Lecture 7
Decision Trees

Alice Gao

November 2, 2021

Contents

1 Learning Goals

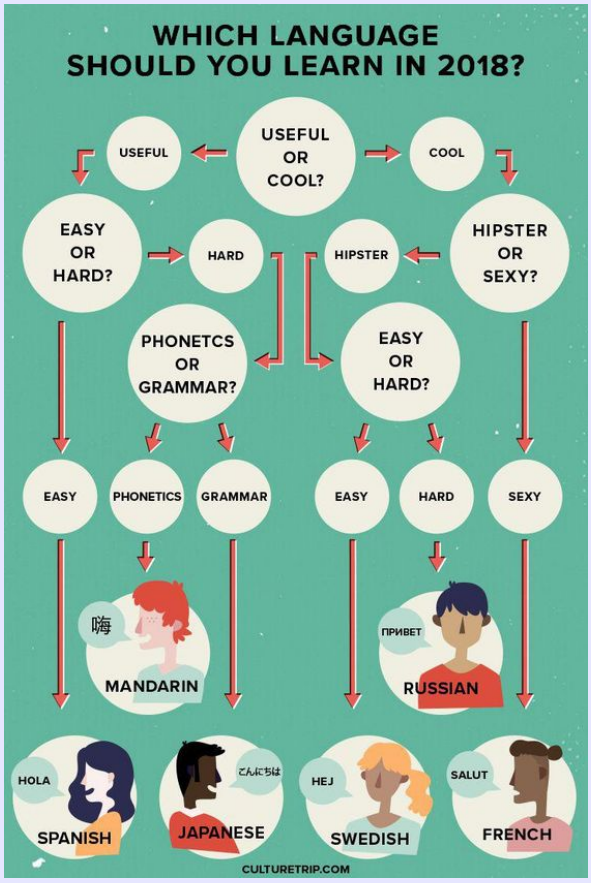
By the end of the lecture, you should be able to

- Describe the components of a decision tree.
- Construct a decision tree given an order of testing the features.
- Determine the prediction accuracy of a decision tree on a test set.
- Compute the entropy of a probability distribution.
- Compute the expected information gain for selecting a feature.
- Trace the execution of and implement the ID3 algorithm.

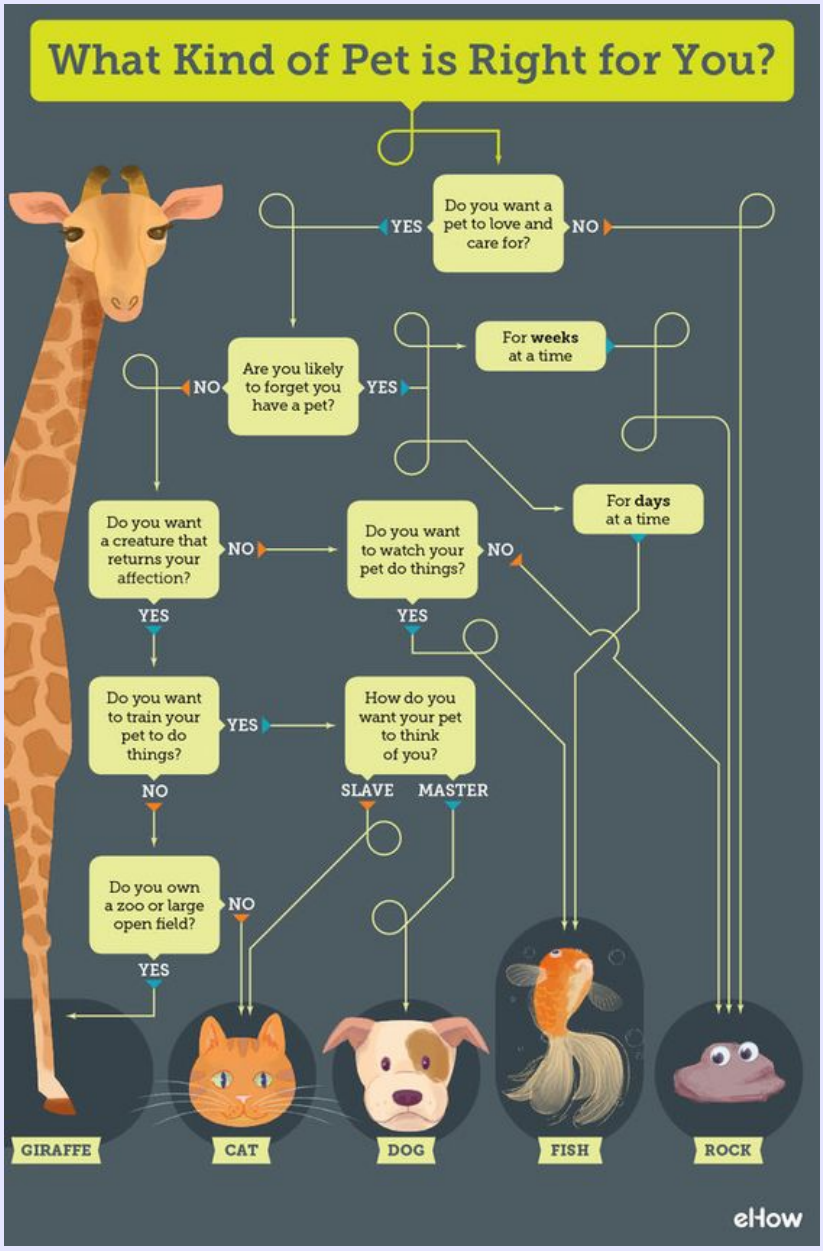
2 Examples of Decision Trees

Our first machine learning algorithm will be decision trees. A decision tree is a very common algorithm that we humans use to make many different decisions. You may be using one without realizing it. Here are some examples of decision trees.

