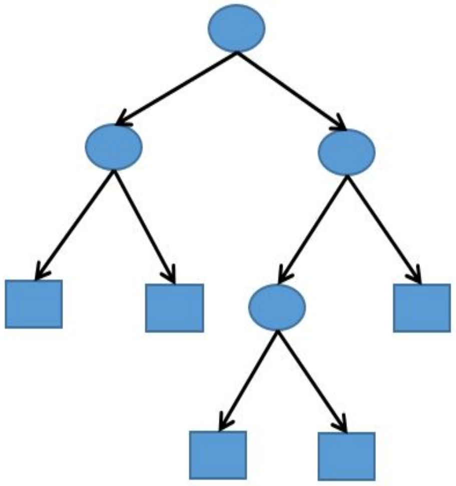
Introduction to Machine Learning

Part 3: Decision Trees

Zengchang Qin (Ph.D.)

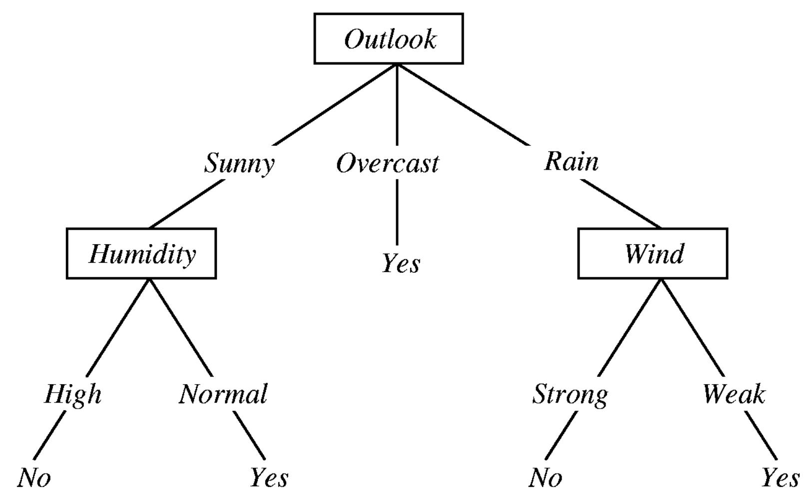


Decision Tree Learning

Play-Tennis Problem

The Play-Tennis data from T. Mitchell's book [3], we can find a tree to represent "Yes" and "No" by leaves.

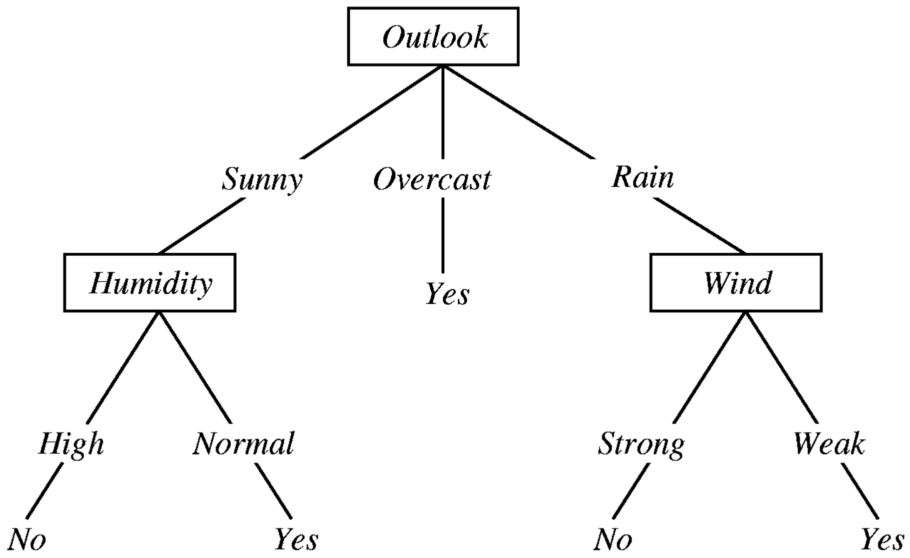
Day	*Outlook*	*Temperature*	*Humidity*	*Wind*	*PlayTennis*
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool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



[3] T. Mitchell (1997), Machine Learning, McGraw Hill.

Impurity

The Play-Tennis data from T. Mitchell's book [3], we can find a tree to represent "Yes" and "No" by leaves.



Greedy approach:

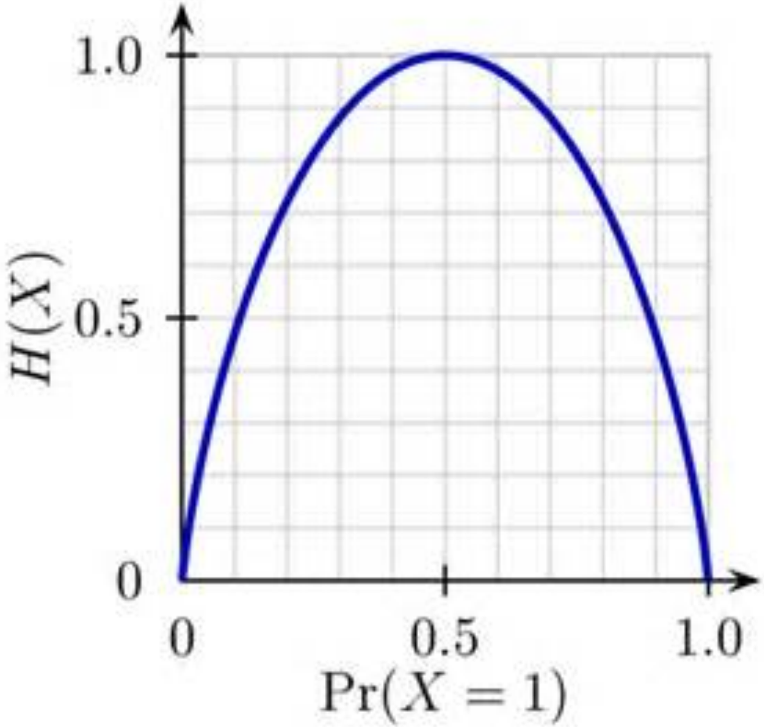
Nodes with homogeneous class distribution are preferred

Need a measure of node impurity

Multi-dimensional Attributes (Features)

Shannon's solution follows from the fundamental properties of information.

