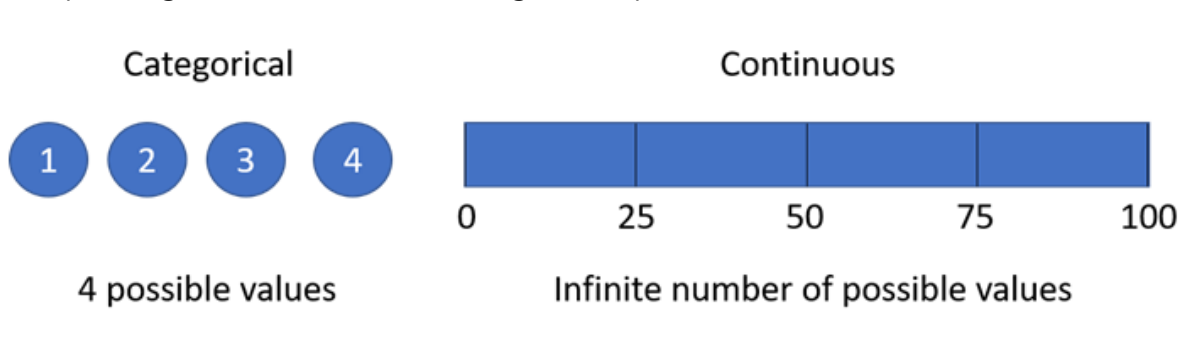
Introduction about Decision Tree.

Trees have long been a subject of interest and a topic of discussion — and it's no wonder; they represent life, growth, peace, and nature. Trees provide us with many benefits necessary for survival, including clean air, filtered water, shade, and food. They also give us hope and insight, and courage to persevere — even in the harshest conditions. Trees teach us to stay rooted while soaring to great heights. As we know the tree has been useful to us in many different forms, but in the recent times, its structure has given us inspiration for an algorithm to solve problems and make a machine learn things we want them to learn. Yes, I am talking about "Tree Based Algorithm". Tree based learning algorithms are considered to be one of the best and mostly used supervised learning methods. Tree based methods empower predictive models with high accuracy, stability and ease of interpretation. Unlike linear models, they map non-linear relationships quite well. They are adaptable at solving any kind of problem at hand (classification or regression). Methods like decision trees, random forest, gradient boosting are being popularly used in all kinds of data science problems. Hence, for every analyst (fresher also), it's important to learn these algorithms and use them for modelling. This tutorial is meant to help beginners learn "Decision Tree Algorithm", which is the stepping stone to learning tree-based modelling, from scratch. After the successful completion of this tutorial, one is expected to become proficient at using Decision Tree Algorithm and build predictive models. This post is going to cover everything you need to start with so it will be a long one...

Introduction

As a marketing manager, you want a set of customers who are most likely to purchase your product. This is how you can save your marketing budget by finding your audience. As a loan manager, you need to identify risky loan applications to achieve a lower loan default rate. This process of classifying customers into a group of potential and non-potential customers or safe or risky loan applications is known as a classification problem. Classification is a two-step process, learning step and prediction step. In the learning step, the model is developed based on given training data. In the prediction step, the model is used to predict the response for given data. Decision Tree is one of the easiest and popular classification algorithms to understand and interpret. For the practical part of this topic, you may read about it in this post. Decision Trees can be utilized for both classification and regression kind of problem. Regression-type problems. Regression-type problems are generally those where we attempt to predict the values of a continuous variable from one or more continuous and/or categorical predictor variables. For example, we may want to predict the selling prices of single-family homes (a continuous dependent variable) from various other continuous predictors (e.g., square footage) as well as categorical predictors (e.g., style of home, such as ranch, two-story, etc.; zip code or telephone area code where the property is located, etc.; note that this latter variable would be categorical in nature, even though it would contain numeric values or codes). If we used simple multiple regression, or some general linear model (GLM) to predict the selling prices of single-family homes, we would determine a linear equation for these variables that can be used to compute predicted selling prices.

Classification-type problems. Classification-type problems are generally those where we attempt to predict values of a categorical dependent variable (class, group membership, etc.) from one or more continuous and/or categorical predictor variables. For example, we may be interested in predicting who will or will not graduate from college, or who will or will not renew a subscription. These would be examples of simple binary classification problems, where the categorical dependent variable can only assume two distinct and mutually exclusive values. In other cases, we might be interested in predicting which one of multiple different alternative consumer products (e.g., makes of cars) a person decides to purchase, or which type of failure occurs with different types of engines. In those cases, there are multiple categories or classes for the categorical dependent variable



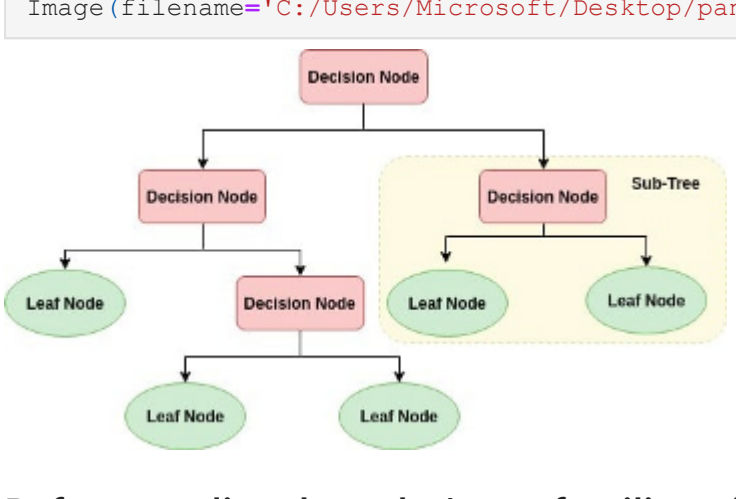
What is Decision Tree Algorithm

A decision tree is a flowchart-like tree structure where an internal node represents feature (or attribute), the branch represents a decision rule, and each leaf node represents the outcome. The topmost node in a decision tree is known as the root node. It learns to partition based on the attribute value. It partitions the tree in a recursive manner called recursive partitioning. This flowchart-like structure helps you in decision making. It's visualization like a flowchart diagram which easily mimics the human-level thinking. That is why decision trees are easy to understand and interpret.

