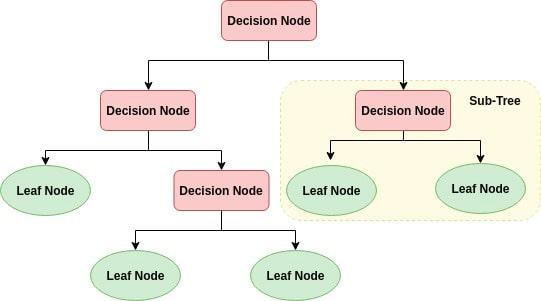
**Decision Tree**

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

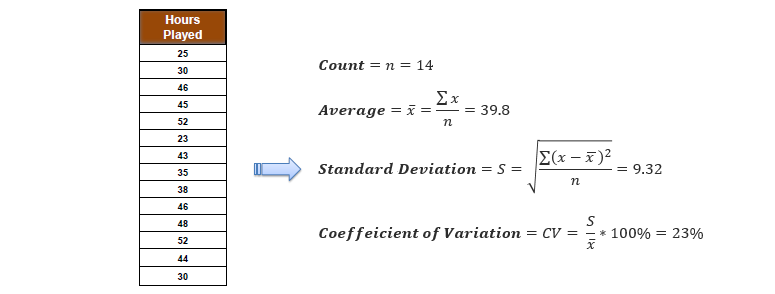
- Gini Impurity

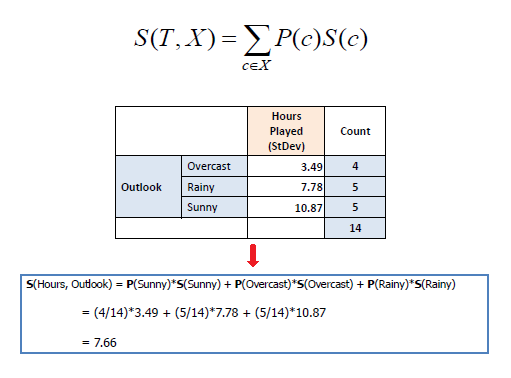
**Reduction in standard deviation**

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.

How it works

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





Steps

Step 1: Calculate standard deviation of target

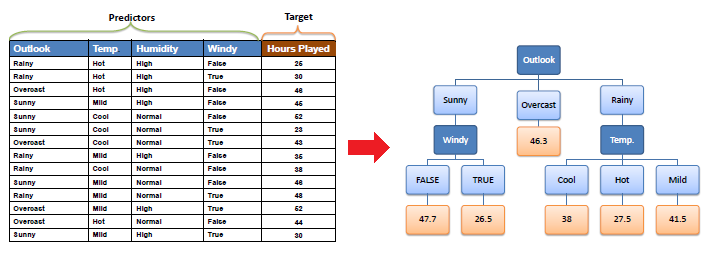
Step 2: Calculate the standard deviation of each split as the weighted average standard deviation of child nodes

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

Step 4: Check if coefficient of variation is less than the threshold or not and/or when too few instances (n) remain in the branch (e.g. 3). A branch doesn’t need splitting if its coefficient of variation of the feature splitting the branch is less than the threshold and/or when too few instances (n) remain in the branch (e.g. 3) and the related leaf node gets the average of the target variable. If coefficient of variation is equal to or more than threshold, select the split with the highest standard deviation variance

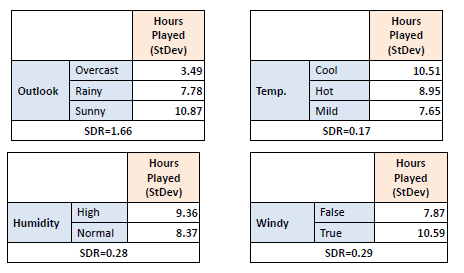
Step 5: Repeat above steps until all data is processed.