1. *I(p)* is anti-monotonic in *p* – increases and decreases in the probability of an event produce decreases and increases in information, respectively
2. *I(p) ≥ 0* – information is a non-negative quantity
3. *I(1) = 0* – events that always occur do not communicate information
4. *I(p1, p2) = I(p1) + I(p2)* – information due to independent events is additive



Information Gain

$$Gain(S, Wind) = Entropy(S) - \sum_{v \in \{Weak, Strong\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$= Entropy(S) - (8/14) Entropy(S_{Weak})$$

$$- (6/14) Entropy(S_{Strong})$$

$$= 0.940 - (8/14)0.811 - (6/14)1.00$$

$$= 0.048$$

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

$$Values(Wind) = Weak, Strong$$

$$S = [9+, 5-]$$

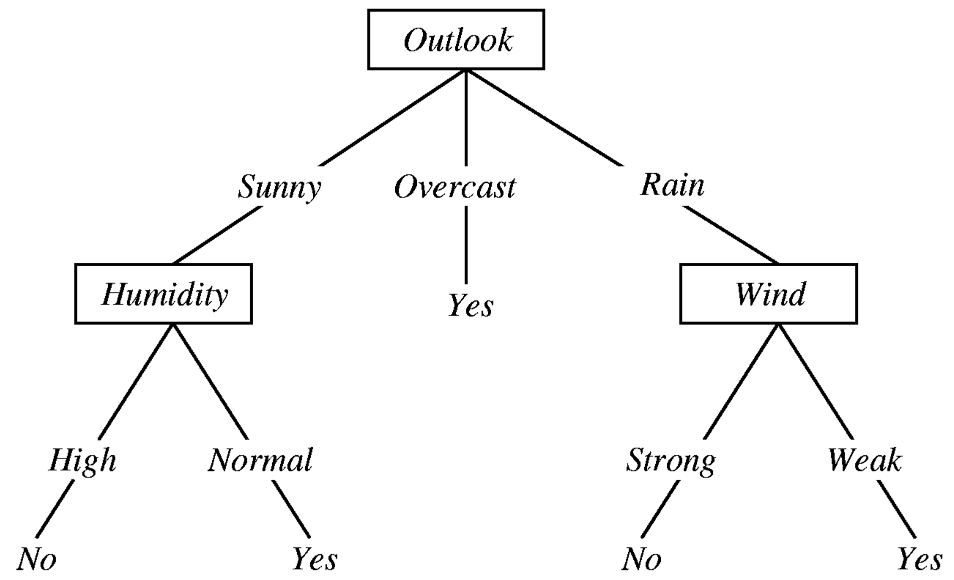
$$S_{Weak} \leftarrow [6+, 2-]$$

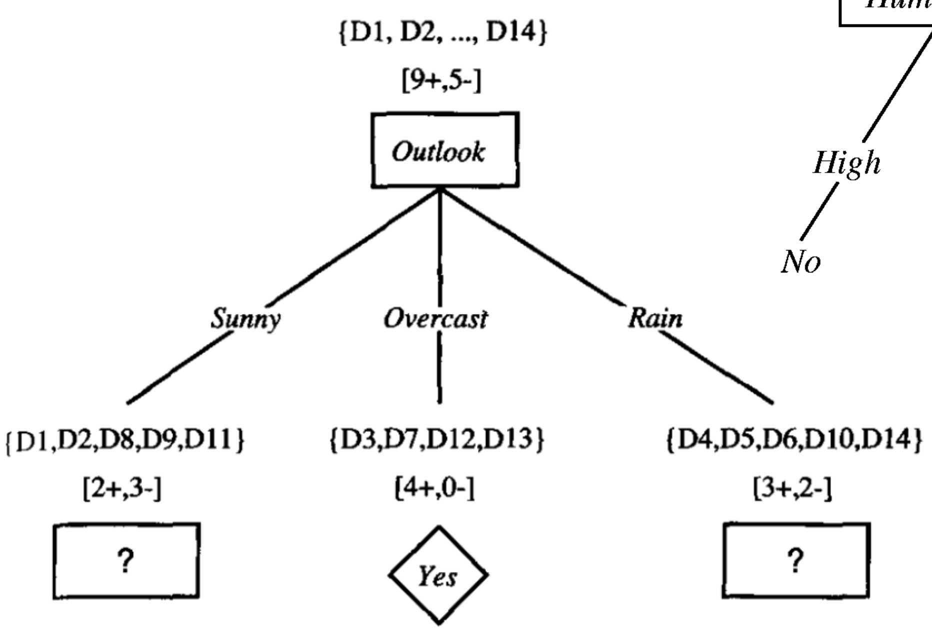
$$S_{Strong} \leftarrow [3+, 3-]$$

$$Gain(S, A) \equiv Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

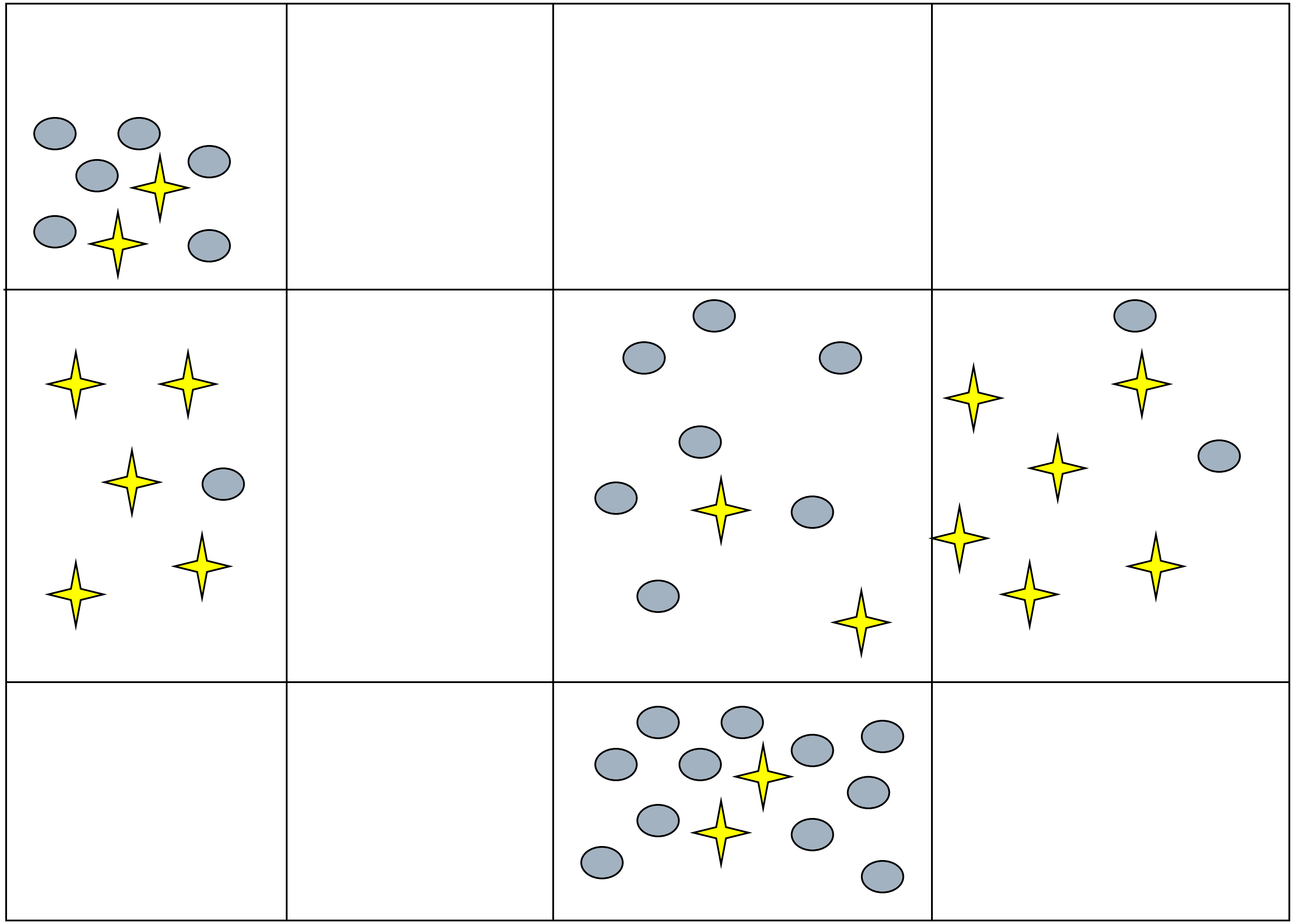
Sub-Trees

$$Gain(S, Outlook) = 0.246$$
$$Gain(S, Humidity) = 0.151$$
$$Gain(S, Wind) = 0.048$$
$$Gain(S, Temperature) = 0.029$$





Partition



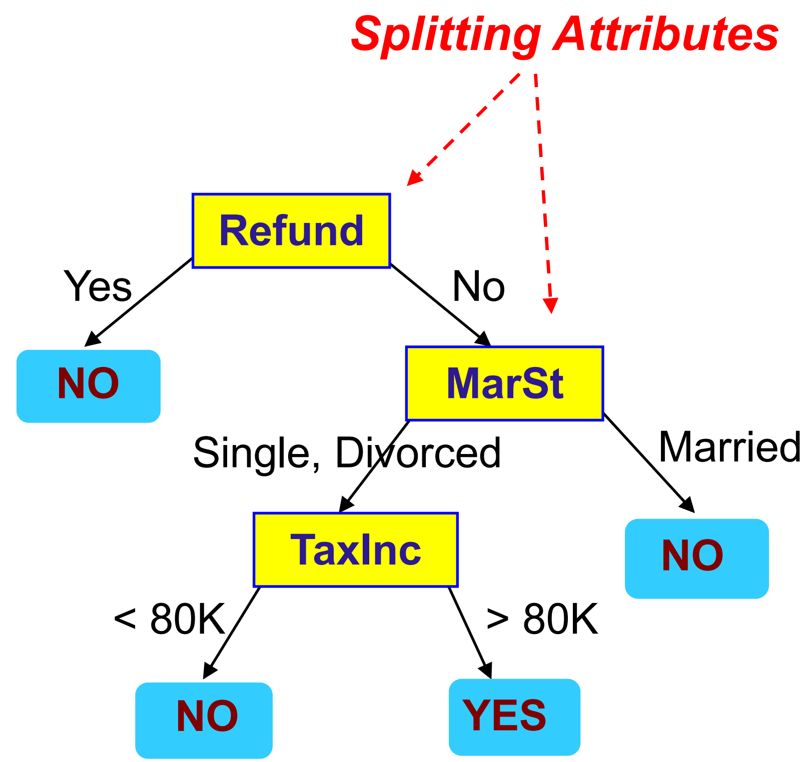
General Way of Building Trees

- Greedy strategy.
  - Split the records based on an attribute test that optimizes certain criterion.
- Issues
  - Determine how to split the records
    - How to specify the attribute test condition?
    - How to determine the best split?
  - Determine when to stop splitting

Attribute Types

	categorical	categorical	continuous	class
Tid	**Refund**	**Marital Status**	**Taxable Income**	**Cheat**
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

Training Data



Model: Decision Tree

