Decision Trees

A decision tree is a flowchart-like structure in which each internal node represents a test on a feature (e.g. whether a coin flip comes up heads or tails) , each leaf node represents a class label (decision taken after computing all features) and branches represent conjunctions of features that lead to those class labels. The paths from root to leaf represent classification rules.

Decision trees are constructed via an algorithmic approach that identifies ways to split a data set based on different conditions. It is one of the most widely used and practical methods for supervised learning. Decision Trees are a non-parametric **supervised learning** method used for both **classification** and **regression** tasks.

Terminologies related to Decision Tree

- **Root node:** the topmost tree node which divides into two homogeneous sets.
- **Decision node:** a sub-node which further splits into other two sub-nodes.
- **Terminal/Leaf node:** the lowermost nodes or the nodes with no children that represents a class label (decision taken after computing all attributes)
- **Splitting**: dividing a node into two or more nodes. The splitting technique results in fully grown trees until the criteria of a class attribute are met. But a fully-grown tree is likely to over-fit the data which leads to poor accuracy on unseen observations. This is when Pruning comes into the picture.
- **Pruning**: Process of reducing the size of the tree by removing the nodes which play a minimal role in classifying an instance without reducing the predictive accuracy as measured by a cross-validation set.
- **Branch:** a sub-section of a decision tree is called a branch

Assumptions

- In the beginning, the whole dataset is considered the root.
- Best attribute is selected as the root node using some statistical approach.
- Records are split recursively to produce homogeneous sub-nodes.
- If the attributes are continuous, they are discretized before building the model.

Pseudo-code for Decision Tree Algorithm

1. Select the most powerful attribute as the root node.
2. Split the training set into sub-nodes such that each sub-node has identical attribute values.
3. Repeat Step 1 and 2 until you meet the criteria of the class attribute.
4. Perform pruning or remove unwanted nodes if you have a fully grown tree such that it doesn’t affect the prediction accuracy.

The primary differences and similarities between Classification and Regression Trees are:

1. Regression trees are used when dependent variable is continuous. Classification Trees are used when dependent variable is categorical.
2. In case of Regression Tree, the value obtained by terminal nodes in the training data is the mean response of observation falling in that region. Thus, if an unseen data observation falls in that region, we'll make its prediction with mean value.
3. In case of Classification Tree, the value (class) obtained by terminal node in the training data is the mode of observations falling in that region. Thus, if an unseen data observation falls in that region, we'll make its prediction with mode value.
4. Both the trees divide the predictor space (independent variables) into distinct and non-overlapping regions.
5. Both the trees follow a top-down greedy approach known as ***recursive binary splitting. We call it as 'top-down' because it begins from the top of tree when all the observations are available in a single region and successively splits the predictor space into two new branches down the tree. It is known as 'greedy' because, the algorithm cares (looks for best variable available) about only the current split, and not about future splits which will lead to a better tree.***
6. This splitting process is continued until a user defined stopping criteria is reached. For e.g.: we can tell the algorithm to stop once the number of observations per node becomes less than 50.
7. In both the cases, the splitting process results in fully grown trees until the stopping criteria is reached. But, the fully grown tree is likely to over fit data, leading to poor accuracy on unseen data. This bring 'pruning'. Pruning is one of the technique used tackle overfitting.

https://medium.com/greyatom/decision-trees-a-simple-way-to-visualize-a-decision-dc506a403aeb

Metrics

- Gini impurity
- Information gain
- Variance reduction

Algorithm

Impurity - Entropy & Gini

There are three commonly used impurity measures used in binary decision trees: **Entropy**, **Gini index**, and **Classification Error**.
Entropy (a way to measure impurity):

$$Entropy = -\sum_j p_j \log_2 p_j$$

Gini index (a criterion to minimize the probability of misclassification):

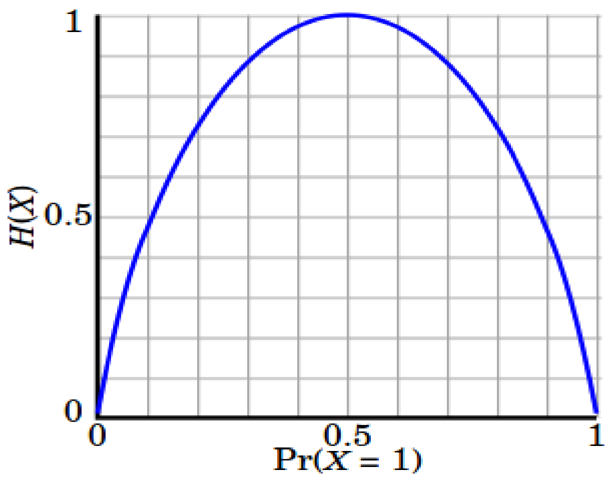
$$Gini = 1 - \sum_j p_j^2$$

Classification Error:

$$ClassificationError = 1 - \max p_j$$

where pjpj is the probability of class jj.

The entropy is 0 if all samples of a node belong to the same class, and the entropy is maximal if we have a uniform class distribution. In other words, the entropy of a node (consist of single class) is zero because the probability is 1 and log (1) = 0. Entropy reaches maximum value when all classes in the node have equal probability.



1. Entropy of a group in which all examples belong to the same class:

$$entropy = -1 \log_2 1 = 0$$

This is not a good set for training.

