UNIT-III

Classification and Prediction

Classification:

- predicts categorical class labels
- 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

Prediction

models continuous-valued functions, i.e., predicts unknown or missing values

Typical applications

- Credit approval
- Target marketing
- Medical diagnosis
- Fraud detection

Classification: Basic Concepts

Supervised learning (classification)

o Supervision: The training data (observations, measurements, etc.) are accompanied by **labels** indicating the class of the observations a New data is classified based on the training set

Unsupervised learning (clustering)

- The class labels of training data is unknown

o Given a set of measurements, observations, etc. with the aim of establishing the existence of classes or clusters in the data

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

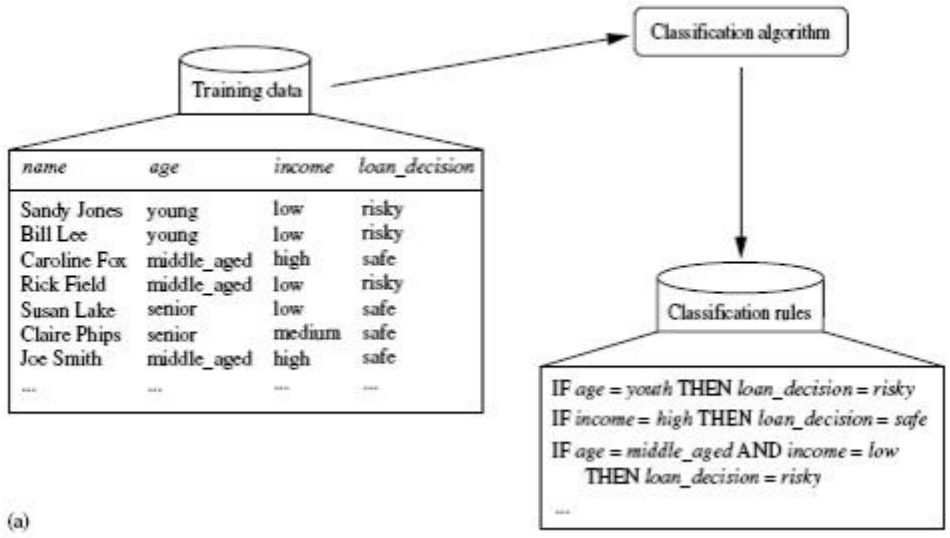
- Each tuple/sample is assumed to belong to a predefined class, as determined by the class label attribute
- The set of tuples used for model construction: training set
- The model is represented as classification rules, decision trees, or mathematical formulae

Model usage: for classifying future or unknown objects

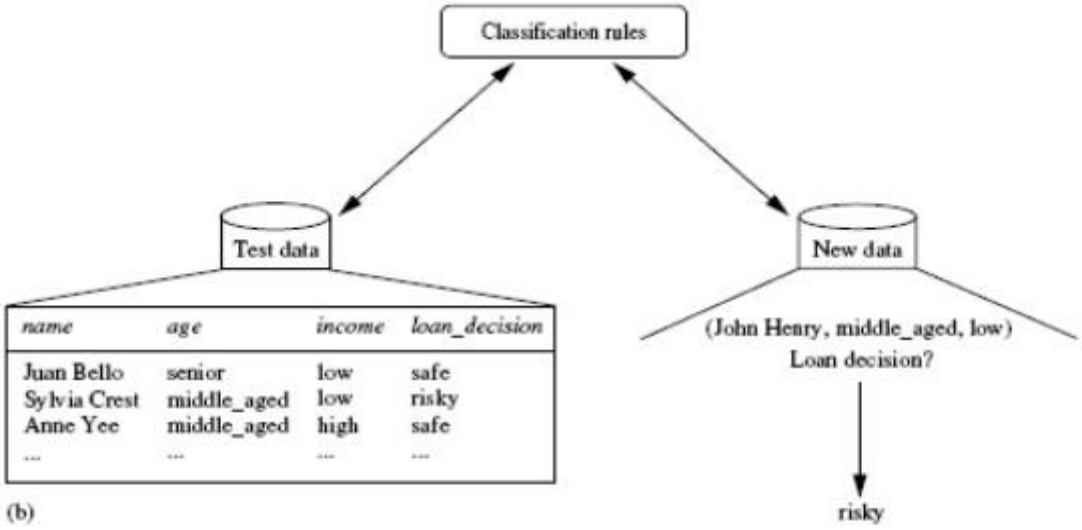
Estimate accuracy of the model

- 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 over-fitting will occur

Process (1): Model Construction



Process (2): Using the Model in Prediction



The data classification process: (a) *Learning*: Training data are analyzed by a classification algorithm. Here, the class label attribute is *loan_decision*, and the learned model or classifier is represented in the form of classification rules. (b) *Classification*: Test data are used to estimate the accuracy of the classification rules. If the accuracy is considered acceptable, the rules can be applied to the classification of new data tuples.