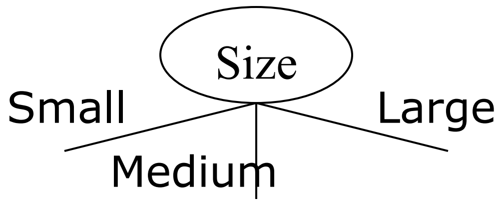
Sub-Trees

- Depends on attribute types
  - Nominal
  - Ordinal
  - Continuous

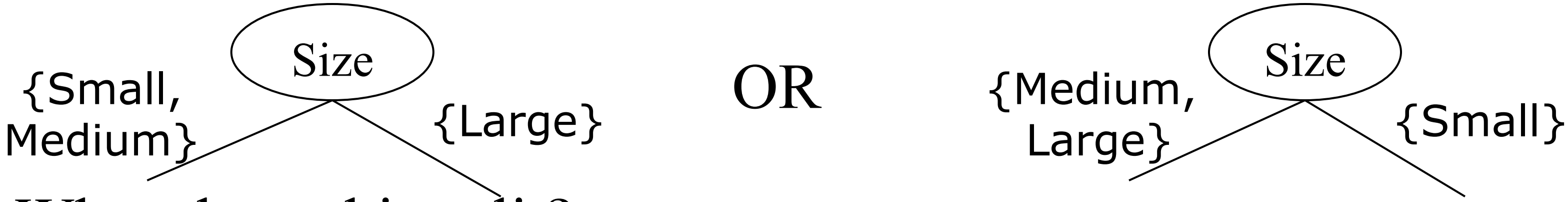
- Depends on number of ways to split
  - 2-way split
  - Multi-way split

Splitting

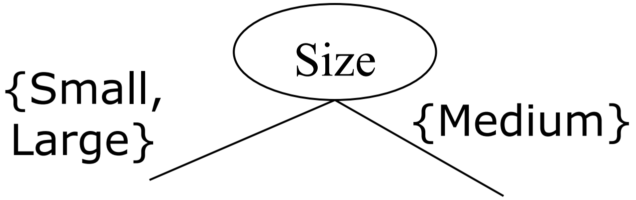
- Multi-way split: Use as many partitions as distinct values.



- Binary split: Divides values into two subsets.
  Need to find optimal partitioning.



- What about this split?



Discretization

- Different ways of handling
  - Discretization to form an ordinal categorical attribute
    - Static – discretize once at the beginning
    - Dynamic – ranges can be found by equal interval bucketing, equal frequency bucketing (percentiles), or clustering.
  - Binary Decision: $(A < v)$ or $(A \geq v)$
    - consider all possible splits and finds the best cut
    - can be more compute intensive

Gini Index

- Gini Index for a given node t :

$$GINI(t) = 1 - \sum_{j} [p(j \mid t)]^2$$

(NOTE: $p(j \mid t)$ is the relative frequency of class j at node t).

  - Maximum ($1 - 1/n_c$) when records are equally distributed among all classes, implying least interesting information
  - Minimum (0.0) when all records belong to one class, implying most interesting information

C1	**0**
C2	**6**
Gini=0.000	

C1	**1**
C2	**5**
Gini=0.278	

C1	**2**
C2	**4**
Gini=0.444	

C1	**3**
C2	**3**
Gini=0.500	

Detailed Calculation

$$GINI(t) = 1 - \sum_j [p(j \mid t)]^2$$

C1	**0**
C2	**6**

P(C1) = 0/6 = 0   P(C2) = 6/6 = 1

Gini = 1 – P(C1)^2 – P(C2)^2 = 1 – 0 – 1 = 0

C1	**1**
C2	**5**

P(C1) = 1/6   P(C2) = 5/6

Gini = 1 – (1/6)^2 – (5/6)^2 = 0.278

C1	**2**
C2	**4**

P(C1) = 2/6   P(C2) = 4/6

Gini = 1 – (2/6)^2 – (4/6)^2 = 0.444

Gini Split – Looks Familiar?