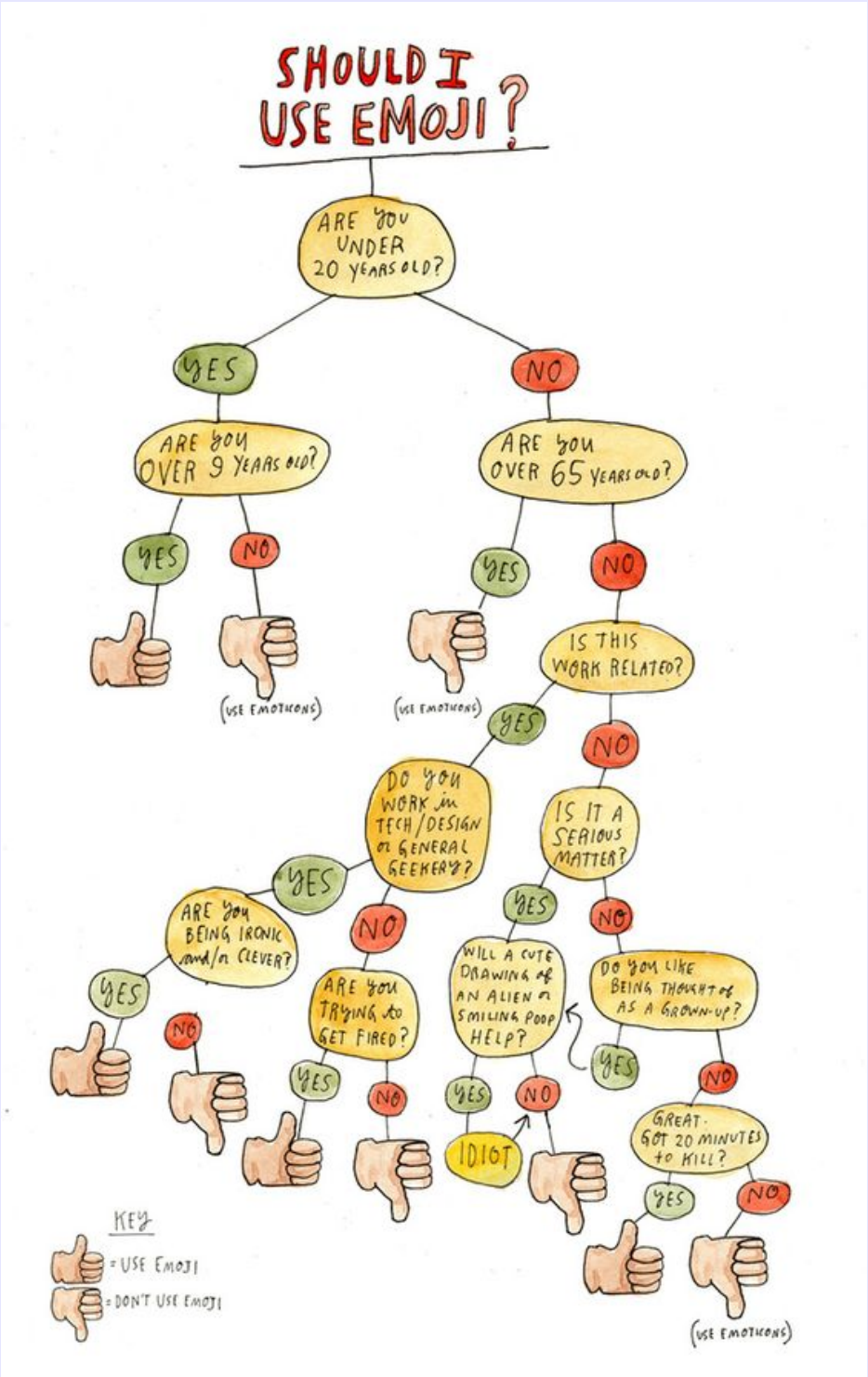
Example: Which language should you learn?



Example: What kind of pet is right for you?



Example: Should you use emoji in a conversation?



We will use the following example as a running example in this unit.

Example: Jeeves is a valet to Bertie Wooster. On some days, Bertie likes to play tennis and asks Jeeves to lay out his tennis things and book the court. Jeeves would like to predict whether Bertie will play tennis (and so be a better valet). Each morning over the last two weeks, Jeeves has recorded whether Bertie played tennis on that day and various attributes of the weather (training set).

Day	Outlook	Temp	Humidity	Wind	Tennis?
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

Jeeves would like to evaluate the classifier he has come up with for predicting whether Bertie will play tennis. Each morning over the next two weeks, Jeeves records the following data (test set).

Day	Outlook	Temp	Humidity	Wind	Tennis?
1	Sunny	Mild	High	Strong	No
2	Rain	Hot	Normal	Strong	No
3	Rain	Cool	High	Strong	No
4	Overcast	Hot	High	Strong	Yes
5	Overcast	Cool	Normal	Weak	Yes
6	Rain	Hot	High	Weak	Yes
7	Overcast	Mild	Normal	Weak	Yes
8	Overcast	Cool	High	Weak	Yes
9	Rain	Cool	High	Weak	Yes
10	Rain	Mild	Normal	Strong	No
11	Overcast	Mild	High	Weak	Yes
12	Sunny	Mild	Normal	Weak	Yes
13	Sunny	Cool	High	Strong	No
14	Sunny	Cool	High	Weak	No

3 Definition and Classifying an Example

3.1 What is a decision tree?

A decision tree is a simple model for supervised classification. It is used for classifying a single discrete target feature.

Each internal node performs a Boolean test on an input feature (in general, a test may have more than two options, but these can be converted to a series of Boolean tests). The edges are labeled with the values of that input feature.

Each leaf node specifies a value for the target feature.

3.2 Classifying an example using a decision tree

Classifying an example using a decision tree is very intuitive. We traverse down the tree, evaluating each test and following the corresponding edge. When a leaf is reached, we return the classification on that leaf.

Example: Here is an example of using the emoji decision tree. Assume:

- I am 30 years old.
- This is work related.
- I am an accountant.
- I am not trying to get fired.

Following the tree, we answer no (not under 20 years old), no (not over 65 years old), yes (work related), no (not working in tech), and no (not trying to get fired). The leaf we reach is a thumb down, meaning we should not use emoji.

Problem: If we convert a decision tree to a program, what does it look like?

Solution: A decision tree corresponds to a program with a big nested if-then-else structure.

4 The Decision Tree Learning Algorithm

4.1 Issues in learning a decision tree

How can we build a decision tree given a data set? First, we need to decide on an order of testing the input features. Next, given an order of testing the input features, we can build a decision tree by splitting the examples whenever we test an input feature.

Let's take a look at the Jeeves training set again. Each row is an example. There are 14 examples. For each example, we have five feature values: day, outlook, temperature, humidity, and wind. In fact, Day is not a useful feature since it's different for every example. So, we will focus on the other four input features. We have one target feature or label, which is whether Bertie decided to play tennis or not.

The decision tree is a powerful and flexible model. Given a data set, we can generate many different decision trees. Therefore, there are a few questions we need to think about when deciding which tree we should build.

First, different orders of testing the input features will lead to different decision trees. So, which order should we use? The number of possible orders is quite large, so it is challenging to find the optimal order in the search space. Given this, we will use a greedy or myopic approach. Instead of finding the optimal order of testing the features, we will make the myopic best choice at each step.

You might be wondering, what is the difference between making the optimal choice versus making the myopic best choice at each step? The main difference is that the optimal strategy considers the future, whereas the myopic strategy only cares about the current step. To generate an optimal tree, at each step, we need to think about how our feature choice at this step might affect our choices in the future. On the contrary, the myopic strategy does not think about the future at all and only cares about finding the best feature to test at the current step.

Now, suppose that we have chosen an order of testing the features using the myopic strategy. We have another choice. Should we grow a full tree or should we stop growing the tree earlier? Recall that I discussed over-fitting previously. The full tree may over-fit the training data, so a smaller tree might be better since it might generalize better to unseen test data.

To answer this question, we need a bias or an assumption that we make about which tree we prefer. One possible assumption could be based on Occam's razor principle, which says that the simplest model or hypothesis is probably the best. Based on Occam's razor, we would prefer a tree that is smaller than the full tree. This still leaves lots of options for us. For example, we can grow the tree until a certain depth, or we can grow the tree until it has a certain number of nodes.

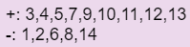
4.2 Grow a full tree given an order of testing features

Let's go through an example of constructing the full tree using the Jeeves training set. I've given you an order of testing the input features below.

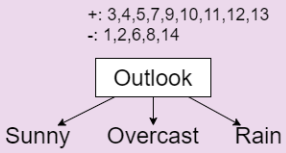
Problem: Construct a (full) decision tree for the Jeeves data set using the following order of testing features.

