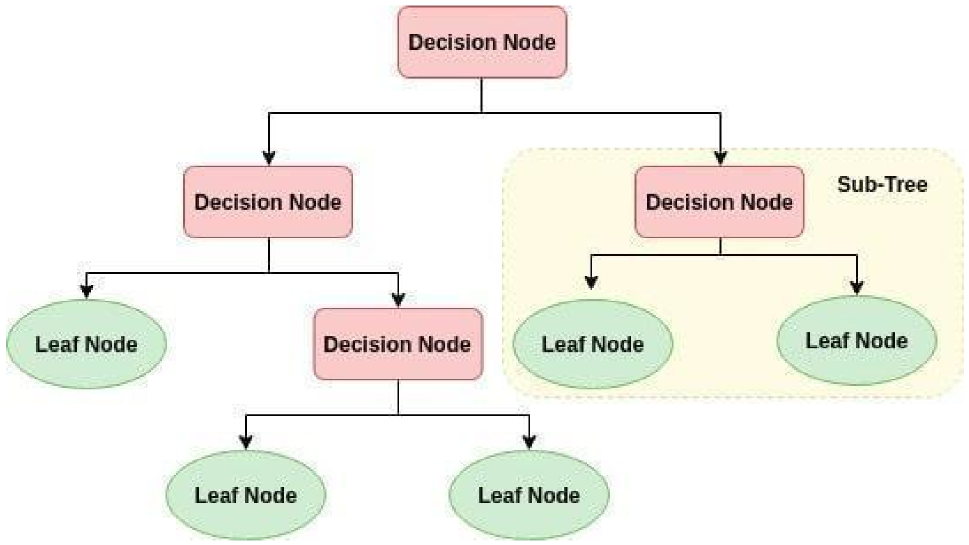
Decision Tree

Decision tree is a type of supervised learning algorithm that can be used for both regression and classification.

A Decision Tree is a supervised predictive model that can learn to predict answering a set of simple questions. It answers sequential questions which send us down a certain route of the tree. The decision rules are generally in form of if-then-else statements. The deeper the tree, the more complex the rules and fitter the model.



How decision tree works?

Decision Trees use different attributes to repeatedly split the data into subsets and repeats the process until the subsets are pure. Pure means that they all share the same target value.

How is split made or how does the tree decide at which variable to split?

The input variables can be numeric or categorical or a mix of numeric and categorical regardless of whether target variable is numeric i.e. a regression problem or target variable is categorical i.e. a classification problem.

In case of a categorical input variable, we have to find out a variable to split the dataset. To calculate this variable, we need to split the dataset using different variables and choose the variable for which the below mentioned metric are best.

In case of a numerical input variable, we have to find out a variable and a value combination to split the dataset. To calculate this variable and value combination, we need to split the dataset using different variables and different values of those different variables and choose the variable and value combination for which the below mentioned metric are best.

Regression

- Reduction in Standard Deviation

Classification

- Information Gain

- Gini Impurity

Reduction in standard deviation

Reduction in standard deviation is a method for splitting the node used when the target variable is continuous. We use standard deviation to calculate the homogeneity of a numerical sample. If the numerical sample is completely homogeneous its standard deviation is zero.

Calculation of standard deviation and coefficient of variance for one attribute and for two attributes respectively

Hours Played
25
30
46
45
52
23
43
35
38
46
48
52
44
30

$$\textbf{Count} = n = 14$$

$$\textbf{Average} = \bar{x} = \frac{\sum x}{n} = 39.8$$

$$\textbf{Standard Deviation} = S = \sqrt{\frac{\sum(x-\bar{x})^2}{n}} = 9.32$$

$$\textbf{Coeffeicient of Variation} = CV = \frac{S}{\bar{x}} * 100\% = 23\%$$

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

		Hours Played (StDev)	Count
Outlook	Overcast	3.49	4
	Rainy	7.78	5
	Sunny	10.87	5
			14

S(Hours, Outlook) = **P**(Sunny)***S**(Sunny) + **P**(Overcast)***S**(Overcast) + **P**(Rainy)***S**(Rainy)

= (4/14)*3.49 + (5/14)*7.78 + (5/14)*10.87

= 7.66

How it works

Step 1: Calculate standard deviation of target

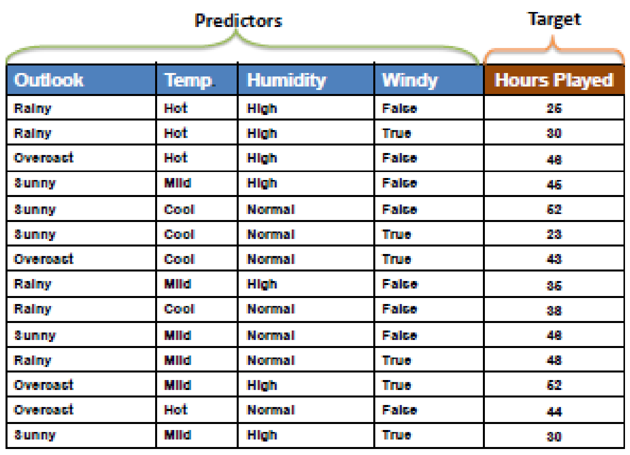
Step 2: Calculate the standard deviation of each split as the weighted average standard deviation of child nodes