Decision trees classify the examples by sorting them down the tree from the root to some leaf node, with the leaf node providing the classification to the example, this approach is called a Top-Down approach. Each node in the tree acts as a test case for some attribute, and each edge descending from that node corresponds to one of the possible answers to the test case. This process is recursive and is repeated for every subtree rooted at the new nodes.

In [3]:
```python
from IPython.display import Image
Image(filename='C:/Users/Microsoft/Desktop/pandas/1_eVthVN_eXjNLofZ4s4KOAw.png')
```

Out[3]:

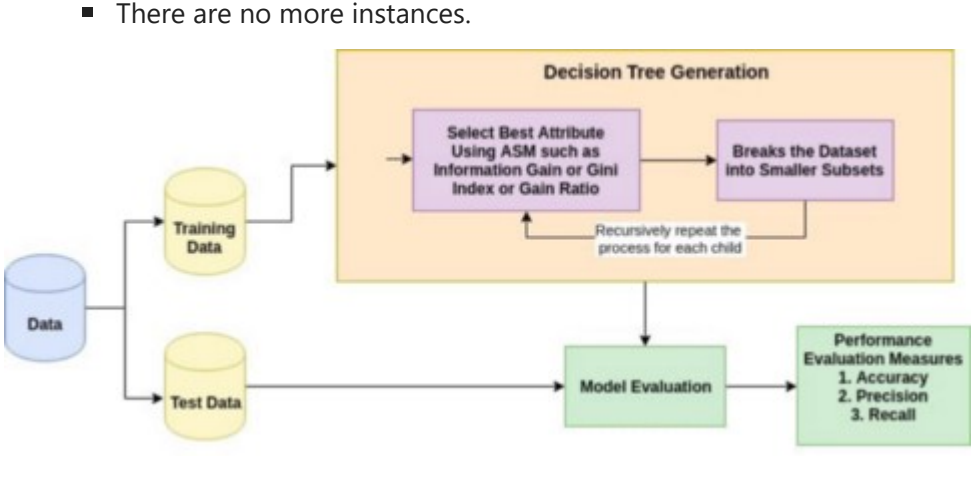


Before we dive deep, let's get familiar with some of the terminologies:

- Root Node (Top Decision Node): It represents the entire population or sample and this further gets divided into two or more homogeneous sets.
- Splitting: It is a process of dividing a node into two or more sub-nodes.
- Decision Node: When a sub-node splits into further sub-nodes, then it is called a decision node.
- Leaf/ Terminal Node: Nodes with no children (no further split) is called Leaf or Terminal node.
- Pruning: When we reduce the size of decision trees by removing nodes (opposite of Splitting), the process is called pruning.
- Branch / Sub-Tree: A sub section of the decision tree is called branch or sub-tree.
- Parent and Child Node: A node, which is divided into sub-nodes is called a parent node of sub-nodes whereas sub-nodes are the child of a parent node.

How does the Decision Tree algorithm work?

- The basic idea behind any decision tree algorithm is as follows:
- Select the best attribute using Attribute Selection Measures(ASM) to split the records.
- Make that attribute a decision node and breaks the dataset into smaller subsets.
- Starts tree building by repeating this process recursively for each child until one of the condition will match:
  - All the tuples belong to the same attribute value.
  - There are no more remaining attributes.
  - There are no more instances.



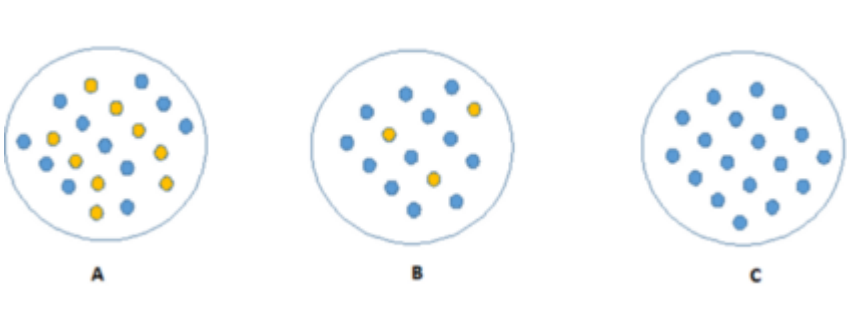
The Math Behind Decision Trees:

Attribute Selection Measures.

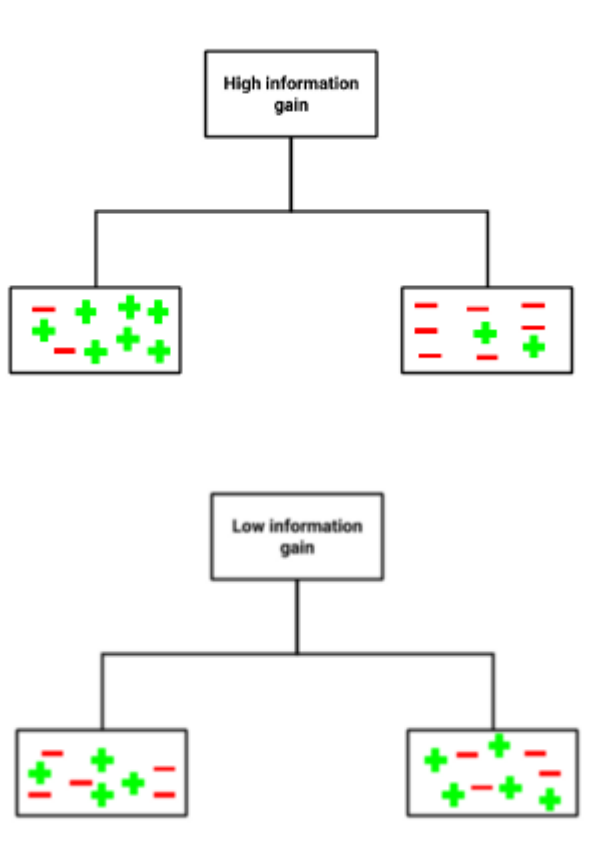
Attribute selection measure is a heuristic for selecting the splitting criterion that partition data into the best possible manner. It is also known as splitting rules because it helps us to determine breakpoints for tuples on a given node. ASM provides a rank to each feature(or attribute) by explaining the given dataset. Best score attribute will be selected as a splitting attribute (Source). In the case of a continuous-valued attribute, split points for branches also need to define. Most popular selection measures are Information Gain, Gain Ratio, and Gini Index.

