CS 559 Machine Learning

Lecture 6: Decision Trees and Ensemble Methods

Ping Wang
Department of Computer Science
Stevens Institute of Technology

Today’s Lecture

- Decision Tree
- Ensemble Methods
    - Bagging
    - Boosting
    - Random Forests

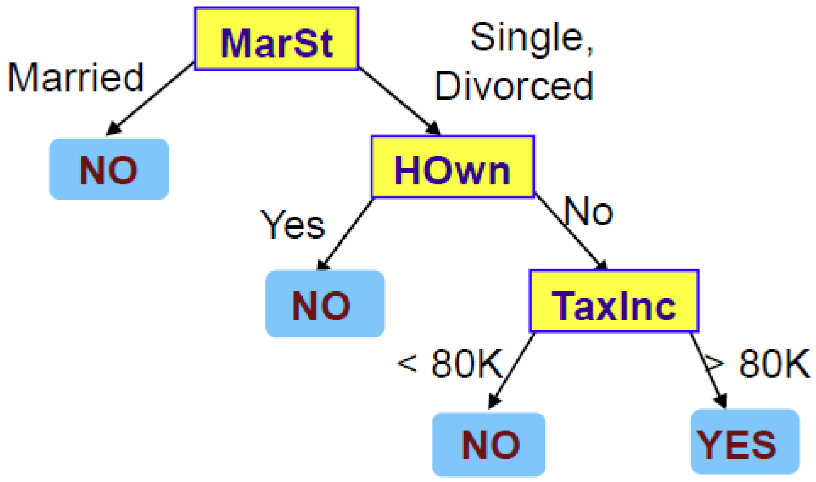
Example of a Decision Tree

- Input: 10 data points with discrete features/attributes
- Goal: predict if a person will be a defaulted borrower.
- Need to find: $f\colon\ X \rightarrow Y$

		categorical	categorical	continuous	class
	ID	**Home Owner**	**Marital Status**	**Annual Income**	**Defaulted Borrower**
	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

Decision Trees $f: X \rightarrow Y$

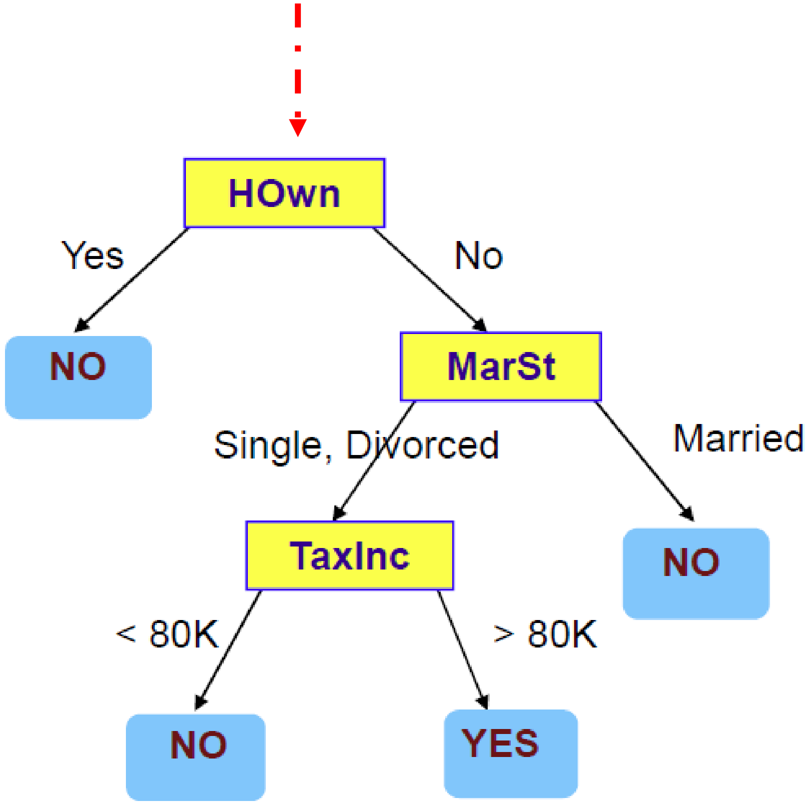
- Each internal node tests an attribute x_i
- Each branch assigns an attribute value $x_i = v$
- Each leaf assigns a class
- To classify input x: traverse the tree from root to leaf, output the labeled y



There could be more than one tree that fits the same data!

Apply Model to Test Data

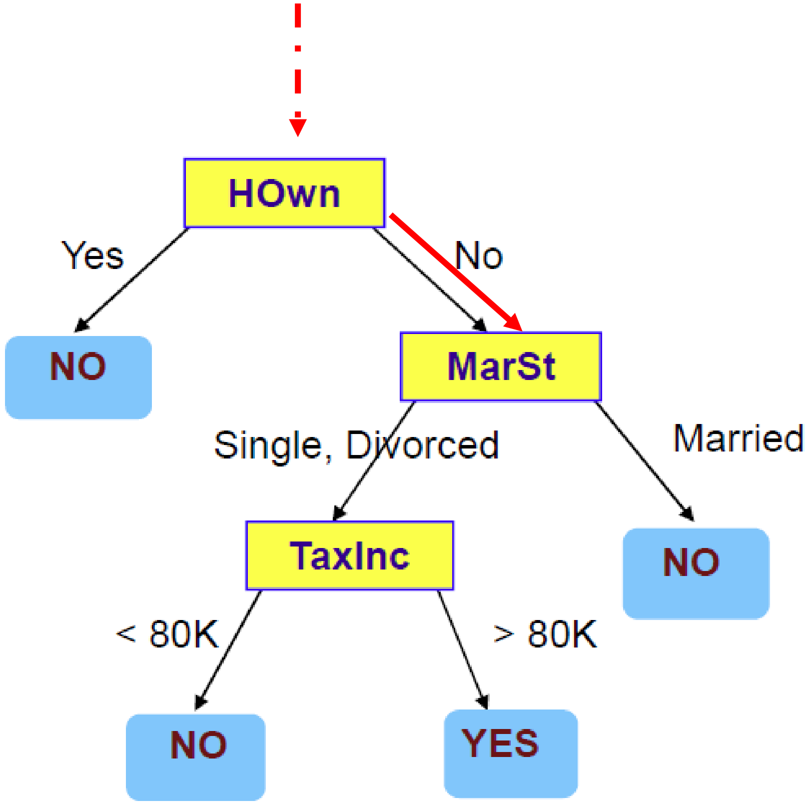
Start from the root of tree



Home Owner	Marital Status	Taxable Income	Cheat
No	Married	80K	?

Apply Model to Test Data

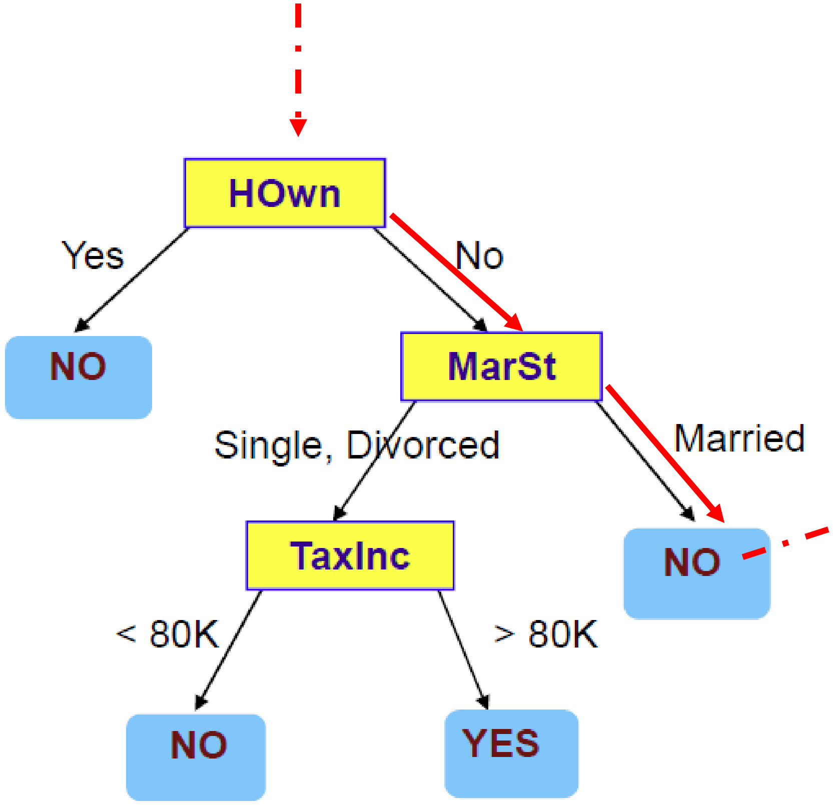
Start from the root of tree



Home Owner	Marital Status	Taxable Income	Cheat
No	Married	80K	?

Apply Model to Test Data

Start from the root of tree



Home Owner	Marital Status	Taxable Income	Cheat
No	Married	80K	?