- First, test Outlook.
- For Outlook = Sunny, test Temp.
- For Outlook = Rain, test Wind.
- For other branches, test Humidity before testing Wind.

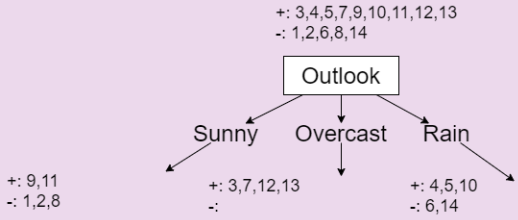
Solution: Here is the process to generate the decision tree by the given order.



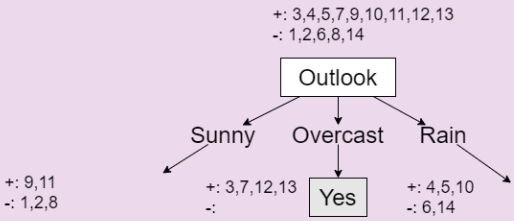
We have 9 positive and 5 negative examples. They are not in the same class, so we will have to choose a feature to test.



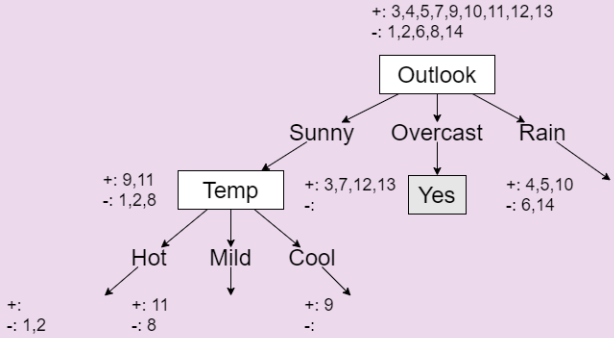
Based on the given order, we will test Outlook first. Outlook has three values: Sunny, Overcast, and Rain. We split the examples into three branches.



The three sets look like this. Example 1 has Outlook equal to Sunny, so it goes into the left branch. Example 3 has Outlook equal to Overcast, so it goes into the middle branch, etc.

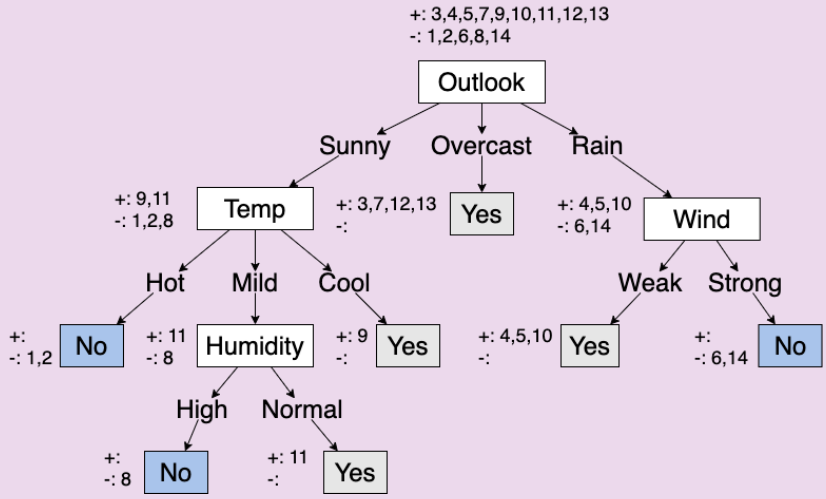


In the middle branch, all the examples are positive. There is no need to test another feature, and so we may make a decision. We create a leaf node with the label Yes, and we are done with this branch.



Looking at the left branch next, there are two positive and three negative examples. We have to test another feature. Based on the given order, we will test Temp next. Temp has three values — Hot, Mild, and Cool. We create three branches again. The five examples are split between these branches.

We will repeat this process at every node. First, check if all the examples are in the same class. If they are, create a leaf node with the class label and stop. Otherwise, choose the next feature to test and split the examples based on the chosen feature.



The above is the final decision tree. Each internal node represents a test—Not all of these are Boolean. Beside each node, the training examples are partitioned into two classes based on whether Bertie played tennis that day: Yes (+) and No (−).

4.3 When do we stop?

There are three possible stopping criteria for the decision tree algorithm. For the example in the previous section, we encountered the first case only: when all of the examples belong to the same class. In this case, we make the decision of that class and then we're done.

4.4 Base case 2: no features left

Let's look at the second case: what should we do if there are no more features to test?

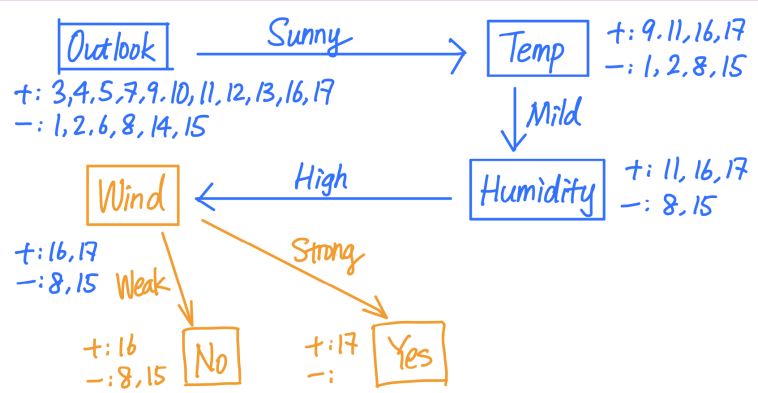
Problem: I took our training set and added a few examples. It now has 17 instead of 14 examples. For this modified training set, let's construct one branch of the decision tree where Outlook is Sunny, Temperature is Mild, and Humidity is High.

Day	Outlook	Temp	Humidity	Wind	Tennis?
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No
15	Sunny	Mild	High	Weak	No
16	Sunny	Mild	High	Weak	Yes
17	Sunny	Mild	High	Strong	Yes

Solution: After testing Humidity is High, what should we do next? We have tested three of the four input features. To continue, testing Wind is the only option. When Wind is Strong, we have one positive example, and the decision is Yes. When Wind is Weak, we have three examples: one positive and two negative examples. The examples are mixed, so we cannot make a decision. But, we have tested all four features — there's no more feature to test. What should we do in this case?

Let's take another look at our data set. There are three examples when Wind is Weak. Note that, for the three examples, the values of all input features are all the same — Sunny, Mild, High, and Weak, but they have different labels, No for 8 and 15 and Yes for 16. This is an example of a noisy data set. With noisy data, even if we know the values of all the input features, we are still unable to make a deterministic decision. One reason for having a noisy data set is that the decision may be influenced by some features that we do not observe. Perhaps, another factor not related to weather influences Bertie's decision, but we don't know what that factor is.

What should we do when we run out of features to test? There are a few options. One option is to predict the majority class. In this case, the majority decision is No. Another option is to make a randomized decision. We can think of the three examples as a probability distribution. There is a 1/3 probability of predicting Yes and a 2/3 probability of predicting No. We will make the decision based on a random draw from the distribution. For this example, let's use the majority decision.



4.5 Base case 3: no examples left

Let's look at the last possible stopping criteria: what should we do if there are no examples left?

Problem: Let's consider another modified Jeeves training set. Complete one branch of the decision tree where Temperature is Hot, Wind is Weak, and Humidity is High.

