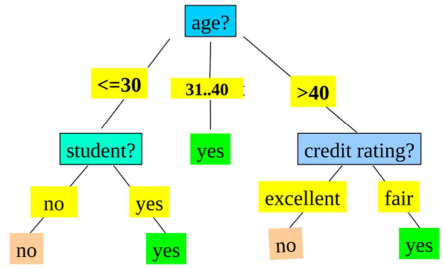
<table>
<tr><td rowspan="21">Classification</td><td colspan="4">Predicting discrete and nominal categorical class labels
Training set and class labels are used to classify data attributes (constructs a model), which can then be used to classify new data</td></tr>
<tr><td rowspan="3">Model Construction</td><td colspan="3">Describing set of predetermined classes</td></tr>
<tr><td>Assumption</td><td colspan="2">Each tuple belongs to a predefined class, determined by class label attribute</td></tr>
<tr><td>Model Representation</td><td colspan="2">Classification rules
Decision tree
Mathematical formula</td></tr>
<tr><td rowspan="4">Model Usage</td><td colspan="3">For classifying unknown objects</td></tr>
<tr><td rowspan="3">Estimation Accuracy</td><td colspan="2">Known label of test samples compared with model's classified results</td></tr>
<tr><td>Test set</td><td>Independent of training set, otherwise over-fitting will occur</td></tr>
<tr><td>Accuracy Rate</td><td>Accuracy rate = % of test set samples correctly classified</td></tr>
<tr><td rowspan="2">Supervision</td><td>Supervised Learning (Classification)</td><td colspan="2">- Training data accompanied by labels indicating class of observations
- New data classified based on training set</td></tr>
<tr><td>Unsupervised Learning (Clustering)</td><td colspan="2">- Class labels of training data unknown
- Try to extract classes or clusters in a set of training data</td></tr>
<tr><td rowspan="11">Issues</td><td rowspan="3">Data Preparation</td><td>Cleaning</td><td>Reduce noise and handle missing values</td></tr>
<tr><td>Relevance Analysis</td><td>Remove irrelevant or redundant attributes</td></tr>
<tr><td>Data transformation</td><td>Generalise or normalise data</td></tr>
<tr><td rowspan="8">Evaluating Classification Methods</td><td rowspan="2">Accuracy</td><td>Classifier Accuracy | Predicting class labels</td></tr>
<tr><td>Predictor Accuracy | Guessing value of predicted attributes</td></tr>
<tr><td>Speed</td><td>Training Time
Model construction time</td></tr>
<tr><td>Classification/Prediction Time</td><td>Time to use model</td></tr>
<tr><td>Robustness</td><td>Handles noise and missing values</td></tr>
<tr><td>Scalability</td><td>Efficiency on disk-resident databases</td></tr>
<tr><td>Interpretability</td><td>Understanding provided by model</td></tr>
</table>

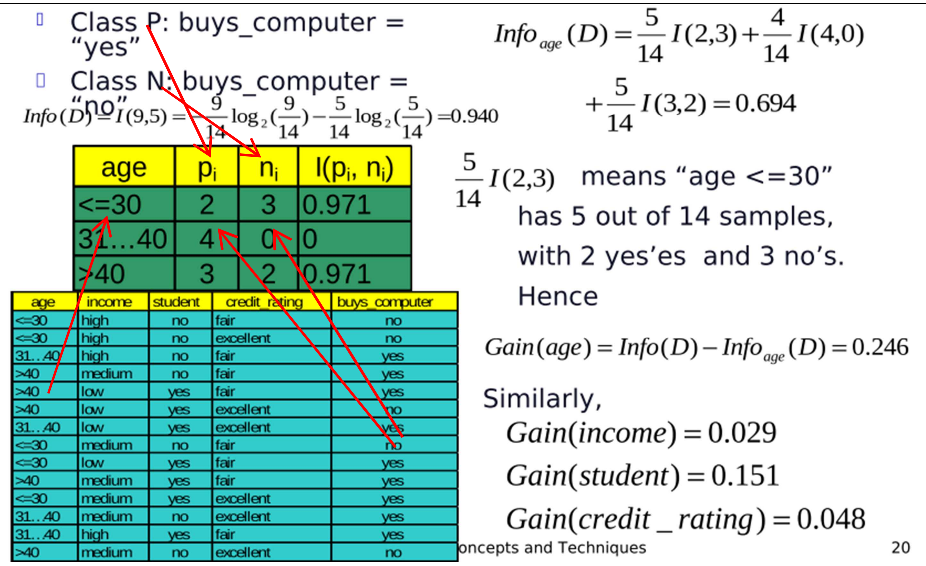
<table>
<tr><td rowspan="6">Decision Trees</td><td colspan="3">
</td></tr>
<tr><td>Benefits</td><td colspan="2">- relatively fast learning speeds
- convertible to simple classification rules
- can use SQL
- comparable classification accuracy</td></tr>
<tr><td>Greedy Algorithm</td><td colspan="2">Top-down recursive drive-and-conquer method

1) All training examples are at the root
2) Attributes categorical (continuous-valued are discretised in advance)
3) Examples partitioned recursively based on selected attributes
4) Test attributes selected based on heuristic of statistical measure (information gain)
5) Stop when either:
- all samples on given node belong to same class
- no more remaining attributes for further partitioning
- no samples are left</td></tr>
<tr><td rowspan="3">Basic Information Theory</td><td colspan="2">Describing a situation using probabilities introduces uncertainty</td></tr>
<tr><td>Entropy</td><td>Measure of uncertainty</td></tr>
<tr><td>Shannon Entropy</td><td>Mathematical entity, U[X], takes a probability distribution as input and outputs a measure of uncertainty as a quantity.