Issues Regarding Classification and Prediction

Data cleaning: This refers to the preprocessing of data in order to remove or reduce *noise* (by applying smoothing techniques, for example) and the treatment of *missing values* (e.g., by replacing a missing value with the most commonly occurring value for that attribute, or with the most probable value based on statistics).

Relevance analysis: Many of the attributes in the data may be *redundant*. Correlation analysis can be used to identify whether any two given attributes are statistically related.

Data transformation and reduction: The data may be transformed by normalization, particularly when neural networks or methods involving distance measurements are used in the learning step. Normalization involves scaling all values for a given attribute so that they fall within a small specified range, such as -1.0 to 1.0, or 0.0 to 1.0. In methods that use distance

Comparing Classification and Prediction Methods

Classification and prediction methods can be compared and evaluated according to the following criteria:

- **Accuracy**
- **Speed**
- **Robustness**
- **Scalability**
- **Interpretability**

Classification by Decision Tree Induction

Decision tree

A flow-chart-like tree structure

Internal node denotes a test on an attribute node (nonleaf node) denotes a test on an attribute

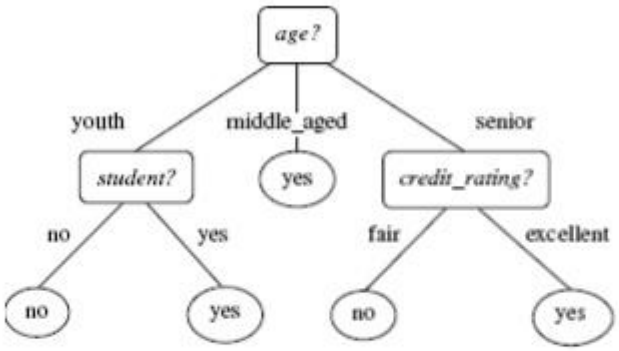
Branch represents an outcome of the test

Leaf nodes represent class labels or class distribution(Terminal node)

The topmost node in a tree is the root node.

Decision tree generation consists of two phases

- Tree construction
- At start, all the training examples are at the root
- Partition examples recursively based on selected attributes Tree pruning
- Identify and remove branches that reflect noise or outliers



A decision tree for the concept *buys_computer*, indicating whether a customer at *AllElectronics* is likely to purchase a computer. Each internal (nonleaf) node represents a test on an attribute. Each leaf node represents a class (either *buys_computer = yes* or *buys_computer = no*).

A typical decision tree is shown in Figure. It represents the concept *buys computer*, that is, it predicts whether a customer at *AllElectronics* is likely to purchase a computer. Internal nodes are denoted by rectangles, and leaf nodes are denoted by ovals. Some decision tree algorithms produce

only *binary* trees (where each internal node branches to exactly two other nodes), whereas others can produce non binary trees.

"How are decision trees used for classification?" Given a tuple, ***X***, for which the associated class label is unknown, the attribute values of the tuple are tested against the decision tree. A path is traced from the root to a leaf node, which holds the class prediction for that tuple. Decision trees can easily be converted to classification rules.

Decision Tree Induction

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:

```
(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 ← 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 Dj be the set of data tuples in D satisfying outcome j; // a partition
(12)     if Dj 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(Dj, attribute_list) to node N;
     endfor
(15) return N;
```

Basic algorithm for inducing a decision tree from training tuples.

