From tree to forest

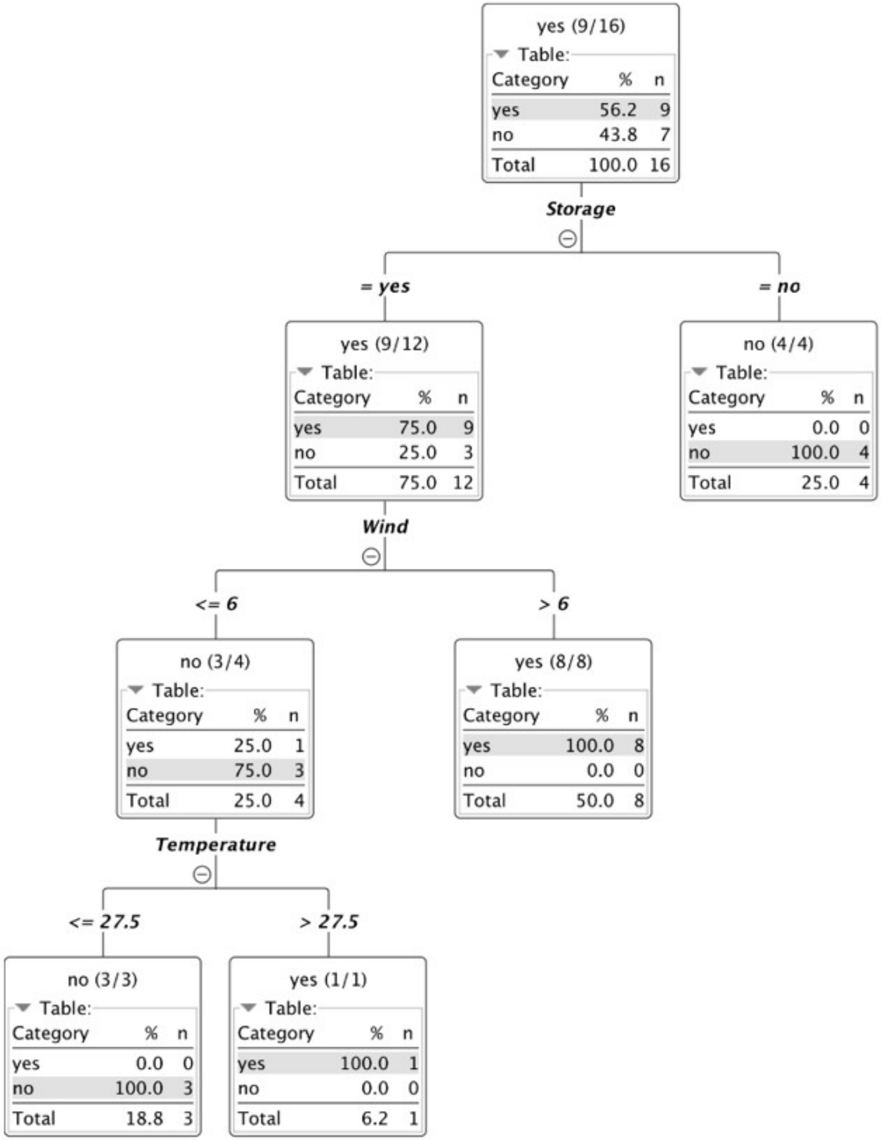
Decision tree

- Decision trees are very popular supervised classification algorithms:
  - They perform quite well on classification problems
  - The decision path is relatively easy to interpret
  - The algorithm to build (train) them is fast and simple

- A decision tree is a flowchart-like structure made of nodes and branches:
  - At each node, a split on the data is performed based on one of the input features, generating two or more branches.
  - More and more splits are made in the upcoming nodes to partition the original data.
  - This continues until a node is generated where all or almost all of the data belong to the same class.

Example: Sailing plan

Outlook	Wind	Temp	Storage	Sailing
sunny	3	30	yes	yes
sunny	3	25	yes	no
rain	12	15	yes	yes
overcast	15	2	no	no
rain	16	25	yes	yes
sunny	14	18	yes	yes
rain	3	5	no	no
sunny	9	20	yes	yes
overcast	14	5	no	no
sunny	1	7	no	no
rain	4	25	yes	no
rain	14	24	yes	yes
sunny	11	20	yes	yes
sunny	2	18	yes	no
overcast	8	22	yes	yes
overcast	13	24	yes	yes



Building a decision tree

- There are several automatic procedures (like **C4.5**, **ID3** or the **CART** algorithm) to extract the rules from the data to build a decision tree.

