**Classification:** Decision Trees Induction, Method for Comparing Classifiers, Rule Based Classifiers, Nearest Neighbor Classifiers, Bayesian Classifiers.

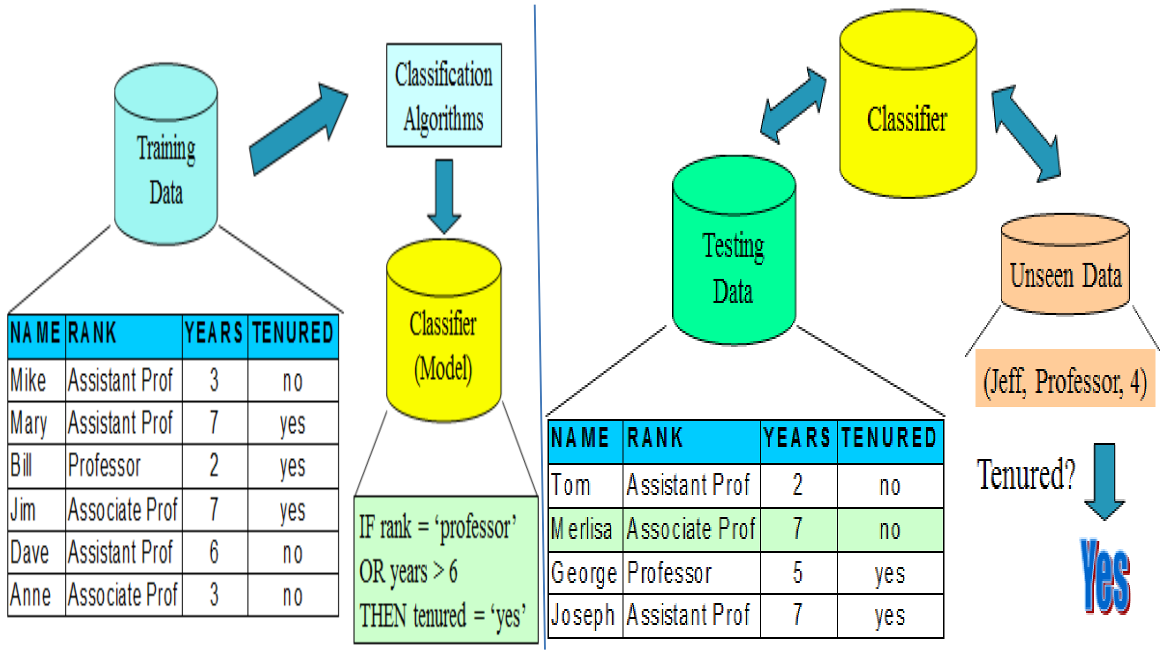
---

## Classification: Basic Concepts

- Supervised learning (classification)
    - Supervision: The Learning data (observations, measurements, etc.) are accompanied by labels indicating the class of the observations
    - New data is classified based on the Learning set
- Unsupervised learning (clustering)
    - The class labels of Learning data is unknown
    - Given a set of measurements, observations, etc. with the aim of establishing the existence of classes or clusters in the data

**Classification—A Two-Step Process**

- Model construction(Learning step): 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 Learning data set
    - The model is represented as classification rules, decision Trees, or mathematical formulae
- Model usage (classification step): 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 Learning set (otherwise overfitting)
    - If the accuracy is acceptable, use the model to classify new data



| 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 |

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

**Learning Step** **Classification Step**

## Decision Tree Induction

- Basic algorithm (a greedy algorithm)
    - Tree is constructed in a top-down recursive divide-and-conquer manner
    - At start, all the Learning 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: Many Algorithms:**

- Hunt's Algorithm (one of the earliest)
- CART
- ID3, C4.5

**Hunt's Algorithm**

1. Let $D_t$ be the set of Learning records that reach a node t
   General Procedure:
2. If $D_t$ contains records that belong the same class $y_t$, then t is a leaf node labeled as $y_t$
3. If $D_t$ contains records with the same attribute values, then t is a leaf node labeled with the majority class $y_t$
4. If $D_t$ is an empty set, then t is a leaf node labeled by the default class, $y_d$
5. If $D_t$ contains records that belong to more than one class, use an attribute test to split the data into smaller subsets.
6. Recursively apply the procedure to each subset.