- Used in CART:
- When a node p is split into k partitions (children), the quality of split is computed as,

$$GINI_{split} = \sum_{i=1}^{k} \frac{n_i}{n} GINI(i)$$

where, n_i = number of records at child i,

n = number of records at node p.

Gini Split

- For efficient computation: for each attribute,
  - Sort the attribute on values
  - Linearly scan these values, each time updating the count matrix and computing gini index
  - Choose the split position that has the least gini index

Cheat		No		No		No		Yes		Yes		Yes		No		No		No		No		
	Taxable Income																					
Sorted Values		60		70		75		85		90		95		100		120		125		220		
Split Positions	55		65		72		80		87		92		97		110		122		172		230	
	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>
Yes	0	3	0	3	0	3	0	3	1	2	2	1	3	0	3	0	3	0	3	0	3	0
No	0	7	1	6	2	5	3	4	3	4	3	4	3	4	4	3	5	2	6	1	7	0
Gini	0.420		0.400		0.375		0.343		0.417		0.400		***0.300***		0.343		0.375		0.400		0.420	

Misclassification Error

$$Error(t) = 1 - \max_i P(i \mid t)$$

C1	**0**
C2	**6**

P(C1) = 0/6 = 0   P(C2) = 6/6 = 1

Error = 1 – max (0, 1) = 1 – 1 = 0

C1	**1**
C2	**5**

P(C1) = 1/6   P(C2) = 5/6

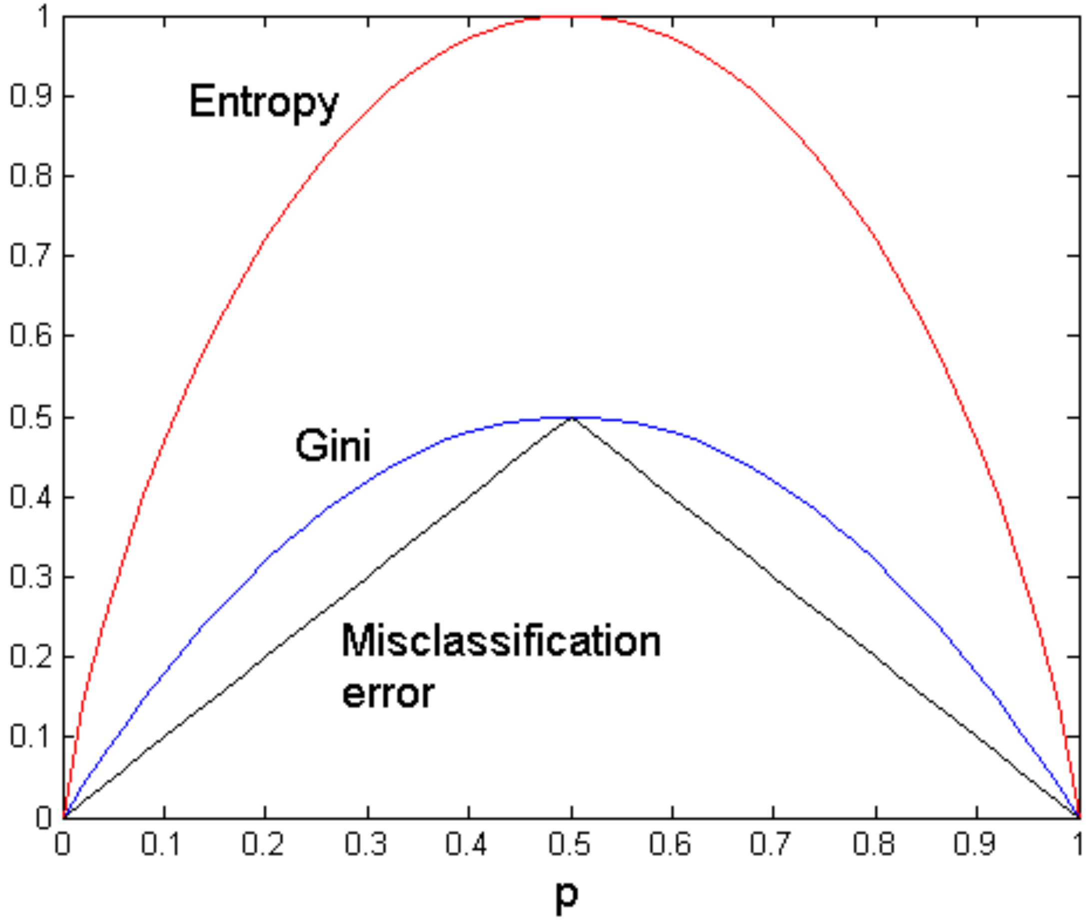
Error = 1 – max (1/6, 5/6) = 1 – 5/6 = 1/6