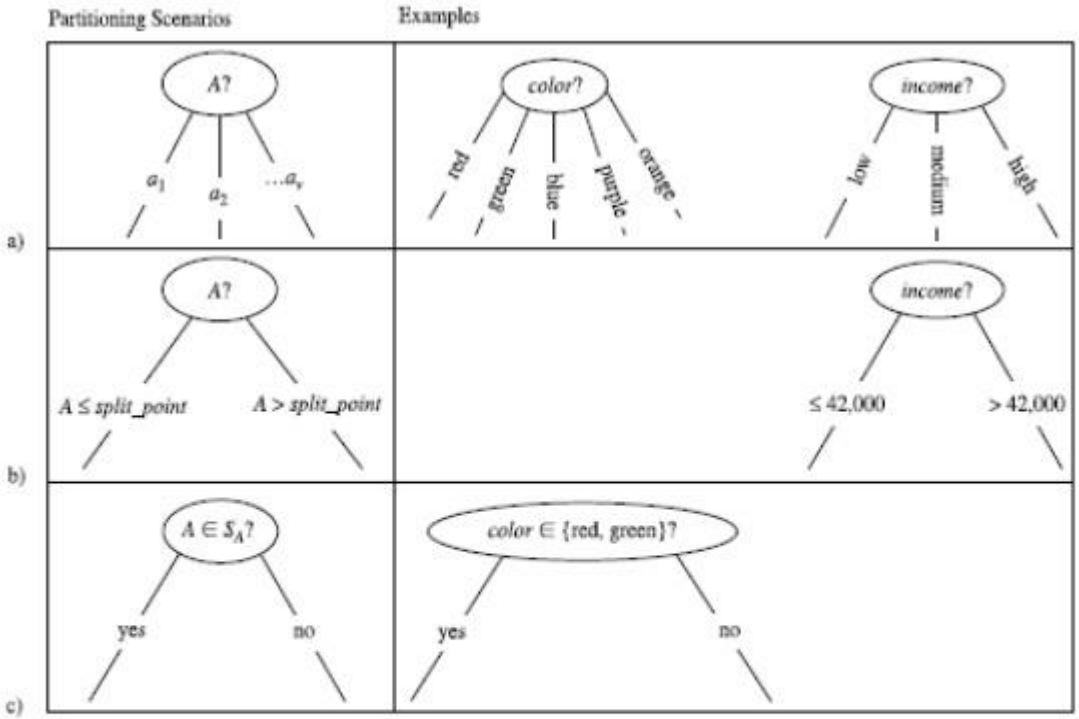
The tree starts as a single node, N, representing the training tuples in D (step 1)

If the tuples in *D* are all of the same class, then node *N* becomes a leaf and is labeled with that class (steps 2 and 3). Note that steps 4 and 5 are terminating conditions. All of the terminating conditions are explained at the end of the algorithm.

Otherwise, the algorithm calls *Attribute selection method* to determine the splitting criterion. The splitting criterion tells us which attribute to test at node *N* by determining the ―best‖ way to separate or partition the tuples in *D* into individual classes(step 6). The splitting criterion also tells us which branches to grow from node *N* with respect to the outcomes of the chosen test. More specifically, the splitting criterion indicates the splitting attribute and may also indicate either a split-point or a splitting subset. The splitting criterion is determined so that, ideally, the resulting partitions at each branch are as ―pure‖ as possible.

A partition is pure if all of the tuples in it belong to the same class. In other words, if we were to split up the tuples in *D* according to the mutually exclusive outcomes of the splitting criterion, we hope for the resulting partitions to be as pure as possible.

The node *N* is labeled with the splitting criterion, which serves as a test at the node (step 7). A branch is grown from node *N* for each of the outcomes of the splitting criterion. The tuples in *D* are partitioned accordingly (steps 10 to 11). There are three possible scenarios, as illustrated in Figure. Let *A* be the splitting attribute. *A* has *v* distinct values, $\{a1, a2, : : : , av\}$, based on the training data.



Three possibilities for partitioning tuples based on the splitting criterion, shown with examples. Let A be the splitting attribute. (a) If A is discrete-valued, then one branch is grown for each known value of A. (b) If A is continuous-valued, then two branches are grown, corresponding to $A \leq split\_point$ and $A > split\_point$. (c) If A is discrete-valued and a binary tree must be produced, then the test is of the form $A \in S_A$, where S_A is the splitting subset for A.

Attribute Selection Measures

An attribute selection measure is a heuristic for selecting the splitting criterion that ―best‖ separates a given data partition, *D*, of class-labeled training tuples into individual classes. If we were to split *D* into smaller partitions according to the outcomes of the splitting criterion, If the splitting attribute is continuous-valued or if we are restricted to binary trees then, respectively, either a *split point* or a *splitting subset* must also be determined as part of the splitting criterion This section describes three popular attribute selection measures—*information gain, gain ratio*, and *gini inde*

Information gain:ID3 uses information gain as its attribute selection measure.

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

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

Information gain is defined as the difference between the original information requirement (i.e., based on just the proportion of classes) and the new requirement (i.e., obtained after partitioning on *A*). That is,

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

In other words, *Gain(A)* tells us how much would be gained by branching on *A*. It is the expected reduction in the information requirement caused by knowing the value of *A*. The attribute *A* with the highest information gain, (*Gain*(*A*)), is chosen as the splitting attribute at node *N*.

Example Induction of a decision tree using information gain.

