Data Mining: Concepts and Techniques

(3rd ed.)

— Chapter 8 —

Jianjun Cheng

School of Information Science & Engineering

Lanzhou University

Chapter 8. Classification: Basic Concepts

Machine Learming

- Supervised learning (classification)
  - the task of inferring a function from labeled training data
  - each example in training data is a pair consisting of an input object and a desired output value
  - a supervised learning algorithm analyzes the traning data and produces an inferred function, which can be used for mapping new examples

Machine Learming

- Supervised learning (classification)
  - Given a set of N training examples of the form $\{(x_1,y_1), \ldots, (x_N, y_N)\}$ such that x_i is the feature vector of the i-th example and y_i is its label.
  - A learning algorithm seeks a function $f: X \rightarrow Y$, where X is the input space and Y is the output space.
  - The function f is an element of some space of possible functions F, usually called the hypothesis space.
  - Define a scoring function $g: X \times Y \rightarrow R$, then

$$f(x)=\arg\max_y g(x, y).$$

Machine Learming

- Supervised learning (classification)
  - an optimal scenario will allow for the algorithm to correctly determine the class labels for unseen instances
  - requires the algorithm to generalize from the training data to unseen situations in a "reasonalbe" way
  - Supervision: The training data (observations, measurements, etc.) are accompanied by labels indicating the class of the observations

Machine Learming

- Unsupervised learning (clustering)
  - The task of inferring a function to describe hidden structure from unlabeled data
  - The class labels of training data is unknown
  - Given a set of measurements, observations, etc. with the aim of establishing the existence of classes or clusters in the data
  - Partition the dataset into some groups, data points within the same group are more similar than those inter groups

Machine Learming

- Semi-supervised learning
  - a class of supervised learning tasks and techniques that also make use of unlabeled data for training ---- typically a small amount of labeled data with a large amount of unlabeled data
  - falls between unsupervised learning and supervised learning
  - inspired by observations by many machine-learning researchers: when used in conjuction with a small amount of labeled data, the learner can produce considerable improvement in learning accuracy

Machine Learming

- Semi-supervised learning
  - the acquisition of labeled data for a learning problem often requires a skilled human agent or a physical experiment
  - the cost associated with the labeling process thus may render a fully labeled training set infeasible, whereas acquisition of unlabeled data is relatively inexpensive
  - in situations mentioned above, semi-supervised learning can be of great practical value.

Machine Learming

- Semi-supervised learning
  - given a set of *l* examples $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_l \in \mathbf{X}$ with corresponding labels $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_l \in \mathbf{Y}$, and **u** unlabeled examples $\mathbf{x}_{l+1}, \mathbf{x}_{l+2}, \ldots, \mathbf{x}_{l+u} \in \mathbf{X}$.
  - make use of this combined information to surpass the classification performance that could be obtained either by discarding the unlabeled data and doing supervised learning or by discarding the labels and doing unsupervised learning.
  - transductive learning: infer the correct labels for the given unlabeled data $\mathbf{x}_{l+1}, \ldots, \mathbf{x}_{l+u}$ only ---- homework
  - inductive learning: infer the correct mapping from X to Y ---- class examination

Machine Learming

- Semi-supervised learning
  - two types of semi-supervised components: labeled data, pair-wise constraints
  - labeled data
  - pair-wise constraints: Must-Link and Cannot-Link
  - Must-Link: **(a,b)** $\in$ **C_m** ⇨ **a** and **b** must be classified into the same class
  - Cannot-Link: **(a,b)** $\in$ **C_c** ⇨ **a** and **b** cannot be classifed into the same class, and they must be allocated into different classes

Machine Learming

- Active learning
  - a special case of semi-supervised learning, in which a learning algorithm is able to interactively query an **oracle** to obtain the desired outputs at new data points
  - in many situations, unlabeled data is aboundant, but manually labeling is expensive
  - the learner can choose some examples actively to query the **oracle** for their labels
  - its goal is to maximize the leaner's performance with the least cost
  - the number of selected examples are often much lower than the number required in normal supervised learning

Machine Learming

- Active learning
  - Query strategies: select the most informative data points or the data points with least certainty
    - Uncertainty sampling: label those points for which the current model is least certain as to what the correct output should be
    - Query by committee: a variety of models are trained on the current labeled data, and vote on the output for unlabeled data; label those points for which the "committee" disagrees the most
    - Expected model change: label those points that would most change the current model

Machine Learning

- Active learning
  - Query strategies: select the most informative data points or the data points with least certainty
    - Expected error reduction: label those points that would most reduce the model's generalization error
    - Variance reduction: label those points that would minimize output variance, which is one of the components of error

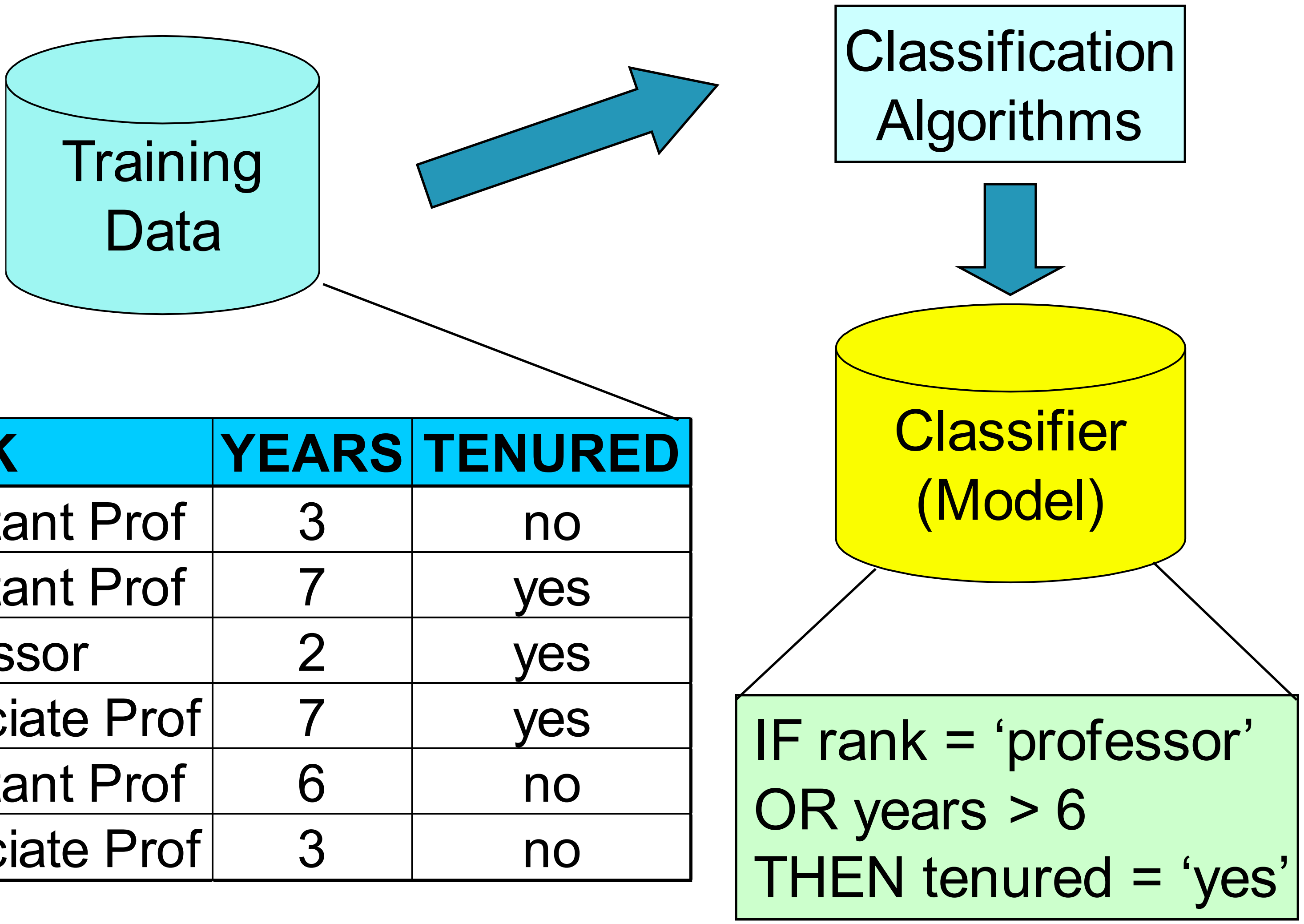
Prediction Problems: Classification vs. Numeric Prediction

- Classification
  - predicts categorical class labels (discrete or nominal)
  - classifies data (constructs a model) based on the training set and the values (class labels) in a classifying attribute and uses it in classifying new data
- Numeric Prediction
  - models continuous-valued functions, i.e., predicts unknown or missing values
- Typical applications
  - Credit/loan approval:
  - Medical diagnosis: if a tumor is cancerous or benign
  - Fraud detection: if a transaction is fraudulent
  - Web page categorization: which category it is

Classification—A Two-Step Process

- Model construction: describing a set of predetermined classes
  - Each tuple/sample is assumed to belong to a predefined class, as determined by the class label attribute
  - The set of tuples used for model construction is training set
  - The model is represented as classification rules, decision trees, or mathematical formulae
- Model usage: for classifying future or unknown objects
  - Estimate accuracy of the model
    - The known label of test sample is compared with the classified result from the model
    - Accuracy rate is the percentage of test set samples that are correctly classified by the model
    - Test set is independent of training set (otherwise overfitting)
  - If the accuracy is acceptable, use the model to classify new data
- Note: If *the test set* is used to select models, it is called validation (test) set

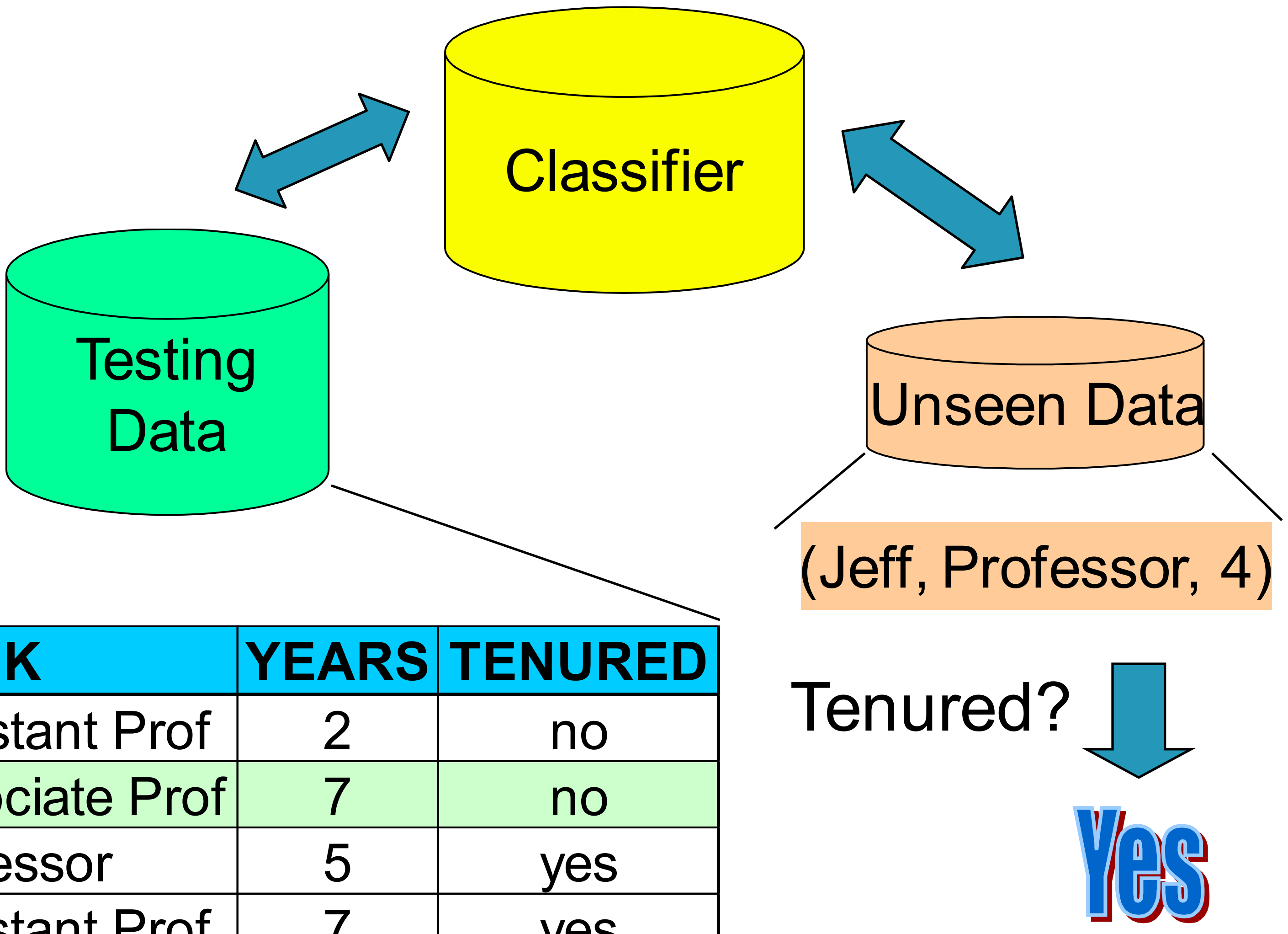
Process (1): Model Construction



NAME	RANK	YEARS	TENURED
Mike	Assistant Prof	3	no
Mary	Assistant Prof	7	yes
Bill	Professor	2	yes
Jim	Associate Prof	7	yes
Dave	Assistant Prof	6	no
Anne	Associate Prof	3	no

IF rank = ‘professor’
OR years > 6
THEN tenured = ‘yes’

Process (2): Using the Model in Prediction



NAME	RANK	YEARS	TENURED
Tom	Assistant Prof	2	no
Merlisa	Associate Prof	7	no
George	Professor	5	yes
Joseph	Assistant Prof	7	yes

Chapter 8. Classification: Basic Concepts

- Classification: Basic Concepts
- ☞ Decision Tree Induction
- Bayes Classification Methods
- Rule-Based Classification
- Model Evaluation and Selection
- Techniques to Improve Classification Accuracy: Ensemble Methods
- Summary