C1	**2**
C2	**4**

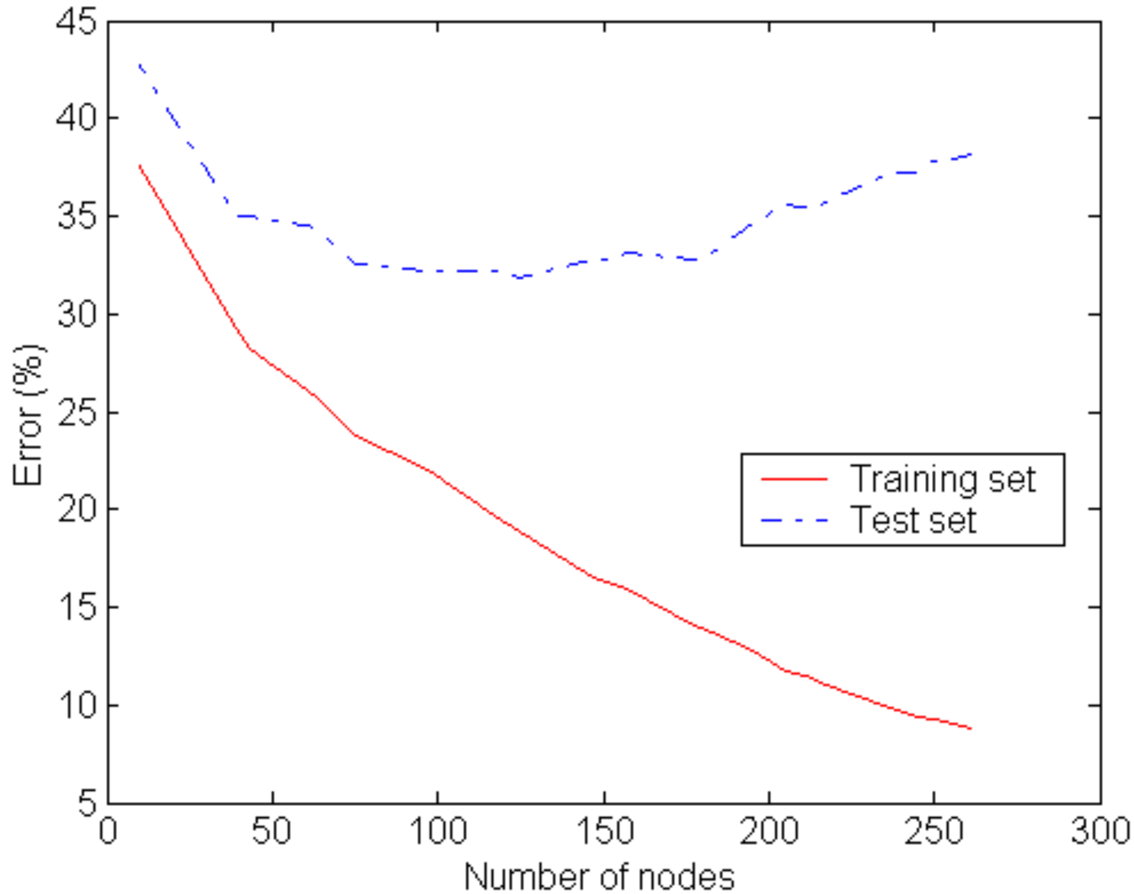
P(C1) = 2/6   P(C2) = 4/6

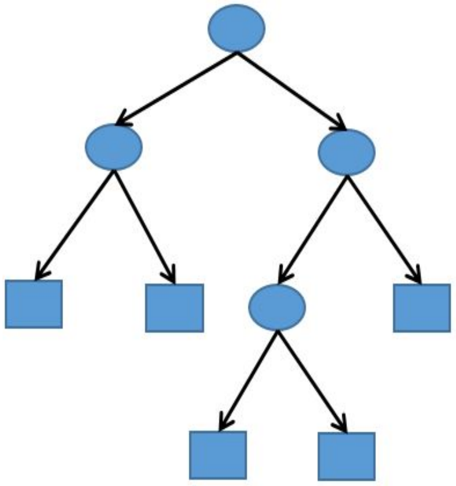
Error = 1 – max (2/6, 4/6) = 1 – 4/6 = 1/3

Measure of Impurity for 2-Class Problems



Training and Test Errors



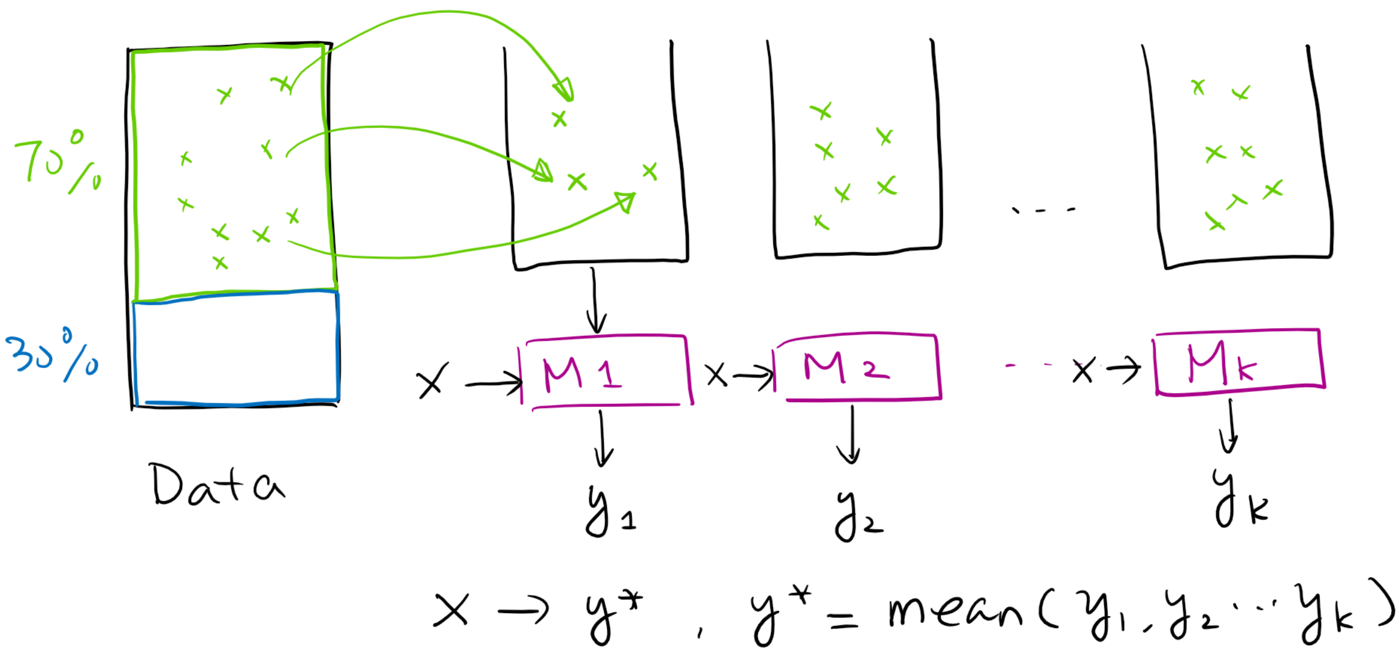


Tree Ensembles

Bagging

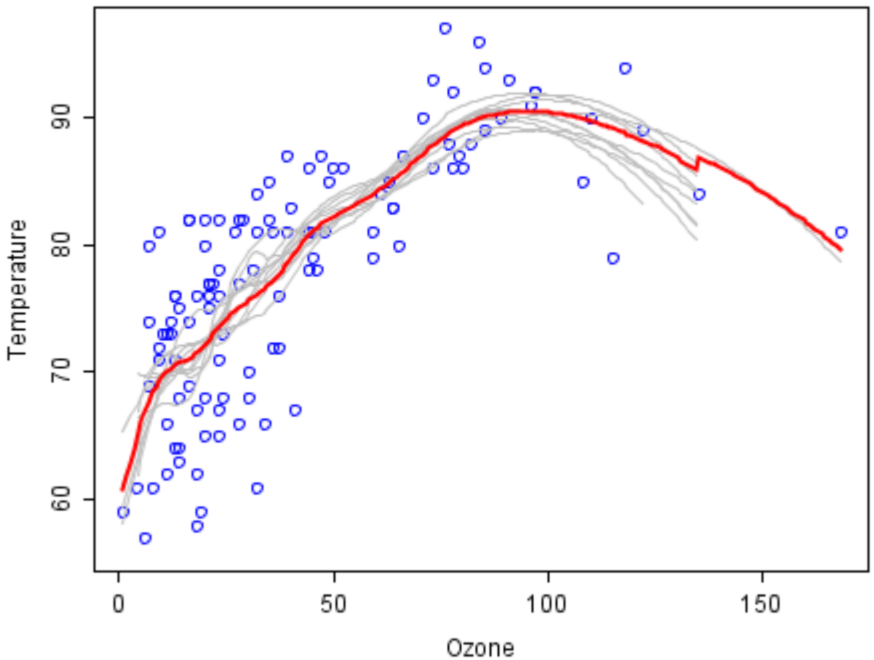
Bagging is the abbreviation of Bootstrap Aggregating.

