Classification and Prediction

Classification is the process of associating classes with a data set or dividing the data sets into specific class names.

On the other hand, **Prediction** is the process of finding a specific value based trained data model. For example, we can build a classification model to categorize bank loan applications as either safe or risky, or a prediction model to predict the expenditures in dollars of potential customers on computer equipment given their income and occupation

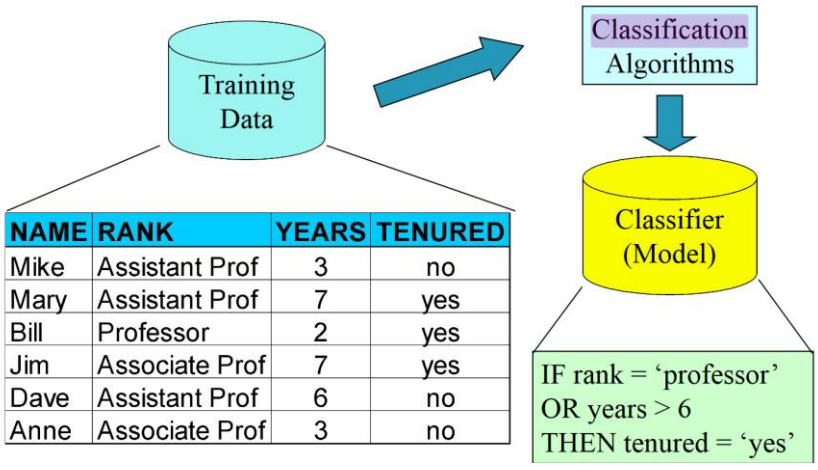
Learning and testing of classification

How classification Works:

In detail, classification is a two-step process:

- **Model construction:**

It is the process of 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.

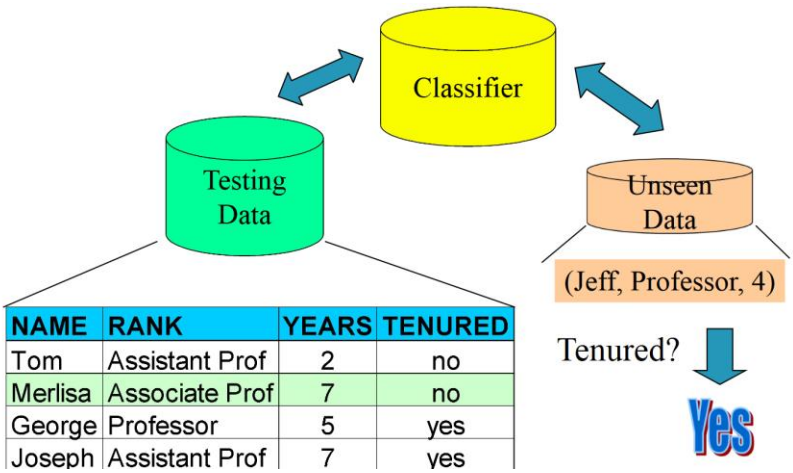


NAME	RANK	YEARS	TENURED
Mike	Assistant Prof	3	no
Mary	Assistant Prof	7	yes
Bill	Professor	2	yes
Jim	Associate Prof	7	yes
Dave	Assistant Prof	6	no
Anne	Associate Prof	3	no

- **Model usage:**

It is the process of classifying the future or unknown objects. 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. If the accuracy is acceptable, use the model to classify data tuples whose class labels are not known.



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

Issues in Data Classification:

The accuracy of the classifier is should be high to correctly predict the class labels for future objects. To improve accuracy, efficiency and scalability of a classifier, following measures can be used:

Data cleaning: This refers to the preprocessing of data in order to remove or reduce noise and the treatment of missing values. Although most classification algorithms have some mechanisms for handling noisy or missing data, this step can help reduce confusion during learning.

Relevance analysis: Correlation analysis can be used to identify whether any two given attributes are statistically related. A database may also contain irrelevant attributes. Attribute subset selection can be used in these cases to find a reduced set of attributes such that the resulting probability distribution of the data classes is as close as possible to the original distribution obtained using all attributes.