Let X be random variable; The probability distribution of X is, X = {Pr(1), Pr(2), ... , Pr(N)}
$$U[X] = -\sum_{x \in X} \Pr(x)\log_2(\Pr(x))$$</td></tr>
</table>

			Properties of U[X]	- must be maximised by a uniform distribution (which corresponds to complete uncertainty) - U[X] should be a continuous function of probabilities
	Attribute Selection Measures	**Information Gain (ID3)**	Biased towards multivalued attributes 1) **Select attribute with highest information gain** 2) Let **p_i be probability that an arbitrary tuple in D, belongs to C_i**, estimated by: $\frac{\lvert C_{i,D} \rvert}{\lvert D \rvert}$	
			Expected Information (Entropy)	Needed to classify a tuple D: $$Info(D) = -\sum_{i=1}^{m} p_i \log_2(p_i)$$
			Information	After using A to split D into v partitions, needed to classify D: $$Info_A(D) = \sum_{j=1}^{v} \frac{\lvert D_j \rvert}{\lvert D \rvert} \times I(D_j)$$
			Information Gained	By branching on attribute A $$Gain(A) = Info(D) - Info_A(D)$$

Example



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

1) Let class P be those who buy computers and class N be those who don’t
2) Decide to categorise/divide laptop sales in terms of age of customer
3) Tally the total number of people who do and don’t buy laptops within each age category
4) Calculate the information (Entropy) of the initial, uncategorised data set using

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

Which becomes, for this example:

$$Info(D) = -p_p \log_2(p_p) - p_n \log_2(p_n)$$

Where p_p is the probability of a randomly selected record being one where a computer was purchased and p_n being the probability that it was a record where a computer was **not** purchased.

				$Info(D) = I(9,5) = -\frac{9}{14}\log_2\left(\frac{9}{14}\right) - \frac{5}{14}\log_2\left(\frac{5}{14}\right) = 0.940$ 5) Calculate the new information (Entropy) of each category. For example, the age bracket <= 30 has 2 purchases and 3 non-purchases, thus: $Info(D) = -p_p \log_2(p_p) - p_n \log_2(p_n)$ $Info(D) = I(2,3) = -\frac{2}{3}\log_2\left(\frac{2}{3}\right) - \frac{1}{3}\log_2\left(\frac{1}{3}\right) = 0.971$ Similarly, for age bracket *31…40* I**(4,0) = 0** and for *>40* **I(3,2) = 0.971.** 6) Calculate the new total entropy using a weighted sum of the individual entropies of each category, based on the relative size of that category: $Info_A(D) = \sum_{j=1}^{v} \frac{\lvert D_j \rvert}{\lvert D \rvert} \times I(D_j)$ $Info_{age}(D) = \frac{5}{14}I(2,3) + \frac{4}{14}I(4,0) + \frac{5}{14}I(3,2) = 0.698$ 7) Calculate the information gain: $Gain(age) = Info(D) - Info_{age}(D) = 0.246$	
				Split Points	Determining the best split point for a continuous-valued attribute 1) Sort into increasing order - each midpoint between adjacent values is possible split point 2) Point with minimum expected information requirement for A is selected as the split-point
		Gain Ratio (C4.5)	Tends to prefer unbalanced splits where one partition is much smaller than the others		

			- **C4.5 (successor of ID3) uses gain ratio (normalisation to information gain)** - attribute with **max gain ratio is selected as splitting attribute** $$SplitInfo_A(D) = -\sum_{j=1}^{v} \frac{\lvert D_j \rvert}{\lvert D \rvert} \times \log_2(\frac{\lvert D_j \rvert}{\lvert D \rvert})$$ $$Gain\ Ratio(A) = \frac{Gain(A)}{SplitInfo(A)}$$	
		Gini Index	- Biased to multivalued attributes - Has difficulty with large number of classes - Favours tests that result in equal sized partitions and purity in both partitions $$gini(D) = 1 - \sum_{j=1}^{n} p_j^2$$ Where: - D is data set - n is number of classes that exist - p_j relative frequency of class j in D If D is split on attribute A, into two subsets D_1 and D_2 then: $$gini_A(D) = \frac{\lvert D_1 \rvert}{\lvert D \rvert} gini(D_1) + \frac{\lvert D_2 \rvert}{\lvert D \rvert} gini(D_2)$$ Reduction in impurity: $$\Delta gini(A) = gini(D) - gini_A(D)$$ Attribute that provides smallest $gini_{split}(D)$ (or largest reduction in impurity) is chosen - need to enumerate all possible splitting points for each attribute	
			Example	Let there be 9 tuples in buys_computers = “yes” and 5 in “no” 1) Calculate gini index:

				$gini(D) = 1 - \sum_{j=1}^{n} p_j^2$ $gini(D) = 1 - \left(\frac{9}{14}\right)^2 - \left(\frac{5}{14}\right)^2 = 0.459$ 2) Suppose the income attribute divides C into 10 in D_1 and 4 in D_2: {low, medium}. $gini_{income(low,medium)}(D) = \left(\frac{10}{14}\right) gini(D_1) + \left(\frac{4}{14}\right) gini(D_2) = 0.45$ Similarly, $\text{gini}_{(\text{medium,high})}$ is 0.30 and is thus the best division NB: - Attributes are assumed continuous-valued - Other tools may be required to get possible split values (clustering, etc.) - Can be modified for categorical attributes
		Others	- 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-statistics: 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.	
	Overfitting	**Induced tree may** overfit training data and **create too many branches where some reflect anomalies due to noise** or outliers which then **end up with poor accuracy for unseen samples**		
		Avoiding overfitting	**Prepruning**	**Halt tree execution early** and **stop** from splitting a node if it would result in the **goodness measure falling below threshold**
			Postpruning	**Remove branches from fully realised tree** to get s**equence of progressively pruned trees** and **use set of data** on each to **identify optimally pruned tree**.
	Enhancements	**Allow for continuous-**		**Dynamically define new discrete-valued attributes to partition** into discrete set of

		valued attributes	intervals
		Handle missing attributes	Assign **most common value**, **or** a **probability to each possible value**
		Attribute construction	Create new **attributes based on existing ones sparsely represented**, to **reduce fragmentation, repetition and replication**
	Large Databases	Classification	studied by statisticians and machine learning researchers
		Scalability	Classifying data sets with millions of examples and hundreds of attributes with reasonable speed
		Bootstrapped Optimistic Algorithm (BOAT)	- Uses bootstrapping to create small samples (subsets) that each fit in memory - Tree created from each subset - New tree, T' constructed from several smaller trees, which is very close to tree that would have been generated with whole data set Advantages - Requires only 2 scan of DB - Incremental algorithm
Bayesian Classification	Characteristics	Statistical Classifier	Predicts **class membership probabilities**
		Foundation	Based on **Bayes' Theorem**
		Performance	Simple Bayesian classifier has **comparable performance with decision tree** & selected neural network classifiers
		Incremental	Each training example can **incrementally increase/decrease probability hypothesis is correct** (**prior knowledge combined with observed data**)
		Standard	Provide **standard of optimal decision making** to measure other methods against
	Definition	Let **X** be **sample data** where **class label is unknown** Let **H** be **hypothesis that X belongs to C** **P(H|X)** is **classification** **P(H)** ***prior probability*** (eg. X will by computer regardless of age, income, etc.) **P(X) probability sample data is observed** (eg. a record with age, income, etc is between range or category) **P(X|H)** ***posteriori probability***, **probability of observing sample X, given hypothesis holds**. $P(H\|X) = \frac{P(X\|H)P(H)}{P(X)}$	

		Requires initial knowledge of many probabilities, which comes at computational cost	
	Naive Bayesian Classifier	Let **D** be **training set of tuples** as associate class labels; **each tuple represented by n-D attribute vector X = (x_1,x_2, …, x_n)** **Suppose** there are **m classes C_1, C_2, …, C_m**. **Classification** is to **derive max posteriori, P(C_i|X).** Derived from Bayes' theorem: $P(C_i \mid X) = \frac{P(X \mid C_i)P(C_i)}{P(X)}$ P(X) is const. for all classes: $\boldsymbol{P(C_i \mid X) = P(X \mid C_i)P(C_i)}$ **Needs to be maximised.** **Simplifying assumption:** attributes are conditionally independent $P(X \mid C_i) = \prod_{k=1}^{m} P(x_k \mid C_i)$ Reduced computational cost because only counts class distribution. If A_k is categorical, P(x_k|C_i) is number of tuples in C_i having value x_k for A_k divided by |$C_{i,D}$|. if A_k is continuous-values, P(x_k|C_i) usually computed based on Gaussian distribution with mean μ and standard dev. Σ $g(x,\mu,\sigma) = \frac{1}{\sqrt{2\pi\sigma}} e^{\frac{(x-\mu)^2}{2\sigma^2}}$ $P(X \mid C_i) = g(x_k, \mu_{C_i}, \sigma_{C_i})$	
		Advantages	- Easy to implement - Good results from most cases
		Disadvantages	- Assumption class conditional independence results in loss of accuracy

Example

Class:
C1:buys_computer = 'yes'
C2:buys_computer = 'no'

Data sample
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

Using Naive Bayesian Classifiers, find out which category (buys computer, doesn't buy), X = (age <= 30, income = medium, student = yes, credit_rating = fair) belongs to:

1) Trying to find which category has the greatest probability of containing X. Or, to maximise

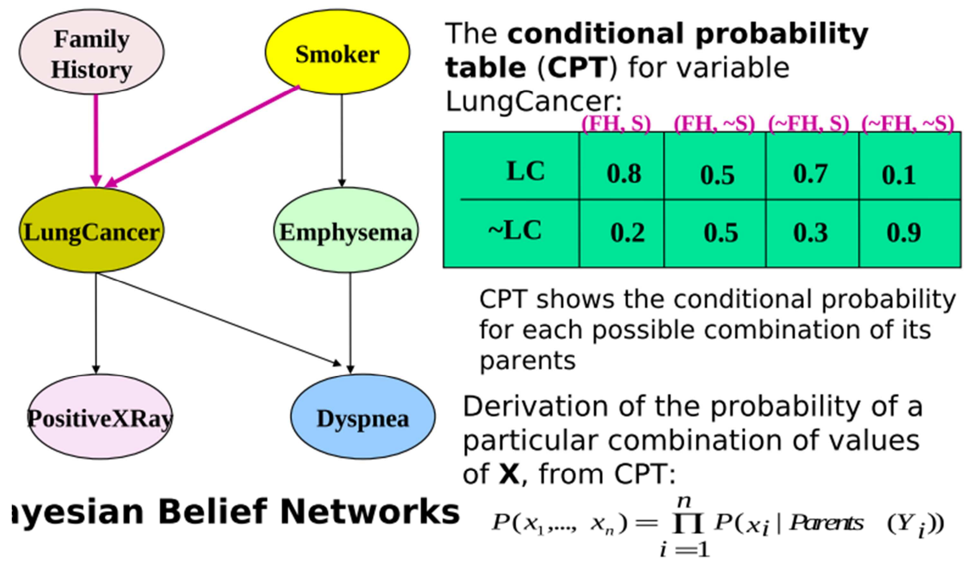
$$P(C_i|X) = P(X|C_i)P(C_i)$$

2) Calculate $P(C_i)$, for each category:
   P(buys_computer = "yes") = 9/14 = 0.643
   P(buys_computer = "no") = 5/14 = 0.357

3) Calculate $P(X|C_i)$ for each class:
   P(age = "<=30" | buys_computer = "yes") = 2/9 = 0.222
   P(age = "<=30" | buys_computer = "no") = 3/5 = 0.6
   P(income = "medium" | buys_computer = "yes") = 4/9 = 0.444
   P(income = "medium" | buys_computer = "no") = 2/5 = 0.4
   P(student = "yes" | buys_computer = "yes") = 6/9 = 0.667
   P(student = "yes" | buys_computer = "no") = 1/5 = 0.2

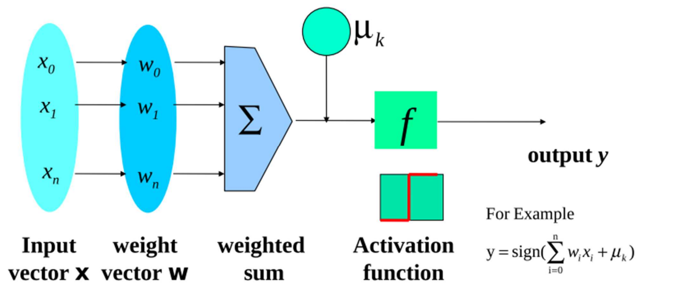
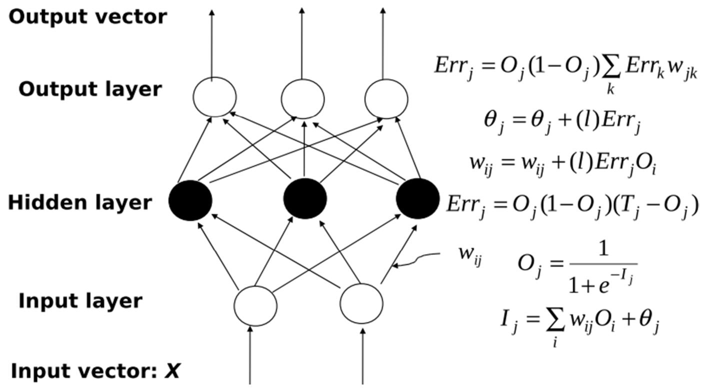
   P(credit_rating = "fair" | buys_computer = "yes") = 6/9 = 0.667
   P(credit_rating = "fair" | buys_computer = "non") = 2/5 = 0.4

