CSE 5243 INTRO. TO DATA MINING

Classification (Basic Concepts)

Yu Su, CSE@The Ohio State University

Slides adapted from UIUC CS412 by Prof. Jiawei Han and OSU CSE5243 by Prof. Huan Sun

Classification: Basic Concepts

- Classification: Basic Concepts
- Decision Tree Induction
- Model Evaluation and Selection
- Practical Issues of Classification

This class

- Bayes Classification Methods
- Techniques to Improve Classification Accuracy: Ensemble Methods
- Summary

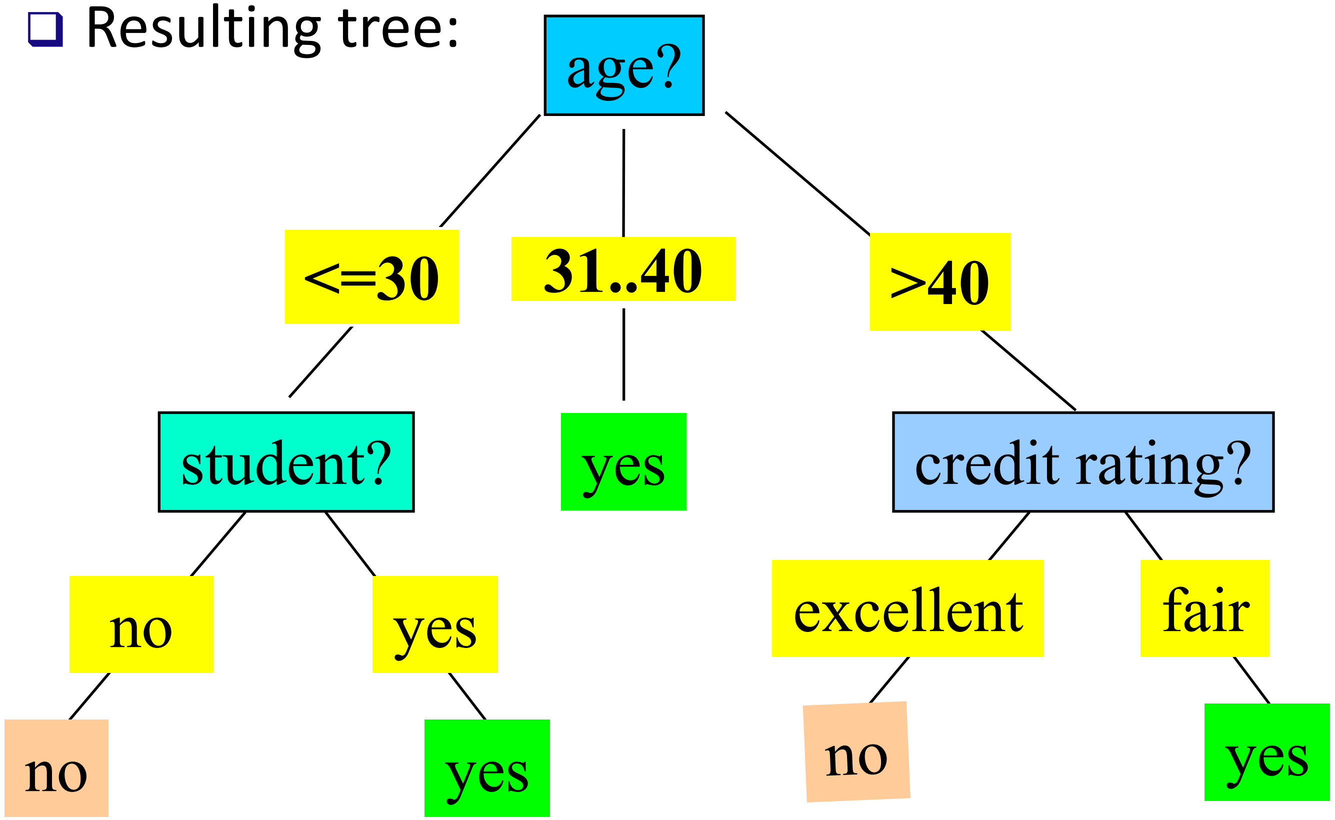
Next class

Classification: Basic Concepts

- Classification: Basic Concepts
- Decision Tree Induction
- Model Evaluation and Selection
- Practical Issues of Classification
- Bayes Classification Methods
- Techniques to Improve Classification Accuracy: Ensemble Methods
- Summary

Decision Tree Induction: An Example

- Training data set: Buys_computer
- The data set follows an example of Quinlan's ID3 (Playing Tennis)
- Resulting tree:



age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

Algorithm for Decision Tree Induction

- Basic algorithm (a greedy algorithm)
  - Tree is constructed in a **top-down recursive divide-and-conquer manner**
  - At start, all the training examples are at the root
  - Examples are partitioned recursively based on selected attributes
  - Test attributes are selected on the basis of a heuristic or statistical measure (e.g., **information gain**)
- Conditions for stopping partitioning
  - All examples for a given node belong to the same class, or
  - There are no remaining attributes for further partitioning—**majority voting** is employed for classifying the leaf, or
  - There are no examples left

Algorithm Outline

- Split (node, {data tuples})
  - A ← the best attribute for splitting the {data tuples}
  - Decision attribute for this node ← A
  - For each value of A, create new child node
  - For each child node / subset:
    - If one of the stopping conditions is satisfied: STOP
    - Else: Split (child_node, {subset})

ID3 algorithm: how it works
https://www.youtube.com/watch?v=_XhOdSLlE5c

Algorithm Outline

- Split (node, {data tuples})
  - A ← the best attribute for splitting the {data tuples}
  - Decision attribute for this node ← A
  - For each value of A, create new child node
  - For each child node / subset:
    - If one of the stopping conditions is satisfied: STOP
    - Else: Split (child_node, {subset})

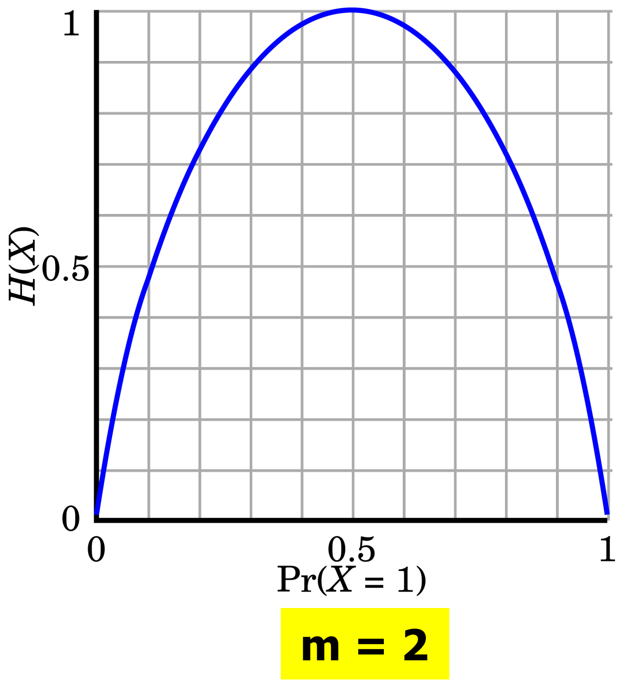
ID3 algorithm: how it works
https://www.youtube.com/watch?v=_XhOdSLIE5c

Brief Review of Entropy

- Entropy (Information Theory)
  - A measure of uncertainty associated with a random variable
  - Calculation: For a discrete random variable Y taking m distinct values $\{y_1, y_2, \ldots, y_m\}$

$$H(Y) = -\sum_{i=1}^{m} p_i \log(p_i) \quad where\ p_i = P(Y = y_i)$$

  - Interpretation
    - Higher entropy → higher uncertainty
    - Lower entropy → lower uncertainty



- Conditional entropy

$$H(Y|X) = \sum_{x} p(x) H(Y|X = x)$$

Attribute Selection Measure: Information Gain (ID3/C4.5)

- Select the attribute with the highest information gain
- Let p_i be the probability that an arbitrary tuple in D belongs to class C_i, estimated by $|C_{i,D}|/|D|$
- Expected information (entropy) needed to classify a tuple in D:

$$Info(D) = -\sum_{i=1}^{m} p_i \log_2(p_i)$$

- Information needed (after using A to split D into v partitions) to classify D:

$$Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times Info(D_j)$$

- Information gained by branching on attribute A

$$Gain(A) = Info(D) - Info_A(D)$$

Attribute Selection: Information Gain

- Class P: buys_computer = "yes"
- Class N: buys_computer = "no"

$$Info(D) = I(9,5) = -\frac{9}{14}\log_2(\frac{9}{14}) - \frac{5}{14}\log_2(\frac{5}{14}) = 0.940$$

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

Look at "age":

age	p_i	n_i	$I(p_i, n_i)$
<=30	2	3	0.971
31…40	4	0	0
>40	3	2	0.971

$$Info_{age}(D) = \frac{5}{14} I(2,3) + \frac{4}{14} I(4,0) + \frac{5}{14} I(3,2) = 0.694$$

Attribute Selection: Information Gain

- Class P: buys_computer = "yes"
- Class N: buys_computer = "no"

$$Info(D) = I(9,5) = -\frac{9}{14}\log_2(\frac{9}{14}) - \frac{5}{14}\log_2(\frac{5}{14}) = 0.940$$

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

$$Info_{age}(D) = \frac{5}{14}I(2,3) + \frac{4}{14}I(4,0) + \frac{5}{14}I(3,2) = 0.694$$

$$Gain(age) = Info(D) - Info_{age}(D) = 0.246$$

Similarly,

$$Gain(income) = 0.029$$

$$Gain(student) = 0.151$$

$$Gain(credit\_rating) = 0.048$$

Recursive Procedure

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

1. After selecting age at the root node, we will create three child nodes.

2. One child node is associated with red data tuples.

3. How to continue for this child node?

Now, you will make D = {red data tuples}

and then select the best attribute to further split D.

A recursive procedure.

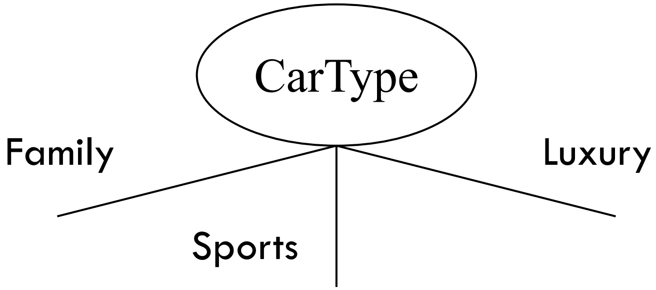
How to Select Test Attribute?

- Depends on attribute types
  - Nominal
  - Ordinal
  - Continuous

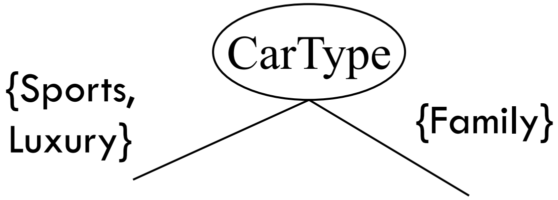
- Depends on number of ways to split
  - 2-way split
  - Multi-way split

Splitting Based on Nominal Attributes

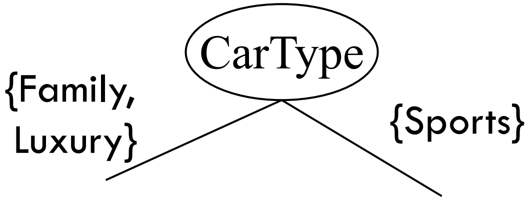
- Multi-way split: Use as many partitions as distinct values.



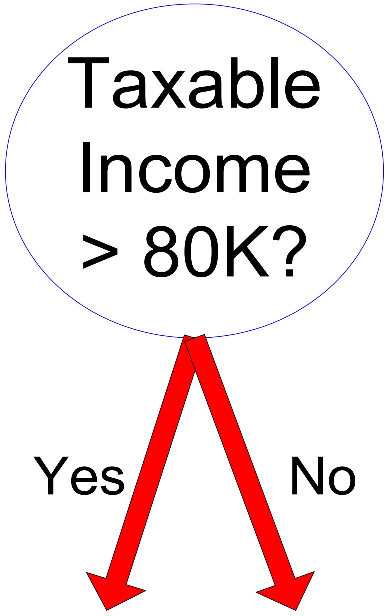
- Binary split: Divides values into two subsets. Need to find optimal partitioning.



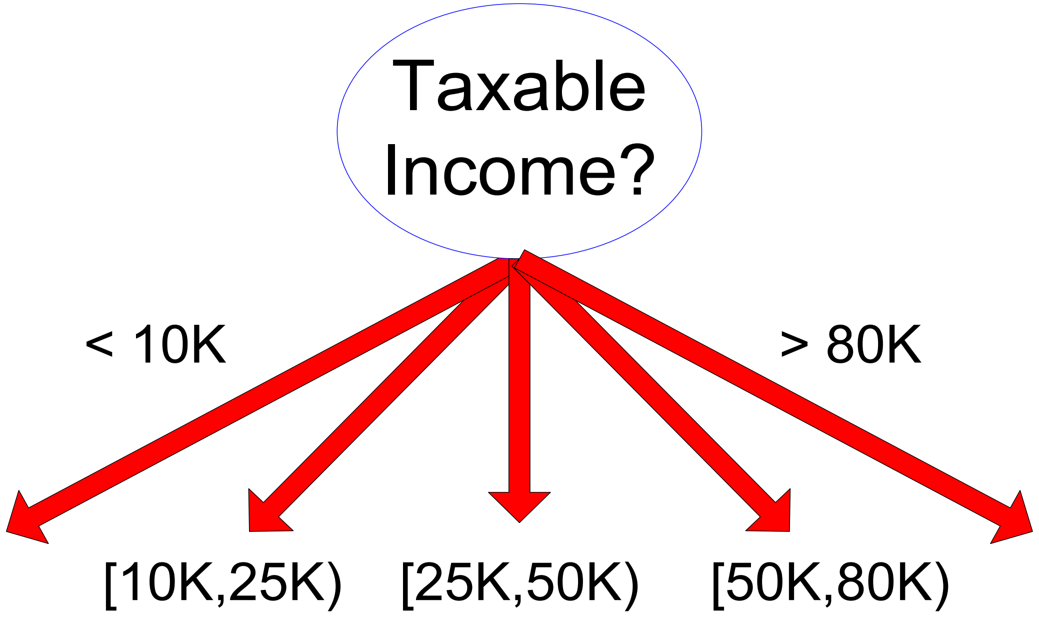
OR



Splitting Based on Continuous Attributes



(i) Binary split



(ii) Multi-way split

How to Determine the Best Split

- Greedy approach:
  - Nodes with homogeneous class distribution are preferred

Ideally, data tuples at that node belong to the same class.