Table 6.1 presents a training set, *D*, of class-labeled tuples randomly selected from the *AllElectronics* customer database. (The data are adapted from [Qui86]. In this example, each attribute is discrete-valued. Continuous-valued attributes have been generalized.) The class label attribute, *buys computer*, has two distinct values (namely, {*yes, no*}); therefore, there are two distinct classes (that is, $m = 2$). Let class *C*1 correspond to *yes* and class *C*2 correspond to *no*. There are nine tuples of class *yes* and five tuples of class *no*. A (root) node *N* is created for the tuples in *D*. To find the splitting criterion for these tuples, we must compute the information gain of each attribute. We first use Equation (6.1) to compute the expected information needed to classify a tuple in *D*:

$$Info(D) = -\frac{9}{14}\log_2\left(\frac{9}{14}\right) - \frac{5}{14}\log_2\left(\frac{5}{14}\right) = 0.940 \text{ bits.}$$

Table 6.1 Class-labeled training tuples from the *AllElectronics* customer database.

RID	age	income	student	credit_rating	Class: buys_computer
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middle_aged	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
6	senior	low	yes	excellent	no
7	middle_aged	low	yes	excellent	yes
8	youth	medium	no	fair	no
9	youth	low	yes	fair	yes
10	senior	medium	yes	fair	yes
11	youth	medium	yes	excellent	yes
12	middle_aged	medium	no	excellent	yes
13	middle_aged	high	yes	fair	yes
14	senior	medium	no	excellent	no

The expected information needed to classify a tuple in D if the tuples are partitioned according to *age* is

$$\begin{aligned} Info_{age}(D) &= \frac{5}{14} \times \left(-\frac{2}{5}\log_2\frac{2}{5} - \frac{3}{5}\log_2\frac{3}{5}\right) \\ &+ \frac{4}{14} \times \left(-\frac{4}{4}\log_2\frac{4}{4} - \frac{0}{4}\log_2\frac{0}{4}\right) \\ &+ \frac{5}{14} \times \left(-\frac{3}{5}\log_2\frac{3}{5} - \frac{2}{5}\log_2\frac{2}{5}\right) \\ &= 0.694 \text{ bits.} \end{aligned}$$

Hence, the gain in information from such a partitioning would be

$$Gain(age) = Info(D) - Info_{age}(D) = 0.940 - 0.694 = 0.246 \text{ bits.}$$

Similarly, we can compute *Gain*(*income*) = 0.029 bits, *Gain*(*student*) = 0.151 bits, and *Gain*(*credit rating*) = 0.048 bits. Because *age* has the highest information gain among the attributes, it is selected as the splitting attribute. Node N is labeled with *age*, and branches are grown for each of the attribute's values. The tuples are then partitioned accordingly, as shown in Figure 6.5. Notice that the tuples falling into the partition for *age* = *middle aged* all belong to the same class. Because they all belong to class *"yes,"* a leaf should therefore be created at the end of this branch and

labeled with *"yes."* The final decision tree returned by the algorithm is shown in Figure 6.5.

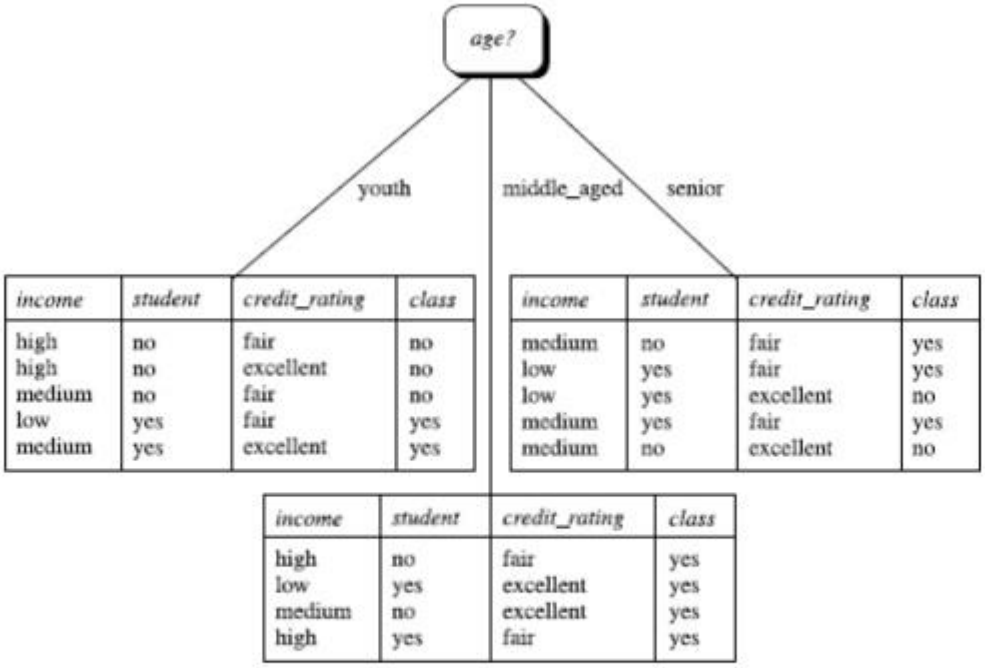


Figure 6.5 The attribute *age* has the highest information gain and therefore becomes the splitting attribute at the root node of the decision tree. Branches are grown for each outcome of *age*. The tuples are shown partitioned accordingly.