Assign Cheat to “No”

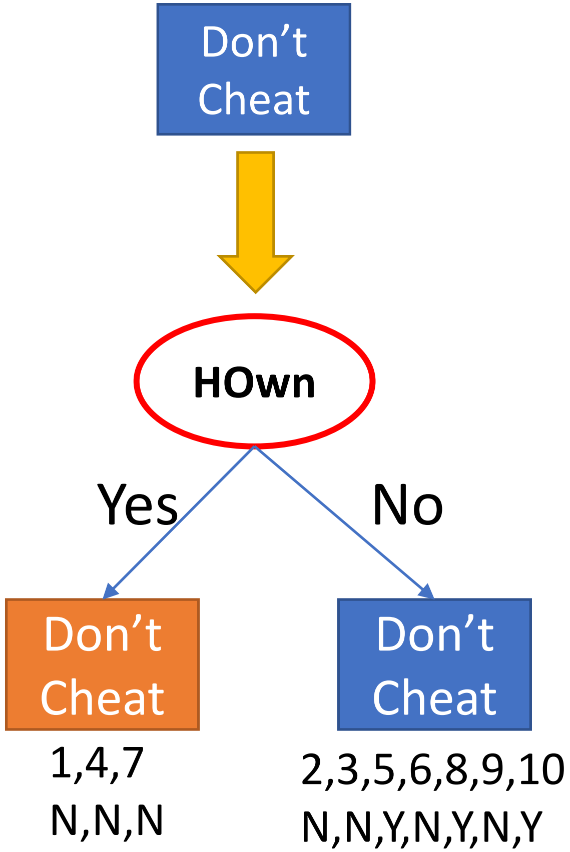
Learning Decision Tree

- Many Algorithms:
  - Hunt’s Algorithm (one of the earliest)
  - CART
  - ID3, C4.5
  - SLIQ, SPRINT

Hunt's Algorithm

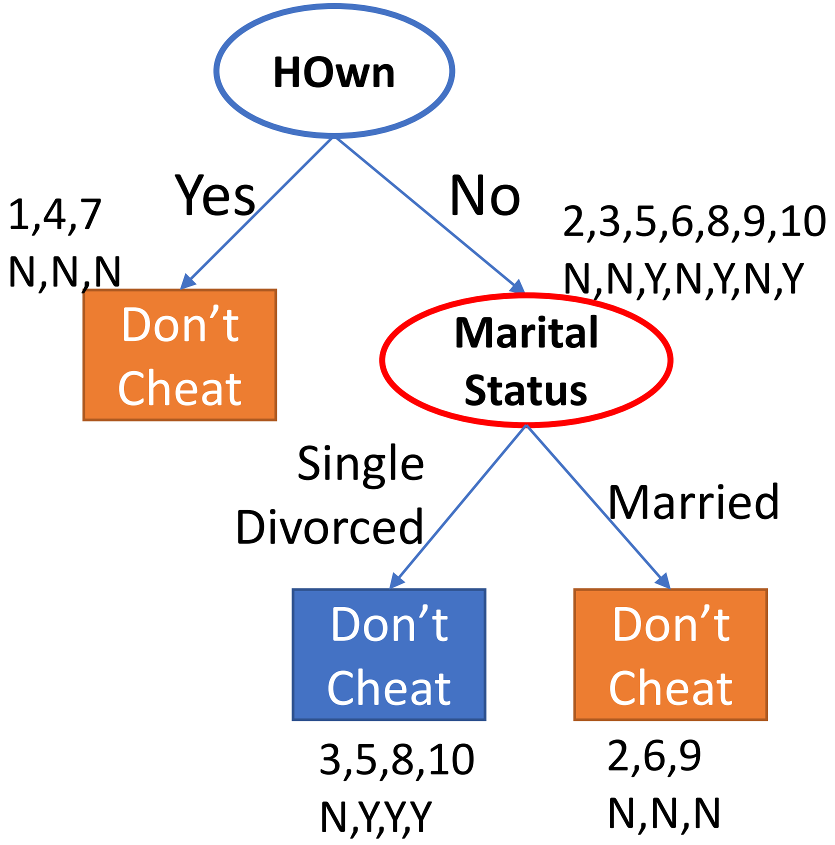
- Let D_t be the set of training records that reach a node t
- General Procedure:
  - If D_t contains records that belong to the same class y_t, then t is a leaf node labeled as y_t.
  - If D_t is an empty set, then t is a leaf node labeled by the default class y_d
  - If D_t contains records that belong to more than one class, use an attribute test to split the data into smaller subsets.
  - Recursively apply the procedure to each subset.

Hunt’s Algorithm



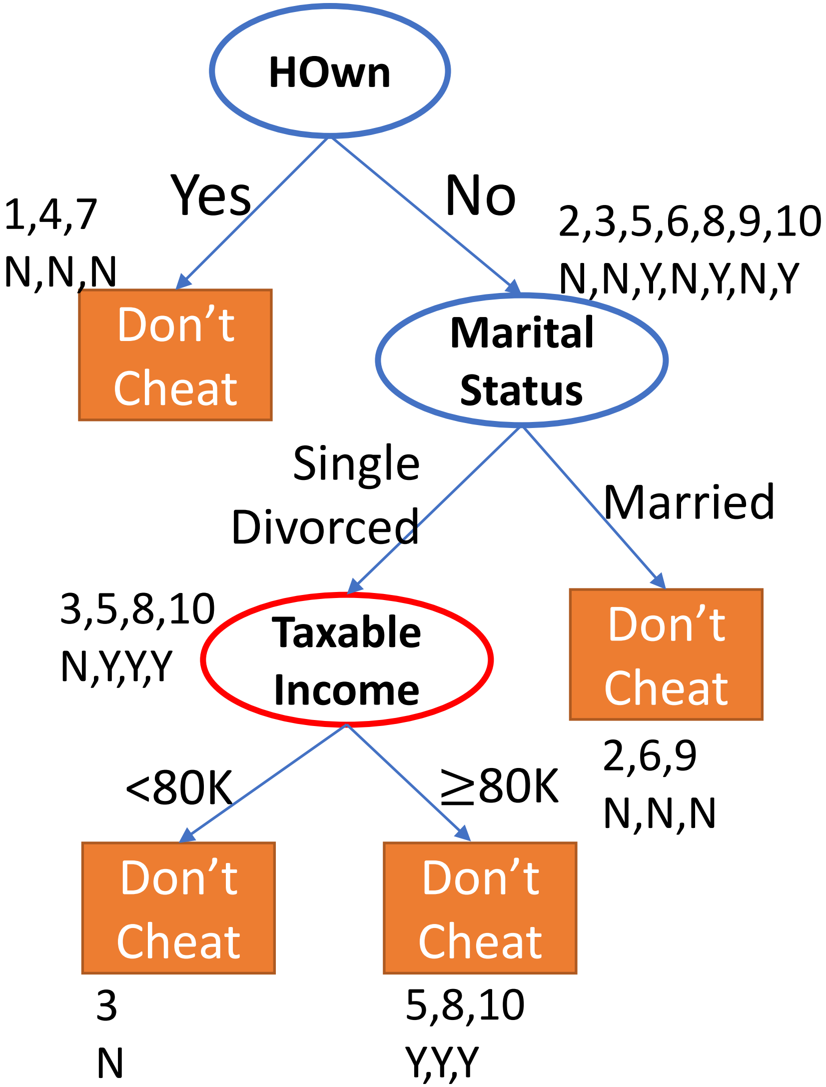
ID	Home Owner	Marital Status	Annual Income	Defaulted Borrower
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

Hunt’s Algorithm



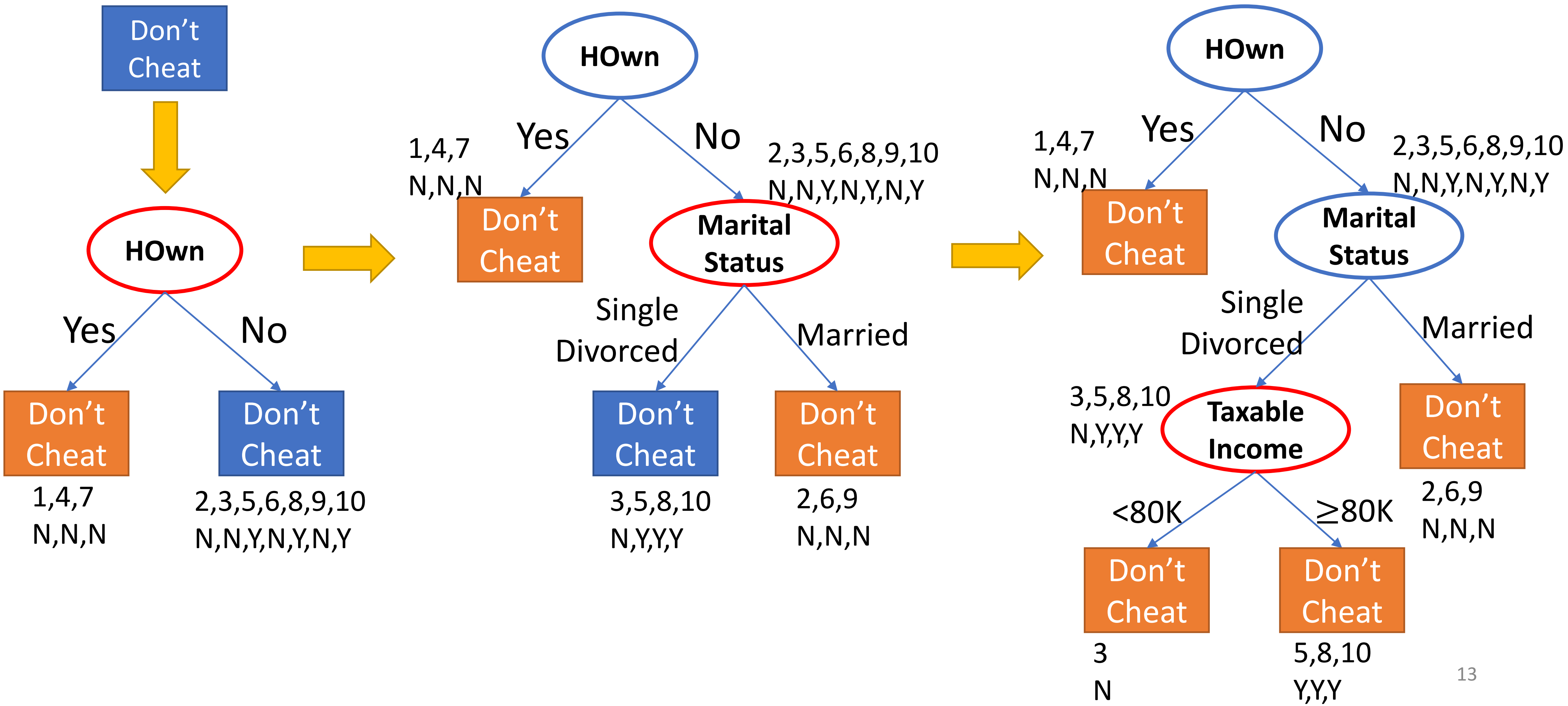
ID	Home Owner	Marital Status	Annual Income	Defaulted Borrower
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

Hunt's Algorithm



ID	Home Owner	Marital Status	Annual Income	Defaulted Borrower
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

Hunt’s Algorithm



How to Split

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

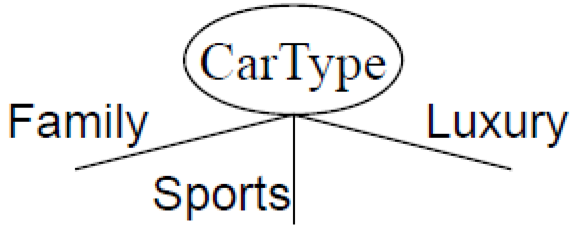
How to Specify the Attribute Condition?