C0: 5 C1: 5	C0: 9 C1: 1
Non-homogeneous, High degree of impurity	Homogeneous, Low degree of impurity

Rethink about Decision Tree Classification

- Greedy approach:
  - Nodes with homogeneous class distribution are preferred

- Need a measure of node impurity:

C0: 5 C1: 5	C0: 9 C1: 1
Non-homogeneous, High degree of impurity	Homogeneous, Low degree of impurity

Measures of Node Impurity

- Entropy: $$H(Y) = -\sum_{i=1}^{m} p_i \log(p_i) \quad where\ p_i = P(Y = y_i)$$
  - Higher entropy => higher uncertainty, higher node impurity
- Gini Index
- Misclassification error

Gain Ratio for Attribute Selection (C4.5)

- Information gain measure is biased towards attributes with a large number of values
- C4.5 (a successor of ID3) uses **gain ratio** to overcome the problem (normalization to information gain)

$$SplitInfo_A(D) = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} \times \log_2\left(\frac{|D_j|}{|D|}\right)$$

  - The entropy of the partitioning, or the potential information generated by splitting *D* into *v* partitions.
  - **GainRatio(A) = Gain(A)/SplitInfo(A)** (normalizing Information Gain)

Gain Ratio for Attribute Selection (C4.5)

- C4.5 (a successor of ID3) uses gain ratio to overcome the problem (normalization to information gain)

$$SplitInfo_A(D) = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} \times \log_2\left(\frac{|D_j|}{|D|}\right)$$

  - GainRatio(A) = Gain(A)/SplitInfo(A)
- Ex.

$$\boxed{Gain(income) = 0.029}$$

$$SplitInfo_{income}(D) = -\frac{4}{14} \times \log_2\left(\frac{4}{14}\right) - \frac{6}{14} \times \log_2\left(\frac{6}{14}\right) - \frac{4}{14} \times \log_2\left(\frac{4}{14}\right) = 1.557$$

  - gain_ratio(income) = 0.029/1.557 = 0.019
- The attribute with the **maximum gain ratio** is selected as the splitting attribute

Gini Index (CART, IBM IntelligentMiner)

- If a data set D contains examples from n classes, gini index, $gini(D)$ is defined as

  $gini(D) = 1 - \sum_{j=1}^{n} p_j^2$, where p_j is the relative frequency of class j in D

- If a data set *D* is split on A into two subsets D_1 and D_2, the *gini* index *after the split* is defined as:

$$gini_A(D) = \frac{|D_1|}{|D|} gini(D_1) + \frac{|D_2|}{|D|} gini(D_2)$$

- Reduction in impurity:

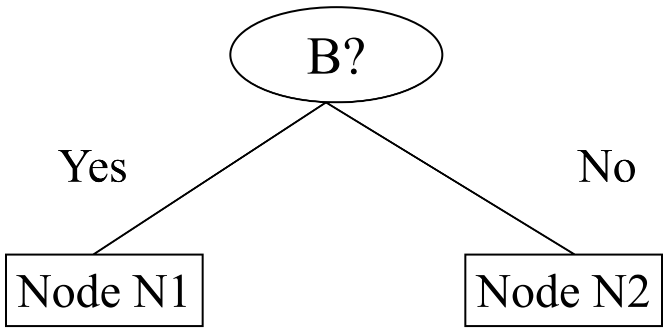
$$\Delta gini(A) = gini(D) - gini_A(D)$$

- The attribute provides the smallest $gini_A(D)$ (or, the largest reduction in impurity) is chosen to split the node.

Binary Attributes: Computing Gini Index

$$gini(D) = 1 - \sum_{j=1}^{n} p_j^2$$

- Splits into two partitions
- Effect of weighing partitions:
  - Larger and Purer Partitions are sought for.

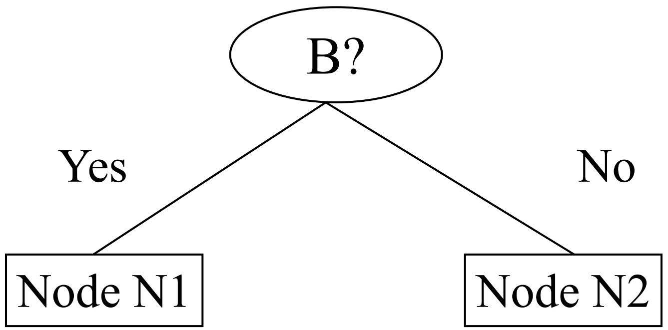


	Parent
C1	**6**
C2	**6**
Gini = ?	

Binary Attributes: Computing Gini Index

- Splits into two partitions
- Effect of weighting partitions:
  - Larger and Purer Partitions are sought for.

$$gini(D) = 1 - \sum_{j=1}^{n} p_j^2$$



	Parent
C1	**6**
C2	**6**
Gini = 0.500	

Gini(N1)
$= 1 - (5/7)^2 - (2/7)^2$
$= 0.194$

Gini(N2)
$= 1 - (1/5)^2 - (4/5)^2$
$= 0.528$

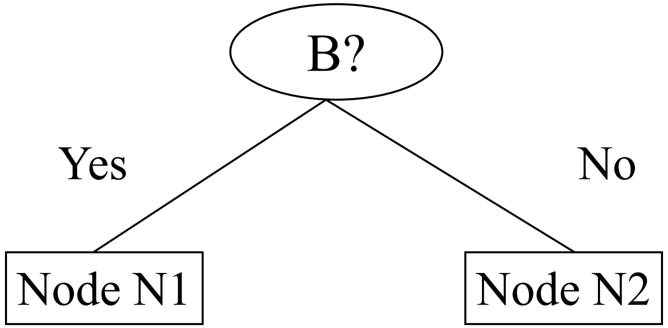
	N1	**N2**
C1	**5**	**1**
C2	**2**	**4**
Gini=?		

Binary Attributes: Computing Gini Index

- Splits into two partitions
- Effect of weighting partitions:
  - Prefer Larger and Purer Partitions.

$$gini(D) = 1 - \sum_{j=1}^{n} p_j^2$$

	Parent
C1	**6**
C2	**6**
Gini = ?	



Gini(N1)
$= 1 - (5/7)^2 - (2/7)^2$
$= 0.194$

Gini(N2)
$= 1 - (1/5)^2 - (4/5)^2$
$= 0.528$

	N1	N2
C1	**5**	**1**
C2	**2**	**4**
Gini=0.333		

Gini(Children)
$= 7/12 * 0.194 +$
$5/12 * 0.528$
$= 0.333$

weighting

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	2	1
C2	4	1	1
Gini	0.393		

Two-way split
(find best partition of values)

	CarType	
	{Sports, Luxury}	{Family}
C1	3	1
C2	2	4
Gini	0.400	

	CarType	
	{Sports}	{Family, Luxury}
C1	2	2
C2	1	5
Gini	0.419	

Continuous Attributes: Computing Gini Index or Information Gain

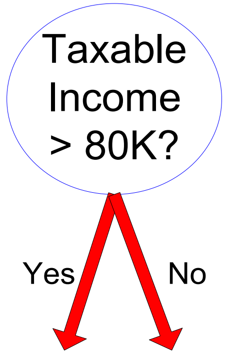
- **To discretize the attribute values**
  - **Use Binary Decisions based on one splitting value**

- Several Choices for the splitting value
  - Number of possible splitting values = Number of distinct values -1
  - Typically, the midpoint between each pair of adjacent values is considered as a possible *split point*
    - $(a_i+a_{i+1})/2$ is the midpoint between the values of a_i and a_{i+1}

- Each splitting value has a count matrix associated with it
  - 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.

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



Continuous Attributes: Computing Gini Index or expected information requirement

First decide the splitting value to discretize the attribute:

- For efficient computation: for each attribute,

  Step 1: Sort the attribute on values

Step 1:

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
Possible Splitting Values →	65	72	80	87	92	97	110	122	172	

← Use midpoint

Continuous Attributes:
Computing Gini Index or expected information requirement

First decide the splitting value to discretize the attribute:

- For efficient computation: for each attribute,
  - Step 1: Sort the attribute on values
  - Step 2: **Linearly scan these values, each time updating the count matrix**

Step 1:

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
Possible Splitting Values →	65									

Step 2:

	65: <=	65: >
Yes	0	3
No	1	6

For each splitting value, get its count matrix: how many data tuples have: **(a)** Taxable income <=65 with class label "Yes" , **(b)** Taxable income <=65 with class label "No", **(c)** Taxable income >65 with class label "Yes", **(d)** Taxable income >65 with class label "No".

Continuous Attributes: Computing Gini Index or expected information requirement

First decide the splitting value to discretize the attribute:

- For efficient computation: for each attribute,

  Step 1: Sort the attribute on values

  Step 2: **Linearly scan these values, each time updating the count matrix**

Step 1:

Sorted Values →

Possible Splitting Values →

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

Step 2:

Possible Splitting Values	65		72	
	<=	>	<=	>
Yes	0	3	0	3
No	1	6	2	5

For each splitting value, get its count matrix: how many data tuples have: **(a)** Taxable income <=72 with class label "Yes" , **(b)** Taxable income <=72 with class label "No", **(c)** Taxable income >72 with class label "Yes", **(d)** Taxable income >72 with class label "No".

Continuous Attributes: Computing Gini Index or expected information requirement

First decide the splitting value to discretize the attribute:

- For efficient computation: for each attribute,
  - Step 1: Sort the attribute on values
  - Step 2: **Linearly scan these values, each time updating the count matrix**

Step 1:

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

Possible Splitting Values →

Step 2:

	65		72		80	
	<=	>	<=	>	<=	>
Yes	0	3	0	3	0	3
No	1	6	2	5	3	4

For each splitting value, get its count matrix: how many data tuples have: **(a)** Taxable income <=80 with class label "Yes" , **(b)** Taxable income <=80 with class label "No", **(c)** Taxable income >80 with class label "Yes", **(d)** Taxable income >80 with class label "No".

Continuous Attributes: Computing Gini Index or expected information requirement

First decide the splitting value to discretize the attribute:

- For efficient computation: for each attribute,
  - Step 1: Sort the attribute on values
  - Step 2: **Linearly scan these values, each time updating the count matrix**

Step 1:

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

Possible Splitting Values →

Step 2:

	65		72		80		87		92		97		110		122		172	
	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>
Yes	0	3	0	3	0	3	1	2	2	1	3	0	3	0	3	0	3	0
No	1	6	2	5	3	4	3	4	3	4	3	4	4	3	5	2	6	1

For each splitting value, get its count matrix: how many data tuples have: **(a)** Taxable income <=172 with class label "Yes" , **(b)** Taxable income <=172 with class label "No", **(c)** Taxable income >172 with class label "Yes", **(d)** Taxable income >172 with class label "No".

Continuous Attributes: Computing Gini Index or expected information requirement

First decide the splitting value to discretize the attribute:

- For efficient computation: for each attribute,
  - Step 1: Sort the attribute on values
  - Step 2: Linearly scan these values, each time updating the count matrix
  - Step 3: **Computing Gini index and choose the split position that has the least Gini index**

Step 1: Sorted Values → ; Possible Splitting Values →

Step 2:

Step 3:

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
Possible Splitting Values	65									
	<=	>								
Yes	0	3								
No	1	6								
Gini	0.400									

For each splitting value v (e.g., 65), compute its Gini index:

$$gini_{Taxable\_Income}(D) = \frac{|D_1|}{|D|} gini(D_1) + \frac{|D_2|}{|D|} gini(D_2)$$

Here D1 and D2 are two partitions based on v: D1 has taxable income <=v and D2 has >v

Continuous Attributes: Computing Gini Index or expected information requirement

First decide the splitting value to discretize the attribute:

- For efficient computation: for each attribute,
  - Step 1: Sort the attribute on values
  - Step 2: Linearly scan these values, each time updating the count matrix
  - Step 3: **Computing Gini index and choose the split position that has the least Gini index**

Step 1: Sorted Values → ; Possible Splitting Values →

Step 2:

Step 3:

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

	65		72	
	<=	>	<=	>
Yes	0	3	0	3
No	1	6	2	5
Gini	0.400		0.375	

For each splitting value v (e.g., 72), compute its Gini index:

$$gini_{Taxable\_Income}(D) = \frac{|D_1|}{|D|} gini(D_1) + \frac{|D_2|}{|D|} gini(D_2)$$

Here D1 and D2 are two partitions based on v: D1 has taxable income <=v and D2 has >v

Continuous Attributes:
Computing Gini Index or expected information requirement

First decide the splitting value to discretize the attribute:

- For efficient computation: for each attribute,
  - Step 1: Sort the attribute on values
  - Step 2: Linearly scan these values, each time updating the count matrix
  - Step 3: **Computing Gini index and choose the split position that has the least Gini index**

	Cheat	No	No	No	Yes	Yes	Yes	No	No	No	No
		Taxable Income									
Step 1: Sorted Values →		60	70	75	85	90	95	100	120	125	220

		65		72		80		87		92		97		110		122		172	
Step 1: Possible Splitting Values →		<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>
Step 2:	Yes	0	3	0	3	0	3	1	2	2	1	3	0	3	0	3	0	3	0
	No	1	6	2	5	3	4	3	4	3	4	3	4	4	3	5	2	6	1
Step 3:	Gini	0.400		0.375		0.343		0.417		0.400		***0.300***		0.343		0.375		0.400	

Choose this splitting value (=97) with the least Gini index to discretize Taxable Income

Continuous Attributes:
Computing Gini Index or expected information requirement

First decide the splitting value to discretize the attribute:

- For efficient computation: for each attribute,
  - Step 1: Sort the attribute on values
  - Step 2: Linearly scan these values, each time updating the count matrix
  - Step 3: **Computing expected information requirement and choose the split position that has the least value**

Step 1:

Sorted Values →

Possible Splitting Values →

Step 2:

Step 3:

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

Possible Splitting Values	65		72		80		87		92		97		110		122		172	
	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>
Yes	0	3	0	3	0	3	1	2	2	1	3	0	3	0	3	0	3	0
No	1	6	2	5	3	4	3	4	3	4	3	4	4	3	5	2	6	1
Info	?		?		?		?		?		?		?		?		?	

If Information Gain is used for attribute selection,

Similarly to calculating Gini index, for each splitting value, compute **Info_{Taxable Income}**:

$$Info_{Taxable-Income}(D) = \sum_{j=1}^{2} \frac{|D_j|}{|D|} \times Info(D_j)$$

Continuous Attributes: Computing Gini Index or expected information requirement

First decide the splitting value to discretize the attribute:

- For efficient computation: for each attribute,

  Step 1: Sort the attribute on values

  Step 2: Linearly scan these values, each time updating the count matrix

  Step 3: **Computing Gini index and choose the split position that has the least Gini index**

	Cheat	No	No	No	Yes	Yes	Yes	No	No	No	No
		Taxable Income									
Step 1:	Sorted Values	60	70	75	85	90	95	100	120	125	220

		65		72		80		87		92		97		110		122		172	
Step 1:	Possible Splitting Values	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>
Step 2:	Yes	0	3	0	3	0	3	1	2	2	1	3	0	3	0	3	0	3	0
	No	1	6	2	5	3	4	3	4	3	4	3	4	4	3	5	2	6	1
Step 3:	Gini	0.400		0.375		0.343		0.417		0.400		*0.300*		0.343		0.375		0.400	

Choose this splitting value (=97 here) *with the least Gini index or expected information requirement* to discretize Taxable Income

Continuous Attributes: Computing Gini Index or expected information requirement

First decide the splitting value to discretize the attribute:

- For efficient computation: for each attribute,

  Step 1: Sort the attribute on values

  Step 2: Linearly scan these values, each time updating the count matrix

  Step 3: **Computing Gini index and choose the split position that has the least Gini index**

Step 1: Sorted Values →

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

Step 1: Possible Splitting Values →

Step 2:

Step 3:

Possible Splitting Values	65		72		80		87		92		97		110		122		172	
	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>
Yes	0	3	0	3	0	3	1	2	2	1	3	0	3	0	3	0	3	0
No	1	6	2	5	3	4	3	4	3	4	3	4	4	3	5	2	6	1
Gini	0.400		0.375		0.343		0.417		0.400		*0.300*		0.343		0.375		0.400	

At each level of the decision tree, for attribute selection, (1) First, discretize a continuous attribute by deciding the splitting value; (2) Then, compare the discretized attribute with other attributes in terms of Gini Index reduction or Information Gain.

