How many people are gathering (0-5, 5-10, over 10)?
3. Will you be travelling outside of your current county?
4. Will you travel by yourself or need to use public transportation?

Perhaps after Thanksgiving break, you decide to collect data on the Covid-19 infection rate for people who travelled and visited their families and/or friends. One approach to formalizing your decision-making process it to construct a decision tree.

1.1 Definition: Decision tree

A decision tree is a classification method that "decides" which class a data sample belongs to based on its feature values. The model is a tree-like structure, where any path from root to a terminal node corresponds to a possible set of feature values. In other words, it is a mapping from set of feature values to a class label.

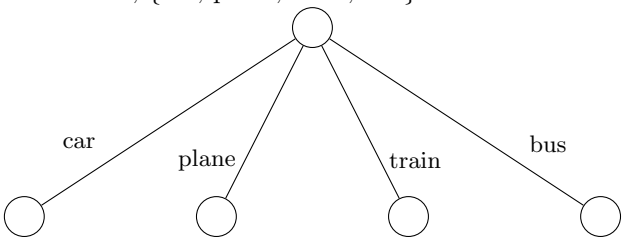
1.1.1 Features

Features can be both discreet and continuous values. For example, in our holiday gathering problem, we could include a discreet feature such as mode of transportation: {car, plane, train, bus}. Or, we could use a continuous feature such distance traveled. It is possible to use both types of features in a single decision tree.

```
# sample = [type of transportation, distance, number of people]
sample = ['car', 62.5, 5]
```

1.1.2 Internal Nodes

Each internal node executes a rule, or an instruction, associated with a particular feature. These rules determine which branch we take next. Consider an internal node corresponding to a feature that can takes discreet values, {car, plane, train, bus}.



When we arrive at an internal node, it "splits" the current path into k branches, so that we take a different path depending on our input value. We say that an internal node represents a **feature split**, that is a rule for splitting the path.

For a continuous feature, we might choose a threshold t for the split.

if value $\leq t$ **then**
    Take left branch
else
    Take right branch
end if

It can also be useful to reuse features for different internal nodes. For example, at one node we might ask for feature x is $x \leq 0$, and later on ask is $x \leq -10$.

1.1.3 Terminal Nodes

Terminal (leaf) nodes represent a predicted outcome. When we reach a terminal node, the data sample is classified.

2 Constructing a Decision Tree

We construct a decision tree using labeled training data. However, it is possible to make many trees from the same training data. To reduce space and time complexity, we want to minimize the depth of our decision tree, that is minimize the number of times we split our path. Thus we want a method for measuring the quality of a split, in order to prioritize better splits.

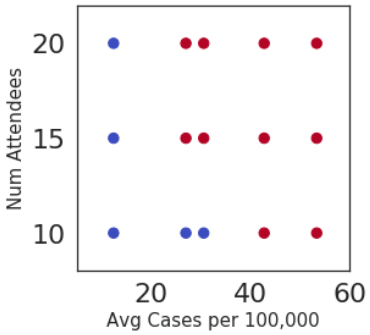
2.0.1 Impurity

We use **impurity** to measure the quality of a split. Generally speaking, impurity is the amount of dispersion or variability among the outcomes of a set of data points. Let's consider an example. Say we have the following data samples:

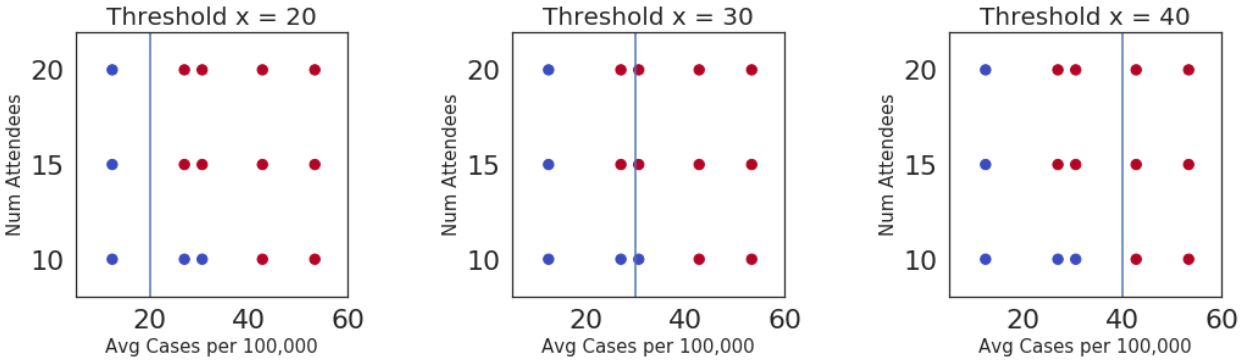
Num Attendees	County	Avg Daily Cases per 100,000	Risk
10	Alameda	12.4	.6
10	Santa Cruz	27	.11
10	Sacramento	30.6	.13
10	San Bernadino	53.4	.24
10	Los Angelos	42.8	.17
10	Alameda	12.4	.9
10	Santa Cruz	27	.16
10	Sacramento	30.6	.20
15	San Bernadino	53.4	.34
15	Los Angelos	42.8	.25
10	Alameda	12.4	.12
10	Santa Cruz	27	.20
10	Sacramento	30.6	.25
20	San Bernadino	53.4	.42
20	Los Angelos	42.8	.32