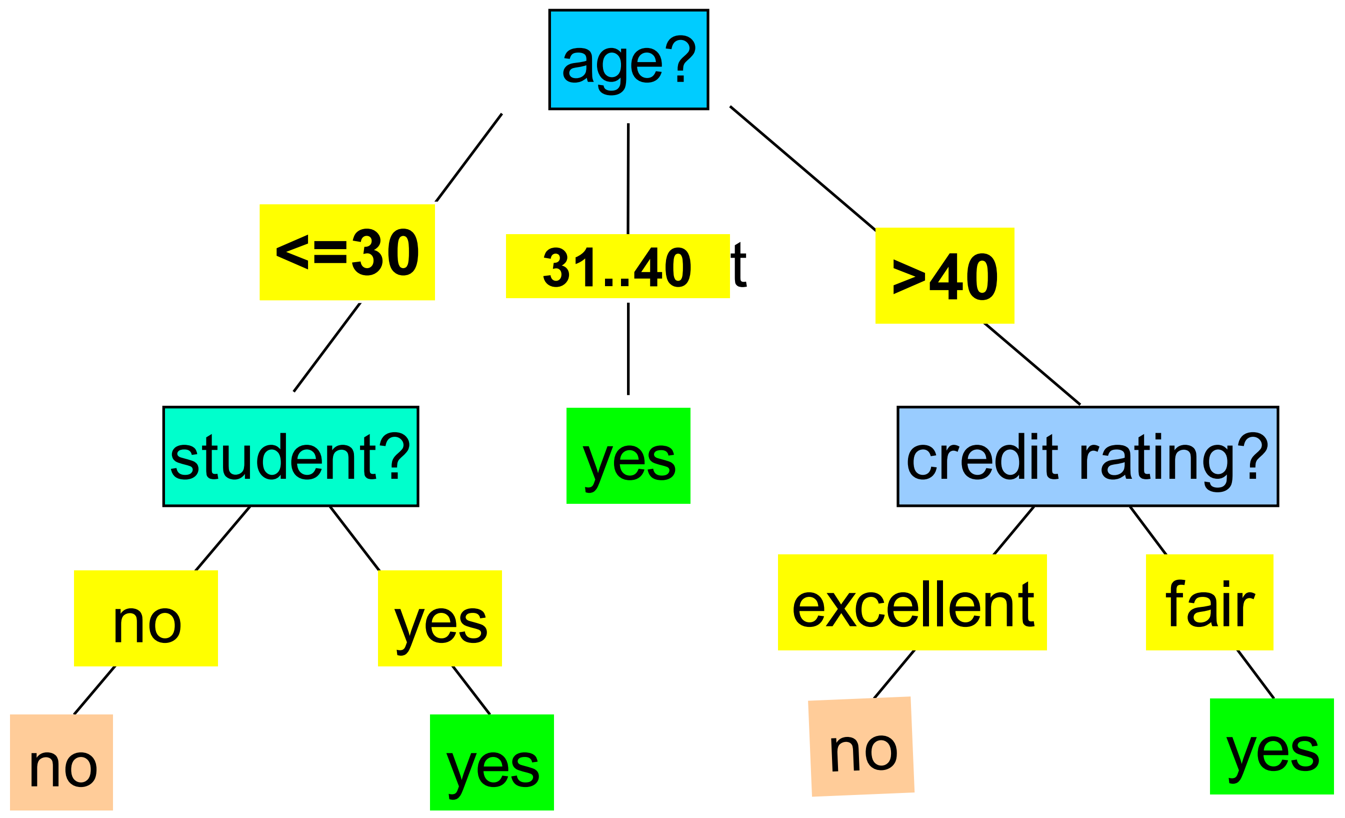
Decision Tree Induction: An Example

- Training data set: Buys_computer

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

Decision Tree Induction: An Example

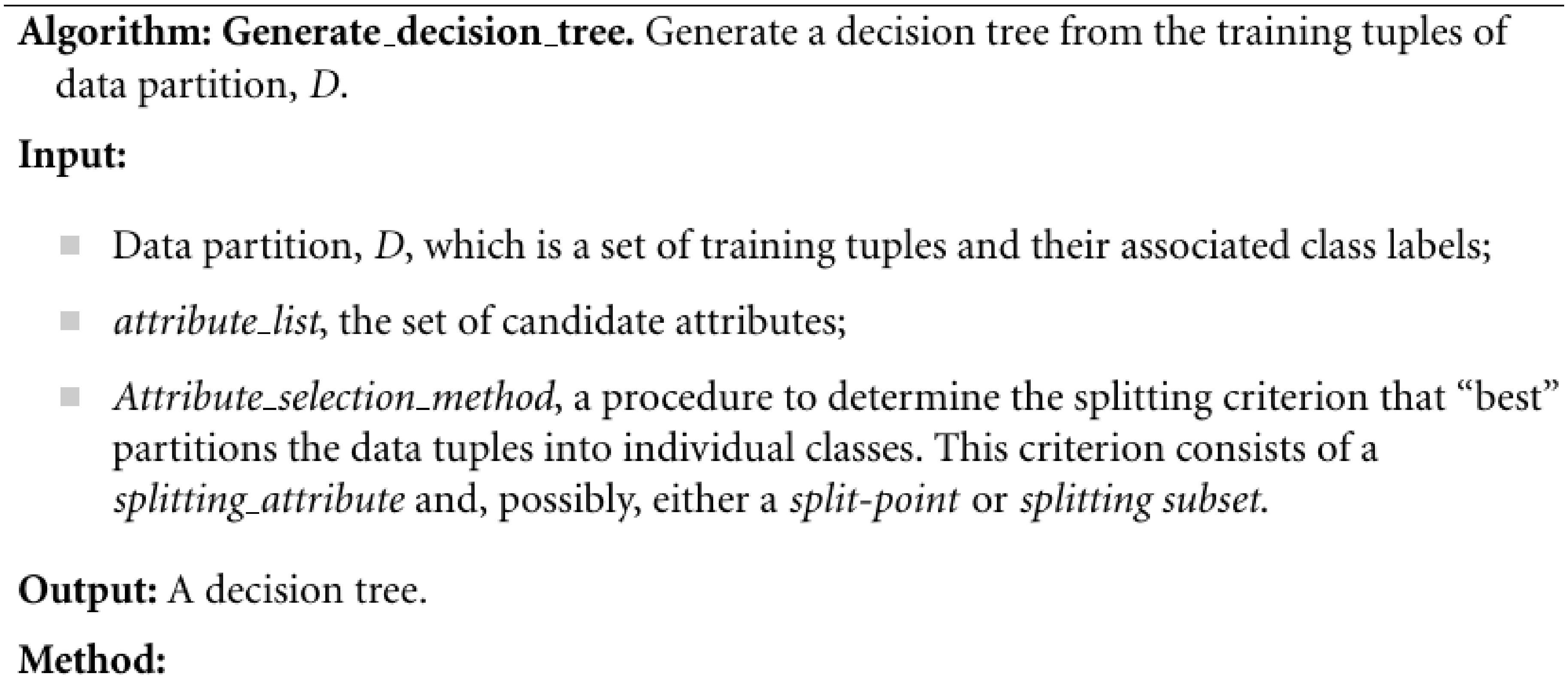
- Resulting tree:



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



Algorithm: Generate_decision_tree. Generate a decision tree from the training tuples of data partition, *D*.

Input:

- Data partition, *D*, which is a set of training tuples and their associated class labels;
- *attribute_list*, the set of candidate attributes;
- *Attribute_selection_method*, a procedure to determine the splitting criterion that "best" partitions the data tuples into individual classes. This criterion consists of a *splitting_attribute* and, possibly, either a *split-point* or *splitting subset*.

Output: A decision tree.

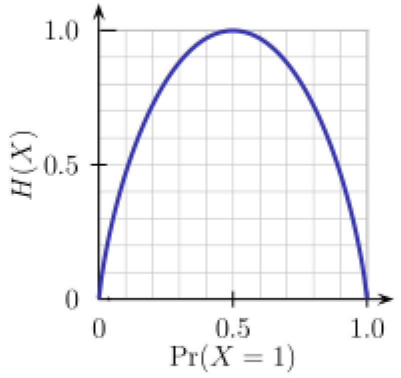
Method:

Algorithm

```
(1)  create a node $N$;
(2)  if tuples in $D$ are all of the same class, $C$, then
(3)      return $N$ as a leaf node labeled with the class $C$;
(4)  if attribute_list is empty then
(5)      return $N$ as a leaf node labeled with the majority class in $D$; // majority voting
(6)  apply Attribute_selection_method($D$, attribute_list) to find the "best" splitting_criterion;
(7)  label node $N$ with splitting_criterion;
(8)  if splitting_attribute is discrete-valued and
         multiway splits allowed then // not restricted to binary trees
(9)      attribute_list $\leftarrow$ attribute_list $-$ splitting_attribute; // remove splitting_attribute
(10) for each outcome $j$ of splitting_criterion
     // partition the tuples and grow subtrees for each partition
(11)     let $D_j$ be the set of data tuples in $D$ satisfying outcome $j$; // a partition
(12)     if $D_j$ is empty then
(13)         attach a leaf labeled with the majority class in $D$ to node $N$;
(14)     else attach the node returned by Generate_decision_tree($D_j$, attribute_list) to node $N$;
     endfor
(15) return $N$;
```

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 $\{y_1, \dots, y_m\}$,
    - $H(Y) = -\sum_{i=1}^{m} p_i \log(p_i)$, where $p_i = P(Y = y_i)$
  - Interpretation:
    - Higher entropy => higher uncertainty
    - Lower entropy => lower uncertainty
- Conditional Entropy
  - $H(Y|X) = \sum_x p(x) H(Y|X = x)$



m = 2

Attribute Selection Measure: Information Gain (ID3)

- 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"

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

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

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

$\frac{5}{14}I(2,3)$ means "age <=30" has 5 out of 14 samples, with 2 yes'es and 3 no's. Hence

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

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

Similarly,

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

Computing Information-Gain for Continuous-Valued Attributes

- Let attribute A be a continuous-valued attribute
- Must determine the *best split point* for A
  - Sort the value A in increasing order
  - 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}
  - The point with **the minimum expected information requirement** for A, **$Info_A(D)$** is selected as the split-point for A
- Split:
  - D1 is the set of tuples in D satisfying A ≤ split-point, and D2 is the set of tuples in D satisfying A > split-point

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)

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

GainRatio(A) = Gain(A)/SplitInfo(A)

- Ex.

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

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(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 *gini(D)* 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 $gini_{split}(D)$ (or the largest reduction in impurity, **Δgini(A)**) is chosen to split the node (*need to enumerate all the possible splitting points for each attribute*)

Gini index (CART, IBM IntelligentMiner)

- Split criterion for attribute A for Gini index: binary split
- For discrete-valued attributes:
  - A have v distinct values, $\{a_1, a_2, \ldots, a_v\}$, occuring in D
  - examine all of the possible subsets that can be formed using known values of A: each subset, S_A, can be considered as a binary test for attribute A of the form "$A \in S_A$"
  - give a tuple, this test is satisfied if the value of A for the tuple is among the values listed in S_A
  - empty set and the power set are excluded, therefore, there are 2^v-2 possible ways to form two partitions of the data, D, based on a binary split on A
- For continuous-valued attributes:
  - each possible split-point must be considered
  - strategy is similar to that used for information gain

Computation of Gini Index

- Ex. D has 9 tuples in buys_computer = “yes” and 5 in “no”

$$gini(D) = 1 - \left(\frac{9}{14}\right)^2 - \left(\frac{5}{14}\right)^2 = 0.459$$

- Suppose the attribute income partitions D into 10 in D_1: {low, medium} and 4 in D_2: {high}

$$gini_{income \in \{low, medium\}}(D) = \left(\frac{10}{14}\right) Gini(D_1) + \left(\frac{4}{14}\right) Gini(D_2)$$

$$\begin{aligned}
&= \frac{10}{14}\left(1 - \left(\frac{7}{10}\right)^2 - \left(\frac{3}{10}\right)^2\right) + \frac{4}{14}\left(1 - \left(\frac{2}{4}\right)^2 - \left(\frac{2}{4}\right)^2\right) \\
&= 0.443 \\
&= Gini_{income \in \{high\}}(D).
\end{aligned}$$

  $Gini_{\{low,high\}}$ is 0.458; $Gini_{\{medium,high\}}$ is 0.450. Thus, split on the {low,medium} (and {high}) since it has the lowest Gini index

- All attributes are assumed continuous-valued
- May need other tools, e.g., clustering, to get the possible split values
- Can be modified for categorical attributes

Comparing Attribute Selection Measures

- The three measures, in general, return good results but
  - **Information gain**:
    - biased towards multivalued attributes
  - **Gain ratio**:
    - tends to prefer unbalanced splits in which one partition is much smaller than the others
  - **Gini index**:
    - biased to multivalued attributes
    - has difficulty when # of classes is large
    - tends to favor tests that result in equal-sized partitions and purity in both partitions

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

Overfitting and Tree Pruning

- Overfitting: An induced tree may overfit the training data
  - Too many branches, some may reflect anomalies due to noise or outliers
  - Poor accuracy for unseen samples
- Two approaches to avoid overfitting
  - Prepruning: *Halt tree construction early*-do not split a node if this would result in the goodness measure falling below a threshold
    - Difficult to choose an appropriate threshold
  - Postpruning: *Remove branches* from a "fully grown" tree—get a sequence of progressively pruned trees
    - Use a set of data different from the training data to decide which is the "best pruned tree"

Enhancements to Basic Decision Tree Induction

- Allow for **continuous-valued attributes**
  - Dynamically define new discrete-valued attributes that partition the continuous attribute value into a discrete set of intervals
- Handle **missing attribute values**
  - Assign the most common value of the attribute
  - Assign probability to each of the possible values
- **Attribute construction**
  - Create new attributes based on existing ones that are sparsely represented
  - This reduces fragmentation, repetition, and replication

Classification in Large Databases

- Classification—a classical problem extensively studied by statisticians and machine learning researchers
- Scalability: Classifying data sets with millions of examples and hundreds of attributes with reasonable speed
- Why is decision tree induction popular?
  - relatively faster learning speed (than other classification methods)
  - convertible to simple and easy to understand classification rules
  - can use SQL queries for accessing databases
  - comparable classification accuracy with other methods
- RainForest (VLDB'98 — Gehrke, Ramakrishnan & Ganti)
  - Builds an AVC-list (attribute, value, class label)

Scalability Framework for RainForest

- Separates the scalability aspects from the criteria that determine the quality of the tree
- Builds an AVC-list: **AVC (Attribute, Value, Class_label)**
- **AVC-set** (of an attribute *X*)
  - Projection of training dataset onto the attribute *X* and class label where counts of individual class label are aggregated
- **AVC-group** (of a node *n*)
  - Set of AVC-sets of all predictor attributes at the node *n*