<table>
<tr>
<td rowspan="7"></td>
<td rowspan="3"></td>
<td></td>
<td colspan="3">4) Calculate $P(C_i|X) = P(X|C_i)P(C_i)$ for each category:
$$P(X|C_i) = \prod_{k=1}^{m} P(x_k|C_i)$$
$$P(X|C_{buys-computer=yes}) = 0.222 \times 0.444 \times 0.667 \times 0.667 = 0.044$$
$$P(X|C_{buys-computer=no}) = 0.6 \times 0.4 \times 0.2 \times 0.4 = 0.019$$
$$P(C_i|X) = P(X|C_i)P(C_i)$$
$$P(C_{buys-computer=yes}|X) = 0.044 \times 0.643 = 0.028$$
$$P(C_{buys-computer=no}|X) = 0.019 \times 0.357 = 0.007$$
X belongs to buys_computer=yes because 0.028 > 0.007</td>
</tr>
<tr>
<td rowspan="2">Avoiding 0-Problem</td>
<td colspan="3">Naïve Bayesian prediction requires each conditional probability be non-zero, else predicted probability will be zero.</td>
</tr>
<tr>
<td>Laplacian Correction/Estimator</td>
<td colspan="2">When there is a case where there is 0 members of a class, add 1 to each case to get close estimates of actual probabilities</td>
</tr>
<tr>
<td rowspan="4">Bayesian Belief Networks</td>
<td colspan="4">- Allows a subset of variables conditionally independent
- Graphical model of causal relationships that represents dependency among variables
- Gives specification of joint probability distribution
- Nodes: random variables
- Links: dependencies
- No loops or circles</td>
</tr>
<tr>
<td>Conditional Probability Table (CPT)</td>
<td colspan="3">Shows conditional probability for each possible combination of parents</td>
</tr>
<tr>
<td>Training</td>
<td>Network Structure Know</td>
<td>Variables Observable</td>
<td>Algorithms</td>
</tr>
</table>

			Yes	All	Learn only CPTs
			Yes	Some Hidden	Gradient descent (greedy hill-climbing) method, analogous to neural network learning
			No	All	search through model space to reconstruct network topology
			No	Hidden	No good algorithms known for this
		Example			

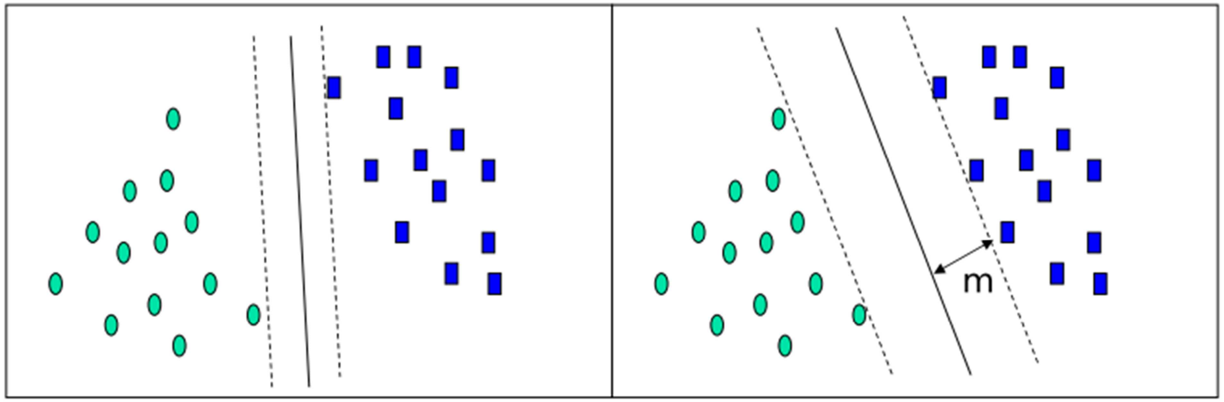
Rule-based Classification	IF age = youth AND student = yes THEN buys_computer = yes		
	Antecedent/precondition	IF age = youth AND student = yes	
	Consequent	buys_computer = yes	
	Coverage	**tuples covered by R/ size of training set**	
	Accuracy	**tuples correctly classified by R/ tuples covered by R**	
	Conflict Resolution	**Size Ordering**	Assign **highest priority to** triggering rules with **toughest requirements** (most attribute tests)
		Class-based Ordering	**Decrease order of prevalent or misclassification cost per class**
		Rule-based ordering	Rules organised into **priority list**