Tree Pruning

Introduction: When a decision tree is built, many of the branches will reflect anomalies in the training data due to noise or outliers. Tree pruning methods address this problem of over fitting the data. Such methods typically use statistical measures to remove the least reliable branches. An un-pruned tree and a pruned version of it are shown in Figure 6.6. Pruned trees tend to be smaller and less complex and, thus, easier to comprehend. They are usually faster and better at correctly classifying independent test data (i.e., of previously unseen tuples) than un-pruned trees.

"How does tree pruning work?" **There are two common approaches to tree pruning: pre pruning and post pruning.**

In the pre pruning approach, a tree is "pruned" by halting its construction early (e.g., by deciding not to further split or partition the subset of training tuples at a given node).

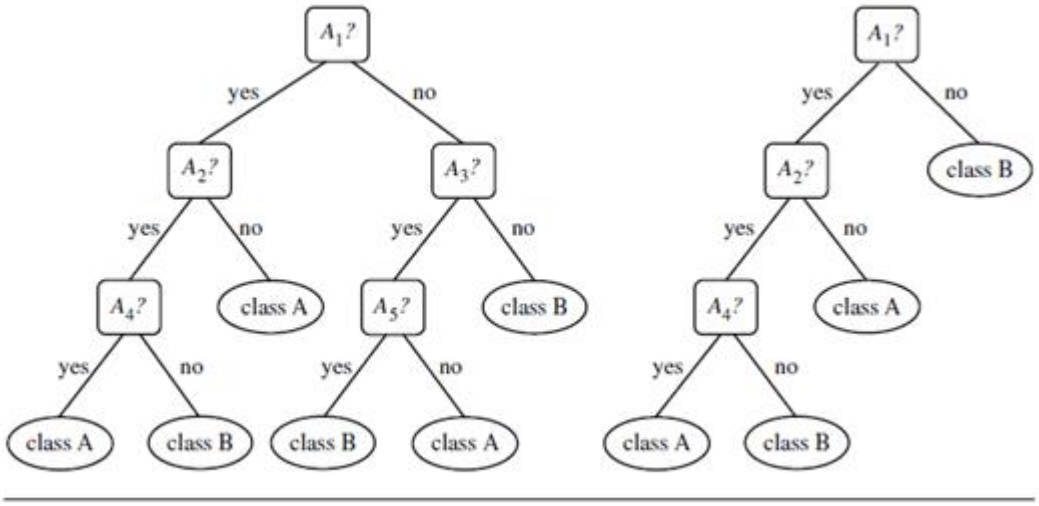


Figure 6.6 An unpruned decision tree and a pruned version of it.

Upon halting, the node becomes a leaf. The leaf may hold the most frequent class among the subset tuples or the probability distribution of those tuples.

When constructing a tree, measures such as statistical significance, information gain, Gini index, and so on can be used to assess the goodness of a split. If partitioning the tuples at a node would result in a split that falls below a pre specified threshold, then further partitioning of the given subset is halted. There are difficulties, however, in choosing an appropriate threshold. High thresholds could result in oversimplified trees, whereas low thresholds could result in very little simplification.

The second and more common approach is post pruning, which removes subtrees from a "fully grown" tree. A subtree at a given node is pruned by removing its branches and replacing it with a leaf. The leaf is labeled with the most frequent class among the subtree being replaced. For example, notice the subtree at node "A3?" in the un pruned tree of Figure 6.6. Suppose that the most common class within this subtree is "class B." In the pruned version of the tree, the sub tree in question is pruned by replacing it with the leaf "class B."

Bayesian Classification

"What are Bayesian classifiers?" Bayesian classifiers are statistical classifiers. They can predict class membership probabilities, such as the probability that a given tuple belongs to a particular class.

Bayesian classification is based on Bayes' theorem, a simple Bayesian classifier known as the *naïve Bayesian classifier* Bayesian classifiers have also exhibited high accuracy and speed when applied to large databases.

1. Bayes' Theorem

Let ***X*** be a data tuple. In Bayesian terms, ***X*** is considered ―evidence.‖ As usual, it is described by measurements made on a set of *n* attributes. Let *H* be some hypothesis, such as that the data tuple ***X*** belongs to a specified class *C*. For classification problems, we want to determine $P(H/X)$, the probability that the hypothesis *H* holds given the ―evidence‖ or observed data tuple ***X***. In other words, we are looking for the probability that tuple ***X*** belongs to class *C*, given that we know the attribute description of ***X***.