2. entropy of a group with 50% in either class:

$$entropy = -0.5\log_2 0.5 - 0.5\log_2 0.5 = 1$$

This is a good set for training.
So, basically, the entropy attempts to maximize the mutual information (by constructing a equal probability node) in the decision tree.

Similar to entropy, the **Gini index** is maximal if the classes are perfectly mixed, for example, in a binary class:

$$Gini = 1 - (p_1^2 + p_2^2) = 1 - (0.5^2 + 0.5^2) = 0.5$$

Information Gain (IG)

Using a decision algorithm, we start at the tree root and split the data on the feature that results in the largest **information gain (IG)**.

We repeat this splitting procedure at each child node down to the empty leaves. This means that the samples at each node all belong to the same class.

However, this can result in a very **deep tree** with many nodes, which can easily lead to overfitting. Thus, we typically want to **prune** the tree by setting a limit for the **maximum depth** of the tree.

Basically, using IG, we want to determine **which attribute** in a given set of training feature vectors is **most useful**. In other words, IG tells us how important a given attribute of the feature vectors is.

We will use it to decide the **ordering of attributes** in the nodes of a **decision tree**.

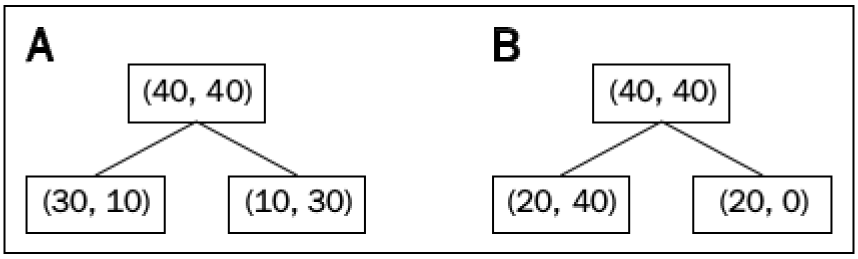
The **Information Gain (IG)** can be defined as follows:

$$IG(D_p) = I(D_p) - \frac{N_{left}}{N_p} I(D_{left}) - \frac{N_{right}}{N_p} I(D_{right})$$

where II could be **entropy**, **Gini index**, or **classification error**, Dp, Dleft, and Dright are the dataset of the parent, left and right child node.
FOR further info refer imp link at last

Information Gain (IG) - Examples

In this section, we'll get IG for a specific case as shown below:



First, **IG** with **Classification Error** (IG_E):

$$ClassificationError = 1 - \max p_j$$

$$\boxed{I_E(D_p) = 1 - \frac{40}{80} = 1 - 0.5 = 0.5}$$

$$A : I_E(D_{left}) = 1 - \frac{30}{40} = 1 - \frac{3}{4} = 0.25$$

$$A : I_E(D_{right}) = 1 - \frac{30}{40} = 1 - \frac{3}{4} = 0.25$$

$$IG(D_p) = I(D_p) - \frac{N_{left}}{N_p} I(D_{left}) - \frac{N_{right}}{N_p} I(D_{right})$$

$$A : IG_E = 0.5 - \frac{40}{80} \times 0.25 - \frac{40}{80} \times 0.25 = 0.5 - 0.125 - 0.125 = 0.25$$

$$B : I_E(D_{left}) = 1 - \frac{40}{60} = 1 - \frac{2}{3} = \frac{1}{3}$$

$$B : I_E(D_{right}) = 1 - \frac{20}{20} = 1 - 1 = 0$$

$$B : IG_E = 0.5 - \frac{60}{80} \times \frac{1}{3} - \frac{20}{80} \times 0 = 0.5 - 0.25 - 0 = 0.25$$

The information gains using the classification error as a splitting criterion are the same (0.25) in both cases A and B.

IG with **Gini index** (IG_G):

$$Gini = 1 - \sum_j p_j^2$$

$$I_G(D_p) = 1 - \left(\left(\frac{40}{80}\right)^2 + \left(\frac{40}{80}\right)^2\right) = 1 - (0.5^2 + 0.5^2) = 0.5$$

$$A: I_G(D_{left}) = 1 - \left(\left(\frac{30}{40}\right)^2 + \left(\frac{10}{40}\right)^2\right) = 1 - \left(\frac{9}{16} + \frac{1}{16}\right) = \frac{3}{8} = 0.375$$

$$A: I_G(D_{right}) = 1 - \left(\left(\frac{10}{40}\right)^2 + \left(\frac{30}{40}\right)^2\right) = 1 - \left(\frac{1}{16} + \frac{9}{16}\right) = \frac{3}{8} = 0.375$$

$$A: I_G = 0.5 - \frac{40}{80} \times 0.375 - \frac{40}{80} \times 0.375 = 0.125$$

$$B: I_G(D_{left}) = 1 - \left(\left(\frac{20}{60}\right)^2 + \left(\frac{40}{60}\right)^2\right) = 1 - \left(\frac{9}{16} + \frac{1}{16}\right) = 1 - \frac{5}{9} = 0.44$$

$$B: I_G(D_{right}) = 1 - \left(\left(\frac{20}{20}\right)^2 + \left(\frac{0}{20}\right)^2\right) = 1 - (1 + 0) = 1 - 1 = 0$$

$$B: I_G = 0.5 - \frac{60}{80} \times 0.44 - 0 = 0.5 - 0.33 = 0.17$$

So, the **Gini index** favors the split **B**.

