Decision Trees

December 10, 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 is to construct a decision tree.

2   Review: Trees

In your data structures course, you learned how to construct and traverse **trees**. As a reminder, trees are structures that consist of a **set of nodes** and **set of edges** that connect the nodes. An essential property of trees is that there are no cycles and that there is only one path to reach each node. [1]

For example, a binary search tree made from the data-set $\{1, 2, 3, 4, 5, 6\}$ could look like
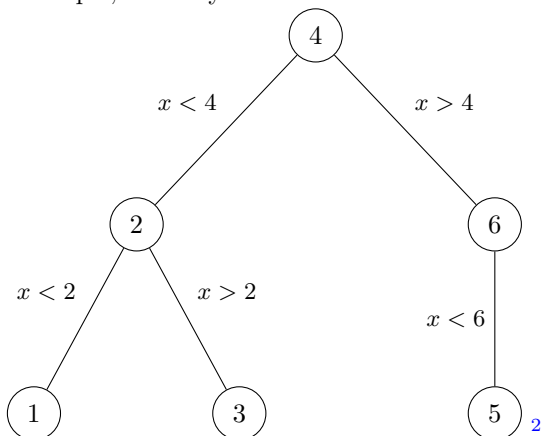
Recall how we perform binary search:

1. If our current node is the value we are looking for, then we return

---

[1]Trees and Graph Traversals Guide, CS 61B Fall 2020, https://fa20.datastructur.es/materials/lectures/lec23/lec23
[2]Binary Search Tree - GeeksforGeeks, https://www.geeksforgeeks.org/binary-search-tree-data-structure/

2. Else if our current node has no children, then our value is not in the tree

3. Else if our value is less than the current node, then take the left branch

4. If our value is greater than the current node, then take the right branch

3 Definition: Decision tree

Similarly, decision trees consist of edges and nodes that form a connected, acyclic graph. However, in contrast to binary search trees or other types of trees you may have encountered, our goal is to *classify* our input. We take in a data sample, and traverse the tree until we reach a **terminal node** (leaf node), which returns a class label.

3.0.1 Internal Nodes

Each internal (non-terminal) node executes a rule on the input data that tells us which branch to take next. You can think of an internal node as asking a question about our data sample. As an example, consider how you might construct a tree to help you determine whether you can safely travel home for winter break.
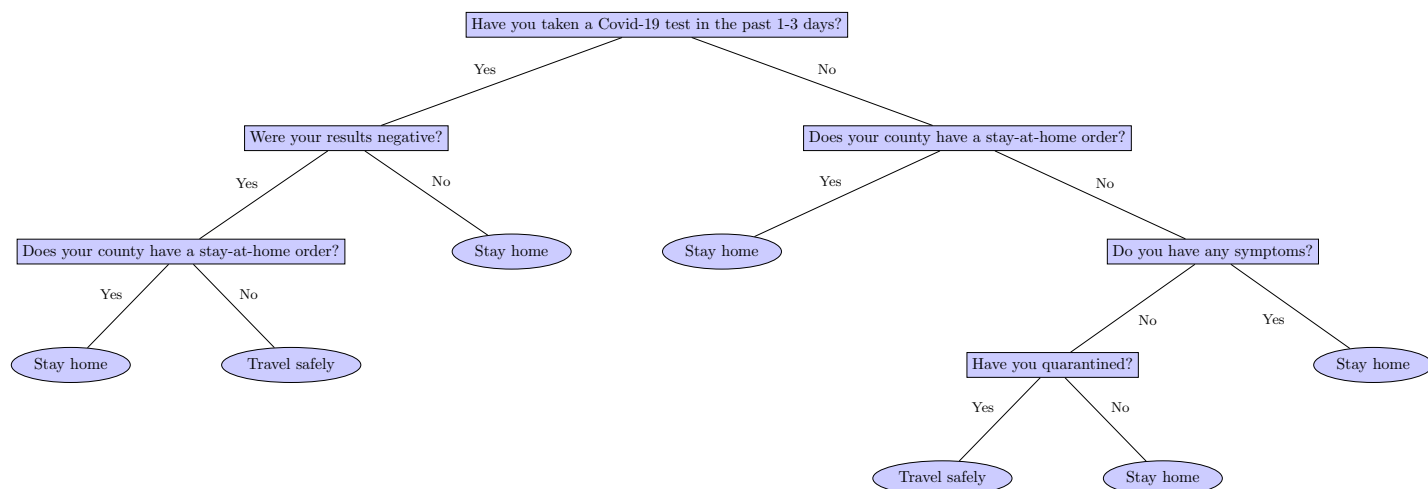

Figure 1: Decision-Tree Example

Internal nodes typically test on a single **feature** of the data. Similarly, in linear regression, we use weights that correspond to specific features. In the next section we discuss features for decision trees.