**CART, ID3, C4.5 Algorithm : Generate decision Tree.** Generate a decision Tree from the Learning tuples of data partition, *D*.

**Input:**

- Data partition, *D*, which is a set of Learning 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*.

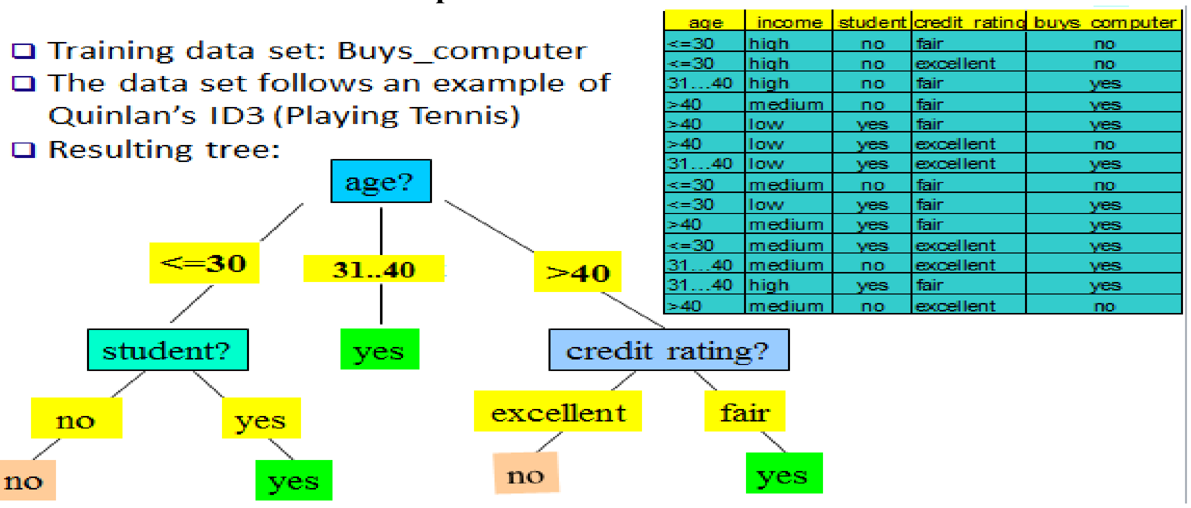
**Output:** A decision Tree.