IG with **Entropy** (IG_H):

$$Entropy = -\sum_j p_j \log_2 p_j$$

$$I_H(D_p) = -(0.5\log_2(0.5) + 0.5\log_2(0.5)) = 1$$

$$A: I_H(D_{left}) = -\left(\frac{30}{40}\log_2\left(\frac{30}{40}\right) + \frac{10}{40}\log_2\left(\frac{10}{40}\right)\right) = 0.81$$

$$A: I_H(D_{right}) = -\left(\frac{10}{40}\log_2\left(\frac{10}{40}\right) + \frac{30}{40}\log_2\left(\frac{30}{40}\right)\right) = 0.81$$

$$A: IG_H = 1 - \frac{40}{80} \times 0.81 - \frac{40}{80} \times 0.81 = 0.19$$

$$B: I_H(D_{left}) = -\left(\frac{20}{60}\log_2\left(\frac{20}{60}\right) + \frac{40}{60}\log_2\left(\frac{40}{60}\right)\right) = 0.92$$

$$B: I_H(D_{right}) = -\left(\frac{20}{20}\log_2\left(\frac{20}{20}\right) + 0\right) = 0$$

$$B: IG_H = 1 - \frac{60}{80} \times 0.92 - \frac{20}{80} \times 0 = 0.31$$

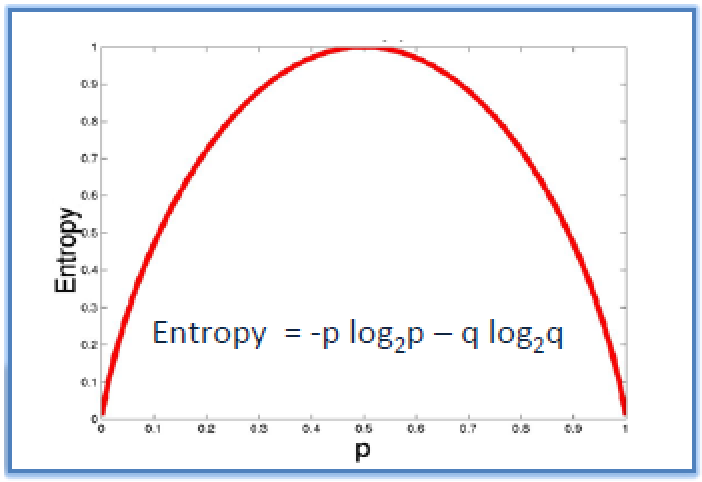
So, the entropy criterion favors **B**.

Algorithm – explained using entropy

The core algorithm for building decision trees called **ID3** by J. R. Quinlan which employs a top-down, greedy search through the space of possible branches with no backtracking. **ID3 uses *Entropy* and *Information Gain* to construct a decision tree.**

Entropy

A decision tree is built top-down from a root node and involves partitioning the data into subsets that contain instances with similar values (homogenous). ID3 algorithm uses entropy to calculate the homogeneity of a sample. If the sample is completely homogeneous the entropy is zero and if the sample is an equally divided it has entropy of one.

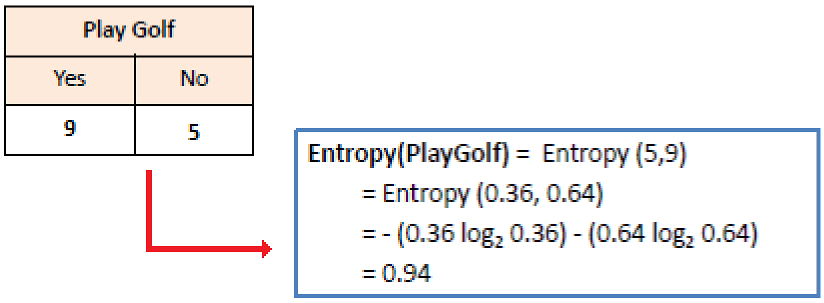


$$\text{Entropy} = -0.5 \log_2 0.5 - 0.5 \log_2 0.5 = 1$$

To build a decision tree, we need to calculate two types of entropy using frequency tables as follows:

a) Entropy using the frequency table of one attribute:

$$E(S) = \sum_{i=1}^{c} - p_i \log_2 p_i$$



Play Golf	
Yes	No
9	5

Entropy(PlayGolf) = Entropy (5,9)