3.0.2 Features

In previous topics in this class, we learned that data points consist of one or more features. Features can be discrete or continuous — for example a person's body temperature is a continuous feature, while a positive or negative test result is discrete. Choosing (or having access to) useful features is essential to creating an accurate decision tree. If we have features that are uncorrelated to our class labels, then our decision tree will likely make poor predictions on our validation and test data.

When classifying an entire data set, we split our set into subsets at each internal node according to that node's rule. We call this rule a **feature split**. Consider the feature split for a continuous feature where t is a predefined threshold value.

```
    if value ≤ t then
        take left branch
    else
        take right branch
    end if
```

Decision trees can also use "reuse" features — in Fig 1, the question "Does your county have a stay-at-home order?" appeared in two different paths.

4 Constructing a Decision Tree

Given a labeled training data set, there are many decision trees you can build. We will introduce a greedy algorithm for decision tree learning. Can you think of how you might design such an algorithm?

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 feature split, in order to prioritize better splits.

4.1 Impurity

We can use **impurity** to measure the quality of a split. Generally speaking, statistical 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 the 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 at an event. Blue labels denote ≤ 0.15 risk, and red, > 0.15 risk (these labels were chosen arbitrarily for demonstration purposes, and do not reflect actual safety indications.)
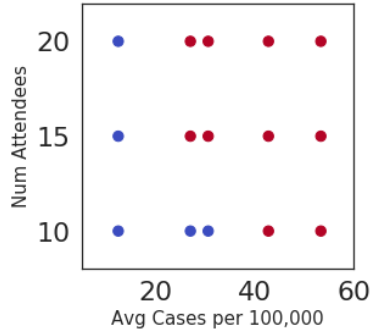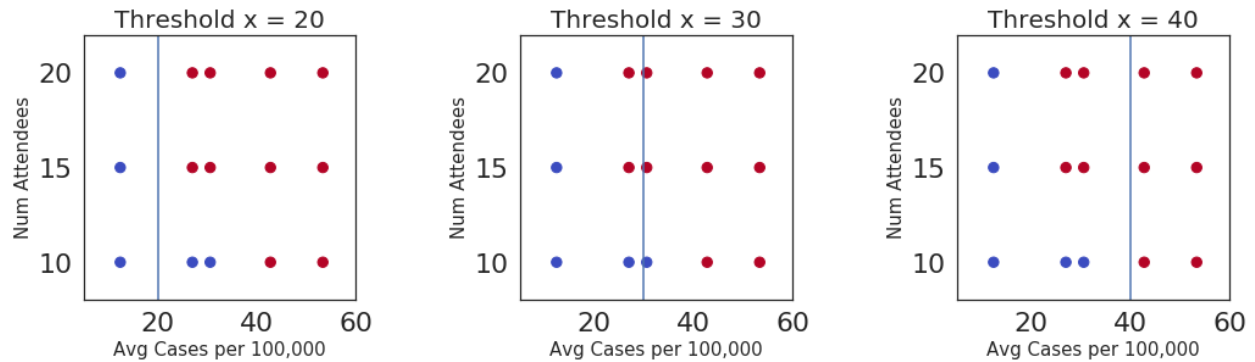
Figure 2: Scatter plot

Consider several thresholds used to split the data along the x-axis.

We can create smaller, shallower decision trees by specifying a threshold that reduces impurity the most.

The impurity of a data set is measured with an impurity function. Before we define this however, we'll review some probability fundamentals. If you are comfortable with probability, skip the next section.

4.2 Review: Probability Fundamentals

What is the probability that you get a 2 when you roll a fair die? You know from elementary school math that the answer is $\frac{1}{6}$, but how can you arrive at this probability more formally?

4.2.1 Probability of an event

A **sample space** Ω is the set of all possible outcomes in our experiment. In the above example, Ω consists of all possible rolls of the die, $\{1, 2, 3, 4, 5, 6\}$.

An **event** A is a subset of the possible outcomes. For example, we might be interested in the event that we roll an even number, which we can describe in words or in notation, e.g. define the event of rolling an even number as $A = \{2, 4, 6\}$.

