Classification: Decision Trees

Prof. (Dr.) Honey Sharma

Refrence: https://towardsdatascience.com/

Classification

A big part of machine learning is classification — we want to know what class (a.k.a. group) an observation belongs to. The ability to precisely classify observations is extremely valuable for various business applications like predicting whether a particular user will buy a product or forecasting whether a given loan will default or not.

The Basics of Decision Trees

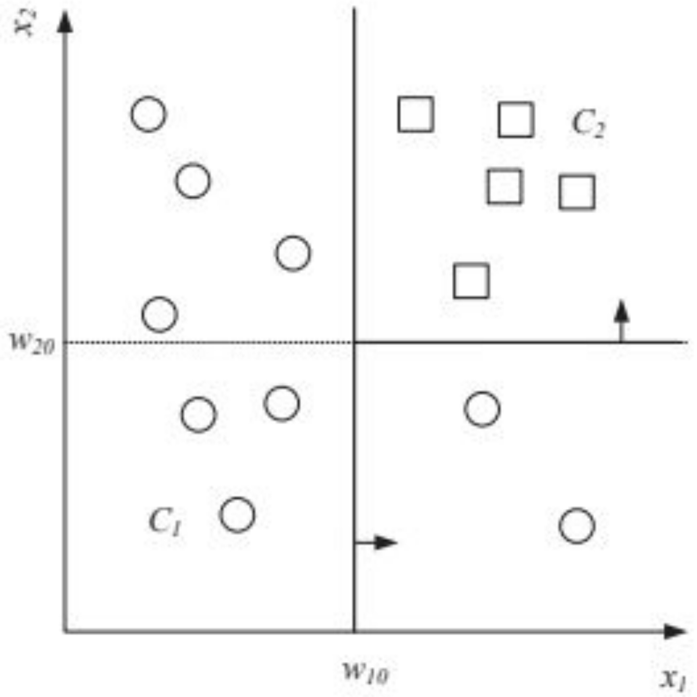
Decision trees which are also modernly known as classification and regression trees (CART) were introduced by Leo Breiman to refer, Decision Tree algorithms. They are supervised learning algorithm which has a pre-defined target variable. In other words, they are adaptable for solving any kind of problem at hand (classification or regression).

They empower predictive modeling with higher accuracy, better stability and provide ease of interpretation. Unlike linear modeling techniques, they map non-linear relationships quite well.

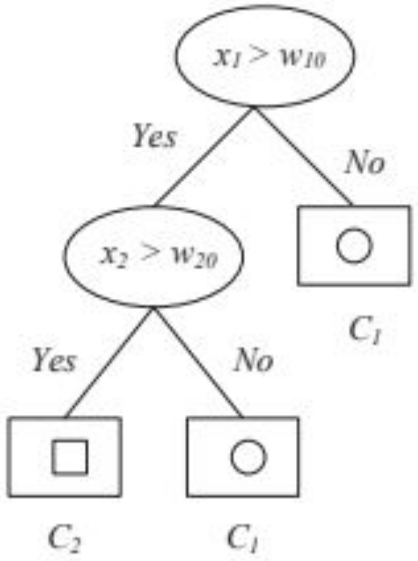
The Basics of Decision Trees

A decision tree consists of the root /Internal node which further splits into decision nodes/branches, depending on the outcome of the branches the next branch or the terminal /leaf nodes are formed.

Given an input, at each node, a test is applied and one of the branches is taken depending on the outcome. This process starts at the root and is repeated recursively until a leaf node is hit, at which point the value written in the leaf constitutes the output.



x_2
C_2
w_{20}
C_1
w_{10}
x_1



$x_1 > w_{10}$
Yes
No
$x_2 > w_{20}$
C_1
Yes
No
C_2
C_1

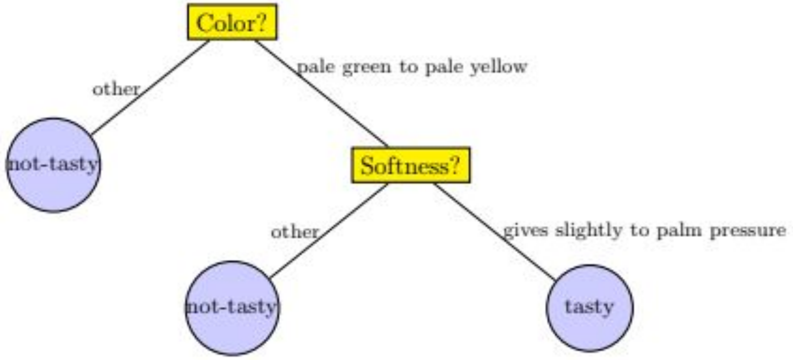
A decision tree is also a nonparametric model in the sense that we do not assume any predetermined set of parameters before training can start as in parametric models - rather the tree structure is not fixed a priori but the tree grows, branches and leaves are added, during learning depending on the complexity of the problem inherent in the data.

Decision trees can be applied to both regression and classification problems.

Classification Trees: A classification tree is an algorithm where the target variable is fixed or categorical. The algorithm is then used to identify the "class" within which a target variable would most likely fall.

An example of a classification-type problem would be determining who will or will not subscribe to a digital platform; or who will or will not graduate from high school. In other cases, you might have to predict among a number of different variables. For instance, you may have to predict which type of smartphone a consumer may decide to purchase.

