# Decision Trees

Tuesday, 11 April 2023 12:05 PM

## What is a decision tree?

A graphical model, often termed as flowchart, that divides the feature space into partitions. A model is a connected acyclic graph.

In graphical form, decision tree 1) is interpretable, 2) have sufficient complex decision boundaries, and 3) boundaries are simple to describe mathematically.

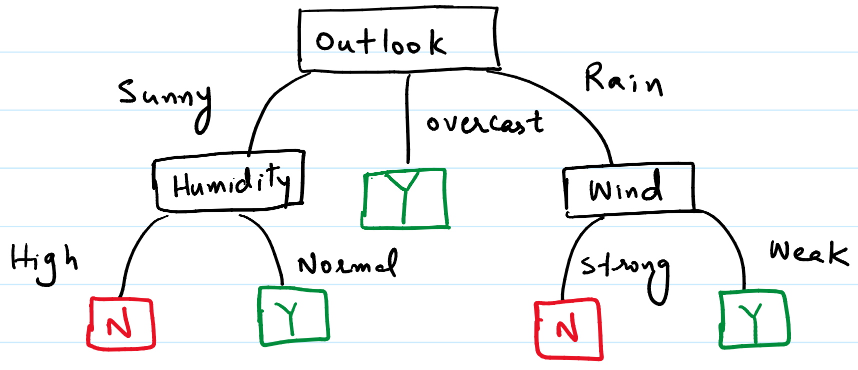
Let's look at a couple of examples before we formally define a decision tree.

### Example 1:

*A decision tree to classify Play Tennis (Y | N)

Features: Outlook
Humidity
Wind.

* Decision tree carries out a bunch of comparisons before it makes a decision (Yes/No).

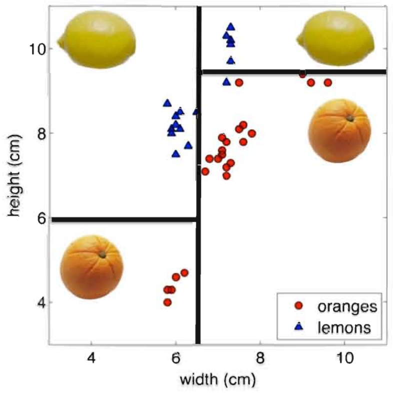


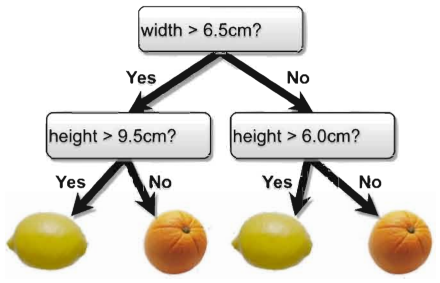
### Example 2:

* Lemon vs orange classification

* Lemon vs orange classification

* Features : width and height





## Decision Tree Structure:

* Decision tree is a graph (connected and acyclic).
* Output of a model is based on a series of comparisons of the values of the features against reference (threshold for each feature).

* Three types of nodes: Root Node, Internal Nodes, Terminal Leaf Nodes

* Root Node is the top node that branches into two (or multiple) nodes.

* Branching / splitting to the next-level is determined by the value of feature

e.g. height > 9.5 in example 2

feature reference value (threshold)

* Terminal Leaf Nodes represent class assignment.

* Example 2 reveals that each splitting corresponds to

corresponds to

partition of the feature space by "axis- aligned hyperplanes". { This makes decision tree interpretable

* Each comparison/splitting partitions along a single feature.

## Constructing Decision Tree (Learning the Model from Data)

Given $D = \{(x_i, y_i)\}_{i=1}^{n}$, learning decision tree means

* determining "optimal" partition of feature space with axis aligned linear boundaries

* class label (classification) or value (regression) is assigned to each region

How? Classification; Assigned label is a label of majority class of points in a region

Regression; Assigned value is an average of values ($y_i$) of the points in a region.

Even for simple definitions of optimal, learning the smallest decision tree is computationally infeasible (NP hard).

In practice, we employ greedy algorithm for learning the reasonable (not optimal and smallest) decision tree.

IDEA:

* For each node, starting from Root Node, choose the "optimal feature" Which to split!

At each node, starting from root node,
choose the "optimal feature" Which to split!
and "optimal threshold" Where to split!