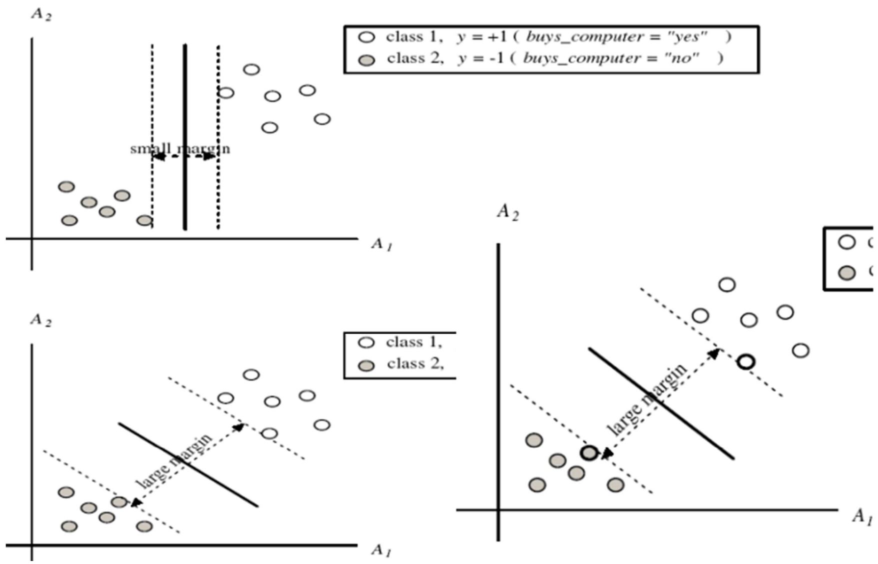
		(Decision List)
	Training	1) Rules learned one-at-a-time 2) Each time a rule is learned, tuples covered by rules are removed 3) Repeats until termination condition reached (no more tuples or user threshold)
Linear Classification	Data divided by a line – every point on one side of the line is in the same category.	
Discriminative Classifiers	Advantages	- prediction accuracy generally high (compared to Bayesian methods) - robust (works when training examples contain errors) - fast evaluation of learned target function
	Disadvantages	- Long training time - Difficult to understand learned function (weights) - Difficult to incorporate domain knowledge
Classification by **Backpropagation**	A **neural network learning algorithm** started by psychologists and neurobiologists to develop and test computational analogues of neurons.	
	Neural Network (Connectionist Learning)	Set of **connected input/output units where each connection is weighted** During **learning phase**, network **learns by adjusting weightings** to be able **to predict correct class** label
	Advantages	- High **tolerance to noisy data** - Ability to **classify untrained patterns** - **Well-suited** for **continuous-valued inputs and outputs** - Successful on a wide array of real-world data - Algorithms are **inherently parallel** - Techniques have recently been developed for extraction of rules from trained neural networks
	Disadvantages	- **Long training time** - **Requires** number of **parameters best determined empirically** (network topology, etc.) - **Difficult to interpret symbolic meaning** behind learned weights and of hidden units

<table>
<tr><td></td><td>Neuron</td><td colspan="2">
</td></tr>
<tr><td></td><td>Multi-Layer Feed-Forward Neural Network</td><td colspan="2">

- Inputs to network are attributes measured for each training tuple and are simultaneously fed into input layer.
- Then weighted and fed simultaneously into hidden layer (number of which is arbitrary, although usually one)
- Weighted outputs of hidden layer are fed to output layer and network predictions are made.
- Network is feed-forward, i.e. none of the weights are cycled back into input or output of previous layer
- Networks perform nonlinear regression
- With enough training samples & time, can closely approximate any function</td></tr>
<tr><td></td><td></td><td>Defining Network Topology</td><td>1) Decide number of:
o units in input layer
o hidden layers
o units in output layer</td></tr>
</table>

			2) Normalise input values for each attribute measured in training tuples to [0.0 - 1.0] 3) Assign 1 input unit per domain value, initialised to 0 4) Assign 1 output unit per class (for classification with more than 2 classes) 5) Train network and achieve acceptable accuracy. 6) Repeat training with different network topology or set of initial weights
	Backpropagation	- **Iteratively process training tuples** and **compare network's predictions with actual** known target value - For each training tuple, **weights modified to achieve minimum mean squared erro**r between network's prediction and actual target value - **Modifications made backwards**: **from outer layer, though each hidden layer down to first hidden layer**. 1) **Initialise weights** and bias (**small random numbers**) in the network 2) **Propagate inputs forwards** (apply activation function) 3) **Backpropagate the error by updating weights and biases** 4) **Terminate when error becomes small**	
	Efficiency	**Each epoch** (single **iteration through training set**) **take O(|D|*w)**, with |D| tuples and w weights. - **number of epochs can be exponential to n,** number of inputs (in **worst case**)	
	Rule Extraction	1) **Simplify network structure** by **removing weighted links** that have **least effect on trained network** 2) Perform **link, unit or activation value clustering** 3) Set of **input and activation values studied to derive rules for relationship between input and hidden unit layers**	
	Sensitivity Analysis	**Assessment of impact given each variable has on network output**. Knowledge gained represented in rules.	
Support Vector Machines	- **Classification method** for **linear and non-linear data** - Uses **nonlinear mapping to transform original training data into higher dimension** - With new dimension, **searches for linear optimal separating hyperplane** (decision boundary) - Using appropriate nonlinear mapping to sufficiently high dimension, **data from two classes can always be separated by hyperplane**, which **SVM finds using support vectors ("essential" training tuples)** and **margins (defined by support vectors)**		
	Advantages	- **High accuracy** to **ability to model complex nonlinear decision boundaries** (margin maximisation) - Useful for **classification and prediction**	
	Disadvantages	- **Training** can be **slow**	
	Applications	- Handwritten digit recognition - Object recognition - Speaker identification	

		- Benchmarking time-series prediction tests
	Linearly Separable	 Let **data D be (X_1, y_1), … , ($X_{\|D\|}$, $y_{\|D\|}$)** where **X_i is set of training tuples associated with class labels y_i.** Interested in the best hyperplane, or line that separates two classes (one that minimizes classification error on unseen data) out of the infinite hyperplanes that exist. SVM searches for hyperplane with largest margin (**maximum marginal hyperplane MMH**).