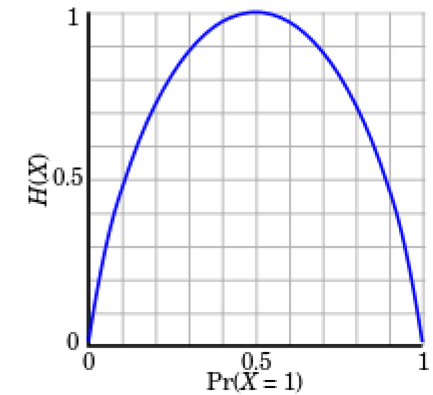
- These algorithms partition the training set into subsets until each partition is either “pure” in terms of target class or sufficiently small:
  - A pure subset is a subset that contains only samples of one class.
  - Each partitioning operation is implemented by a rule that splits the incoming data based on the values of one of the input features.

Split rules

- How does an algorithm decide which feature to use at each point to split the input subset?
  - At each step, the algorithm uses the feature that leads to the purest output subsets.
  - Therefore, we need a metric to measure the purity of a split:
    - Information gain
    - Gini index
    - Gain ration

Entropy

- Entropy is used to measure purity, information, or disorder:

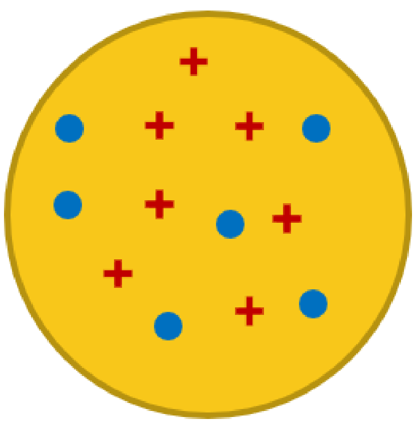


$$Entropy\ (p) = -\sum_{i=1}^{N} p_i \, log_2 \, p_i$$

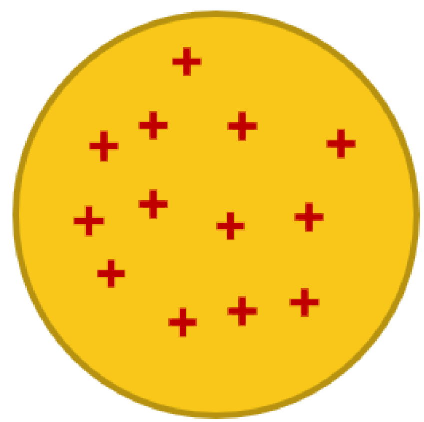
where p is the whole dataset, N is the number of classes, and p_i is the frequency of class *i* in the same dataset.

$p_1 = {}^{7}/_{13}$

$p_2 = {}^{6}/_{13}$



$Entropy\ (p) = \left({}^{7}/_{13} \log_2({}^{7}/_{13}) + {}^{6}/_{13} \log_2({}^{6}/_{13})\right)$
$= 0{,}995$