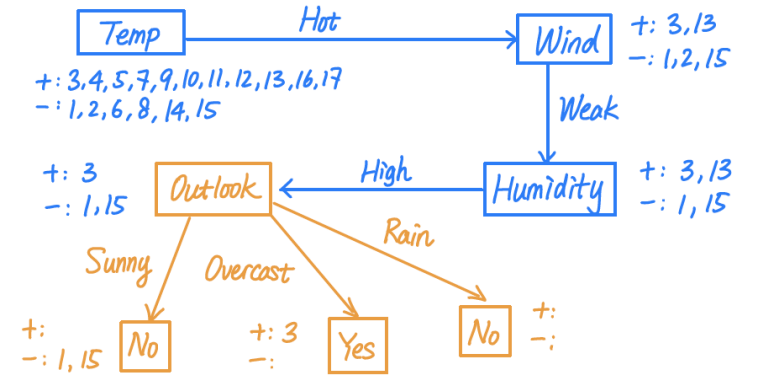
Day	Outlook	Temp	Humidity	Wind	Tennis?
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No
15	Sunny	Hot	High	Weak	No

Solution: After testing the three features, we have three examples left: one positive and two negative. The only feature left is Outlook. Outlook has three values. Let's split up the examples into three branches.

If Outlook is Sunny, we have two negative examples, and the decision is No. If Outlook is Overcast, we have one positive example, and the decision is Yes. Finally, if Outlook is Rain, there are no examples left. What should we do here?

Before we decide on what to do, let's ask a related question. Why did we encounter this case? Let's take another look at the modified data set. The case we are looking for is: Temperature is Hot, Wind is Weak, Humidity is High, and Outlook is Rain. After going through the data set, you will realize that no example in this data set has this combination of input feature values. This is the reason that we had no examples left. If we never observe a combination of feature values, we don't know how to predict it.

Now that we understand why this happened, how should we handle this case? One idea is to try to find some examples that are close to this case. If we go up to the parent node, the parent has some examples for different values of Outlook. Arguably, these are the closest examples to our case. Therefore, we can use the examples at the parent node to make a decision. At the parent node, the examples are likely to be mixed. Most likely, we cannot make a deterministic decision. Similar to the previous case, we can either make the majority decision or decide based on a random draw from a probability distribution.



4.6 Pseudo-code for the decision tree learner algorithm

Algorithm 1 Decision Tree Learner (examples, features)

```
1: if all examples are in the same class then
2:     return the class label.
3: else if no features left then
4:     return the majority decision.
5: else if no examples left then
6:     return the majority decision at the parent node.
7: else
8:     choose a feature f.
9:     for each value v of feature f do
10:        build edge with label v.
11:        build sub-tree using examples where the value of f is v.
```

Here is the pseudocode for the algorithm. Since a tree is recursive, we will naturally use a recursive algorithm to build it.

The algorithm starts with three base cases.

1. If all the examples are in the same class, we will return the class label.
2. If there are no features left, we have noisy data. We can either return the majority decision or make a decision probabilistically.
3. If there are no examples left, then some combination of input features is absent in the training set. We can use the examples at the parent node to make a decision. If the examples are not in the same class, we can either return the majority decision or make a decision probabilistically.

Next, we have the recursive part. Suppose that we have a pre-specified order of testing the input features. At each step, we will split up the examples based on the chosen feature's values. We will label the edges with the feature's value. Each subtree only has examples where the value of the feature corresponds to the value on the edge.

There's one crucial step left. So far, we have assumed that a pre-defined order of testing the input features. Where does this order come from? In practice, we have to choose this order ourselves.

5 Determine the Order of Testing Features

5.1 Which feature should we test at each step?

In a previous section, I discussed strategies for choosing a decision tree. Given a data set, we can build many different decision trees. Which tree should we build? One critical issue in machine learning is over-fitting. Over-fitting often happens for a complex model, which captures both the useful patterns in the data and the noise. One way to avoid over-fitting is to constrain our model to be a simple one. In our case, we will try to minimize the number of features we have to test by learning a small and shallow tree.

Ideally, we would like to find the optimal order of testing features, which will minimize the size of our tree. Unfortunately, finding the optimal order is too expensive computationally. Instead, we will use a greedy and myopic approach.

The myopic approach will make the best choice at each step without worrying about how our current choice could affect the potential choices in the future. More concretely, at each step, we will choose a feature that makes the biggest difference to the classification, or a feature that helps us make a decision as quickly as possible.

5.2 Identifying the most important feature

In the previous section, we decided to choose a feature that helps us make a decision as soon as possible, that is, a feature that reduces our uncertainty at much as possible.

To measure the reduction in uncertainty, we will calculate the uncertainty in the examples before testing the feature, and subtract the uncertainty in the examples after testing the feature. The difference measures the information content of the feature. Intuitively, testing the feature allows us to reduce our uncertainty and gain some useful information. We will select the feature that has the highest information content.

Finally, the only remaining question is: How do we measure the uncertainty in a set of examples? We will use a concept called entropy from information theory.

Let's look at the definition of entropy. Consider a probability distribution over k outcomes: c_1 to c_k. The probability of each outcome c_i is $P(c_i)$.

Given a distribution, we will calculate its entropy as follows: The entropy I is the summation of several terms. For each outcome, multiply the probability of the outcome with the log base 2 of the probability of the outcome. We are using base 2 here because the information is usually measured in bits. Finally, we'll add all the terms together and negate the result, giving us a non-negative value. s

$$I(P(c_1), \ldots, P(c_2)) = -\sum_{i=1}^{k} P(c_i) \log_2(P(c_i)). \tag{1}$$

5.3 Entropy of a distribution over two outcomes

Let's practice using the formula to calculate entropy For the two questions, we will calculate the entropy of two binary distributions. While you work on these questions, think about how the entropy changes as the distribution changes.

Problem: What is the entropy of the distribution $(0.5, 0.5)$?

(A) 0.2

(B) 0.4

(C) 0.6

(D) 0.8

(E) 1

Solution: The correct answer is (E).

Performing the calculation, we have

$$-(0.5 \log_2(0.5)) \times 2 = 1$$

The entropy is 1 bit.

You might be wondering, is one bit of entropy a lot or very little? You will get a better idea of this after question 2. For now, believe me when I say that there is a lot of uncertainty in this binary distribution.

This should make intuitive sense. This distribution is uniform. A uniform distribution has a lot of uncertainty since every outcome is equally likely to become true.

Problem: What is the entropy of the distribution $(0.01, 0.99)$?

We have a very skewed distribution. Almost all the probability is on the second outcome.

(A) 0.02

(B) 0.04

(C) 0.06

(D) 0.08

(E) 0.1

Solution: The correct answer is (D).

Performing the calculation, we have

$$\begin{aligned} & -\left(0.01 \log_2(0.01) + 0.99 \log_2(0.99)\right) \\ &= -(-0.0664 - 0.01435) \\ &\approx 0.08. \end{aligned}$$

This distribution has 0.08 bit of uncertainty.

Compared to the previous distribution, this distribution has very little uncertainty. Again, this result makes intuitive sense. Looking at the probabilities, we almost know for sure that the second outcome will become true. This means that we don't really have that much uncertainty. If you are making a bet, you will probably bet on the second outcome and you will likely win the bet.