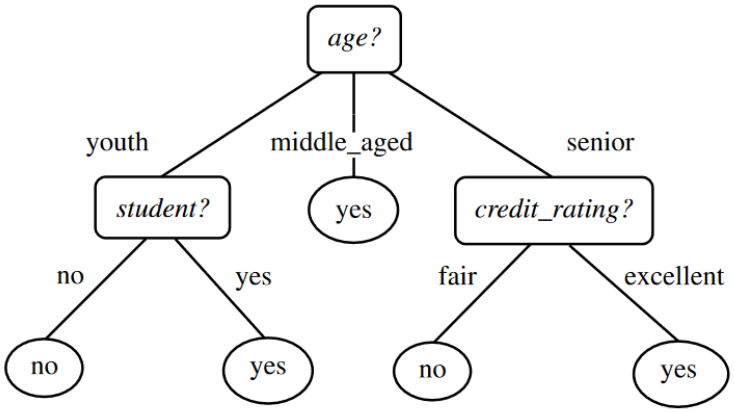
Data transformation and reduction: The data may be transformed by **normalization**, particularly when neural networks or methods involving distance measurements are used in the learning step. The data can also be transformed by **generalizing** it to higher-level concepts. Data can also be reduced by applying many other methods, ranging from wavelet transformation and

principle components analysis to discretization techniques, such as binning, histogram analysis, and clustering.

Classification by decision tree induction

Decision tree induction is the learning of decision trees from class-labeled training tuples. A decision tree is a flowchart-like tree structure, where each internal node (nonleaf node) denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (or terminal node) holds a class label. The topmost node in a tree is the root node.

For example:



Data Set	Age	Student	Credit Rating	Buys computer
Ram	Youth	No	Excellent	Yes
Shyam	Middle aged	Yes	Excellent	Yes
Hari	Senior	No	excellent	Yes
Gopal	Youth	yes	Fair	Yes
Chandu	Middle aged	No	fair	No

Attribute selection Measure (ID3 as attribute selection algorithm)

Which attribute to select first or next?

Iterative Dichotomiser 3 is one of the best algorithms that uses Information gain for selecting attribute for decision tree induction.

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

$$\text{Info}_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} * \text{Info}(D_j)$$

$$\text{Gain}(A) = \text{Info}(D) - \text{Info}_A(D)$$

For example:

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

$$\boldsymbol{Info(D) = -\frac{9}{14}\log_2\left(\frac{9}{14}\right) - \frac{5}{14}\log_2\left(\frac{5}{14}\right)}$$

$$\boldsymbol{= 0.940\ bits}$$

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

$$\boldsymbol{Gain(age) = Info(D) - Info_{age}(D) = 0.94 - 0.694 = 0.246\ bits}$$