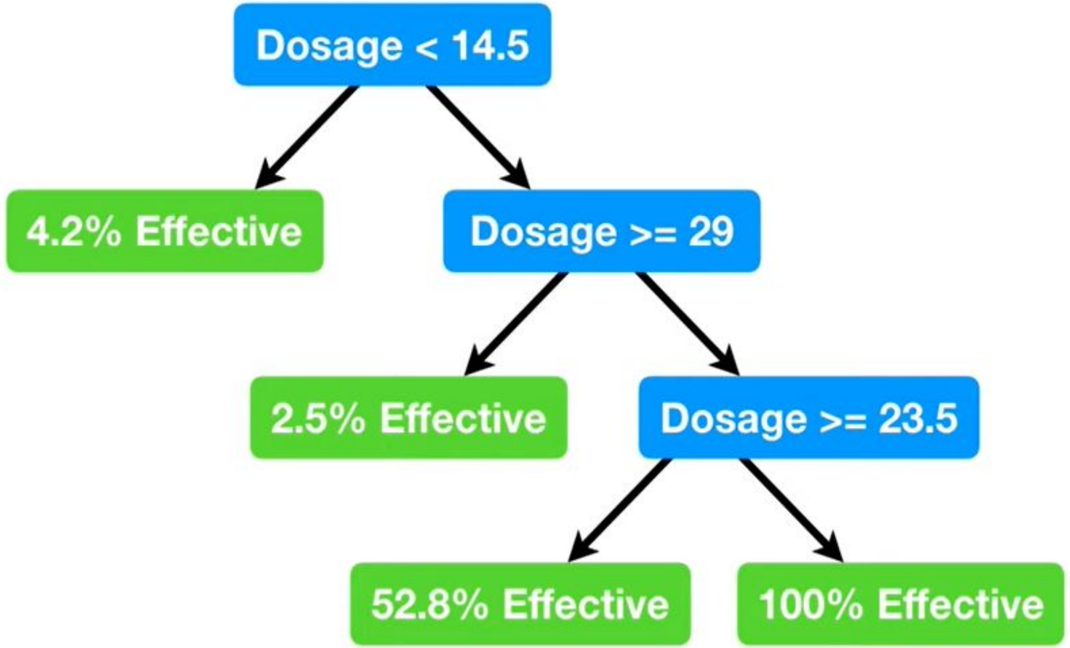
To check if a given papaya is tasty or not, the decision tree first examines the color of the Papaya. If this color is not in the range pale green to pale yellow, then the tree immediately predicts that the papaya is not tasty without additional tests. Otherwise, the tree turns to examine the softness of the papaya.If the softness level of the papaya is such that it gives slightly to palm pressure, the decision tree predicts that the papaya is tasty. Otherwise, the prediction is “not-tasty.”



Regression Trees

A regression tree refers to an algorithm where the target variable is and the algorithm is used to predict its value. As an example of a regression type problem, you may want to predict the selling prices of a residential house, which is a continuous dependent variable.

This will depend on both continuous factors like square footage as well as categorical factors like the style of home, area in which the property is located, and so on.



Dosage	Drug Effect.
10	58
20	60
35	57
5	44
etc...	etc...

The advantages of decision trees include :

Decision trees are:

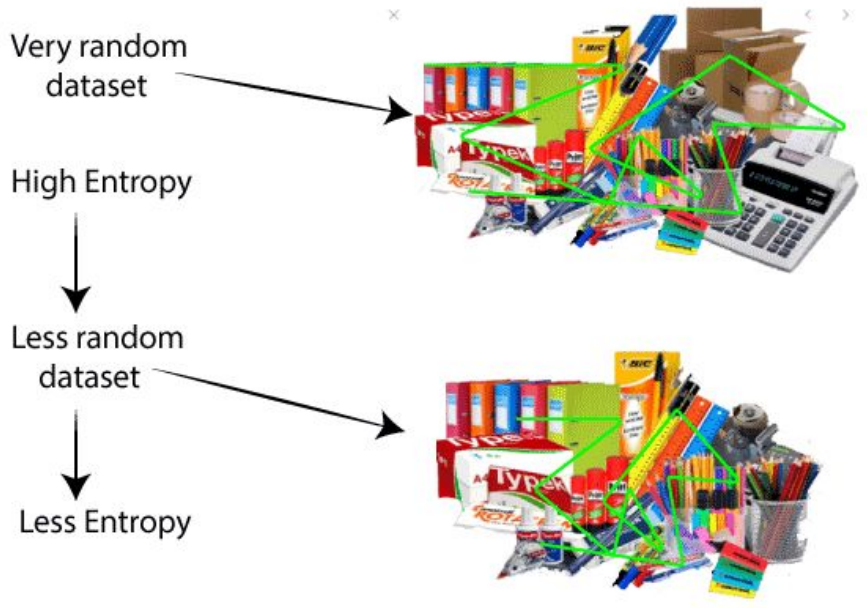
- Efficient in dealing with non-linear data
- Can handle complex relationships among the features
- Easy to understand, interpret and visualize
- Easy to implement — a minuscule amount of effort is needed for data preparation (eg- one doesn’t need to worry much about outliers, data scaling, etc.)

The disadvantages of decision trees include :

- Decision-tree learners can easily overfit thereby learning extremely complex relations that do not generalize well for the data.
- Decision trees can be unstable because small variations in the data might result in a completely different tree being generated.
- Decision tree learners create biased trees if some classes dominate. It is therefore recommended to balance the dataset prior to fitting with the decision tree.