Information Gain:

Information gain is a statistical property that measures how well a given attribute separates the training examples according to their target classification. Look at the image below and think which node can be described easily. I am sure, your answer is C because it requires less information as all values are similar. On the other hand, B requires more information to describe it and A requires the maximum information. In other words, we can say that C is a Pure node, B is less Impure and A is more impure.



Now, we can build a conclusion that less impure node requires less information to describe it. And, the more impure node requires more information. In a parallel example considering Information Gain, In the figure below, we can see that an attribute with low information gain (bottom) splits the data relatively evenly and as a result doesn't bring us any closer to a decision. Whereas, an attribute with high information gain (top) splits the data into groups with an uneven number of positives and negatives and as a result, helps in separating the two from each other.

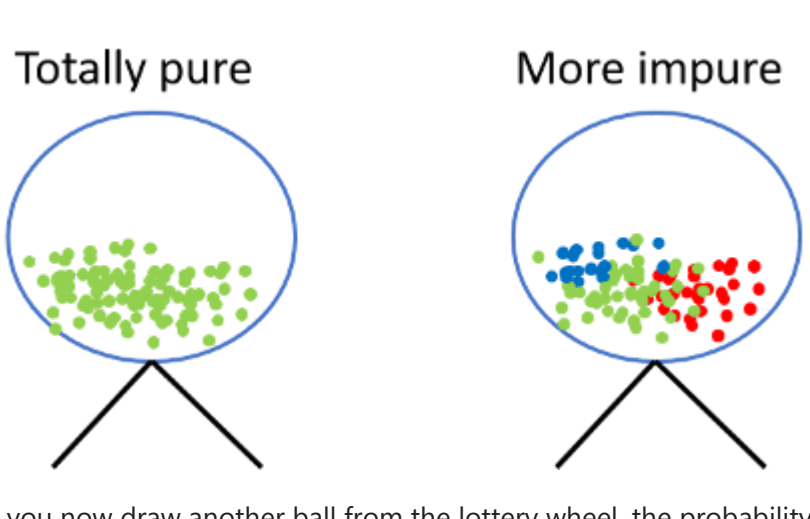


To be able to calculate the information gain, we have to first introduce the term entropy of a dataset.

Entropy

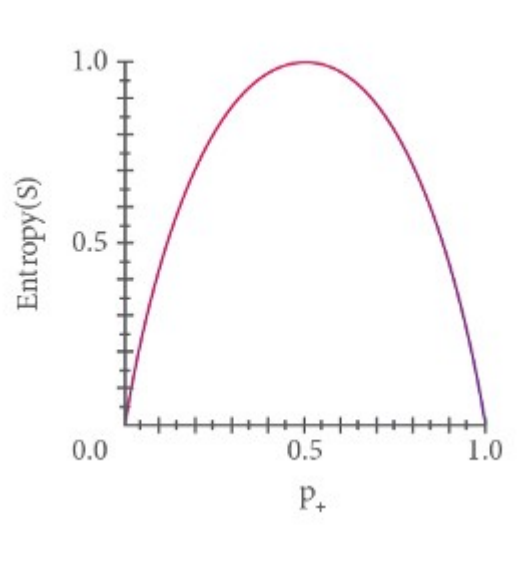
Shannon invented the concept of entropy, which measures the impurity of the input set. In physics and mathematics, entropy referred to as the randomness or the impurity in the system. In information theory, it refers to the impurity in a group of examples. Information gain is a decrease in entropy.

The idea behind the entropy is, in simplified terms, the following: Imagine you have a lottery wheel which includes 100 green balls. The set of balls within the lottery wheel can be said to be totally pure because only green balls are included. To express this in the terminology of entropy, this set of balls has an entropy of 0 (we can also say zero impurity). Consider now, 30 of these balls are replaced by red and 20 by blue balls.



If you now draw another ball from the lottery wheel, the probability of receiving a green ball has dropped from 1.0 to 0.5. Since the impurity increased, the purity decreased, hence also the entropy increased. Hence we can say, the more "impure" a dataset, the higher the entropy and the less "impure" a dataset, the lower the entropy Note that entropy is 0 if all the members of S belong to the same class. For example, if all members are positive, Entropy(S) = 0.

Entropy is 1 when the sample contains an equal number of positive and negative examples. If the sample contains an unequal number of positive and negative examples, entropy is between 0 and 1. The following figure shows the form of the entropy function relative to a boolean classification as entropy varies between 0 and 1.



Entropy can be calculated using the formula:-

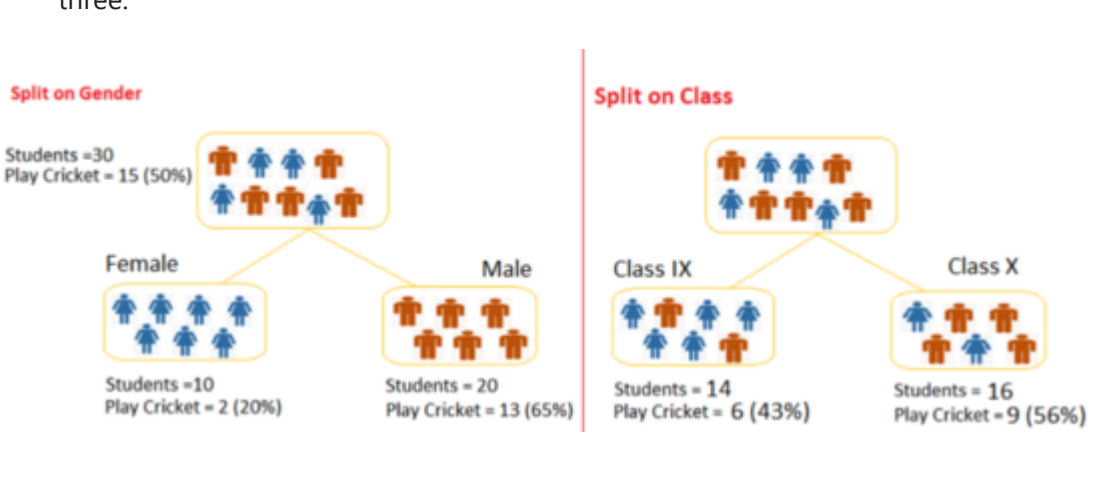
- Here p and q is the probability of success and failure respectively in that node. Entropy is also used with the categorical target variable. It chooses the split which has the lowest entropy compared to the parent node and other splits. The lesser the entropy, the better it is.