Step 3: Chose the branch to split whose giving the maximum reduction of standard deviation

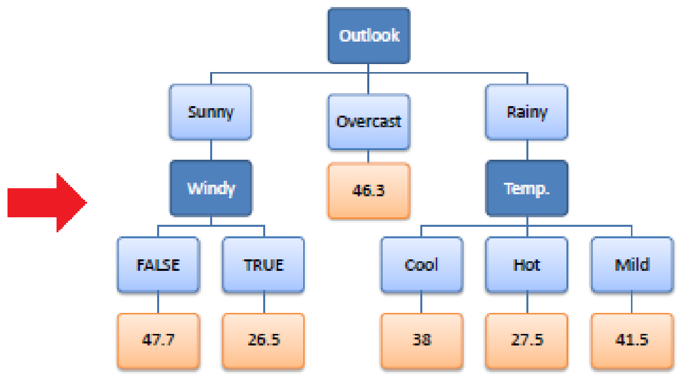
Step 4: In practice, we need some termination criteria. Check if coefficient of variation of the branch is less than the threshold or not and/or when too few instances (n) remain in the branch (e.g. 3), then terminate the splitting and the related leaf node gets the average of the target variable.

Step 5: Repeat above steps until all data is processed.

Example



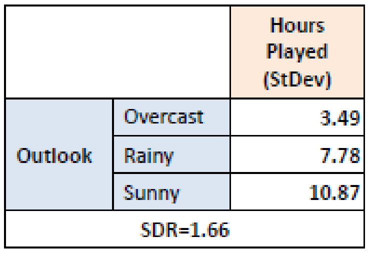
Predictors				Target
Outlook	Temp.	Humidity	Windy	Hours Played
Rainy	Hot	High	False	25
Rainy	Hot	High	True	30
Overcast	Hot	High	False	46
Sunny	Mild	High	False	45
Sunny	Cool	Normal	False	52
Sunny	Cool	Normal	True	23
Overcast	Cool	Normal	True	43
Rainy	Mild	High	False	35
Rainy	Cool	Normal	False	38
Sunny	Mild	Normal	False	46
Rainy	Mild	Normal	True	48
Overcast	Mild	High	True	52
Overcast	Hot	Normal	False	44
Sunny	Mild	High	True	30



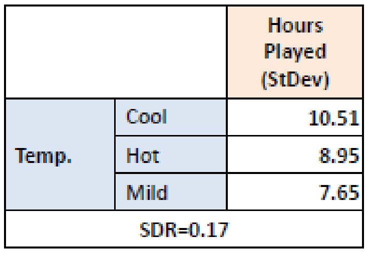
Step 1: The standard deviation of the target is calculated.

Standard deviation (Hours Played) = 9.32

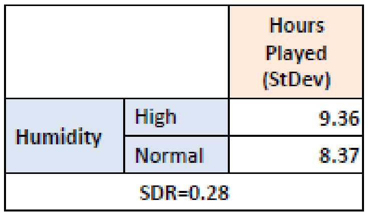
Step 2: The dataset is then split on the different attributes. The standard deviation for each branch is calculated. The resulting standard deviation is subtracted from the standard deviation before the split. The result is the standard deviation reduction



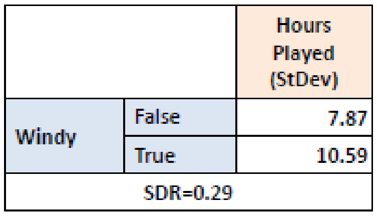
		Hours Played (StDev)
Outlook	Overcast	3.49
	Rainy	7.78
	Sunny	10.87
SDR=1.66		



		Hours Played (StDev)
Temp.	Cool	10.51
	Hot	8.95
	Mild	7.65
SDR=0.17		



		Hours Played (StDev)
Humidity	High	9.36
	Normal	8.37
SDR=0.28		



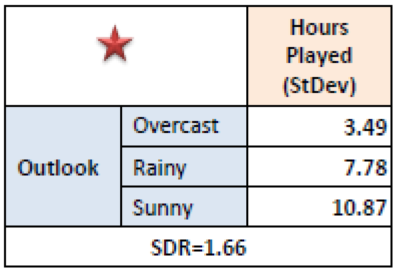
		Hours Played (StDev)
Windy	False	7.87
	True	10.59
SDR=0.29		

$$SDR(T,X) = S(T) - S(T,X)$$

SDR(Hours , Outlook) = **S**(Hours) – **S**(Hours, Outlook)