A **probability distribution** is a function that maps an event to a value between 1 and 0 inclusive, which denotes the likelihood of the event occurring. For a discrete, uniform distribution, this function is,

$$p(A) = \frac{\text{size of } A}{\text{size of } \Omega} = \frac{|A|}{|\Omega|}$$

There are certain properties that are true for any probability distribution[3]

1. Nonnegativity: $p(A) \geq 0$

2. Additivity of events: $p(A \cup B) = p(A) + p(B) - p(A \cap B)$

3. Normalization: $p(\Omega) = 1$

In some cases we don't know the probability distribution, but we want to estimate the probability of an event using data. Suppose we have some labeled training data with k classes. Let our sample space be the k possible classes. If we randomly select a data sample from our set, what is the probability that it belongs to class $j \in \Omega$?. One way to estimate this is to use the fraction of samples belonging to class j.

$$p(A = \{j\}) = \frac{\text{number of samples labeled } j}{\text{number of total samples}}$$

You can verify for yourself that this satisfies each of the properties of a probability distribution.

4.2.2 Random Variables

Suppose we want to know the probability of each outcome in a sample space. As we have seen above, we can define an event $A_i = \{i\}$ for each outcome and find $p(A_i)$ according to our probability distribution. However, sometimes we have many (possibly infinitely many!) outcomes. Alternatively we can define a **random variable**, denoted as X, which takes on a numerical value associated with an outcome[4]. For example, let the random variable X be the number we get when we roll a die. We can then describe our probability distribution for any outcome x in our sample space as $p(X = x)$. There are various commonly studied probability distributions which can be described generally as functions of x (e.g. the uniform distribution, the normal distribution, etc.).

4.3 Impurity Function

Suppose we have k classes. Let p_j be the probability that an arbitrary 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[5]:

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. $\text{Im}(p_1, p_2) = \text{Im}(p_2, p_1)$

When we are training our decision tree, the labeled data can be used to compute the fraction of data samples that belong in each class. We can then use these fractions as probabilities. Thus, for example, to calculate the impurity of the data set in Figure 2, we let $p_{\text{Blue}} = \frac{5}{15}$ and $p_{\text{Red}} = \frac{10}{15}$. We then calculate $\text{Im}(\frac{5}{15}, \frac{10}{15})$.

---

[3]Dimitri P. Bertsekas and John N. Tsitsiklis, *Introduction to Probability* 2nd Edition, (Belmont, Massachusetts: Athena Scientific, 2008) 6-9

[4]Dimitri P. Bertsekas and John N. Tsitsiklis, *Introduction to Probability* 2nd Edition, (Belmont, Massachusetts: Athena Scientific, 2008) 82

[5]The Impurity Function, STAT 508, https://online.stat.psu.edu/stat508/lesson/11/11.2

4.4 Entropy

One widely used impurity metric is the entropy of a random variable X, $H(X)$. 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 as

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

Entropy reaches its maximum when the probability space is uniform. When $p(X) = 1$, that is there is only one outcome, $H(X) = 0$. Simply put, we have the lowest entropy when there is no uncertainty. If you have a background in probability, feel free to look over the following proof.

4.4.1 Optional proof

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 $p(X = i)$ is the same across all possible outcomes i, that is, when X is uniformly distributed.

4.5 Gini Index

Another common impurity function is the Gini Index. The Gini Index is a value ranging from $[0, 1]$ that represents the probability a feature will be incorrectly classified when randomly selected. A lower Gini Index is representative of a purer classification. The Gini Index for a parent node can be calculated with the following formula:

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

Since the likelihood of the outcomes in our probability space sum to 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 $p_i = 0$.

The following optional proof requires background knowledge in optimization.

4.5.1 Optional proof

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 take the partial derivative of the Lagrangian with respect to each element of the probability vector, p_j,

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

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

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.

4.6   Impurity Reduction and Information Gain

Once we have our impurity function, we need a way to measure the change in impurity before and 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 refer to impurity reduction as **information gain**. Formally this is the amount of information gained about a random variable from observing another random sample. In terms of decision trees, the information we gain after a split can be thought of as how much better we did compared to our starting point, the parent node. You can imagine that better features will provide us better information gain.

The general formula for a random variable X and information a is,

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

This simply calculates the difference between the current amount of entropy and the entropy given the additional information.

An important property of both of these functions is **nonnegativity**. With information gain, you begin with a certain amount of information $H(X)$, 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.

5   Algorithm for Learning a Decision Tree

Unfortunately, there is no efficient algorithm to find the optimal decision tree. However, we can define a greedy algorithm that considers all possible features given our training data at the current node, then 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 (class label) that is most frequent among the samples (i.e. the mode).

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

When does this algorithm stop? 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 **overfitting**. We know from polynomial regression that increasing the dimension in order to reduce error for the training data can overfit. Similarly, we may not want our decision tree algorithm 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