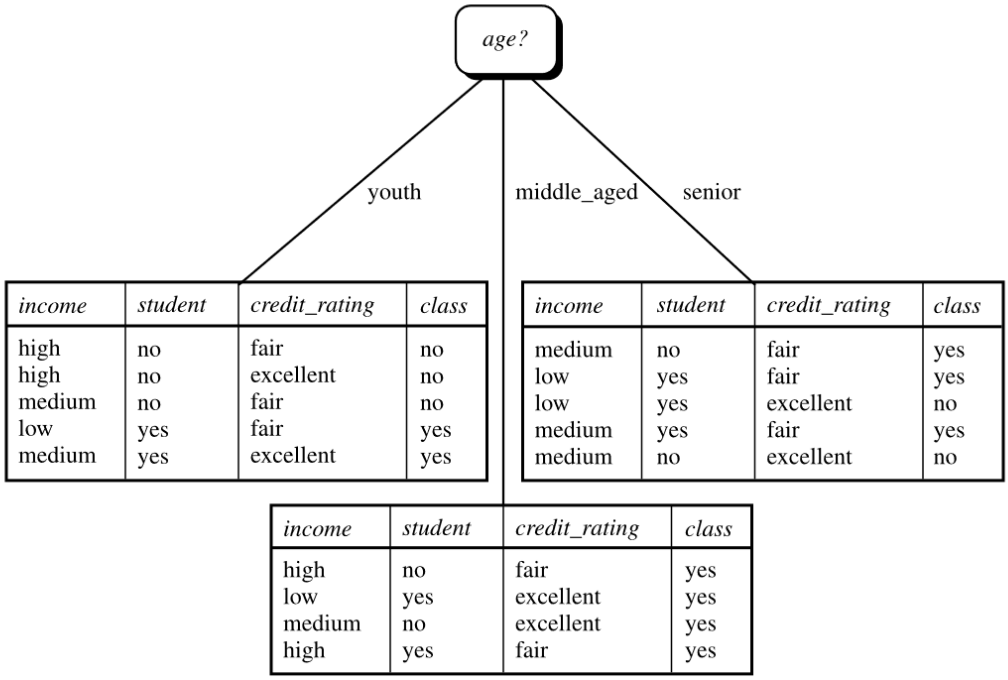
Similarly,

Gain (income) = 0.029 bits,

Gain (student) = 0.151 bits, and

Gain (credit rating) = 0.048 bits.

Hence, age has the highest information gain and is selected as the splitting attribute.



income	*student*	*credit_rating*	*class*
high	no	fair	no
high	no	excellent	no
medium	no	fair	no
low	yes	fair	yes
medium	yes	excellent	yes

income	*student*	*credit_rating*	*class*
medium	no	fair	yes
low	yes	fair	yes
low	yes	excellent	no
medium	yes	fair	yes
medium	no	excellent	no

income	*student*	*credit_rating*	*class*
high	no	fair	yes
low	yes	excellent	yes
medium	no	excellent	yes
high	yes	fair	yes

Bayesian classification

Bayesian classification is based on Bayes' theorem.

Bayes' Theorem:

Bayes' Theorem states that the conditional probability of an event, based on the occurrence of another event, is equal to the likelihood of the second event given the first event multiplied by the probability of the first event.

$$P\left(\frac{A}{B}\right) = \frac{P(A \cap B)}{P(B)} \quad = P(A) \cdot \frac{P\left(\frac{B}{A}\right)}{P(B)}$$

where:

$P(A) = \ The\ probability\ of\ A\ occurring$

$P(B) = \ The\ probability\ of\ B\ occurring$

$P\left(\frac{A}{B}\right) = The\ probability\ of\ A\ given\ B$

$P\left(\frac{B}{A}\right) = \ The\ probability\ of\ B\ given\ A$

$P(A \cap B) = \ The\ probability\ of\ both\ A\ and\ B\ occurring$

The naïve Bayesian classifier, or simple Bayesian classifier, works as follows:

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 = (x1, x2, . . . , xn), depicting n measurements made on the tuple from n attributes, respectively, A1, A2, . . . , An. 2. Suppose that there are m classes, C1, C2, . . . , Cm. Given a tuple, X, the classifier will predict that X belongs to the class having the **highest posterior probability, conditioned on X (**maximum posteriori hypothesis). By Bayes' theorem,

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

1. As P(X) is constant for all classes, only $P\left(\frac{X}{C_i}\right)P(C_i)$ need be maximized. If the class prior probabilities are not known, then it is commonly assumed that the equally likely, that is, P(C1) = P(C2) = · · · = P(Cm), and we would therefore maximize P(X|Ci). Otherwise, we maximize P(X|Ci)P(Ci).
2. Given data sets with many attributes, it would be extremely computationally expensive to compute P(X|Ci). In order to reduce computation in evaluating P(X|Ci), 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,

$$P\left(\frac{X}{Ci}\right) = \prod_{k=1}^{n} P\left(\frac{x_k}{C_i}\right) = P\left(\frac{x_1}{C_i}\right) \times P\left(\frac{X_2}{C_i}\right) \times \cdots \times P\left(P\left(\frac{X_n}{C_i}\right)\right)$$

We can easily estimate the probabilities P(x1|Ci), P(x2|Ci), . . . , P(xn|Ci) from the training tuples.