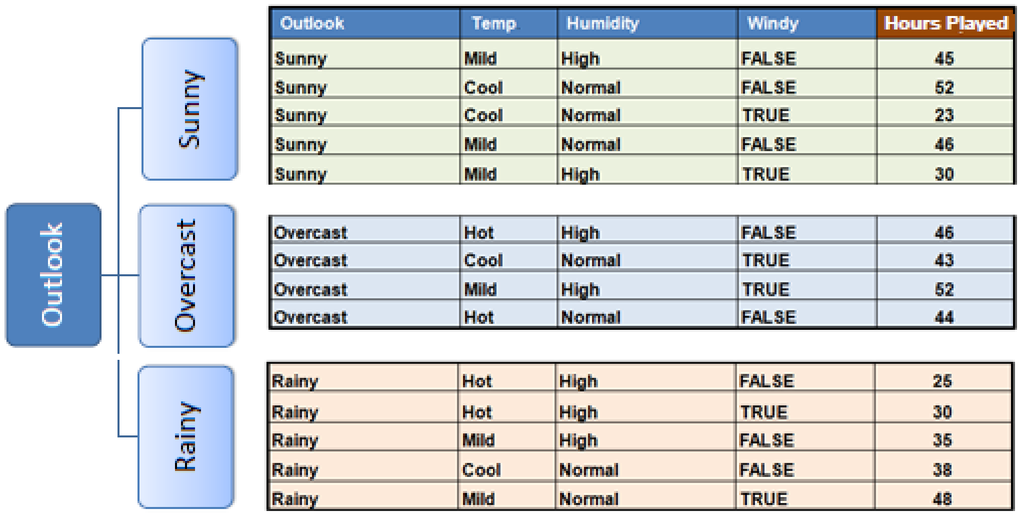
= 9.32 – 7.66 = 1.66

Step 3: The attribute with the largest standard deviation reduction is chosen for the decision node.



		Hours Played (StDev)
Outlook	Overcast	3.49
	Rainy	7.78
	Sunny	10.87
SDR=1.66		

Step 4a: The dataset is divided based on the values of the selected attribute. This process is run recursively on the non-leaf branches until all data is processed.

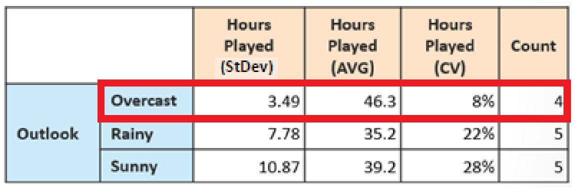


Outlook	Temp	Humidity	Windy	Hours Played
Sunny	Mild	High	FALSE	45
Sunny	Cool	Normal	FALSE	52
Sunny	Cool	Normal	TRUE	23
Sunny	Mild	Normal	FALSE	46
Sunny	Mild	High	TRUE	30
Overcast	Hot	High	FALSE	46
Overcast	Cool	Normal	TRUE	43
Overcast	Mild	High	TRUE	52
Overcast	Hot	Normal	FALSE	44
Rainy	Hot	High	FALSE	25
Rainy	Hot	High	TRUE	30
Rainy	Mild	High	FALSE	35
Rainy	Cool	Normal	FALSE	38
Rainy	Mild	Normal	TRUE	48

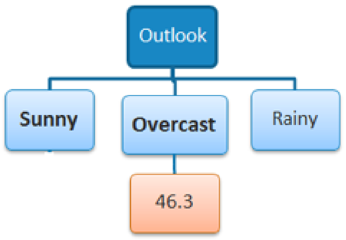
In practice, we need some termination criteria. For example, when the coefficient of deviation (CV) for a branch becomes smaller than a certain threshold (e.g. 10%) and/or when too few instances (n) remain in the branch (e.g. 3).

Step 4b: “Overcast” subset does not need any further splitting because its CV (8%) is less than the threshold (10%). The related leaf node gets the average of the "Overcast" subset.

Outlook - Overcast

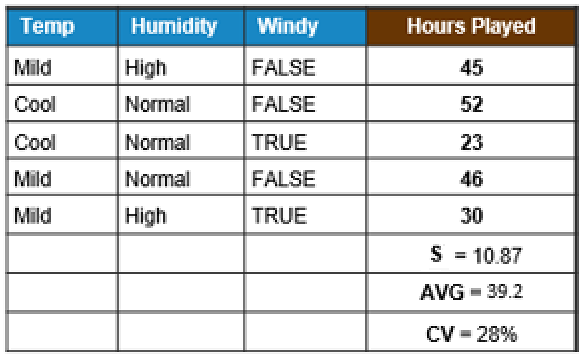


		Hours Played (StDev)	Hours Played (AVG)	Hours Played (CV)	Count
Outlook	Overcast	3.49	46.3	8%	4
	Rainy	7.78	35.2	22%	5
	Sunny	10.87	39.2	28%	5

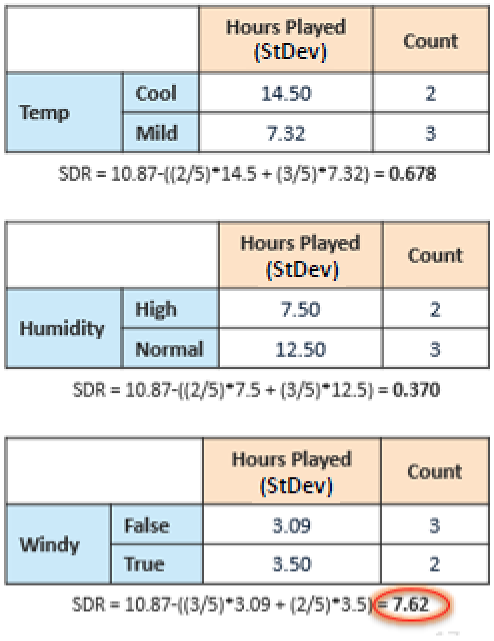


Step 4c: However, the "Sunny" branch has an CV (28%) more than the threshold (10%) which needs further splitting. We select "Windy" as the best node after "Outlook" because it has the largest SDR