Tree induction – split algorithm based on information theory

- **Entropy** refers to a common way to measure impurity. In the decision tree, it measures the randomness or impurity in data sets.



Mathematically the formula for entropy is: $E = -\sum_{i=1}^{N} p_i log_2 p_i$

pi is the probability of randomly selecting an example in class i.

Let's have an example to better our understanding of entropy and its calculation. Let's have a dataset made up of three colors; red, purple, and yellow. If we have one red, three purple, and four yellow observations in our set, our equation becomes:

$$E = -(p_r log_2 p_r + p_p log_2 p_p + p_y log_2 p_y)$$

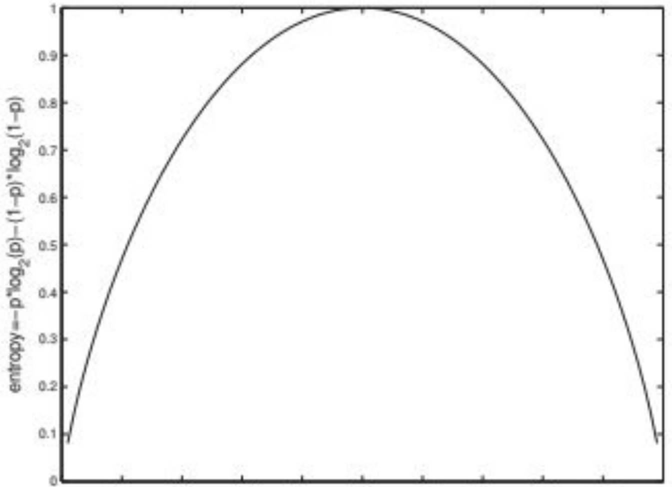
Where, pr, pp and py are the probabilities of choosing a red, purple and yellow example respectively. Our equation now becomes:

$$E = -(\frac{1}{8} log_2(\frac{1}{8}) + \frac{3}{8} log_2(\frac{3}{8}) + \frac{4}{8} log_2(\frac{4}{8}))$$

Our entropy would be: 1.41

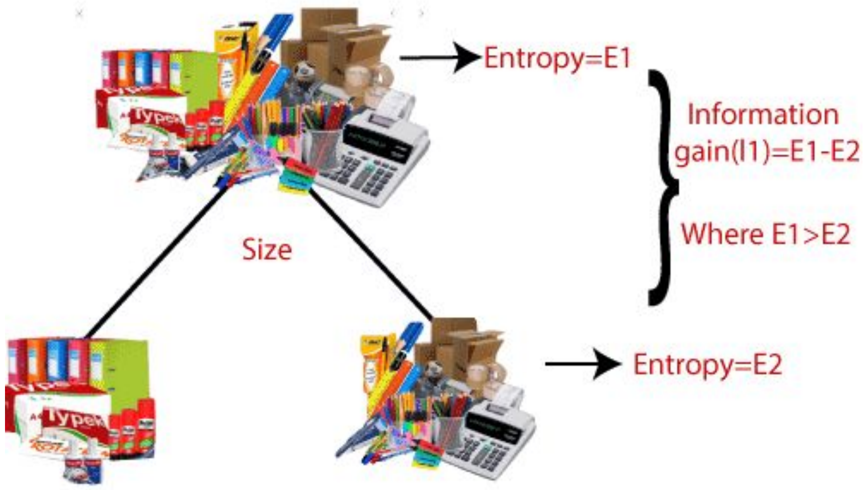
What happens when all observations belong to the same class? In such a case, the entropy will always be zero. Such a dataset has no impurity. This implies that such a dataset would not be useful for learning.

However, if we have a dataset with say, two classes, half made up of yellow and the other half being purple, the entropy will be one. This kind of dataset is good for learning.



Information Gain

Information Gain refers to the decline in entropy after the dataset is split. It is also called Entropy Reduction. Building a decision tree is all about discovering attributes that return the highest data gain.



We can define information gain as a measure of how much information a feature provides about a class. Information gain helps to determine the order of attributes in the nodes of a decision tree.

The calculation of information gain should help us understand this concept better.

$$Gain = E_{parent} - E_{children}$$

The term Gain represents information gain. E_parent is the entropy of the parent node and **E_{children} is the average entropy of the child nodes.**

Let’s use an example to visualize information gain and its calculation.

Suppose we have a dataset with two classes. This dataset has 5 purple and 5 yellow examples. The initial value of entropy will be given by the equation below. Since the dataset is balanced, we expect the answer to be 1.

Say we split the dataset into two branches. One branch ends up having four values while the other has six. The left branch has four purples while the right one has five yellows and one purple.

the entropy is zero since the dataset is pure. As such, the entropy of the left branch E_left=0. On the other hand, the right branch has five yellows and one purple. E_right=0.65

Let us calculate the quality of the split. The one on the left has 4, while the other has 6 out of a total of 10. Therefore, the weighting goes as shown below:

E_split=0.6∗0.65+0.4∗0=0.39

The entropy before the split, which we referred to as initial entropy E_initial=1. After splitting, the current value is 0.39. We can now get our information gain, which is the entropy we “lost” after splitting.

Gain=1–0.39=0.61

The more the entropy removed, the greater the information gain. The higher the information gain, the better the split.