- Information gain computes the difference between entropy before split and average entropy after split of the dataset based on given attribute values.

- Information Gain = Entropy(parent node) — [Avg Entropy(children)]

- Steps to calculate entropy for a split:
- Calculate the entropy of the parent node
- Calculate entropy of each individual node of split and calculate the weighted average of all sub-nodes available in a split.

- Example: Let's say we have a sample of 30 students with three variables Gender (Boy/ Girl), Class(IX/ X) and Height (5 to 6 ft). 15 out of these 30 play cricket in leisure time. Now, I want to create a model to predict who will play cricket during a leisure period? In this problem, we need to segregate students who play cricket in their leisure time based on highly significant input variable among all three.



- Entropy for parent node = -(15/30) log2 (15/30) — (15/30) log2 (15/30) = 1
- Here 1 shows that it is a impure node.
- For Split on gender:
- Entropy for Female node = -(2/10) log2 (2/10) — (8/10) log2 (8/10) = 0.72 Entropy for Male node = -(13/20) log2 (13/20) — (7/20) log2 (7/20) = 0.93 Entropy for split Gender = (10/30)*0.72 + (20/30)*0.93 = 0.86
- Information Gain for split on gender = 1–0.86 = 0.14
- For Split on Class:
- Entropy for Class IX node = -(6/14) log2 (6/14) — (8/14) log2 (8/14) = 0.99 Entropy for Class X node = -(9/16) log2 (9/16) — (7/16) log2 (7/16) = 0.99 Entropy for split Class = (14/30)*0.99 + (16/30)*0.99 = 0.99
- Information Gain for split on Class = 1– 0.99 = 0.01
- Above, you can see that Information Gain for Split on Gender is the Highest among all, so the tree will split on Gender.

2. Gini

- Gini says, if we select two items from a population at random then they must be of same class and probability for this is 1 if population is pure.
- It works with categorical target variable "Success" or "Failure".
- It performs only Binary splits
- Higher the value of Gini higher the homogeneity.
- CART (Classification and Regression Tree) uses the Gini method to create binary splits.
- Steps to Calculate Gini for a split
- Calculate Gini for sub-nodes, using formula sum of the square of probability for success and failure (p^2+q^2).
- Calculate Gini for split using weighted Gini score of each node of that split
- Example: — Referring to the example used above, where we want to segregate the students based on the target variable (playing cricket or not). In the snapshot below, we split the population using two input variables Gender and Class. Now, I want to identify which split is producing more homogeneous sub-nodes using Gini.

Split on Gender:

- Calculate, Gini for sub-node Female = (0.2)*(0.2)+(0.8)*(0.8)=0.68
- Gini for sub-node Male = (0.65)*(0.65)+(0.35)*(0.35)=0.55
- Calculate weighted Gini for Split Gender = (10/30)*0.68+(20/30)*0.55 = 0.59
- Similar for Split on Class:
- Gini for sub-node Class IX = (0.43)*(0.43)+(0.57)*(0.57)=0.51
- Gini for sub-node Class X = (0.56)*(0.56)+(0.44)*(0.44)=0.51
- Calculate weighted Gini for Split Class = (14/30)*0.51+(16/30)*0.51 = 0.51
- Above, you can see that Gini score for Split on Gender is higher than Split on Class, hence, the node split will take place on Gender.
- You might often come across the term 'Gini Impurity' which is determined by subtracting the Gini value from 1. So mathematically we can say,
- Gini Impurity = 1-Gini
- Gini is default parameter in Decision Tree Algorithm.
- There are other measures as well but mostly you'll use this two. I personally prefer doing Information Gain :)

ANOTHER EXAMPLE.

Introduction.

This post is towards the practical side of the decision tree rather than the theory behind it. According to sources, Up to 60,000 times better, it seems. It is generally easier to understand visuals than to read entire blocks of text. In fact, a whopping 90% of the information that the brain receives is non-verbal. And of course, the first written language of humans consisted of pictures. Our brain is hardwired to understand visuals better than text. Because of this, people remember up to 80% of what they see, compared to only about 20% of what they read. So when it comes to memorability, visuals are key to your content creation. Hence in this post, we will be understanding the decision tree working on decision tree (graphical representation) itself rather than monotonous textual explanation. Then we'll use the Decision Tree Algorithm on a dataset to get familiar with solving the problem with algorithm (in Python) and visualize the tree you created. After that, I have a bonus content which is slightly skewed towards advance part, but gaining an understanding about it will make you curious to learn more about it and you'll get indulged in the learning process. So what do we know until now?

In principal decision trees can be used to predict the target feature of an unknown query instance by building a model based on existing data for which the target feature values are known (supervised learning). Additionally, we know that this model can make predictions for unknown query instances because it models the relationship between the known descriptive features and the know target feature. In our following example, the tree model learns "how a specific animal species look" respectively the combination of descriptive feature values distinctive for animal species.