$p_1 = {}^{13}/_{13} = 1$

$p_2 = {}^{0}/_{13} = 0$

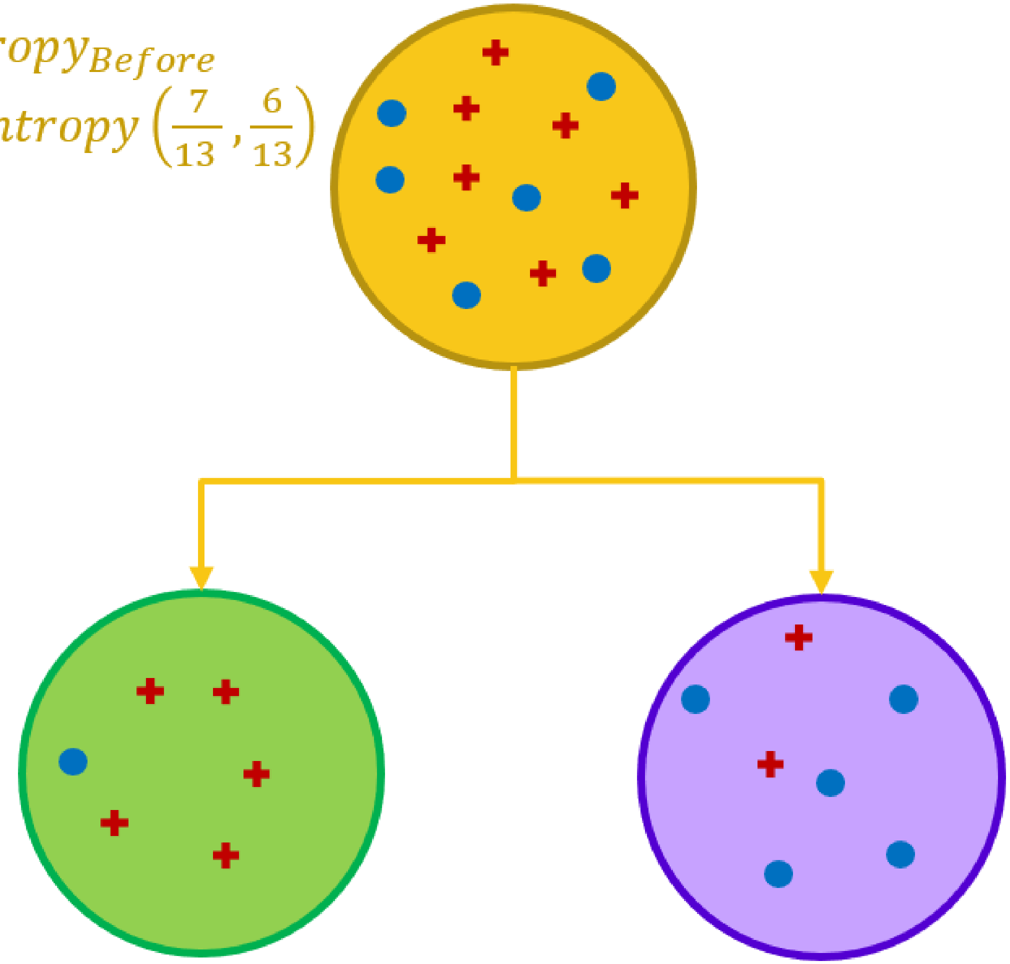
$Entropy\ (p) = \left({}^{13}/_{13} \log_2({}^{13}/_{13}) + {}^{0}/_{13} \log_2({}^{0}/_{13})\right)$
$= 0$

Entropy based splits

The goal of each split in a decision tree is to move from a confused dataset to two (or more) purer subsets with lesser entropys.

Ideally, the split should lead to subsets with an entropy of 0.0.

$$Entropy_{Before} = Entropy\left(\frac{7}{13}, \frac{6}{13}\right)$$



$$Entropy_1 = Entropy\left(\frac{5}{6}, \frac{1}{6}\right)$$

$$w_1 = {}^{6}/_{13}$$

$$Entropy_2 = Entropy\left(\frac{2}{7}, \frac{5}{7}\right)$$

$$w_2 = {}^{7}/_{13}$$

Information Gain (ID3)

- In order to evaluate *how good a feature is for splitting*, the difference in entropy before and after the split is calculated:

$$Information\ Gain\ =\ Entropy(before) - \sum_{j=1}^{K} Entropy(j,\ after)$$

where “before” is the dataset before the split,
K is the number of subsets generated by the split,
(j, after) is subset j after the split.

- We choose to split the data on the feature with the highest value in information gain.

Gain Ratio (C4.5)

The information gained by a balanced split is higher than the information gained by an unbalanced split.

$$Gain\ Ratio = \frac{Information\ Gain}{SplitInfo} = \frac{Entropy\ (before) - \sum_{j=1}^{K} Entropy(j, after)}{\sum_{j=1}^{K} w_j \log_2 w_j}$$

$$w_j = \frac{\#\ samples\ in\ subset\ (j, after)}{\#\ samples\ in\ dataset\ (before)}$$

Gini index

- Gini impurity is a measure of how often a randomly chosen element from the set would be incorrectly labeled:

$$Gini\ Impurity\ (p) = 1 - \sum_{i=1}^{N} p_i^2$$

Proof:

$$\mathrm{I}_G(p) = \sum_{i=1}^{J} p_i \sum_{k \neq i} p_k = \sum_{i=1}^{J} p_i (1 - p_i) = \sum_{i=1}^{J} (p_i - {p_i}^2) = \sum_{i=1}^{J} p_i - \sum_{i=1}^{J} {p_i}^2 = 1 - \sum_{i=1}^{J} {p_i}^2$$

- The Gini index:

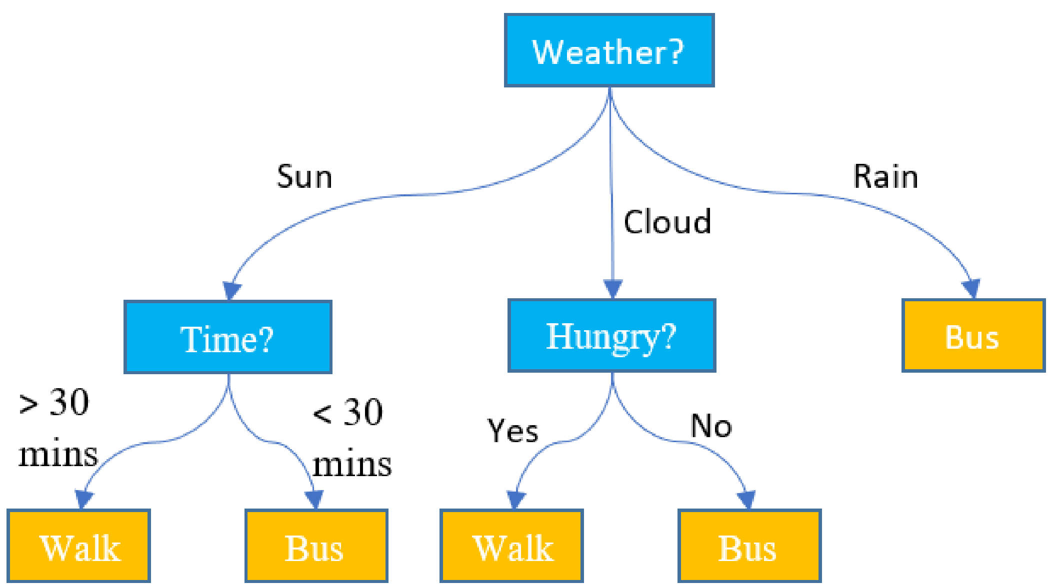
$$Gini\ Index = \sum_{j=1}^{K} w_j\ Gini\ Impurity\ (j,\ after)$$

$$w_j = \frac{\#\ data\ in\ in\ subset\ (j, after)}{\#\ data\ in\ dataset\ (before)}$$

where K is the number of subsets generated by the split and (j, after) is subset j after the split.

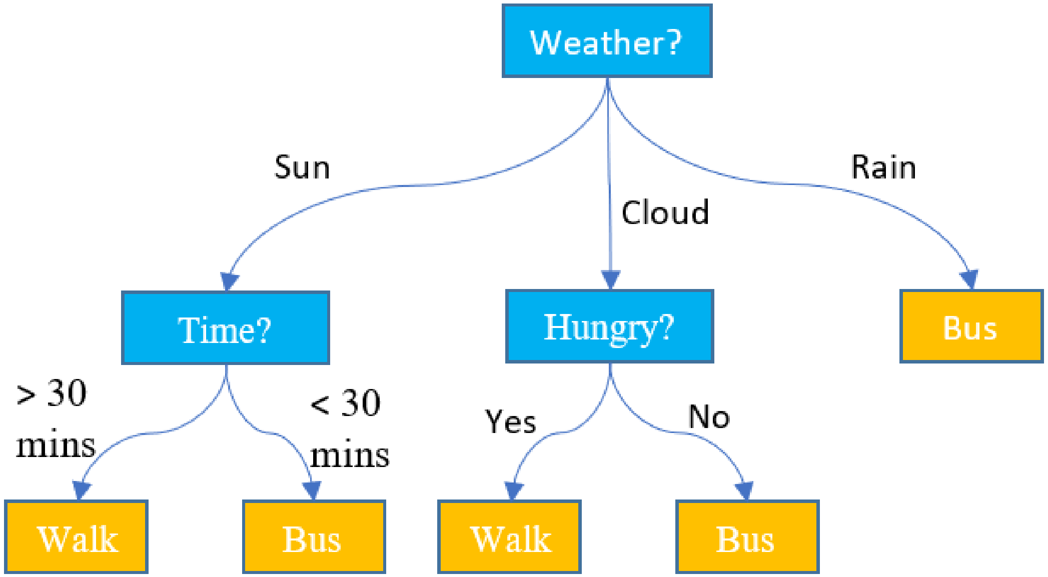
Split candidates

- Nominal features:
  - We can create a child node for each possible value (a wider tree)
  - we can make a binary split (a higher tree)