Decision Trees: ID3 Algorithm

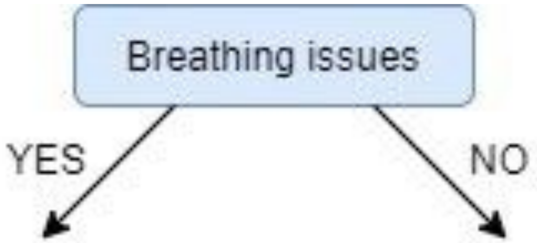
ID3 stands for Iterative Dichotomiser 3 and is named such because the algorithm iteratively (repeatedly) dichotomizes(divides) features into two or more groups at each step.
Invented by Ross Quinlan, ID3 uses a top-down greedy approach to build a decision tree. In simple words, the top-down approach means that we start building the tree from the top and the greedy approach means that at each iteration we select the best feature at the present moment to create a node.
Most generally ID3 is only used for classification problems with nominal features (Nominal data is data that can be labelled or classified into mutually exclusive categories within a variable) only.

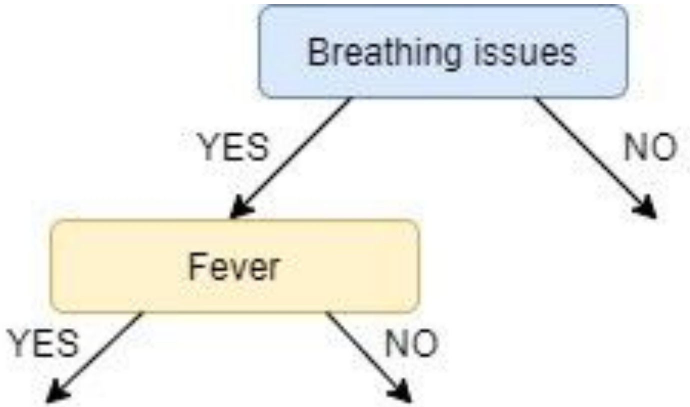
ID3 Steps

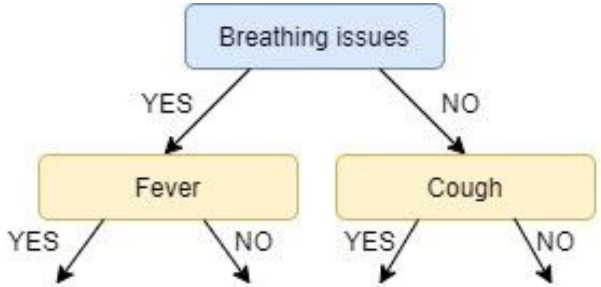
1. Calculate the Information Gain of each feature.
2. Considering that all rows don't belong to the same class, split the dataset S into subsets using the feature for which the Information Gain is maximum.
3. Make a decision tree node using the feature with the maximum Information gain.
4. If all rows belong to the same class, make the current node as a leaf node with the class as its label.
5. Repeat for the remaining features until we run out of all features, or the decision tree has all leaf nodes.

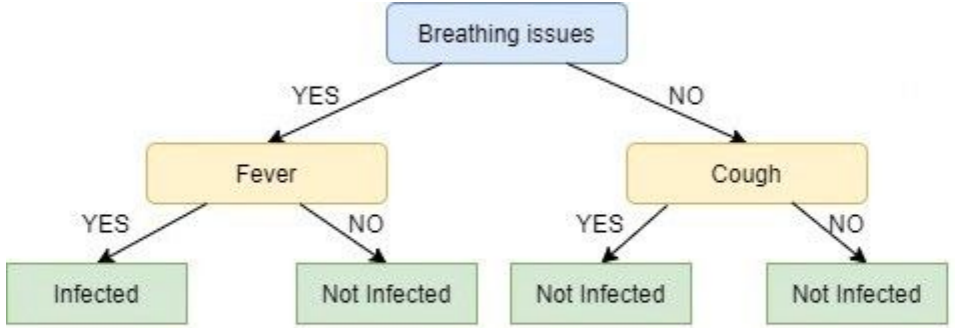
ID3 Algorithm on COVID Detection

https://docs.google.com/spreadsheets/d/1UfDW1pNvrr-oWK3D3GnTUFWo496AnhOqnuhVZjvZdak/edit#gid=0