For example:

In above table suppose we need to classify:

X = (age = youth, income = medium, student = yes, credit rating = fair)

$$P(buys\ computer = yes) = 9/14 = 0.643$$

$$P(buys\ computer = no) = 5/14 = 0.357$$

To compute $P\left(\frac{X}{Ci}\right)$ for i = 1, 2, we compute the following conditional probabilities:

$$P(age = youth \mid buys\ computer = yes) = 2/9 = 0.222$$

$$P(age = youth \mid buys\ computer = no) = 3/5 = 0.600$$

$$P(income = medium \mid buys\ computer = yes) = 4/9 = 0.444$$

$$P(income = medium \mid buys\ computer = no) = 2/5 = 0.400$$

$$P(student = yes \mid buys\ computer = yes) = 6/9 = 0.667$$

$$P(student = yes \mid buys\ computer = no) = 1/5 = 0.200$$

$$P(credit\ rating\ =\ fair \mid buys\ computer\ =\ yes)\ =\ 6/9\ =\ 0.667$$

$$P(credit\ rating\ =\ fair \mid buys\ computer\ =\ no)\ =\ 2/5\ =\ 0.400$$

Using the above probabilities, we obtain

$$\boldsymbol{P(X|buys\ computer\ =\ yes)}$$

$$=\ P(age\ =\ youth \mid buys\ computer\ =\ yes)\ \times$$

$$P(income\ =\ medium \mid buys\ computer\ =\ yes)\ \times$$

$$P(student\ =\ yes \mid buys\ computer\ =\ yes)\ \times$$

$$P(credit\ rating\ =\ fair \mid buys\ computer\ =\ yes)$$

$$=\ 0.222\ \times\ 0.444\ \times\ 0.667\ \times\ 0.667$$

$$=\ 0.044$$

Similarly,

$$\boldsymbol{P(X|buys\ computer\ =\ no)}$$

$$=\ 0.600\ \times\ 0.400\ \times\ 0.200\ \times\ 0.400$$

$$=\ 0.019$$

Therefore, the naïve Bayesian classifier predicts buys computer = yes for tuple X.

Laplace smoothing (*What if we get probability values of zero?)*

We can assume that our training database, D, is so large that adding one to each count that we need would only make a negligible difference in the estimated probability value, yet would conveniently avoid the case of probability values of zero. This technique for probability estimation is known as the **Laplacian correction** or **Laplace estimator** or **Laplace smoothing**.

For example:

Suppose that for the class buys computer = yes in some training database, D, containing 1,000 tuples,

we have 0 tuples with income = low,

990 tuples with income = medium, and

10 tuples with income = high.

The probabilities of these events, **without the Laplacian correction**, are

0,

0.990 and

0.010

respectively. **Using the Laplacian correction** for the three quantities, we pretend that we have 1 more tuple for each income-value pair. In this way, we instead obtain the following probabilities:

$$\frac{1}{1003} = 0.001,$$

$$\frac{991}{1003} = 0.988, and$$

$$\frac{11}{1003} = 0.011,$$

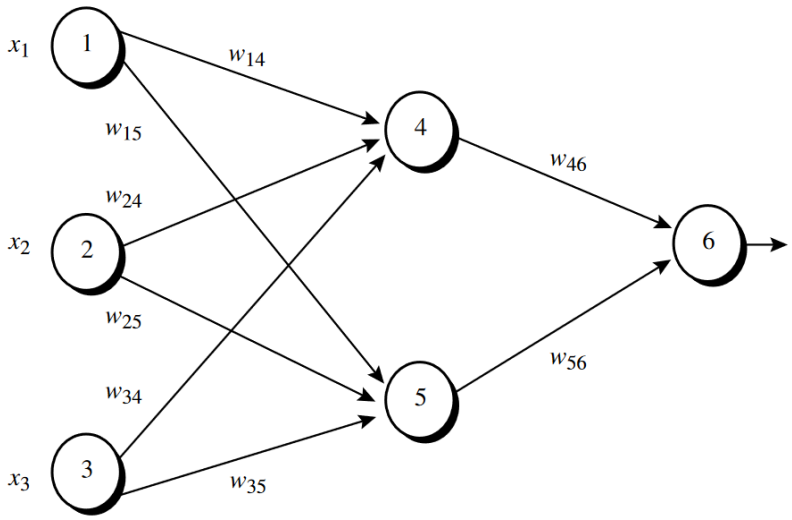
respectively. The "corrected" probability estimates are close to their "uncorrected" counterparts, yet the zero probability value is avoided.