- Depends on attribute types
  - Nominal
  - Ordinal
  - Continuous

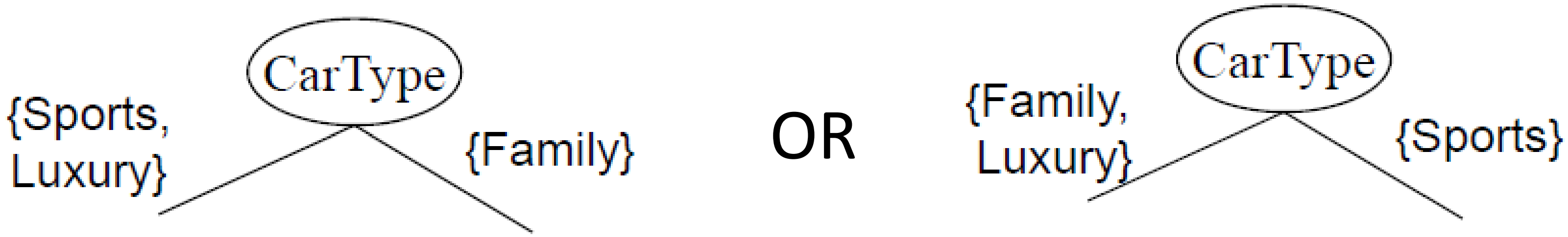
- Depends on number of ways to split
  - 2-way split
  - Multi-way split

Splitting Based on Nominal Attributes

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

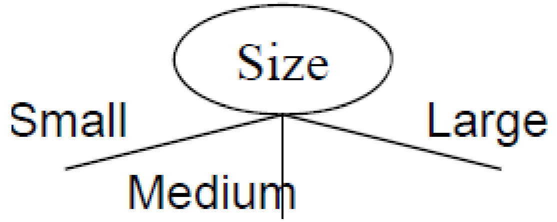


- Binary split: Divides values into two subsets. Need to find optimal partitioning.

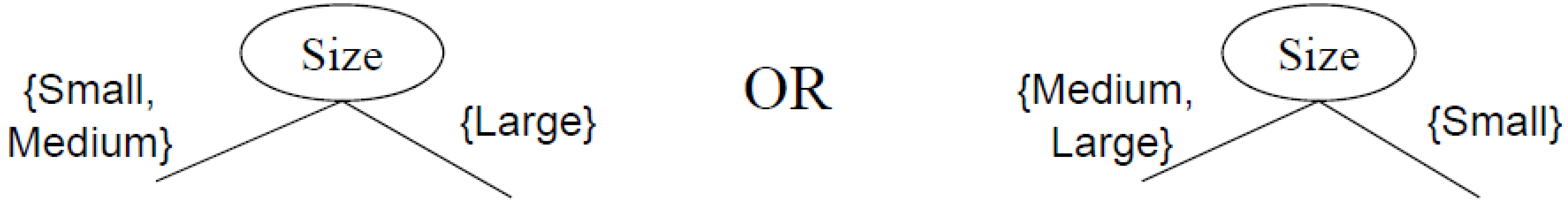


Splitting Based on Ordinal Attributes

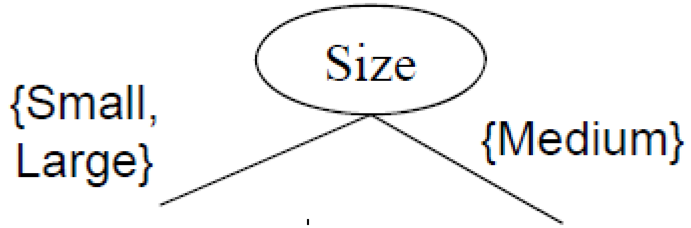
- Multi-way split: Use as many partitions as distinct values.



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



- What about this split? Preserve order property among attribute values

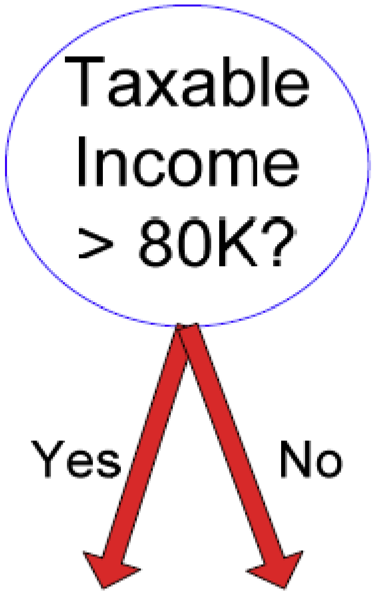


Splitting Based on Continuous Attributes

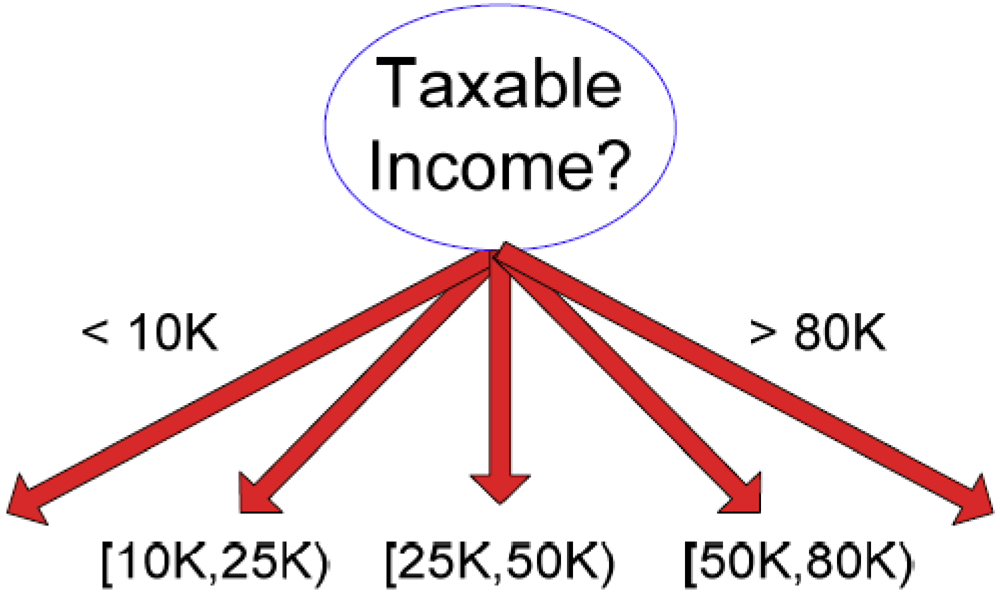
Different ways of handling:

- Discretization to form an ordinal categorical attribute
  - 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

Splitting Based on Continuous Attributes



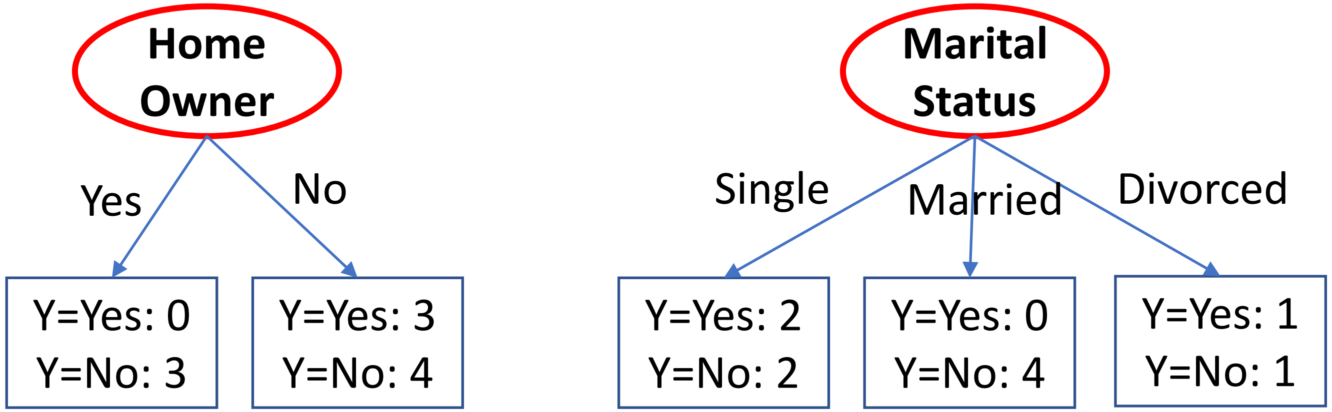
Binary split



Multi-way split

How to Determine the Best Split

- Which attribute we prefer to split on?
- Idea: use counts as leaves to define probability distribution, so we can measure uncertainty.



ID	Home Owner	Marital Status	Annual Income	Defaulted Borrower
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