We define risk as risk that at least one person at the event has Covid-19. The data comes from this tool and NY Times.

In the scatter plots below, our x-axis is the current average daily cases (per 100,000 people) and the y-axis is number of attendees to an event. Blue labels denote ≤ 0.15 risk, and red, > 0.15 risk (these labels were chosen arbitrary for demonstration purposes, and do not reflect actual safety indications.)



Consider various thresholds to split the data along the x-axis.



When we choose a threshold, we split the data into two sets. Using splits that reduce impurity by the highest margins, we can create smaller shallower decision trees.

We measure the impurity of a data-set with an impurity function. In the example above, we have two classes, but suppose we have k. Let p_j be the probability that any point in the data-set belongs to class $j \in \{1, 2, \ldots, k\}$. An impurity function Im() takes as input these probabilities and returns a single value, upholding the following proprieties:

1. $\arg\max_{p_1,\ldots,p_k} \text{Im}(p_1, \ldots, p_k) = (\frac{1}{k}, \ldots, \frac{1}{k})$, i.e., the maximum value occurs for a uniform distribution
2. $\arg\min_{p_1,\ldots,p_k} \text{Im}(p_1, \ldots, p_k) = \{(1, 0, \ldots, 0), (0, 1, \ldots, 0), \ldots, (0, 0, \ldots, 1)\})$, i.e. minimum value occurs when a single $p_j = 1$ and all other $p_i = 0$
3. The function is symmetric, i.e. Im(p) is equivalent for any permutation of p

Where do these probabilities come from? When we are training our decision tree, we have labeled data, therefore we can compute the fraction of data samples that belong to each class. We then use these fractions as probabilities. Thus for example, to calculate the impurity of the data-set before we split, we let $p_{\text{Blue}} = \frac{5}{15}$ and $p_{\text{Red}} = \frac{10}{15}$. We then calculate Im($\frac{5}{15}, \frac{10}{15}$).

2.0.2 Entropy

One widely used impurity function is the entropy function, $H(X)$. Intuitively, entropy can be interpreted as a measure of expected uncertainty, information, or surprise in a random variable's outcomes. For a random variable X with n discrete outcomes and probability of each outcome p_i, we define entropy

$$H(X) = -\sum_{i=1}^{n} p_i \ln p_i$$

Entropy reaches its maximum when the probability space is uniform. In order to see this, we can use Jensen's inequality for a concave functions φ and random variable X

$$\varphi(\mathbb{E}[X]) \geq \mathbb{E}[\varphi(X)]$$

Let $\varphi(x) = \ln(x)$. We know that this is concave since $\frac{d^2\varphi(x)}{dx} = \frac{-1}{x^2} < 0$. Thus,

$$\ln \mathbb{E}[\frac{1}{p(X)}] \geq \mathbb{E}[\ln \frac{1}{p(X)}] = H(X)$$

Where $p(X)$ is the probability mass function for X. Then, we can see that the equality holds when $p(X)$ is constant, that is, when X is uniformly distributed.

Furthermore, if $p(X) = 1$, that is there is only one outcome, $H(X) = 0$. Simply put, we have the lowest entropy when there is no uncertainty.