Rainforest: Training Set and Its AVC Sets

Training Examples

age	income	student	credit_rating	comp
<=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

AVC-set on *Age*

Age	Buy_Computer	
	yes	no
<=30	2	3
31..40	4	0
>40	3	2

AVC-set on *income*

income	Buy_Computer	
	yes	no
high	2	2
medium	4	2
low	3	1

AVC-set on *Student*

student	Buy_Computer	
	yes	no
yes	6	1
no	3	4

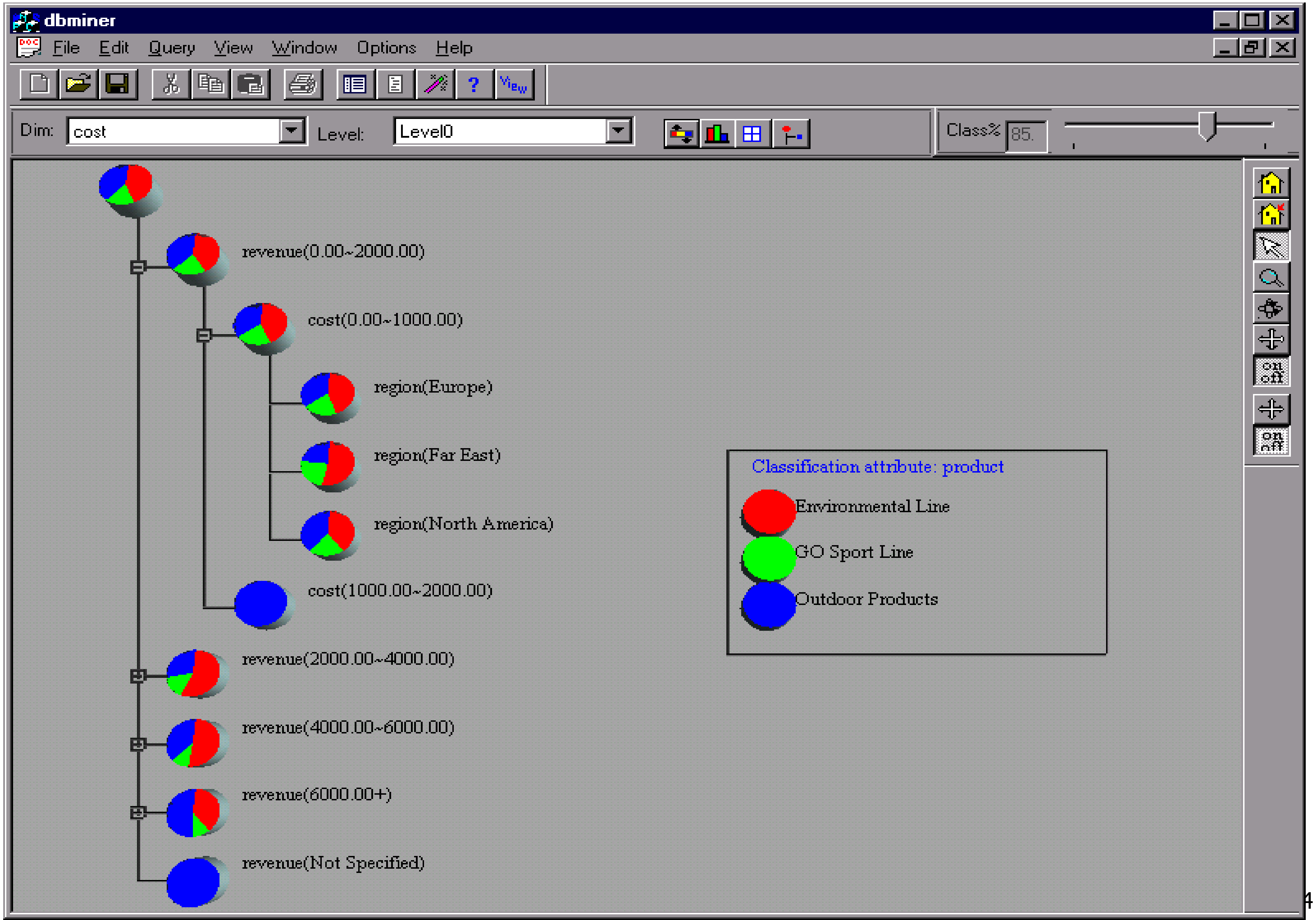
AVC-set on *credit_rating*

Credit rating	Buy_Computer	
	yes	no
fair	6	2
excellent	3	3

BOAT (Bootstrapped Optimistic Algorithm for Tree Construction)

- Use a statistical technique called *bootstrapping* to create several smaller samples (subsets), each fits in memory
- Each subset is used to create a tree, resulting in several trees
- These trees are examined and used to construct a new tree *T'*
  - It turns out that *T'* is very close to the tree that would be generated using the whole data set together
- Adv: requires only two scans of DB, an incremental alg.

Presentation of Classification Results

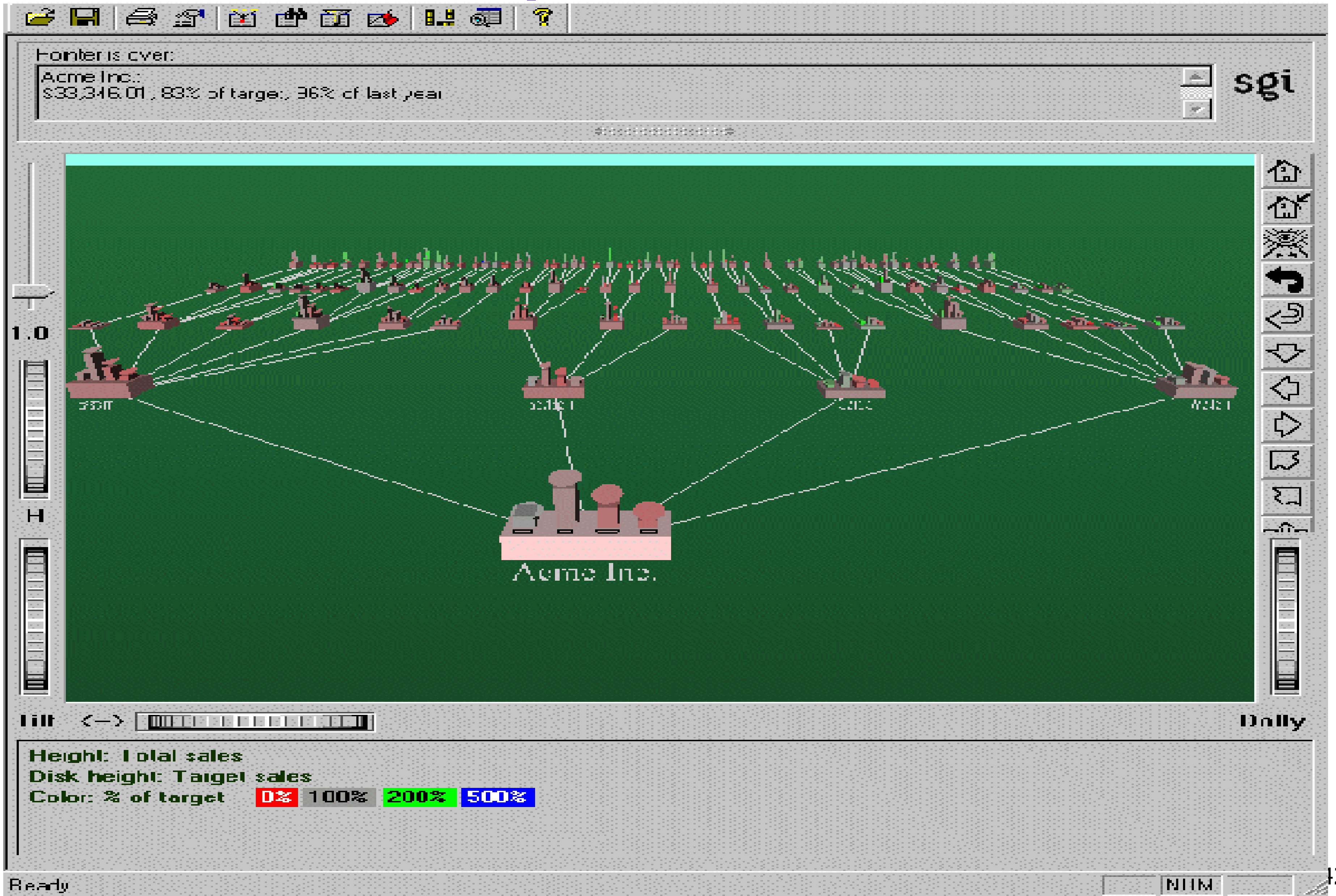


revenue(0.00~2000.00)
cost(0.00~1000.00)
region(Europe)
region(Far East)
region(North America)
cost(1000.00~2000.00)
revenue(2000.00~4000.00)
revenue(4000.00~6000.00)
revenue(6000.00+)
revenue(Not Specified)

Classification attribute: product

🟥	Environmental Line
🟩	GO Sport Line
🟦	Outdoor Products

Visualization of a Decision Tree in SGI/MineSet 3.0

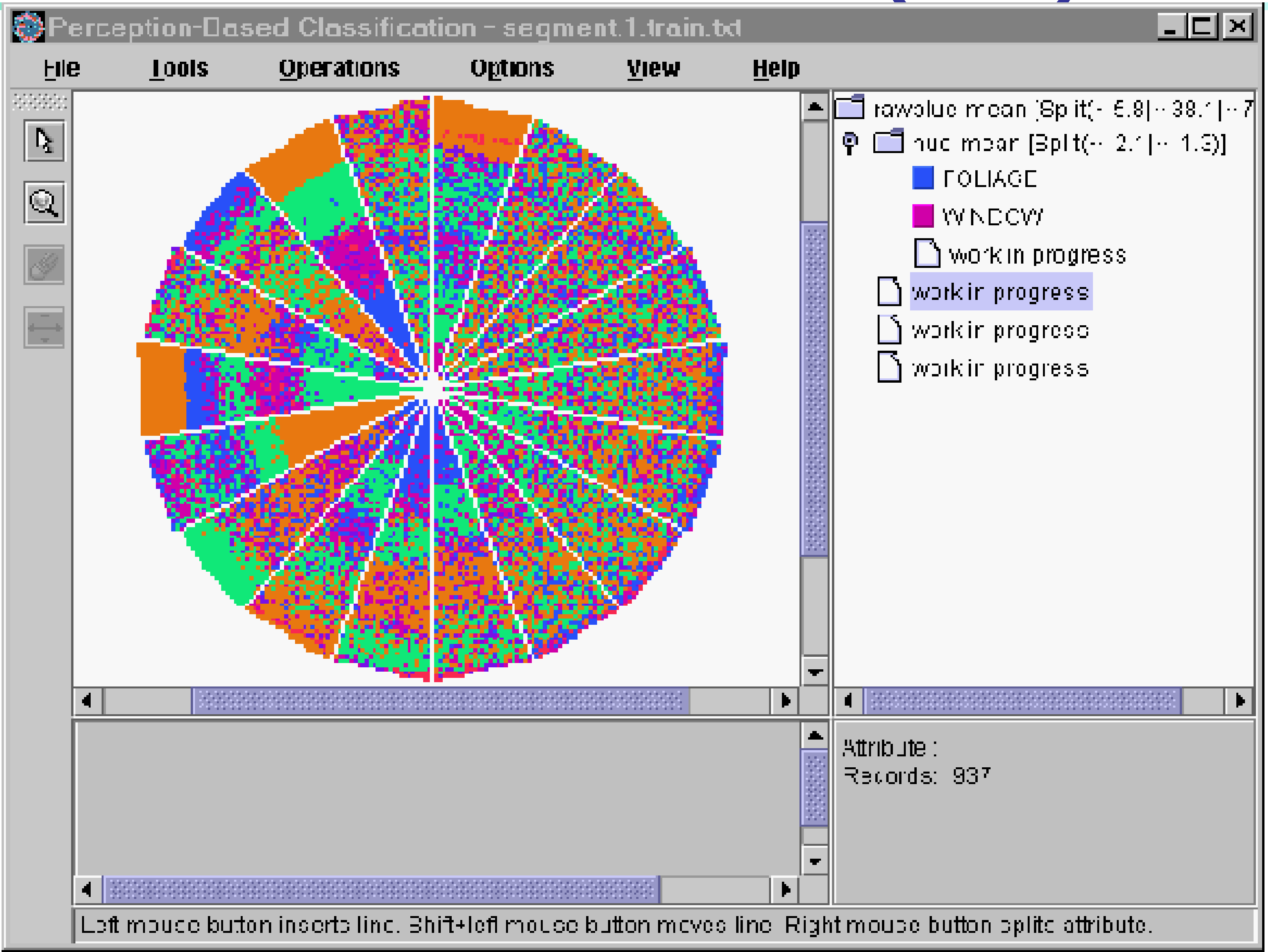


Pointer is over:

Acme Inc.:
$33,346.01, 83% of target, 36% of last year

Height: Total sales
Disk height: Target sales
Color: % of target 🟥 0% ▨ 100% 🟩 200% 🟦 500%

Interactive Visual Mining by Perception-Based Classification (PBC)



rawvalue mean [Split(- 5.8|- 38.1|- 7

nuc mear [Split(- 2.1|- 1.3)]

🟦 FOLIAGE

🟪 WINDOW

work in progress

work in progress

work in progress

work in progress

Attribute :
Records: 937

Left mouse button inserts line. Shift+left mouse button moves line. Right mouse button splits attribute.

Chapter 8. Classification: Basic Concepts

- Classification: Basic Concepts
- Decision Tree Induction
- ☞ Bayes Classification Methods
- Rule-Based Classification
- Model Evaluation and Selection
- Techniques to Improve Classification Accuracy: Ensemble Methods
- Summary

Bayesian Classification: Why?