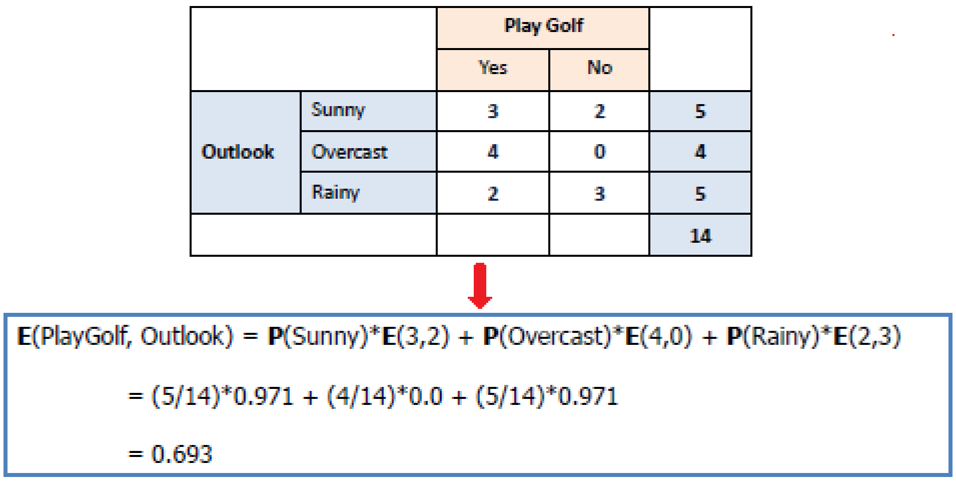
= Entropy (0.36, 0.64)

= - (0.36 $\log_2$ 0.36) - (0.64 $\log_2$ 0.64)

= 0.94

b) Entropy using the frequency table of two attributes:

$$E(T, X) = \sum_{c \in X} P(c)E(c)$$



		Play Golf		
		Yes	No	
Outlook	Sunny	3	2	5
	Overcast	4	0	4
	Rainy	2	3	5
				14

E(PlayGolf, Outlook) = **P**(Sunny)***E**(3,2) + **P**(Overcast)***E**(4,0) + **P**(Rainy)***E**(2,3)

= (5/14)*0.971 + (4/14)*0.0 + (5/14)*0.971

= 0.693

Information Gain

The information gain is based on the decrease in entropy after a dataset is split on an attribute. Constructing a decision tree is all about finding attribute that returns the highest information gain (i.e., the most homogeneous branches).

Step 1: Calculate entropy of the target.

Entropy(PlayGolf) = Entropy (5,9)

= Entropy (0.36, 0.64)

= - (0.36 $\log_2$ 0.36) - (0.64 $\log_2$ 0.64)

= 0.94

Step 2: The dataset is then split on the different attributes. The entropy for each branch is calculated. Then it is added proportionally, to get total entropy for the split. The resulting entropy is subtracted from the entropy before the split. The result is the Information Gain, or decrease in entropy.

		Play Golf	
		Yes	No
Outlook	Sunny	3	2
	Overcast	4	0
	Rainy	2	3
Gain = 0.247			

		Play Golf	
		Yes	No
Temp.	Hot	2	2
	Mild	4	2
	Cool	3	1
Gain = 0.029			

		Play Golf	
		Yes	No
Humidity	High	3	4
	Normal	6	1
Gain = 0.152			

		Play Golf	
		Yes	No
Windy	False	6	2
	True	3	3
Gain = 0.048			

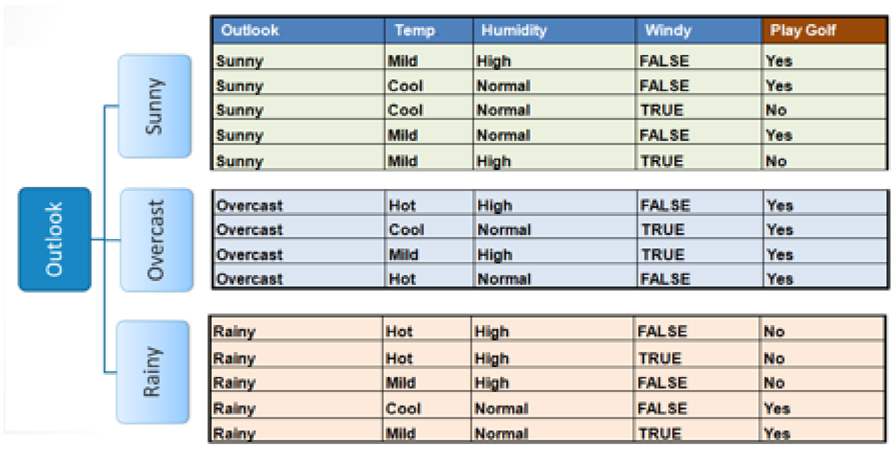
$$Gain(T,X) = Entropy(T) - Entropy(T,X)$$

G(PlayGolf, Outlook) = **E**(PlayGolf) – **E**(PlayGolf, Outlook)

= 0.940 – 0.693 = 0.247

Step 3: Choose attribute with the largest information gain as the decision node, divide the dataset by its branches and repeat the same process on every branch.

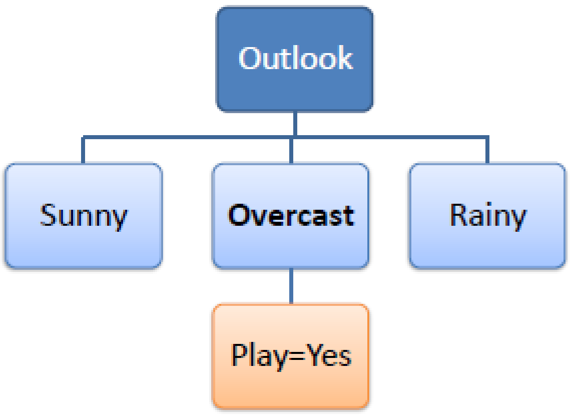
★		Play Golf	
		Yes	No
Outlook	Sunny	3	2
	Overcast	4	0
	Rainy	2	3
Gain = 0.247			