Bootstrapping is any test or metric that relies on random sampling with replacement.



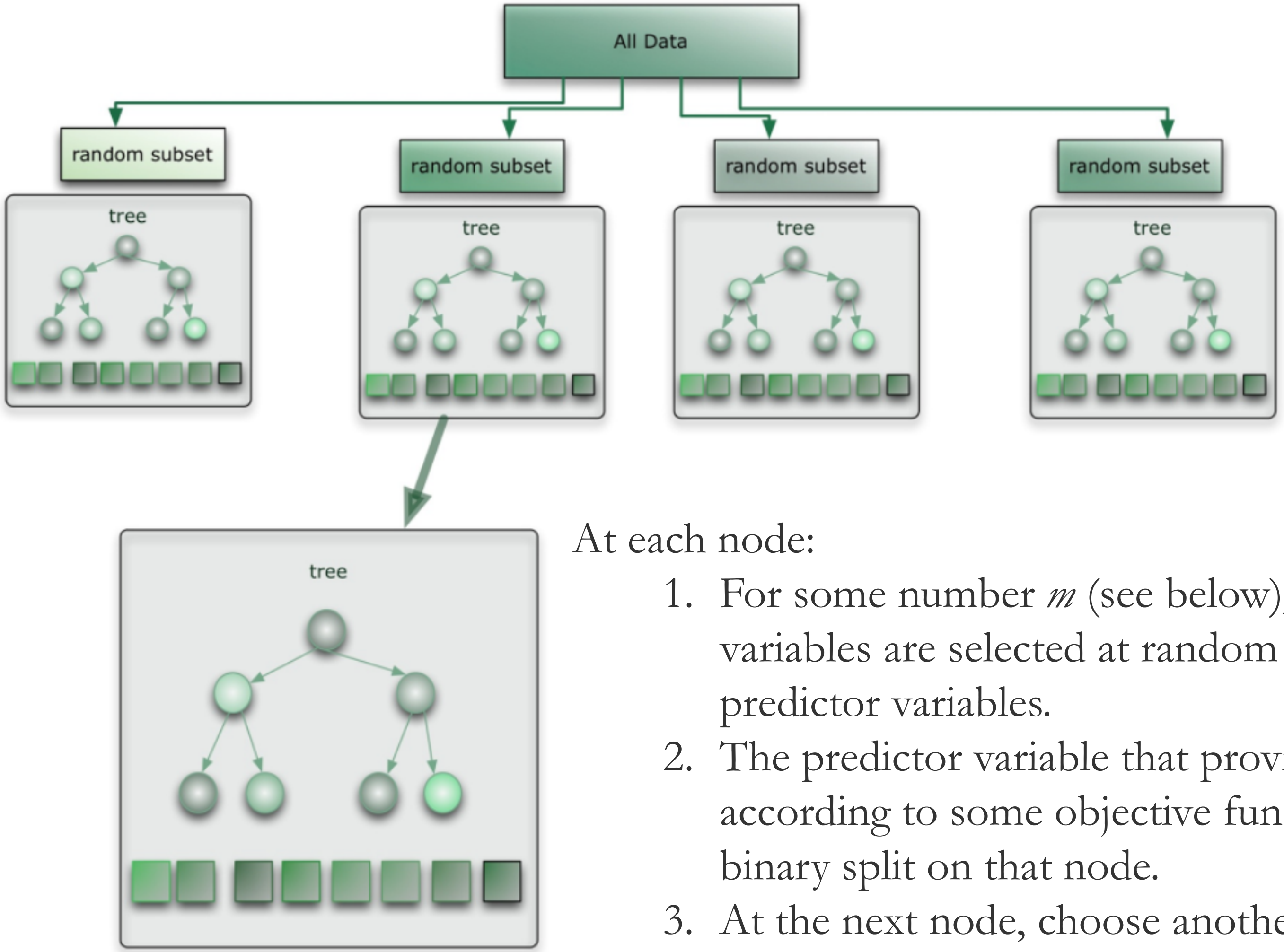
Bagging

proposed by Leo Breiman in 1994 to improve classification by combining classifications of randomly generated training sets.



Given a standard training set D of size *n*, bagging generates *m* new training sets, each of size n′, by sampling from D uniformly with replacement (bootstrapping). By sampling with replacement, some observations may be repeated in each round. If *n′=n*, then for large *n* the sampled set is expected to have the fraction (1 - 1/e) of original data (≈63.2%)