Continuous Attributes: Computing Gini Index or expected information requirement

First decide the splitting value to discretize the attribute:

For each attribute, only scan the data tuples once

- For efficient computation: for each attribute,
  - Step 1: Sort the attribute on values
  - Step 2: Linearly scan these values, each time updating the count matrix
  - Step 3: **Computing Gini index and choose the split position that has the least Gini index**

Step 1: Sorted Values → ; Possible Splitting Values →

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

Possible Splitting Values	65		72		80		87		92		97		110		122		172	
	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>
Yes	0	3	0	3	0	3	1	2	2	1	3	0	3	0	3	0	3	0
No	1	6	2	5	3	4	3	4	3	4	3	4	4	3	5	2	6	1
Gini	0.400		0.375		0.343		0.417		0.400		*0.300*		0.343		0.375		0.400	

Step 2:

Step 3:

At each level of the decision tree, for attribute selection, (1) First, discretize a continuous attribute by deciding the splitting value; (2) Then, compare the discretized attribute with other attributes in terms of Gini Index reduction or Information Gain.

Another Impurity Measure: Misclassification Error

- Classification error at a node t :

$$\boxed{Error(t) = 1 - \max_{i} P(i \mid t)}$$

  - $P(i \mid t)$ means the relative frequency of class i at node t.

- Measures misclassification error made by a node.
  - Maximum ($1 - 1/n_c$) when records are equally distributed among all classes, implying most impurity
  - Minimum (0.0) when all records belong to one class, implying least impurity

Examples for 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

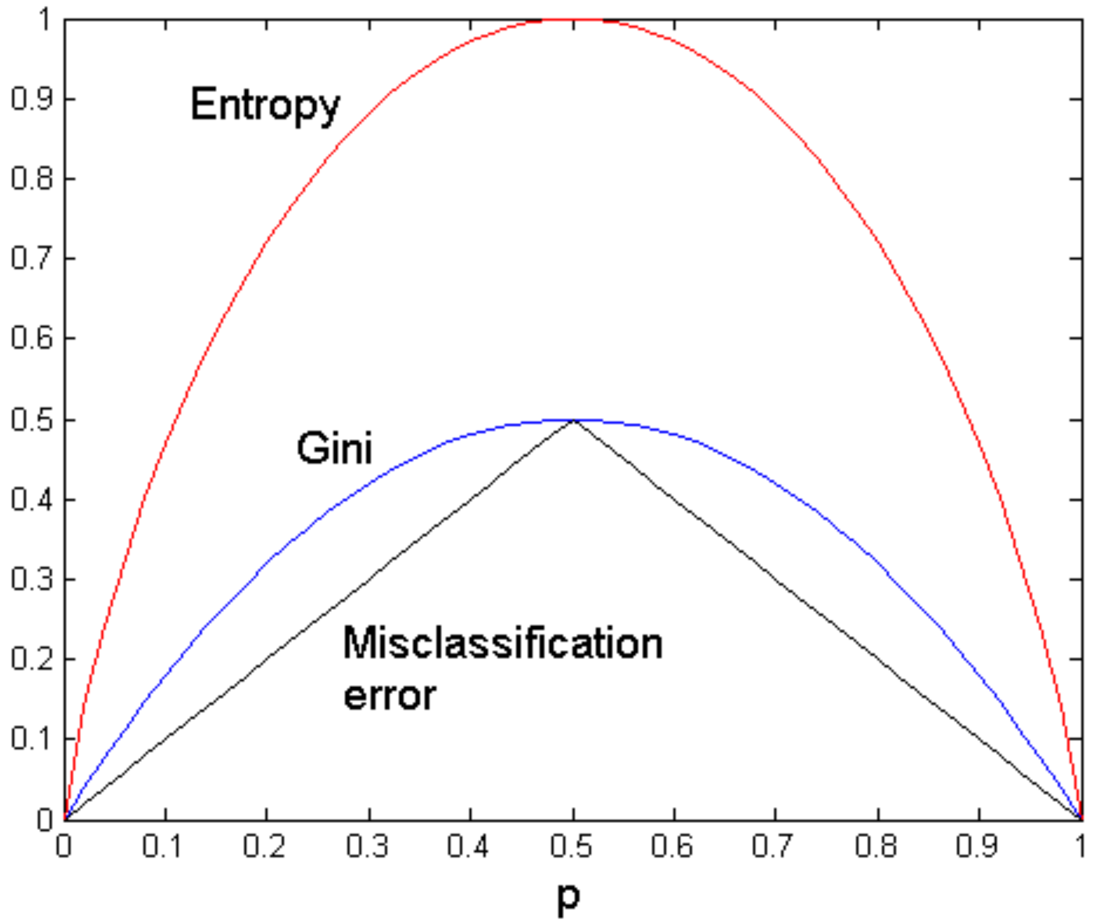
C1	**2**
C2	**4**

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

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

Comparison among Impurity Measure

For a 2-class problem:



$$\boxed{Entropy = -p\log(p) - (1-p)\log(1-p)}$$

$$\boxed{Gini = 1 - p^2 - (1-p)^2}$$

$$\boxed{Error = 1 - \max(p, 1-p)}$$

Other Attribute Selection Measures

- CHAID: a popular decision tree algorithm, measure based on χ^2 test for independence
- C-SEP: performs better than info. gain and gini index in certain cases
- G-statistic: has a close approximation to χ^2 distribution
- MDL (Minimal Description Length) principle (i.e., the simplest solution is preferred):
  - The best tree as the one that requires the fewest # of bits to both (1) encode the tree, and (2) encode the exceptions to the tree
- Multivariate splits (partition based on multiple variable combinations)
  - CART: finds multivariate splits based on a linear comb. of attrs.
- Which attribute selection measure is the best?
  - Most give good results, none is significantly superior than others

Decision Tree Based Classification

- Advantages:
  - 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

Example: C4.5

- Simple depth-first construction.
- Uses Information Gain Ratio
- Sorts Continuous Attributes at each node.
- Needs entire data to fit in memory.
- Unsuitable for Large Datasets.
  - Needs out-of-core sorting.

- You can download the software online, e.g., http://www2.cs.uregina.ca/~dbd/cs831/notes/ml/dtrees/c4.5/tutorial.html

Classification: Basic Concepts

- Classification: Basic Concepts
- Decision Tree Induction
- Model Evaluation and Selection
- Practical Issues of Classification
- Bayes Classification Methods
- Techniques to Improve Classification Accuracy: Ensemble Methods
- Summary

Model Evaluation

- Metrics for Performance Evaluation
  - How to evaluate the performance of a model?
- Methods for Performance Evaluation
  - How to obtain reliable estimates?
- Methods for Model Comparison
  - How to compare the relative performance among competing models?

Metrics for Performance Evaluation

- Focus on the predictive capability of a model
  - Rather than how fast it takes to classify or build models, scalability, etc.
- Confusion Matrix:

	PREDICTED CLASS		
		Class=Yes	Class=No
ACTUAL CLASS	Class=Yes	a	b
	Class=No	c	d

a: TP (true positive)

b: FN (false negative)

c: FP (false positive)

d: TN (true negative)

Classifier Evaluation Metrics: Confusion Matrix

Confusion Matrix:

Actual class\Predicted class	C_1	$\neg C_1$
C_1	**True Positives (TP)**	**False Negatives (FN)**
$\neg C_1$	**False Positives (FP)**	**True Negatives (TN)**

Example of Confusion Matrix:

Actual class\Predicted class	buy_computer = yes	buy_computer = no	Total
buy_computer = yes	**6954**	**46**	7000
buy_computer = no	**412**	**2588**	3000
Total	7366	2634	10000

- Given *m* classes, an entry, $CM_{i,j}$ in a **confusion matrix** indicates # of tuples in class *i* that were labeled by the classifier as class *j*
  - May have extra rows/columns to provide totals

Classifier Evaluation Metrics: Accuracy, Error Rate

A\P	C	¬C	
C	**TP**	**FN**	**P**
¬C	**FP**	**TN**	**N**
	P'	**N'**	**All**

- **Classifier Accuracy,** or recognition rate: percentage of test set tuples that are correctly classified

  Accuracy = (TP + TN)/All

- **Error rate:** *1 – accuracy*, or

  Error rate = (FP + FN)/All

Limitation of Accuracy

- Consider a 2-class problem
  - Number of Class 0 examples = 9990
  - Number of Class 1 examples = 10
- If a model predicts everything to be class 0,

  Accuracy is 9990/10000 = 99.9 %

  - Accuracy is misleading because model does not detect any class 1 example

Cost Matrix

	PREDICTED CLASS		
ACTUAL CLASS	$C(i \mid j)$	**Class=Yes**	**Class=No**
	Class=Yes	*C(Yes|Yes)*	C(No|Yes)
	Class=No	C(Yes|No)	C(No|No)

$C(i \mid j)$: Cost of misclassifying one class j example as class i

Computing Cost of Classification

Cost Matrix	PREDICTED CLASS		
	C(i|j)	+	-
ACTUAL CLASS	+	-1	100
	-	1	0

Model M_1	PREDICTED CLASS		
		+	-
ACTUAL CLASS	+	150	40
	-	60	250

Accuracy = 80%

Cost = 3910

Model M_2	PREDICTED CLASS		
		+	-
ACTUAL CLASS	+	250	45
	-	5	200

Accuracy = 90%

Cost = 4255

Cost-Sensitive Measures

$$\text{Precision (p)} = \frac{a}{a+c}$$

$$\text{Recall (r)} = \frac{a}{a+b}$$

$$\text{F - measure (F)} = \frac{2rp}{r+p} = \frac{2a}{2a+b+c}$$

		PREDICTED CLASS	
ACTUAL CLASS		Class=Yes	Class=No
	Class=Yes	a (TP)	b (FN)
	Class=No	c (FP)	d (TN)

- Precision is biased towards C(Yes|Yes) & C(Yes|No)
- Recall is biased towards C(Yes|Yes) & C(No|Yes)
- F-measure is biased towards all except C(No|No)

$$\text{Weighted Accuracy} = \frac{w_1 a + w_4 d}{w_1 a + w_2 b + w_3 c + w_4 d}$$

Classifier Evaluation Metrics: **Sensitivity and Specificity**

A\P	C	¬C	
C	**TP**	**FN**	**P**
¬C	**FP**	**TN**	**N**
	P'	**N'**	**All**

- Classifier Accuracy, or recognition rate: percentage of test set tuples that are correctly classified

  Accuracy = (TP + TN)/All

- Error rate: *1 – accuracy*, or

  Error rate = (FP + FN)/All

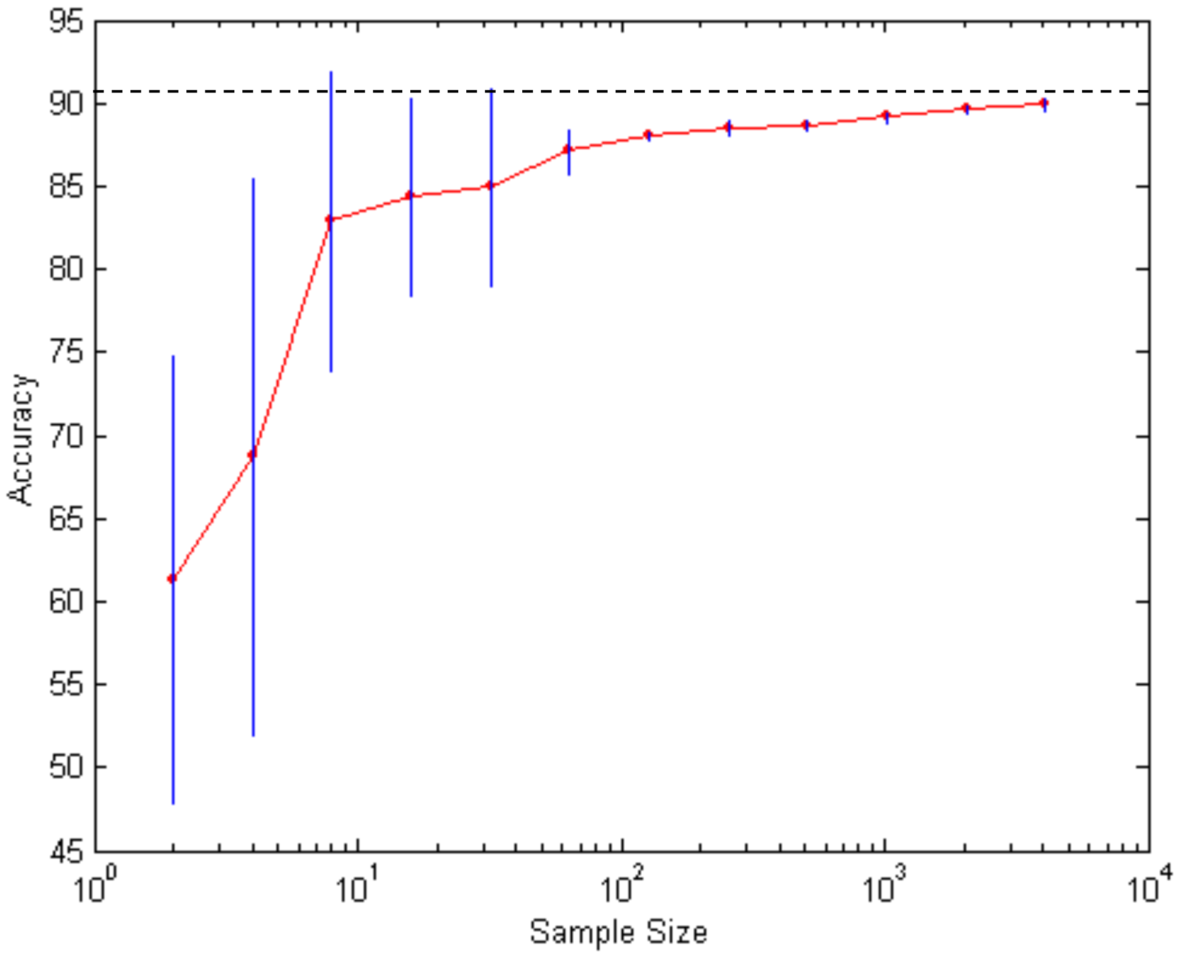
- **Class Imbalance Problem:**
  - One class may be *rare*, e.g. fraud, or HIV-positive
  - Significant *majority of the negative class* and minority of the positive class
  - **Sensitivity:** True Positive recognition rate
    - **Sensitivity = TP/P**
  - **Specificity:** True Negative recognition rate
    - **Specificity = TN/N**

Methods for Performance Evaluation

- How to obtain a reliable estimate of performance?