**Questions:**

Q1: How to specify the feature test condition?
Q2: How to determine the best split? (Splitting Criterion)
Q3: When to stop splitting? (Stopping Condition)

**Q1:**

* Depends on feature type
  - Nominal
  - Ordinal
  - Continuous

* Depends on the number of splits
  - Binary
  - Multiway split

In example 1; outlook (Multiway)
humidity (2-way)

**Splitting Criterion:**

Key idea; Splitting that enables homogeneous (pure) class distribution are preferred.

Why? Improves classification accuracy

In other words, points in each partition after splitting should belong to one class (Ideal)

**Example**

Consider 2 partitions with the following points for a binary classification problem.

for a binary classification problem.

| | |
|---|---|
| Class 0: | 6 |
| Class 1: | 6 |

Non-homogeneous

(high degree of impurity)

| | |
|---|---|
| Class 0 : | 11 |
| Class 1 ; | 1 |

Homogeneous

(low degree of impurity)

## Measures of Node Impurity :

- GINI Index
- Entropy
- Misclassification error.

Before we define these measures, let's define a notation for region and partitioning for classification

* Region $R$ with $n$ data points, $D = \{(x_i, y_i)\}_{i=1}^{n}$

* Partition results in two regions $R_1$ and $R_2$ containing $n_1$ and $n_2$ data points.

* M-class classification problem; $y_i \in \{0, 1, \dots, M-1\}$

* We split along $j$-th feature using $t_j$ (threshold)

Example $M = 2$

| | Class 0 | Class 1 | GINI | Entropy | Misclassification error |
|---|---|---|---|---|---|
| $R_1$ | 0 | 6 | | | |
| [illegible] | 5 | 8 | | | |

|  |  |  |  |  |  |
|---|---|---|---|---|---|
| $R_1$ | 0 | 6 |  |  |  |
| $R_2$ | 5 | 8 |  |  |  |
|  | less impure | Highly impure |  |  |  |

## GINI Index

- Measures impurity of each region

$$GINI(\underbrace{i}_{\text{i-th region}} \mid \underbrace{j}_{\text{feature}}, \underbrace{t_j}_{\text{threshold}}) = 1 - \sum_{k=0}^{M-1} P(k \mid R_i)^2$$

Here $P(k \mid R_i)$ quantifies proportion of points in $R_i$ region with label $k$.

Example $M=2$

|  | Class 0 | Class 1 | GINI |
|---|---|---|---|
| $R_1$ | 0 | 6 | $1-\left[\left(\frac{6}{6}\right)^2+\left(\frac{0}{6}\right)^2\right]=0$ |
| $R_2$ | 5 | 8 | $1-\left[\left(\frac{5}{13}\right)^2+\left(\frac{8}{13}\right)^2\right]=\frac{80}{169}$ |

## ENTROPY:

Entropy is used to analyze the distribution in information theory.

Mathematically; entropy of random variable $X$ with pdf $p(x)$ is given by

$$H(X) = -\sum_{x} p(x) \log p(x)$$

For a flat distribution; entropy is high

For a peaky distribution; entropy is low

Since we require region after partitioning to be pure. In other words, we prefer a peaky distribution of $p(k|R_i)$.

Noting this, we define entropy for region $i$ as

$$\text{Entropy}(i \mid j, t_j) = -\sum_{k=0}^{M-1} p(k|R_i) \log p(k|R_i)$$

Example $M=2$

| | Class 0 | Class 1 | Entropy |
|---|---|---|---|
| $R_1$ | 0 | 6 | $-\frac{6}{6}\log\frac{6}{6} - \frac{0}{6}\log\frac{0}{6} = 0$ |
| $R_2$ | 5 | 8 | $-\frac{5}{13}\log\frac{5}{13} - \frac{8}{13}\log\frac{8}{13} = 1.38$ |

Misclassification error :

$$\text{Error} = 1 - \max_{k=0,\dots,M-1} p(k|R_i)$$

Example $M=2$

| | $R_1$ | $R_2$ | Misclassification error |
|---|---|---|---|
| Class 0 | 0 | 6 | $1 - \max\left(\frac{0}{6}, \frac{6}{6}\right) = 0$ |
| Class 1 | 5 | 8 | $1 - \max\left(\frac{5}{13}, \frac{8}{13}\right) = \frac{5}{13}$ |
| | less impure | Highly impure | |

Now, we have defined these impurity measures given $j$ (which to split) and $t_j$ (where to split).