"How are these probabilities estimated?" $P(H)$, $P(\boldsymbol{X}/H)$, and $P(\boldsymbol{X})$ may be estimated from the given data, as we shall see below. Bayes' theorem is useful in that it provides a way of calculating the posterior probability, $P(H/\boldsymbol{X})$, from $P(H)$, $P(\boldsymbol{X}/H)$, and $P(\boldsymbol{X})$.

Bayes' theorem is

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

2. Naïve Bayesian Classification

1. Let D be a training set of tuples and their associated class labels. As usual, each tuple is represented by an n-dimensional attribute vector, $X = (x_1, x_2, \ldots, x_n)$, depicting n measurements made on the tuple from n attributes, respectively, $A_1, A_2, \ldots, A_n$.

2. Suppose that there are m classes, $C_1, C_2, \ldots, C_m$. Given a tuple, X, the classifier will predict that X belongs to the class having the highest posterior probability, conditioned on X. That is, the naïve Bayesian classifier predicts that tuple X belongs to the class C_i if and only if

$$P(C_i|X) > P(C_j|X) \quad \text{for } 1 \leq j \leq m, j \neq i.$$

Thus we maximize $P(C_i|X)$. The class C_i for which $P(C_i|X)$ is maximized is called the *maximum posteriori hypothesis*. By Bayes' theorem (Equation (6.10)),

$$P(C_i|X) = \frac{P(X|C_i)P(C_i)}{P(X)}. \tag{6.11}$$

3. As $P(X)$ is constant for all classes, only $P(X|C_i)P(C_i)$ need be maximized. If the class prior probabilities are not known, then it is commonly assumed that the classes are equally likely, that is, $P(C_1) = P(C_2) = \cdots = P(C_m)$, and we would therefore maximize $P(X|C_i)$. Otherwise, we maximize $P(X|C_i)P(C_i)$. Note that the class prior probabilities may be estimated by $P(C_i) = |C_{i,D}|/|D|$, where $|C_{i,D}|$ is the number of training tuples of class C_i in D.

4. Given data sets with many attributes, it would be extremely computationally expensive to compute $P(X|C_i)$. In order to reduce computation in evaluating $P(X|C_i)$, the naive assumption of **class conditional independence** is made. This presumes that the values of the attributes are conditionally independent of one another, given the class label of the tuple (i.e., that there are no dependence relationships among the attributes). Thus,

$$\begin{aligned} P(X|C_i) &= \prod_{k=1}^{n} P(x_k|C_i) \\ &= P(x_1|C_i) \times P(x_2|C_i) \times \cdots \times P(x_n|C_i). \end{aligned}$$

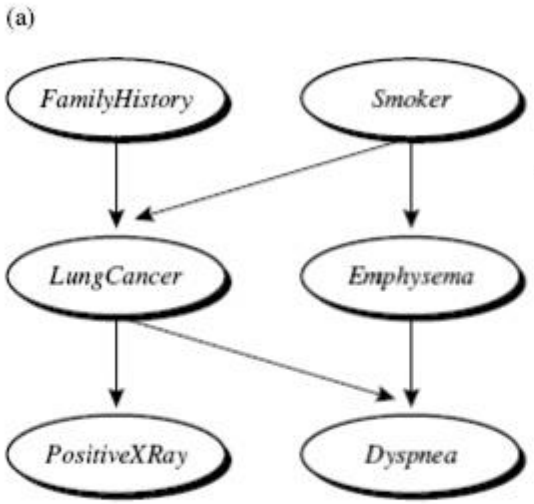
5. In order to predict the class label of X, $P(X|C_i)P(C_i)$ is evaluated for each class C_i. The classifier predicts that the class label of tuple X is the class C_i if and only if

$$P(X|C_i)P(C_i) > P(X|C_j)P(C_j) \quad \text{for } 1 \leq j \leq m, j \neq i. \tag{6.15}$$

In other words, the predicted class label is the class C_i for which $P(X|C_i)P(C_i)$ is the maximum.

Bayesian Belief Networks

A belief network is defined by two components—a *directed acyclic graph* and a set of *conditional probability tables* (Figure 6.11). Each node in the directed acyclic graph represents a random variable. The variables may be discrete or continuous-valued. They may correspond to actual attributes given in the data or to ―hidden variables‖ believed to form a relationship (e.g., in the case of medical data, a hidden variable may indicate a syndrome, representing a number of symptoms that, together, characterize a specific disease). Each arc represents a probabilistic dependence. If an arc is drawn from a node *Y* to a node *Z*, then *Y* is a parent or immediate predecessor of *Z*, and *Z* is a descendant of *Y*. *Each variable is conditionally independent of its non descendants in the graph, given its parents.*



(b)

	FH, S	FH, ~S	~FH, S	~FH, ~S
LC	0.8	0.5	0.7	0.1
~LC	0.2	0.5	0.3	0.9

A simple Bayesian belief network: (a) A proposed causal model, represented by a directed acyclic graph. (b) The conditional probability table for the values of the variable *LungCancer (LC)* showing each possible combination of the values of its parent nodes, *FamilyHistory (FH)* and *Smoker (S)*. Figure is adapted from [RBKK95].