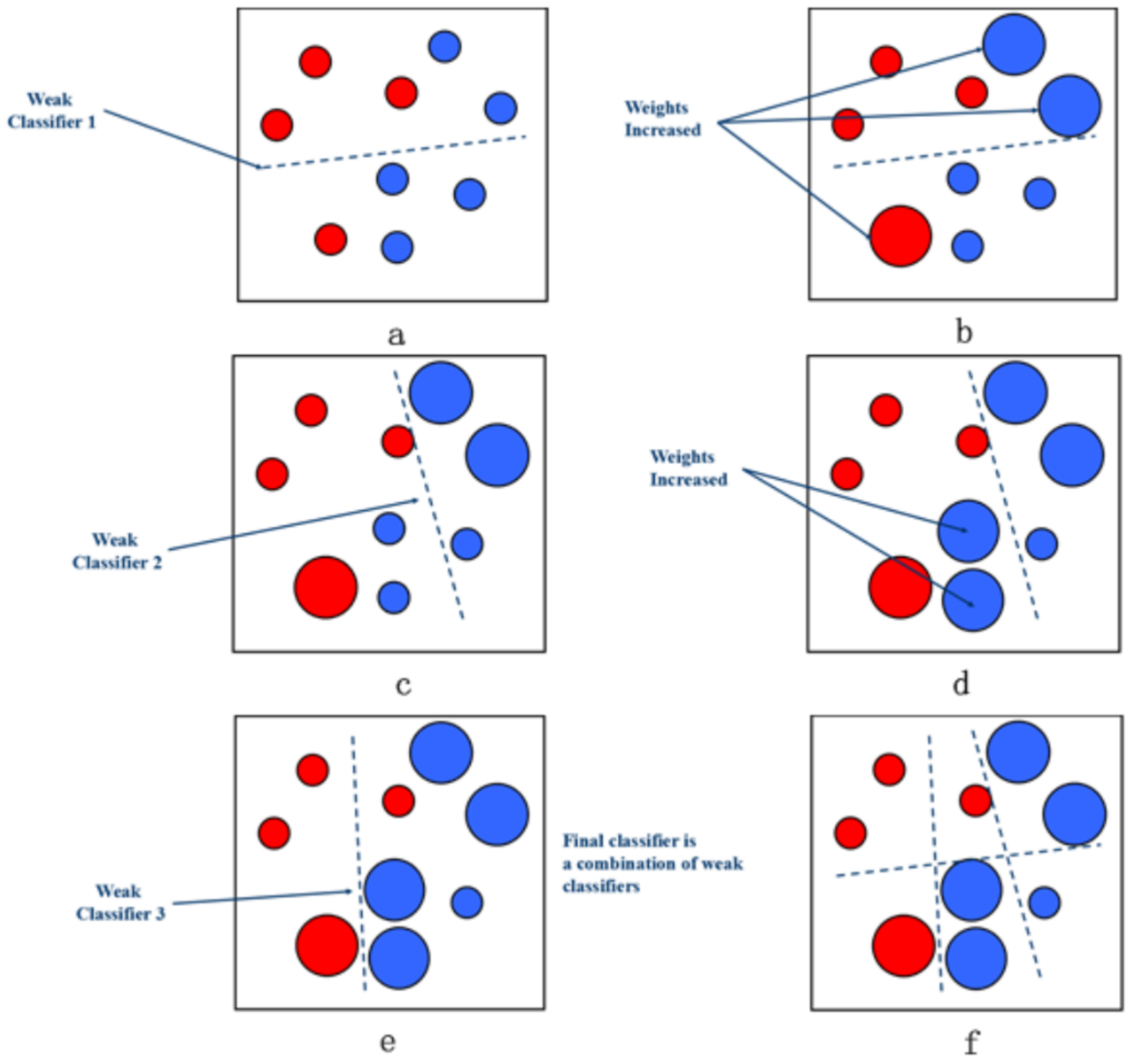
Random Forest



At each node:

1. For some number *m* (see below), *m* predictor variables are selected at random from all the predictor variables.
2. The predictor variable that provides the best split, according to some objective function, is used to do a binary split on that node.
3. At the next node, choose another *m* variables at random from all predictor variables and do the same.

Partition


Weak
Classifier 1
a
Weights
Increased
b
Weak
Classifier 2
c
Weights
Increased
d
Weak
Classifier 3
e
Final classifier is
a combination of weak
classifiers
f

Adaptive Boost

The output of the other learning algorithms ('weak learners') is combined into a weighted sum that represents the final output of the boosted classifier. AdaBoost is adaptive in the sense that subsequent weak learners are tweaked in favor of those instances misclassified by previous classifiers. AdaBoost is sensitive to noisy data and outliers.

- Samples $x_1 \dots x_n$
- Desired outputs $y_1 \dots y_n, y \in \{-1, 1\}$
- Initial weights $w_{1,1} \dots w_{n,1}$ set to $\frac{1}{n}$
- Error function $E(f(x), y, i) = e^{-y_i f(x_i)}$
- Weak learners $h\colon x \rightarrow [-1, 1]$

Information Gain

For t in $1 \dots T$:

- Choose $h_t(x)$:

  - Find weak learner $h_t(x)$ that minimizes ϵ_t,

    the weighted sum error for misclassified points $\epsilon_t = \sum_{\substack{i=1 \\ h_t(x_i) \neq y_i}}^{n} w_{i,t}$

  - Choose $\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right)$

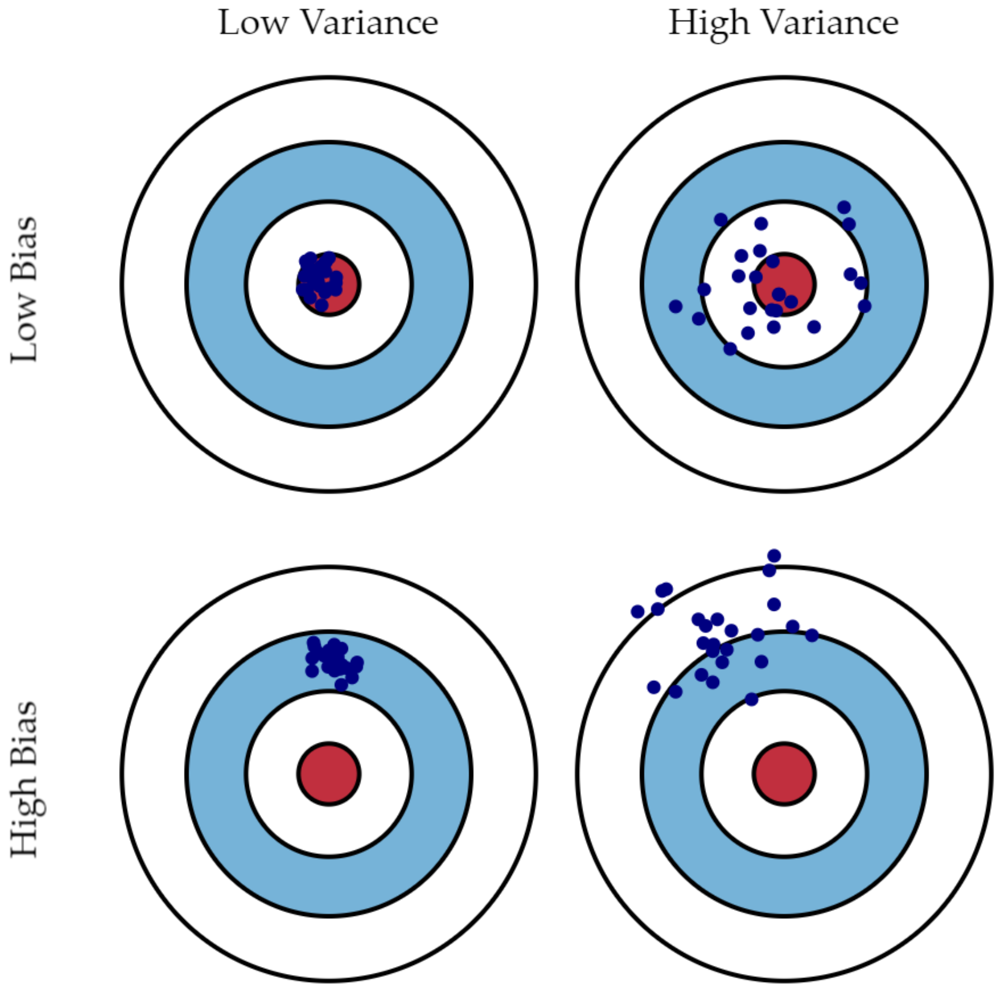
  - Add to ensemble:
    - $F_t(x) = F_{t-1}(x) + \alpha_t h_t(x)$
  - Update weights:
    - $w_{i,t+1} = w_{i,t} e^{-y_i \alpha_t h_t(x_i)}$ for all i