Outlook	Temp	Humidity	Windy	Play Golf
Sunny	Mild	High	FALSE	Yes
Sunny	Cool	Normal	FALSE	Yes
Sunny	Cool	Normal	TRUE	No
Sunny	Mild	Normal	FALSE	Yes
Sunny	Mild	High	TRUE	No
Overcast	Hot	High	FALSE	Yes
Overcast	Cool	Normal	TRUE	Yes
Overcast	Mild	High	TRUE	Yes
Overcast	Hot	Normal	FALSE	Yes
Rainy	Hot	High	FALSE	No
Rainy	Hot	High	TRUE	No
Rainy	Mild	High	FALSE	No
Rainy	Cool	Normal	FALSE	Yes
Rainy	Mild	Normal	TRUE	Yes

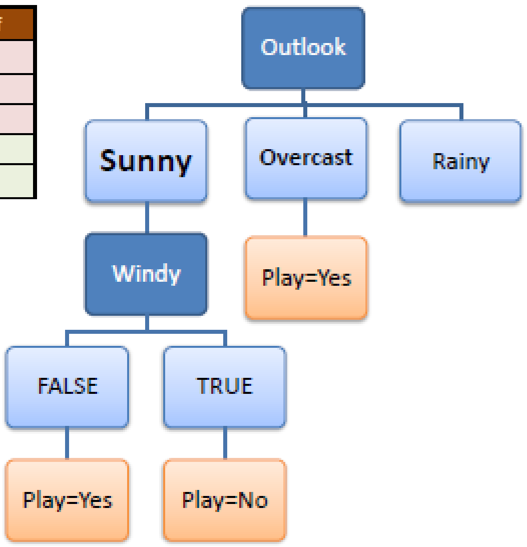
Step 4a: A branch with entropy of 0 is a leaf node.

Temp	Humidity	Windy	Play Golf
Hot	High	FALSE	Yes
Cool	Normal	TRUE	Yes
Mild	High	TRUE	Yes
Hot	Normal	FALSE	Yes



Step 4b: A branch with entropy more than 0 needs further splitting.

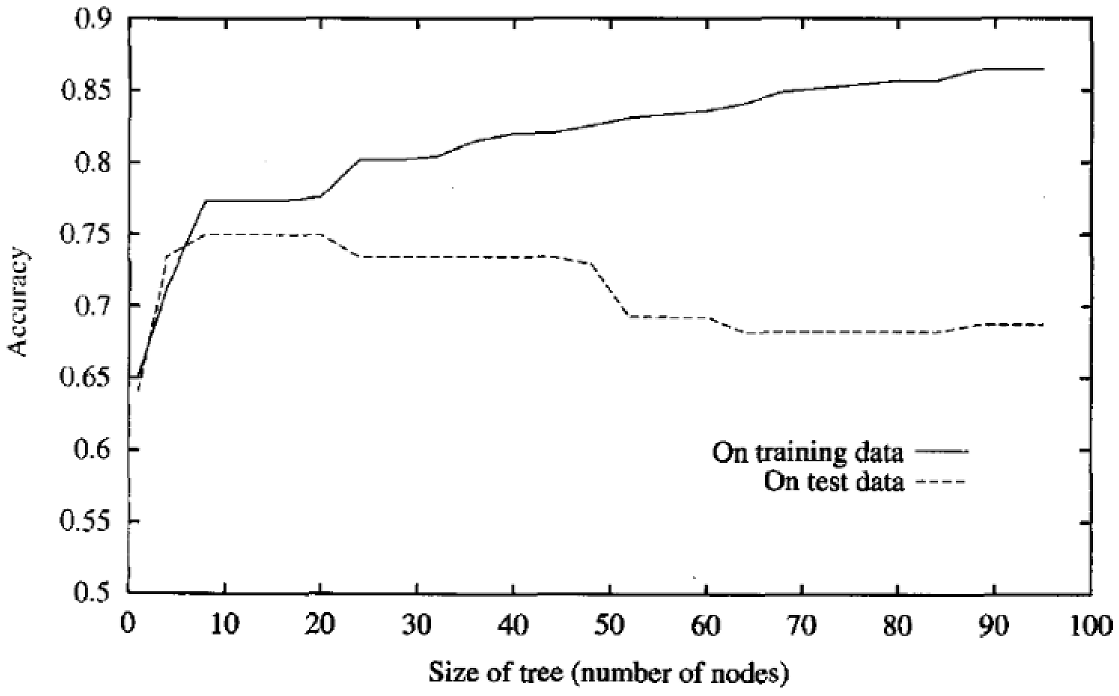
Temp	Humidity	Windy	Play Golf
Mild	High	FALSE	Yes
Cool	Normal	FALSE	Yes
Mild	Normal	FALSE	Yes
Cool	Normal	TRUE	No
Mild	High	TRUE	No



Step 5: The ID3 algorithm is run recursively on the non-leaf branches, until all data is classified.

How to avoid overfitting when using ID3?

The ID3 algorithm grows each branch of the tree just deeply enough to perfectly classify the training examples. While this is sometimes a reasonable strategy, in fact it can lead to difficulties when there is noise in the data,or when the number of training examples is too small to produce a representative sample of the true target function. In either of these cases, this simple algorithm can produce trees that overfit the training examples.