**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 resLiicted 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;
```

**Decision Tree Induction: An Example**



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

| age | income | student | credit rating | buys computer |
|---|---|---|---|---|
| <=30 | high | no | fair | no |
| <=30 | high | no | excellent | no |
| 31…40 | high | no | fair | yes |
| >40 | medium | no | fair | yes |
| >40 | low | yes | fair | yes |
| >40 | low | yes | excellent | no |
| 31…40 | low | yes | excellent | yes |
| <=30 | medium | no | fair | no |
| <=30 | low | yes | fair | yes |
| >40 | medium | yes | fair | yes |
| <=30 | medium | yes | excellent | yes |
| 31…40 | medium | no | excellent | yes |
| 31…40 | high | yes | fair | yes |
| >40 | medium | no | excellent | no |

**Attribute Selection Measures:**

1. **Information Gain (ID3/C4.5)**
   - Select the attribute with the highest information gain
   - Let $p_i$ be the probability that an arbiLiary tuple in D belongs to class $C_i$, estimated by $|C_{i,D}|/|D|$
   - Expected information (enLiopy) 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

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

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

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

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

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

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

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

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

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 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

2. **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)

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

- GainRatio(A) = Gain(A)/SplitInfo(A)
- Example:

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

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

3. **Gini Index (CART, IBM IntelligentMiner)**

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

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

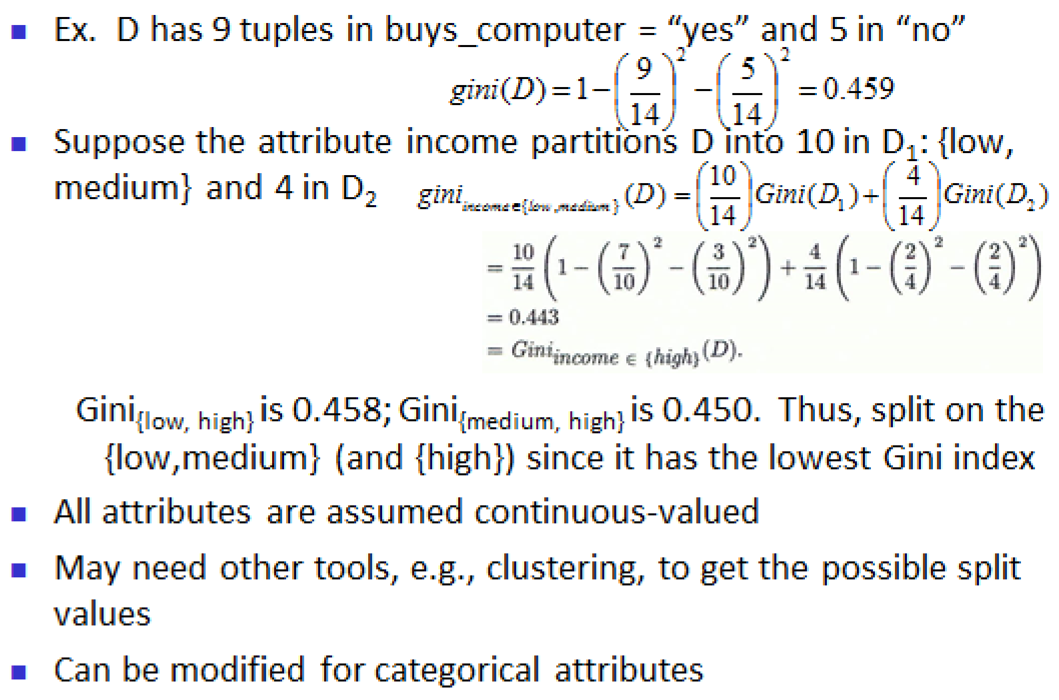
where $p_j$ is the relative frequency of class *j* in *D*

- If a data set $D$ is split on A into two subsets $D_1$ and $D_2$, the *gini* index *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) is chosen to split the node (*need to enumerate all the possible splitting points for each attribute*)

**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 $D_2$

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

$$= \frac{10}{14}\left(1 - \left(\frac{7}{10}\right)^2 - \left(\frac{3}{10}\right)^2\right) + \frac{4}{14}\left(1 - \left(\frac{2}{4}\right)^2 - \left(\frac{2}{4}\right)^2\right)$$

$$= 0.443$$

$$= Gini_{income \in \{high\}}(D).$$

$Gini_{\{low, high\}}$ 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

**Overfitting and Tree Pruning**

- Overfitting: An induced Tree may overfit the Learning 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 on 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 Learning 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

**Pros and Cons of decision Trees**

- Pros

Reasonable Learning time
Fast application
Easy to interpret
Easy to implement
Can handle large number of features

- Cons

Cannot handle complicated relationship between features
simple decision boundaries
problems with lots of missing data

## Methods for Comparing Classifiers

It examines some of the statistical tests available to compare the performance of different models and classifiers.

For illusLiative purposes, consider a pair of classification models, $M_A$ and $M_B$. Suppose $M_A$ achieves 85% accuracy when evaluated on a test set containing 30 records, while $M_B$ achieves 75% accuracy on a different test set containing 5000 records. Based on this information, is $M_A$ a better model than $M_B$?

The preceding example raises two key questions regarding the statistical significance of the performance meLiics:

1. Although $M_A$ has a higher accuracy than $M_B$, it was tested on a smaller test set. How much confidence can we place on the accuracy for Ma?
2. Is it possible to explain the difference in accuracy as a result of variations in the composition of the test sets?
The first question relates to the issue of estimating the confidence interval of a given model accuracy. The second question relates to the issue of testing the statistical significance of the observed deviation. These issues are investigated in the remainder of this section.