- A statistical classifier: performs *probabilistic prediction, i.e.,* predicts class membership probabilities
- Foundation: Based on Bayes' Theorem.
- Performance: A simple Bayesian classifier, **naïve Bayesian classifier**, has comparable performance with decision tree and selected neural network classifiers
- Incremental: Each training example can incrementally increase/decrease the probability that a hypothesis is correct — prior knowledge can be combined with observed data
- Standard: Even when Bayesian methods are computationally intractable, they can provide a standard of optimal decision making against which other methods can be measured

Bayes' Theorem: Basics

- Total probability Theorem: $P(B) = \sum_{i=1}^{M} P(B \mid A_i)P(A_i)$
- Bayes' Theorem:

$$P(H \mid \mathbf{X}) = \frac{P(\mathbf{X} \mid H)P(H)}{P(\mathbf{X})}$$

- 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., posteriori probability): the probability that the hypothesis holds given the observed data sample **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**, posteriori probability of a hypothesis **H**, **P(H|X)**, follows the Bayes' theorem

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

- Informally, this can be viewed as

  posteriori = likelihood $\times$ 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 Posteriori

- Let D be a training set of tuples and their associated class labels, and each tuple is represented by an n-D attribute vector **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$_i$|X)**
- This can be derived from Bayes' theorem

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

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

$$P(C_i|X) \propto P(X|C_i)P(C_i)$$

needs to be maximized

Naïve Bayes Classifier

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

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

- This greatly reduces the computation cost: only counts the class distribution
- 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 of C_i in D)
- If A_k is continous-valued, $P(x_k|C_i)$ is usually computed based on Gaussian distribution with a mean μ and standard deviation σ

and $P(x_k|C_i)$ is

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

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

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	comp
<=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

- **C_1: buys_computer="yes"** $P(C_1) = 9/14 = 0.643$

  C_2: buys_computer="no" $P(C_2) = 5/14 = 0.357$
- Compute $P(X|C_i)$ for each class

  $P(\text{age} = \text{“<=30”} \mid C_1) = 2/9 = 0.222$

  $P(\text{age} = \text{“<= 30”} \mid C_2) = 3/5 = 0.6$

  $P(\text{income} = \text{“medium”} \mid C_1) = 4/9 = 0.444$

  $P(\text{income} = \text{“medium”} \mid C_2) = 2/5 = 0.4$

  $P(\text{student} = \text{“yes”} \mid C_1) = 6/9 = 0.667$

  $P(\text{student} = \text{“yes”} \mid C_2) = 1/5 = 0.2$

  $P(\text{credit\_rating} = \text{“fair”} \mid C_1) = 6/9 = 0.667$

  $P(\text{credit\_rating} = \text{“fair”} \mid C_2) = 2/5 = 0.4$

age	income	student	credit_ratin	comp
<=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

- **X=(age <= 30 , income = medium, student = yes, credit_rating = fair)**

  $P(X|C_i)$: $P(X|C_1) = 0.222 \times 0.444 \times 0.667 \times 0.667 = 0.044$

  $P(X|C_2) = 0.6 \times 0.4 \times 0.2 \times 0.4 = 0.019$

  $P(X|C_i) \times P(C_i)$: $P(X|C_1) \times P(C_1) = 0.028$

  $P(X|C_2) \times P(C_2) = 0.007$

Therefore, X belongs to class C_1 (“buys_computer = yes”)

Naïve Bayesian Classifier: An Example

Build a Naïve Bayesian Classifier from the training data in the table, in whic x_1 and x_2 are attributes. The value of $x_1 \in \{1, 2, 3\}$, $x_2 \in \{S, M, L\}$, and Y is the label of the tubples, i.e., $Y \in C=(1, -1)$. To classify the tuple $t=(2,S)^T$ using the classifier.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
x_1	1	1	1	1	1	2	2	2	2	2	3	3	3	3	3
x_2	S	M	M	S	S	S	M	M	L	L	L	M	M	L	L
Y	-1	-1	1	1	-1	-1	-1	1	1	1	1	1	1	1	-1

Naïve Bayesian Classifier: An Example

C_1: Y=1 C_2: Y=-1

$P(C_1)=9/15$ $P(C_2)=6/15$

$P(x_1=1\|C_1)=2/9$	$P(x_1=2\|C_1)=3/9$	$P(x_1=3\|C_1)=4/9$
$P(x_2=S\|C_1)=1/9$	$P(x_2=M\|C_1)=4/9$	$P(x_2=L\|C_1)=4/9$
$P(x_1=1\|C_2\}=3/6$	$P(x_1=2\|C_2)=2/6$	$P(x_1=3\|C_2)=1/6$
$P(x_2=S\|C_2)=3/6$	$P(x_2=M\|C_2)=2/6$	$P(x_2=L\|C_2)=1/6$

For the tuple $t=(2,S)^T$:

$P(C_1|t) \propto P(C_1)P(x_1=2|C_1)P(x_2=S|C_1)=9/15*3/9*1/9=1/45$

$P(C_2|t) \propto P(C_2)P(x_1=2|C_2)P(x_2=S|C_2)=6/15*2/6*3/6=1/15$

so, the label of $x=(2,S)^T$ is -1

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**

    P(income = low) = 1/1003

    P(income = medium) = 991/1003

    P(income = high) = 11/1003
  - The "corrected" prob. estimates are close to their "uncorrected" counterparts

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)

Chapter 8. Classification: Basic Concepts

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

Using IF-THEN Rules for Classification

- Represent the knowledge in the form of IF-THEN rules

  R: **IF** *age* = youth AND *student* = yes

  THEN *buys_computer* = yes

  - Rule antecedent/precondition vs. rule consequent
- If the condition in a rule antecedent holds true for a given tuple, it is said that the antecedent is **satisfied**, and the rule **covers** the tuple
- Assessment of a rule: *coverage* and *accuracy*
  - n_{covers} = # of tuples covered by R
  - $n_{correct}$ = # of tuples correctly classified by R

  $$\textbf{coverage(R)} = n_{covers} / |D|$$

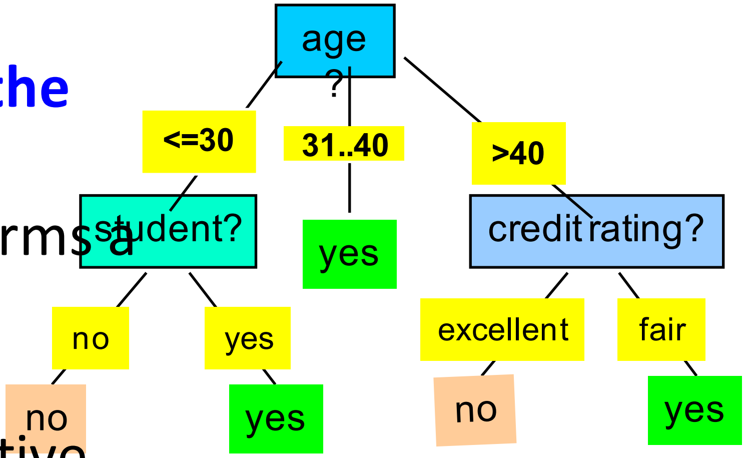
  $$\textbf{accuracy(R)} = n_{correct} / n_{covers}$$
- if a rule is satisfied by a tuple, the rule is said to be **triggered**

Using IF-THEN Rules for Classification

- If more than one rule are triggered, need **conflict resolution**
  - Size ordering: assign the highest priority to the triggering rules that has the "toughest" requirement (i.e., with the *most attribute tests*)
  - Class-based ordering:
    - classes are sorted in order of decreasing "importance" such as by decreasing order of prevalence
    - classes are sorted based on the misclassification cost per class
  - Rule-based ordering (**decision list**): rules are organized into one long priority list, according to some measure of rule quality or by experts
- When there is no rule satisfied by a tuple, how to determine the class label of the tuple?
  - **A fallback or default rule** can be set up to specify a default class, based on the training set.

Rule Extraction from a Decision Tree

- Rules are *easier to understand* than large trees
- **One rule is created *for each path* from the root to a leaf**
- Each attribute-value pair along a path forms a conjunction: the leaf holds the class prediction
- Rules are mutually exclusive and exhaustive



- Example: Rule extraction from our *buys_computer* decision-tree

  IF *age* = young AND *student* = *no* THEN *buys_computer* = *no*

  IF *age* = young AND *student* = *yes* THEN *buys_computer* = *yes*

  IF *age* = mid-age THEN *buys_computer* = *yes*

  IF *age* = old AND *credit_rating* = *excellent* THEN *buys_computer* = *no*

  IF *age* = old AND *credit_rating* = *fair* THEN *buys_computer* = *yes*

Rule Induction: Sequential Covering Method

- **Sequential covering algorithm**: Extracts rules directly from training data
- Typical sequential covering algorithms: FOIL, AQ, CN2, RIPPER
- Rules are learned *sequentially*, each for a given class C_i will cover many tuples of C_i but none (or few) of the tuples of other classes
- Steps:
  - **Rules are learned one at a time**
  - Each time a rule is learned, the tuples covered by the rules are removed
  - Repeat the process on the remaining tuples until *termination condition*, e.g., when no more training examples or when the quality of a rule returned is below a user-specified threshold
- Comp. w. decision-tree induction: learning a set of rules *simultaneously*

Sequential Covering Algorithm

Algorithm: Sequential covering. Learn a set of IF-THEN rules for classification.

Input:

- *D*, a data set of class-labeled tuples;
- *Att_vals*, the set of all attributes and their possible values.

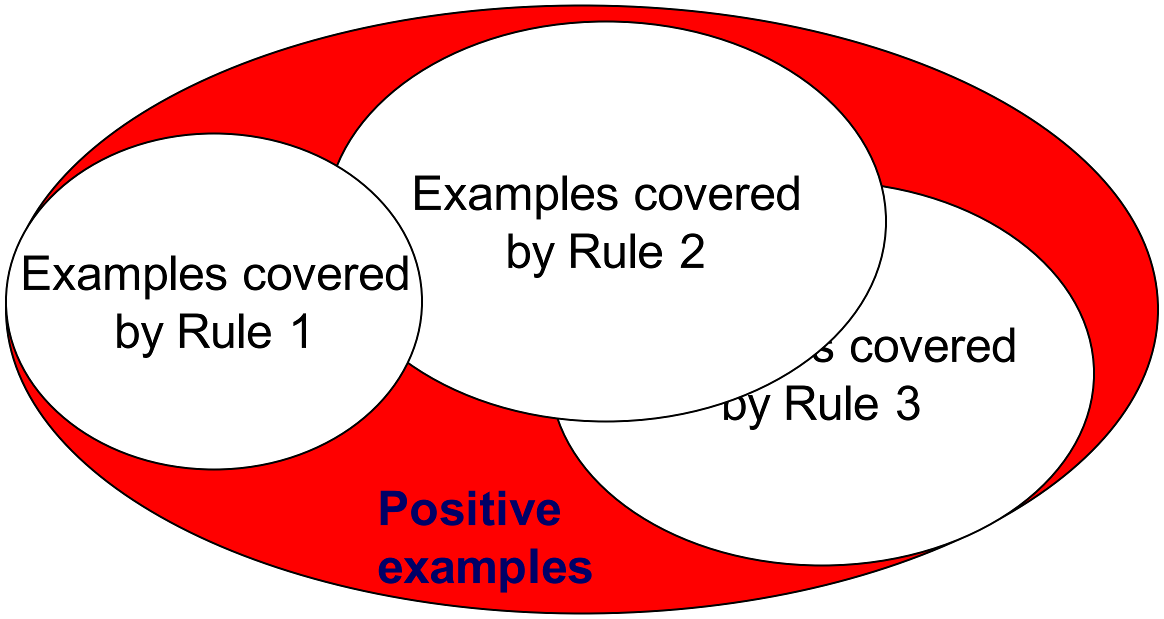
Output: A set of IF-THEN rules.

Method:

```
(1)  Rule_set = {}; // initial set of rules learned is empty
(2)  for each class c do
(3)      repeat
(4)          Rule = Learn_One_Rule(D, Att_vals, c);
(5)          remove tuples covered by Rule from D;
(6)          Rule_set = Rule_set + Rule; // add new rule to rule set
(7)      until terminating condition;
(8)  endfor
(9)  return Rule_Set;
```

Sequential Covering Algorithm

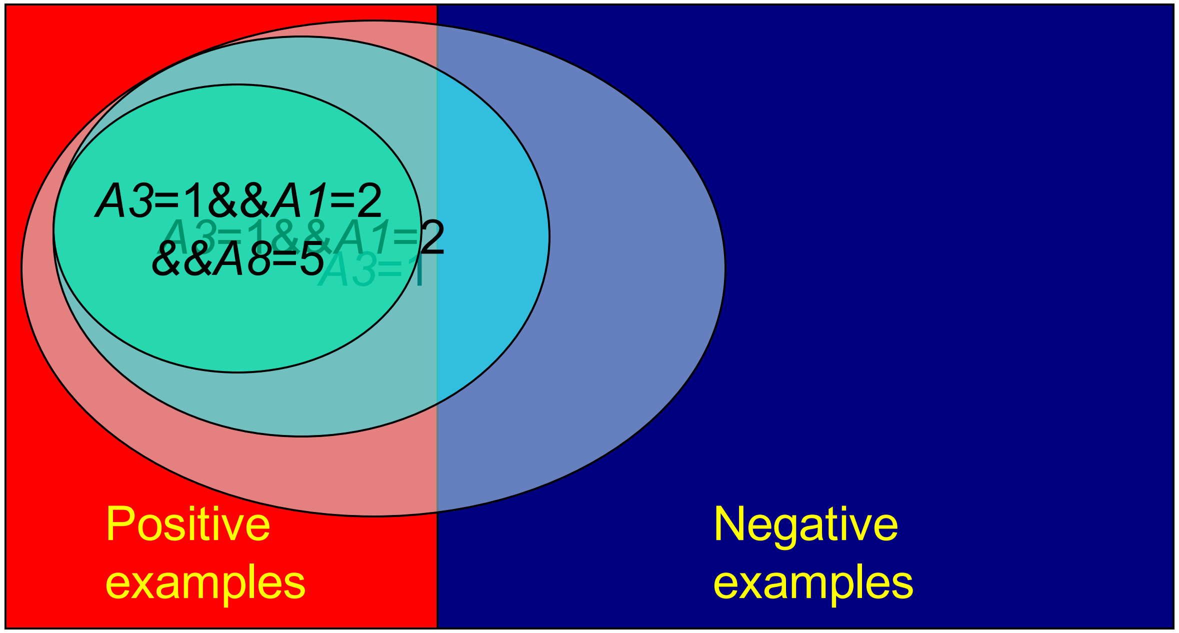
```
while (enough target tuples left)
    generate a rule
    remove positive target tuples satisfying this rule
```



Rule Generation

- To generate a rule

```
while(true)
    find the best predicate p
    if foil-gain(p) > threshold then add p to current rule
    else break
```