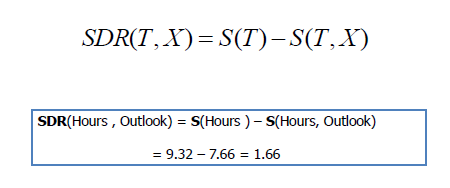
Example



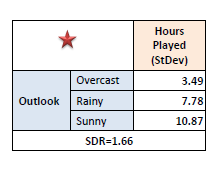
| Step 1: The standard deviation of the target is calculated. |  |  |
| --- | --- | --- |
|  |  |  |
| Standard deviation (Hours Played) = 9.32 |  |  |

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

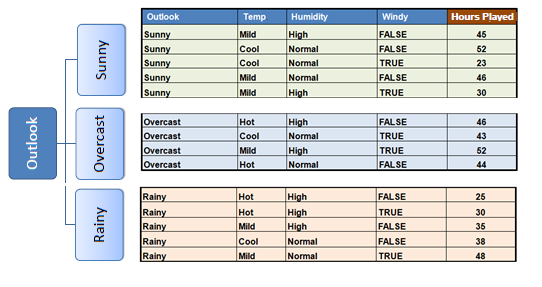




Step 3: The attribute with the largest standard deviation reduction is chosen for the decision node.

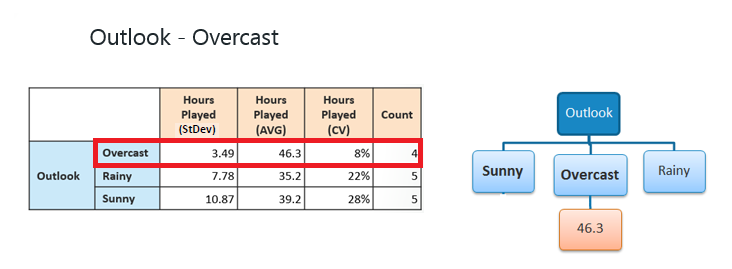


Step 4a: The dataset is divided based on the values of the selected attribute. This process is run recursively on the non-leaf branches, until all data is processed.

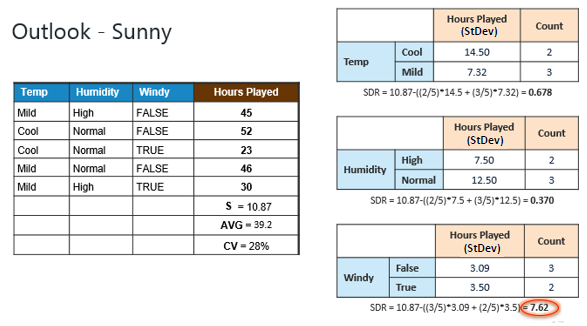


In practice, we need some termination criteria. For example, when coefficient of deviation (CV) for a branch becomes smaller than a certain threshold (e.g. 10%) and/or when too few instances (n) remain in the branch (e.g. 3).

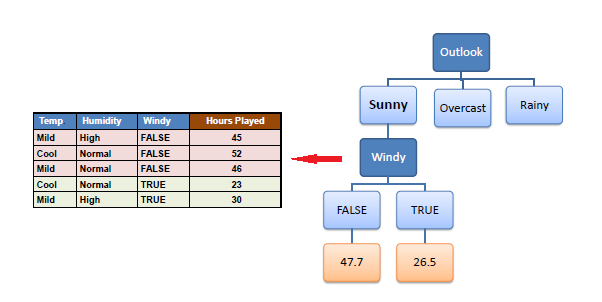
Step 4b: “Overcast” subset does not need any further splitting because its CV (8%) is less than the threshold (10%). The related leaf node gets the average of the "Overcast" subset.



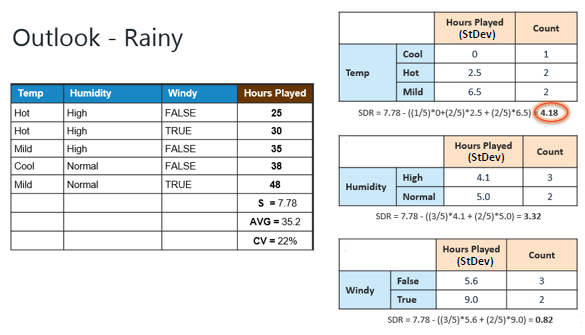
Step 4c: However, the "Sunny" branch has an CV (28%) more than the threshold (10%) which needs further splitting. We select "Windy" as the best node after "Outlook" because it has the largest SDR



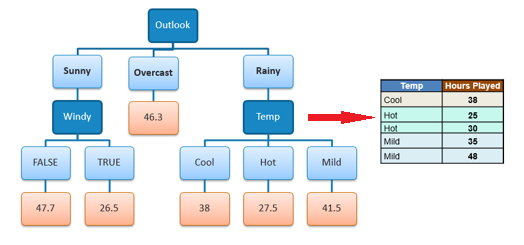
Because the number of data points for both branches (FALSE and TRUE) is equal or less than 3 we stop further branching and assign the average of each branch to the related leaf node.



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.



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.



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**

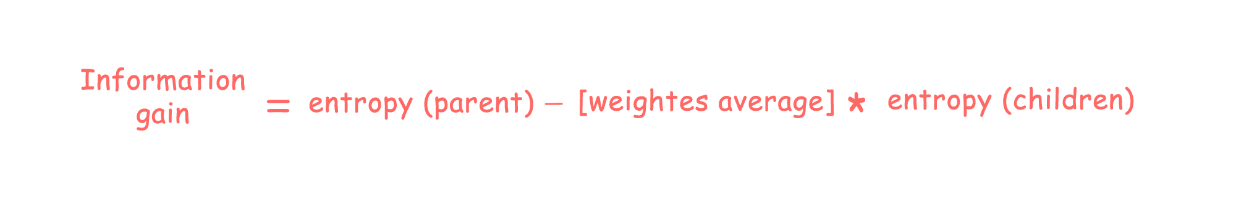
A measure of the decrease in the entropy after the data set is split is the information gain.

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

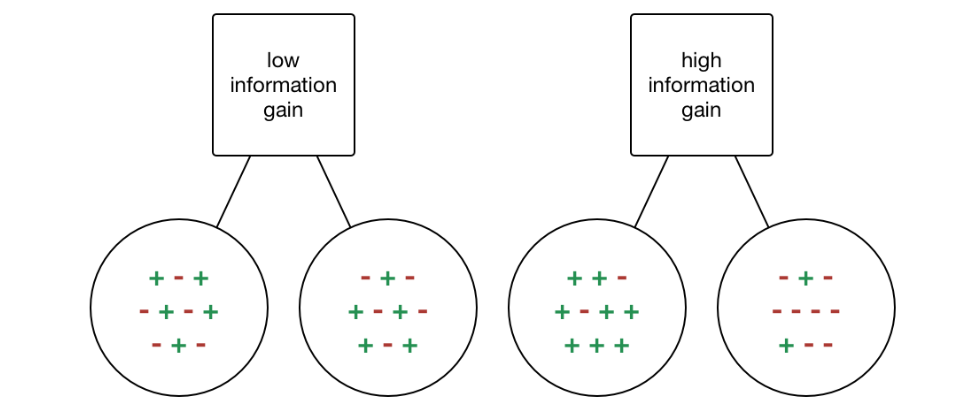


Where [Weighted average] \* entropy(children) = (no. of examples in left child node) / (total no. of examples in parent node) \* (entropy of left node) + (no. of examples in right child node)/ (total no. of examples in parent node) \* (entropy of right node)

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=−1log1=0

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

Entropy=−0.5log0.5−0.5log0.5=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

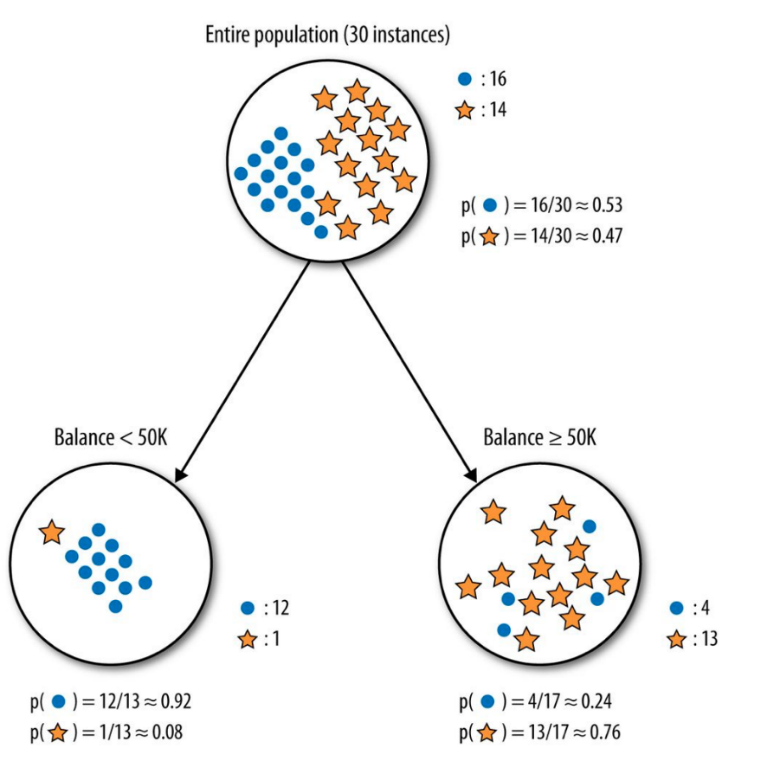
- Calculate the entropy of each split as the weighted average entropy of child nodes

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

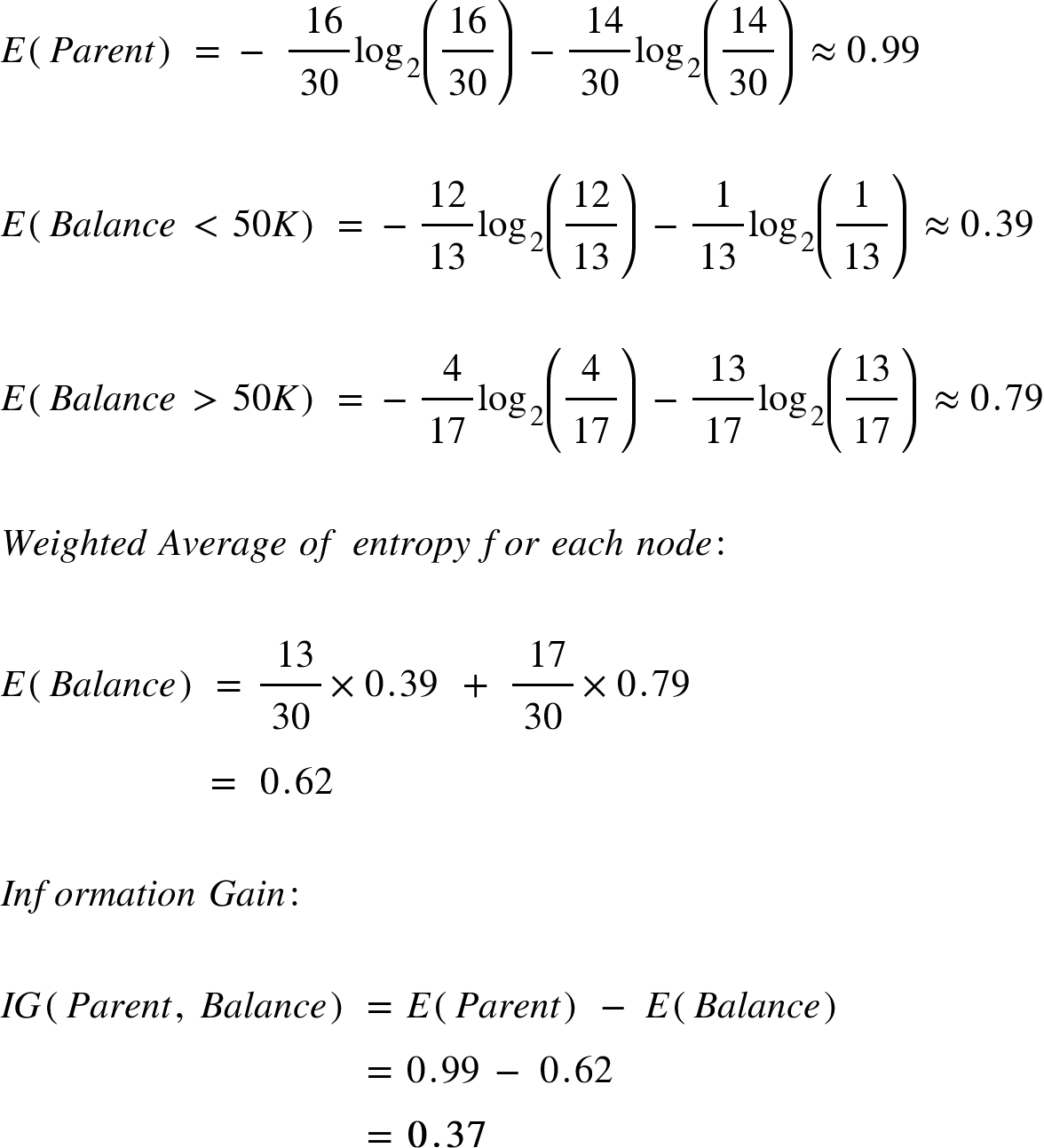
- Repeat above steps until you achieve homogeneous nodes

Example

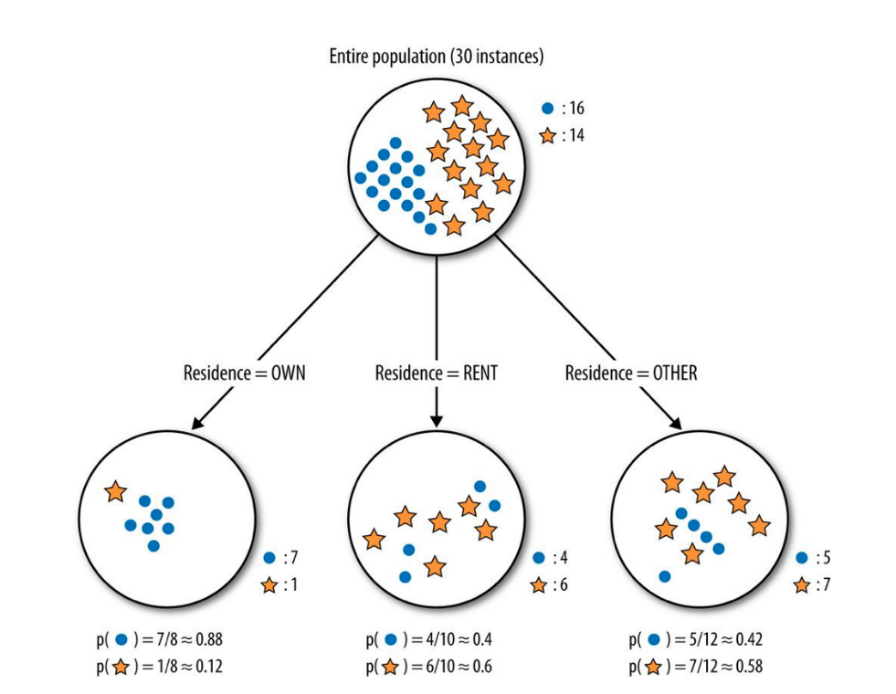
Consider an example where we are building a decision tree to predict whether a loan given to a person would result in a write-off or not. Our entire population consists of 30 instances. 16 belong to the write-off class and the other 14 belong to the non-write-off class. We have two features, namely “Balance”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

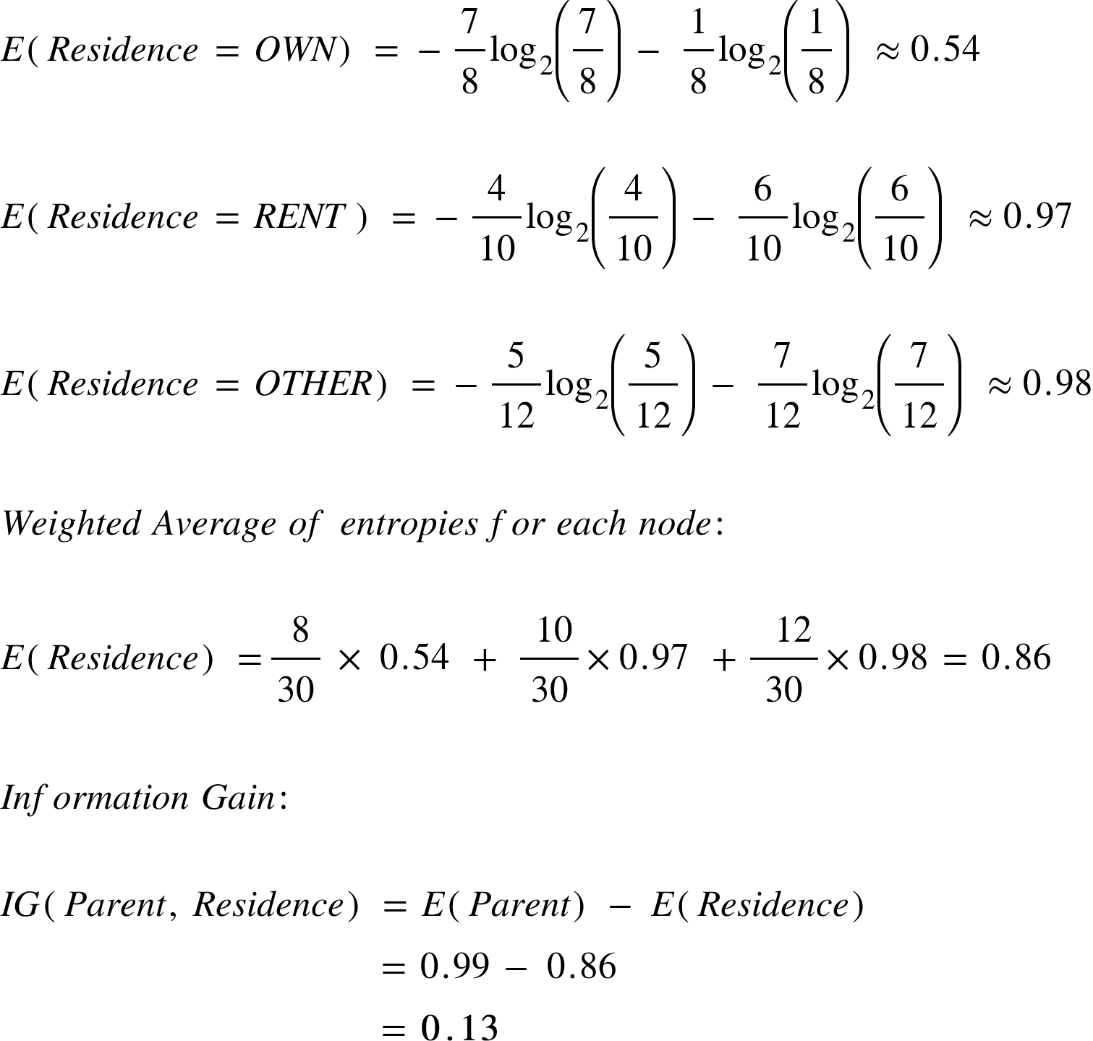


The dots are the data points with class right-off and the stars are the non-write-offs. Splitting the parent node on attribute balance gives us 2 child nodes. The left node gets 13 of the total observations with 12/13 (0.92 probability) observations from the write-off class and only 1/13(0.08 probability) 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.