Overfitting in decision tree learning. As ID3 adds new nodes to grow the decision tree, the accuracy of the tree measured over the training examples increases monotonically. However, when measured over a set of test examples independent of the training examples, accuracy first increases, then decreases.

One way this can occur is when the training examples contain random errors or noise. Moreover, overfitting is possible even when the training data are noise-free, especially when small numbers of examples are associated with leaf nodes. In this case, it is quite possible for coincidental regularities to occur, in which some attribute happens to partition the examples very well, despite being unrelated to the actual target function.

There are several approaches to avoiding overfitting in decision tree learning which can be grouped into two classes:

- approaches that stop growing the tree earlier, before it reaches the point where it perfectly classifies the training data(i.e set **maximum depth**-length of the longest path from a root to a leaf)
- approaches that allow the tree to overfit the data, and then post-prune the tree.

Although the first of these approaches might seem more direct, the second approach of post-pruning overfit trees has been found to be more successful in practice. This is due to the difficulty in the first approach of estimating precisely when to stop growing the tree.

Pruning** involves **removing the branches that make use of features having low importance. This way, we reduce the complexity of tree, and thus increasing its predictive power by reducing overfitting.

Pruning can start at either root or the leaves. The simplest method of pruning starts at leaves and removes each node with the most popular class in that leaf, this change is kept if it doesn't deteriorate accuracy. Its also called **reduced error pruning**. More sophisticated pruning methods can be used such as **cost complexity pruning** where a learning parameter (alpha) is used to weigh whether nodes can be removed based on the size of the sub-tree.

Optimizing Decision Tree Performance

- **criterion : optional (default="gini") or Choose attribute selection measure**: This parameter allows us to use the different-different attribute selection measure. Supported criteria are "gini" for the Gini index and "entropy" for the information gain.

- **splitter : string, optional (default="best") or Split Strategy**: This parameter allows us to choose the split strategy. Supported strategies are "best" to choose the best split and "random" to choose the best random split.

- **max_depth : int or None, optional (default=None) or Maximum Depth of a Tree**: The maximum depth of the tree. If None, then nodes are expanded until all the leaves contain less than min_samples_split samples. The higher value of maximum depth causes overfitting, and a lower value causes underfitting (Source).

In Scikit-learn, optimization of decision tree classifier performed by only pre-pruning. Maximum depth of the tree can be used as a control variable for pre-pruning. In the following the example, you can plot a decision tree on the same data with max_depth=3. Other than pre-pruning parameters, You can also try other attribute selection measure such as entropy.

Scikit-Learn Decision Tree Parameters

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**. In **Figure-4**, you can see what happens if you don't set the depth of the tree—pure madness!