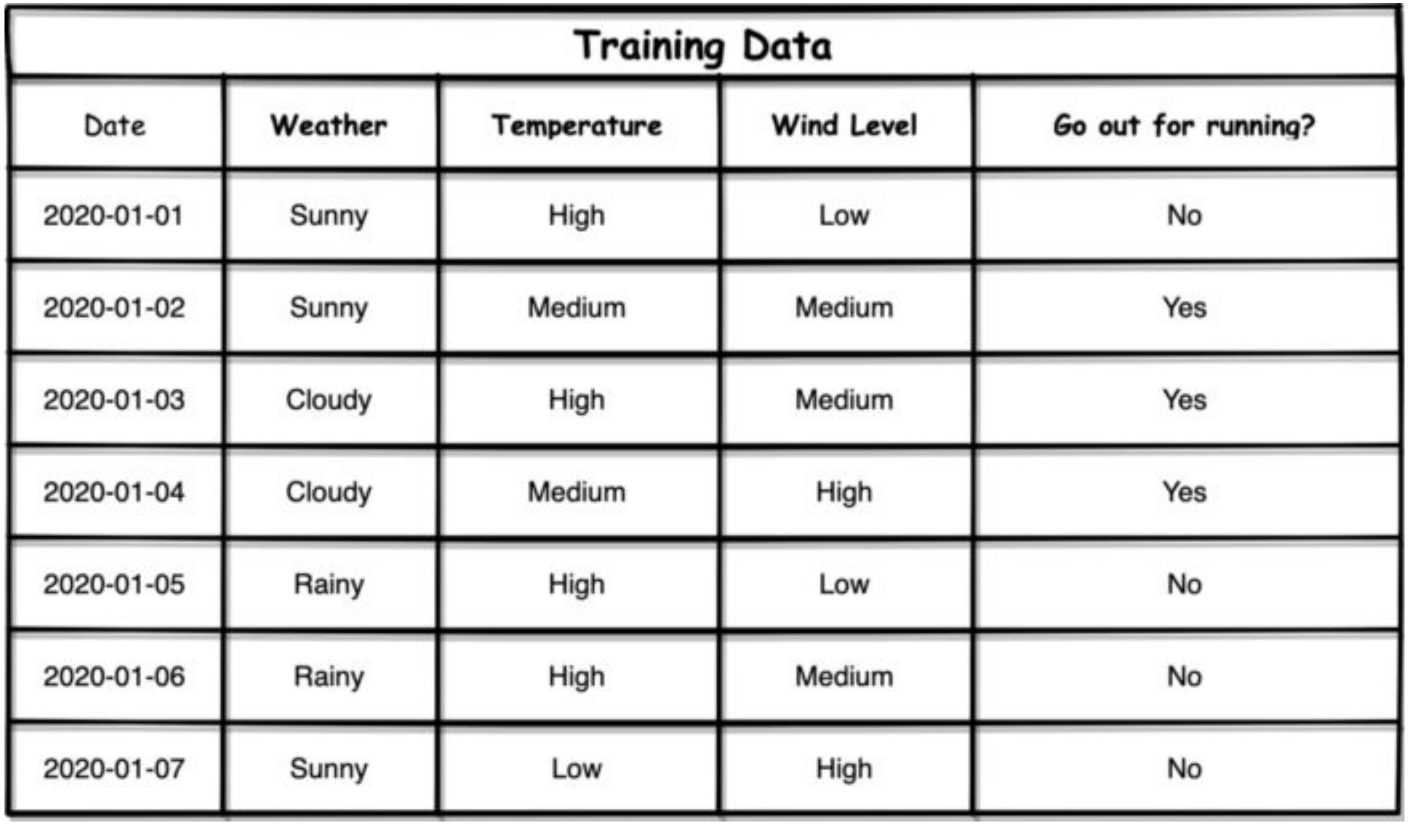




Looks Strange, doesn’t it?

I know! The right node of Breathing issues is as good as just a leaf node with class ‘Not infected’. This is one of the Drawbacks of ID3, it doesn’t do pruning.

C4.5 Algorithm



Training Data				
Date	Weather	Temperature	Wind Level	Go out for running?
2020-01-01	Sunny	High	Low	No
2020-01-02	Sunny	Medium	Medium	Yes
2020-01-03	Cloudy	High	Medium	Yes
2020-01-04	Cloudy	Medium	High	Yes
2020-01-05	Rainy	High	Low	No
2020-01-06	Rainy	High	Medium	No
2020-01-07	Sunny	Low	High	No

Feature “Date” should not be considered in this case because it intuitively will not be helpful to decide whether we should go out for running or not. However, practically, we may have much more complicated dataset to be classified, and we may not be able to understand all the features. So, we may not always be able to determine whether a feature does make sense or not. In here, I will just use “Date” as an example.

Now, let’s calculate the Information Gain for “Date”. We can start to calculate the entropy for one of the dates, such as “2020–01–01”.

$$Entropy(Date = 2020\text{-}01\text{-}01) = -(\frac{1}{1} log_2 \frac{1}{1}) = 0$$

Since there is only 1 row for each date, the final decision must be either “Yes” or “No”. So, the entropy must be 0! In terms of the information theory, it is equivalent to say: The date tells us nothing, because the result is just one, which is certain. So, there is no “uncertainty” at all.

Now, let’s calculate the entropy for the date itself.

$$Entropy(Date) = -(\tfrac{1}{7}log_2\tfrac{1}{7}+\ldots+\tfrac{1}{7}log_2\tfrac{1}{7})$$

$$Entropy(Date) = -(\tfrac{1}{7}log_2\tfrac{1}{7}) \times 7$$

$$Entropy(Date) = 2.807$$

that is a pretty large number compared to the other features. So, we can calculate the Information Gain of “Date” now

$$Gain(Date, a) = Entropy(Date) - (\tfrac{1}{7} Entropy(Date = 2020\text{-}01\text{-}01)) \times 7$$