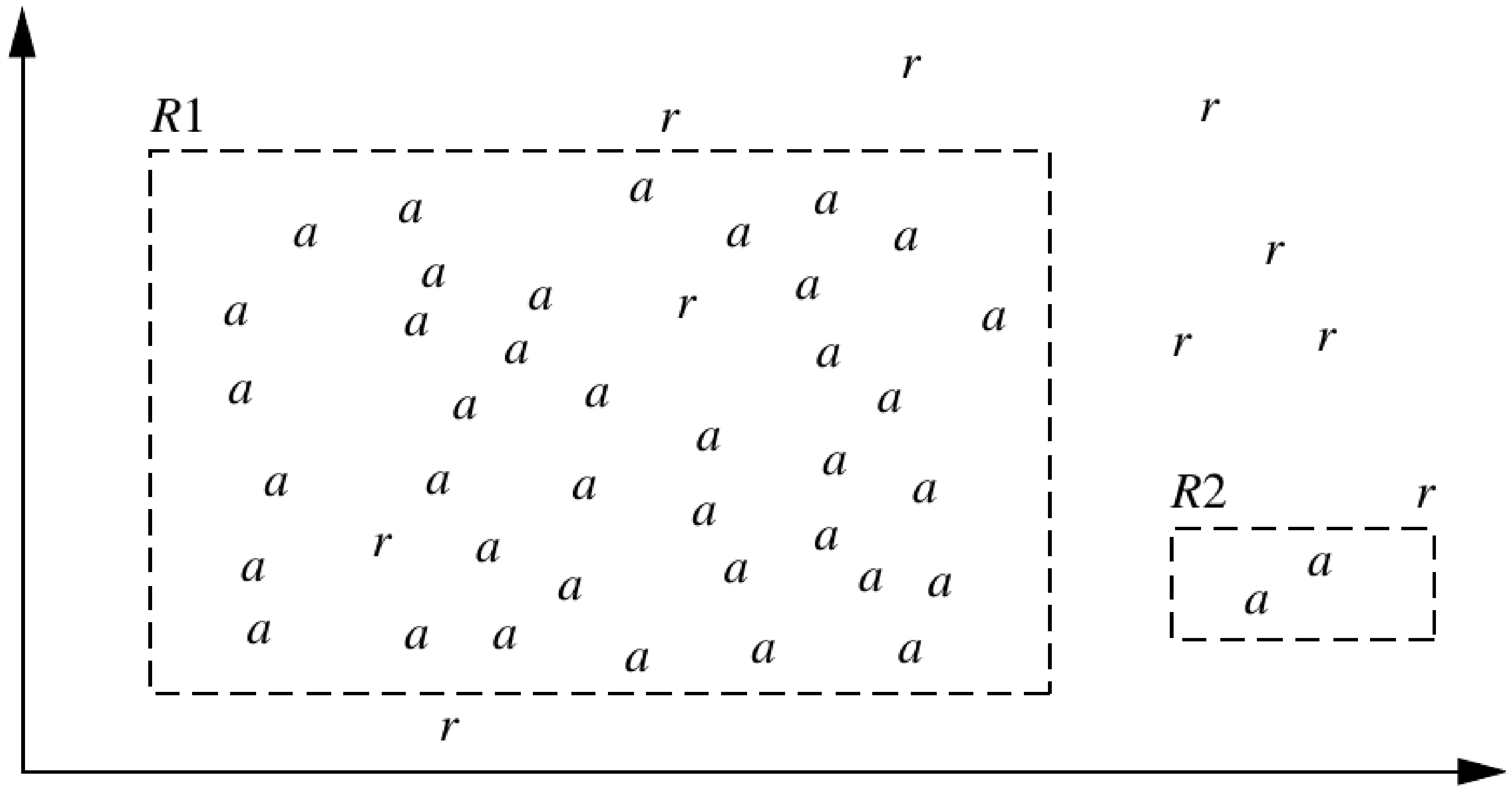


How to Learn-One-Rule?

- Rules are grown in a general-to-specific manner
- Start with the *most general rule* possible: condition = empty
  - E.g.: IF THEN loan_decision=accept
- Then consider each possible attribute test that may be added to the rule
  - can be derived from the param. Att_vals
  - e.g.: for (att, val), consider attribute tests:
    att=val, att≤val, att>val
  - Finding an optimal rule set becomes computationally explosive
- *Adding new attributes* by adopting a greedy depth-first strategy
  - Picks the one that most improves the rule quality
  - e.g.: IF income=high THEN loan_decision=accept
    IF income=high AND credit_rating=excellent
    THEN load_decision=accept

Rule Quality Measure

- Rule-Quality measures: consider both accuracy and coverage
  - accuracy: problematic (only accuracy)



Rule Quality Measure

- Rule-Quality measures: consider both accuracy and coverage
  - Learing rules for the class c
  - current rule is R: IF condition Then class=c
  - new rule is R': IF condition' Then class=c
  - condition' = condition And att_i
- **Entropy: information gain**

$$H(D) = -\sum_{i=1}^{m} p_i \log p_i$$

where D is the set of tuples covered by condition', p_i is the probability of class C_i in D

- The lower the entropy, the better condition' is

Rule Quality Measure

- Rule-Quality measures: consider both accuracy and coverage
  - Foil-gain (in FOIL & RIPPER): assesses info_gain by extending condition

$$FOIL\_Gain = pos' \times \left(\log_2 \frac{pos'}{pos'+neg'} - \log_2 \frac{pos}{pos+neg}\right)$$

    - favors rules that have high accuracy and cover many positive tuples
- likelihood ratio statistic

$$likelihood\_ratio = 2\sum_{i=1}^{m} f_i \log\left(\frac{f_i}{e_i}\right)$$

where m is # of classes, f_i is the observed freq. of class i among the tuples, e_i is the expected freq.

  - The higher the likelihood ratio, the performance of the rule is not due to chance.

Rule Pruning

- Rule pruning based on an independent set of test tuples

$$FOIL\_Prune(R) = \frac{pos - neg}{pos + neg}$$

pos/neg are # of positive/negative tuples covered by R.

If *FOIL_Prune* is higher for the pruned version of R, prune R

Chapter 8. Classification: Basic Concepts

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

Model Evaluation and Selection

- Evaluation metrics: How can we measure accuracy? Other metrics to consider?
- Use **validation test set** of class-labeled tuples instead of training set when assessing accuracy
- Methods for estimating a classifier's accuracy:
  - Holdout method, random subsampling
  - Cross-validation
  - Bootstrap
- Comparing classifiers:
  - Confidence intervals
  - Cost-benefit analysis and ROC Curves

Classifier Evaluation Metrics

- Terminology
  - **Poisitive tuples(P)**: tuples of the main class of interest
  - **Negative tuples(N)**: all other tuples
  - **True positives(TP)**: the positive tuples that were correctly labeled by the classifier
  - **True negative(TN)**: the negative tuples that were correctly labeled by the classifier
  - **False positive(FP)**: the negative tuples that were misclassfied as positive
  - **False negative(FN)**: the positive tuples that were misclassified as negative

Classifier Evaluation Metrics

Measure	Formula
accuracy, recognition rate	**(TP+TN)/(P+N)**
error rate, misclassification rate	**(FP+FN)/(P+N)**
sensitivity, true positive rate, **recall**	**TP /P**
specificity, true negative rate	**TN / N**
precision	**TP /(TP+FP)**
F, F_1, F-score harmonic mean of precision and recall	**2×precision×recall /(precision+recall)**
F_β, where β is a non-negative number	**$(1+\beta^2)$×precision×recall /(β^2×precision+recall)**

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

Classifier Evaluation Metrics: Accuracy, Error Rate

Actual Class\Predicted class	cancer = yes	cancer = no	Total	Recognition(%)
cancer = yes	**90**	**210**	300	30.00 (*sensitivity*
cancer = no	**140**	**9560**	9700	98.56 (*specificity*)
Total	230	9770	10000	96.40 (*accuracy*)

Precision = 90/230 = 39.13%

Recall = 90/300 = 30.00%

Classifier Evaluation Metrics: Sensitivity and Specificity

- **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

accuracy=sensitivity × P /(P+N) + specificity × N /(P+N)

Classifier Evaluation Metrics: Precision and Recall, and F-measures

- **Precision**: exactness – what % of tuples that the classifier labeled as positive are actually positive

  precision=TP/(TP+FP)

- **Recall:** completeness – what % of positive tuples did the classifier label as positive?

  recall=TP/(TP+FN)

- Perfect score is 1.0
- Inverse relationship between precision & recall
- **F measure (*F_1* or *F*-score)**: harmonic mean of precision and recall

  2×precision×recall /(precision+recall)

- **F_β:** weighted measure of precision and recall
  - assigns **β** times as much weight to recall as to precision

  (1+β^2)×precision×recall /(β^2×precision+recall)

Classifier Evaluation Metrics: other measures

- **Speed**: the compuational costs involved in generating and using the classifier
- **Robustness**: the ability to make prediction given noisy data
- **Scalability**: the ability to construct the classifier efficiently given large amounts of data
- **Interpretability**: the level of understanding and insight that is provided by the classifier

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 subsampling: 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*
    - i.e., 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 $\mathbf{(1 - 1/d)^d \approx e^{-1} = 0.368}$)
  - Repeat the sampling procedure *k* times, overall accuracy of the model:

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

Estimating Confidence Intervals: Classifier Models M_1 vs. M_2

- Suppose we have 2 classifiers, M_1 and M_2, which one is better?
- Use 10-fold cross-validation to obtain and
- These mean error rates are just *estimate* $\overline{err}(M_1)$ on th $\overline{err}(M_2)$ population of *future* data cases
- What if the difference between the 2 error rates is just attributed to *chance*?
  - Use a **test of statistical significance**
  - Obtain **confidence limits** for our error estimates

Estimating Confidence Intervals: Null Hypothesis

- Perform 10-fold cross-validation
- Assume samples follow a **t distribution** with k–1 **degrees of freedom** (here, *k=10*)
- Use **t-test** (or **Student's t-test**)
- **Null Hypothesis**: M_1 & M_2 are the same
- If we can **reject** null hypothesis, then
  - we conclude that the difference between M_1 & M_2 is **statistically significant**
  - Chose model with lower error rate

Estimating Confidence Intervals: t-test

- If only 1 test set available: **pairwise comparison**
  - For ith round of 10-fold cross-validation, the same cross partitioning is used to obtain $err(M_1)_i$ and $err(M_2)_i$
  - Average over 10 rounds to get $\overline{err}(M_1)$ and $\overline{err}(M_2)$
  - **t-test** computes **t-statistic** with ***k-1* degrees of freedom:** $$t = \frac{\overline{err}(M_1) - \overline{err}(M_2)}{\sqrt{var(M_1 - M_2)/k}}$$ where

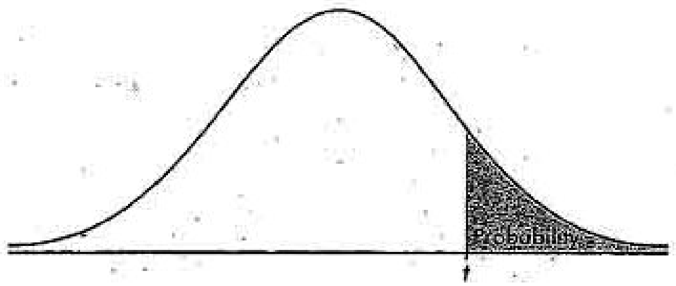
$$var(M_1 - M_2) = \frac{1}{k}\sum_{i=1}^{k}\left[err(M_1)_i - err(M_2)_i - (\overline{err}(M_1) - \overline{err}(M_2))\right]^2$$

- If two test sets available: use **non-paired t-test**

where $$var(M_1 - M_2) = \sqrt{\frac{var(M_1)}{k_1} + \frac{var(M_2)}{k_2}},$$

where k_1 & k_2 are # of cross-validation samples used for M_1 & M_2, resp.

Estimating Confidence Intervals: Table for t-distribution



- Symmetric
- **Significance level**, e.g., *sig = 0.05* or *5%* means M_1 & M_2 are *significantly different* for 95% of population
- **Confidence limit**, *z = sig/2*