  - Renormalize $w_{i,t+1}$ such that $\sum_i w_{i,t+1} = 1$

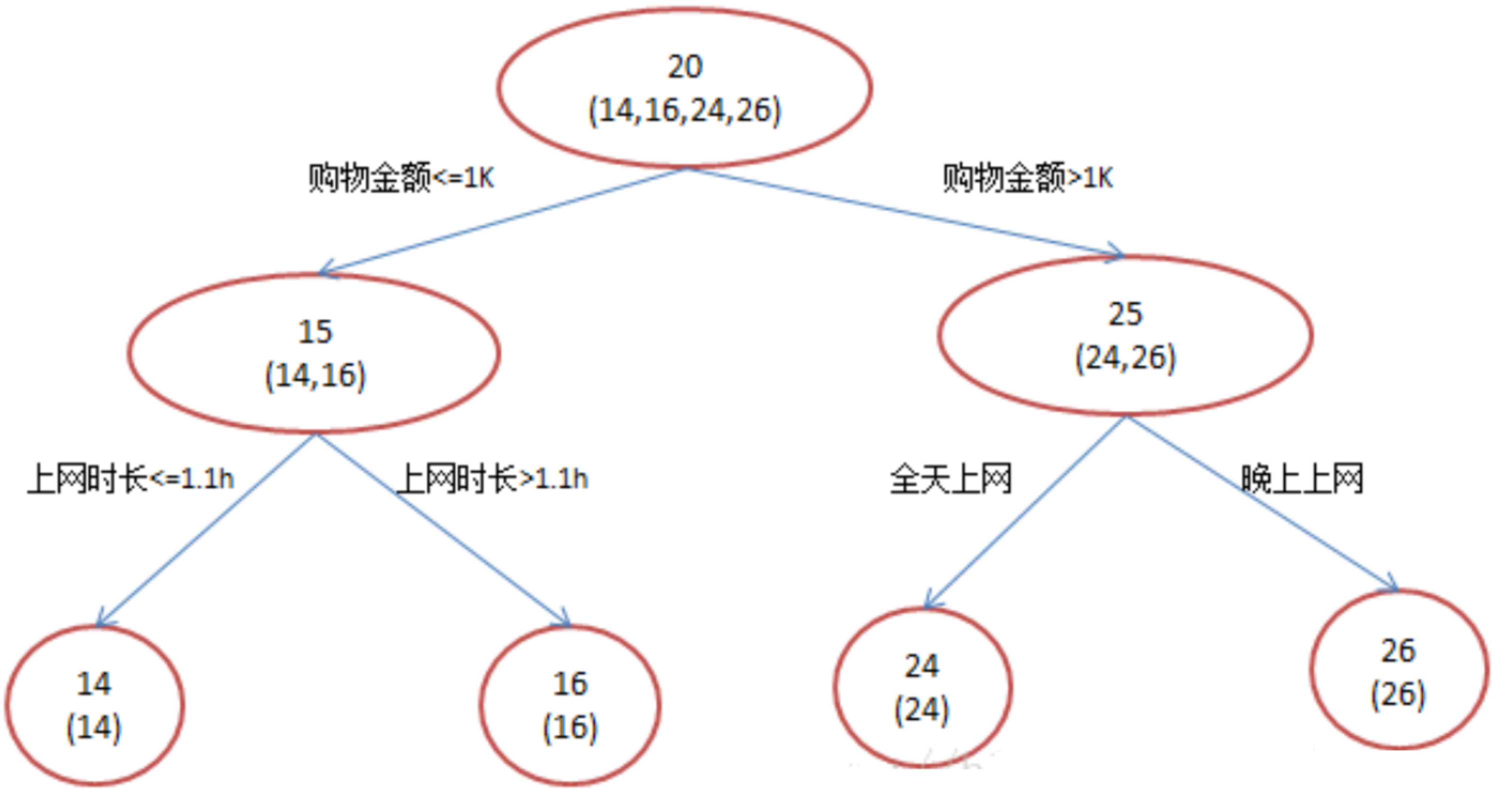
  - (Note: It can be shown that $\frac{\sum_{h_{t+1}(x_i) = y_i} w_{i,t+1}}{\sum_{h_{t+1}(x_i) \neq y_i} w_{i,t+1}} = \frac{\sum_{h_t(x_i) = y_i} w_{i,t}}{\sum_{h_t(x_i) \neq y_i} w_{i,t}}$

    at every step, which can simplify the calculation of the new weights.)

Bias-Variance

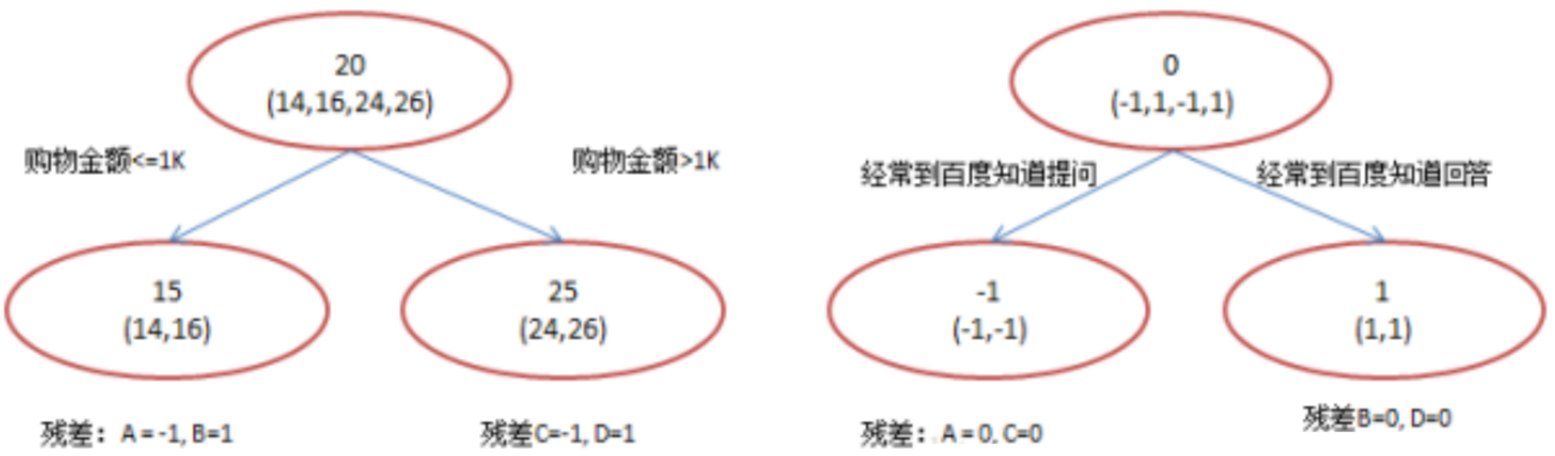


GBDT Example



https://toutiao.io/posts/u52t61/preview

Discretization



XGBoost

What is XGBoost?

XGBoost stands for e**X**treme **G**radient **B**oosting.

The name xgboost, though, actually refers to the engineering goal to push the limit of computations resources for boosted tree algorithms. Which is the reason why many people use xgboost.

https://homes.cs.washington.edu/~tqchen/2016/03/10/story-and-lessons-behind-the-evolution-of-xgboost.html