Outlook - Sunny



Temp	Humidity	Windy	Hours Played
Mild	High	FALSE	45
Cool	Normal	FALSE	52
Cool	Normal	TRUE	23
Mild	Normal	FALSE	46
Mild	High	TRUE	30
			S = 10.87
			AVG = 39.2
			CV = 28%



		Hours Played (StDev)	Count
Temp	Cool	14.50	2
	Mild	7.32	3

SDR = 10.87-((2/5)*14.5 + (3/5)*7.32) = **0.678**

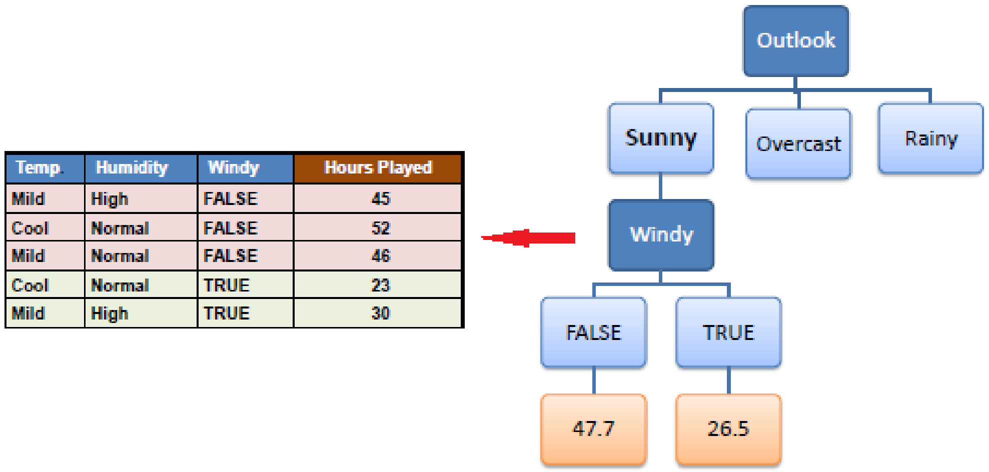
		Hours Played (StDev)	Count
Humidity	High	7.50	2
	Normal	12.50	3

SDR = 10.87-((2/5)*7.5 + (3/5)*12.5) = **0.370**

		Hours Played (StDev)	Count
Windy	False	3.09	3
	True	3.50	2

SDR = 10.87-((3/5)*3.09 + (2/5)*3.5) = **7.62**

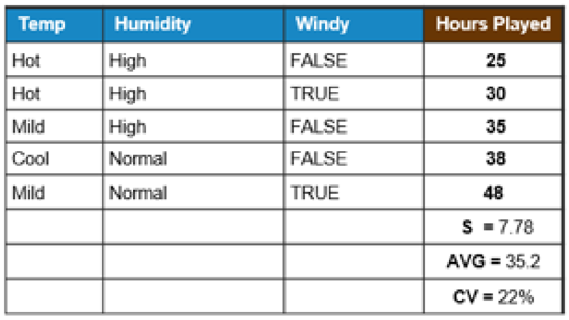
Because the number of data points for both branches (FALSE and TRUE) is equal or less than 3 we stop further branching and assign the average of each branch to the related leaf node.



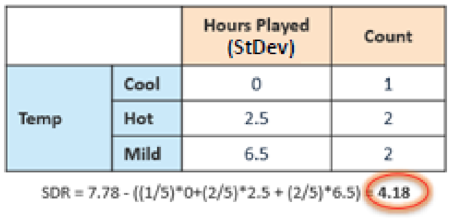
Temp	Humidity	Windy	Hours Played
Mild	High	FALSE	45
Cool	Normal	FALSE	52
Mild	Normal	FALSE	46
Cool	Normal	TRUE	23
Mild	High	TRUE	30

Step 4d: Moreover, the "rainy" branch has an CV (22%) which is more than the threshold (10%). This branch needs further splitting. We select "Windy" as the best node because it has the largest SDR.

Outlook - Rainy

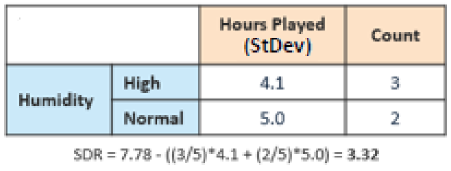


Temp	Humidity	Windy	Hours Played
Hot	High	FALSE	**25**
Hot	High	TRUE	**30**
Mild	High	FALSE	**35**
Cool	Normal	FALSE	**38**
Mild	Normal	TRUE	**48**
			S = 7.78
			AVG = 35.2
			CV = 22%



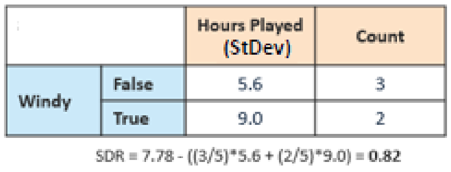
		Hours Played (StDev)	Count
Temp	Cool	0	1
	Hot	2.5	2
	Mild	6.5	2