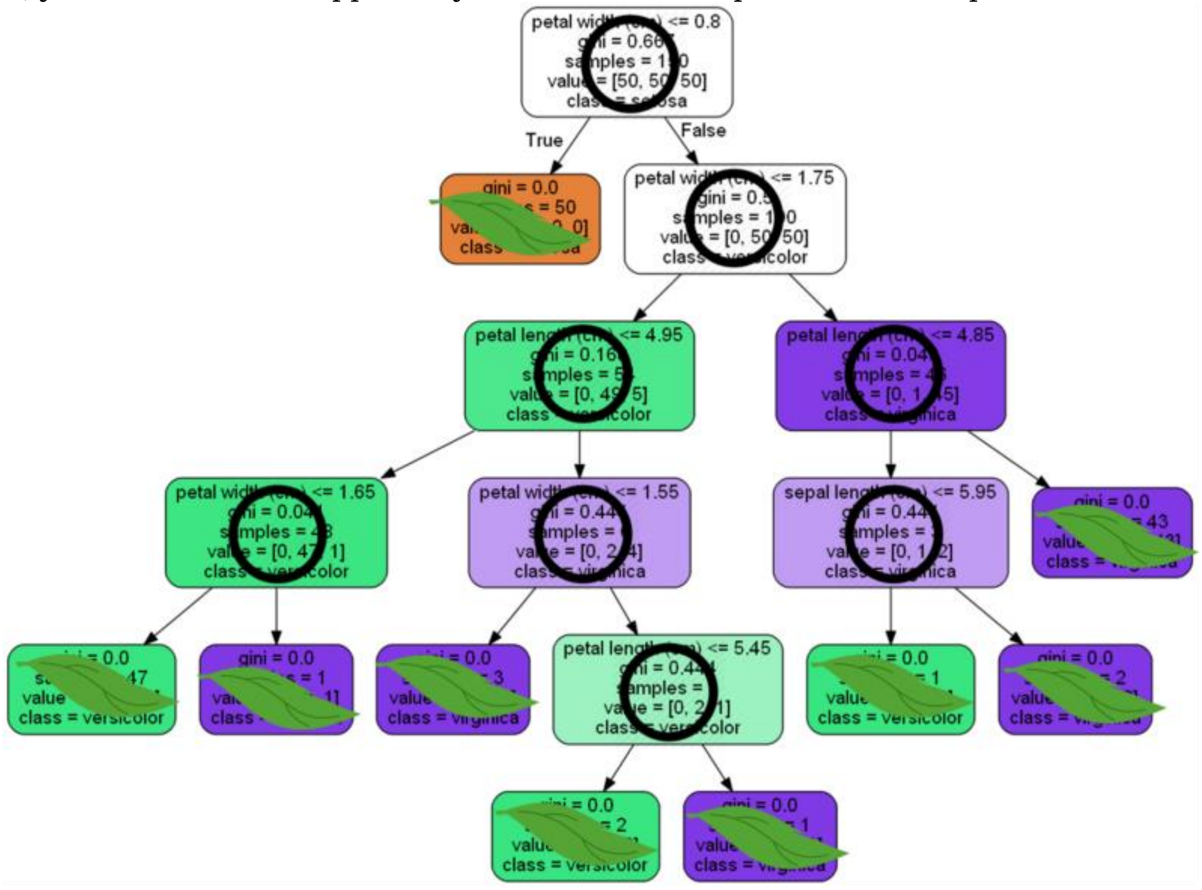


Figure-4) A fully grown Decision Tree: In the tree shown above, none of the parameters were set. The tree grows to a fully to a depth of five. There are eight nodes and nine leaves. Not limiting the growth of a decision tree may lead to over-fitting.

min_samples_split: The minimum number of samples a node must contain in order 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. Here is an article that recommends how to set max_features.

Advantages of Decision Tree:

1. **Easy to Understand**: Decision tree output is very easy to understand even for people from non-analytical background. It does not require any statistical knowledge to read and interpret them. Its graphical representation is very intuitive and users can easily relate their hypothesis.

2. **Useful in Data exploration:** Decision tree is one of the fastest way to identify most significant variables and relation between two or more variables. With the help of decision trees, we can create new variables / features that has better power to predict target variable. It can also be used in data exploration stage. For e.g., we are working on a problem where we have information available in hundreds of variables, there decision tree will help to identify most significant variable.

3. Decision trees implicitly perform variable screening or feature selection.

4. Decision trees require relatively **little effort from users for data preparation**.

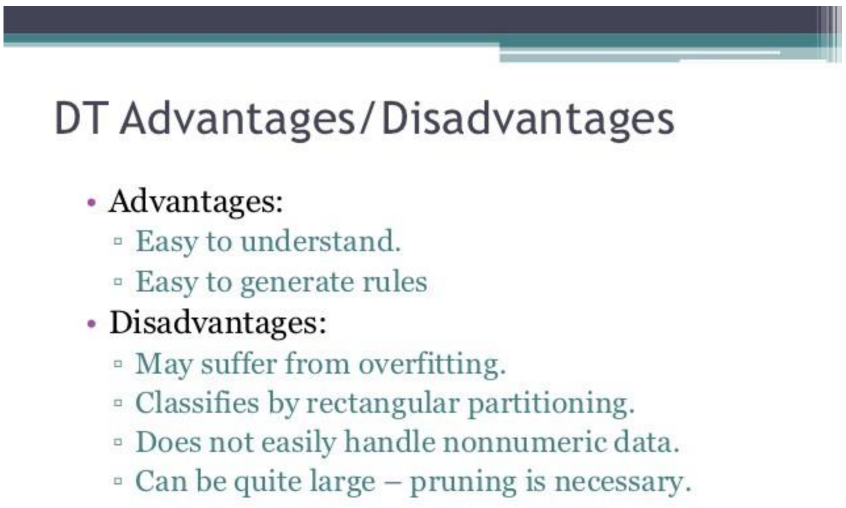
5. **Less data cleaning required:** It requires less data cleaning compared to some other modeling techniques. It is not influenced by outliers and missing values to a fair degree.

6. **Data type is not a constraint:** It can handle both numerical and categorical variables. Can also *handle multi-output problems.*

7. **Non-Parametric Method:** Decision tree is considered to be a non-parametric method. This means that decision trees have no assumptions about the space distribution and the classifier structure.

8. Non-linear relationships between parameters do not affect tree performance.

9. The number of hyper-parameters to be tuned is almost null.



DT Advantages/Disadvantages

- Advantages:
  - Easy to understand.
  - Easy to generate rules
- Disadvantages:
  - May suffer from overfitting.
  - Classifies by rectangular partitioning.
  - Does not easily handle nonnumeric data.
  - Can be quite large – pruning is necessary.

Disadvantages of Decision Tree:

1. **Over fitting:** Decision-tree learners can create over-complex trees that do not generalize the data well. This is called overfitting. Over fitting is one of the most practical difficulty for decision tree models. This problem gets solved by setting constraints on model parameters and pruning.

2. **Not fit for continuous variables**: While working with continuous numerical variables, decision tree loses information, when it categorizes variables in different categories.

3. Decision trees can be unstable because small variations in the data might result in a completely different tree being generated. This is called **variance**, which needs to be lowered by methods like **bagging** and **boosting**.

4. *Greedy* algorithms cannot guarantee to return the globally optimal decision tree. This can be mitigated by training multiple trees, where the features and samples are randomly sampled with replacement.

5. Decision tree learners create *biased* trees if some classes dominate. It is therefore recommended to balance the data set prior to fitting with the decision tree.

6. Information gain in a decision tree with categorical variables gives a biased response for attributes with greater no. of categories.