Hyperplane can be written a

$$\boldsymbol{w} \circ \boldsymbol{x} + \boldsymbol{b} = \boldsymbol{0}$$

Where **w = {w_1,w_2, …, w_n} is weight vector** and **b is scalar (bias)**

For 2-D, can be simplified as:

$$w_0 + w_1x_1 + w_2w_2 = 0$$

With hyperplane defining sides of margin:

$$H_1\colon w_0 + w_1x_1 + w_2x_2 \geq 1 \;\; for\; y_i = \; +1$$
$$H_2\colon w_0 + w_1x_1 + w_2x_2 \leq \; -1\, for\; y_i - 1$$

Becomes a constrained (convex) quadratic optimisation problem.

- *Quadratic objective function & linear constraints → Quadratic Programming (QP) → Lagrangian Multipliers*

Support Vectors	Any training tuples that fall on hyperplane H_1 or H_2 (sides of the defining margin)
High Dimensional	- **Trained classifier complexity characterised by number of support vectors**, **not dimensionality of data**

<table>
<tr><td rowspan="5"></td><td></td><td>Data</td><td>

Support vectors are essential or critical training examples (would produce same hyperplane if all other training data was removed)
Number of support vectors found can be used to compute upper bound on expected error rate of SVM classifier
SVM with few support vectors can have good generalisation even with high dimensional data

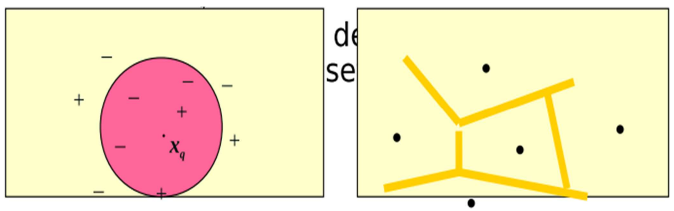
</td></tr>
<tr><td>Linearly Inseparable</td><td colspan="2">
1) Transform original input data into higher dimensional space
2) Search for a linear separating hyperplane in new space
<table>
<tr><td>Example</td><td>A 3D input vector $X = (x_1, x_2, x_3)$ is mapped into a 6D space Z using mappings $\Phi_1(X) = x_1, \Phi_2(X) = x_2, \Phi_3(X) = x_3, \Phi_4(X) = (x_1)^2, \Phi_5(X) = x_1 x_2, \Phi_6(X) = x_1 x_3$.

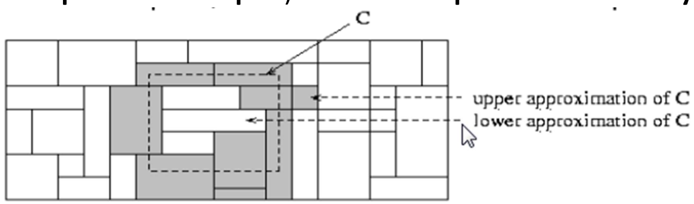
A decision hyperplane in the new space is $d(Z) = WZ + b$ where W and Z are vectors. This is linear. Solve for W and b and then substitute back so can see that linear decision hyperplane in new (Z) space corresponds to nonlinear second order polynomial in original 3-D space.
$$d(Z) = w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 (x_1)^2 + w_5 x_1 x_2 + w_6 x_1 x_3 + b$$
$$= w_1 z_1 + w_2 z_2 + w_3 z_3 + w_4 z_4 + w_5 z_5 + w_6 z_6 + b$$
</td></tr>
</table>
</td></tr>
<tr><td>Scaling SVM by Hierarchical Micro-Clustering</td><td colspan="2">
- SVM not scalable to number of data objects in terms of training tiem and memory usage
<table>
<tr><td>Clustering-Based SVM (CB-SVM)</td><td>

Optimise SVM accuracy and training speed with limited system resources
Use micro-clustering to reduce number of points to be considered
Derive support vectors by de-clustering micro-clusters near candidate vector for high classification accuracy

</td></tr>
</table>
</td></tr>
<tr><td>Comparison with Neural Networks</td><td colspan="2">
<table>
<tr><th>Property</th><th>SVM</th><th>Neural Network</th></tr>
<tr><td>Age</td><td>Relatively new</td><td>Relatively old</td></tr>
<tr><td>Algorithm</td><td>Deterministic</td><td>Nondeterministic</td></tr>
<tr><td>Generalisation</td><td>Nice generalisation properties</td><td>Generalises well, but doesn’t have strong mathematical foundation</td></tr>
<tr><td>Learning</td><td>Hard to learn – learned in batch mode using quadratic programming</td><td>Can easily be learned in incremental fashion
To learn complex functions, use multilayer perceptron</td></tr>
</table>
</td></tr>
<tr><td colspan="3"></td></tr>
<tr><td>Associative Classification</td><td colspan="3">Search for strong associations between frequent patterns (conjunctions of attribute-value pairs) and class labels</td></tr>
</table>