SDR = 7.78 - ((1/5)*0+(2/5)*2.5 + (2/5)*6.5) = **4.18**



		Hours Played (StDev)	Count
Humidity	High	4.1	3
	Normal	5.0	2

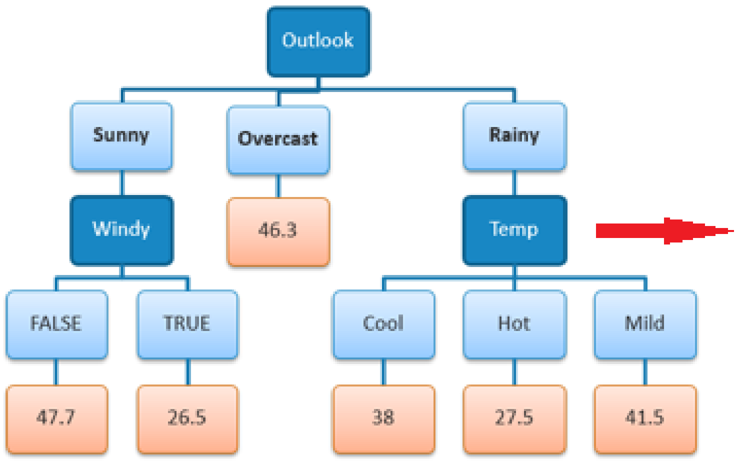
SDR = 7.78 - ((3/5)*4.1 + (2/5)*5.0) = **3.32**

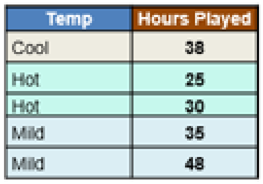


		Hours Played (StDev)	Count
Windy	False	5.6	3
	True	9.0	2

SDR = 7.78 - ((3/5)*5.6 + (2/5)*9.0) = **0.82**

Because the number of data points for all three branches (Cool, Hot and Mild) is equal or less than 3 we stop further branching and assign the average of each branch to the related leaf node.





Temp	Hours Played
Cool	**38**
Hot	**25**
Hot	**30**
Mild	**35**
Mild	**48**

When the number of instances is more than one at a leaf node we calculate the average as the final value for the target.

Information Gain

A measure of the decrease in the entropy after the data set is split is the information gain.

Information gain is a statistical property that measures how well a given attribute separates the training examples according to their target classification.

Information gain if X feature is used to make split can be represented by

Information Gain(T,X) = Entropy (T) – Entropy(T,X)

or

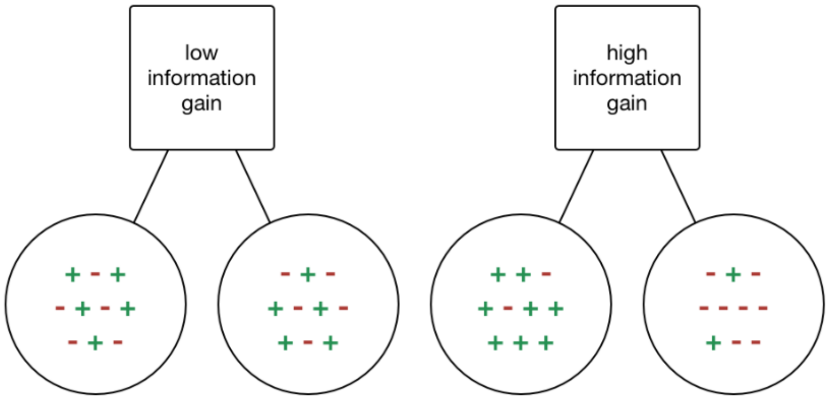
$$\text{Information gain} = \text{entropy (parent)} - \text{[weightes average]} * \text{entropy (children)}$$

Where [Weighted average] * entropy(children) = (no. of examples in left child node) / (total no. of examples in parent node) * (entropy of left node) + (no. of examples in right child node)/ (total no. of examples in parent node) * (entropy of right node)

What is entropy?

Entropy is a way to measure impurity. It is a measure of randomness or unpredictability in the data set. Entropy is a measure of disorder or uncertainty and since the goal of machine learning models in general is to reduce uncertainty the lower the entropy the easy is to draw any conclusion from data and on the contrary the higher the entropy, the harder it is to draw any conclusions from that information.

Entropy=−∑ pj log2pj



- The entropy is 0 if all samples of a node belong to the same class (the sample is homogeneous)

Entropy=$-1\log 1=0$

- The entropy is maximal if we have a uniform class distribution (the sample is equally divided)

Entropy=$-0.5\log 0.5-0.5\log 0.5=1$

- The more the entropy of a split, the less the information gained from that split. The more the entropy of a split, the less the disorder/uncertainty is reduced from target variable

- The less the entropy of a split, the more the information gained from that split. The less the entropy of a split, the more the disorder/uncertainty is reduced from target variable

- A branch with entropy of 0 is a leaf node.

- A branch with entropy more than 0 needs further splitting.