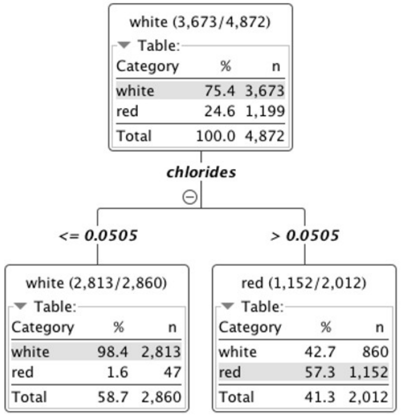
Split candidates

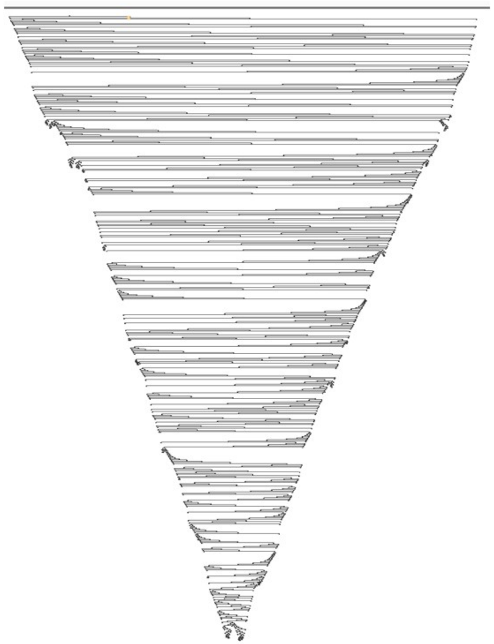
- Numerical features:
  - All numerical values could actually be split candidates (computationally expensive).
  - The candidate split points are taken in between every two consecutive values of the selected numerical feature. the binary split producing the best quality measure is adopted.



Size and Overfitting

- Trees that are too deep can lead to models that are too detailed and don't generalize on new data.
- On the other hand, trees that are too shallow might lead to overly simple models that can't fit the data.





Pruning

- Pruning is a way to avoid overfitting.
- Pruning is applied to a decision tree after the training phase.

- Basically, we let the tree be free to grow as much as allowed by its settings, without applying any explicit restrictions. At the end, we proceed to cut those branches that are not populated sufficiently

reduced error pruning

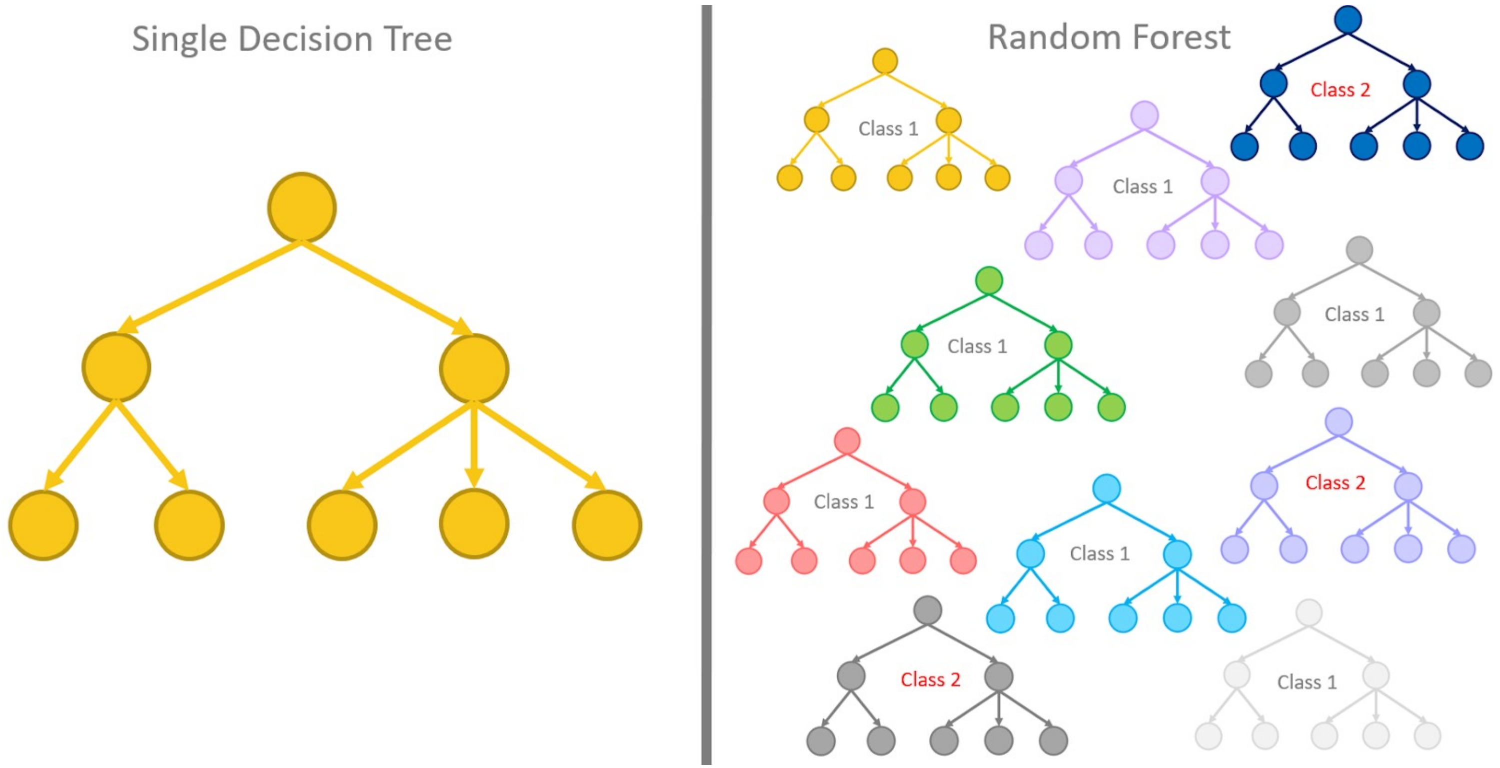
- At each iteration,
  - a low populated branch is pruned
  - The tree is applied again to the training data.
  - If the pruning of the branch doesn’t decrease the accuracy on the training set, the branch is removed.