- Performance of a model may depend on other factors besides the learning algorithm:
  - Class distribution
  - Cost of misclassification
  - Size of training and test sets

Learning Curve



- Learning curve shows how accuracy changes with varying sample size

- Requires a sampling schedule for creating learning curve:
  - Arithmetic sampling (Langley, et al)
  - Geometric sampling (Provost et al)

Effect of small sample size:

- Bias in the estimate
- Variance of estimate

Methods of Estimation

- Holdout
  - E.g., reserve 2/3 for training and 1/3 for testing
- Random subsampling
  - Repeated holdout
- Cross validation
  - Partition data into k disjoint subsets
  - k-fold: train on k-1 partitions, test on the remaining one
  - Leave-one-out: k=n
- Stratified sampling
  - oversampling vs undersampling
- Bootstrap
  - Sampling with replacement

Evaluating Classifier Accuracy: Holdout & Cross-Validation Methods

- **Holdout method**
  - Given data is randomly partitioned into two independent sets
    - Training set (e.g., 2/3) for model construction
    - Test set (e.g., 1/3) for accuracy estimation
  - Random sampling: a variation of holdout
    - Repeat holdout k times, accuracy = avg. of the accuracies obtained

- **Cross-validation** (*k*-fold, where k = 10 is most popular)
  - Randomly partition the data into *k mutually exclusive* subsets, each approximately equal size
  - At *i*-th iteration, use D_i as test set and others as training set
  - Leave-one-out: *k* folds where *k* = # of tuples, for small sized data
  - ***Stratified cross-validation***: folds are stratified so that class dist. in each fold is approx. the same as that in the initial data

Evaluating Classifier Accuracy: Bootstrap

- **Bootstrap**
  - Works well with small data sets
  - Samples the given training tuples uniformly *with replacement*
    - Each time a tuple is selected, it is equally likely to be selected again and re-added to the training set
- Several bootstrap methods, and a common one is **.632 boostrap**
  - A data set with *d* tuples is sampled *d* times, with replacement, resulting in a training set of *d* samples. The data tuples that did not make it into the training set end up forming the test set. About 63.2% of the original data end up in the bootstrap, and the remaining 36.8% form the test set (since $(1 - 1/d)^d \approx e^{-1} = 0.368$)
  - Repeat the sampling procedure *k* times, overall accuracy of the model:

$$Acc(M) = \frac{1}{k}\sum_{i=1}^{k}(0.632 \times Acc(M_i)_{test\_set} + 0.368 \times Acc(M_i)_{train\_set})$$

Model Evaluation

- Metrics for Performance Evaluation
  - How to evaluate the performance of a model?
- Methods for Performance Evaluation
  - How to obtain reliable estimates?
- Methods for Model Comparison
  - How to compare the relative performance among competing models?

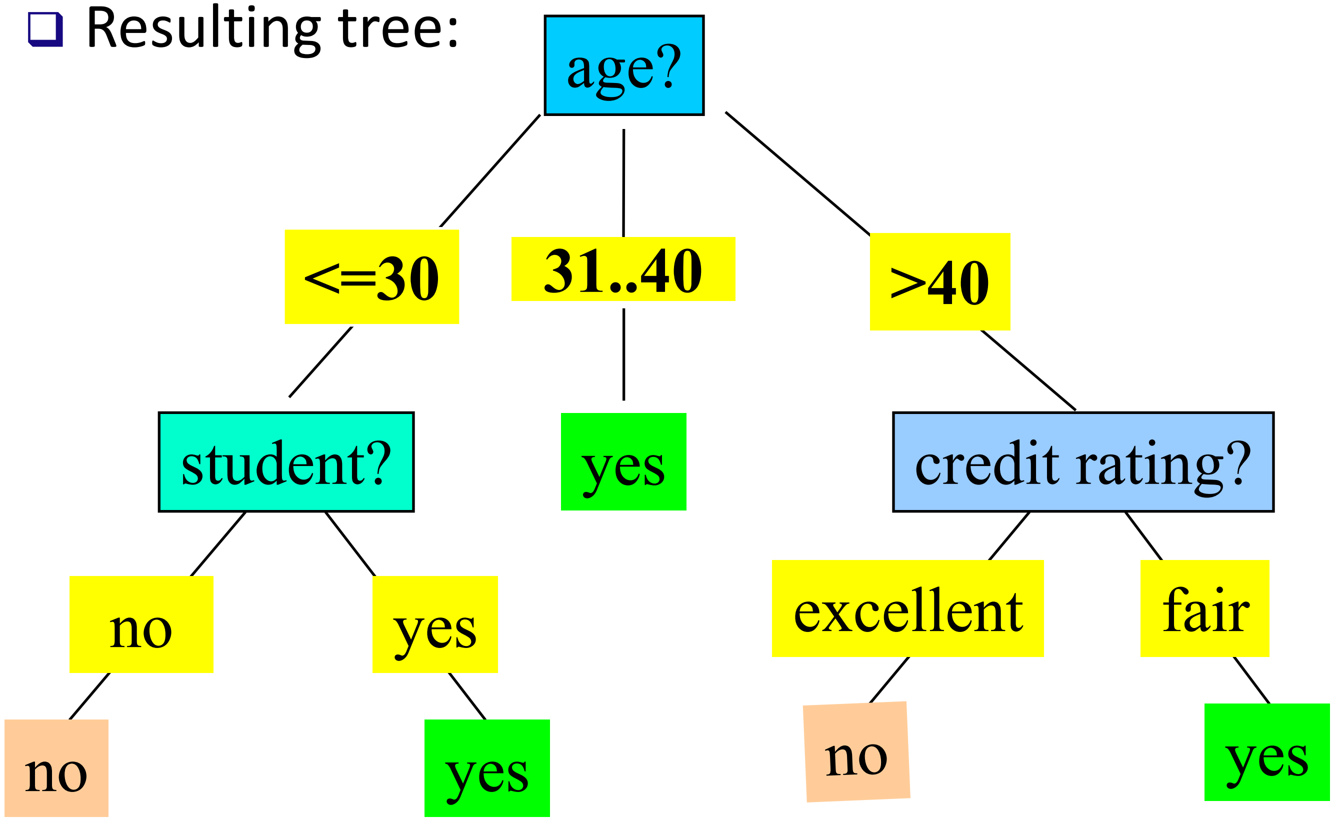
Classification: Basic Concepts

- Classification: Basic Concepts
- Decision Tree Induction
- Model Evaluation and Selection
- Practical Issues of Classification
- Bayes Classification Methods
- Techniques to Improve Classification Accuracy: Ensemble Methods
- Summary

This class

Decision Tree Induction: An Example

- Training data set: Buys_computer
- The data set follows an example of Quinlan's ID3 (Playing Tennis)
- Resulting tree:



age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

Algorithm for Decision Tree Induction

- Basic algorithm (a greedy algorithm)
  - Tree is constructed in a **top-down recursive divide-and-conquer manner**
  - At start, all the training examples are at the root
  - Attributes are categorical (if continuous-valued, they are discretized in advance)
  - Examples are partitioned recursively based on selected attributes
  - Test attributes are selected on the basis of a heuristic or statistical measure (e.g., **information gain**)
- Conditions for stopping partitioning
  - All samples for a given node belong to the same class
  - There are no remaining attributes for further partitioning—**majority voting** is employed for classifying the leaf
  - There are no samples left

Algorithm Outline

- Split (node, {data tuples})
  - A <= the best attribute for splitting the {data tuples}
  - Decision attribute for this node <= A
  - For each value of A, create new child node
  - For each child node / subset:
    - If one of the stopping conditions is satisfied: STOP
    - Else: Split (child_node, {subset})

ID3 algorithm: how it works
https://www.youtube.com/watch?v=_XhOdSLIE5c

Attribute Selection Measure: Information Gain (ID3/C4.5)

- Select the attribute with the highest information gain
- Let p_i be the probability that an arbitrary tuple in D belongs to class C_i, estimated by $|C_{i,D}|/|D|$
- Expected information (entropy) needed to classify a tuple in D:

$$Info(D) = -\sum_{i=1}^{m} p_i \log_2(p_i)$$

- Information needed (after using A to split D into v partitions) to classify D:

$$Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times Info(D_j)$$

- Information gained by branching on attribute A

$$Gain(A) = Info(D) - Info_A(D)$$

Attribute Selection: Information Gain

- Class P: buys_computer = "yes"
- Class N: buys_computer = "no"

$$Info(D) = I(9,5) = -\frac{9}{14}\log_2(\frac{9}{14}) - \frac{5}{14}\log_2(\frac{5}{14}) = 0.940$$

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

$$Info_{age}(D) = \frac{5}{14}I(2,3) + \frac{4}{14}I(4,0) + \frac{5}{14}I(3,2) = 0.694$$

$$Gain(age) = Info(D) - Info_{age}(D) = 0.246$$

Similarly,

$$Gain(income) = 0.029$$
$$Gain(student) = 0.151$$
$$Gain(credit\_rating) = 0.048$$

Gain Ratio for Attribute Selection (C4.5)

- Information gain measure is biased towards attributes with a large number of values
- C4.5 (a successor of ID3) uses **gain ratio** to overcome the problem (normalization to information gain)

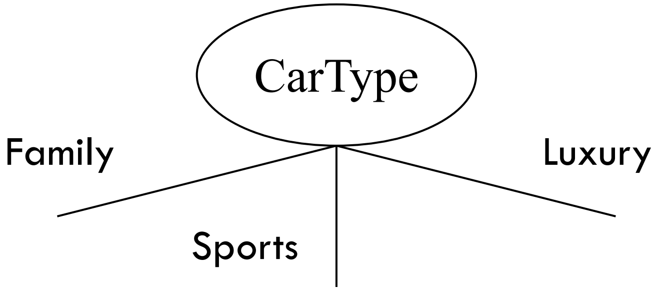
$$SplitInfo_A(D) = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} \times \log_2\left(\frac{|D_j|}{|D|}\right)$$

  - The entropy of the partitioning, or the potential information generated by splitting *D* into *v* partitions.

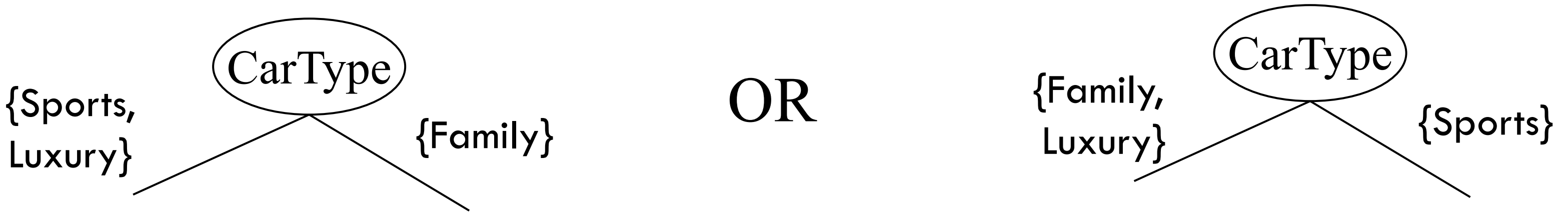
  - **GainRatio(A) = Gain(A)/SplitInfo(A)** (normalizing Information Gain)

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.



Measures of Node Impurity

- Entropy: $$H(Y) = -\sum_{i=1}^{m} p_i \log(p_i) \quad where\ p_i = P(Y = y_i)$$
  - Higher entropy => higher uncertainty, higher node impurity
  - Why entropy is used in information gain
- Gini Index
- Misclassification error

Gini Index (CART, IBM IntelligentMiner)

- If a data set D contains examples from n classes, gini index, $gini(D)$ is defined as

  $$gini(D) = 1 - \sum_{j=1}^{n} p_j^2$$, where p_j is the relative frequency of class j in D

- If a data set D is split on A into two subsets D_1 and D_2, the *gini* index *after the split* is defined as

  $$gini_A(D) = \frac{|D_1|}{|D|} gini(D_1) + \frac{|D_2|}{|D|} gini(D_2)$$

- Reduction in impurity:

  $$\Delta gini(A) = gini(D) - gini_A(D)$$

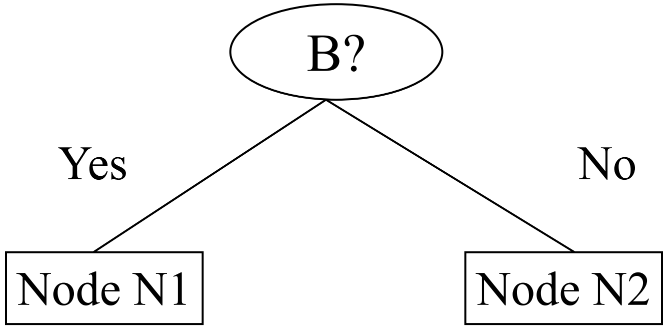
- The attribute provides the smallest $gini_A(D)$ (or, the largest reduction in impurity) is chosen to split the node.

Binary Attributes: Computing Gini Index

- Splits into two partitions
- Effect of weighing partitions:
  - Prefer Larger and Purer Partitions.

$$\boxed{gini(D) = 1 - \sum_{j=1}^{n} p_j^2}$$

	Parent
C1	**6**
C2	**6**
Gini = 0.500	



Gini(N1)
= 1 – $(5/\mathbf{7})^2$ – $(2/\mathbf{7})^2$
= 0.408

Gini(N2)
= 1 – $(1/\mathbf{5})^2$ – $(4/\mathbf{5})^2$
= 0.320

	N1	**N2**
C1	**5**	**1**
C2	**2**	**4**
Gini=0.371		

Gini(Children)
= 7/12 * 0.408 +
5/12 * 0.320
= 0.371

weighting

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	2	1
C2	4	1	1
Gini	0.393		

Two-way split
(find best partition of values)

	CarType	
	{Sports, Luxury}	{Family}
C1	3	1
C2	2	4
Gini	0.400	

	CarType	
	{Sports}	{Family, Luxury}
C1	2	2
C2	1	5
Gini	0.419	

Continuous Attributes: Computing Gini Index or Information Gain

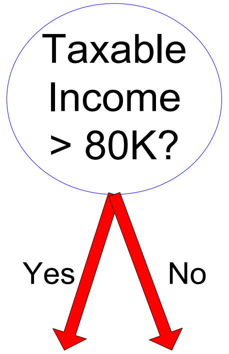
- **To discretize the attribute values**
  - **Use Binary Decisions based on one splitting value**

- Several Choices for the splitting value
  - Number of possible splitting values = Number of distinct values -1
  - Typically, the midpoint between each pair of adjacent values is considered as a possible *split point*
    - $(a_i+a_{i+1})/2$ is the midpoint between the values of a_i and a_{i+1}

- Each splitting value has a count matrix associated with it
  - 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.

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



Continuous Attributes: Computing Gini Index or expected information requirement

First decide the splitting value to discretize the attribute:

- For efficient computation: for each attribute,
  - Step 1: Sort the attribute on values
  - Step 2: Linearly scan these values, each time updating the count matrix
  - Step 3: **Computing Gini index and choose the split position that has the least Gini index**

Step 1: Sorted Values →

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

Step 1: Possible Splitting Values →

Step 2: Yes / No counts

Step 3: Gini

Possible Splitting Values	65		72		80		87		92		97		110		122		172	
	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>
Yes	0	3	0	3	0	3	1	2	2	1	3	0	3	0	3	0	3	0
No	1	6	2	5	3	4	3	4	3	4	3	4	4	3	5	2	6	1
Gini	0.400		0.375		0.343		0.417		0.400		*0.300*		0.343		0.375		0.400	

At each level of the decision tree, for attribute selection, (1) First, discretize a continuous attribute by deciding the splitting value; (2) Then, compare the discretized attribute with other attributes in terms of Gini Index reduction or Information Gain.