Steps

- Calculate entropy of target

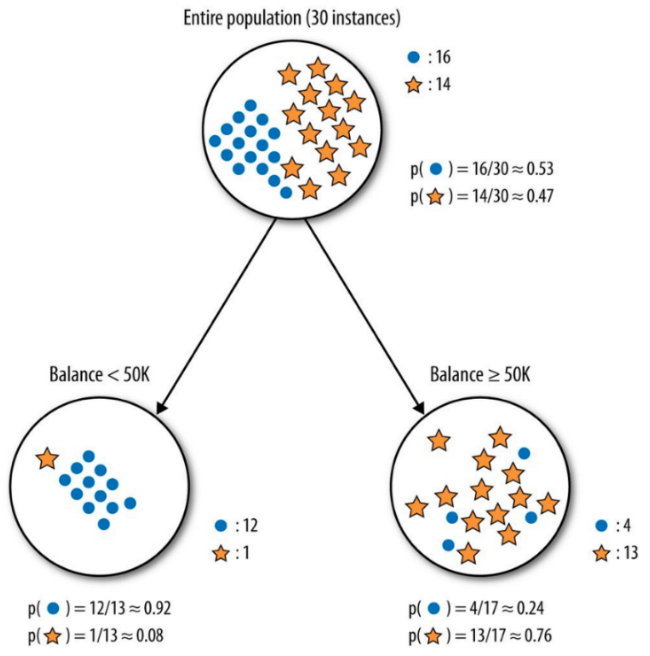
- Calculate the entropy of each split as the weighted average entropy of child nodes

- Select the split with the lowest entropy or highest information gain

- Repeat above steps until you achieve homogeneous nodes

Example

Consider an example where we are building a decision tree to predict whether a loan given to a person would result in a write-off or not. Our entire population consists of 30 instances. 16 belong to the write-off class and the other 14 belong to the non-write-off class. We have two features, namely “Balance” that can take on two values - “< 50K” or “>50K” and “Residence” that can take on three values - “OWN”, “RENT” or “OTHER”.



The dots are the data points with class right-off and the stars are the non-write-offs. Splitting the parent node on attribute balance gives us 2 child nodes. The left node gets 13 of the total observations with 12/13 (0.92 probability) observations from the write-off class and only 1/13(0.08 probability) observations from the non-write of class. The right node gets 17 of the total observation with 13/17(0.76 probability) observations from the non-write-off class and 4/17 (0.24 probability) from the write-off class. Let's calculate the entropy for the parent node and see how much uncertainty the tree can reduce by splitting on Balance.

$$E(\mathit{Parent}) \ = - \ \frac{16}{30}\log_2\left(\frac{16}{30}\right) - \frac{14}{30}\log_2\left(\frac{14}{30}\right) \approx 0.99$$

$$E(\mathit{Balance} \ < \ 50K) \ = - \frac{12}{13}\log_2\left(\frac{12}{13}\right) - \frac{1}{13}\log_2\left(\frac{1}{13}\right) \approx 0.39$$

$$E(\mathit{Balance} \ > \ 50K) \ = - \frac{4}{17}\log_2\left(\frac{4}{17}\right) - \frac{13}{17}\log_2\left(\frac{13}{17}\right) \approx 0.79$$

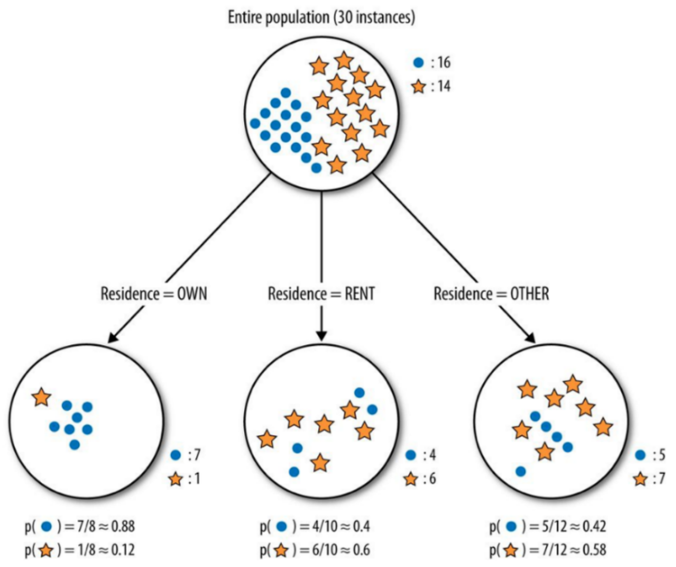
$\mathit{Weighted\ Average\ of\ \ entropy\ for\ each\ node}:$

$$\begin{aligned} E(\mathit{Balance}) \ &= \frac{13}{30}\times 0.39 \ + \ \frac{17}{30}\times 0.79 \\ &= \ 0.62 \end{aligned}$$

$\mathit{Information\ Gain}:$

$$\begin{aligned} IG(\mathit{Parent},\ \mathit{Balance}) \ &= E(\mathit{Parent}) \ - \ E(\mathit{Balance}) \\ &= 0.99 \ - \ 0.62 \\ &= \mathbf{0.37} \end{aligned}$$

Splitting on feature, “Balance” leads to an information gain of 0.37 on our target variable. Let’s do the same thing for feature, “Residence” to see how it compares. Splitting the tree on Residence gives us 3 child nodes. The left child node gets 8 of the total observations with 7/8 (0.88 probability) observations from the write-off class and only 1/8 (0.12 probability) observations from the non-write-off class. The middle child nodes get 10 of the total observations with 4/10 (0.4 probability) observations of the write-off class and 6/10 (0.6 probability) observations from the non-write-off class. The right child node gets 12 of the total observations with 5/12 (0.42 probability) observations from the write-off class and 7/12 (0.58) observations from the non-write-off class. We already know the entropy for the parent node. We simply need to calculate the entropy after the split to compute the information gain from “Residence”



$$E(Residence = OWN) = -\frac{7}{8}\log_2\left(\frac{7}{8}\right) - \frac{1}{8}\log_2\left(\frac{1}{8}\right) \approx 0.54$$

$$E(Residence = RENT) = -\frac{4}{10}\log_2\left(\frac{4}{10}\right) - \frac{6}{10}\log_2\left(\frac{6}{10}\right) \approx 0.97$$

$$E(Residence = OTHER) = -\frac{5}{12}\log_2\left(\frac{5}{12}\right) - \frac{7}{12}\log_2\left(\frac{7}{12}\right) \approx 0.98$$