7. Generally, it gives low prediction accuracy for a dataset as compared to other machine learning algorithms.

8. Calculations can become complex when there are many class label.

Tips on practical use

- Decision trees tend to overfit on data with a large number of features. Getting the right ratio of samples to number of features is important, since a tree with few samples in high dimensional space is very likely to overfit.
- Consider performing dimensionality reduction (PCA, ICA, or Feature selection) beforehand to give your tree a better chance of finding features that are discriminative.
- Understanding the decision tree structure will help in gaining more insights about how the decision tree makes predictions, which is important for understanding the important features in the data.
- Visualise your tree as you are training by using the export function. Use max_depth=3 as an initial tree depth to get a feel for how the tree is fitting to your data, and then increase the depth.
- Remember that the number of samples required to populate the tree doubles for each additional level the tree grows to. Use max_depth to control the size of the tree to prevent overfitting.
- Use min_samples_split or min_samples_leaf to ensure that multiple samples inform every decision in the tree, by controlling which splits will be considered. A very small number will usually mean the tree will overfit, whereas a large number will prevent the tree from learning the data. Try min_samples_leaf=5 as an initial value. If the sample size varies greatly, a float number can be used as percentage in these two parameters. While min_samples_split can create arbitrarily small leaves, min_samples_leaf guarantees that each leaf has a minimum size, avoiding low-variance, over-fit leaf nodes in regression problems. For classification with few classes, min_samples_leaf=1 is often the best choice.
- Balance your dataset before training to prevent the tree from being biased toward the classes that are dominant. Class balancing can be done by sampling an equal number of samples from each class, or preferably by normalizing the sum of the sample weights (sample_weight) for each class to the same value. Also note that weight-based pre-pruning criteria, such as min_weight_fraction_leaf, will then be less biased toward dominant classes than criteria that are not aware of the sample weights, like min_samples_leaf.
- If the samples are weighted, it will be easier to optimize the tree structure using weight-based pre-pruning criterion such as min_weight_fraction_leaf, which ensure that leaf nodes contain at least a fraction of the overall sum of the sample weights.
- All decision trees use np.float32 arrays internally. If training data is not in this format, a copy of the dataset will be made.
- If the input matrix X is very sparse, it is recommended to convert to sparse csc_matrix before calling fit and sparse csr_matrix before calling predict. Training time can be orders of magnitude faster for a sparse matrix input compared to a dense matrix when features have zero values in most of the samples.

Tree algorithms: ID3, C4.5, C5.0 and CART

What are all the various decision tree algorithms and how do they differ from each other? Which one is implemented in scikit-learn?

ID3 (Iterative Dichotomiser 3) was developed in 1986 by Ross Quinlan. The algorithm creates a multiway tree, finding for each node (i.e. in a greedy manner) the categorical feature that will yield the largest information gain for categorical targets. Trees are grown to their maximum size and then a pruning step is usually applied to improve the ability of the tree to generalise to unseen data.

C4.5 is the successor to ID3 and removed the restriction that features must be categorical by dynamically defining a discrete attribute (based on numerical variables) that partitions the continuous attribute value into a discrete set of intervals. C4.5 converts the trained trees (i.e. the output of the ID3 algorithm) into sets of if-then rules. These accuracy of each rule is then evaluated to determine the order in which they should be applied. Pruning is done by removing a rule's precondition if the accuracy of the rule improves without it.

C5.0 is Quinlan's latest version release under a proprietary license. It uses less memory and builds smaller rulesets than C4.5 while being more accurate.

CART (Classification and Regression Trees) is very similar to C4.5, but it differs in that it supports numerical target variables (regression) and does not compute rule sets. CART constructs binary trees using the feature and threshold that yield the largest information gain at each node.

scikit-learn uses an optimised version of the CART algorithm; however, scikit-learn implementation does not support categorical variables for now.

Take-home message

1. Overfitting the training data is an important issue in decision tree learning. Methods for post-pruning the decision tree are therefore important to avoid overfitting in decision tree learning.
2. A large variety of extensions to the basic ID3 algorithm has been developed by different researchers. These include methods for post-pruning trees, handling real-valued attributes etc.
3. An improvement over decision tree learning is made using techniques of **boosting and bagging**

Links

https://scikit-learn.org/stable/modules/tree.html

https://medium.com/@rishabhjain_22692/decision-trees-it-begins-here-93ff54ef134

https://medium.com/@mohtedibf/indepth-parameter-tuning-for-decision-tree-6753118a03c3

https://medium.com/datadriveninvestor/decision-tree-adventures-2-explanation-of-decision-tree-classifier-parameters-84776f39a28 https://medium.com/coinmonks/what-is-entropy-and-why-information-gain-is-matter-4e85d46d2f01

https://towardsdatascience.com/entropy-how-decision-trees-make-decisions-2946b9c18c8

http://seaborn.pydata.org/generated/seaborn.pairplot.html

https://en.wikipedia.org/wiki/Decision_tree_learning

Method parameters –

https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier

https://www.datacamp.com/community/tutorials/decision-tree-classification-python

IMP LINK

https://www.bogotobogo.com/python/scikit-learn/scikt_machine_learning_Decision_Tree_Learning_Informatioin_Gain_IG_Impurity_Entropy_Gini_Classification_Error.php