Classification: Basic Concepts

- Classification: Basic Concepts
- Decision Tree Induction
- Model Evaluation and Selection
- Practical Issues of Classification
- Bayes Classification Methods
- Techniques to Improve Classification Accuracy: Ensemble Methods
- Summary

Classifier Evaluation Metrics: Confusion Matrix

Confusion Matrix:

Actual class\Predicted class	C_1	$\neg C_1$
C_1	**True Positives (TP)**	**False Negatives (FN)**
$\neg C_1$	**False Positives (FP)**	**True Negatives (TN)**

Example of Confusion Matrix:

Actual class\Predicted class	buy_computer = yes	buy_computer = no	Total
buy_computer = yes	**6954**	**46**	7000
buy_computer = no	**412**	**2588**	3000
Total	7366	2634	10000

- Given *m* classes, an entry, $CM_{i,j}$ in a **confusion matrix** indicates # of tuples in class *i* that were labeled by the classifier as class *j*
  - May have extra rows/columns to provide totals

Classifier Evaluation Metrics: Accuracy, Error Rate

A\P	C	¬C	
C	**TP**	**FN**	**P**
¬C	**FP**	**TN**	**N**
	P'	**N'**	**All**

- **Classifier Accuracy,** or recognition rate: percentage of test set tuples that are correctly classified

  Accuracy = (TP + TN)/All

- **Error rate:** *1 – accuracy*, or

  Error rate = (FP + FN)/All

Limitation of Accuracy

- Consider a 2-class problem
  - Number of Class 0 examples = 9990
  - Number of Class 1 examples = 10

- If a model predicts everything to be class 0,

  Accuracy is 9990/10000 = 99.9 %

  - Accuracy is misleading because model does not detect any class 1 example

Cost-Sensitive Measures

$$\text{Precision (p)} = \frac{a}{a+c}$$

$$\text{Recall (r)} = \frac{a}{a+b}$$

$$\text{F - measure (F)} = \frac{2rp}{r+p} = \frac{2a}{2a+b+c}$$

		PREDICTED CLASS	
		Class=Yes	Class=No
ACTUAL CLASS	Class=Yes	a (TP)	b (FN)
	Class=No	c (FP)	d (TN)

- Precision is biased towards C(Yes|Yes) & C(Yes|No)
- Recall is biased towards C(Yes|Yes) & C(No|Yes)
- F-measure is biased towards all except C(No|No)

$$\text{Weighted Accuracy} = \frac{w_1 a + w_4 d}{w_1 a + w_2 b + w_3 c + w_4 d}$$

Evaluating Classifier Accuracy: Holdout & Cross-Validation Methods

- **Holdout method**
  - Given data is randomly partitioned into two independent sets
    - Training set (e.g., 2/3) for model construction
    - Test set (e.g., 1/3) for accuracy estimation
  - Random sampling: a variation of holdout
    - Repeat holdout k times, accuracy = avg. of the accuracies obtained

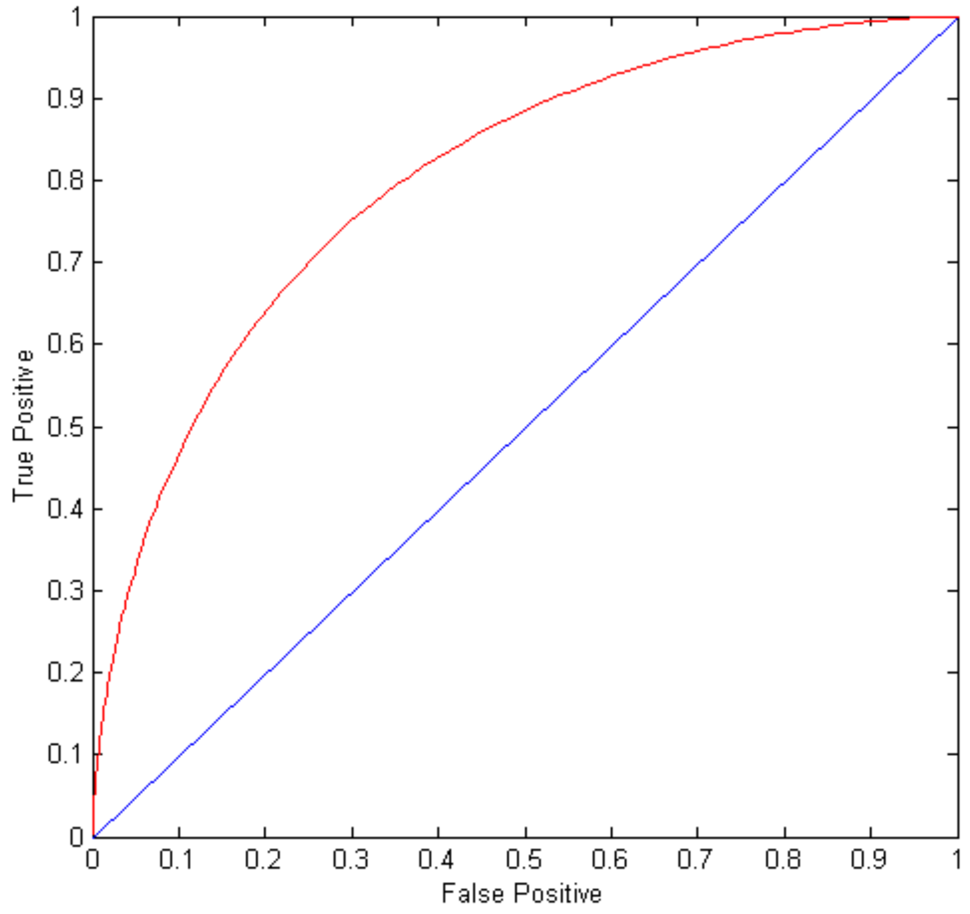
- **Cross-validation** (*k*-fold, where k = 10 is most popular)
  - Randomly partition the data into *k mutually exclusive* subsets, each approximately equal size
  - At *i*-th iteration, use D_i as test set and others as training set
  - Leave-one-out: *k* folds where *k* = # of tuples, for small sized data
  - ***Stratified cross-validation***: folds are stratified so that class dist. in each fold is approx. the same as that in the initial data

ROC (Receiver Operating Characteristic) Curve

(False Positive Rate, True Positive Rate):

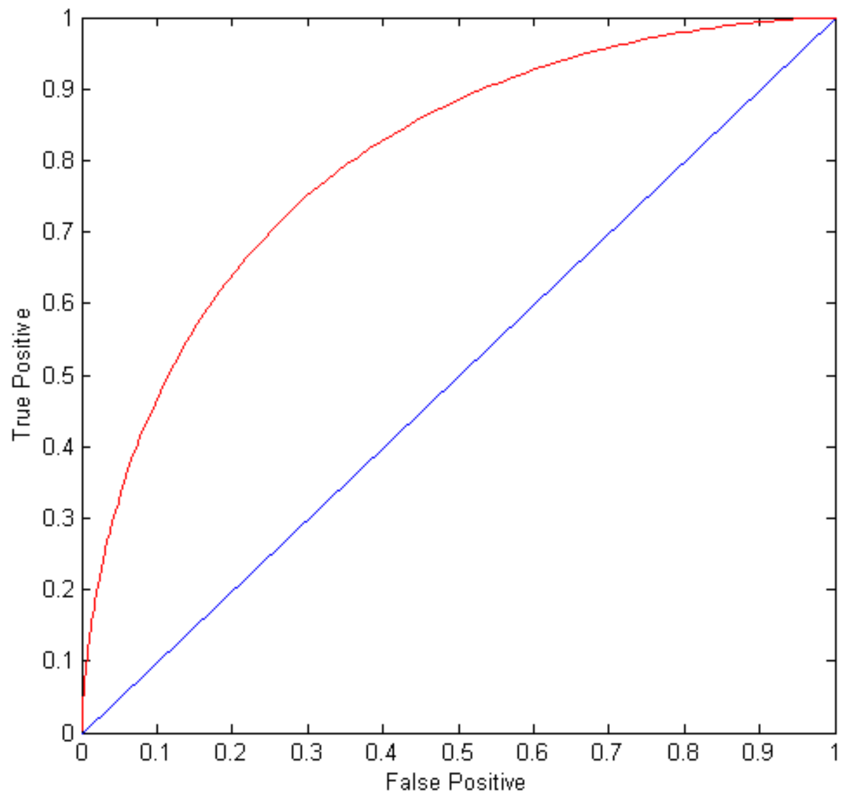
$$FPR = \frac{FP}{N} \qquad TPR = \frac{TP}{P}$$

- (0,0): declare everything to be negative class
- (1,1): declare everything to be positive class
- (0,1): ideal
- Diagonal line:
  - Random guessing
  - Below diagonal line:
    - prediction is opposite of the true class



Using ROC for Classification Model Comparison

- **ROC** (Receiver Operating Characteristics) curves: for visual comparison of classification models
- Originated from signal detection theory
- Shows the trade-off between the true positive rate and the false positive rate
- The area under the ROC curve is a measure of the accuracy of the model
- The closer to the diagonal line (i.e., the closer the area is to 0.5), the less accurate is the model



ROC Calculation

- Rank the test examples by prediction probability in descending order
- Gradually decreases the classification threshold from 1.0 to 0.0 and calculate the true positive and false positive rate along the way

$p \geq 1.0 \rightarrow$ Yes

$TPR = 0.0$
$FPR = 0.0$

Input	Prebability of Prediction	Actual Class
x_1	0.95	Yes
x_2	0.85	Yes
x_3	0.75	No
x_4	0.65	Yes
x_5	0.4	No
x_6	0.3	No

ROC Calculation

- Rank the test examples by prediction probability in descending order
- Gradually decreases the classification threshold from 1.0 to 0.0 and calculate the true positive and false positive rate along the way

$p \geq 0.9 \rightarrow$ Yes

Input	Prebability of Prediction	Actual Class
x_1	0.95	Yes
x_2	0.85	Yes
x_3	0.75	No
x_4	0.65	Yes
x_5	0.4	No
x_6	0.3	No

$TPR = 0.334$
$FPR = 0.0$

ROC Calculation

- Rank the test examples by prediction probability in descending order
- Gradually decreases the classification threshold from 1.0 to 0.0 and calculate the true positive and false positive rate along the way

$p \geq 0.8 \rightarrow$ Yes

Input	Prebability of Prediction	Actual Class
x_1	0.95	Yes
x_2	0.85	Yes
x_3	0.75	No
x_4	0.65	Yes
x_5	0.4	No
x_6	0.3	No

$TPR = 0.666$
$FPR = 0.0$

ROC Calculation

- Rank the test examples by prediction probability in descending order
- Gradually decreases the classification threshold from 1.0 to 0.0 and calculate the true positive and false positive rate along the way

Input	Prebability of Prediction	Actual Class
x_1	0.95	Yes
x_2	0.85	Yes
x_3	0.75	No
x_4	0.65	Yes
x_5	0.4	No
x_6	0.3	No

$p \geq 0.7 \rightarrow$ Yes

$TPR = 0.666$
$FPR = 0.334$

ROC Calculation

- Rank the test examples by prediction probability in descending order
- Gradually decreases the classification threshold from 1.0 to 0.0 and calculate the true positive and false positive rate along the way

Input	Prebability of Prediction	Actual Class
x_1	0.95	Yes
x_2	0.85	Yes
x_3	0.75	No
x_4	0.65	Yes
x_5	0.4	No
x_6	0.3	No

$p \geq 0.5 \rightarrow$ Yes

$TPR = 1.0$
$FPR = 0.334$

ROC Calculation

- Rank the test examples by prediction probability in descending order
- Gradually decreases the classification threshold from 1.0 to 0.0 and calculate the true positive and false positive rate along the way

Input	Prebability of Prediction	Actual Class
x_1	0.95	Yes
x_2	0.85	Yes
x_3	0.75	No
x_4	0.65	Yes
x_5	0.4	No
x_6	0.3	No

$p \geq 0.4 \rightarrow$ Yes

$TPR = 1.0$
$FPR = 0.666$

ROC Calculation

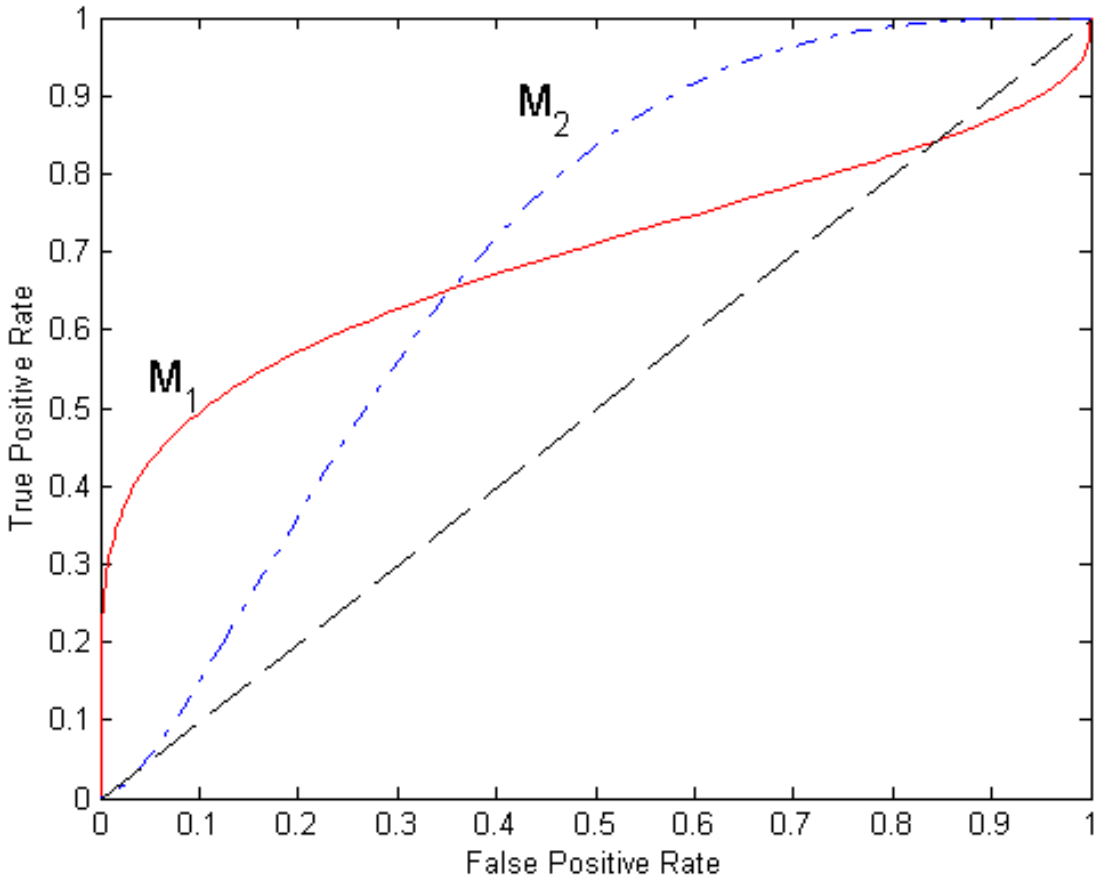
- Rank the test examples by prediction probability in descending order
- Gradually decreases the classification threshold from 1.0 to 0.0 and calculate the true positive and false positive rate along the way

Input	Prebability of Prediction	Actual Class
x_1	0.95	Yes
x_2	0.85	Yes
x_3	0.75	No
x_4	0.65	Yes
x_5	0.4	No
x_6	0.3	No