What have we learned about entropy from these two questions? Let me summarize some important points.

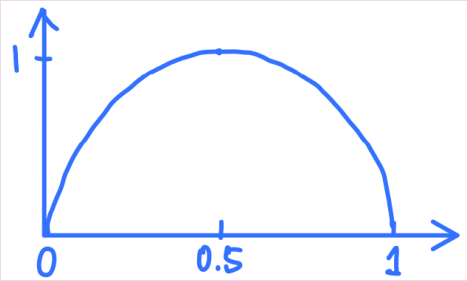
Problem: Consider a distribution $(p, 1-p)$ where $0 \leq p \leq 1$.

1. What is the maximum entropy of this distribution?
2. What is the minimum entropy of this distribution?
3. Plot the entropy of the distribution $(p, 1-p)$ with respect to p.

Solution: For any binary distribution, its entropy achieves its maximum value of one when the distribution is uniform, and achieves its minimum value of zero when the distribution is a point mass — one outcome has a probability of 1.

When p is 0 or 1, calculating the entropy is a bit tricky. since $log_2(0)$ is undefined. In this case, let's the term $0 * log_2(0)$ to be 0. This definition is reasonable since limit of $x * log_2(x)$ is 0 when x approaches 0.

Finally, here is the plot of the distribution's entropy with respect to p. As p increases from 0 to 1, the entropy increases first, reaches the maximum value when p is 0.5, and then decreases after that.

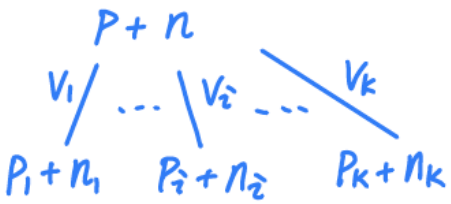


Here is a practice question for you. I've summarized some properties of entropy for a binary distribution. What about a distribution with more than two outcomes. For example, what are the maximum and minimum entropy for a distribution with three outcomes?

5.4 Expected information gain of testing a feature

I have discussed how to measure the uncertainty of a distribution using entropy. The next question is, how can we quantify the information content of a feature?

Consider a feature with k values, v_1 to v_k. There are p positive examples and n negative examples before testing this feature. After testing this feature, we divide the examples into k sets. For each feature value v_i, let's suppose that the set has p_i positive examples and n_i negative examples. This picture shows you the setting.



Recall that the information content of a feature is the entropy of the examples before testing the feature minus the entropy of the examples after testing the future. Formally, we call this the expected information gain of testing this feature.

Let me write down the formula for calculating the expected information gain.

Before testing the feature, we have p positive and n negative examples. Let's convert this to a binary distribution: Yes has a probability of p / (p + n), and No has a probability of n / (p + n). We can calculate the entropy of this distribution using the formula we saw.

$$I_{\text{before}} = I\left(\frac{p}{p+n}, \frac{n}{p+n}\right). \tag{2}$$

After testing the feature, we have k distributions, one for each of the k feature values. We have a problem here. Without an example, we don't know which distribution will be useful. How do we calculate the entropy of k distributions? The solution is to calculate the expected

entropy of these distributions. We will first calculate the entropy of each distribution. Then, we will take the expected value of the k entropies. In the expected value, the weight of each distribution is the fraction of examples in that distribution.

For example, for the feature value v_i, there are p_i positive examples and n_i negative examples. The entropy is given by $I(p_i/(p_i+n_i), n_i/(p_i+n_i))$. The fraction of examples in the this distribution is $(p_i+n_i)/(p+n)$ — this is the weight of the i-th entropy in the expected value.

The expected entropy after testing the feature is

$$EI_{\text{after}} = \sum_{i=1}^{k} \frac{p_i + n_i}{p+n} * I\left(\frac{p_i}{p_i+n_i}, \frac{n_i}{p_i+n_i}\right). \tag{3}$$

Finally, the expected information gain or entropy reduction is the entropy before testing the feature minus the expected entropy after testing the feature.

$$\text{InfoGain} = I_{\text{before}} - I_{\text{after}}. \tag{4}$$

5.5 A full example

Here, we will work through generating a complete decision tree based on the rules introduced in this section. We can start with the following questions:

Problem: What is the entropy of the examples before we select a feature for the root node of the tree?

(A) 0.54

(B) 0.64

(C) 0.74

(D) 0.84

(E) 0.94

Solution: There are 14 examples: 9 positive, 5 negative. Applying the formula gives

$$\begin{aligned} H_{\text{before}} &= -\left(\frac{9}{14}\log_2\left(\frac{9}{14}\right) + \frac{5}{14}\log_2\left(\frac{5}{14}\right)\right) \\ &= -\left(\frac{9}{14}(-0.637) + \frac{5}{14}(-1.485)\right) \\ &= -(-0.939) \\ &\approx 0.94. \end{aligned}$$

The correct answer is (E).

(In an exam, you could calculate $\log_2 x = \frac{\ln x}{\ln 2}$).

Problem: What is the expected information gain if we select Outlook as the root node of the tree?

(A) 0.237

(B) 0.247

(C) 0.257

(D) 0.267

(E) 0.277

Solution: Testing Outlook yields three branches:

$$\text{Outlook} = \begin{cases} \text{Sunny} & 2+ & 3- & 5 \text{ total} \\ \text{Overcast} & 4+ & 0- & 4 \text{ total} \\ \text{Rain} & 3+ & 2- & 5 \text{ total} \end{cases}$$

The expected information gain is

$$\begin{aligned} \text{Gain(Outlook)} &= 0.94 - \left(\frac{5}{14} \cdot I\left(\frac{2}{5}, \frac{3}{5}\right) + \frac{4}{14} \cdot I\left(\frac{4}{4}, \frac{0}{4}\right) + \frac{5}{14} \cdot I\left(\frac{3}{5}, \frac{2}{5}\right) \right) \\ &= 0.94 - \left(\frac{5}{14}(0.971) + \frac{4}{14}(0) + \frac{5}{14}(0.971) \right) \\ &= 0.94 - 0.694 \\ &= 0.247. \end{aligned}$$

The correct answer is (B).

Problem: What is the expected information gain if we select Humidity as the root node of the tree?

(A) 0.151

(B) 0.251

(C) 0.351

(D) 0.451

(E) 0.551

Solution: Testing Humidity yields two branches:

$$\text{Humidity} = \begin{cases} \text{Normal} & 6+ & 1- & 7 \text{ total} \\ \text{High} & 3+ & 4- & 7 \text{ total} \end{cases}$$

The expected information gain is

$$\begin{aligned} \text{Gain}(\text{Humidity}) &= 0.94 - \left(\frac{7}{14} \cdot I\left(\frac{6}{7}, \frac{1}{7}\right) + \frac{7}{14} \cdot I\left(\frac{3}{7}, \frac{4}{7}\right) \right) \\ &= 0.94 - \left(\frac{7}{14}(0.592) + \frac{7}{14}(0.985) \right) \\ &= 0.94 - 0.789 \\ &= 0.151. \end{aligned}$$

The correct answer is (A).

Comparing Outlook and Humidity, the expected information gain of testing Outlook first is greater than that of testing Humidity first.

You may also notice that the expected information gain at a node is always taken with respect to the entropy before the node, so when comparing features to test at that node we could simply ignore the entropy before testing and select the feature with the lowest post-testing entropy.