**Estimating a Confidence Interval for Accuracy**

To determine the confidence interval, we need to establish the probability distribution that governs the accuracy measure. This section describes an approach for deriving the confidence interval by modeling the classification task as a binomial experiment. Following is a list of characteristics of a binomial experiment:
1. The experiment consists of N independent Trials, where each Trial has two possible outcomes: success or failure.
2. The probability of success, p, in each Trial is constant.
An example of a binomial experiment is counting the number of heads that turn up when a coin is flipped N times. If X is the number of successes observed in N Trials, then the probability that X takes a particular value is given by a binomial distribution with mean Np and variance Np(l - p):

$$P(X = v) = \binom{N}{p} p^v (1-p)^{N-v}.$$

For example, if the coin is fair ($p = 0.5$) and is flipped fifty times, then the probability that the head shows up 20 times is

$$P(X = 20) = \binom{50}{20} 0.5^{20} (1-0.5)^{30} = 0.0419.$$

If the experiment is repeated many times, then the average number of heads expected to show up is $50 \times 0.5 = 25$, while its variance is $50 \times 0.5 \times 0.5 = 12.5$.

**Comparing the Performance of Two Models**

Consider a pair of models, $M_1$ and $M_2$ that are evaluated on two independent test sets, D1 and D2. Let n1 denote the number of records in D1 and n2 denote the number of records in D2. In addition, suppose the error rate for M1 on Dl is $e_l$ and the error rate for $M_2$ on $D_2$ is $e_2$. Our goal is to test whether the observed difference between e1 and e2 is statistically significant.
Assuming that n1 and n2 are sufficiently large, the error rates e1 and e2 can be approximated using normal distributions. If the observed difference in the error rate is denoted as d : $e_1$ - $e_2$, then d is also normally distributed with mean d1,its true difference.
The variance of d can be computed as follows:

$$\sigma_d^2 \simeq \hat{\sigma}_d^2 = \frac{e_1(1-e_1)}{n_1} + \frac{e_2(1-e_2)}{n_2},$$

Where $e_1(1- e_1)/n_1$ and $e_2(1 – e_2)/n_2$ are the variances of the error rates.

**Comparing the Performance of Two Classifiers**

Suppose we want to compare the performance of two classifiers using the k-fold cross-validation approach. Initially, the data set D is divided into k equal-sized partitions. We then apply each classifier to consLiuct a model from k - 1 of the partitions and test it on the remaining partition. This step is repeated k times, each time using a different partition as the test set.

Let $M_{ij}$ denote the model induced by classification technique $L_i$; during the jth iteration. Note that each pair of models $M_{1j}$ and, $M_{2j}$ are tested on the same partition j. Let $e_{1j}$ and $e_{2j}$ be their respective error rates. The difference between their error rates during the jth fold can be written as $d_j= e_{1j} - e_{2j}$. If k is sufficiently large, then $d_j$ is normally distributed with mean $d_t^{cv}$, which is the true difference in their error rates, and variance $\sigma^{cv}$. Unlike the previous approach, the overall variance in the observed differences is estimated using the following formula:

$$\hat{\sigma}^2_{d^{cv}} = \frac{\sum_{j=1}^{k}(d_j - \overline{d})^2}{k(k-1)},$$

where $\overline{d}$ is the average difference.

**Rule Based Classifiers: 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 and rule consequent
- Assessment of a rule: *coverage* and *accuracy*
  - $n_{covers}$ = # of tuples covered by R
  - $n_{correct}$ = # of tuples correctly classified by R

coverage(R) = $n_{covers}$ /|D| /* D: training data set */

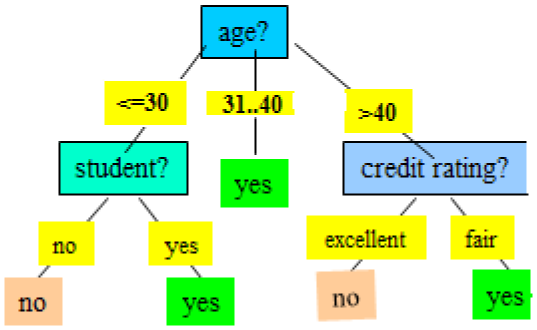
accuracy(R) = $n_{correct}$ / $n_{covers}$

- 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 Training Tuples: 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:**

- *Rule set* D={}; // initial set of rules learned is empty
- **for each** class *c* **do**
- **repeat**
- Rule D **Learn One Rule**(*D*, *Att vals*, *c*);
- remove tuples covered by *Rule* from *D*;
- *Rule set* D *Rule set* C*Rule*; // add new rule to rule set
- **until** terminating condition;
- **endfor**
- return *Rule Set* ;

We start off with an empty rule and then gradually keep appending attribute tests to it. We append by adding the attribute test as a logical conjunct to the existing condition of the rule antecedent. Suppose our training set, *D*, consists of loan application data. Attributes regarding each applicant include their age, income, education level, residence, credit rating, and the term of the loan. The classifying attribute is *loan decision*, which indicates whether a loan is accepted (considered safe) or rejected (considered risky). To learn a rule for the class "accept," we start off with the most general rule possible, that is, the condition of the rule antecedent is empty. The rule is IF …… THEN *loan_decision=accept*.