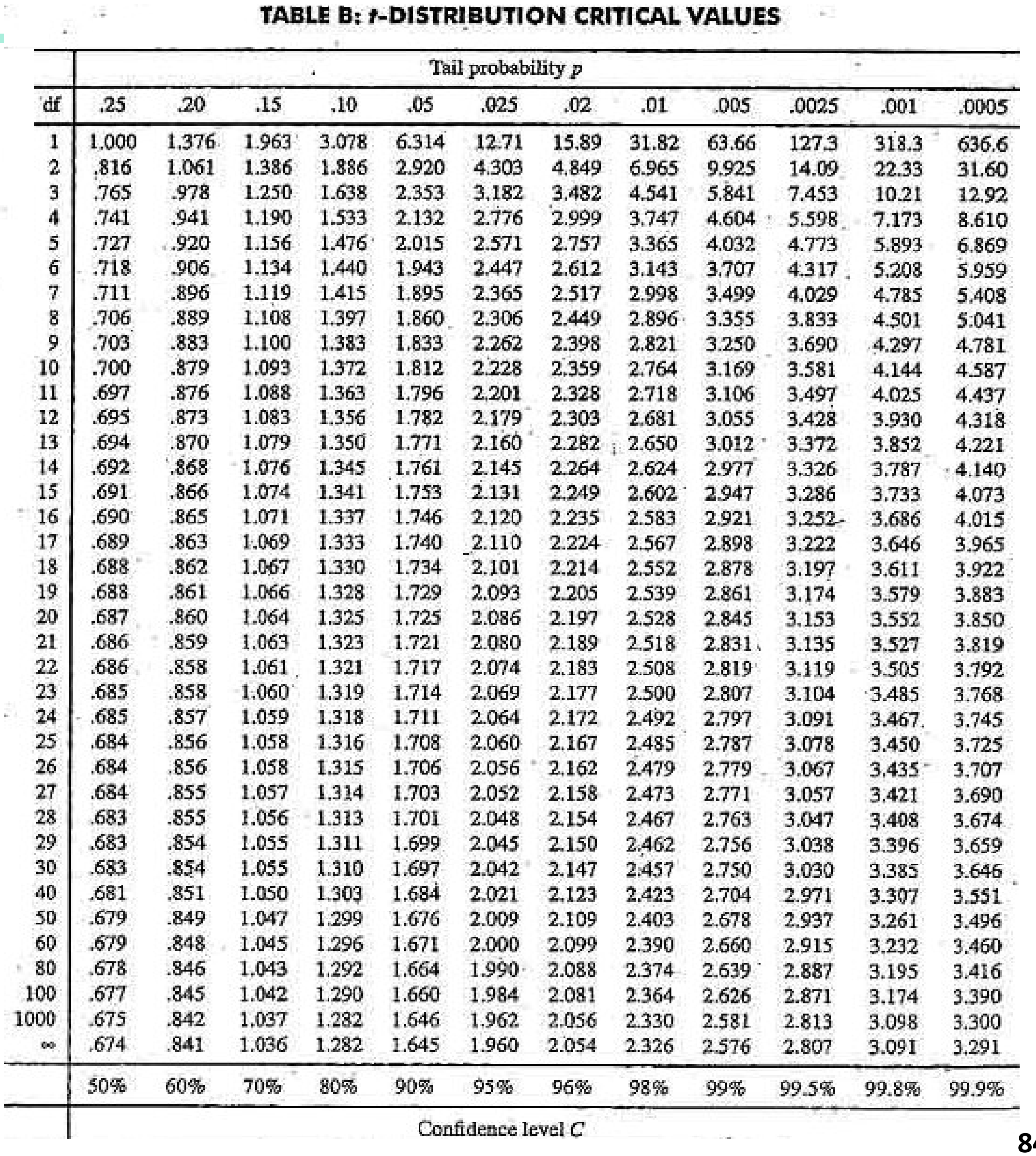


TABLE B: *t*-DISTRIBUTION CRITICAL VALUES

	Tail probability *p*											
df	.25	.20	.15	.10	.05	.025	.02	.01	.005	.0025	.001	.0005
1	1.000	1.376	1.963	3.078	6.314	12.71	15.89	31.82	63.66	127.3	318.3	636.6
2	.816	1.061	1.386	1.886	2.920	4.303	4.849	6.965	9.925	14.09	22.33	31.60
3	.765	.978	1.250	1.638	2.353	3.182	3.482	4.541	5.841	7.453	10.21	12.92
4	.741	.941	1.190	1.533	2.132	2.776	2.999	3.747	4.604	5.598	7.173	8.610
5	.727	.920	1.156	1.476	2.015	2.571	2.757	3.365	4.032	4.773	5.893	6.869
6	.718	.906	1.134	1.440	1.943	2.447	2.612	3.143	3.707	4.317	5.208	5.959
7	.711	.896	1.119	1.415	1.895	2.365	2.517	2.998	3.499	4.029	4.785	5.408
8	.706	.889	1.108	1.397	1.860	2.306	2.449	2.896	3.355	3.833	4.501	5.041
9	.703	.883	1.100	1.383	1.833	2.262	2.398	2.821	3.250	3.690	4.297	4.781
10	.700	.879	1.093	1.372	1.812	2.228	2.359	2.764	3.169	3.581	4.144	4.587
11	.697	.876	1.088	1.363	1.796	2.201	2.328	2.718	3.106	3.497	4.025	4.437
12	.695	.873	1.083	1.356	1.782	2.179	2.303	2.681	3.055	3.428	3.930	4.318
13	.694	.870	1.079	1.350	1.771	2.160	2.282	2.650	3.012	3.372	3.852	4.221
14	.692	.868	1.076	1.345	1.761	2.145	2.264	2.624	2.977	3.326	3.787	4.140
15	.691	.866	1.074	1.341	1.753	2.131	2.249	2.602	2.947	3.286	3.733	4.073
16	.690	.865	1.071	1.337	1.746	2.120	2.235	2.583	2.921	3.252	3.686	4.015
17	.689	.863	1.069	1.333	1.740	2.110	2.224	2.567	2.898	3.222	3.646	3.965
18	.688	.862	1.067	1.330	1.734	2.101	2.214	2.552	2.878	3.197	3.611	3.922
19	.688	.861	1.066	1.328	1.729	2.093	2.205	2.539	2.861	3.174	3.579	3.883
20	.687	.860	1.064	1.325	1.725	2.086	2.197	2.528	2.845	3.153	3.552	3.850
21	.686	.859	1.063	1.323	1.721	2.080	2.189	2.518	2.831	3.135	3.527	3.819
22	.686	.858	1.061	1.321	1.717	2.074	2.183	2.508	2.819	3.119	3.505	3.792
23	.685	.858	1.060	1.319	1.714	2.069	2.177	2.500	2.807	3.104	3.485	3.768
24	.685	.857	1.059	1.318	1.711	2.064	2.172	2.492	2.797	3.091	3.467	3.745
25	.684	.856	1.058	1.316	1.708	2.060	2.167	2.485	2.787	3.078	3.450	3.725
26	.684	.856	1.058	1.315	1.706	2.056	2.162	2.479	2.779	3.067	3.435	3.707
27	.684	.855	1.057	1.314	1.703	2.052	2.158	2.473	2.771	3.057	3.421	3.690
28	.683	.855	1.056	1.313	1.701	2.048	2.154	2.467	2.763	3.047	3.408	3.674
29	.683	.854	1.055	1.311	1.699	2.045	2.150	2.462	2.756	3.038	3.396	3.659
30	.683	.854	1.055	1.310	1.697	2.042	2.147	2.457	2.750	3.030	3.385	3.646
40	.681	.851	1.050	1.303	1.684	2.021	2.123	2.423	2.704	2.971	3.307	3.551
50	.679	.849	1.047	1.299	1.676	2.009	2.109	2.403	2.678	2.937	3.261	3.496
60	.679	.848	1.045	1.296	1.671	2.000	2.099	2.390	2.660	2.915	3.232	3.460
80	.678	.846	1.043	1.292	1.664	1.990	2.088	2.374	2.639	2.887	3.195	3.416
100	.677	.845	1.042	1.290	1.660	1.984	2.081	2.364	2.626	2.871	3.174	3.390
1000	.675	.842	1.037	1.282	1.646	1.962	2.056	2.330	2.581	2.813	3.098	3.300
∞	.674	.841	1.036	1.282	1.645	1.960	2.054	2.326	2.576	2.807	3.091	3.291
	50%	60%	70%	80%	90%	95%	96%	98%	99%	99.5%	99.8%	99.9%
	Confidence level *C*											

Estimating Confidence Intervals: Statistical Significance

- Are M_1 & M_2 **significantly different**?
  - Compute t. Select *significance level* (e.g. *sig = 5%)*
  - Consult table for t-distribution: Find t *value* corresponding to k-1 *degrees of freedom* (here, 9)
  - t-distribution is symmetric: typically upper % points of distribution shown → look up value for **confidence limit** *z=sig/2* (here, 0.025)
  - **If t > z or t < -z**, then t value lies in rejection region:
    - **Reject null hypothesis** that mean error rates of M_1 & M_2 are same
    - Conclude: statistically significant difference between M_1 & M_2
  - **Otherwise**, conclude that any difference is **chance**

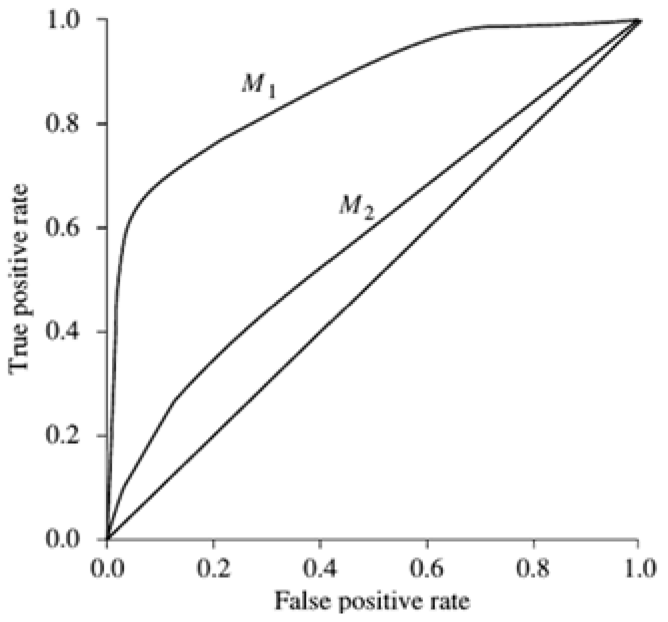
Model Selection: ROC Curves

- **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 (TPR)** and the **false positive rate (FPR)**
- **TPR** is the proportion of positive tuples that are correctly labeled by the model
- **FPR** is the proportion of negative tuples that are mislabeled as positive.
- **TP**, **FP**, **P**, and **N** are the number of true positive, false positive, positive, and negative tuples

  TPR = TP/P=sensitivity   FPR = FP/N = 1 – specificity
- The area under the ROC curve is a measure of the accuracy of the model

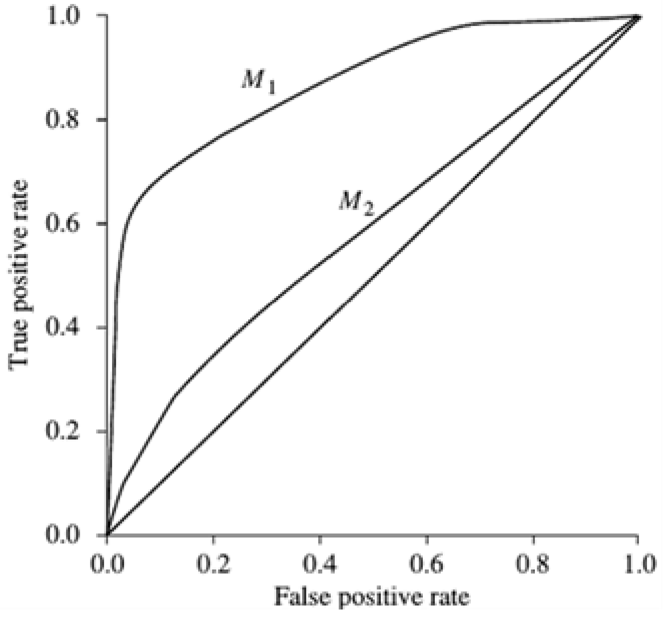
Model Selection: ROC Curves

- Any increase in TPR occurs at the cost of an increase in FPR
- The area under the ROC curve is a measure of the accuracy of the model
- To plot an ROC curve for a given classification model, M, **the model must be able to return a probability of the predicted class for each test tuple**.



Model Selection: ROC Curves

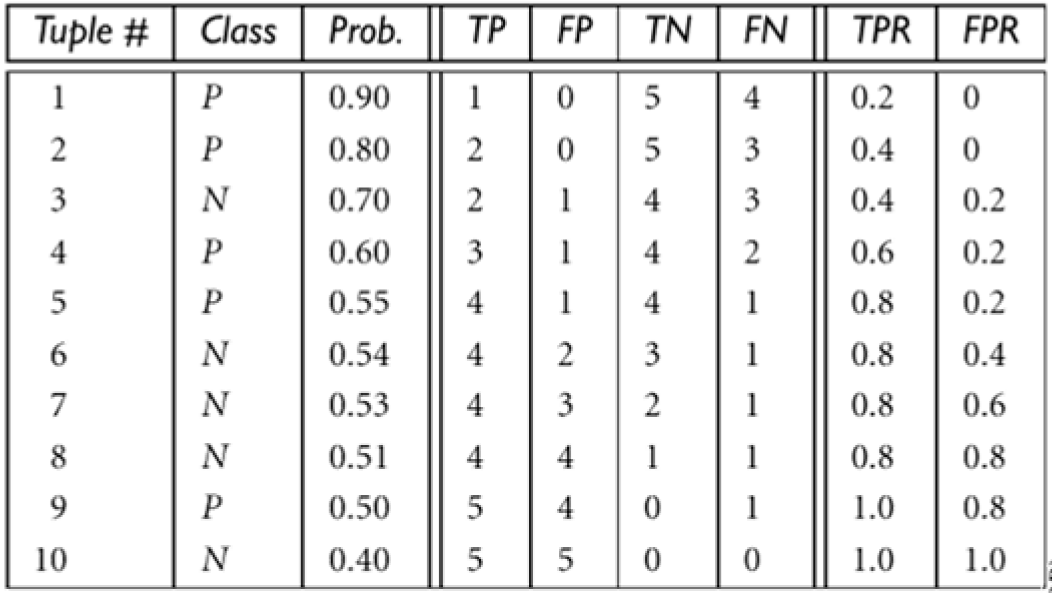
- For a tuple X, the probablistic classifier returns f(X)→[0,1]
- For a binary problem, a threshold t is selected so that tuples where f(X) ≥ t are considered positive and all the other tuples are considered negative
- the number of true positives and the number of false positives are both functions of t, TP(t) and FP(t)



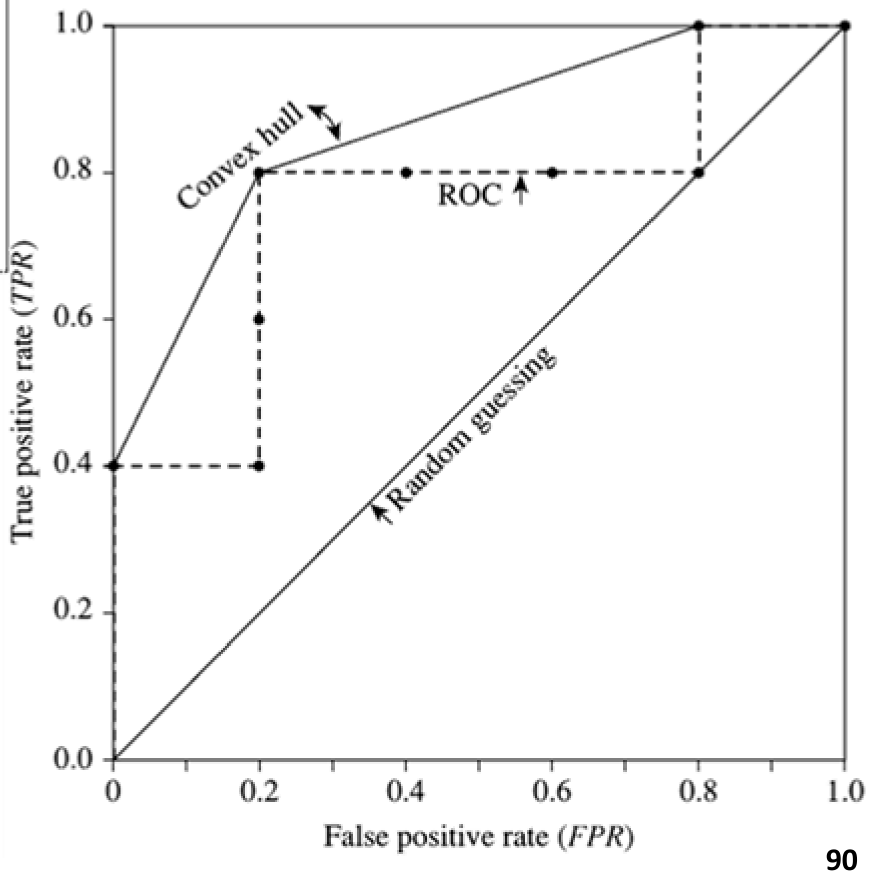
Model Selection: ROC Curves

- Tuples are ranked and sorted so that the tuple with the most positive prob. appears at the top of the list, and the tuple with the least positive prob. lands at the bottom of the list
- The vert. axis of an ROC curve represents **TPR**, the horiz. axis represents **FPR**
- Starting at the bottom left corner: **TPR = FPR = 0**
- if the tuple is true positive, then **TP** and **TPR** increase. On the graph, move up and plot a point.
- If it is false positive, then both **FP** and **FPR** increase. On the graph, move right and plot a point.
- This process is repeated for each of the test tuples in ranked order, each time moving up on the graph for a true positive or toward the right for a false positive