How to Determine the Best Split

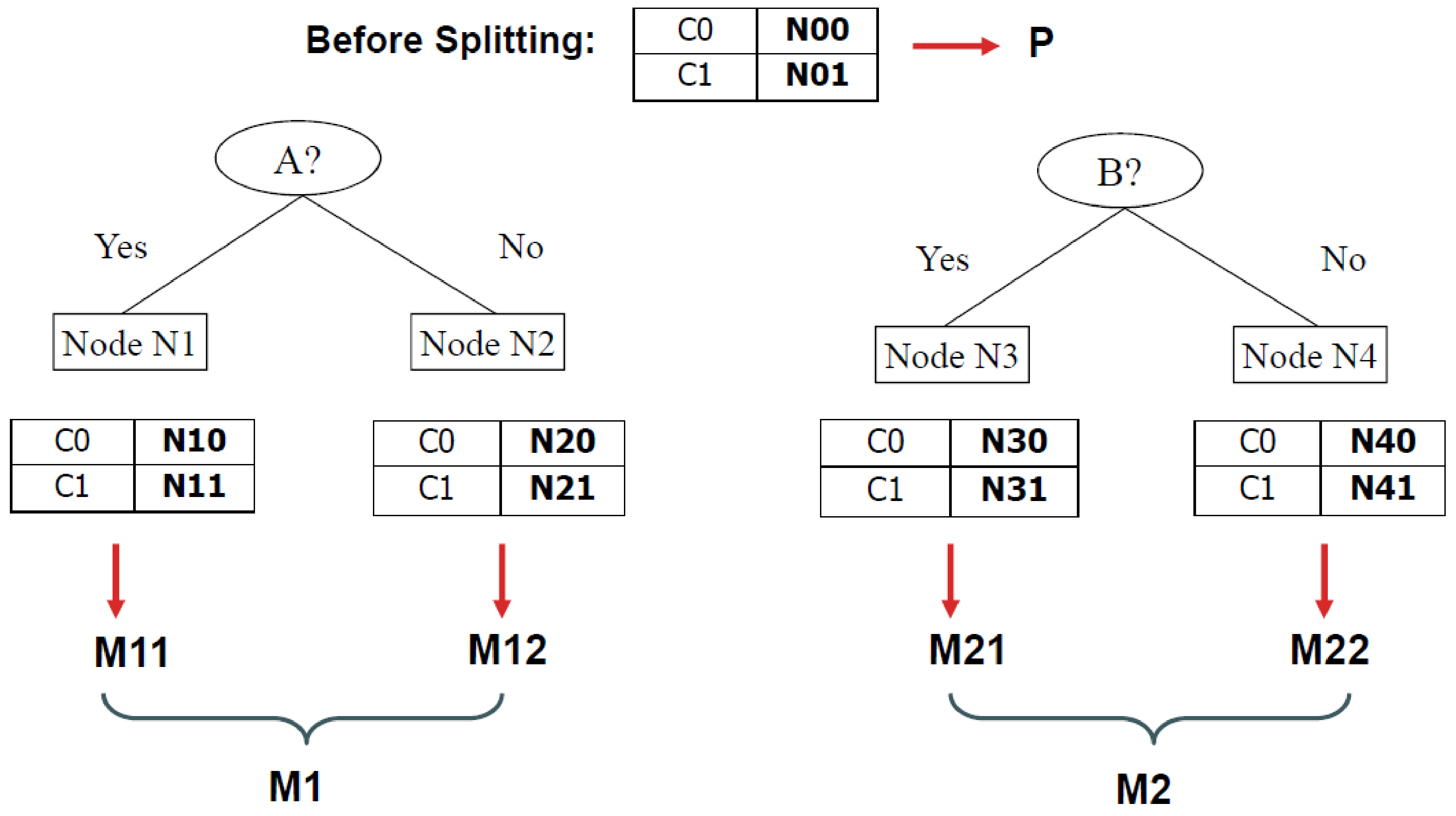
- Greedy approach:
  - Nodes with homogeneous class distribution are preferred
- Need a measure of node impurity:

Y=Yes: 5 Y=No: 5	Y=Yes: 9 Y=No: 1
Non-homogeneous High degree of impurity	Homogeneous Low degree of impurity

Measures of Node Impurity

- Gini Index
- Entropy

How to Find the Best Split



$$Gain = P - \mathbf{M1}$$

$$Gain = P - \mathbf{M2}$$

How to Find the Best Split

1. Compute impurity measure (P) before splitting
2. Compute impurity measure (M) after splitting
   - Compute impurity measure of each child node
   - Compute the average impurity of the children (M)
3. Choose the attribute test condition that produces the highest gain

$$Gain = P - \mathrm{M}$$

   or equivalently, lowest impurity measure after splitting (M)

Measure of Impurity: GINI

- Gini Index for a given node t:
  - $GINI(t) = 1 - \sum_j [p(j|t)]^2$
  - Note: $p(j|t)$ is the relative frequency of class *j* at node t.
- Maximum $\left(1 - \frac{1}{n_c}\right)$ 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	

Examples for Computing GINI

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

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

Note: $p(j|t)$ is the relative frequency of class *j* at node t.

Split Based on GINI

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

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

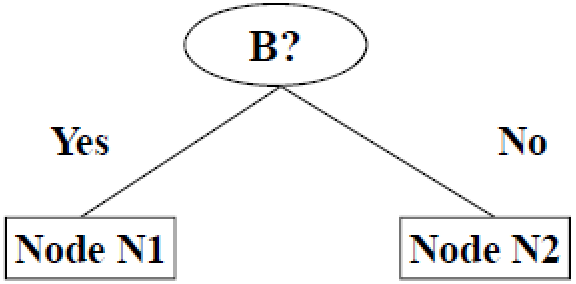
  - n_i = number of records at child i,
  - n = number of records at node p.
- Choose the attribute that minimizes weighted average Gini index of the children.

Binary Attributes: Computing GINI Index

- Splits into two partitions
- Effect of Weighing partitions: Larger and Purer Partitions needed

	Parent
C1	7
C2	5
Gini = 0.486	

$Gini(Parent)$
$= 1 - \left(\frac{7}{12}\right)^2 - \left(\frac{5}{12}\right)^2$
=0.4867



	N1	N2
C1	5	2
C2	1	4
Gini=0.361		

$Gini(N1)$
$= 1 - \left(\frac{5}{6}\right)^2 - \left(\frac{1}{6}\right)^2$
=0.278

$Gini(N2)$
$= 1 - \left(\frac{2}{6}\right)^2 - \left(\frac{4}{6}\right)^2$
=0.444

$Gini(Children)$
$= \frac{6}{12} \times 0.278 + \frac{6}{12} \times 0.444$
=0.361

Categorical Attributes: Computing Gini Index

- For each distinct value, gather counts for each class in the dataset
- Use the count matrix to make decisions

Multi-way split

	CarType		
	Family	Sports	Luxury
C1	1	8	1
C2	3	0	7
Gini	0.163		

Gini(Multi-way)
$= (4/20) * (1 – (1/4)^2 – (3/4)^2) +$
$(8/20) * (1 – (0/8)^2 – (8/8)^2) +$
$(8/20) * (1 – (1/8)^2 – (7/8)^2))$
$= 0.163$

Two-way split
(find best partition of values)

	CarType	
	{Sports, Luxury}	{Family}
C1	9	1
C2	7	3
Gini	0.468	

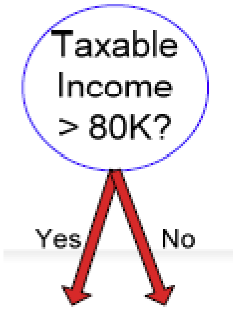
	CarType	
	{Sports}	{Family, Luxury}
C1	8	2
C2	0	10
Gini	0.167	

Gini(Two-way)
$= (16/20) * (1 – (9/16)^2 – (7/16)^2) +$
$(4/20) * (1 – (1/4)^2 – (3/4)^2)$
$= 0.468$

Continuous Attributes: Computing Gini Index

- Use Binary Decisions based on one value
- Several Choices for the splitting value
  - Possible splitting values: distinct values
- Each splitting value has a count matrix
  - Class counts in each of the partitions, $A < v$ and $A \geq v$
- Simple method to choose best v
  - For each v, scan the database to gather count matrix and compute its Gini index
  - Computationally inefficient! Repetition of work.

ID	Home Owner	Marital Status	Annual Income	Defaulted Borrower
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



Continuous Attributes: Computing Gini Index

- 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	

Gini(Split-55)

$= (0/10) * (1 – (0/0)^2 – (0/0)^2) +$
$(10/10) * (1 – (3/10)^2 – (7/10)^2)$
$= 1– 0.09 – 0.49 = 0.42$

Gini(Split-87)

$= (4/10) * (1 – (1/4)^2 – (3/4)^2) +$
$(6/10) * (1 – (2/6)^2 – (4/6)^2)$
$= 0.417$

Entropy

- Entropy for a given node t:
  - $\text{Entropy}(t) = -\sum_j p(j|t) \log_2 p(j|t)$
  - Note: $p(j|t)$ is the relative frequency of class *j* at node t.
- Measures homogeneity of a node.
  - Maximum $(\log_2 n_c)$ 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.
- Entropy based computations are similar to the GINI index computations

Examples for Computing Entropy

$$\text{Entropy}(t) = -\sum_j p(j|t) \log_2 p(j|t)$$

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

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

Entropy = – 0 log 0 – 1 log 1 = – 0 – 0 = 0

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

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

Entropy = – (1/6) $\log_2$ (1/6) – (5/6) $\log_2$ (5/6) = 0.65

C1	**3**
C2	**3**

P(C1) = 3/6   P(C2) = 3/6

Entropy = – (3/6) $\log_2$ (3/6) – (3/6) $\log_2$ (3/6) = 1

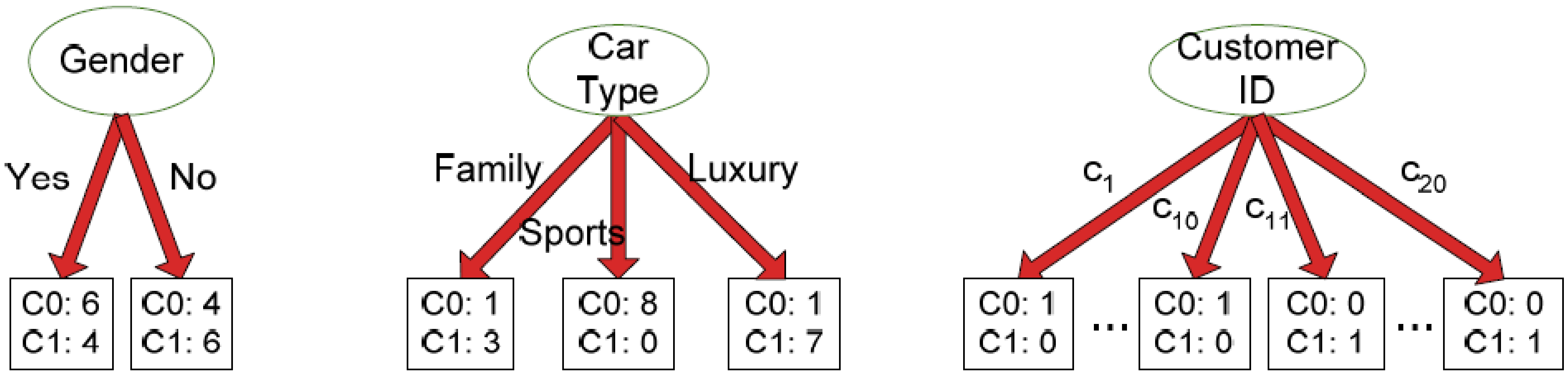
Note: $p(j|t)$ is the relative frequency of class *j* at node t.

Splitting Based on Information Gain

- Information Gain:
  - $Gain_{split} = Entropy(p) - \sum_{i=1}^{k} \frac{n_i}{n} Entropy(i)$
  - Parent Node p is split into k partitions;
  - n_i is number of records in partition i
- Measures reduction in Entropy achieved because of the split.
- Choose the split that achieves most reduction (maximizes GAIN).
- Disadvantage:
  - Tends to prefer splits that result in large number of partitions, each being small but pure.