Example: Here is the continuation of the example. The calculations have been condensed, but you should verify them yourself.

For the root node, we can choose from all four features to test:

$$\text{Gain}(\textbf{Outlook}) = \textbf{0.247} \quad \text{Gain}(\text{Humidity}) = 0.151$$
$$\text{Gain}(\text{Temp}) = 0.029 \qquad \text{Gain}(\text{Wind}) = 0.048$$

We pick Outlook since it has the greatest expected information gain.

Case 1: Outlook = Sunny. $+: 9, 11 \quad -: 1, 2, 8$

$$\text{Temp} = \begin{cases} \text{Hot} & +: & -: 1, 2 \\ \text{Mild} & +: 11 & -: 8 \\ \text{Cool} & +: 9 & -: \end{cases}$$

$$\begin{aligned} \text{Gain(Temp)} &= I\left(\frac{2}{5}, \frac{3}{5}\right) - \left(\frac{2}{5} \cdot I\left(\frac{0}{2}, \frac{2}{2}\right) + \frac{2}{5} \cdot I\left(\frac{1}{2}, \frac{1}{2}\right) + \frac{1}{5} \cdot I\left(\frac{1}{1}, \frac{0}{1}\right)\right) \\ &= 0.97 - \left(\frac{2}{5}(0) + \frac{2}{5}(1) + \frac{1}{5}(0)\right) \\ &= 0.97 - 0.4 \\ &= 0.57 \end{aligned}$$

$$\textbf{Humidity} = \begin{cases} \text{High} & +: & -: 1, 2, 8 \\ \text{Normal} & +: 9, 11 & -: \end{cases}$$

$$\begin{aligned} \text{Gain(Humidity)} &= I\left(\frac{2}{5}, \frac{3}{5}\right) - \left(\frac{3}{5} \cdot I\left(\frac{0}{3}, \frac{3}{3}\right) + \frac{2}{5} \cdot I\left(\frac{2}{2}, \frac{0}{2}\right)\right) \\ &= 0.97 - \left(\frac{3}{5}(0) + \frac{2}{5}(0)\right) \\ &= 0.97 - 0 \\ &= \mathbf{0.97} \end{aligned}$$

$$\text{Wind} = \begin{cases} \text{Weak} & +: 9 & -: 11 \\ \text{Strong} & +: 1, 8 & -: 2 \end{cases}$$

$$\begin{aligned} \text{Gain(Wind)} &= I\left(\frac{2}{5}, \frac{3}{5}\right) - \left(\frac{3}{5} \cdot I\left(\frac{1}{3}, \frac{2}{3}\right) + \frac{2}{5} \cdot I\left(\frac{1}{2}, \frac{1}{2}\right)\right) \\ &= 0.97 - \left(\frac{3}{5}(0.918) + \frac{2}{5}(1)\right) \\ &= 0.97 - 0.951 \\ &= 0.019 \end{aligned}$$

We pick Humidity since it has the greatest expected information gain. This makes sense since the positive and negative examples are completely separated after testing Humidity.

Case 2: Outlook = Overcast. $+: 3, 7, 12, 13 \quad -:$

No need to test further, since the examples are already classified.

Case 3: Outlook = Rain. $+: 4, 5, 10 \quad -: 6, 14$

$$\text{Temp} = \begin{cases} \text{Hot} & +: & -: \\ \text{Mild} & +: 4, 10 & -: 14 \\ \text{Cool} & +: 5 & -: 6 \end{cases}$$

$$\begin{aligned} \text{Gain(Temp)} &= I\left(\frac{2}{5}, \frac{3}{5}\right) - \left(\frac{3}{5} \cdot I\left(\frac{2}{3}, \frac{1}{3}\right) + \frac{2}{5} \cdot I\left(\frac{1}{2}, \frac{1}{2}\right)\right) \\ &= 0.97 - \left(\frac{3}{5}(0.918) + \frac{2}{5}(1)\right) \\ &= 0.97 - 0.951 \\ &= 0.019 \end{aligned}$$

$$\text{Humidity} = \begin{cases} \text{High} & +: 4 & -: 14 \\ \text{Normal} & +: 5, 10 & -: 6 \end{cases}$$

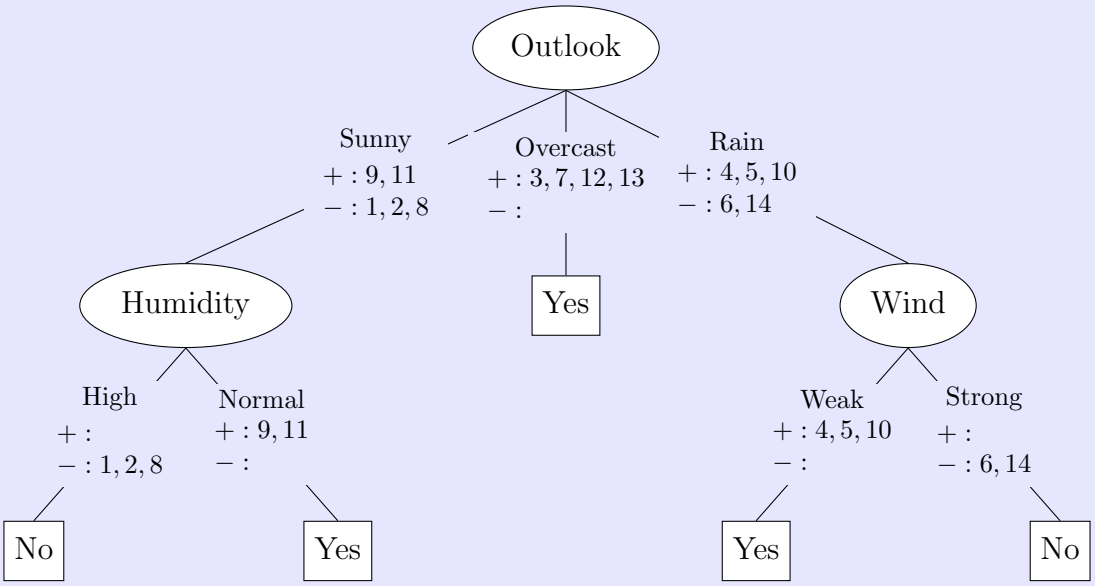
$$\begin{aligned} \text{Gain(Humidity)} &= I\left(\frac{2}{5}, \frac{3}{5}\right) - \left(\frac{2}{5} \cdot I\left(\frac{1}{2}, \frac{1}{2}\right) + \frac{3}{5} \cdot I\left(\frac{2}{3}, \frac{1}{3}\right)\right) \\ &= 0.97 - \left(\frac{2}{5}(1) + \frac{3}{5}(0.918)\right) \\ &= 0.97 - 0.951 \\ &= 0.019 \end{aligned}$$

$$\textbf{Wind} = \begin{cases} \text{Weak} & +: 4, 5, 10 & -: \\ \text{Strong} & +: & -: 6, 14 \end{cases}$$

$$\begin{aligned} \text{Gain(Wind)} &= I\left(\frac{2}{5}, \frac{3}{5}\right) - \left(\frac{3}{5} \cdot I\left(\frac{3}{3}, \frac{0}{3}\right) + \frac{2}{5} \cdot I\left(\frac{0}{2}, \frac{2}{2}\right)\right) \\ &= 0.97 - \left(\frac{3}{5}(0) + \frac{2}{5}(0)\right) \\ &= 0.97 - 0 \\ &= \mathbf{0.97} \end{aligned}$$

We pick Wind since it has the greatest expected information gain. This makes sense since the positive and negative examples are completely separated after testing Wind.