Model Selection: ROC Curves

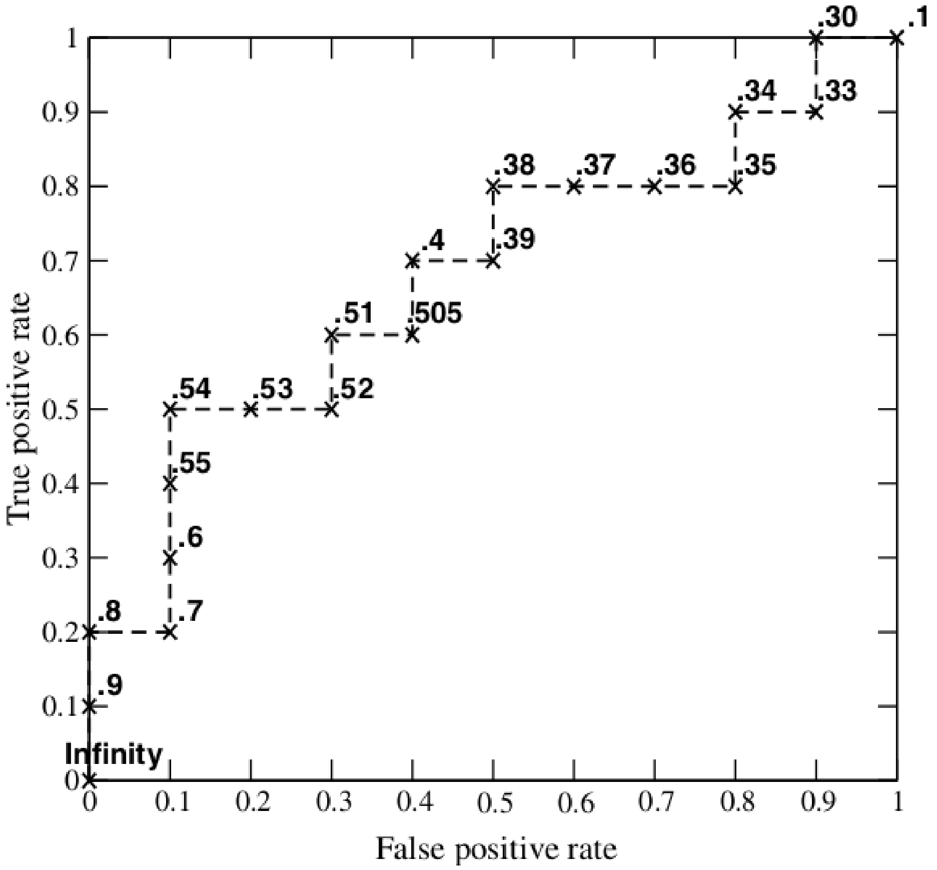


Tuple #	Class	Prob.	TP	FP	TN	FN	TPR	FPR
1	*P*	0.90	1	0	5	4	0.2	0
2	*P*	0.80	2	0	5	3	0.4	0
3	*N*	0.70	2	1	4	3	0.4	0.2
4	*P*	0.60	3	1	4	2	0.6	0.2
5	*P*	0.55	4	1	4	1	0.8	0.2
6	*N*	0.54	4	2	3	1	0.8	0.4
7	*N*	0.53	4	3	2	1	0.8	0.6
8	*N*	0.51	4	4	1	1	0.8	0.8
9	*P*	0.50	5	4	0	1	1.0	0.8
10	*N*	0.40	5	5	0	0	1.0	1.0



Model Selection: ROC Curves

Inst#	Class	Score	Inst#	Class	Score
1	p	.9	11	p	.4
2	p	.8	12	n	.39
3	n	.7	13	p	.38
4	p	.6	14	n	.37
5	p	.55	15	n	.36
6	p	.54	16	n	.35
7	n	.53	17	p	.34
8	n	.52	18	n	.33
9	p	.51	19	p	.30
10	n	.505	20	n	.1



Issues Affecting Model Selection

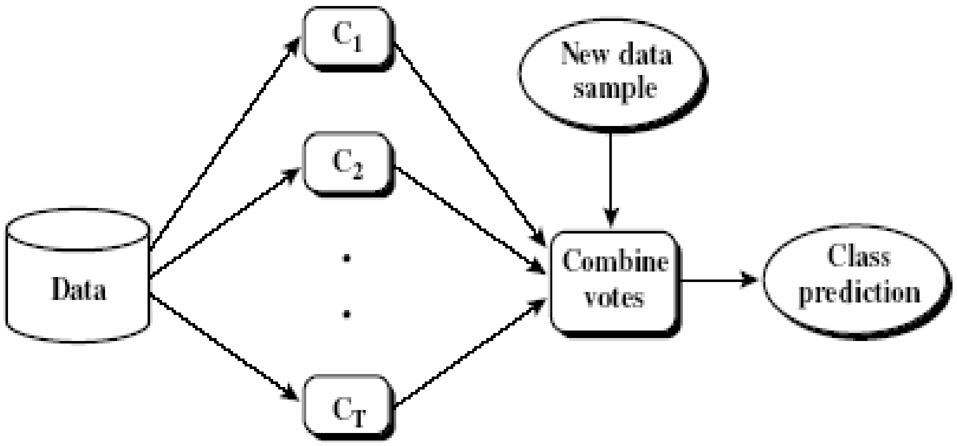
- **Accuracy**
  - classifier accuracy: predicting class label
- **Speed**
  - time to construct the model (training time)
  - time to use the model (classification/prediction time)
- **Robustness**: handling noise and missing values
- **Scalability**: efficiency in disk-resident databases
- **Interpretability**
  - understanding and insight provided by the model
- Other measures, e.g., goodness of rules, such as decision tree size or compactness of classification rules

Chapter 8. Classification: Basic Concepts

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

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*

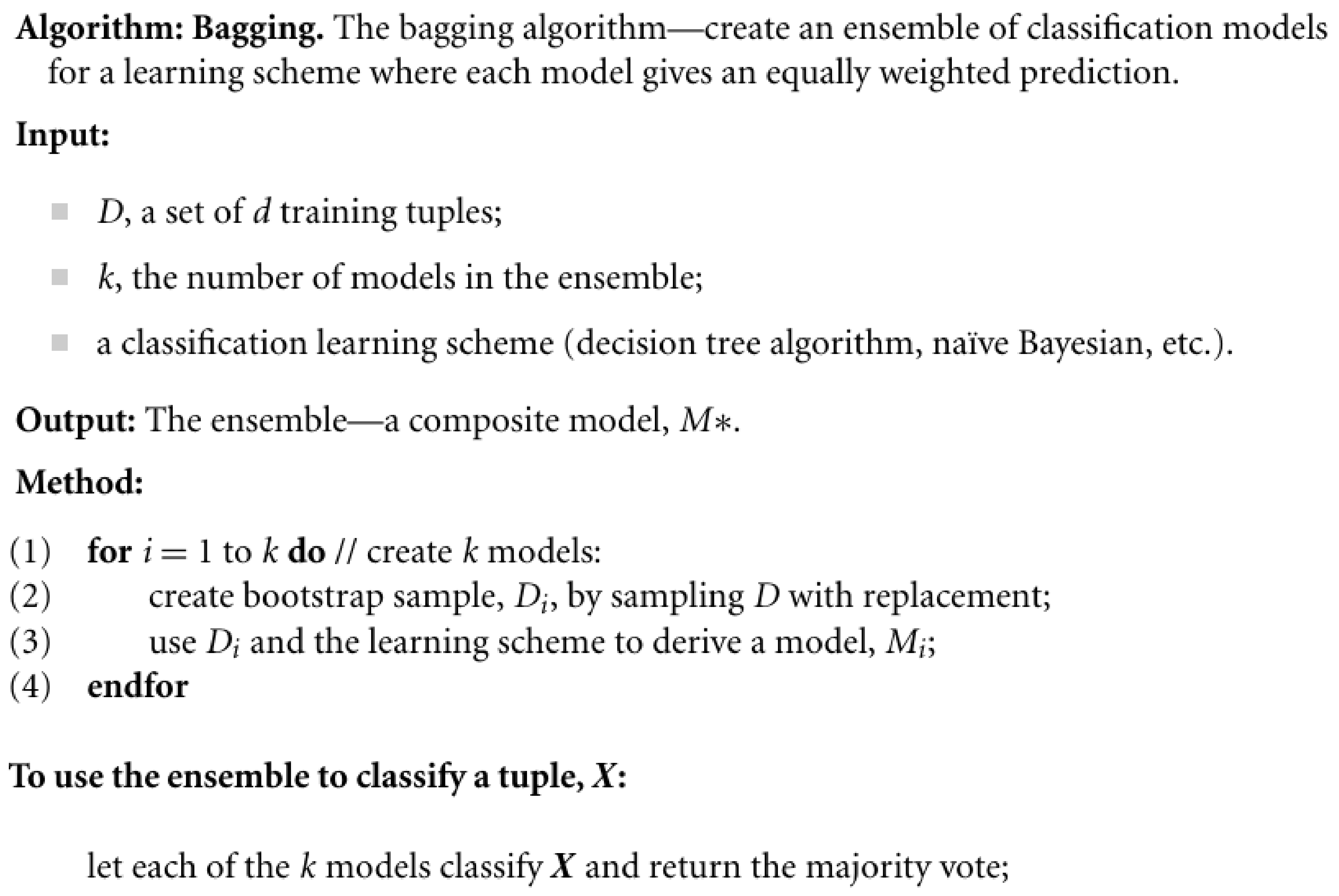


- Popular ensemble methods
  - **Bagging**: averaging the prediction over a collection of classifiers
  - **Boosting**: weighted vote with a collection of classifiers
  - Ensemble: combining a set of heterogeneous classifiers

Bagging: Bootstrap 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 $\mathbf{M_i}$ is learned for each training set D_i
- Classification: classify an unknown sample **X**
  - Each classifier $\mathbf{M_i}$ returns its class prediction
  - The bagged classifier **M*** counts the votes and assigns the class with the most votes to **X**

Bagging: Bootstrap Aggregation



Algorithm: Bagging. The bagging algorithm—create an ensemble of classification models for a learning scheme where each model gives an equally weighted prediction.

Input:

- D, a set of d training tuples;
- k, the number of models in the ensemble;
- a classification learning scheme (decision tree algorithm, naïve Bayesian, etc.).

Output: The ensemble—a composite model, $M*$.

Method:

```
(1)  for $i = 1$ to $k$ do // create $k$ models:
(2)      create bootstrap sample, $D_i$, by sampling $D$ with replacement;
(3)      use $D_i$ and the learning scheme to derive a model, $M_i$;
(4)  endfor
```

To use the ensemble to classify a tuple, *X*:

let each of the k models classify $\boldsymbol{X}$ and return the majority vote;

Bagging: Bootstrap Aggregation

- Prediction: can be applied to the prediction of continuous values by taking the average value of each prediction for a given test tuple
- Accuracy
  - Often significantly better than a single classifier derived from D
  - For noise data: not considerably worse, more robust
  - Proved improved accuracy in prediction

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

- Boosting algorithm can be extended for numeric prediction
- Comparing with bagging:
  - Boosting tends to have greater accuracy,
  - it risks overfitting the model to misclassified data

Adaboost (Adaptive Boosting)

- 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, w_j
  - A classification model M_i is derived from D_i

Adaboost (Adaptive Boosting)

- Generate k classifiers in k rounds. At round i,
  - ...
  - Its error rate is calculated using D_i as a test set

$$error(M_i) = \sum_{j=1}^{d} w_j \times err(X_j)$$

where $err(X_j)=1$ if X_j is misclassified, o.w., $err(X_j)=0$

  - If tuple j is misclassified

$$w'_j \leftarrow w_j \times \frac{error(M_j)}{1 - error(M_j)}$$

  - After the weights of all the correctly classified tuples are updated, the weights for all tuples are normalized

$$w''_j \leftarrow w'_j \times \frac{\sum_{j=1}^{d} w_j}{\sum_{j=1}^{d} w'_j}$$

Algorithm: Adaboost. A boosting algorithm—create an ensemble of classifiers. Each one gives a weighted vote.

Input:

- D, a set of d class-labeled training tuples;
- k, the number of rounds (one classifier is generated per round);
- a classification learning scheme.

Output: A composite model.