Problems with Information Gain

- Information gain tends to prefer splits that result in large number of partitions, each being small but pure.
- Customer ID has the highest information gain because entropy for all the children is zero.



Splitting Based on Gain Ratio

- Gain Ratio:
  - $GainRATIO_{split} = \frac{Gain_{split}}{SplitINFO}$, $SplitINFO = -\sum_{i=1}^{k} \frac{n_i}{n} \log \frac{n_i}{n}$
  - Parent Node p is split into k partitions;
  - n_i is number of records in partition i
- Adjusts information gain by the entropy of the partitioning (SplitINFO). Higher entropy partitioning (large number of small partitions) is penalized!
- Designed to overcome the disadvantage of Information Gain
  - For example : 10 records (parent node)
  - Two equal partitions (5,5): SplitINFO = - 2*(5/10)$\log_2(5/10)$ = 1
  - Ten equal partitions: SplitINFO = -10*(1/10)$\log_2(1/10)$ = 3.32

Stopping Criteria

- Stop expanding a node when all the records belong to the same class
- Stop when the number of records have fallen below some minimum threshold
- Early termination

Decision Tree Based Classification

- Advantages:
  - Computationally inexpensive to construct
  - Extremely fast at classifying unknown records
  - Easy to interpret for small-sized trees
  - Accuracy is comparable to other classification techniques for many simple data sets
- Disadvantages: Overfitting
  - Overfitting can be due to lack of representative samples or due to some noise
  - Overfitting results in decision trees that are more complex than necessary
  - Training error no longer provides a good estimate of how well the tree will perform on previously unseen records

How to Address Overfitting: Pre-Pruning (Early Stopping Rule)

- Stop the algorithm before it becomes a fully-grown tree
- Typical stopping conditions for a node:
  - Stop if all instances belong to the same class
  - Stop if all the attribute values are the same
- More restrictive conditions:
  - Stop if number of instances is less than some user-specified threshold
  - Stop if class distribution of instances are independent of the available features
  - Stop if expanding the current node does not improve impurity measures (e.g., Gini or information gain).

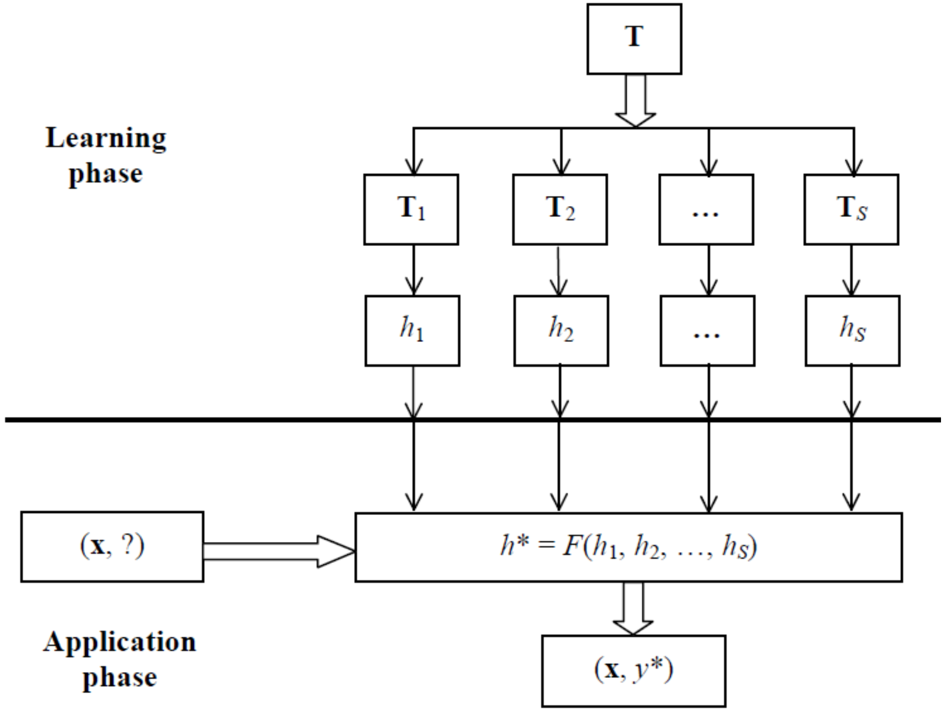
How to Address Overfitting: Post-Pruning

- Grow decision tree to its entirety
- Trim the nodes of the decision tree in a bottom-up fashion
- If generalization error improves after trimming, replace sub-tree by a leaf node.
- Class label of leaf node is determined from majority class of instances in the sub-tree.

Ensemble Methods

Ensemble

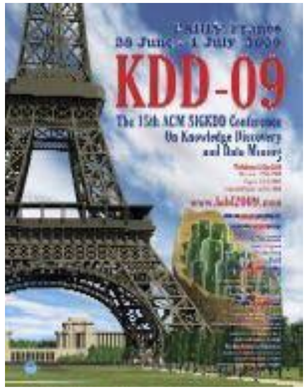
- Basic idea: Combine multiple models into one!
  - Build different “experts” and let them vote



Different training sets and/or learning algorithms

Stories of Success

- Million-dollar prize
  - Improve the baseline movie recommendation approach of Netflix by 10% in accuracy
  - The top submissions all combine several teams and algorithms as an ensemble



- Data mining competitions
  - Classification problems
  - Winning teams employ an ensemble of classifiers

Netflix Prize

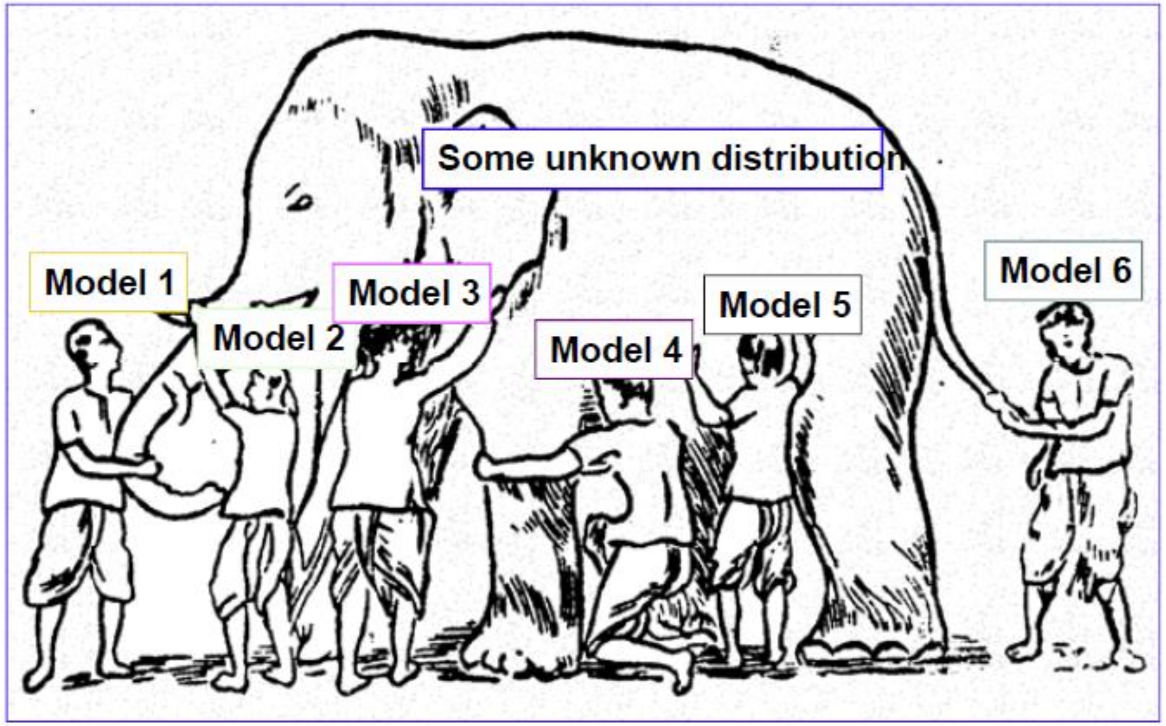
- Supervised learning task
  - Training data is a set of users and ratings (1,2,3,4,5 stars) those users have given to movies.
  - Construct a classifier that given a user and an unrated movie, correctly classifies that movie as either 1, 2, 3, 4, or 5 stars
  - $1 million prize for a 10% improvement over Netflix's current movie recommender
- Competition
  - At first, single-model methods are developed, and performances are improved
  - However, improvements slowed down
  - Later, individuals and teams merged their results, and significant improvements are observed

Motivations of Ensemble Methods

- Ensemble model improves accuracy and robustness over single model methods
- Efficiency: a complex problem can be decomposed into multiple sub-problems that are easier to understand and solve (divide-and-conquer approach)
- Applications:
  - Distributed computing
  - Privacy-preserving applications
  - Large-scale data with reusable models
  - Multiple sources of data

Why Ensemble Works?

- The given task may be too complex, or lie outside the space of functions that can be implemented by the chosen classifier method
- Appropriate combinations of simple (e.g., linear) classifiers can learn complex (e.g., non-linear) boundaries
- **Ensemble gives the global picture!**



Pros and Cons

- Advantages:
  - Improve predictive performance
  - Different types of classifiers can be directly included
  - Easy to implement
  - Not too much parameter tuning
- Disadvantages:
  - The combined classifier is not transparent (black box)
  - Not a compact representation

How to Make an Effective Ensemble?

- Two basic decisions when designing ensembles:
  - How to generate the base classifiers?
  - How to integrate/combine them?

Generating Base Classifiers

- Sampling training examples
  - Train k classifiers on k subsets drawn from the training set
- Using different learning models
  - Use all the training examples, but apply different learning algorithms
- Sampling features
  - Train k classifiers on k subsets of features drawn from the feature space
- Learning “randomly”
  - Introduce randomness into learning procedures

How to Integrate?

- Majority voting
- Weighted majority voting

Diversity and Accuracy

- The individual classifiers must be diverse (errors on different data)
- If they make the same errors, such mistakes will be carried into the final prediction
- The component classifiers need to be “reasonably accurate“ to avoid poor classifiers to obtain the majority of votes.