Classification by backpropagation

Backpropagation learns by iteratively processing a data set of training tuples, comparing the network's prediction for each tuple with the actual known target value. **The target value may be the known class label of the training tuple (for classification problems) or a continuous value (for prediction).** For each training tuple, the weights are modified so as to minimize the mean squared error between the network's prediction and the actual target value. These modifications are made in the "backwards" direction, that is, from the output layer, through each hidden layer down to the first hidden layer (hence the name backpropagation). Although it is not guaranteed, in general the weights will eventually converge, and the learning process stops.

Initialize all weights and biases in *network*;
while terminating condition is not satisfied {
    for each training tuple $\boldsymbol{X}$ in D {
        // Propagate the inputs forward:
        for each input layer unit j {
            $O_j = I_j$; // output of an input unit is its actual input value
        for each hidden or output layer unit j {
            $I_j = \sum_i w_{ij} O_i + \theta_j$; //compute the net input of unit j with respect to the previous layer, i
            $O_j = \frac{1}{1+e^{-I_j}}$; } // compute the output of each unit j
        // Backpropagate the errors:
        for each unit j in the output layer
            $Err_j = O_j(1 - O_j)(T_j - O_j)$; // compute the error
        for each unit j in the hidden layers, from the last to the first hidden layer
            $Err_j = O_j(1 - O_j)\sum_k Err_k w_{jk}$; // compute the error with respect to the next higher layer, k
        for each weight w_{ij} in *network* {
            $\Delta w_{ij} = (l) Err_j O_i$; // weight increment
            $w_{ij} = w_{ij} + \Delta w_{ij}$; } // weight update
        for each bias θ_j in *network* {
            $\Delta\theta_j = (l) Err_j$; // bias increment
            $\theta_j = \theta_j + \Delta\theta_j$; } // bias update
    } }

For example:



Initial input, weight and bias values:

x_1	x_2	x_3	w_{14}	w_{15}	w_{24}	w_{25}	w_{34}	w_{35}	w_{46}	w_{56}	θ_4	θ_5	θ_6
1	0	1	0.2	−0.3	0.4	0.1	−0.5	0.2	−0.3	−0.2	−0.4	0.2	0.1

Calculate Net Input and Output for each hidden and output layer:

Unit j	*Net input,* I_j	*Output,* O_j
4	$0.2+0-0.5-0.4=-0.7$	$1/(1+e^{0.7})=0.332$
5	$-0.3+0+0.2+0.2=0.1$	$1/(1+e^{-0.1})=0.525$
6	$(-0.3)(0.332)-(0.2)(0.525)+0.1=-0.105$	$1/(1+e^{0.105})=0.474$

Calculating errors:

Unit j	Err_j
6	$(0.474)(1-0.474)(1-0.474)=0.1311$
5	$(0.525)(1-0.525)(0.1311)(-0.2)=-0.0065$
4	$(0.332)(1-0.332)(0.1311)(-0.3)=-0.0087$

Update weight and bias:

Weight or bias	*New value*
w_{46}	$-0.3+(0.9)(0.1311)(0.332)=-0.261$
w_{56}	$-0.2+(0.9)(0.1311)(0.525)=-0.138$
w_{14}	$0.2+(0.9)(-0.0087)(1)=0.192$
w_{15}	$-0.3+(0.9)(-0.0065)(1)=-0.306$
w_{24}	$0.4+(0.9)(-0.0087)(0)=0.4$
w_{25}	$0.1+(0.9)(-0.0065)(0)=0.1$
w_{34}	$-0.5+(0.9)(-0.0087)(1)=-0.508$
w_{35}	$0.2+(0.9)(-0.0065)(1)=0.194$
θ_6	$0.1+(0.9)(0.1311)=0.218$
θ_5	$0.2+(0.9)(-0.0065)=0.194$
θ_4	$-0.4+(0.9)(-0.0087)=-0.408$

Rule based classifier

Rules are a good way of representing information or bits of knowledge. A rule-based classifier uses a set of IF-THEN rules for classification. An IF-THEN rule is an expression of the form IF condition THEN conclusion. For example:

R1: IF age = youth AND student = yes THEN buys computer = yes.