Early Stopping

- Another option to avoid overfitting is early stopping, based on a stopping criterion.
- One common stopping criterion is the minimum number of samples per node.
  - A higher value of this minimum number leads to shallower trees
  - While a smaller value leads to deeper trees
- What other criteria?

Random forest

- Many is better than one.
  - Several decision trees together can produce more accurate predictions than just one single decision tree by itself.
- Random forest algorithm builds N slightly differently trained decision trees and merges them together to get more accurate and stable predictions.



Bootstrapping of Training Sets

In a random forest, N decision trees are trained each on a subset of the original training set obtained via bootstrapping of the original dataset (random sampling with replacement.)

Original Training Set

Col1	Col2	Col3	Col4	Col5	Col6
1	Sdf	200	A	1	.88
3	Fg	200	A	1	.67
2	Wdv	290	A	1	.36
4	Gh	345	B	0	.85
1	J	125	AB	0	.72
3	Xcv	543	B	0	.93
2	gbn	367	A	1	.18

Training Subsets via Bootstrapping

Col1	Col2	Col4	Col5	Col6
1	Sdf	A	1	.88
3	Fg	A	1	.67

Col2	Col3	Col4	Col5	Col6
Wdv	290	A	1	.36
Gh	345	B	0	.85

Col1	Col2	Col3	Col5	Col6
3	Fg	200	1	.67
2	Wdv	290	1	.36

Col1	Col2	Col3	Col4	Col6
1	Sdf	200	A	.88
3	Fg	200	A	.67

Col2	Col3	Col4	Col5	Col6
Sdf	200	A	1	.88
Wdv	290	A	1	.36
J	125	AB	0	.72
Xcv	543	B	0	.93

Col1	Col3	Col4	Col5	Col6
1	200	A	1	.88
3	200	A	1	.67

Col1	Col2	Col3	Col4	Col5
1	Sdf	200	A	1
2	Wdv	290	A	1

Col1	Col2	Col3	Col4	Col6
1	Sdf	200	A	.88
3	Fg	200	A	.67

Col1	Col2	Col3	Col4	Col5
3	Fg	200	A	1
2	Wdv	290	A	1

Col2	Col3	Col4	Col5	Col6
Sdf	200	A	1	.88
Fg	200	A	1	.67
J	125	AB	0	.72
Xcv	543	B	0	.93

The Majority Rule

- The N slightly differently trained trees will produce N slightly different predictions for the same input vector.
- Usually, the majority rule is applied to make the final decision.
- The prediction offered by the majority of the N trees is adopted as the final one.
- While the predictions from a single tree are highly sensitive to noise in the training set, predictions from the majority of many trees are not (if trees are diverse).