Ensemble Methods

- Predict class label for unseen data by aggregating a set of predictions (classifiers learned from the training data)
  - Bagging (Breiman 1994 “Bagging Predictors")
  - Boosting (Freund and Schapire 1995, Friedman et al. 1998)
  - Random forests (Breiman 2001 “Random Forests")

Bagging: Bootstrap Aggregation

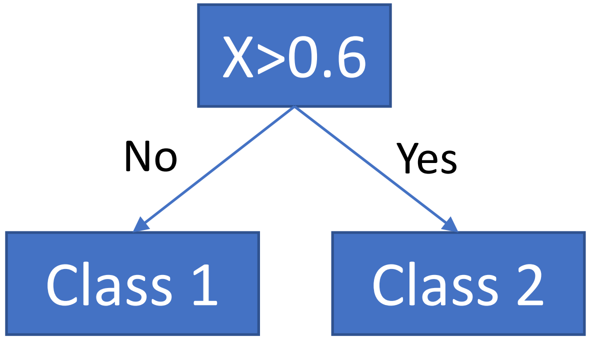
- Take repeated bootstrap samples from training set D (Breiman, 1994)
- Bootstrap sampling:
  - Bootstrap sampling: Given set D containing N training examples, create D' by drawing N examples at random with replacement from D
- Bagging:
  - Create k bootstrap samples $D_1, \ldots, D_k$
  - Train distinct classifier on each D_i
  - Classify new instances by majority vote/average

Bagging Example

- Only one feature

x	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
Y	1	1	1	-1	-1	-1	-1	1	1	1

- Decision stump



Bagging Example: Classifier 1-5

Bagging Round 1:

x	0.1	0.2	0.2	0.3	0.4	0.4	0.5	0.6	0.9	0.9
y	1	1	1	1	-1	-1	-1	-1	1	1

x <= 0.35 ==> y = 1
x > 0.35 ==> y = -1

Bagging Round 2:

x	0.1	0.2	0.3	0.4	0.5	0.8	0.9	1	1	1
y	1	1	1	-1	-1	1	1	1	1	1

x <= 0.65 ==> y = 1
x > 0.65 ==> y = 1

Bagging Round 3:

x	0.1	0.2	0.3	0.4	0.4	0.5	0.7	0.7	0.8	0.9
y	1	1	1	-1	-1	-1	-1	-1	1	1

x <= 0.35 ==> y = 1
x > 0.35 ==> y = -1

Bagging Round 4:

x	0.1	0.1	0.2	0.4	0.4	0.5	0.5	0.7	0.8	0.9
y	1	1	1	-1	-1	-1	-1	-1	1	1

x <= 0.3 ==> y = 1
x > 0.3 ==> y = -1

Bagging Round 5:

x	0.1	0.1	0.2	0.5	0.6	0.6	0.6	1	1	1
y	1	1	1	-1	-1	-1	-1	1	1	1

x <= 0.35 ==> y = 1
x > 0.35 ==> y = -1

Bagging Example: Classifier 6-10

Bagging Round 6:

x	0.2	0.4	0.5	0.6	0.7	0.7	0.7	0.8	0.9	1
y	1	-1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 ==> y = -1
x > 0.75 ==> y = 1

Bagging Round 7:

x	0.1	0.4	0.4	0.6	0.7	0.8	0.9	0.9	0.9	1
y	1	-1	-1	-1	-1	1	1	1	1	1

x <= 0.75 ==> y = -1
x > 0.75 ==> y = 1

Bagging Round 8:

x	0.1	0.2	0.5	0.5	0.5	0.7	0.7	0.8	0.9	1
y	1	1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 ==> y = -1
x > 0.75 ==> y = 1

Bagging Round 9:

x	0.1	0.3	0.4	0.4	0.6	0.7	0.7	0.8	1	1
y	1	1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 ==> y = -1
x > 0.75 ==> y = 1

Bagging Round 10:

x	0.1	0.1	0.1	0.1	0.3	0.3	0.8	0.8	0.9	0.9
y	1	1	1	1	1	1	1	1	1	1

x <= 0.05 ==> y = -1
x > 0.05 ==> y = 1

Bagging Example

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	1	1	1	-1	-1	-1	-1	-1	-1	-1
2	1	1	1	1	1	1	1	1	1	1
3	1	1	1	-1	-1	-1	-1	-1	-1	-1
4	1	1	1	-1	-1	-1	-1	-1	-1	-1
5	1	1	1	-1	-1	-1	-1	-1	-1	-1
6	-1	-1	-1	-1	-1	-1	-1	1	1	1
7	-1	-1	-1	-1	-1	-1	-1	1	1	1
8	-1	-1	-1	-1	-1	-1	-1	1	1	1
9	-1	-1	-1	-1	-1	-1	-1	1	1	1
10	1	1	1	1	1	1	1	1	1	1
Sum	2	2	2	-6	-6	-6	-6	2	2	2
Sign	1	1	1	-1	-1	-1	-1	1	1	1
True Class	1	1	1	-1	-1	-1	-1	1	1	1

Figure 5.36. Example of combining classifiers constructed using the bagging approach.

Boosting

Basic idea:

- Assign a weight to every training set instance
- Initially, all instances have the same weight
- As the boosting proceeds, adjust weights based on how well we have predicted data points so far
  - data points correctly predicted → low weight
  - data points mis-predicted → high weight

- Results: as learning proceeds, the learner is forced to focus on portions of data space not previously well predicted

Formal Description of Boosting

- Given training set $(x_1, y_1), \ldots, (x_N, y_N)$
- $y_i \in \{-1, +1\}$ correct labels of instance $x_i \in X$
- For $t = 1, \ldots, T$:
  - Construct weight distribution $D^t(i)$ on $\{1, \ldots, N\}$
  - Find weak classifier : $h_t: X \rightarrow \{-1, +1\}$
  - With error ϵ_t on $D^t(i)$: $\epsilon_t = P_{i \sim D^{(t)}}[h_t(x_i) \neq y_i]$
- Output final/combined classifier H_{final}

AdaBoost

- Given: $(x_1, y_1), \ldots, (x_N, y_N)$ where $x_i \in X, y_i \in \{-1, +1\}$
- Initialize $D_1(i) = \frac{1}{N}$
- For $t = 1, \ldots, T$:
  - Train weak learner using distribution D_t
  - Get weak classifier $h_t: X \rightarrow \{-1, +1\}$
  - Choose $\alpha_t \in \mathbb{R}$.
  - Update: $D_t(i) = \frac{D_t(i)}{Z_t} \begin{cases} e^{-\alpha_t} & if\ y_i = h_t(x_i) \\ e^{\alpha_t} & if\ y_i \neq h(x_i) \end{cases} = \frac{D_t(i)\exp(-\alpha_t y_i h_t(x_i))}{Z_t}$

    with Z_t as a normalization factor $Z_t = \sum_{i=1}^{N} D_t(i)\exp(-\alpha_t y_i h_t(x_i))$.
- Output the final classifier: $H_{final}(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$

Naïve Bayes, decision stump,…

Increase weight if predicting incorrectly

How to Choose α_t for Hypothesis h_t

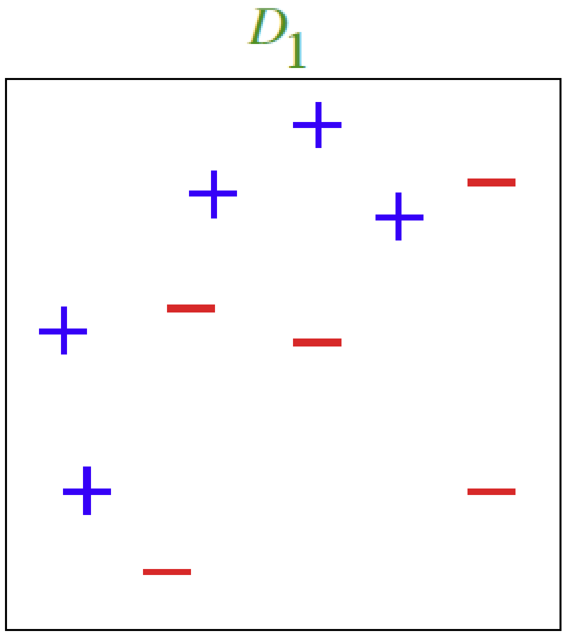
$$\alpha_t = \frac{1}{2} \ln\left(\frac{1 - \epsilon_t}{\epsilon_t}\right)$$

$$\epsilon_t = P_{i \sim D^{(t)}}[h_t(x_i) \neq y_i] = \sum_{i=1}^{N} D_t(i)\delta(h_t(x_i) \neq y_i)$$

- $\epsilon_t = 0, \alpha_t = \infty$: if h_t perfectly classifies all weighted data points
- $\epsilon_t = 1, \alpha_t = -\infty$: if h_t classifies incorrectly on all points
- $\epsilon_t = 0.5, \alpha_t = 0.5$

Smaller error rate, larger weight for voting!