Covers (n_{covers}):

If the condition (that is, all of the attribute tests) in a rule antecedent holds true for a given tuple, we say that the rule is satisfied and that the rule covers the tuple.

Coverage:

Coverage is the percentage of tuples that are covered by the rule (i.e., whose attribute values hold true for the rule's antecedent).

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

Accuracy:

Accuracy is percentage of covered tuples that the rule can correctly classify. i.e.

$$accuracy(R) = \frac{n_{correct}}{n_{covers}}$$

For example:

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

R1: IF age = youth AND student = yes THEN buys computer = yes.

Then,

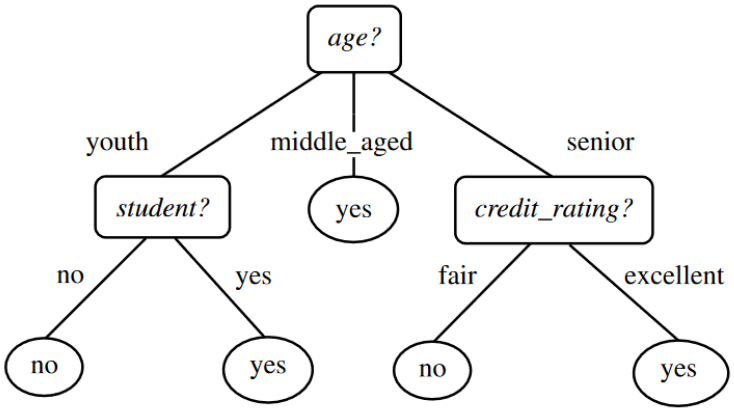
$n_{covers} = 2$

$$coverage(R) = \frac{n_{covers}}{|D|} = \frac{2}{14} = 14.28\%$$

$$accuracy(R) = \frac{n_{correct}}{n_{covers}} = \frac{2}{2} = 100\%$$

Decision Trees to Rules:

Decision Tree:



Equivalent Rules:

R1: IF age = youth AND student = no THEN buys computer = no

R2: IF age = youth AND student = yes THEN buys computer = yes

R3: IF age = middle aged THEN buys computer = yes

R4: IF age = senior AND credit rating = excellent THEN buys computer = no

R5: IF age = senior AND credit rating = fair THEN buys computer = yes

A disjunction (logical OR) is implied between each of the extracted rules. Because the rules are extracted directly from the tree, they are mutually exclusive and exhaustive. \

- By mutually exclusive, this means that we cannot have rule conflicts here because no two rules will be triggered for the same tuple.
- By exhaustive, there is one rule for each possible attribute-value combination, so that this set of rules does not require a default rule. Therefore, the order of the rules does not matter—they are unordered.

Efficient rule simplification:

Although it is easy to extract rules from a decision tree, we may need to do some more work by pruning the resulting rule set **to prevent subtree repetition and replication**. **For a given rule**

antecedent, any condition that does not improve the estimated accuracy of the rule can be pruned (i.e., removed), thereby generalizing the rule.

For example:

- C4.5 (variant of ID3 algorithm) extracts rules from an unpruned tree, and then **prunes** the rules using a **pessimistic approach**.
- However, the rules then will no longer be mutually exclusive and exhaustive and hence C4.5 adopts a **class-based ordering scheme**. It groups all rules for a single class together, and then determines a ranking of these class rule sets. C4.5 orders the class rule sets so as to minimize the number of false-positive errors (i.e., where a rule predicts a class, C, but the actual class is not C).

Support vector machine

Evaluating accuracy (precision, recall, f-measure)

Issues in classification

Overfitting and underfitting

K-fold cross validation

Comparing two classifiers (McNemar's Test)