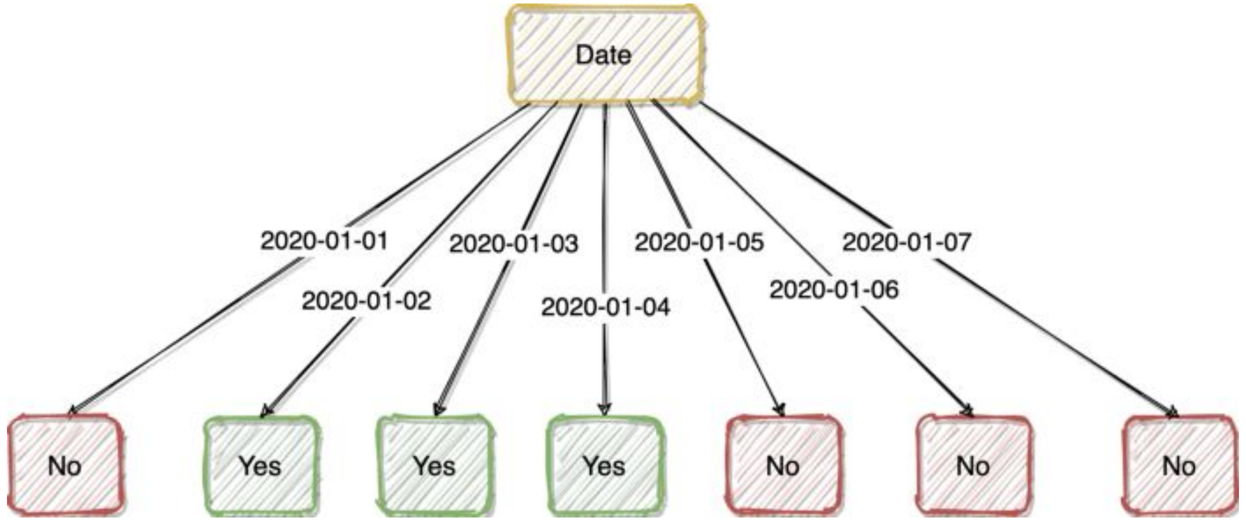
$$Gain(Date, a) = 2.807 - \tfrac{1}{7} \times 0 \times 7$$

$$Gain(Date, a) = 2.807$$

If we calculate the Information Gain for the other three features (you can find details in the article that is linked in the introduction), they are:

- Information Gain of Weather is 0.592
- Information Gain of Temperature is 0.522
- Information Gain of Wind Level is 0.306

Obviously, the Information Gain of Date is overwhelmingly larger than the others.



Algorithm is from another Decision Tree algorithm called C4.5. The basic idea of reducing this issue is to use Information Gain Ratio rather than Information Gain. Information Gain Ratio is simply adding a penalty on the Information Gain by dividing with the entropy of the parent node.

$$GainRatio(T, a) = \frac{Gain(T, a)}{Entropy(T)}$$

$$GainRatio(T, a) = \frac{\left(Entropy(T) - \sum_{i=1}^{|a|} \frac{|a_i|}{|T|} Entropy(a_i)\right)}{Entropy(T)}$$

GINI INDEX

Gini Index, also known as Gini impurity, calculates the amount of probability of a specific feature that is classified incorrectly when selected randomly. If all the elements are linked with a single class then it can be called pure.

- The Gini Index, like the properties of entropy, the Gini index varies between values 0 and 1.
- 0 expresses the purity of classification, i.e. All the elements belong to a specified class or only one class exists there.
- 1 indicates the random distribution of elements across various classes.
- The value of 0.5 of the Gini Index shows an equal distribution of elements over some classes.

The Gini Index is determined by deducting the sum of squared of probabilities of each class from one, mathematically, Gini Index can be expressed as:

$$\text{Gini Index} = 1 - \sum_{i=1}^{n} (P_i)^2$$

Where Pi denotes the probability of an element being classified for a distinct class.

Let’s have an example to better our understanding of entropy and its calculation. Let’s have a dataset made up of three colors; red, purple, and yellow. If we have one red, three purple, and four yellow observations in our set, then Gini Index will be

$$GiniIndex = 1 - (\frac{1}{64} + \frac{9}{64} + \frac{16}{64}) = 0.593$$

CART

CART Algorithm is an abbreviation of Classification And Regression Trees. It was invented by Breiman et al. in 1984.

It is generally very similar to C4.5, but have the following major characteristics:

- Rather than general trees that could have multiple branches, CART makes use binary tree, which has only two branches from each node.
- CART use Gini Impurity as the criterion to split node, not Information Gain.
- CART supports numerical target variables, which enables itself to become a Regression Tree that predicts continuous values.

The Gini coefficient will guide the CART algorithm to find the node with larger uncertainty (i.e. impurity) and then split it.

CART algorithm makes use a binary tree, each time we need to calculate the weighted sum of the Gini coefficients for two nodes only. The formula is as follows:

$$GINI(T) = \frac{|t_{\text{left}}|}{|T|} \times Gini(t_{\text{left}}) + \frac{|t_{\text{right}}|}{|T|} \times Gini(t_{\text{right}})$$

- |t_left| and |t_right| are the sample size of the nodes on the left and right respectively.
- |T| is the sample size of the candidate parent node T.

CART Example

There are 14 instances of golf playing decisions based on outlook, temperature, humidity and wind factors. We will apply CART algorithm on the data to develop the tree.

Link: https://docs.google.com/spreadsheets/d/1UfDW1pNvrr-oWK3D3GnTUFWo496AnhOqnuhVZjvZdak/edit#gid=473835911

Source: https://sefiks.com/2018/08/27/a-step-by-step-cart-decision-tree-example/

Gini Index vs Information Gain

1. The Gini Index facilitates the bigger distributions so easy to implement whereas the Information Gain favors lesser distributions having small count with multiple specific values.
2. The method of the Gini Index is used by CART algorithms, in contrast to it, Information Gain is used in ID3, C4.5 algorithms.
3. Gini index operates on the categorical target variables in terms of “success” or “failure” and performs only binary split, in opposite to that Information Gain computes the difference between entropy before and after the split and indicates the impurity in classes of elements.