* We find $j$ and $t_j$ such that impurity is minimized, i.e., for binary classification problem:

$$\underset{j,\, t_j}{\text{minimize}} \quad \frac{n_1}{n} m(1 \mid j, t_j) + \frac{n_2}{n} m(2 \mid j, t_j)$$

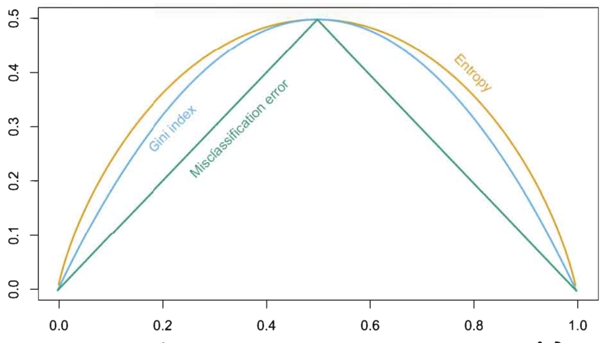
where $m(i \mid j, t_j)$ is any of the impurity measures.

## Comparison of different measures:

For binary classification problem, we plot these impurity measures against different values of purity

* Entropy penalizes impurity the most.

* Misclassification rate is not good at pushing for really pure nodes.



purity (Proportion of points).

## Splitting for Regression:

* Impurity measure is used for splitting for classification task since we assign majority class label to the region

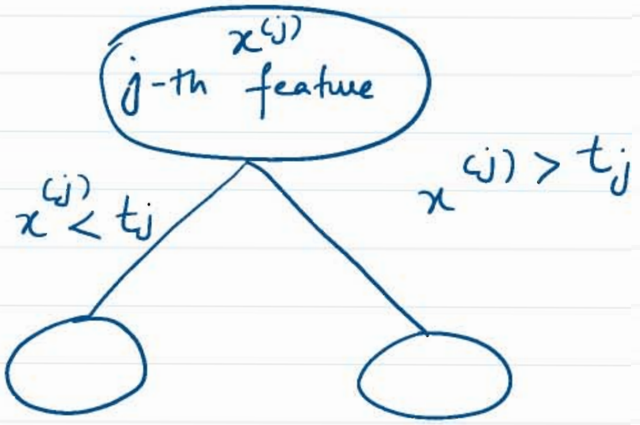
* For regression, splitting is carried out such that improves the predictive accuracy of the model as measured by for example MSE or Variance

improves the predictive accuracy of the model as measured by, for example, MSE or Variance.

e.g.,

* splitting along $x^{(j)}$ at threshold $t_j$

* $j$ and $t_j$ are choosen such that MSE is minimized if we use resulting tree for prediction.



Stopping Condition:

* In the absence of stopping condition, tree keeps on growing until all the leaves contain only one point. This yields over-fitted complex model

Some commonly used stopping conditions include:

- Stop splitting if a region is 100% pure.
- Stop splitting if no. of points < pre-defined value
- Stop splitting if total number of leaves > Pre-defined value
- Stop splitting if impurity measure < Pre-defined value

$\Big\}$ Hyper-parameters

- More restrictive stopping condition evaluates gain in purity or decrease in entropy due to splitting.

Stop splitting if gain for a region is greater than pre-defined value. We define gain as follows.

than pre-defined value. We define gain as follows.

$$\text{Gain}(R) = m(R) - \frac{n_1}{n} m(R_1) - \frac{n_2}{n} m(R_2)$$

Here, $m(\cdot)$ is any of the impurity measures or MSE.
(classification) (regression)

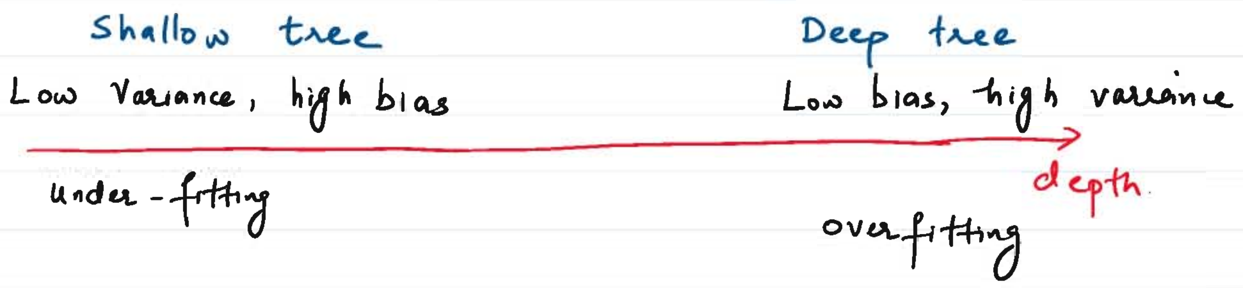
- For regression, stop splitting if MSE < pre-defined value.