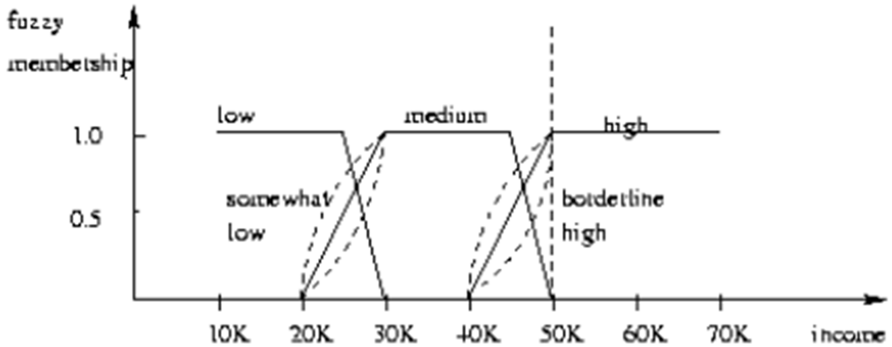
<table>
<tr><td></td><td colspan="3">Classification based on evaluating set of rules in form of $P_1 \wedge p_2 \ldots \wedge p_l$ --> "A_{class} = C"(conf,sup)

Explores highly confident associations among multiple attributes and may overcome some constraints introduced by decision-tree induction, which considers one attribute at a time</td></tr>
<tr><td rowspan="2">Lazy Learning (Instance-based learning)</td><td colspan="3">Stores training data (or with little processing) and waits until given test tuple
- less time spent training, but more time predicting
- Uses richer hypothesis space because uses many local linear functions to form implicit global approximations to target function</td></tr>
<tr><td>Approaches</td><td>K-nearest Neighbour Approach</td><td>- Instances represented as points in Euclidean n-D space
- Nearest neighbour is one with smallest dist(X_1, X_2)
- Target function can be discrete or real-valued
<table>
<tr><th>Inputs</th><th>Output</th></tr>
<tr><td>Discrete Valued</td><td>most common value among k training examples nearest X_q</td></tr>
<tr><td>Real-valued</td><td>Prediction for a given unknown tuple returns the mean values of the k nearest neighbours</td></tr>
</table>
<table>
<tr><td>Voronoi diagram</td><td>The decision surface induced by 1-NN for a typical set of training examples.
</td></tr>
<tr><td>Distance Weighted Nearest Neighbour Algorithm</td><td>Gives greater weight to closer neighbours</td></tr>
<tr><td>Curse of Dimensionality</td><td>Distance between neighbours could be dominated by irrelevant attributes
- Overcome it by stretching aces or elimination of least relevant attributes</td></tr>
</table></td></tr>
</table>

<table>
<tr><td rowspan="5"></td><td rowspan="5"></td><td>Locally Weighted Regression</td><td colspan="2">Constructs Local Approximation</td></tr>
<tr><td rowspan="4">Case-Based Reasoning</td><td colspan="2">Uses database of problem solutions to solve new one
- Stores symbolic description (tuples or cases) instead of points in Euclidean space</td></tr>
<tr><td>Applications</td><td>- Customer-service (product-related diagnosis)
- legal ruling</td></tr>
<tr><td>Methodology</td><td>1) Instances represented by rich symbolic descriptions (e.g. graph functions)
2) Search for similar cases; can combine multiple retrieved cases
3) Tight coupling between case retrieval, knowledge-based reasoning and problem solving</td></tr>
<tr><td>Disadvantages</td><td>- Difficult to find good similarity metric</td></tr>
<tr><td>Eager Learning</td><td colspan="4">Constructs classification model before receiving new test data to classify
- must commit to single hypothesis covering entire instance space</td></tr>
<tr><td>Genetic Algorithms (GA)</td><td colspan="4">- Based on analogy to biological evolution
- Initial population created with randomly generated rules, each represented by bit string
- Based on notion of survival of the fittest as represented by a rule's classification accuracy on a set of training examples
- Offspring generated by crossover and mutation
- Continues until population P evolves when each rule in P satisfies threshold
- Slow, but easily parallised</td></tr>
<tr><td>Rough Set Approach</td><td colspan="4">- Rough set for given class C approximated by lower approximation (certain to be in C) and upper approximation (cannot be described as not belonging to C)
- Find minimal subset (reducts) of attributes for feature reduction is NP-hard but a discernibility matrix (stores difference between attribute values for each pair of data tuples) reduces computational intensity
</td></tr>
<tr><td>Fuzzy Set Approaches</td><td colspan="4">- Fuzzy logic uses truth values between 0.0 and 1.0 to represent degree of membership (such as fuzzy membership graph)
- Attribute values converted to fuzzy values</td></tr>
</table>

<table>
<tr>
<td></td>
<td>

- **For given new sample, <1 fuzzy value may apply**
- Each **applicable rule contributes a vote for membership** in the category
- **Truth values typically summed** for each predicted category



</td>
</tr>
</table>