Build Decision Trees

Step-1: Begin the tree with the root node, says S, which contains the complete dataset.

Step-2: Find the best attribute in the dataset using Attribute Selection Measure (ASM).

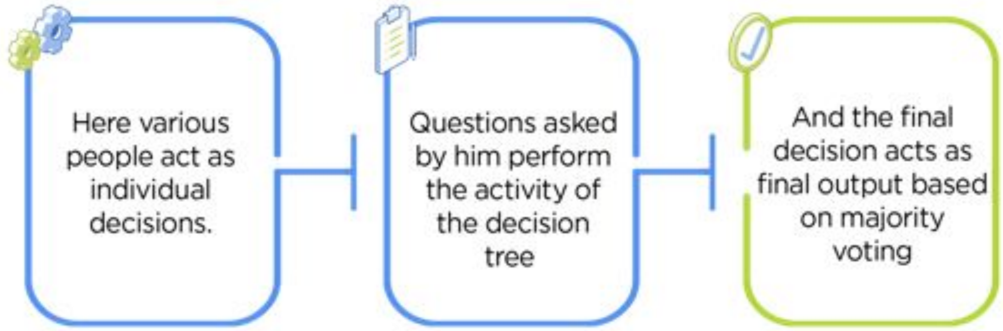
Step-3: Divide the S into subsets that contains possible values for the best attributes.

Step-4: Generate the decision tree node, which contains the best attribute.

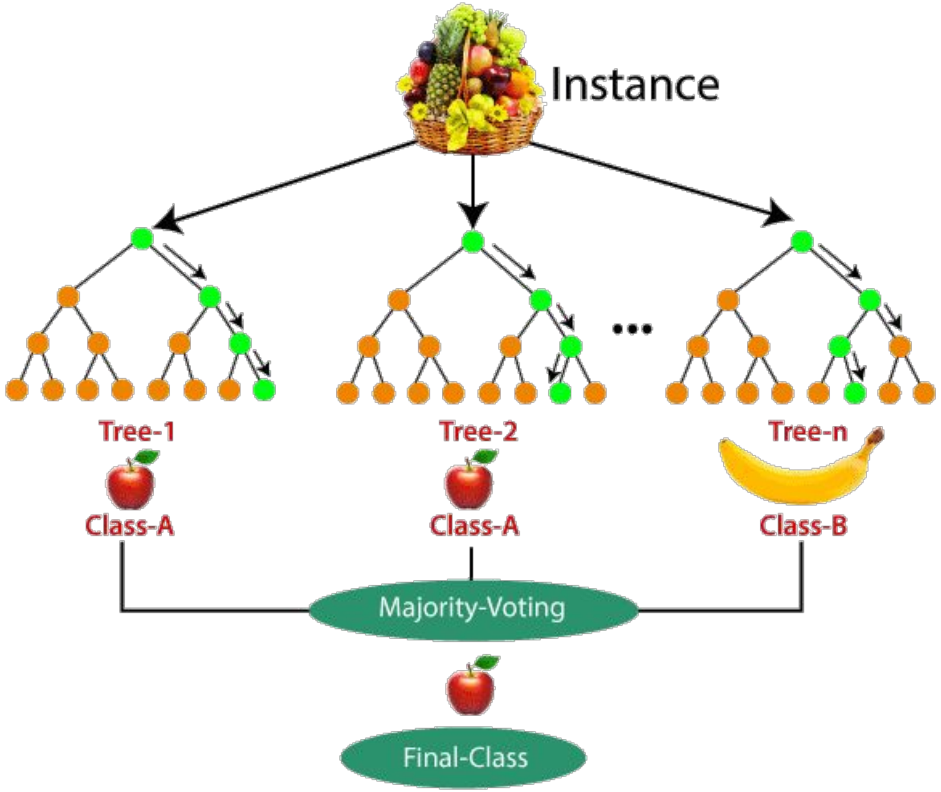
Step-5: Recursively make new decision trees using the subsets of the dataset created in step -3. Continue this process until a stage is reached where you cannot further classify the nodes and called the final node as a leaf node.

Random Forest Classification

Let's dive into a real-life analogy to understand this concept further. A student named X wants to choose a course after his 10+2, and he is confused about the choice of course based on his skill set. So he decides to consult various people like his cousins, teachers, parents, degree students, and working people. He asks them varied questions like why he should choose, job opportunities with that course, course fee, etc. Finally, after consulting various people about the course he decides to take the course suggested by most of the people.


Here various people act as individual decisions.
Questions asked by him perform the activity of the decision tree
And the final decision acts as final output based on majority voting

Random forest, like its name implies, consists of a large number of individual decision trees that operate as an ensemble (Ensemble algorithms are those which combines more than one algorithms of same or different kind for classifying objects.). Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model's prediction.



The Random Forest Classifier is a set of decision trees from randomly selected subset of training set. It aggregates the votes from different decision trees to decide the final class of the test object.