Method:

```
(1)  initialize the weight of each tuple in $D$ to $1/d$;
(2)  for $i = 1$ to $k$ do // for each round:
(3)      sample $D$ with replacement according to the tuple weights to obtain $D_i$;
(4)      use training set $D_i$ to derive a model, $M_i$;
(5)      compute $error(M_i)$, the error rate of $M_i$
(6)      if $error(M_i) > 0.5$ then
(7)          reinitialize the weights to $1/d$
(8)          go back to step 3 and try again;
(9)      endif
(10)     for each tuple in $D_i$ that was correctly classified do
(11)         multiply the weight of the tuple by $error(M_i)/(1 - error(M_i))$; // update weights
(12)     normalize the weight of each tuple;
(13) endfor
```

Adaboost

- To use the ensemble to classify tuple, X:

```
(1)  initialize weight of each class to 0;
(2)  **for** $i = 1$ to $k$ **do** // for each classifier:
(3)      $w_i = log\frac{1-error(M_i)}{error(M_i)}$; // weight of the classifier's vote
(4)      $c = M_i(\boldsymbol{X})$; // get class prediction for $\boldsymbol{X}$ from $M_i$
(5)      add $w_i$ to weight for class $c$
(6)  **endfor**
(7)  return the class with the largest weight;
```

Binary version of Adaboost

- Given a set of N class-labeled tuples, ($\mathbf{X_1}$, y_1), …, ($\mathbf{X_d}$, y_d), in which yi∈{+1,-1}
- Initially, all the weights of tuples are set the same (1/N)

$$D_1 = (w_{11}, w_{12} \cdots w_{1i} \cdots, w_{1N}),\ w_{1i} = \frac{1}{N},\ i = 1,2,\cdots,N$$

- for m=1 to k
  - Train a basic classifier using training data set with weight distribution, D_m: $G_m(x): \chi \to \{-1,+1\}$
  - Caculate the error rate of $G_m(x)$:

$$e_m = P(G_m(x_i) \neq y_i) = \sum_{i=1}^{N} w_{mi} I(G_m(x_i) \neq y_i)$$

  - Calculate the coefficient of $G_m(x)$:

$$\alpha_m = \frac{1}{2} \log \frac{1-e_m}{e_m}$$

Binary version of Adaboost

- for m=1 to k
  - …
  - Update weight distribution of training data:

$$D_{m+1} = \left(w_{m+1,1}, w_{m+1,2} \cdots w_{m+1,i} \cdots, w_{m+1,N}\right),$$

$$w_{m+1,i} = \frac{w_{mi}}{Z_m} \exp\left(-\alpha_m y_i G_m(x_i)\right), \quad i = 1,2,\cdots,N$$

where $Z_m = \sum_{i=1}^{N} w_{mi} \exp\left(-\alpha_m y_i G_m(x_i)\right)$

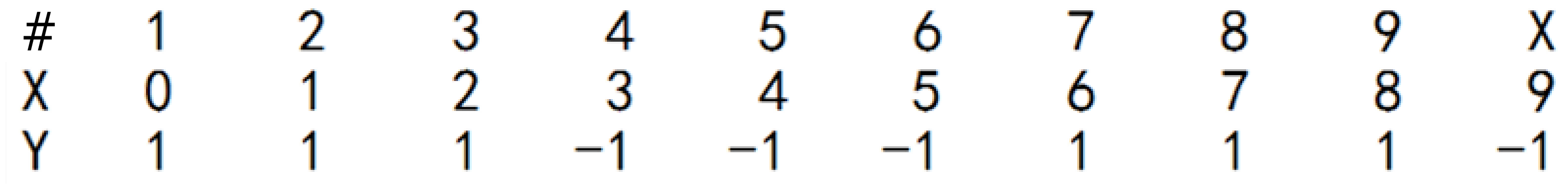
- Construct the linear combination of $G_m(x)$

$$f(x) = \sum_{m=1}^{M} \alpha_m G_m(x)$$

- The final classifier is: $G(x) = sign\left(f(x)\right) = sign\left(\sum_{m=1}^{M} \alpha_m G_m(x)\right)$

Example

- Given the following data, to train a classifier using AdaBoost algorithm.



#	1	2	3	4	5	6	7	8	9	X
X	0	1	2	3	4	5	6	7	8	9
Y	1	1	1	−1	−1	−1	1	1	1	−1

Example

- Initialize the weight distribution of training data

$$D_1 = (w_{11}, w_{12} \cdots w_{1i} \cdots, w_{1N}),\ w_{1i} = \frac{1}{N},\ i = 1,2,\cdots, N$$

i.e., $W_{1i} = 0.1$

- For m=1

$$G_1(x) = \begin{cases} 1, & x < 2.5 \\ -1, & x > 2.5 \end{cases}$$

$e_1 = P(G_1(x_i) \neq y_i) = 0.3$,

$$\alpha_1 = \frac{1}{2}\log\frac{1-e_1}{e_1} = 0.4236$$

$f_1(x) = 0.4236 G_1(x)$

D_2=(0.0715, 0.0715, 0.0715, 0.0715, 0.0715, 0.0715, 0.1666, 0.1666, 0.1666, 0.0715)

Example

- For m=2

$$G_2(x) = \begin{cases} 1, & x < 8.5 \\ -1, & x > 8.5 \end{cases}$$

$e_2 = P(G_2(x_i) \neq y_i) = 0.2143$

$$\alpha_2 = \frac{1}{2}\log\frac{1-e_2}{e_2} = 0.6496$$

D_3=(0.0455, 0.0455, 0.0455, 0.1667, 0.1667,

0.01667, 0.1060, 0.1060, 0.1060, 0.0455)

$f_2(x)=0.4236G_1(x) + 0.6496G_2(x)$

Example

- For m=3

$$G_3(x) = \begin{cases} 1, & x < 5.5 \\ -1, & x > 5.5 \end{cases}$$

$e_3 = P(G_3(x_i) \neq y_i) = 0.1820$

$$\alpha_3 = \frac{1}{2} \log \frac{1 - e_3}{e_3} = 0.7514$$

D_4=(0.125, 0.125, 0.125, 0.102, 0.102,
0.102, 0.065, 0.065, 0.065, 0.125)

$f_3(x) = 0.4236G_1(x) + 0.6496G_2(x) + 0.7514G_3(x)$

- The final classifier is: **$G(x) = \text{sign}(f_3(x))$**

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

Random Forest (Breiman 2001)

- 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 for imbalance data in 2-class classification:
  - **Oversampling**: re-sampling of data from positive class
  - **Under-sampling**: randomly eliminate tuples from negative class
  - **Threshold-moving**: moves 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 introduced above
- Still difficult for class imbalance problem on multiclass tasks

Chapter 8. Classification: Basic Concepts

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

Summary (I)

- Classification is a form of data analysis that extracts models describing important data classes.
- Effective and scalable methods have been developed for decision tree induction, Naive Bayesian classification, rule-based classification, and many other classification methods.
- Evaluation metrics include: accuracy, sensitivity, specificity, precision, recall, *F* measure, and F_{β} measure.
- Stratified k-fold cross-validation is recommended for accuracy estimation. Bagging and boosting can be used to increase overall accuracy by learning and combining a series of individual models.

Summary (II)

- Significance tests and ROC curves are useful for model selection.
- There have been numerous comparisons of the different classification methods; the matter remains a research topic
- No single method has been found to be superior over all others for all data sets
- Issues such as accuracy, training time, robustness, scalability, and interpretability must be considered and can involve trade-offs, further complicating the quest for an overall superior method

References (1)

- C. Apte and S. Weiss. **Data mining with decision trees and decision rules**. Future Generation Computer Systems, 13, 1997
- C. M. Bishop, **Neural Networks for Pattern Recognition**. Oxford University Press, 1995
- L. Breiman, J. Friedman, R. Olshen, and C. Stone. **Classification and Regression Trees**. Wadsworth International Group, 1984
- C. J. C. Burges. **A Tutorial on Support Vector Machines for Pattern Recognition**. *Data Mining and Knowledge Discovery*, 2(2): 121-168, 1998
- P. K. Chan and S. J. Stolfo. **Learning arbiter and combiner trees from partitioned data for scaling machine learning**. KDD'95
- H. Cheng, X. Yan, J. Han, and C.-W. Hsu, **Discriminative Frequent Pattern Analysis for Effective Classification**, ICDE'07
- H. Cheng, X. Yan, J. Han, and P. S. Yu, **Direct Discriminative Pattern Mining for Effective Classification**, ICDE'08
- W. Cohen. **Fast effective rule induction**. ICML'95
- G. Cong, K.-L. Tan, A. K. H. Tung, and X. Xu. **Mining top-k covering rule groups for gene expression data**. SIGMOD'05

References (2)

- A. J. Dobson. **An Introduction to Generalized Linear Models**. Chapman & Hall, 1990.
- G. Dong and J. Li. **Efficient mining of emerging patterns: Discovering trends and differences**. KDD'99.
- R. O. Duda, P. E. Hart, and D. G. Stork. **Pattern Classification**, 2ed. John Wiley, 2001
- U. M. Fayyad. **Branching on attribute values in decision tree generation**. AAAI'94.
- Y. Freund and R. E. Schapire. **A decision-theoretic generalization of on-line learning and an application to boosting**. J. Computer and System Sciences, 1997.
- J. Gehrke, R. Ramakrishnan, and V. Ganti. **Rainforest: A framework for fast decision tree construction of large datasets**. VLDB'98.
- J. Gehrke, V. Gant, R. Ramakrishnan, and W.-Y. Loh, **BOAT -- Optimistic Decision Tree Construction**. SIGMOD'99.
- T. Hastie, R. Tibshirani, and J. Friedman. **The Elements of Statistical Learning: Data Mining, Inference, and Prediction.** Springer-Verlag, 2001.
- D. Heckerman, D. Geiger, and D. M. Chickering. **Learning Bayesian networks: The combination of knowledge and statistical data**. Machine Learning, 1995.
- W. Li, J. Han, and J. Pei, **CMAR: Accurate and Efficient Classification Based on Multiple Class-Association Rules**, ICDM'01.

References (3)

- T.-S. Lim, W.-Y. Loh, and Y.-S. Shih. **A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms.** Machine Learning, 2000.
- J. Magidson. **The Chaid approach to segmentation modeling: Chi-squared automatic interaction detection**. In R. P. Bagozzi, editor, Advanced Methods of Marketing Research, Blackwell Business, 1994.
- M. Mehta, R. Agrawal, and J. Rissanen. **SLIQ : A fast scalable classifier for data mining**. EDBT'96.
- T. M. Mitchell. **Machine Learning**. McGraw Hill, 1997.
- S. K. Murthy, **Automatic Construction of Decision Trees from Data: A Multi-Disciplinary Survey**, Data Mining and Knowledge Discovery 2(4): 345-389, 1998
- J. R. Quinlan. **Induction of decision trees**. *Machine Learning*, 1:81-106, 1986.
- J. R. Quinlan and R. M. Cameron-Jones. **FOIL: A midterm report**. ECML'93.
- J. R. Quinlan. **C4.5: Programs for Machine Learning**. Morgan Kaufmann, 1993.
- J. R. Quinlan. **Bagging, boosting, and c4.5**. AAAI'96.

References (4)

- R. Rastogi and K. Shim. **Public: A decision tree classifier that integrates building and pruning**. VLDB'98.
- J. Shafer, R. Agrawal, and M. Mehta. **SPRINT : A scalable parallel classifier for data mining**. VLDB'96.
- J. W. Shavlik and T. G. Dietterich. **Readings in Machine Learning**. Morgan Kaufmann, 1990.
- P. Tan, M. Steinbach, and V. Kumar. **Introduction to Data Mining**. Addison Wesley, 2005.
- S. M. Weiss and C. A. Kulikowski. **Computer Systems that Learn: Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning, and Expert Systems**. Morgan Kaufman, 1991.
- S. M. Weiss and N. Indurkhya. **Predictive Data Mining**. Morgan Kaufmann, 1997.
- I. H. Witten and E. Frank. **Data Mining: Practical Machine Learning Tools and Techniques**, 2ed. Morgan Kaufmann, 2005.
- X. Yin and J. Han. **CPAR: Classification based on predictive association rules**. SDM'03
- H. Yu, J. Yang, and J. Han. **Classifying large data sets using SVM with hierarchical clusters**. KDD'03.

CS412 Midterm Exam Statistics

- Opinion Question Answering:
  - Like the style: 70.83%, dislike: 29.16%
  - Exam is hard: 55.75%, easy: 0.6%, just right: 43.63%
  - Time: plenty:3.03%, enough: 36.96%, not: 60%
- Score distribution: # of students (Total: 180)
  - >=90: 24
  - 80-89: 54
  - 70-79: 46
  - 60-69: 37
  - 50-59: 15
  - 40-49: 2
  - <40: 2
- Final grading are based on overall score accumulation and relative class distributions

Issues: Evaluating Classification Methods

- Accuracy
  - classifier accuracy: predicting class label
  - predictor accuracy: guessing value of predicted attributes
- Speed
  - time to construct the model (training time)
  - time to use the model (classification/prediction time)
- Robustness: handling noise and missing values
- Scalability: efficiency in disk-resident databases
- Interpretability
  - understanding and insight provided by the model
- Other measures, e.g., goodness of rules, such as decision tree size or compactness of classification rules

Predictor Error Measures

- Measure predictor accuracy: measure how far off the predicted value is from the actual known value
- **Loss function**: measures the error betw. y_i and the predicted value y_i'
  - Absolute error: $| y_i - y_i' |$
  - Squared error: $(y_i - y_i')^2$
- Test error (generalization error): the average loss over the test set
  - Mean absolute error: $\frac{\sum_{i=1}^{d} | y_i - y_i' |}{d}$ Mean squared error: $\frac{\sum_{i=1}^{d} (y_i - y_i')^2}{d}$
  - Relative absolute error: $\frac{\sum_{i=1}^{d} | y_i - y_i' |}{\sum_{i=1}^{d} | y_i - \bar{y} |}$ Relative squared error: $\frac{\sum_{i=1}^{d} (y_i - y_i')^2}{\sum_{i=1}^{d} (y_i - \bar{y})^2}$

The mean squared-error exaggerates the presence of outliers

Popularly use (square) root mean-square error, similarly, root relative squared error

Scalable Decision Tree Induction Methods

- SLIQ (EDBT'96 — Mehta et al.)
  - Builds an index for each attribute and only class list and the current attribute list reside in memory
- SPRINT (VLDB'96 — J. Shafer et al.)
  - Constructs an attribute list data structure
- PUBLIC (VLDB'98 — Rastogi & Shim)
  - Integrates tree splitting and tree pruning: stop growing the tree earlier
- RainForest (VLDB'98 — Gehrke, Ramakrishnan & Ganti)
  - Builds an AVC-list (attribute, value, class label)
- BOAT (PODS'99 — Gehrke, Ganti, Ramakrishnan & Loh)
  - Uses bootstrapping to create several small samples

Data Cube-Based Decision-Tree Induction

- Integration of generalization with decision-tree induction (Kamber et al.'97)
- Classification at primitive concept levels
  - E.g., precise temperature, humidity, outlook, etc.
  - Low-level concepts, scattered classes, bushy classification-trees
  - Semantic interpretation problems
- Cube-based multi-level classification
  - Relevance analysis at multi-levels
  - Information-gain analysis with dimension + level