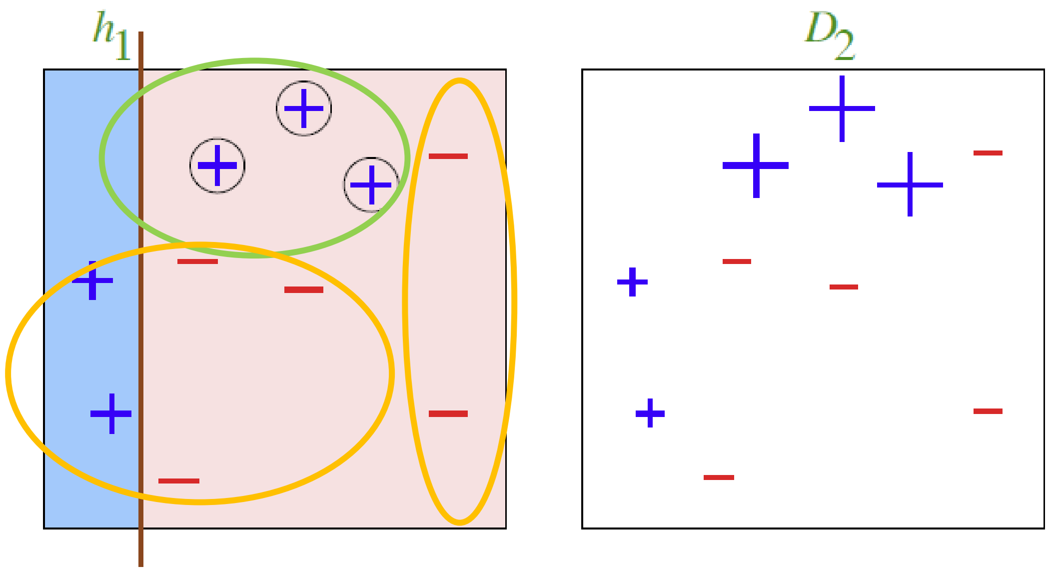
Example of Adaboost



$$D_1(1) = \cdots = D_1(10) = \frac{1}{10}$$

weak classifiers = vertical or horizontal half-planes

Example of Adaboost: Round 1



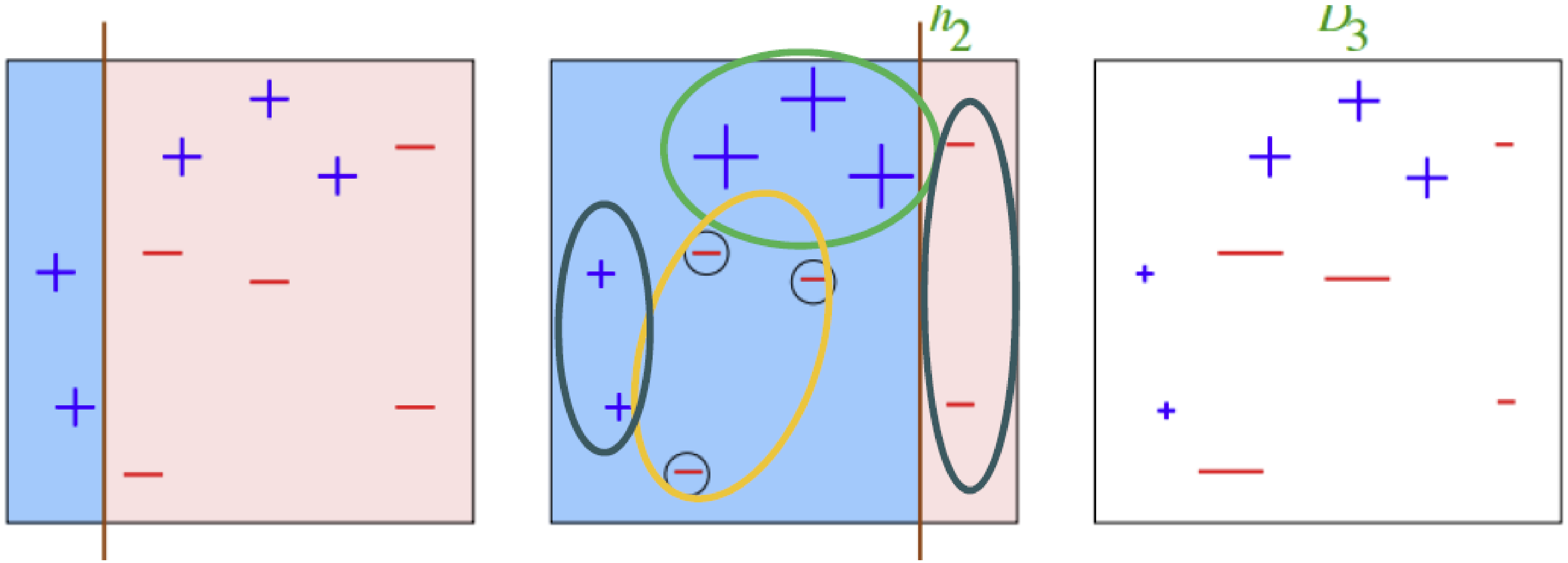
$$err_1 = 0.1 \times 3 = 0.3; \alpha_1 = \frac{1}{2} \times \log(\frac{1-0.3}{0.3}) = 0.42$$

$$D_t(i) = \frac{D_t(i)\exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

For those classified not correctly: $D_2(i) = 0.1 \times e^{0.42 \times 1} = 0.1527$ **Normalization** 0.1667

For those classified correctly: $D_2(i) = 0.1 \times e^{0.42 \times (-1)} = 0.0654$ 0.0714

Example of Adaboost: Round 2



$$err_2 = 0.0714 \times 3 = 0.21; \alpha_2 = \frac{1}{2} \times \log(\frac{1-0.21}{0.21}) = 0.6625$$

For those classified correctly: $D_3(i) = 0.1667 \times e^{0.6625 \times (-1)} = 0.0859$

For those classified not correctly $D_3(i) = 0.0714 \times e^{0.6625 \times 1} = 0.1385$

For those classified correctly: $D_3(i) = 0.0714 \times e^{0.6625 \times (-1)} = 0.0368$

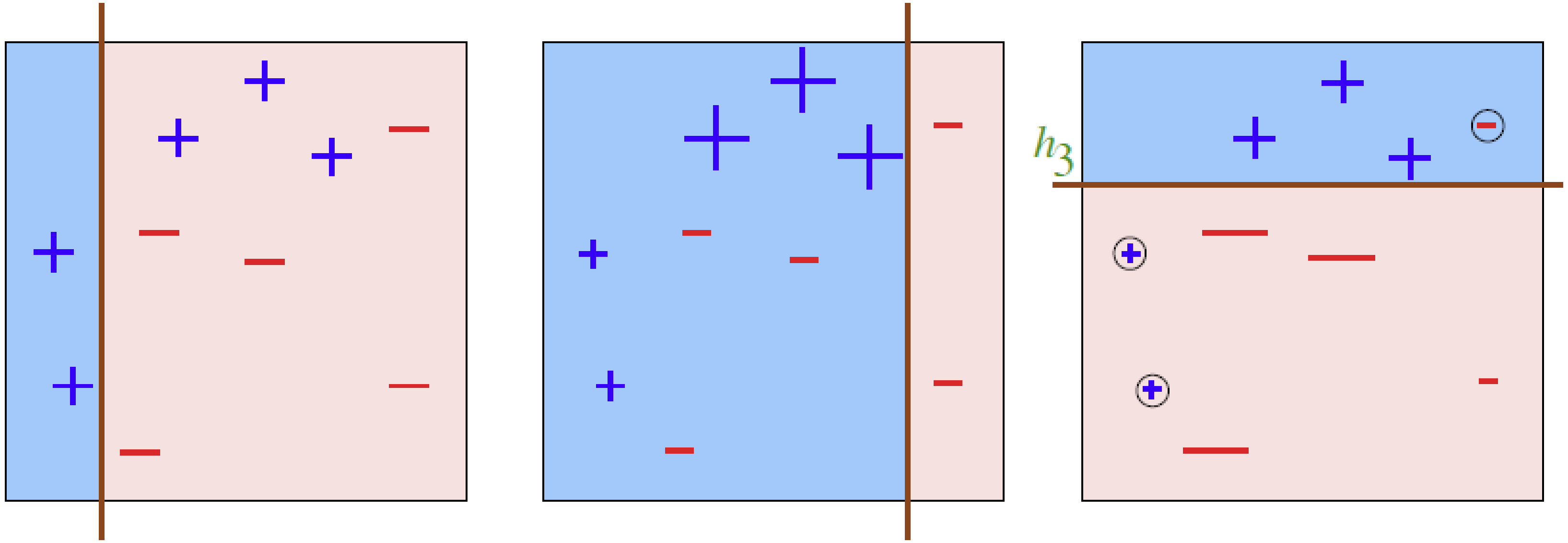
Normalization

0.1047

0.1688

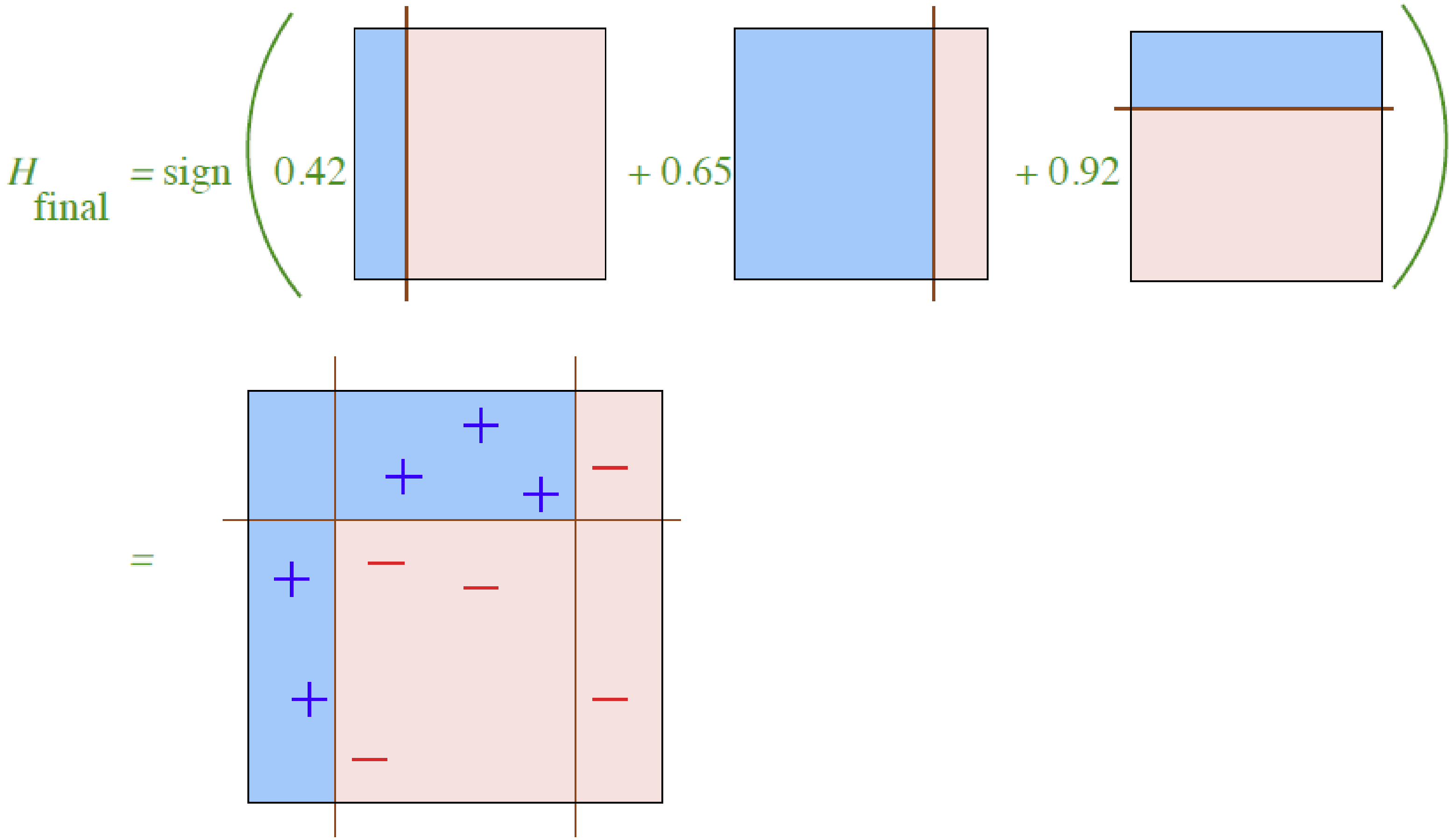
0.0449

Example of Adaboost: Round 3



$$err_3 = 0.0449 \times 3 = 0.1347; \alpha_3 = \frac{1}{2} \times \log(\frac{1-0.1347}{0.1347}) = 0.92$$

Final Classifier



Voted Combination of Classifiers

- The general problem here is to try to combine many simple “weak" classifiers into a single “strong" classifier.
- We consider voted combinations of simple binary component classifiers

$$H_{final}(x) = \alpha_1 h(x; \theta_1) + \cdots + \alpha_T h(x; \theta_T)$$

- Where θ is the model parameter and the non-negative votes α_i can be used to emphasize component classifiers that are more reliable than others.