The fundamental concept behind random forest is a simple but powerful one — the wisdom of crowds. In data science speak, the reason that the random forest model works so well is:

> ***A large number of relatively uncorrelated models (trees) operating as a committee will outperform any of the individual constituent models.***

Why Uncorrelated Outcomes are So Great

Let us try to understand it with an example:

Imagine that we are playing the following game:

- I use a uniformly distributed random number generator to produce a number.
- If the number I generate is greater than or equal to 40, you win (so you have a 60% chance of victory) and I pay you some money. If it is below 40, I win and you pay me the same amount.

Now I offer you the the following choices. We can either:

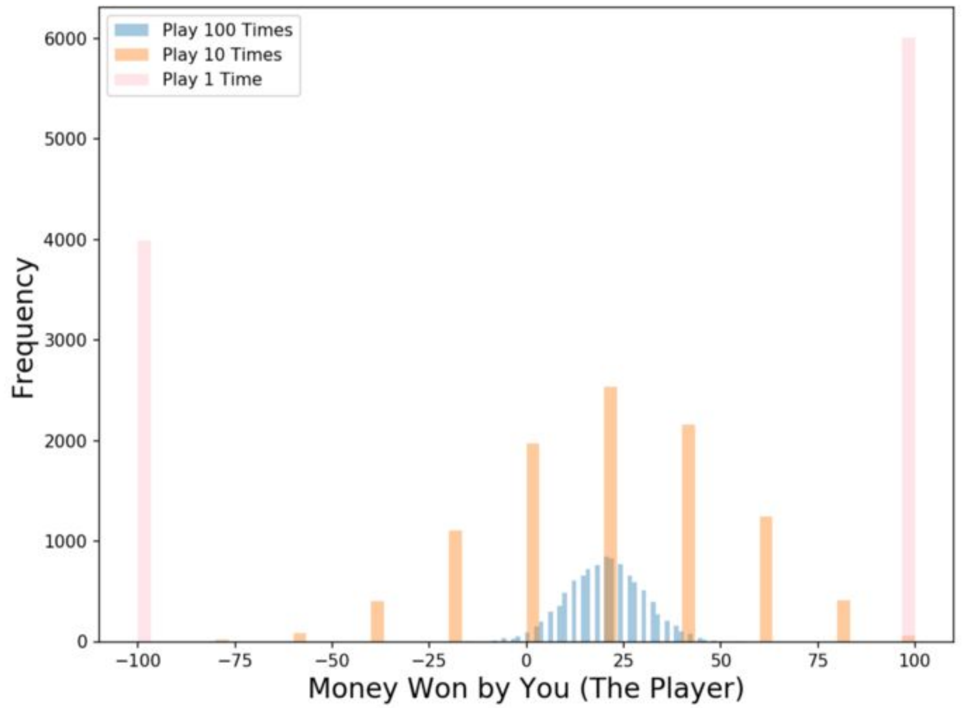
- **Game 1** — play 100 times, betting $1 each time.
- **Game 2**— play 10 times, betting $10 each time.
- **Game 3**— play one time, betting $100.

Which would you pick? The expected value of each game is the same:

- Expected Value Game 1 = (0.60*1 + 0.40*-1)*100 = 20
- Expected Value Game 2= (0.60*10 + 0.40*-10)*10 = 20
- Expected Value Game 3= 0.60*100 + 0.40*-100 = 20

Let’s visualize the results with a Monte Carlo simulation (we will run 10,000 simulations of each game type.

Even though the expected values are the same, **the outcome distributions are vastly different going from positive and narrow (blue) to binary (pink).**



Game 1 (where we play 100 times) offers up the best chance of making some money — out of the 10,000 simulations that I ran, you make money in 97% of them!

Game 2 (where we play 10 times) you make money in 63% of the simulations, a drastic decline (and a drastic increase in your probability of losing money).

Game 3 that we only play once, you make money in 60% of the simulations, as expected.

So even though the games share the same expected value, their outcome distributions are completely different. The more we split up our $100 bet into different plays, the more confident we can be that we will make money. As mentioned previously, this works because each play is independent of the other ones.

Random forest is the same — each tree is like one play in our game earlier. We just saw how our chances of making money increased the more times we played. Similarly, with a random forest model, our chances of making correct predictions increase with the number of uncorrelated trees in our model.

Bagging (Bootstrap Aggregation)

Decisions trees are very sensitive to the data they are trained on — small changes to the training set can result in significantly different tree structures. Random forest takes advantage of this by allowing each individual tree to randomly sample from the dataset with replacement, resulting in different trees. **This process is known as bagging.**

We are not subsetting the training data into smaller chunks and training each tree on a different chunk. Rather, if we have a sample of size N, we are still feeding each tree a training set of size N (unless specified otherwise). But instead of the original training data, we take a random sample of size N with replacement. For example, if our training data was [1, 2, 3, 4, 5, 6] then we might give one of our trees the following list [1, 2, 2, 3, 6, 6].

Feature Randomness

In a normal decision tree, when it is time to split a node, we consider every possible feature and pick the one that produces the most separation between the observations in the left node vs. those in the right node. In contrast, each tree in a random forest can pick only from a random subset of features. This forces even more variation amongst the trees in the model and ultimately results in lower correlation across trees and more diversification.

In random forest, we end up with trees that are not only trained on different sets of data (thanks to bagging) but also use different features to make decisions.

The random forest is a classification algorithm consisting of many decisions trees. It uses bagging and feature randomness when building each individual tree to try to create an uncorrelated forest of trees whose prediction by committee is more accurate than that of any individual tree.