We consider each possible attribute test that may be added to the rule. These can be derived from the parameter *Att_vals*, which contains a list of attributes with their associated values. For example, for an attribute–value pair (*att*, *val)*, we can consider attribute tests such as *att* =*val*, *att* <*val*, *att* > *val*, and so on. Typically, the training data will contain many attributes, each of which may have several possible values. Finding an optimal rule set becomes computationally explosive. Instead, *Learn_One_Rule* adopts a greedy depth-first strategy. Each time it is faced with adding a new attribute test (conjunct) to the current rule, it picks the one that most improves the rule quality, based on the training samples.

**Rule-Quality measures: consider both coverage and accuracy**

- Foil-gain (in FOIL & RIPPER): assesses info_gain by extending condition

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

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

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

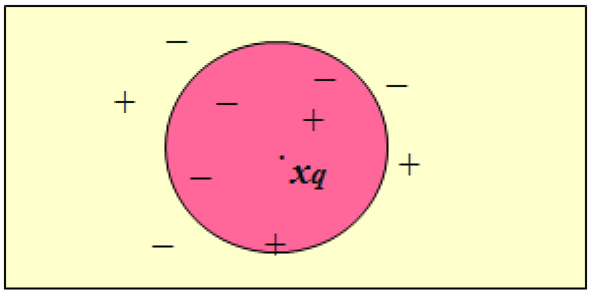
**Nearest Neighbor Classifiers**

- Instance-based learning:
  - Store training examples and delay the processing ("lazy evaluation") until a new instance must be classified
- Typical approaches
  - *k*-nearest neighbor approach
    - Instances represented as points in a Euclidean space.

**The *k*-Nearest Neighbor Algorithm**

1. Load the data
2. Initialize the value of k
3. For getting the predicted class, iterate from 1 to total number of training data points
   1. Calculate the distance between test data and each row of training data. Here we will use Euclidean distance as our distance metric since it's the most popular method. The other metrics that can be used are Chebyshev, cosine, etc.
   2. Sort the calculated distances in ascending order based on distance values
   3. Get top k rows from the sorted array
   4. Get the most frequent class of these rows
   5. Return the predicted class

- All instances correspond to points in the n-D space.
- The nearest neighbor are defined in terms of Euclidean distance.
- The target function could be discrete- or real- valued.
- For discrete-valued function, the *k*-NN returns the most common value among the k training examples nearest to *xq*.
- Vonoroi diagram: the decision surface induced by 1-NN for a typical set of training examples.



- Distance-weighted nearest neighbor algorithm

- Weight the contribution of each of the k neighbors according to their distance to the query point $x_q$
    - give greater weight to closer neighbors
- Similarly, for real-valued target functions

$$w \equiv \frac{1}{d(x_q, x_i)^2}$$

- Robust to noisy data by averaging k-nearest neighbors
- Curse of dimensionality: distance between neighbors could be dominated by irrelevant attributes: To overcome it, axes stretch or elimination of the least relevant attributes.
- **Example: Class example**

**Bayesian Classifiers**

- Bayesian classifiers are statistical classifiers
- For each new sample they provide a probability that the sample belongs to a class (for all classes)

**Bayes' Theorem: Basics**

- 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**), 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) (*posteriori probability*), 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
- 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 written as
    - posteriori = likelihood x prior/evidence
- Predicts **X** belongs to $C_2$ iff the probability $P(C_i|\mathbf{X})$ is the highest among all the $P(C_k|X)$ for all the *k* classes
- Practical difficulty: require initial knowledge of many probabilities, significant computational cost

**Towards Naïve Bayesian Classifiers:** 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 $P(C_i|\mathbf{X})$
- This can be derived from Bayes' theorem

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

- Since P(X) is constant for all classes, only needs to be maximized

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

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

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

- This greatly reduces the computation cost: Only counts the 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

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

- and $P(x_k|C_i)$ is $P(\mathbf{X}|C_i) = g(x_k, \mu_{C_i}, \sigma_{C_i})$

**Example: Class example**