## Overfitting in Decision Trees :

* A shollow tree (smaller depth) underfits; unable to model a complex decision boundary

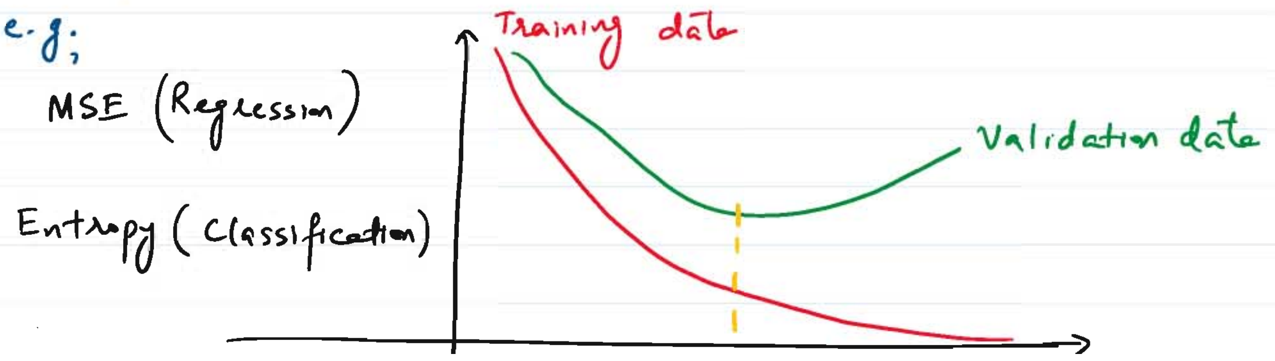
* When the tree is too deep, it overfits because it divides the region into too many regions.

* In other words, we can say that



* We can use validation data to determine the depth of the tree.

e.g;



* Another method to counter overfitting due to large depth of tree is 'Pruning'.

## Pruning

Idea: Instead of preventing a complex tree to grow by limiting the depth of the tree, pruning allows us to obtain a simple tree from the complex one by removing some of the nodes.

* Among many methods to carry out pruning, 'cost complexity pruning' is commonly used.

In this method, we select a subtree for which a balance between the performance (error) and size is optimized, that is, we define a cost-complexity measure as

$$C(T) = \text{error}(T) + \alpha\,|T|$$

error for choosing a subtree T (entropy/MSE)

Number of leaves of subtree T

Trade-off between accuracy and size of tree.

* While carrying out cost-complexity pruning, we follow the following steps for fix $\alpha$.

* While carrying out cost-complexity pruning, we follow the following steps for fix $\alpha$.

- Start with a full-tree (each leaf is pure)
- Among different subtrees, we choose a subtree for which the cost-complexity $C(T)$ is minimized, at each depth level, starting from the leaf node and moving up towards the root.

Bagging, Random Forests, and Boosting

Thursday, 13 April 2023 4:38 AM

## Decision Tree Models – Limitations:

* Since decision tree models split feature space using axis-aligned partitions at each node, we need a large (deep) tree for capturing complex decision boundary.

* Consequently, decision tree model will have high variance (due to overfitting)

* Due to this limitation, decision tree models may underperform, in practice, compared to other regression or classification models.

* One obvious way to address high variance in decision trees, or in any other model as well, is to obtain multiple models yielding multiple outputs which can be combined to obtain a single output.

  This method is known as Bootstrap Aggregating (Bagging)

## BAGGING:

Broadly; bagging can be used to any high variance model.
It is comprised of

Bootstrap: We generate multiple samples (sets) of the data using bootstrapping (sampling with replacement). We train a complex model (deep decision tree) for each training set.

Aggregate: We obtain multiple outputs for a given input using multiple models and then aggregate these outputs as follows:

Classification: majority label

Regression ; (weighted) average

Classification: majority label
Regression ; (weighted) average
to obtain a single output.