$Weighted\ Average\ of\ entropies\ for\ each\ node:$

$$E(Residence) = \frac{8}{30} \times 0.54 + \frac{10}{30} \times 0.97 + \frac{12}{30} \times 0.98 = 0.86$$

$Information\ Gain:$

$$\begin{aligned} IG(Parent, Residence) &= E(Parent) - E(Residence) \\ &= 0.99 - 0.86 \\ &= 0.13 \end{aligned}$$

The information gain from feature Balance is almost 3 times more than the information gain from Residence. Balance provides more information about our target variable than Residence. It reduces more disorder in our target variable. A decision tree algorithm would use this result to make the first split on our data using Balance.

Gini index or Gini Impurity

Is a criterion to minimize the probability of misclassification

It can be thought of as a cost function used to evaluate splits in the dataset.

It is a metric to measure how often a randomly chosen element would be incorrectly identified.

It is calculated by subtracting the sum of the squared probabilities of each class from one. Gini Index works with the categorical target variable “Success” or “Failure”. It performs only Binary splits.

Gini=1−∑ p^2j

- Gini index is maximal if the classes are perfectly mixed (example in a binary class)

Gini=1−(p1^2+p2^2) =1−((0.5^2) +(0.5^2)) =0.5

- An attribute with lower gini index have less chances to be misclassified as compared to one with higher gini index. Therefore, building the decision tree, we would prefer choosing the attribute/feature with the least Gini index to make the split.

- Gini is preferred over Entropy as it is faster in computation

Steps

- Calculate the Gini Impurity of each split as the weighted average Gini Impurity of child nodes

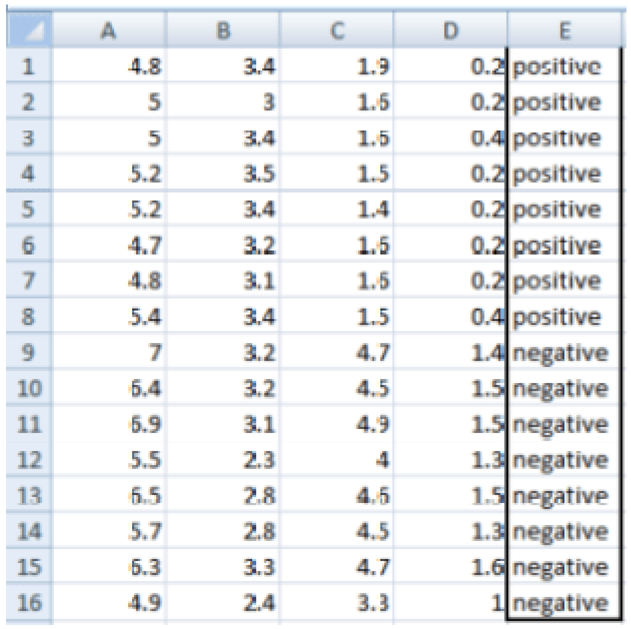
- Select the split with the lowest value of Gini Impurity

- Repeat above steps until you achieve homogeneous nodes

Example

We are going to use same data sample that we used for information gain example. Let’s try to use gini index as a criterion. Here, we have 5 columns out of which 4 columns have continuous data and 5th column consists of class labels.

A, B, C, D attributes can be considered as predictors and E column class labels can be considered as a target variable. For constructing a decision tree from this data, we have to convert continuous data into categorical data.



	A	B	C	D	E
1	4.8	3.4	1.9	0.2	positive
2	5	3	1.6	0.2	positive
3	5	3.4	1.6	0.4	positive
4	5.2	3.5	1.5	0.2	positive
5	5.2	3.4	1.4	0.2	positive
6	4.7	3.2	1.6	0.2	positive
7	4.8	3.1	1.6	0.2	positive
8	5.4	3.4	1.5	0.4	positive
9	7	3.2	4.7	1.4	negative
10	6.4	3.2	4.5	1.5	negative
11	6.9	3.1	4.9	1.5	negative
12	5.5	2.3	4	1.3	negative
13	6.5	2.8	4.6	1.5	negative
14	5.7	2.8	4.5	1.3	negative
15	6.3	3.3	4.7	1.6	negative
16	4.9	2.4	3.3	1	negative

We have chosen some random values to categorize each attribute:

A	B	C	D
>= 5	>= 3.0	>=4.2	>= 1.4
< 5	< 3.0	< 4.2	< 1.4

Gini Index for Var A

Var A has value >=5 for 12 records out of 16 and 4 records with value <5 value.