A belief network has one conditional probability table (CPT) for each variable. The CPT for a variable *Y* specifies the conditional distribution $P(Y|Parents(Y))$, where $Parents(Y)$ are the parents of *Y*. Figure(b) shows a CPT for the variable *LungCancer*. The conditional probability for each known value of *LungCancer* is given for each possible combination of values of its parents. For instance, from the upper leftmost and bottom rightmost entries, respectively, we see that

$$P(LungCancer = yes \mid FamilyHistory = yes, Smoker = yes) = 0.8$$
$$P(LungCancer = no \mid FamilyHistory = no, Smoker = no) = 0.9$$

Let $\boldsymbol{X}$ = ($x1$, : : : , xn) be a data tuple described by the variables or attributes $Y1$, : : : , Yn, respectively. Recall that each variable is conditionally independent of its non descendants in the network graph, given its parents. This allows the network to provide a complete representation of the existing joint probability distribution with the following equation:

$$P(x_1,\ldots,x_n) = \prod_{i=1}^{n} P(x_i|Parents(Y_i)),$$

Rule Based Classification

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

Assessment of a rule: *coverage* and *accuracy*

- n_{covers} = # of tuples covered by R
- $n_{correct}$ = # of tuples correctly classified by R
- o coverage(R) = n_{covers} /|D| /* D: training data set */
- accuracy(R) = $n_{correct}$ / n_{covers}

If more than one rule is triggered, need conflict resolution

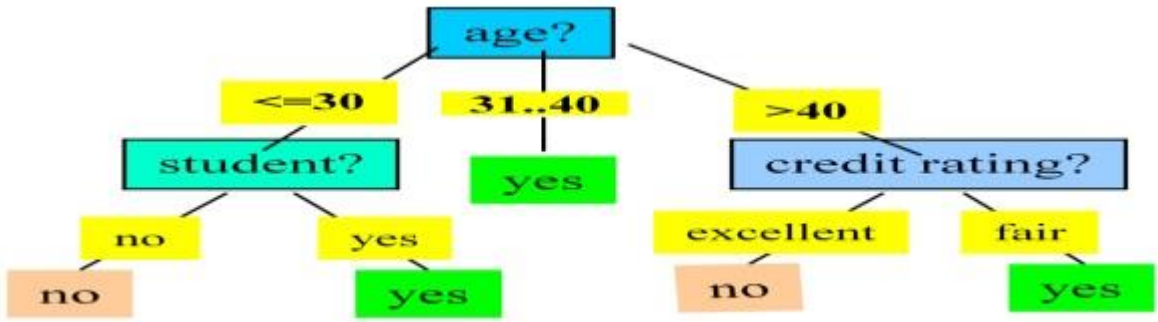
- Size ordering: assign the highest priority to the triggering rules that has the
- toughest‖ requirement (i.e., with the *most attribute test*)
- Class-based ordering: decreasing order of *prevalence or misclassification cost per class*

- Rule-based ordering (decision list): rules are organized into one long priority list, according to some measure of rule quality or by experts

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* = *yes*
- IF *age* = young AND *credit_rating* = *fair* THEN *buys_computer* = *no*

Rule Extraction from the Training Data

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
- The process repeats on the remaining tuples unless *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*

Lazy Learners (or Learning from Your Neighbors)

The classification methods discussed so far in this chapter—decision tree induction, Bayesian classification, rule-based classification, classification by backpropagation, support vector machines, and classification based on association rule mining—are all examples of *eager learners.* Eager learners, when given a set of training tuples, will construct a generalization (i.e., classification) model before receiving new (e.g., test) tuples to classify. We can think of the learned model as being ready and eager to classify previously unseen tuples.

***k*-Nearest-Neighbor Classifiers**

The *k*-nearest-neighbor method was first described in the early 1950s. The method is labor intensive when given large training sets, and did not gain popularity until the 1960s when increased computing power became available. It has since been widely used in the area of pattern recognition.

