**Classification and Prediction**

1. **Classification:**

– predicts categorical (unqualified, unconditional) 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

1. **Prediction**

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

1. **Typical applications**

– Credit approval

– Target marketing

– Medical diagnosis

– Fraud detection

**Classification: Basic Concepts**

1. Supervised learning (classification)
   1. 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
2. Unsupervised learning (clustering)
   1. The class labels of training data is unknown
3. 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**

Data classification is a two-step process, consisting of a **learning step** (where a classification model is constructed) and a **classification step** (where the model is used to predict class labels for given data)

1. **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: training set

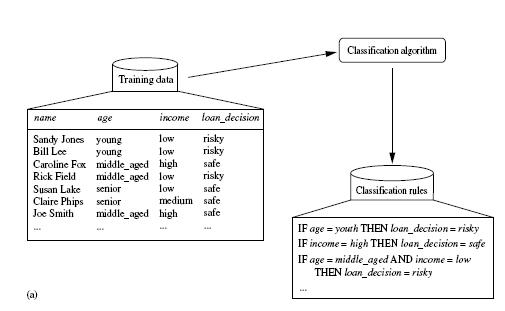
– The model is represented as classification rules, decision trees, or mathematical formulae

1. **Classification step**: for classifying future or unknown objects

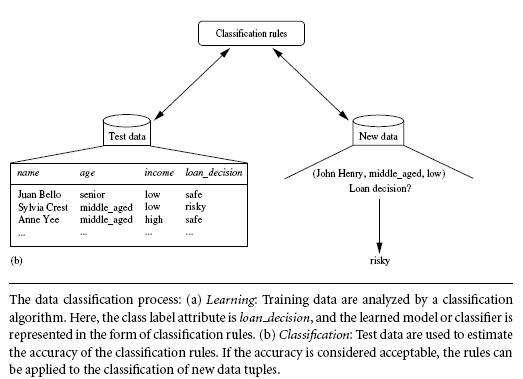
– Estimate accuracy of the model

* + 1. The known label of test sample is compared with the classified result from the model
    2. Accuracy rate is the percentage of test set samples that are correctly classified by the model
    3. Test set is independent of training set, otherwise over-fitting will occur

**Process (1): Learning Step**

****

**Process (2): Classification step**

****

**Issues Regarding Classification and Prediction**

1. **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).
2. **Relevance analysis**: Many of the attributes in the data may be*redundant*. Correlationanalysis can be used to identify whether any two given attributes are statistically related.
3. **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** − Accuracy of classifier refers to the ability of classifier. It predict the class label correctly and the accuracy of the predictor refers to how well a given predictor can guess the value of predicted attribute for a new data.
* **Speed** − This refers to the computational cost in generating and using the classifier or predictor.
* **Robustness** − It refers to the ability of classifier or predictor to make correct predictions from given noisy data.
* **Scalability** − Scalability refers to the ability to construct the classifier or predictor efficiently; given large amount of data.
* **Interpretability** − It refers to what extent the classifier or predictor understands.

**Classification by Decision Tree Induction**

**Decision tree**

1. A flow-chart-like tree structure
2. