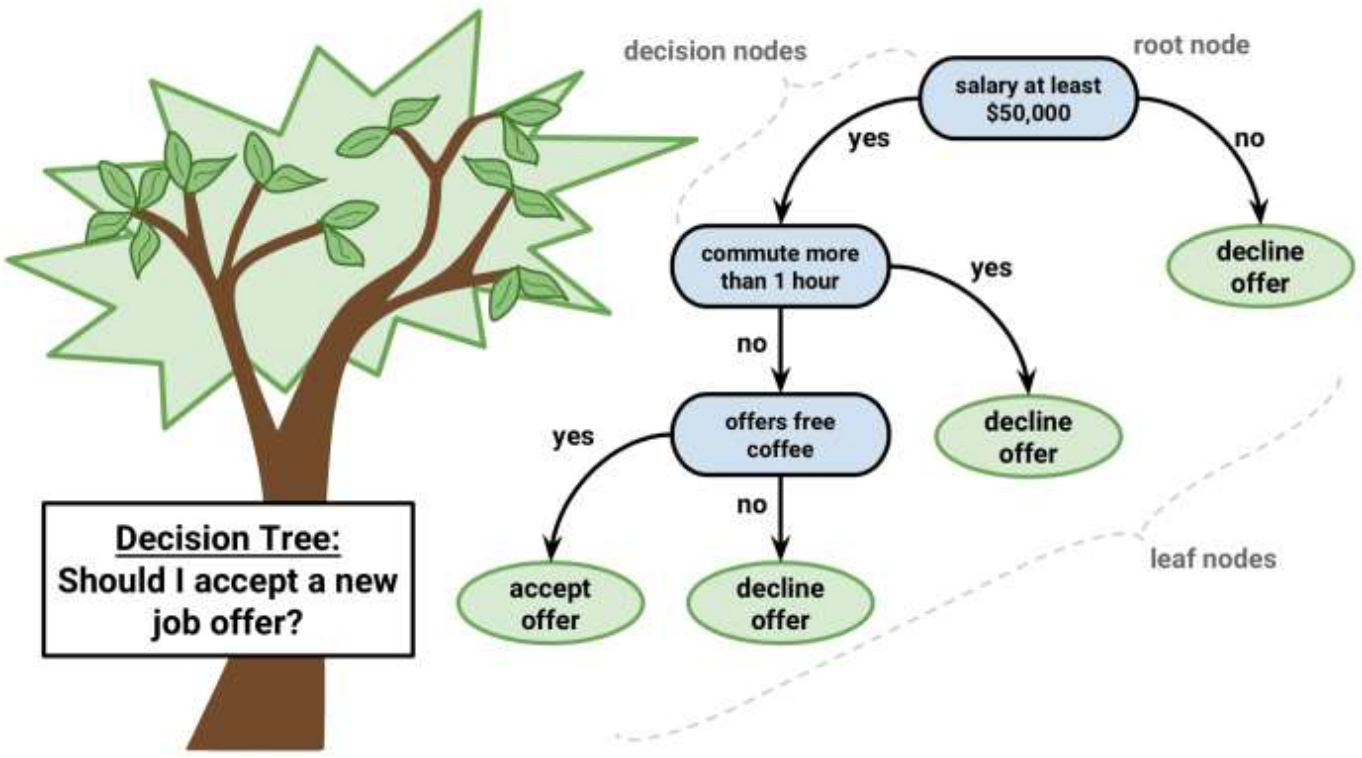
Mathematics behind Decision Tree

Rana singh Follow
Oct 8, 2019 · 6 min read



Decision tree based on the nested if-else classifier. it is the set of the axis-parallel hyperplane which divides the region into a hypercube.

The decision tree builds classification or regression models in the form of a tree structure. It breaks down a dataset into smaller subsets with an increase in depth of the tree. The final result is a tree with **decision nodes** and **leaf nodes**. A decision node (e.g., Outlook) has two or more branches (e.g., Sunny, Overcast and Rainy). Leaf node (e.g., Play) represents a classification or decision. The topmost decision node in a tree that corresponds to the best predictor is called the **root node**. Decision trees can handle both categorical and numerical data.

Terminology:

1. **Root Node:** It represents entire population or sample and this further gets divided into two or more homogeneous sets.
2. **Splitting:** It is a process of dividing a node into two or more sub-nodes.
3. **Decision Node:** When a sub-node splits into further sub-nodes, then it is called decision node.

4. **Leaf/ Terminal Node:** Nodes with no children (no further split) is called Leaf or Terminal node.
5. **Pruning:** When we reduce the size of decision trees by removing nodes (opposite of Splitting), the process is called pruning.
6. **Branch / Sub-Tree:** A sub section of decision tree is called branch or sub-tree.
7. **Parent and Child Node:** A node, which is divided into sub-nodes is called parent node of sub-nodes where as sub-nodes are the child of parent node.

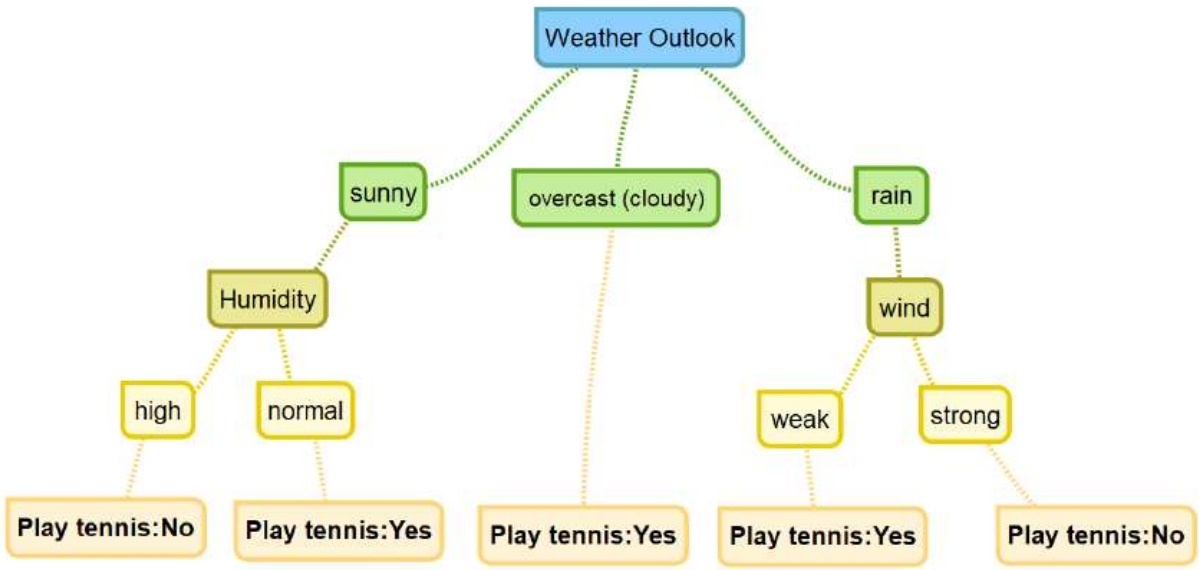


Figure 1: Play Tennis Decision tree

The algorithm used in decision trees:

PlayTennis: training examples

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

$$Entropy = -\sum p(X)\log p(X)$$

here p(x) is a fraction of examples in a given class

since above dataset contain two class in output. first find out probability of each class in output(P(y+) and P(y-)).

P(y+) = 9/14 and P(y-)=5/14

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

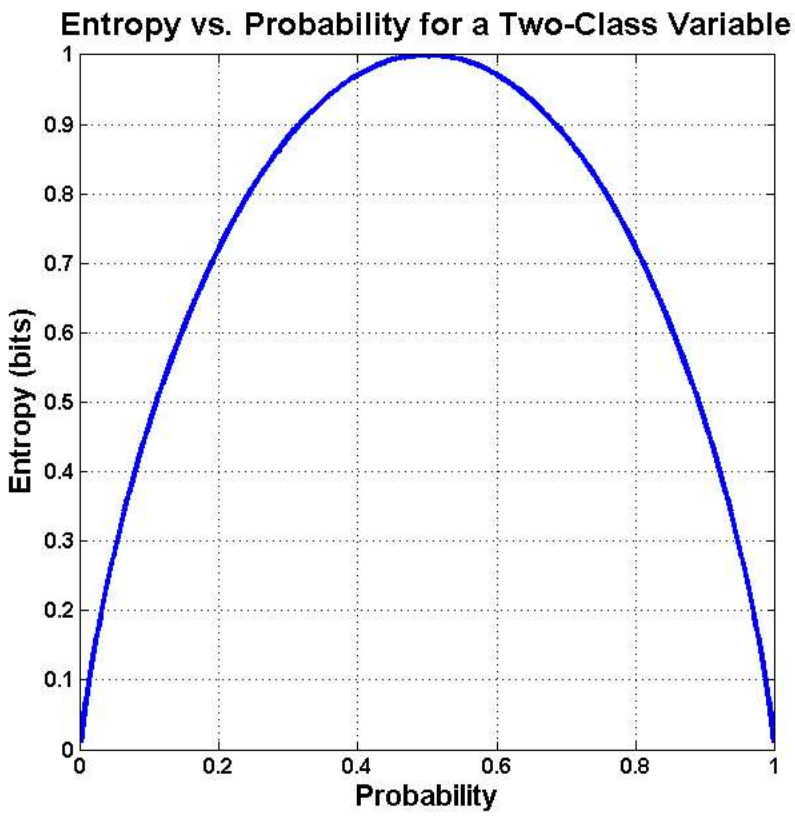
Play Golf	
Yes	No
9	5

Entropy(PlayGolf) = Entropy (5,9)
= Entropy (0.36, 0.64)
= - (0.36 $\log_2$ 0.36) - (0.64 $\log_2$ 0.64)
= 0.94

Properties:

if classes are equally distributed than entropy=1. if one class fully dominated than entropy=0.

If the sample is completely homogeneous the entropy is zero and if the sample is equally divided then it has an entropy of one.



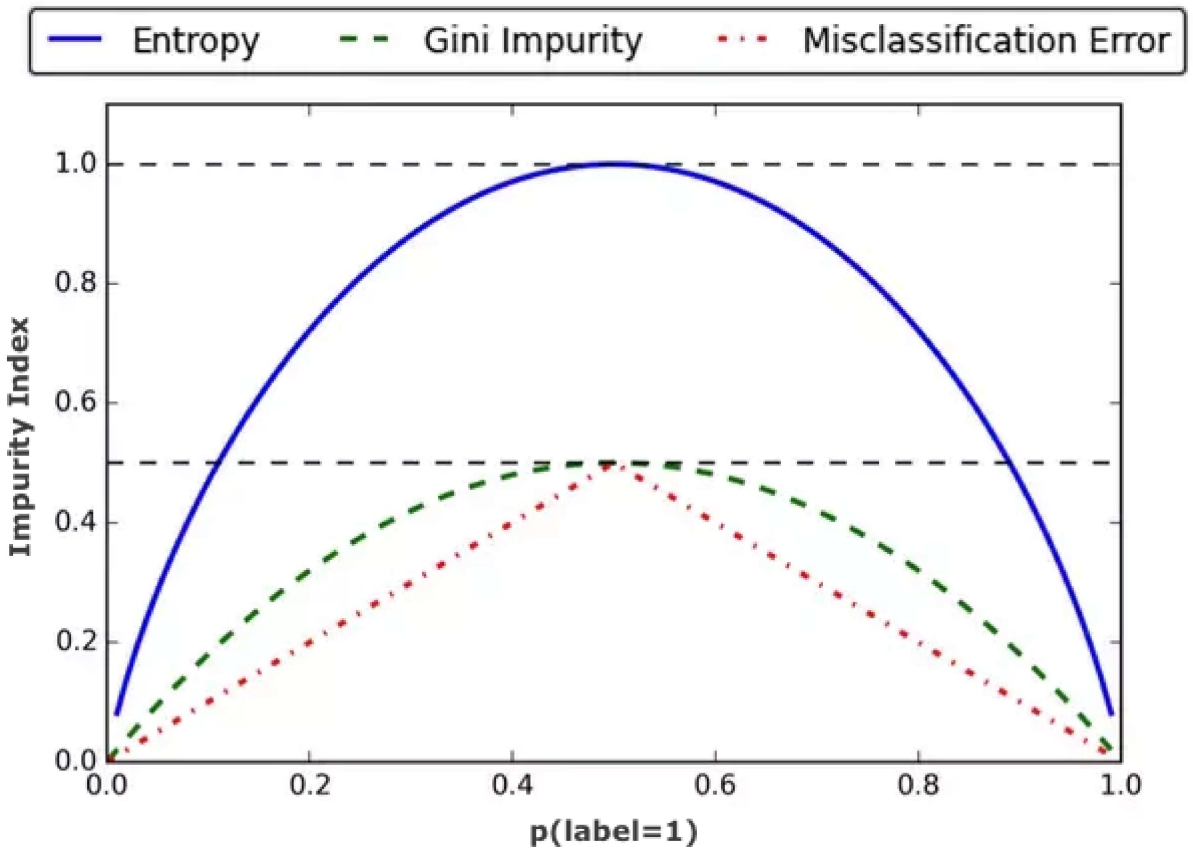
•For Gaussian distribution, the data set is widely distributed so entropy is maximum but less compare to uniform distribution because all have equal value.

•In case of uniform distribution, the dataset is equally distributed so have maximum entropy

- For peaked distribution, entropy is minimum near to zero because data set unequally distributed.

Gini impurity:

$$Gini = 1 - \sum_{i=1}^{C} (p_i)^2$$



Gini impurity faster to compute because squire is very easy to calculate than entropy, so Gini impurity is computationally faster. Time taken in case of log calculation is much faster than a squire. so Gini is much faster.

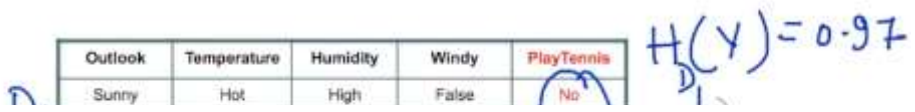
Information Gain:

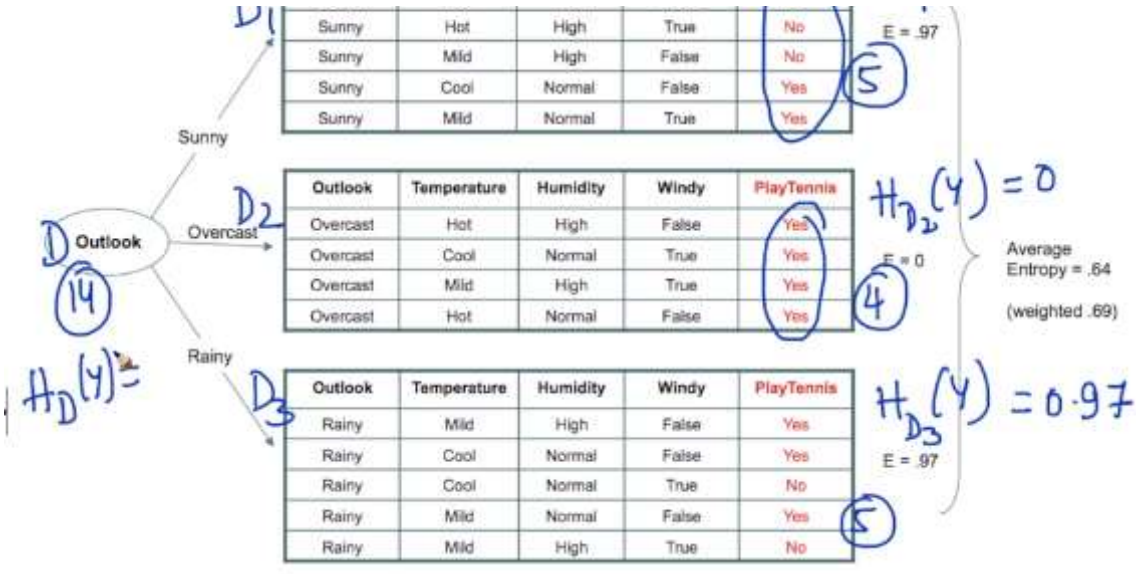
Constructing a decision tree is all about finding an attribute that returns the highest information gain (i.e., the most homogeneous branches). First, calculate entropy before the break of the variable(0.94). than calculate entropy after the break.

Information Gain

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

f: feature split on
D_p: dataset of the parent node
D_{left}: dataset of the left child node
D_{right}: dataset of the right child node
I: impurity criterion (Gini Index or Entropy)
N: total number of samples
N_{left}: number of samples at left child node
N_{right}: number of samples at right child node





calculate the entropy value for each class in the variable.

Construction of DT:

Step 1: Calculate entropy of the target.

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.

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.

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

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

Split categorical variable:

•Try to break using all feature and pick one which has maximum information gain.

•Break each feature, find information gain of each node and select node have maximum IG.

•Break until it becomes a pure node.

•If we have very few points, we don’t grow trees cause overfitting to noise increases.

•If the depth is small than underfitting.

Splitting numerical features:

A categorical variable is easy to break but not a numerical variable. For numerical feature, first sort it in increasing order, make a class (f1,f2…) each f have two datasets so compute ID of each f. then compare with each value as a potential threshold, find f1 which have maximum IG value where you will split to form decision tree.

Feature standardization:

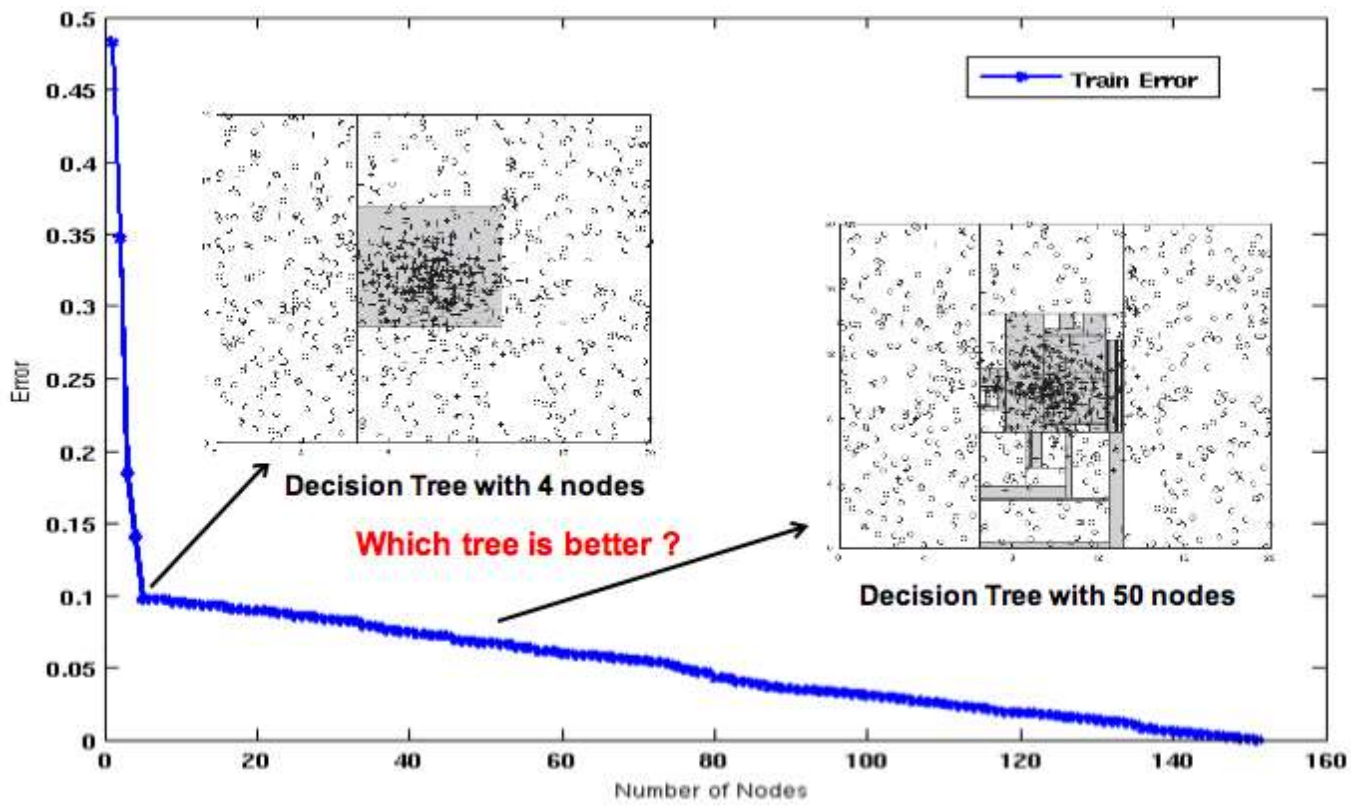
we don't have to do feature standardization because here we care about only less than or greater than. It is not distance base method.

Note: when you have categorical variable like Pincode/zipcode. dataset is very high because all are different. it all depend upon sort value where each sort value is the potential threshold. in that case, we convert categorical variables to numerical variables. By converting into numerical variables, get rid of data sparsity. here each numerical divided into two-part while other cases lots of variables can be useless.

(No normalization, no standardization require)

Overfitting and Underfitting

as depth increases, the probability of very few as pure point increases. so there is chance of overfitting. if depth increases, interpretability decreases. So **right depth obtain using cross-validation**.



Underfit has very few hypercubes where in overfit has many numbers of hypercubes which result in very less point as a pure node in hypercubes i.e overfit. hypercube is a cubic box.

Train and Run time complexity

if you have a **large dataset**, a Decision tree may not be a good option. Decision tree good in the large dataset but less dimension. it also good for low latency requirements.

In the best case of a balanced tree, the depth would be in $O(\log N)$O(logN), but the decision tree does locally optimal splits without caring much about balance. This means that the worst case of depth being in $O(N)$O(N) is possible — basically when each split simply splits data in 1 and n-1 examples, where n is the number of examples of the current node.

Regression using Decision Trees

Instead of using IG, use MSE. Calculate MSE at each node, take waited sum of MSE in the next layer, whichever decreases MSE will be chosen.

Cases:

Imbalance dataset: this impact entropy/MSE calculation. So balance it

Large d: not good, avoid one-hot encoding. if you have categorical variable with lot of feature. convert them into numerica variable.

Multiclass classification: you don't have to do one verse rest(OVR). entropy already consider multi class in calculation.

Decision surface: Non-linear, axis parallel hyper-cube.

Feature interaction: added inbuild in DT like $f1*f2/f1^2$

Outlier: impact the tree and create an unstable tree

Interpretability: super interpretable when depth is not large

Feature importance: calculated If feature occurs a lot and because of it IG increase a lot.

For visualization:

```
import graphviz
dot_data = tree.export_graphviz(clf, out_file=None)
graph = graphviz.Source(dot_data)
graph.render("iris")
```

Reference:

Google image

Applied AI

1.10. Decision Trees - scikit-learn 0.21.3 documentation

Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression . The goal...

scikit-learn.org

sklearn.tree.DecisionTreeClassifier - scikit-learn 0.21.3 documentation

class sklearn.tree. DecisionTreeClassifier(criterion='gini', splitter='best', max_depth=None, min_samples_split=2...

scikit-learn.org

#http://homepage.cs.uri.edu/faculty/hamel/courses/2016/spring2016/csc581/lecture-notes/32-decision-trees.pdf