Significance:

*Aggregation enables the reduction in the variance of the overall prediction since output is an average of multiple high variance models.

* Using a deep tree allows us to model complex decision boundary

Bagging belongs to the broad class of ensemble methods in which we build a single model by training multiple models and aggregating the outputs from the ensemble of models.

Out-of-Bag Error:

To evaluate the performance of ensemble methods such as bagging, we use a metric known as out-of-bag (OOB) error.

If we have an ensemble of B models (e.g., trees), we obtain aggregated output $\tilde{y}_i$ for input $x_i$ as

Regression: $\tilde{y}_i = \frac{1}{B} \sum_{j \in B} \tilde{y}_{i,j}$

Classification: $\tilde{y}_i = \text{majority}(\tilde{y}_{i,j})$

Here $\tilde{y}_{i,j}$ is the output of the j-th model for input $x_i$

Note: B models (trained using subsets of training data) didn't see $(x_i, y_i)$ training point. It means each of the B models here is not trained on $x_i$.

In other words we use all B models that were

B models here is not trained on $x_i$.
In other words, we use all B models that were not trained for $(x_i, y_i)$ during aggregation.

Using aggregated output, we can evaluate OOB error (loss) as:

Regression: $$\mathcal{L}_{OOB} = \frac{1}{n}\sum_{i=1}^{n}\left(\tilde{y}_i - y_i\right)^2 \quad \text{MSE}$$

Classification $$\mathcal{L}_{OOB} = \frac{1}{n}\sum_{i=1}^{n}\left(1 - \delta_{y_i, \tilde{y}_i}\right) \quad 1/0 \text{ Loss}$$

## Correlation between models in Bagging:

In practice, we encounter correlation between multiple models in an ensemble.

Why?
* Recall the construction of decision trees!
* If one of the features is very strong in the training set (e.g., in terms of entropy), we expect different decision trees to split across the same feature in the early iterations.
* Consequently, we will have correlation between different trees in an ensemble.

Solution: Random Forests.

RANDOM FORESTS: Improved variant of bagging that creates (or attempts to create) an ensemble of independent decision trees.

How?
- Train each tree on a different boot-strap sample (similar to bagging)

(This enables decorrelation
- For each tree and for each split, we randomly select a set of k out of d features

(This enables decorrelation between trees) - For each tree and for each split, we randomly select a set of k out of d features.

- We use this set of k features to carry out splitting and determining threshold.

## Hyperparameters for Random Forests:

* Total no. of trees
* 'k'; no. of features to be used for splitting
* depth or number of leaves.

Typically; we use $k \simeq \sqrt{d}$ (Classification) and $k \approx \frac{d}{3}$ (regression).

## Issue with Random Forests:

If only a small number of features are relevant, Random Forests perform poorly.

Why? Since we are randomly choosing k out of d features, there is a high probability of splitting using irrelevant features. Consequently, we will have weak models in our ensemble.

## BOOSTING:

Key Idea: Take an ensemble of simple models and combine them to form a single more complex model.

Motivation: Complex decision trees (larger depth) — High variance
Simple decision trees (smaller depth) — High bias.

- We combined multiple complex decision trees to reduce variance of the combined model in Bagging.

- Instead, we can take an ensemble of simple

- Instead, we can take an ensemble of simple trees and combine them such that the bias is reduced.

Formulation: * Ensemble of simple models (trees) : $\{T_i\}$

* Additively combine the simple models to obtain

$$\sum_i \alpha_i T_i = T \quad \text{(complex model)}$$

## Gradient Boosting Algorithm:

An algorithm that iteratively adds a simple model to form a complex model such that a new addition to the ensemble compensates for the weaknesses for the existing ensemble.

Algorithm (Regression)

01 * Train a simple model $T_0$ on the training data $D = \{(x_i, y_i)\}_{i=1}^{n}$ and set $\underset{\text{ensemble we want to find}}{T} = T_0$

02 * Determine residual, $r_i = y_i - T(x_i)$

Repeat for $k = 1, 2, \ldots$ until stopping criterion

03 * Train a simple model $T_k$ on $\{(x_i, r_i)\}_{i=1}^{n}$

- This model $T_k$ attempts to compensate for the residual that could not be modeled by $T$
- $T_k$ compensates for weaknesses of $T$.

04 * Update ensemble $T \leftarrow T + \alpha T_k$