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”





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

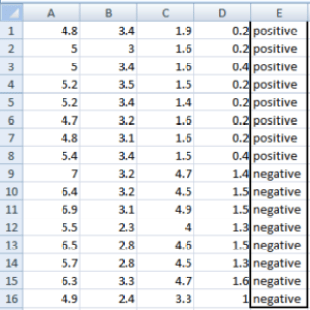
- Select the split with the lowest value of Gini Impurity

- 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.



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:

\textrm{gini(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

\textrm{gini(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

\textrm{gini(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

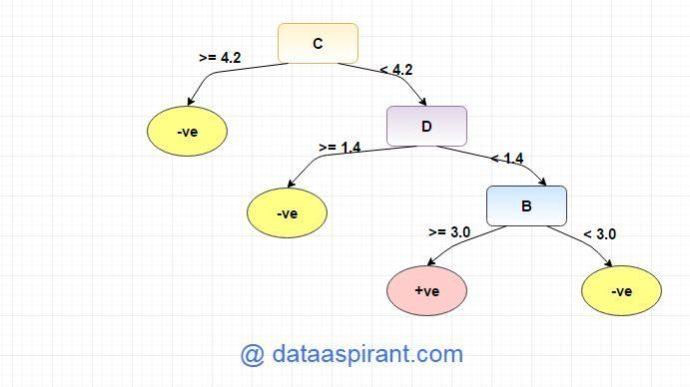
gini(0,5) = 1- ( (0/5)2 + (5/5)2 ) = 0

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

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



**Problem with Decision Trees**

Overfitting is a practical problem while building a decision tree model. It generally happens when it builds many branches in the model most likely to catch patterns that are only applicable to our dataset and not real word. Since the algorithm continues splitting on attributes until either it classifies all the data points or there are no more attributes to splits on. As a result, it is prone to creating decision trees that over fit by performing really well on the training data at the expense of accuracy with respect to the entire distribution of data. In other words, tree suffers from high variance. High Variance means getting high prediction error on unseen data. This means that if we split the training data into two parts at random, and fit a decision tree to both halves, the results that we get could be quite different.

- One more problem with decision tree is that averaging several highly correlated trees doesn't lead to a large reduction in variance. But how do correlated trees emerge? Let's say a data set has a very strong predictor, along with other moderately strong predictors. In bagging, a tree grown every time would consider the very strong predictor at its root node, thereby resulting in trees similar to each other.

**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 improves the performance of a tree. Pruning starts at leaves and removes each node with most popular class in that leaf, this change is kept if it doesn't deteriorate [accuracy](https://medium.com/towards-data-science/balancing-bias-and-variance-to-control-errors-in-machine-learning-16ced95724db). It 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.

The tree-building algorithm makes the best split at the root node where there are the largest number of records, and considerable information. Each subsequent split has a smaller and less representative population with which to work. Towards the end the idiosyncrasies of training records at a particular node display patterns that are peculiar only to those records. These patterns can become meaningless for prediction if you try to extend rules based on them to larger populations. So we must remove these patterns

Two approaches which we can use to avoid overfitting are:

Pre-Pruning

In pre-pruning, it stops the tree construction bit early. It is preferred not to split a node if its goodness measure is below a threshold value. But it’s difficult to choose an appropriate stopping point.

Post-Pruning

In post-pruning first, it goes deeper and deeper in the tree to build a complete tree. If the tree shows the overfitting problem, then pruning is done as a post-pruning step.

How Pruning works?

We can avoid overfitting by changing the parameters like

max\_leaf\_nodes- Reduces the number of leaf nodes

min\_samples\_leaf- Restricts the size of sample leaf.

max\_depth- Reduces the depth of the tree to build a generalized tree