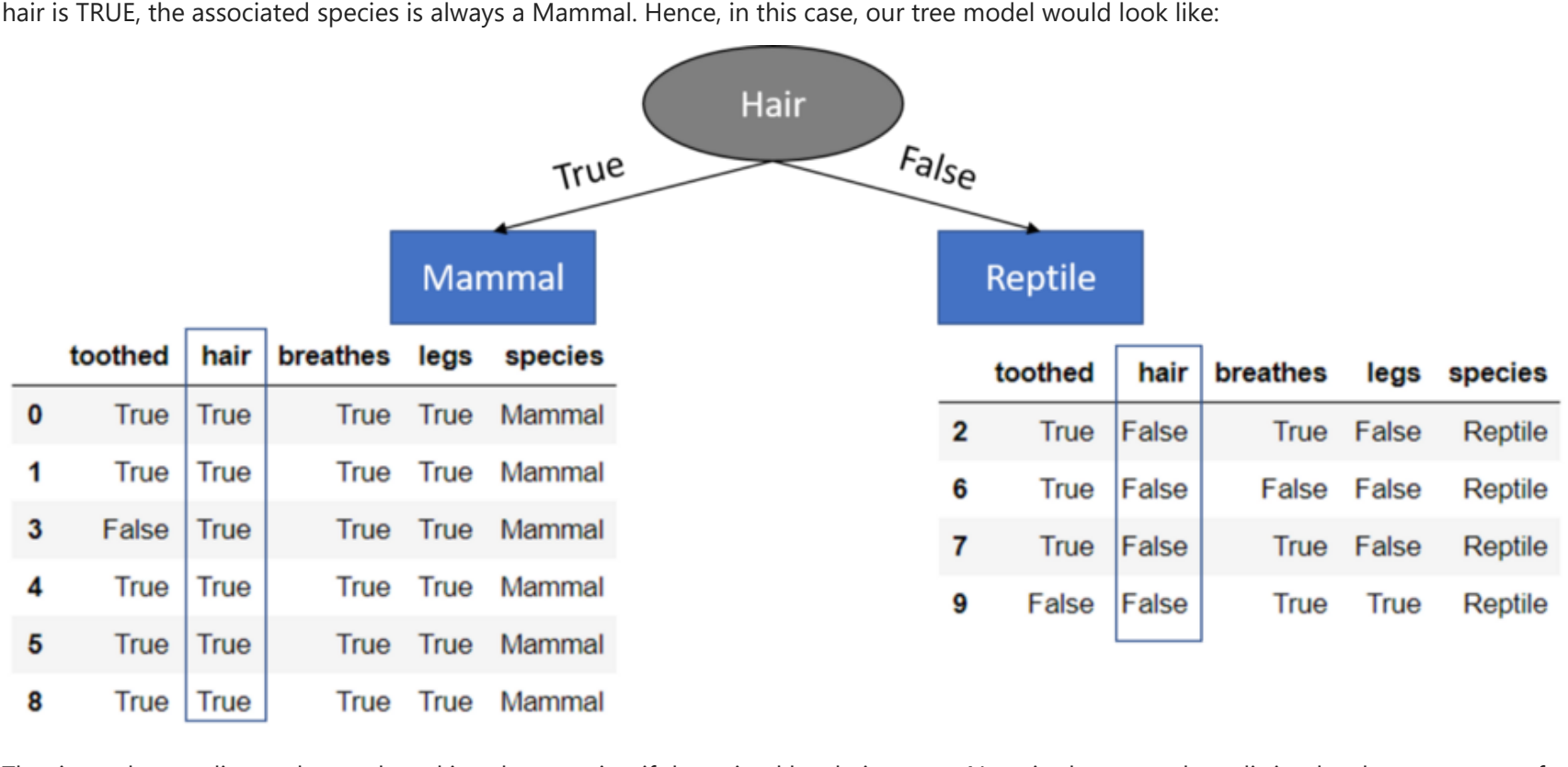
Additionally, we know that to train a decision tree model we need a dataset consisting of several training examples characterized by several descriptive features and a target feature.

Working Example of Decision Tree Algorithm.

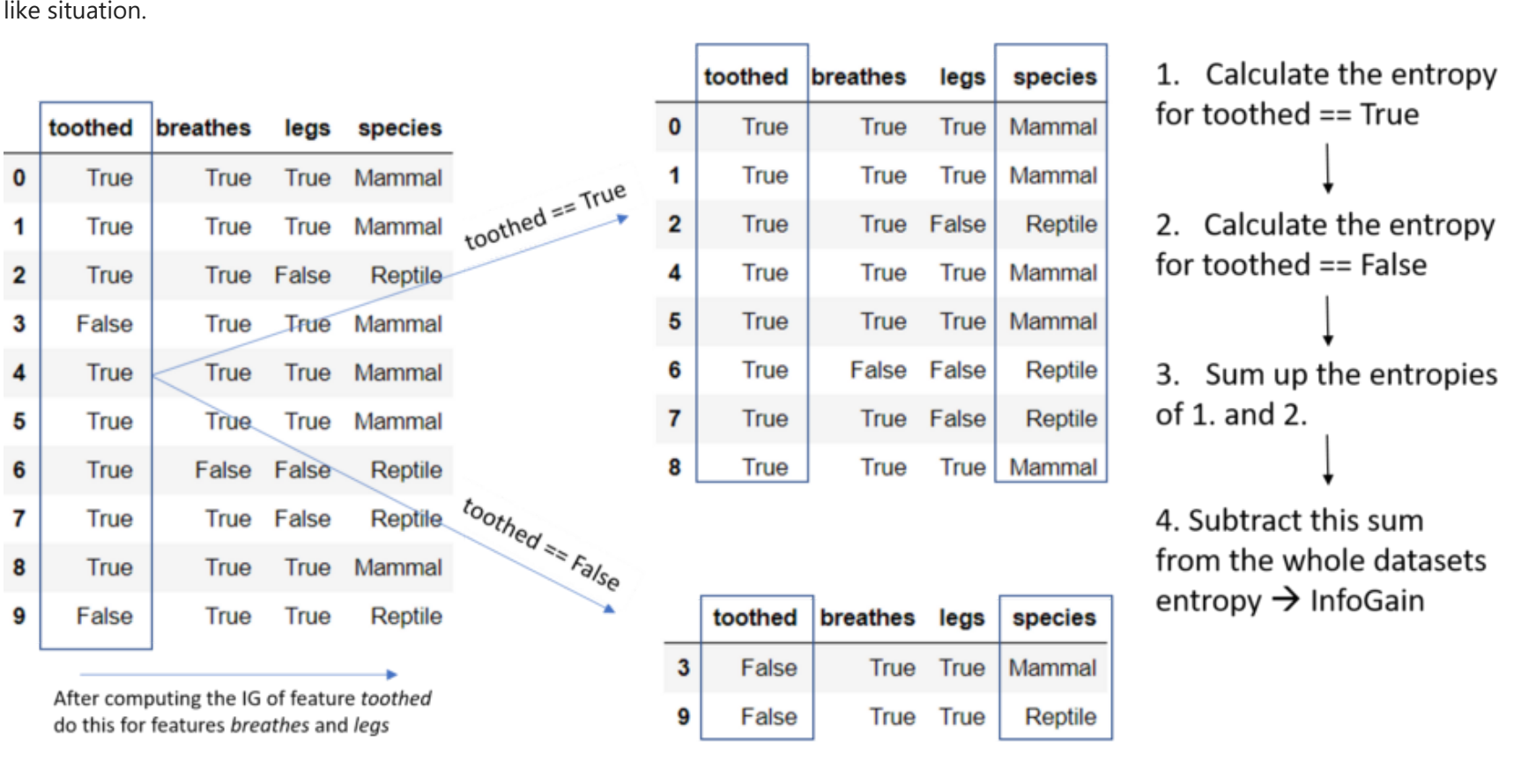
We want, given a dataset, train a model which kind of learns the relationship between the descriptive features and a target feature such that we can present the model a new, unseen set of query instances and predict the target feature values for these query instances. Let's further recapitulate the general shape of a decision tree. We know that we have at the bottom of the tree leaf nodes which contain (in the optimal case) target feature values. To make this more illustrative we use as a practical example a simplified version of the UCI machine learning Zoo Animal Classification dataset which includes properties of animals as descriptive features and the and the animal species as target feature. In our example, the animals are classified as being Mammals or Reptiles based on whether they are toothed, have legs and do breathe. The dataset looks like:

	toothed	hair	breathes	legs	species
0	True	True	True	True	Mammal
1	True	True	True	True	Mammal
2	True	False	True	False	Reptile
3	False	True	True	True	Mammal
4	True	True	True	True	Mammal
5	True	True	True	True	Mammal
6	True	False	False	False	Reptile
7	True	False	True	False	Reptile
8	True	True	True	True	Mammal
9	False	False	True	True	Reptile

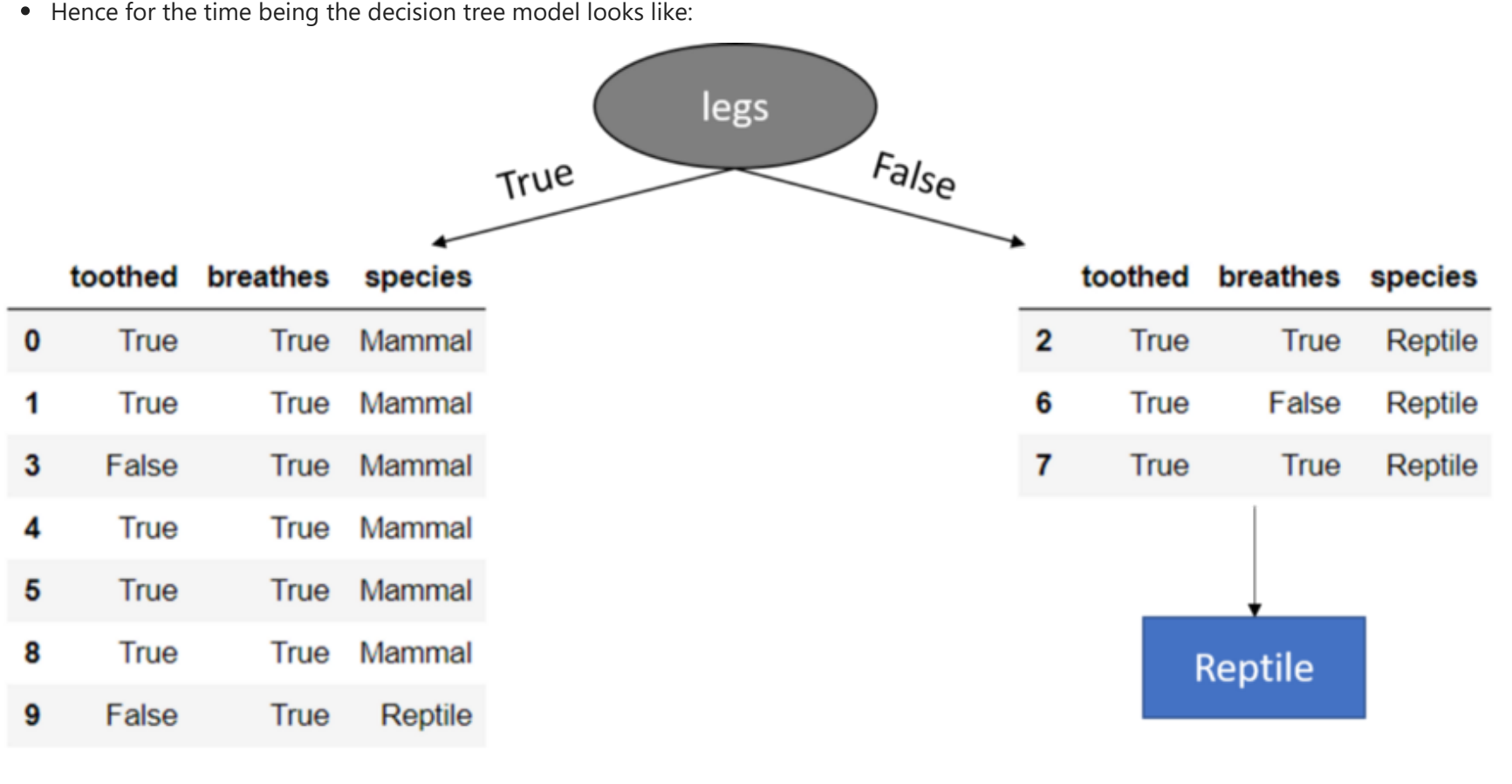
Hence, to come back to our initial question, each leaf node should (in the best case) only contain "Mammals" or "Reptiles". The task for us is now to find the best "way" to split the dataset such that this can be achieved. What do I mean when I say split? Well consider the dataset above and think about what must be done to split the dataset into a Dataset 1 containing as target feature values (species) only Mammals and a Dataset 2, containing only Reptiles. To achieve that, in this simplified example, we only need the descriptive feature hair since if the hair is TRUE, the associated species is always a Mammal. Hence, in this case, our tree model would look like:



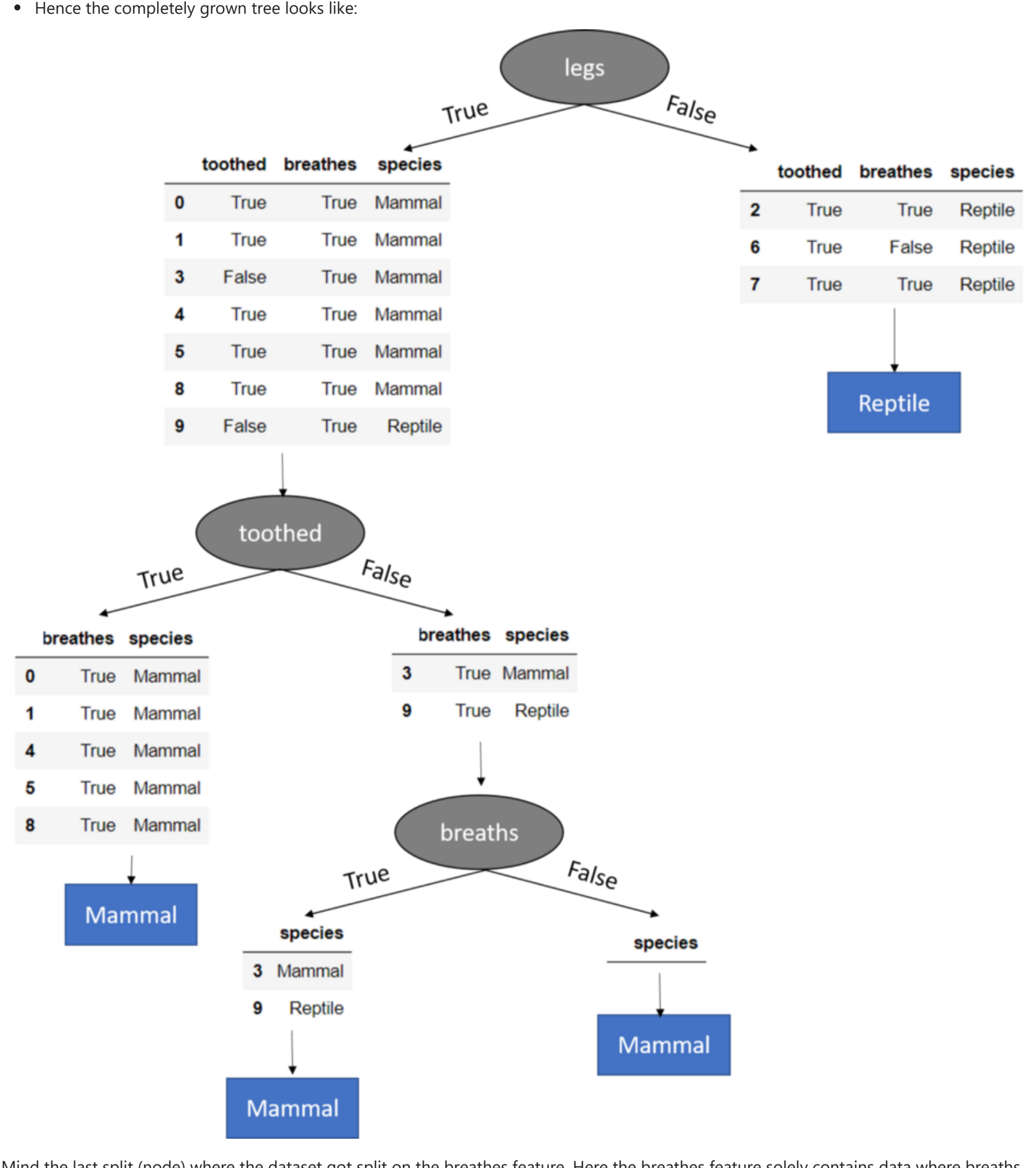
That is, we have split our dataset by asking the question if the animal has hair or not. Now, in that case, the splitting has been very easy for us, because we could easily identify the classification relationship between having hair and being mammal and vice versa. Some may differentiate this table dataset easily because they knew mammal have hair while reptiles don't. But a machine can detect it so easily. Moreover, most of the time datasets are not that easily separable and we must split the dataset more than one time ("ask more than one question"). So leaving hair no other attribute makes it easy to obtain leaf nodes directly. We'll omit the hair attribute to obtain a real-world like situation.



- Hence the entropy of our dataset regarding the target feature is calculated with:
- Entropy of parent node =−(6/10)∗log2(6/10)-(4/10)∗log2(4/10)=0.971
- Entropy of (toothed == True) = -(5/8) log2 (5/8) — (3/8) log2 (3/8) = 0.95 Entropy of (toothed == False) = — (1/2) log2 (1/2) — (1/2) log2 (1/2) = 1 Entropy for split on toothed = 8/10(0.95) + 2/10(1) = 0.96
- Information Gain = 0.971- 0.96 = 0.011
- We'll denote entropy by "H ( )"
- breathes:
- H(breathes) = (9/10∗−((6/9∗log2(6/9))+(3/9∗log2(3/9)))+1/10∗−((0)+(1∗log2(1)))) = 0.82647
- InfoGain(breathes)=0.971−0.82647= 0.1445
- legs:
- H(legs) = 7/10∗−((6/7∗log2(6/7))+(1/7∗log2(1/7)))+3/10∗−((0)+(1∗log2(1))) = 0.41417
- InfoGain(legs)=0.971−0.41417= 0.5568
- Hence the splitting the dataset along the feature legs results in the largest information gain and we should use this feature for our root node.
- Hence for the time being the decision tree model looks like:



- We see that for legs == False, the target feature values of the remaining dataset are all Reptile and hence we set this as leaf node because we have a pure dataset (Further splitting the dataset on any of the remaining two features would not lead to a different or more accurate result since whatever we do after this point, the prediction will remain Reptile). Additionally, you see that the feature legs are no longer included in the remaining datasets. Because we already have used this (categorical) feature to split the dataset on it must not be further used.
- Until now we have found the feature for the root node as well as a leaf node for legs == False. The same steps for information gain calculation must now be accomplished also for the remaining dataset for legs == True since here we still have a mixture of different target feature values. Hence:
- Information gain calculation for the features toothed and breathes for the remaining dataset legs == True:
- The Entropy of the (new) sub data set after the first split:
- H(D)=−((67∗log2(67))+(17∗log2(17)))= 0.5917
- toothed:
- H(toothed)=5/7∗−((1∗log2(1))+(0))+2/7∗−((1/2∗log2(1/2))+(1/2∗log2(1/2)))= 0.285
- InfoGain(toothed)=0.5917−0.285= 0.3067
- breathes:
- H(breathes)= 77∗−((67∗log2(67))+(17∗log2(17)))+0 = 5917
- InfoGain(toothed)= 0.5917−0.5917 = 0
- The dataset for toothed == False still contains a mixture of different target feature values why we proceed partitioning on the last left feature (== breathes)
- Hence the completely grown tree looks like:



Mind the last split (node) where the dataset got split on the breathes feature. Here the breathes feature solely contains data where breaths == True. Hence for breathes == False there are no instances in the dataset and therewith there is no sub-Dataset which can be built. In that case, we return the most frequently occurring target feature value in the original dataset which is Mammal. This is an example of how our tree model generalizes behind the training data.

If we consider the other branch, that is breathes == True we know, that after splitting the Dataset on the values of a specific feature (breathes{True, False}) in our case, the feature must be removed. Well, that leads to a dataset where no more features are available to further split the dataset on. Hence we stop growing the tree and return the mode value of the direct parent node which is "Mammal".

Decision Tree Classifier Building in Scikit-learn

- Importing Required Libraries
- Let's first load the required libraries.

In [22]:
```python
# Load libraries
import pandas as pd
from sklearn.tree import DecisionTreeClassifier # Import Decision Tree Classifier
from sklearn.model_selection import train_test_split # Import train_test_split function
from sklearn import metrics #Import scikit-learn metrics module for accuracy calculation
```

In [24]:
```python
col_names = ['pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bmi', 'pedigree', 'age', 'label']
# load dataset
pima = pd.read_csv("diabetes.csv", header=None, names=col_names)
pima.head()
```

Out[24]:

	pregnant	glucose	bp	skin	insulin	bmi	pedigree	age	label
0	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
1	6	148	72	35	0	33.6	0.627	50	1
2	1	85	66	29	0	26.6	0.351	31	0
3	8	183	64	0	0	23.3	0.672	32	1
4	1	89	66	23	94	28.1	0.167	21	0

In [25]:
```python
#split dataset in features and target variable
feature_cols = ['pregnant', 'insulin', 'bmi', 'age','glucose','bp','pedigree']
X = pima[feature_cols] # Features
y = pima.label # Target variab
```

In [26]:
```python
# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1) # 70% training and 30
```

In [ ]:
```python
# Create Decision Tree classifer object
clf = DecisionTreeClassifier()
# Train Decision Tree Classifer
clf = clf.fit(X_train,y_train)
#Predict the response for test dataset
y_pred = clf.predict(X_test)
```

In [ ]:
```python
# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
Accuracy: 0.6753246753246753
```

Optimizing Decision Tree Performance.

- As you can see above the tree is too vast and not that accurate. This tree also takes time, so for better results, we need to optimize the tree by tuning the parameter. Here I'll be explaining only the important parameter of Decision Tree Algorithm, its tuning will require a whole new blog post. If you take a look at the parameters the DecisionTreeClassifier can take, you might be surprised so, let's look at some of them.
- **criterion:** This parameter determines how the impurity of a split will be measured. The default value is "gini" but you can also use "entropy" as a metric for impurity.
- **splitter:** This is how the decision tree searches the features for a split. The default value is set to "best". That is, for each node, the algorithm considers all the features and chooses the best split. If you decide to set the splitter parameter to "random," then a random subset of features will be considered. The split will then be made by the best feature within the random subset. The size of the random subset is determined by the max_features parameter. This is partly where a Random Forest gets its name.
- **max_depth:** This determines the maximum depth of the tree. In our case, we use a depth of two to make our decision tree. The default value is set to none. This will often result in over-fitted decision trees. The depth parameter is one of the ways in which we can regularize the tree, or limit the way it grows to prevent over-fitting.
- **min_samples_split:** The minimum number of samples a node must contain to consider splitting. The default value is two. You can use this parameter to regularize your tree.
- **min_samples_leaf:** The minimum number of samples needed to be considered a leaf node. The default value is set to one. Use this parameter to limit the growth of the tree.
- **max_features:**The number of features to consider when looking for the best split. If this value is not set, the decision tree will consider all features available to make the best split. Depending on your application, it's often a good idea to tune this parameter.

- For syntax purposes, lets set some of these parameters:

- Setting parameterstree = DecisionTreeClassifier(criterion = "entropy", splitter = "random", max_depth = 2, min_samples_split = 5,min_samples_leaf = 2, max_features = 2).fit(X,y)

- Advanced: Overfitting and Underfitting

- In practice, it's not uncommon for a tree to have 10 splits between the top-level (all data) and a leaf. As the tree gets deeper, the dataset gets sliced up into leaves with fewer data. If a tree only had 1 split, it divides the data into 2 groups. If each group is split again, we would get 4 groups of data. Splitting each of those again would create 8 groups. If we keep doubling the number of groups by adding more splits at each level, we'll have 210210 groups of data by the time we get to the 10th level. That's 1024 leaves.

- When we divide the data amongst many leaves, we also have fewer data in each leaf. Leaves with very few data will make predictions that are quite close to those homes' actual values, but they may make very unreliable predictions for new data (because each prediction is based on only a few data).

- This is a phenomenon called overfitting, where a model matches the training data almost perfectly but does poorly in validation and other new data. On the flip side, if we make our tree very shallow, it doesn't divide up the data into very distinct groups.

- At an extreme, if a tree divides data into only 2 or 4, each group still has a wide variety of data. Resulting predictions may be far off for most data, even in the training data (and it will be bad in validation too for the same reason). When a model fails to capture important distinctions and patterns in the data, so it performs poorly even in training data, that is called underfitting.

Thanks !!