Random Forests

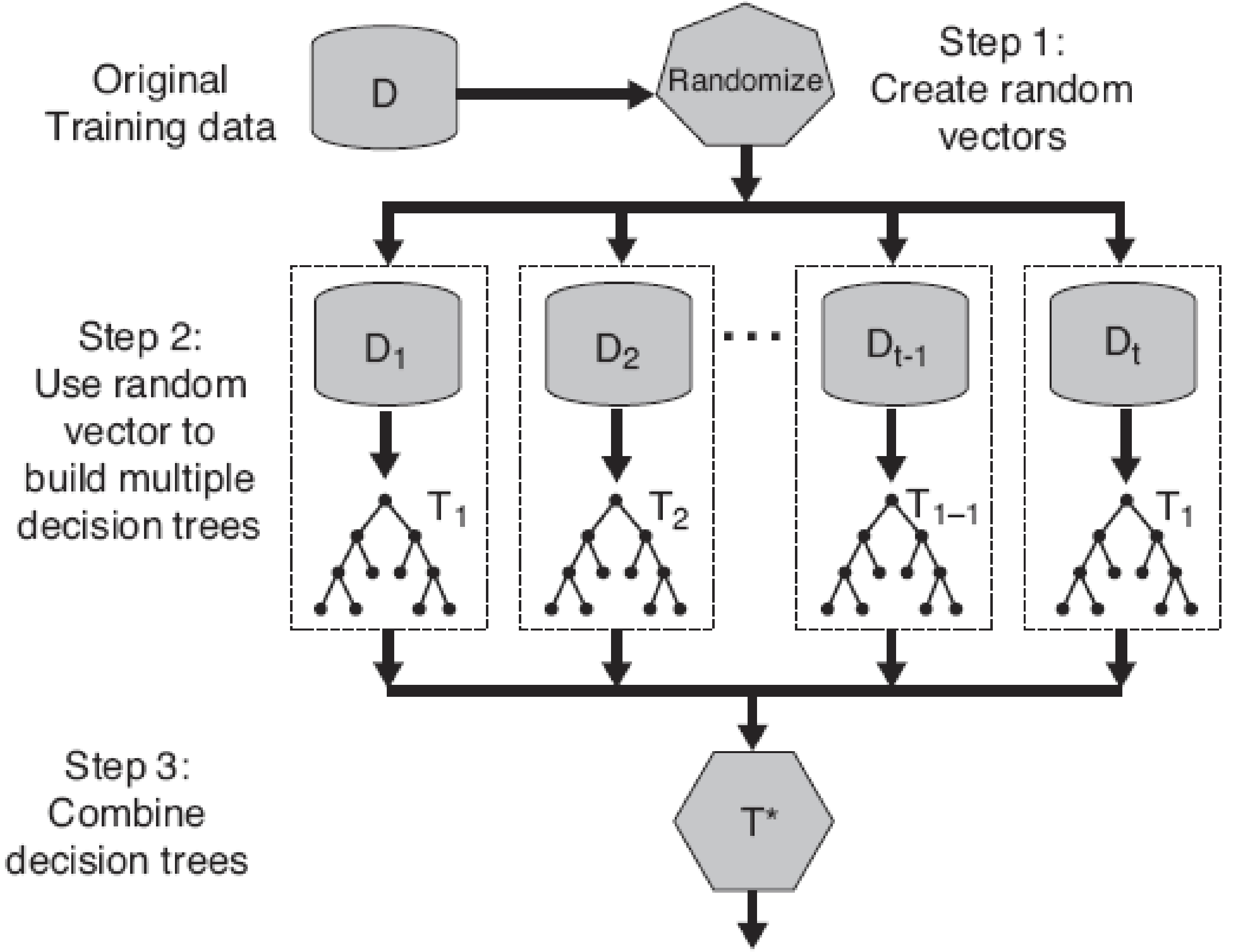
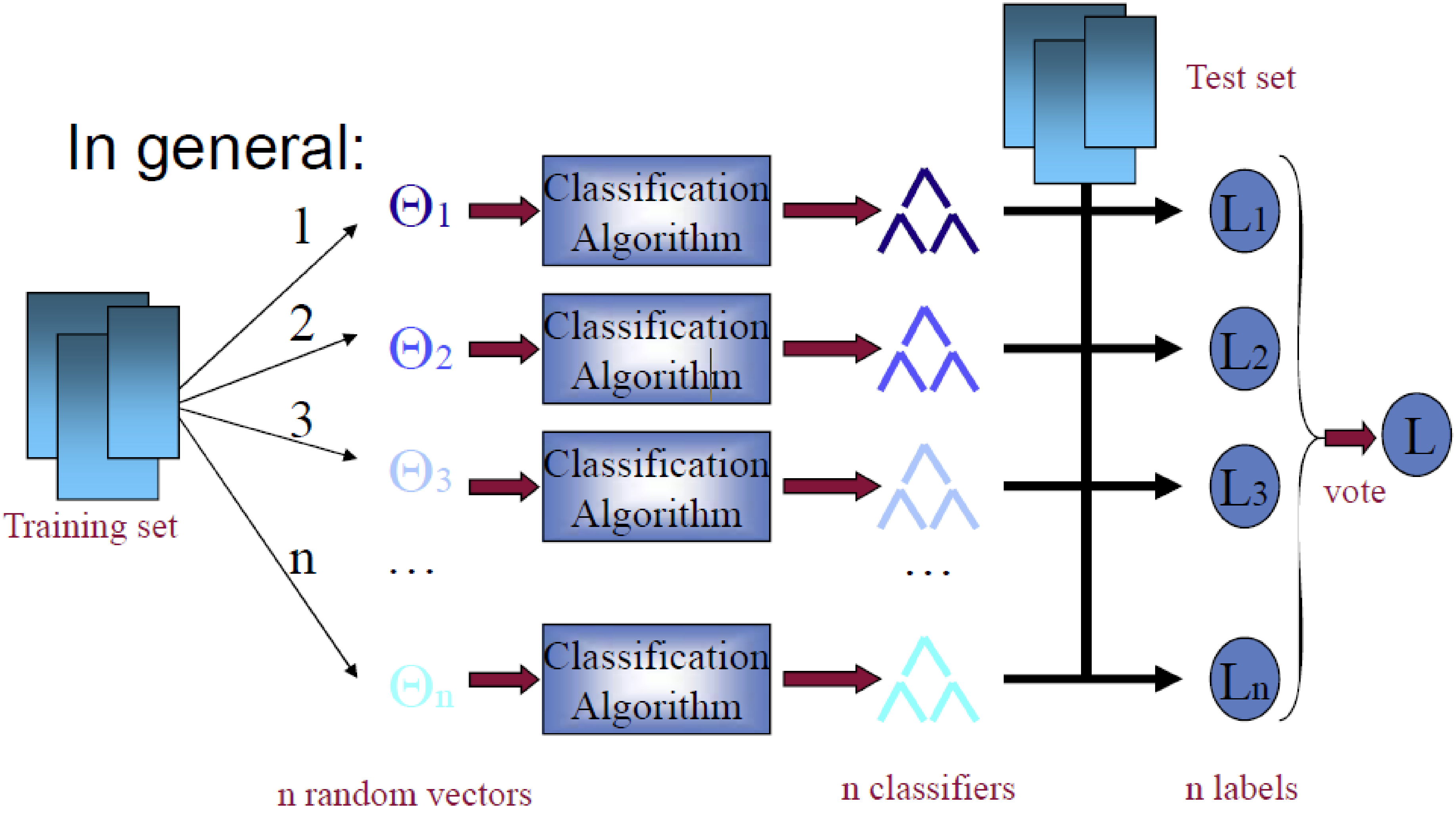


Figure 5.40. Random forests.

Random Forests



Random Forests

- Ensemble method specifically designed for decision tree classifiers
- Two sources of randomness: “bagging” and “random input vectors”
- Use bootstrap aggregation to train many decision trees
  - Randomly subsample N examples
  - Train decision tree on subsample
  - Use average or majority vote among learned trees as prediction
- Also randomly subsample features: best split at each node is chosen from a random sample of m attributes instead of all attributes

Random Forests Algorithm

- For $b = 1$ to B
  - Draw a bootstrap sample of size N from the data
  - Grow a tree T_b using the bootstrap sample as follows
    - Choose m attributes uniformly at random from the data
    - Choose the best attribute among the m to split on
    - Split on the best attribute and recurse until partitions have fewer than s_{min} number of nodes
- Prediction for a new data point x
  - Regression: $\frac{1}{B}\sum_b T_b(x)$
  - Classification: choose the majority class label among $T_1(x), \ldots, T_B(x)$.

Readings

1. https://hunch.net/~coms-4771/quinlan.pdf
2. https://arxiv.org/abs/1603.02754
3. https://www.jstor.org/stable/2699986?refreqid=excelsior%3A4a12047a831ef2f51e03a13d2a4e52ee

Summary of Today's Lecture

- Decision Tree
- Ensemble Methods
  - Bagging
  - Boosting
  - Random Forests