The final decision tree is drawn below:



Recall that we wanted a shallow, small tree, and that's exactly what we got.

6 Real-Valued Features

6.1 Jeeves dataset with real-valued temperatures

So far, the decision tree learner algorithm only works when all of the features have discrete values. In the real world, we are going to encounter a lot of data sets where many features have continuous values, or they're real-valued. Let's see how decision trees can handle them.

Example: A more realistic Jeeves dataset with real-valued temperatures

Day	Outlook	Temp	Humidity	Wind	Tennis?
1	Sunny	29.4	High	Weak	No
2	Sunny	26.6	High	Strong	No
3	Overcast	28.3	High	Weak	Yes
4	Rain	21.1	High	Weak	Yes
5	Rain	20.0	Normal	Weak	Yes
6	Rain	18.3	Normal	Strong	No
7	Overcast	17.7	Normal	Strong	Yes
8	Sunny	22.2	High	Weak	No
9	Sunny	20.6	Normal	Weak	Yes
10	Rain	23.9	Normal	Weak	Yes
11	Sunny	23.9	Normal	Strong	Yes
12	Overcast	22.2	High	Strong	Yes
13	Overcast	27.2	Normal	Weak	Yes
14	Rain	21.7	High	Strong	No

Instead of having the temperature being a discrete feature, having three values: *Mild*, *Hot*, and *Cool*, we will use actual numbers for *Temperature*.

For convenience, we will reorder the data set based on the value of the *Temperature*.

Example: Jeeves dataset ordered by temperatures

Day	Outlook	Temp	Humidity	Wind	Tennis?
7	Overcast	17.7	Normal	Strong	Yes
6	Rain	18.3	Normal	Strong	No
5	Rain	20.0	Normal	Weak	Yes
9	Sunny	20.6	Normal	Weak	Yes
4	Rain	21.1	High	Weak	Yes
14	Rain	21.7	High	Strong	No
8	Sunny	22.2	High	Weak	No
12	Overcast	22.2	High	Strong	Yes
10	Rain	23.9	Normal	Weak	Yes
11	Sunny	23.9	Normal	Strong	Yes
2	Sunny	26.6	High	Strong	No
13	Overcast	27.2	Normal	Weak	Yes
3	Overcast	28.3	High	Weak	Yes
1	Sunny	29.4	High	Weak	No

6.2 Handling a discrete feature

Before seeing how we can handle real-valued features, let's review how we can handle a discrete feature. We considered two options:

- Allow multi-way splits.

  The tree becomes more complex than just if-then-else.

  The tree tends to be shallower.

  The expected information gain metric prefers a variable with a larger domain, because when a variable has a larger domain we can split the data points into more sets, and there's a higher chance to reduce entropy.

- Restrict to binary splits.

  The tree is simpler and more compact.

  The tree tends to be deeper.

Example: As an extreme example, pretend that Day is an additional input feature in the Jeeves dataset, then the expected information gain for splitting on day must be the largest. Since we have one example for each day, and we can make a deterministic decision right away after splitting on day. However, it clearly does not make sense to use day as a feature, as the resulting tree does not generalize to any other data sets.

6.3 Handling a real-valued feature

For handling a real-valued feature:

- Discretize the feature (*i.e.*, put the examples into buckets).

  This is easy to do, but we may lose valuable information as it is challenging to figure out good ranges beforehand.

  There is often no way to figure out the optimal discretization while we are constructing the tree, which may lead to a more complex decision tree.

- Allow multi-way splits.

  Multi-way splits are impractical because the domain of the feature could be unbounded.

- Restrict to binary splits.

  Dynamically choose the split points.

  However, binary splits may test the same feature multiple times and make the tree much deeper.

We will limit ourselves to binary splits.

6.4 Choosing a split point for a real-valued feature

A naïve way to choose a split point is to first sort the instances by the real-valued feature and consider each midpoint of two different consecutive values as a possible split point. Then, we would calculate the expected information gain of each possible split points and pick one with the greatest expected information gain.

However, there may be too many possible points to consider if we use this method.

Example: With the modified Jeeves data set from before, there are 11 possible split points using the naïve method.

A smarter approach is as follows:

1. Sort the instances according to the real-valued feature.
2. Possible split points are values that are midway between two adjacent values that are different.
3. Suppose that the feature changes from X to Y. Should we consider $(X + Y)/2$ as a possible split point?
4. Let L_X be all the labels for the examples where the feature takes the value X.
5. Let L_Y be all the labels for the examples where the feature takes the value Y.
6. If there exists a label $a \in L_X$ and a label $b \in L_Y$ such that $a \neq b$, then $(X + Y)/2$ is a possible split point.

7. Determine the expected information gain for each possible split point and choose the split point with the largest gain.

Example: Using the modified Jeeves data set from before, consider the midway point of $X = 23.9$ and $Y = 26.6$. Then $L_X = \{\text{Yes}\}$ and $L_Y = \{\text{No}\}$. Taking Yes $= a \in L_X$ and No $= b \in L_Y$, we have $a \neq b$, so $(X + Y)/2 = 25.25$ is a possible split point.

Problem: For the Jeeves training set, is the midpoint between 20.0 and 20.6 a possible split point?

(A) Yes (B) No

Solution: This is not a possible split point: $L_{20.0} = \{\text{Yes}\}$ and $L_{20.6} = \{\text{Yes}\}$.

The correct answer is (B).

Problem: For the Jeeves training set, is the midpoint between 21.1 and 21.7 a possible split point?

(A) Yes (B) No

Solution: This is a possible split point: $L_{21.1} = \{\text{Yes}\}$ and $L_{21.7} = \{\text{No}\}$.

The correct answer is (A).

Problem: For the Jeeves training set, is the midpoint between 21.7 and 22.2 a possible split point?

(A) Yes (B) No

Solution: This is a possible split point: $L_{21.7} = \{\text{No}\}$ and $L_{22.2} = \{\text{No}, \text{Yes}\}$.

The correct answer is (A).

Intuitively, you can understand this procedure as considering local changes at each midway point. If the target feature doesn't change locally, that point probably isn't a valuable split point.

7 Over-fitting

7.1 Corrupted data in the Jeeves dataset

Over-fitting is a common problem when we're learning a decision tree. As a model for supervised machine learning, a decision tree has several nice properties. Decision trees are simpler, they're easy to understand and easy to interpret.

When it's important to explain a model to another human being, a decision tree is a good choice. In contrast, a neural network is often complex and difficult or impossible to interpret.