$p \geq 0.3 \rightarrow$ Yes

$TPR = 1.0$
$FPR = 1.0$

Using ROC for Classification Model Comparison

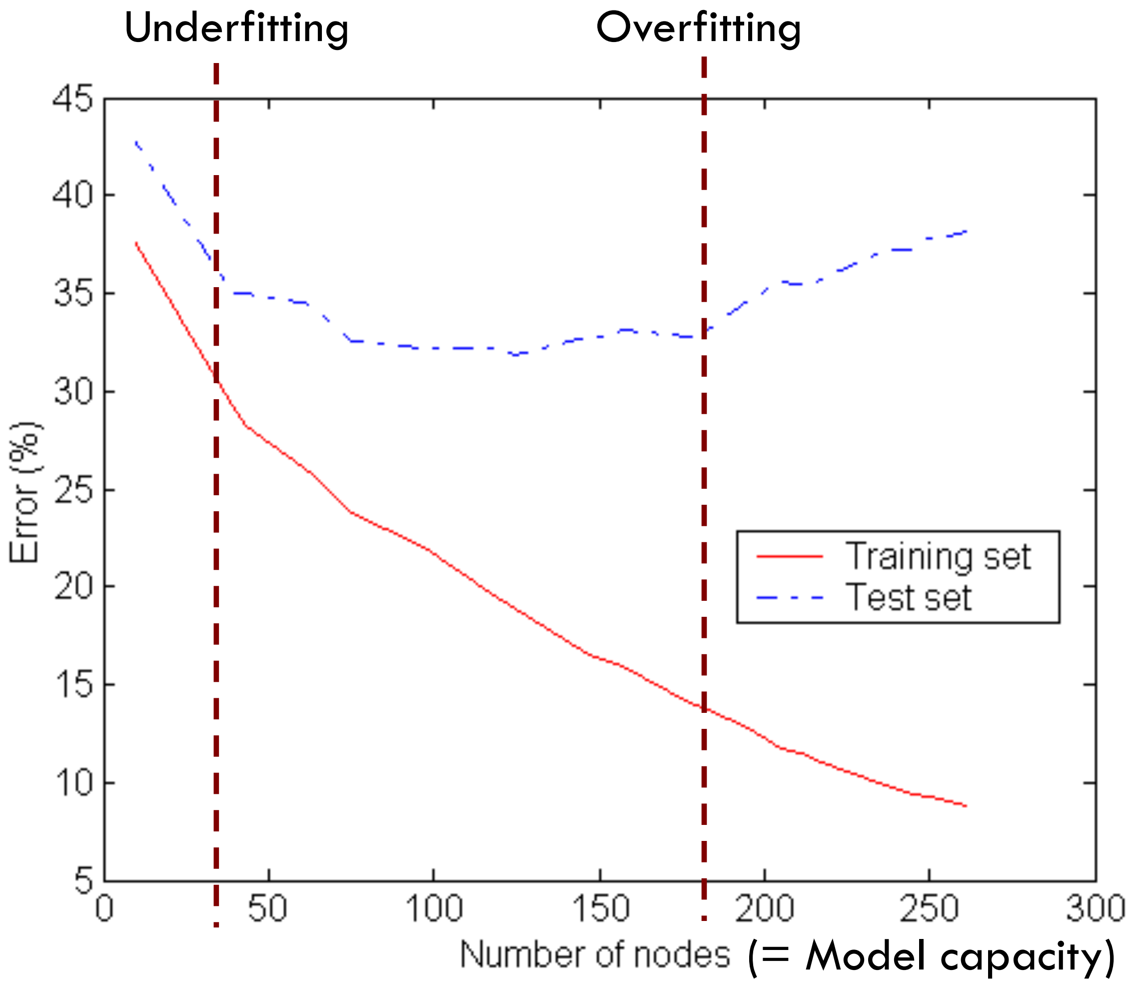


- No model consistently outperform the other
  - M_1 is better for small FPR
  - M_2 is better for large FPR
- Area Under the ROC curve
  - Ideal:
    - Area = 1
  - Random guess (diagonal line):
    - Area = 0.5

Classification: Basic Concepts

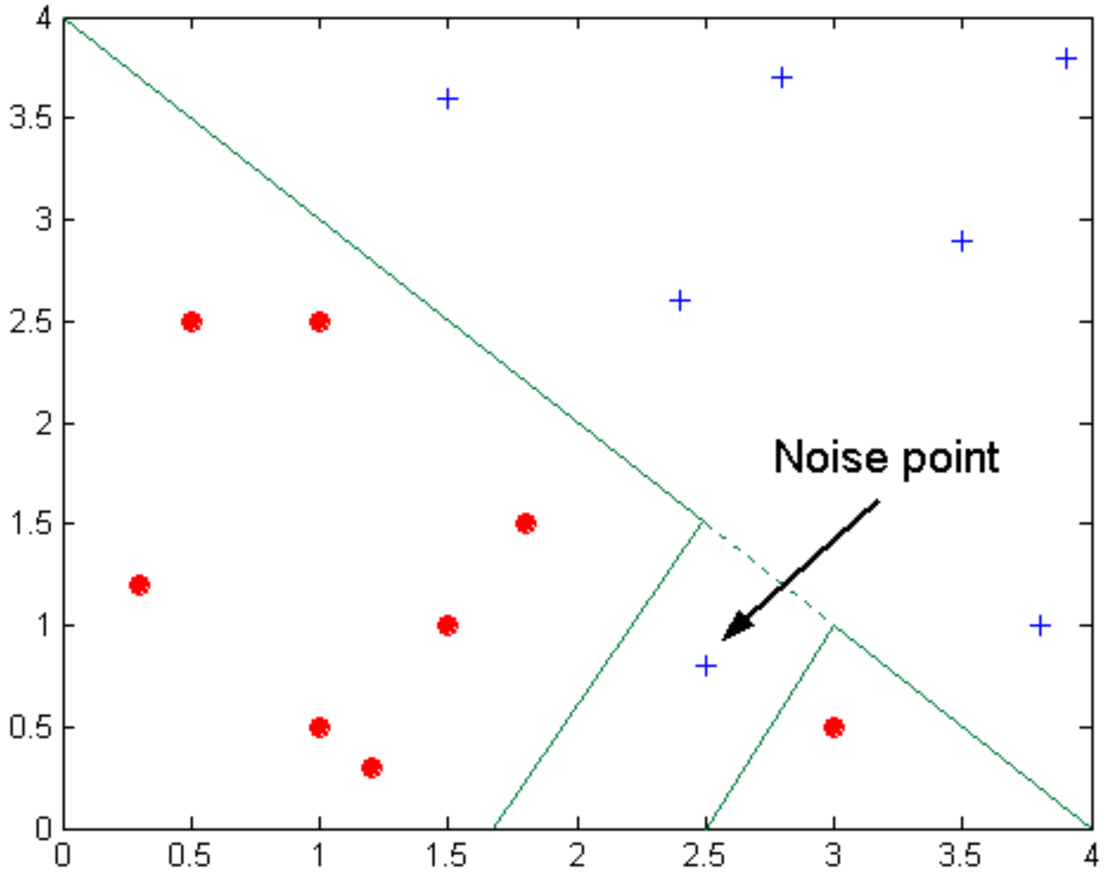
- Classification: Basic Concepts
- Decision Tree Induction
- Model Evaluation and Selection
- Practical Issues of Classification
- Bayes Classification Methods
- Techniques to Improve Classification Accuracy: Ensemble Methods
- Summary

Underfitting and Overfitting



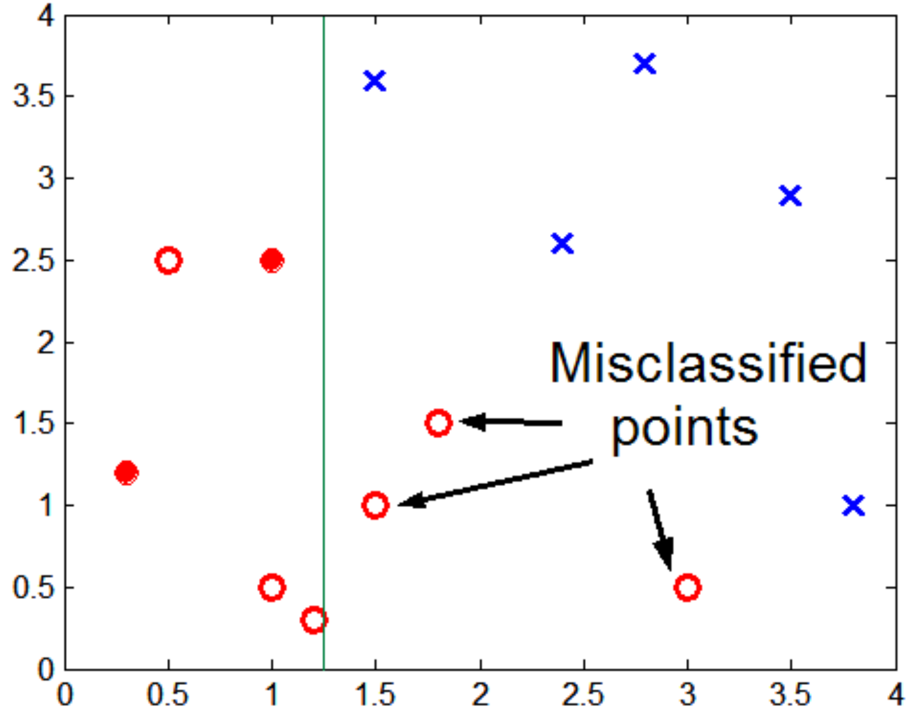
Underfitting: when model is too simple, both training and test errors are large

Overfitting due to Noise



Decision boundary is distorted by noise point

Overfitting due to Insufficient Examples



Lack of data points in the lower half of the diagram makes it difficult to predict correctly the class labels of that region

- Insufficient number of training records in the region causes the decision tree to predict the test examples using other training records that are irrelevant to the classification task

Notes on Overfitting

- 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

- Need new ways for estimating errors

Estimating Generalization Errors

- Re-substitution errors: error on training (Σ e(t))
- Generalization errors: error on testing (Σ e'(t))

Estimating Generalization Errors

- Re-substitution errors: error on training (Σ e(t))
- Generalization errors: error on testing (Σ e'(t))
- Methods for estimating generalization errors:
  - Optimistic approach: e'(t) = e(t)

Estimating Generalization Errors

- Re-substitution errors: error on training (Σ e(t))
- Generalization errors: error on testing (Σ e'(t))
- Methods for estimating generalization errors:
  - Optimistic approach: e'(t) = e(t)
  - Pessimistic approach:
    - For each leaf node: e'(t) = (e(t)+0.5)
    - Total errors: e'(T) = e(T) + N $\times$ 0.5 (N: number of leaf nodes)
    - For a tree with 30 leaf nodes and 10 errors on training (out of 1000 instances):

      Training error = 10/1000 = 1%
      Generalization error = (10 + 30$\times$0.5)/1000 = 2.5%

Estimating Generalization Errors

- Re-substitution errors: error on training (Σ e(t))
- Generalization errors: error on testing (Σ e'(t))

- Methods for estimating generalization errors:
  - Optimistic approach: e'(t) = e(t)

  - Pessimistic approach:
    - For each leaf node: e'(t) = (e(t)+0.5)
    - Total errors: e'(T) = e(T) + N $\times$ 0.5 (N: number of leaf nodes)
    - For a tree with 30 leaf nodes and 10 errors on training (out of 1000 instances):

      Training error = 10/1000 = 1%

      Generalization error = (10 + 30$\times$0.5)/1000 = 2.5%

  - Reduced error pruning (REP):
    - uses validation data set to estimate generalization error

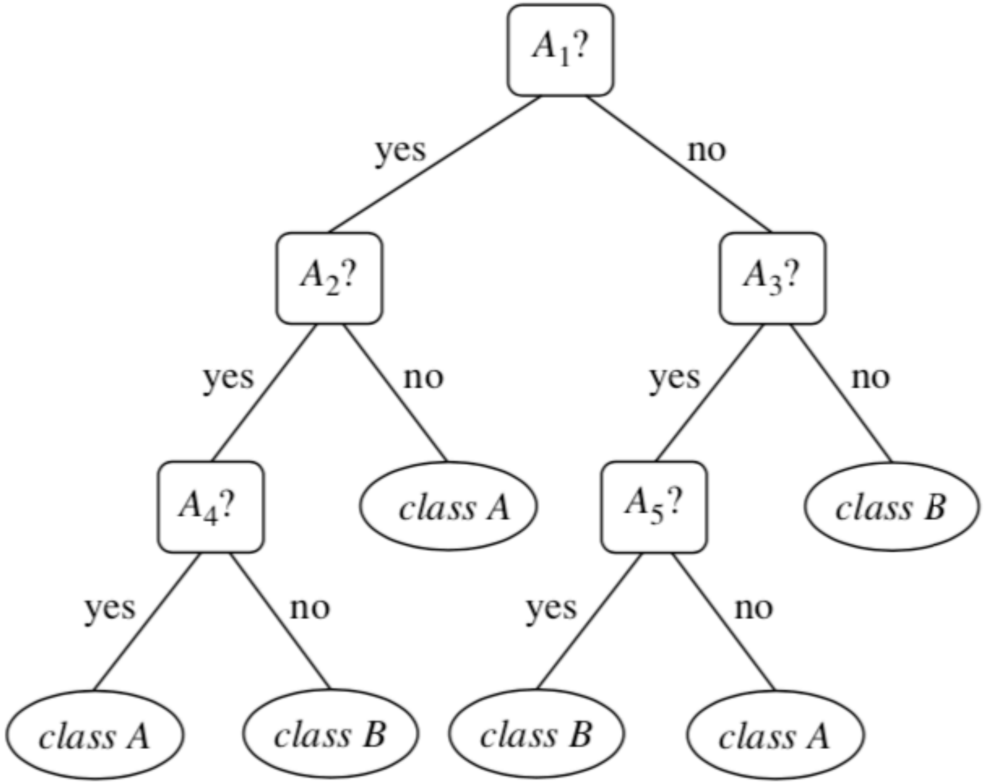
How to Address Overfitting

- 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 (e.g., using χ^2 test)
    - Stop if expanding the current node does not improve impurity measures (e.g., Gini or information gain).