Nearest-neighbor classifiers are based on learning by analogy, that is, by comparing a given test tuple with training tuples that are similar to it. The training tuples are described by *n* attributes. Each tuple represents a point in an *n*-dimensional space. In this way, all of the training tuples are stored in an *n*-dimensional pattern space. When given an unknown tuple, a ***k***-nearest-neighbor classifier searches the pattern space for the *k* training tuples that are closest to the unknown tuple. These *k* training tuples are the *k* ―nearest neighbors‖ of the unknown tuple.

―Closeness‖ is defined in terms of a distance metric, such as Euclidean distance. The

Euclidean distance between two points or tuples, say, $\boldsymbol{X}1 = (x11, x12, : : : , x1n)$ and $\boldsymbol{X}2 = (x21, x22, : : , x2n)$, is

$$dist(X_1, X_2) = \sqrt{\sum_{i=1}^{n} (x_{1i} - x_{2i})^2}.$$

Case-Based Reasoning

Case-based reasoning (CBR) classifiers use a database of problem solutions to solve new problems. Unlike nearest-neighbor classifiers, which store training tuples as points in Euclidean space, CBR stores the tuples or ―cases‖ for problem solving as complex symbolic descriptions. Business applications of CBR include problem resolution for customer service help desks, where cases describe product-related diagnostic problems. CBR has also been applied to areas such as engineering and law, where cases are either technical designs or legal rulings, respectively. Medical education is another area for CBR, where patient case histories and treatments are used to help diagnose and treat new patients.

When given a new case to classify, a case-based reasoner will first check if an identical training case exists. If one is found, then the accompanying solution to that case is returned. If no identical case is found, then the case-based reasoner will search for training cases having SCE Department of Information Technology components that are similar to those of the new case. Conceptually, these training cases may be considered as neighbors of the new case. If cases are represented as graphs, this involves searching for subgraphs that are similar to subgraphs within the new case. The case-based reasoner tries to combine the solutions of the neighbouring training cases in order to propose a solution for the new case. If incompatibilities arise with the individual solutions, then backtracking to search for other solutions may be necessary. The case-based reasoner may employ background knowledge and problem-solving strategies in order to propose a feasible combined solution.

Other Classification Methods

Genetic Algorithms

Genetic Algorithm: based on an analogy to biological evolution

- An initial **population** is created consisting of randomly generated rules

  - Each rule is represented by a string of bits
- E.g., if A_1 and $\neg A_2$ then C_2 can be encoded as 100 o If an attribute has k > 2 values, k bits can be used
- Based on the notion of survival of the **fittest**, a new population is formed to consist of the fittest rules and their offsprings
- The fitness of a rule is represented by its *classification accuracy* on a set of training examples
- Offsprings are generated by *crossover* and *mutation*
- The process continues until a population P evolves *when each rule in P satisfies a prespecified threshold*
- Slow but easily parallelizable

Rough Set Approach:

o Rough sets are used to **approximately or —roughly‖ define equivalent classes**

A rough set for a given class C is approximated by two sets: a lower approximation (certain to be in C) and an upper approximation (cannot be described as not belonging to C)

- Finding the minimal subsets (**reducts**) of attributes for feature reduction is NP-hard but a **discernibility matrix** (which stores the differences between attribute values for each pair of data tuples) is used to reduce the computation intensity

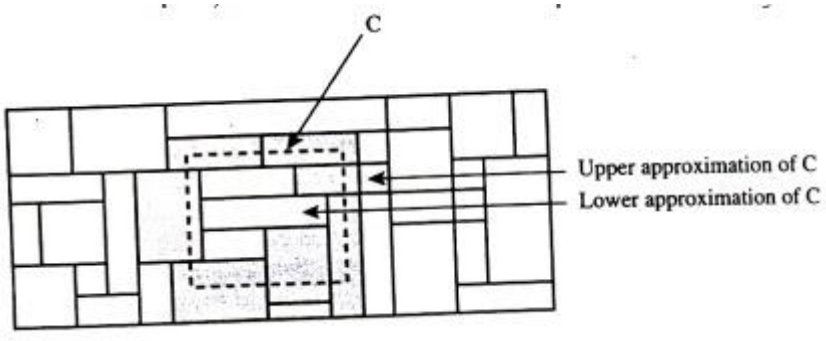


Figure: A rough set approximation of the set of tuples of the class C suing lower and upper approximation sets of C. The rectangular regions represent equivalence classes

Fuzzy Set approaches

- Fuzzy logic uses truth values between 0.0 and 1.0 to represent the degree of membership (such as using fuzzy membership graph)
- Attribute values are converted to fuzzy values

e.g., income is mapped into the discrete categories {low, medium, high} with fuzzy values calculated

- For a given new sample, more than one fuzzy value may apply

- Each applicable rule contributes a vote for membership in the categories
- Typically, the truth values for each predicted category are summed, and these sums are combined