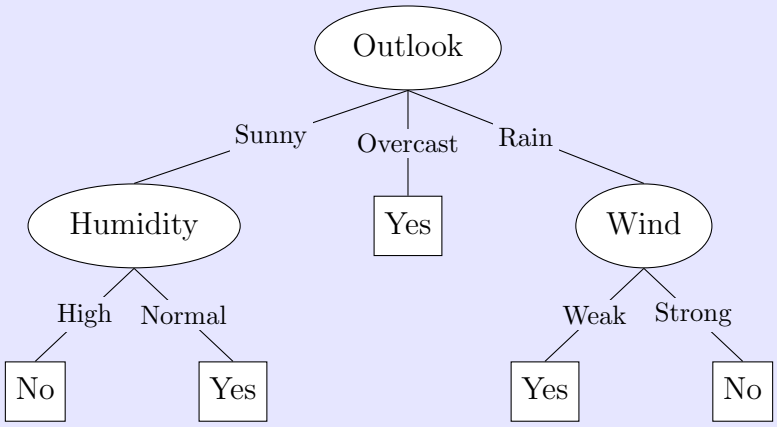
Also decision trees can produce a reasonable model even if you have a tiny data set. In contrast, a neural network often does not work at all with a small data set, and it is extremely prone to over-fitting.

Therefore, even though decision trees have so many nice properties, over-fitting is still a common problem when we're constructing a decision tree.

Example: The Jeeves data set has 14 data points, this is an extremely small data set; we can still learn a reasonable decision tree for this data set.

Day	Outlook	Temp	Humidity	Wind	Tennis?
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

Decision tree generated by the learner algorithm:



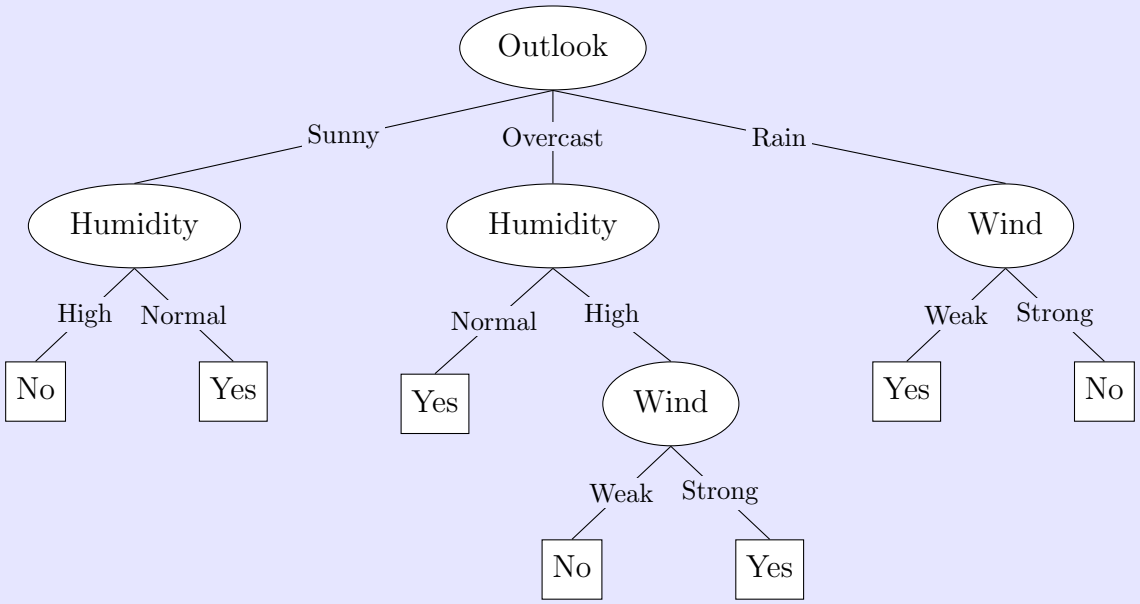
Test error is 0 out of 14.

Example: Suppose that you sent the training set and the test set to your friend. For some reason the data set gets corrupted during the transmission. Your friend gets a corrupted training set where only one data point is different. For this third data, point, it changed from the label "yes" to the label "no".

This is a tiny change to the training set, but how would this change affect the decision tree that we generate?

Day	Outlook	Temp	Humidity	Wind	Tennis?
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	**No**
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

Decision tree generated by the learner algorithm:



We grew an entire middle sub-tree to accommodate one corrupted data point, and this small corruption caused a dramatic change to the tree. This new tree is more complicated and it likely won't generalize well to unseen data.

The decision tree learner algorithm is a perfectionist. The algorithm will keep growing the tree until it perfectly classifies all the examples in the training set. However, this is not necessarily a desirable behavior and this can easily lead to over-fitting.

The new test error is 2/14.

7.2 Dealing with over-fitting with pruning

It would be better to grow a smaller and shallower tree. The smaller and shallower tree may not predict all of the training data points perfectly but it may generalize to test data better.

At a high level we have two options to prevent over-fitting when learning a decision tree:

Pre-pruning: stop growing the tree early.

If we decide not to split the examples at a node and stop growing the tree there, we may still have examples with different labels. At this point, we can decide to use the majority label as the decision for that leaf node. Here are some criteria we can use:

1. Maximum depth: We can decide not to split the examples if the depth of that node has reached a maximum value that we decided beforehand.
2. Minimum number of examples at the leaf node: We can decide not to split the examples if the number of examples remaining at that node is less than a predefined threshold value.

3. Minimum information gain: We can decide not to split the examples if the benefit of splitting at that node is not large enough. We can measure the benefit by calculating the expected information gain. In other words, do not split examples if the expected information gain is less than the threshold.
4. Reduction in training error: We can decide not to split the examples at a node if the reduction in training error is less than a predefined threshold value.

For pre-pruning, we will split the examples at a node only when it's useful. We can use pre-pruning with any of the criteria above.

Post-pruning: grow a full tree first and then trim it afterwards.

Post-pruning is particularly useful when any individual feature is not informative, but multiple features working together is very informative.

Example: Consider a data set with two input features, both features are binary. The target feature is computing the XOR function of the two input features. That means the target feature is true if and only if the two input features have different values.

For this data set, each input feature alone tells us nothing about the value of the target feature. However, if you know both input features then you know the target feature for certain.

For this example,

- With pre-pruning,

  We will test none of the two input features, testing each feature gives us zero information. So we'll end up with a tree that has only the root node and the tree will predict the target feature randomly.

- With post-pruning,

  We will end up growing the full tree first which means testing both input features in some order. Afterwards, we will try to see if we can prune any of the node.

  It turns out it's not a good idea to prune any node, because the second input feature in the order is going to give us all the information that we need.

  So we will end up growing a full tree and the full tree will predict the target feature perfectly.

Post-pruning helps us recognize the cases with multiple features working together will be very beneficial, but any of the features individually will not be very useful. We can apply post-pruning to any of the criteria we have described above. Note that we will only decide to post-prune a node if it has only leaf nodes as its descendants.

Example: Suppose we are considering post-pruning with the minimal information gain metric.

First of all, we will restrict our attention to nodes that only have leaf nodes as its descendants. At a node like this, if the expected information gain is less than a predefined threshold value, we will delete this node's children which are all leaf nodes and then convert this node to a leaf node.

There has to be examples with different labels at this node possibly both positive and negative examples. We can make a majority decision at this node.

8 Practice Problems

1. When learning the tree, we chose a feature to test at each step by maximizing the expected information gain. Does this approach allow us to generate the optimal decision tree? Why or why not?
2. Consider a data-set with real-valued features. Suppose that we perform binary splits only when building a decision tree.

   Is it possible to encounter the "no features left" base case? Why?

   Is it possible to encounter the "no examples left" base case? Why?
3. What is the main advantage of post-pruning over pre-pruning?