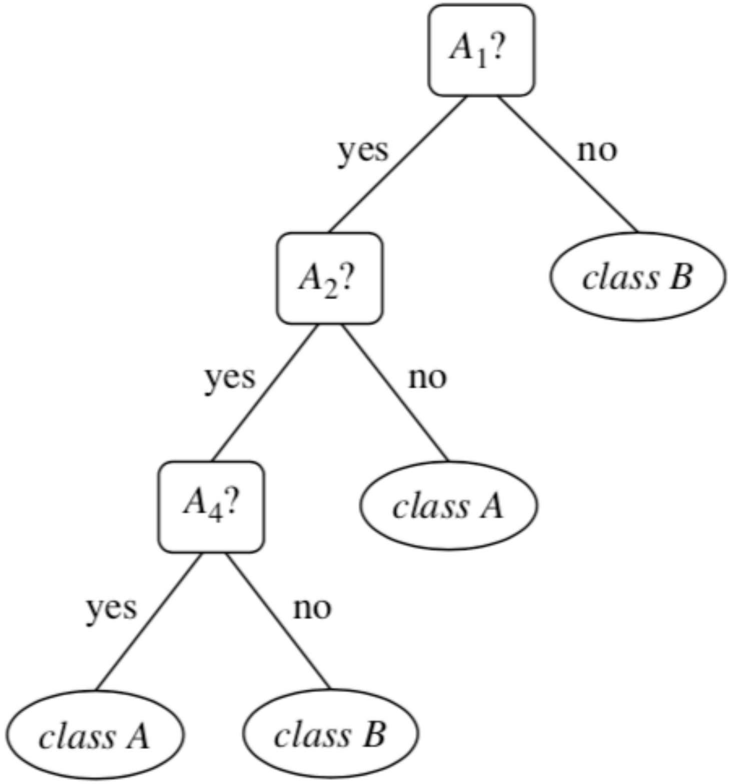
How to Address Overfitting…

- 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

Post-Pruning



Unpruned



Pruned

Example of Post-Pruning

Class = Yes	20
Class = No	10
Error = 10/30	

Training Error (Before splitting) = 10/30

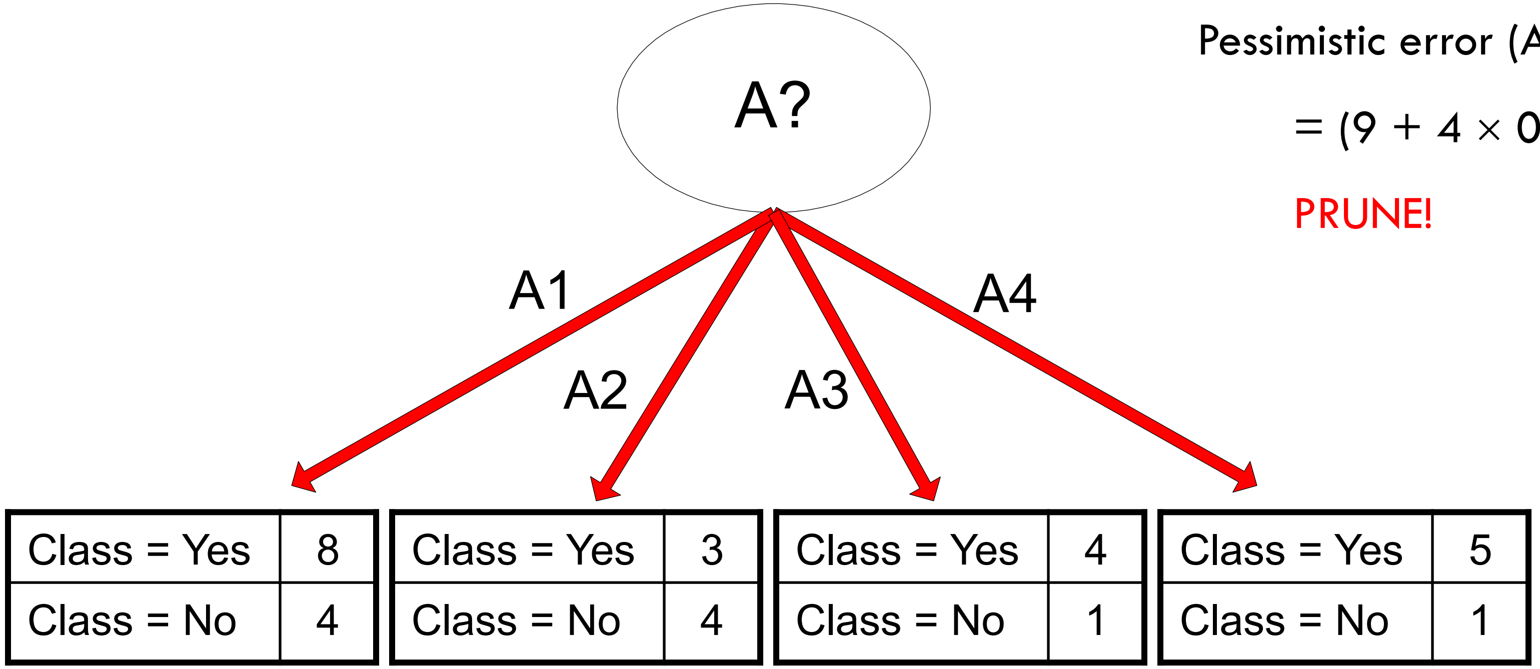
Pessimistic error = (10 + 0.5)/30 = 10.5/30

Training Error (After splitting) = 9/30

Pessimistic error (After splitting)

$= (9 + 4 \times 0.5)/30 = 11/30$

PRUNE!



Examples of Post-pruning

- Optimistic error?

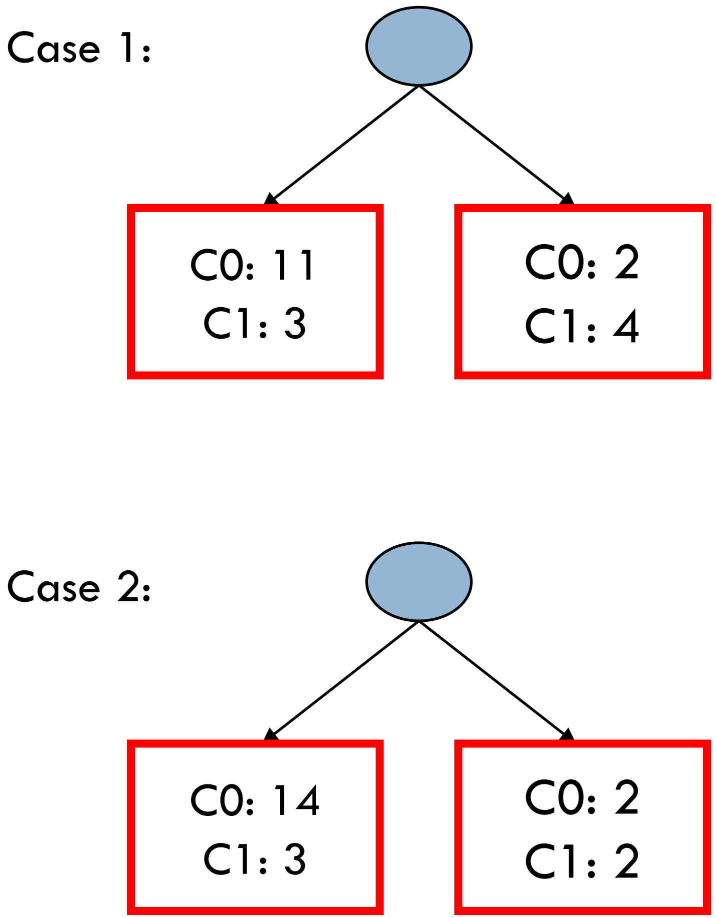
  Don't prune for both cases

- Pessimistic error?

  Don't prune case 1, prune case 2

- Reduced error pruning?

  Depends on validation set



Occam's Razor

- Given two models of similar generalization errors, one should prefer the simpler model over the more complex model
- For complex models, there is a greater chance that it was fitted accidentally by errors in data
- Therefore, one should include model complexity when evaluating a model

Classification: Basic Concepts

- Classification: Basic Concepts
- Decision Tree Induction
- Model Evaluation and Selection
- Practical Issues of Classification
- Bayes Classification Methods
- Techniques to Improve Classification Accuracy: Ensemble Methods

Bayes' Theorem: Basics

- Bayes' Theorem:
  - Let **X** be a data sample ("*evidence*"): class label is unknown
  - Let H be a *hypothesis* that X belongs to class C
  - Classification is to determine P(H|**X**), (i.e., *posterior probability*): the probability that the hypothesis holds given the observed data sample **X**

$$P(H|\mathbf{X}) = \frac{P(\mathbf{X}|H)P(H)}{P(\mathbf{X})} = P(\mathbf{X}|H) \times P(H) / P(\mathbf{X})$$

Bayes' Theorem: Basics

- Bayes' Theorem:
  - Let **X** be a data sample ("*evidence*"): class label is unknown
  - Let H be a *hypothesis* that X belongs to class C
  - Classification is to determine P(H|**X**), (i.e., *posterior probability*): the probability that the hypothesis holds given the observed data sample **X**

$$P(H|\mathbf{X}) = \frac{P(\mathbf{X}|H)P(H)}{P(\mathbf{X})} = P(\mathbf{X}|H) \times P(H) / P(\mathbf{X})$$

  - P(H) (*prior probability*): the initial probability
    - E.g., **X** will buy computer, regardless of age, income, …

Bayes' Theorem: Basics

- Bayes' Theorem:
  - Let **X** be a data sample ("*evidence*"): class label is unknown
  - Let H be a *hypothesis* that X belongs to class C
  - Classification is to determine P(H|**X**), (i.e., *posterior probability*): the probability that the hypothesis holds given the observed data sample **X**

$$P(H|\mathbf{X}) = \frac{P(\mathbf{X}|H)P(H)}{P(\mathbf{X})} = P(\mathbf{X}|H) \times P(H) / P(\mathbf{X})$$

  - P(H) (*prior probability*): the initial probability
    - E.g., **X** will buy computer, regardless of age, income, …
  - P(**X**): probability that sample data is observed

Bayes' Theorem: Basics

- Bayes' Theorem:
  - Let **X** be a data sample ("*evidence*"): class label is unknown
  - Let H be a *hypothesis* that X belongs to class C
  - Classification is to determine P(H|**X**), (i.e., ***posterior probability***): the probability that the hypothesis holds given the observed data sample **X**

$$P(H|\mathbf{X}) = \frac{P(\mathbf{X}|H)P(H)}{P(\mathbf{X})} = P(\mathbf{X}|H) \times P(H) / P(\mathbf{X})$$

  - P(H) (*prior probability*): the initial probability
    - E.g., **X** will buy computer, regardless of age, income, …
  - P(**X**): probability that sample data is observed
  - P(**X**|H) (likelihood): the probability of observing the sample **X**, given that the hypothesis holds
    - E.g., Given that **X** will buy computer, the prob. that X is 31..40, medium income

Prediction Based on Bayes' Theorem

- Given training data **X**, ***posterior probability** of a hypothesis* H, P(H|**X**), follows the Bayes' theorem

$$P(H|\mathbf{X}) = \frac{P(\mathbf{X}|H)P(H)}{P(\mathbf{X})} = P(\mathbf{X}|H) \times P(H) / P(\mathbf{X})$$

- Informally, this can be viewed as

  posterior = likelihood x prior/evidence

- Predicts **X** belongs to C_i iff the probability P(C_i|**X**) is the highest among all the P(C_k|X) for all the *k* classes
- Practical difficulty: It requires initial knowledge of many probabilities, involving significant computational cost

Classification Is to Derive the Maximum A Posteriori

- Let D be a training set of tuples and their associated class labels, and each tuple is represented by an n-dimensional attribute vector $\mathbf{X} = (x_1, x_2, \ldots, x_n)$
- Suppose there are *m* classes $C_1, C_2, \ldots, C_m$.
- Classification is to derive the maximum a posteriori, i.e., the maximal $P(C_i|\mathbf{X})$

Classification Is to Derive the Maximum Posteriori

- Let D be a training set of tuples and their associated class labels, and each tuple is represented by an n-dimensional attribute vector $\mathbf{X} = (x_1, x_2, \ldots, x_n)$
- Suppose there are *m* classes $C_1, C_2, \ldots, C_m$.
- Classification is to derive the maximum posteriori, i.e., the maximal $P(C_i|\mathbf{X})$
- This can be derived from Bayes' theorem

$$P(C_i|\mathbf{X}) = \frac{P(\mathbf{X}|C_i)P(C_i)}{P(\mathbf{X})}$$

- Since P(X) is constant for all classes, only

$$\boxed{P(C_i|\mathbf{X}) = P(\mathbf{X}|C_i)P(C_i)}$$

needs to be maximized

Naïve Bayes Classifier (why Naïve? :-)

- A simplifying assumption: attributes are conditionally independent (i.e., no dependence relation between attributes):

$$P(\mathbf{X}|C_i) = \prod_{k=1}^{n} P(x_k|C_i) = P(x_1|C_i) \times P(x_2|C_i) \times \ldots \times P(x_n|C_i)$$

- This greatly reduces the computation cost: Only counts the per-class distributions

Naïve Bayes Classifier

- A simplifying assumption: attributes are conditionally independent (i.e., no dependence relation between attributes):

$$P(\mathbf{X}|C_i) = \prod_{k=1}^{n} P(x_k|C_i) = P(x_1|C_i) \times P(x_2|C_i) \times \ldots \times P(x_n|C_i)$$

- This greatly reduces the computation cost: Only counts the per-class distributions
- If A_k is categorical, $P(x_k|C_i)$ is the # of tuples in C_i having value x_k for A_k divided by $|C_{i, D}|$ (# of tuples in C_i)

Naïve Bayes Classifier

- A simplifying assumption: attributes are conditionally independent (i.e., no dependence relation between attributes):

$$P(\mathbf{X}|C_i) = \prod_{k=1}^{n} P(x_k|C_i) = P(x_1|C_i) \times P(x_2|C_i) \times \ldots \times P(x_n|C_i)$$

- If A_k is continous-valued, $P(x_k|C_i)$ is usually computed based on Gaussian distribution with sample mean μ and standard deviation σ

$$g(x,\mu,\sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

and $P(x_k|C_i)$ is

$$P(x_k|C_i) = g(x_k, \mu_{C_i}, \sigma_{C_i})$$

Naïve Bayes Classifier

- A simplifying assumption: attributes are conditionally independent (i.e., no dependence relation between attributes):

$$P(\mathbf{X}|C_i) = \prod_{k=1}^{n} P(x_k|C_i) = P(x_1|C_i) \times P(x_2|C_i) \times \ldots \times P(x_n|C_i)$$

- If A_k is continuous-valued, $P(x_k|C_i)$ is usually computed based on Gaussian distribution with a mean μ and standard deviation σ

$$g(x,\mu,\sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

and $P(x_k|C_i)$ is

$$P(x_k|C_i) = g(x_k, \mu_{C_i}, \sigma_{C_i})$$

Here, mean μ and standard deviation σ are estimated based on the values of attribute A_k for training tuples of class C_i.