For Var A >= 5 & class == positive: 5/12

For Var A >= 5 & class == negative: 7/12

gini(5,7) = 1- ((5/12)2 + (7/12)2) = 0.4860

For Var A <5 & class == positive: 3/4

For Var A <5 & class == negative: 1/4

gini(3,1) = 1- ((3/4)2 + (1/4)2) = 0.375

By adding weight and sum each of the gini indices:

$$\text{gini}(\text{Target, A}) = (12/16) * (0.486) + (4/16) * (0.375) = 0.45825$$

Gini Index for Var B

Var B has value >=3 for 12 records out of 16 and 4 records with value <5 value.

For Var B >= 3 & class == positive: 8/12

For Var B >= 3 & class == negative: 4/12

gini(8,4) = 1- ((8/12)2 + (4/12)2) = 0.446

For Var B <3 & class == positive: 0/4

For Var B <3 & class == negative: 4/4

gin(0,4) = 1- ((0/4)2 + (4/4)2) = 0

$$\text{gini}(\text{Target, B}) = (12/16) * 0.446 + (4/16) * 0= 0.3345$$

Gini Index for Var C

Var C has value >=4.2 for 6 records out of 16 and 10 records with value <4.2 value.

For Var C >= 4.2 & class == positive: 0/6

For Var C >= 4.2 & class == negative: 6/6

gini(0,6) = 1- ((0/8)2 + (6/6)2) = 0

For Var C < 4.2& class == positive: 8/10

For Var C < 4.2 & class == negative: 2/10

gin(8,2) = 1- ((8/10)2 + (2/10)2) = 0.32

$$\text{gini}(\text{Target, C}) = (6/16) * 0+ (10/16) * 0.32 = 0.2$$

Gini Index for Var D

Var D has value >=1.4 for 5 records out of 16 and 11 records with value <1.4 value.

For Var D >= 1.4 & class == positive: 0/5

For Var D >= 1.4 & class == negative: 5/5

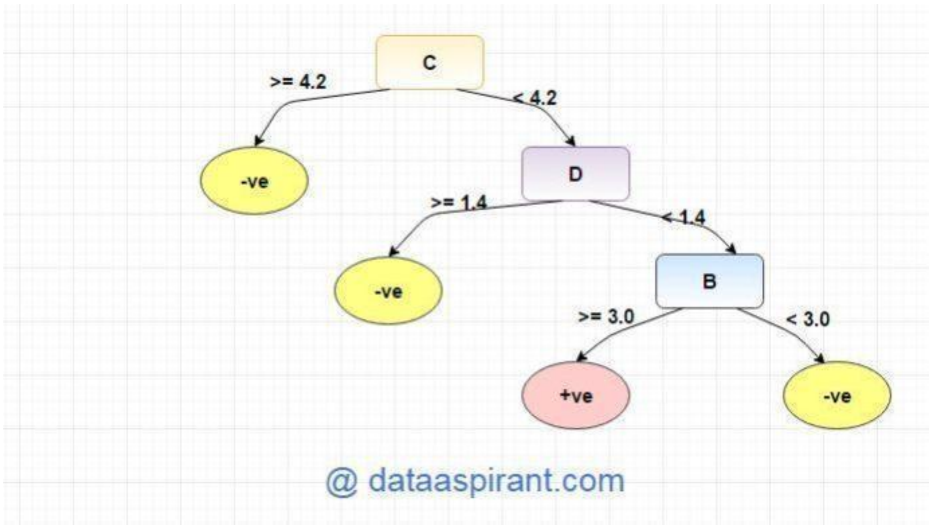
gini(0,5) = 1- ((0/5)2 + (5/5)2) = 0

For Var D < 1.4 & class == positive: 8/11

For Var D < 1.4 & class == negative: 3/11

gini(8,3) = 1- ((8/11)2 + (3/11)2) = 0.397

$$\text{gini}(\text{Target, D}) = (5/16) * 0 + (11/16) * 0.397 = 0.273$$



PROBLEM WITH DECISION TREE

When we build a decision tree, it considers all columns in dataset and chooses the best column branch by branch to make the split. So the column that made split at root node is considered more important than columns at the other branches and hence all predictions will have huge impact of column at the top and when in real time scenario if there comes an observation for which the first column is not the most important feature to make prediction, the model will fail to predict well.

This problem leads to high variance i.e. accuracy decreases when we change the dataset to predict. The solution is to use random forest model which reduces variance by training models using a random subset of features on a random subset of samples. So the output is not from one decision tree where columns with their fixed importance levels are used for all samples' predictions but rather from the output of many decision trees where different features were considered different importances.

Pruning

Overfitting is a practical problem while building a decision tree model. We can overcome the problem of overfitting using Random Forest algorithm or Pruning technique.

Pruning is the process of removing leaves and branches to improve the performance of the decision tree. Pruning is the opposite process of splitting.

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.

Pre-Pruning

In pre-pruning, it stops the tree construction bit early. This approach stops the non-significant branches from generation. It terminates the generation of new branch based on the given condition. It is preferred not to split a node if its goodness measure is below a threshold value.

Post-Pruning

Full tree is generated and then the non-significant branches are pruned/removed. Cross validation is performed at every step to check whether addition of the new branch leads to increase in accuracy. If not the branch is converted to leaf node.

Some important parameters to perform pruning are max_leaf_nodes which reduces the number of leaf nodes, min_samples_leaf which restricts the size of sample leaf, max_depth which reduces the depth of the tree to build a generalized tree