2.0.3 Gini Index

Another common impurity function is the Gini Index,

$$\sum_{j=1}^{n} p_j(1 - p_j) = 1 - \sum_{j=1}^{n} p_j^2$$

Since the sum of our probability space is 1, and $x^2 \leq x$ when $x \leq 1$, we see that $\sum_{j=1}^{n} p_j^2 \leq \sum_{j=1}^{n} p_j = 1$. Therefore, we reach our minimum Gini Index when $1 - \sum_{j=1}^{n} p_j^2 = 1 - 1 = 0$. Heeding the impurity function properties, this only occurs when a single $p_j = 1$ and all other $p_i = 0$.

To compute the maximum, we also have to consider the constraints on the probabilities.

$$\max 1 - \sum_{j=1}^{n} p_j^2$$

$$\textbf{s.t } 1 - \sum_{j=1}^{n} p_j = 0$$

Taking the Lagrangian, we find the gradient of the probability vector,

$$\mathcal{L}(p, \lambda) = (1 - \sum_{j=1}^{n} p_j^2) - \lambda(1 - \sum_{j=1}^{n} p_j)$$

$$\nabla_{p_j} \mathcal{L}(p, \lambda) = -2p_j + \lambda = 0$$

Then, plugging our results back into the constraint,

$$1 = \sum_{j=1}^{n} p_j = \sum_{j=1}^{n} \frac{\lambda}{2}$$

We see that $\lambda = \frac{2}{n}$ and thus all $p_j = \frac{1}{n}$. Our maximum Gini Index is then $1 - n(\frac{1}{n^2}) = \frac{n-1}{n}$ for a uniform probability distribution.

2.0.4 Impurity Reduction

Once we have our impurity function, we need a way to measure the change in impurity before an after a split. Let V be the training data at a given node, V_a be the data assigned to the right branch, and V_b the data assigned to the left. Let $p(V_a)$ and $p(V_b)$ be the fraction of data in V_a and V_b respectively. We define the impurity reduction as:

$$R(V, V_a, V_b) = \text{Im}(V) - (p(V_a)\,\text{Im}(V_a) + p(V_b)\,\text{Im}(V_b)$$

When working with entropy, we can call this impurity reduction **information gain**, and can write a general formula for a random variable X and information a,

$$IG(X, a) = H(X) - H(X|a)$$

This simply means takes the difference of the current amount of entropy, and the entropy given the additional information.

An important property of both of these functions is **nonnegativity**. $R(V, V_a, V_b) \geq 0$. With information gain, you have a certain amount of information to begin with $H(X)$, so knowing some other information "a" should not decrease the amount of information you already know. At worst, it will be useless information that does not tell you anything more about X, e.g. it is information that is independent of X. .

3 Algorithm for Learning a Decision Tree

Finding the smallest decision tree is NP Hard, thus we cannot create an algorithm to deterministically find the optimal tree. However, we can define a function that considers all possible features given our training data at the current node, and chooses the split that reduces impurity the most.

The algorithm in pseudo-code is

```
1:  function DECISION-TREE-LEARN(data-set, outcomes)
2:      tree = new node
3:      if outcomes are all same then
4:          set tree label to outcome
5:          return tree
6:      else if data-set feature values are all same then
7:          set tree label to most frequent outcome
8:          return tree
9:      end if
10:     bestFeature = arg max_f R(f, data-set)
11:     for value v in bestFeature do
12:         subDataSet = data where value of bestFeature is v
13:         subOutcomes = outcomes for data in subDataSet
14:         subtree = DECISION-TREE-LEARN(subDataSet, subOutcomes)
15:         add child subtree to tree
16:     end for
17:     return tree
18: end function
```

Let's consider the base cases. The first is the case that our data samples at the given node all belong to the same class. In this case, we label our leaf node with the outcome (class label). In the second case, we might have samples that belong to different classes, however the samples themselves are the same, that is, for every feature they all share the same value. In order to deal with this situation, we pick the outcome that is most frequent among the data-samples.

On line 10 we say we select the best feature by choosing the feature that maximizes the reduction in impurity.

When does this algorithm end? It turns out that the algorithm runs until the decision-tree achieves 100% accuracy on training data, or near 100% in the case that identical samples have different labels. However, this can lead to **over-fitting**. We know from polynomial regression that increasing the dimension in order to reduce error for the training data can over-fit. Similarly, we may not want our decision-tree to run to completion. In order to fix this, we can use several strategies:

1. Place a limit on tree depth
2. Return a leaf node if the data is a minimum number of samples
3. Prune nodes after completing the tree to improve validation

4. Return a leaf node based on a statistical test on the data
5. Use multiple decision trees