117

Naïve Bayes Classifier: Training Dataset

Class:

C1:buys_computer = ‘yes’

C2:buys_computer = ‘no’

Data to be classified:

X = (age <=30, Income = medium, Student = yes, Credit_rating = Fair)

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

Naïve Bayes Classifier: An Example

- Prior probability $P(C_i)$:

  P(buys_computer = "yes") = 9/14 = 0.643

  P(buys_computer = "no") = 5/14= 0.357

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

Naïve Bayes Classifier: An Example

- $P(C_i)$: P(buys_computer = "yes") = 9/14 = 0.643
  P(buys_computer = "no") = 5/14= 0.357

- Compute $P(X|C_i)$ for each class, where,

X = (age <=30, Income = medium, Student = yes, Credit_rating = Fair)

According to "the naïve assumption", first get:

P(age = "<=30"|buys_computer = "yes") = 2/9 = 0.222

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

Naïve Bayes Classifier: An Example

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

- $P(C_i)$: P(buys_computer = "yes") = 9/14 = 0.643
  P(buys_computer = "no") = 5/14= 0.357

- Compute $P(X|C_i)$ for each class, where,

X = (age <=30, Income = medium, Student = yes, Credit_rating = Fair)

According to "the naïve assumption", first get:

P(age = "<=30"|buys_computer = "yes") = 2/9 = 0.222
P(age = "<= 30"|buys_computer = "no") = 3/5 = 0.6
P(income = "medium" | buys_computer = "yes") = 4/9 = 0.444
P(income = "medium" | buys_computer = "no") = 2/5 = 0.4
P(student = "yes" | buys_computer = "yes) = 6/9 = 0.667
P(student = "yes" | buys_computer = "no") = 1/5 = 0.2
P(credit_rating = "fair" | buys_computer = "yes") = 6/9 = 0.667
P(credit_rating = "fair" | buys_computer = "no") = 2/5 = 0.4

Naïve Bayes Classifier: An Example

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

- $P(C_i)$: P(buys_computer = "yes") = 9/14 = 0.643
  P(buys_computer = "no") = 5/14= 0.357
- Compute $P(X_i|C_i)$ for each class
  P(age = "<=30"|buys_computer = "yes") = 2/9 = 0.222
  P(age = "<= 30"|buys_computer = "no") = 3/5 = 0.6
  P(income = "medium" | buys_computer = "yes") = 4/9 = 0.444
  P(income = "medium" | buys_computer = "no") = 2/5 = 0.4
  P(student = "yes" | buys_computer = "yes) = 6/9 = 0.667
  P(student = "yes" | buys_computer = "no") = 1/5 = 0.2
  P(credit_rating = "fair" | buys_computer = "yes") = 6/9 = 0.667
  P(credit_rating = "fair" | buys_computer = "no") = 2/5 = 0.4
- **X = (age <= 30 , income = medium, student = yes, credit_rating = fair)**

$P(X|C_i)$: P(X|buys_computer = "yes") = P(age = "<=30"|buys_computer = "yes") x P(income = "medium" | buys_computer = "yes") x P(student = "yes" | buys_computer = "yes) x P(credit_rating = "fair" | buys_computer = "yes") = 0.044

Naïve Bayes Classifier: An Example

- $P(C_i)$: P(buys_computer = "yes") = 9/14 = 0.643
  P(buys_computer = "no") = 5/14= 0.357
- Compute $P(X_i|C_i)$ for each class
  P(age = "<=30" | buys_computer = "yes") = 2/9 = 0.222
  P(age = "<= 30" | buys_computer = "no") = 3/5 = 0.6
  P(income = "medium" | buys_computer = "yes") = 4/9 = 0.444
  P(income = "medium" | buys_computer = "no") = 2/5 = 0.4
  P(student = "yes" | buys_computer = "yes) = 6/9 = 0.667
  P(student = "yes" | buys_computer = "no") = 1/5 = 0.2
  P(credit_rating = "fair" | buys_computer = "yes") = 6/9 = 0.667
  P(credit_rating = "fair" | buys_computer = "no") = 2/5 = 0.4
- **X = (age <= 30 , income = medium, student = yes, credit_rating = fair)**

$P(X|C_i)$: P(X|buys_computer = "yes") = 0.222 x 0.444 x 0.667 x 0.667 = 0.044
P(X|buys_computer = "no") = 0.6 x 0.4 x 0.2 x 0.4 = 0.019

$P(X|C_i)*P(C_i)$: P(X|buys_computer = "yes") * P(buys_computer = "yes") = 0.028
P(X|buys_computer = "no") * P(buys_computer = "no") = 0.007

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

Take into account the prior probabilities

Naïve Bayes Classifier: An Example

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

- $P(C_i)$: P(buys_computer = "yes") = 9/14 = 0.643
  P(buys_computer = "no") = 5/14= 0.357
- Compute $P(X_i|C_i)$ for each class
  P(age = "<=30" | buys_computer = "yes") = 2/9 = 0.222
  P(age = "<= 30" | buys_computer = "no") = 3/5 = 0.6
  P(income = "medium" | buys_computer = "yes") = 4/9 = 0.444
  P(income = "medium" | buys_computer = "no") = 2/5 = 0.4
  P(student = "yes" | buys_computer = "yes) = 6/9 = 0.667
  P(student = "yes" | buys_computer = "no") = 1/5 = 0.2
  P(credit_rating = "fair" | buys_computer = "yes") = 6/9 = 0.667
  P(credit_rating = "fair" | buys_computer = "no") = 2/5 = 0.4
- **X = (age <= 30 , income = medium, student = yes, credit_rating = fair)**

$P(X|C_i)$: P(X | buys_computer = "yes") = 0.222 x 0.444 x 0.667 x 0.667 = 0.044
P(X | buys_computer = "no") = 0.6 x 0.4 x 0.2 x 0.4 = 0.019

$P(X|C_i)*P(C_i)$: P(X | buys_computer = "yes") * P(buys_computer = "yes") = 0.028
P(X | buys_computer = "no") * P(buys_computer = "no") = 0.007

Since Red > Blue here, **X belongs to class ("buys_computer = yes")**

Avoiding the Zero-Probability Problem

- Naïve Bayesian prediction requires each conditional prob. be **non-zero**. Otherwise, the predicted prob. will be zero

$$P(X|C_i) = \prod_{k=1}^{n} P(x_k|C_i)$$

- Ex. Suppose a dataset with 1000 tuples, income=low (0), income= medium (990), and income = high (10)

- Use **Laplacian correction** (or Laplacian estimator)
  - *Adding 1 to each case*

    Prob(income = low) = 1/1003

    Prob(income = medium) = 991/1003

    Prob(income = high) = 11/1003
  - *Assumption*: dataset is large enough such that adding 1 would only make a negligible difference in the estimated probability values
  - The "corrected" prob. estimates are close to their "uncorrected" counterparts

Naïve Bayes Classifier

- If A_k is continuous-valued, $P(x_k|C_i)$ is usually computed based on Gaussian distribution with a mean μ and standard deviation σ

$$g(x,\mu,\sigma)=\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{(x-\mu)^2}{2\sigma^2}} \qquad (1)$$

and $P(x_k|C_i)$ is

$$P(x_k \mid C_i) = g(x_k, \mu_{C_i}, \sigma_{C_i})$$

Here, mean μ and standard deviation σ are estimated based on the values of attribute A_k for training tuples of class C_i.

Ex. Let X = (35, $40K), where A1 and A2 are the attribute *age* and *income, class label is buys_computer.*

To calculate P(age = 35 | buys_computer = yes)

1. Estimate the mean and standard deviation of the age attribute for customers in D who buy a computer. Let us say μ = 38 and σ =12.

2. calculate the probability with equation (1).

Naïve Bayes Classifier: Comments

- Advantages
  - Easy to implement
  - Good results obtained in most of the cases

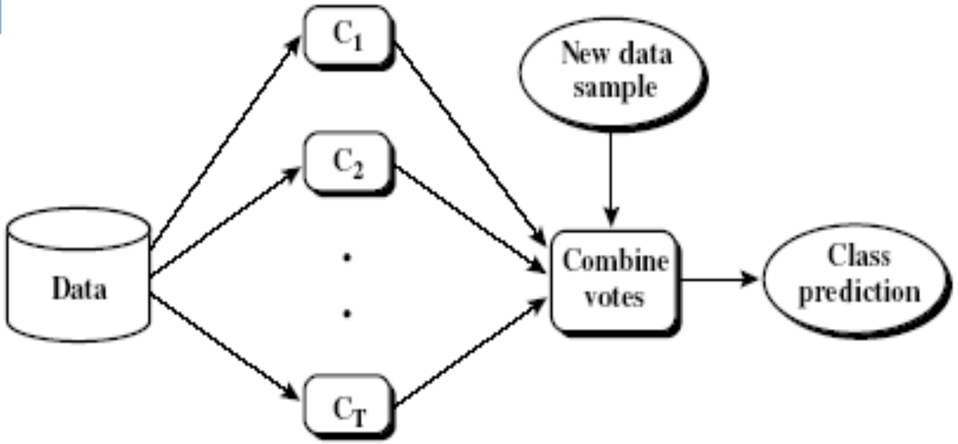
- Disadvantages
  - Assumption: class conditional independence, therefore loss of accuracy
  - Practically, dependencies exist among variables
    - E.g., hospitals: patients: Profile: age, family history, etc.
      Symptoms: fever, cough etc., Disease: lung cancer, diabetes, etc.
    - Dependencies among these cannot be modeled by Naïve Bayes Classifier

- How to deal with these dependencies? Bayesian Belief Networks (Chapter 9 in Han et al.)

Classification: Basic Concepts

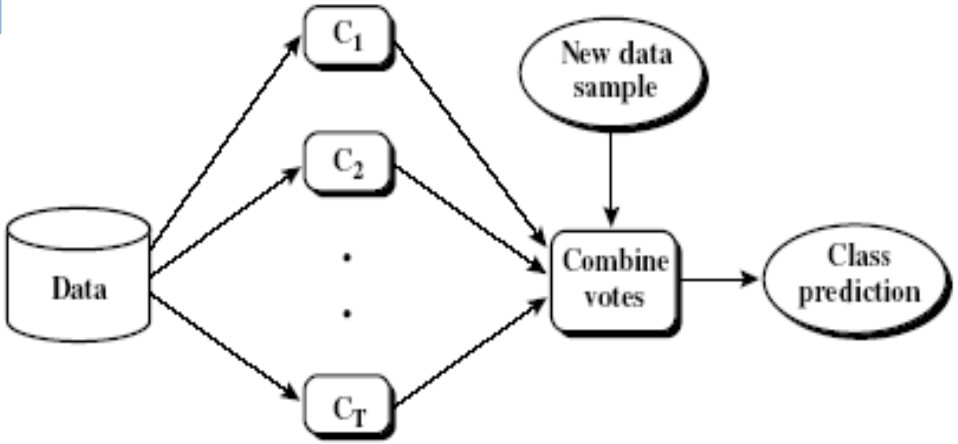
- Classification: Basic Concepts
- Decision Tree Induction
- Model Evaluation and Selection
- Practical Issues of Classification
- Bayes Classification Methods
- Techniques to Improve Classification Accuracy: Ensemble Methods

Ensemble Methods: Increasing the Accuracy



- Ensemble methods
  - Use a combination of models to increase accuracy
  - Combine a series of k learned models, M_1, M_2, …, M_k, with the aim of creating an improved model M^*

Ensemble Methods: Increasing the Accuracy



- Ensemble methods
  - Use a combination of models to increase accuracy
  - Combine a series of k learned models, M_1, M_2, …, M_k, with the aim of creating an improved model M*
- Popular ensemble methods
  - Bagging: averaging the prediction over a collection of classifiers
  - Boosting: weighted vote with a collection of classifiers
  - Random forests: Imagine that each of the classifiers in the ensemble is a decision tree classifier so that the collection of classifiers is a "forest"

Bagging: Boostrap Aggregation

- Analogy: Diagnosis based on multiple doctors' majority vote
- Training
  - Given a set D of *d* tuples, at each iteration *i*, a training set D_i of *d* tuples is sampled with replacement from D (i.e., bootstrap)
  - A classifier model M_i is learned for each training set D_i
- Classification: classify an unknown sample **X**
  - Each classifier M_i returns its class prediction
  - The bagged classifier M* counts the votes and assigns the class with the most votes to **X**
- Regression: can be applied to the prediction of continuous values by taking the average value of each prediction for a given test tuple
- Accuracy: Proved improved accuracy in prediction
  - Often significantly better than a single classifier derived from D
  - For noise data: not considerably worse, more robust

Boosting

- Analogy: Consult several doctors, based on a combination of weighted diagnoses—weight assigned based on the previous diagnosis accuracy
- How boosting works?
  - **Weights** are assigned to each training tuple
  - A series of k classifiers is iteratively learned
  - After a classifier M_i is learned, the weights are updated to allow the subsequent classifier, M_{i+1}, to **pay more attention to the training tuples that were misclassified** by M_i
  - The final **M* combines the votes** of each individual classifier, where the weight of each classifier's vote is a function of its accuracy
- Boosting algorithm can be extended for numeric prediction
- Comparing with bagging: Boosting tends to have greater accuracy, but it also risks overfitting the model to misclassified data

Adaboost (Freund and Schapire, 1997)

- Given a set of *d* class-labeled tuples, ($\mathbf{X_1}$, y_1), …, ($\mathbf{X_d}$, y_d)
- Initially, all the weights of tuples are set the same (1/d)
- Generate k classifiers in k rounds. At round i,
  - Tuples from D are sampled (with replacement) to form a training set D_i of the same size
  - Each tuple's chance of being selected is based on its weight
  - A classification model M_i is derived from D_i
  - Its error rate is calculated using D_i as a test set
  - If a tuple is misclassified, its weight is increased, o.w. it is decreased
- Error rate: err($\mathbf{X_j}$) is the misclassification error of tuple $\mathbf{X_j}$. Classifier M_i error rate is the sum of the weights of the misclassified tuples:

$$error(M_i) = \sum_{j}^{d} w_j \times err(\mathbf{X_j})$$

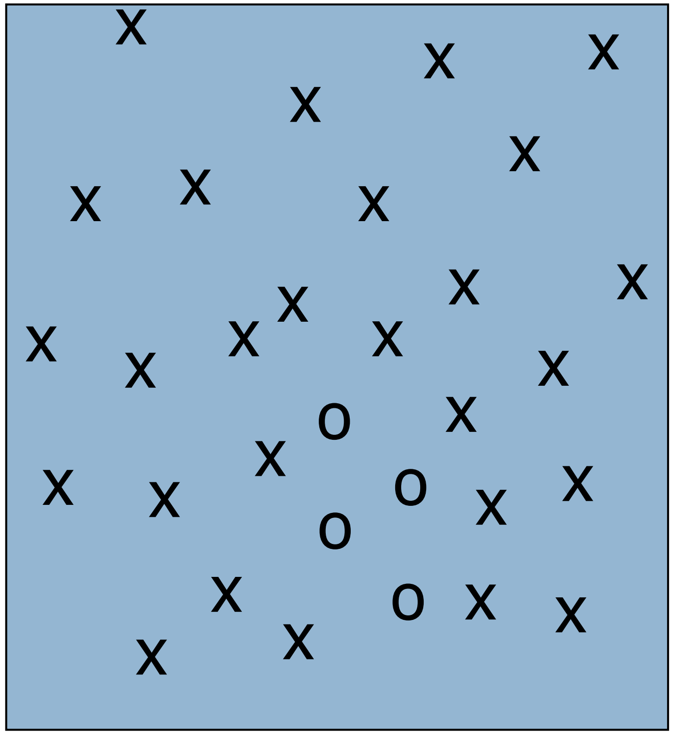
- The weight of classifier M_i's vote is $\log \frac{1 - error(M_i)}{error(M_i)}$

Random Forest (Breiman 2001)

- Random Forest:
  - Each classifier in the ensemble is a *decision tree* classifier and is generated using a random selection of attributes at each node to determine the split
  - During classification, each tree votes and the most popular class is returned
- Two Methods to construct Random Forest:
  - Forest-RI (*random input selection*): Randomly select, at each node, F attributes as candidates for the split at the node. The CART methodology is used to grow the trees to maximum size
  - Forest-RC (*random linear combinations*): Creates new attributes (or features) that are a linear combination of the existing attributes (reduces the correlation between individual classifiers)
- Comparable in accuracy to Adaboost, but more robust to errors and outliers
- Insensitive to the number of attributes selected for consideration at each split, and faster than bagging or boosting

Classification of Class-Imbalanced Data Sets

- Class-imbalance problem: Rare positive example but numerous negative ones, e.g., medical diagnosis, fraud, oil-spill, fault, etc.
- Traditional methods assume a balanced distribution of classes and equal error costs: not suitable for class-imbalanced data



- Typical methods in two-class classification:
  - **Oversampling**: re-sampling of data from positive class
  - **Under-sampling**: randomly eliminate tuples from negative class
  - **Threshold-moving**: move the decision threshold, t, so that the rare class tuples are easier to classify, and hence, less chance of costly false negative errors
  - **Ensemble techniques**: Ensemble multiple classifiers
- Still difficult for class imbalance problem on multiclass tasks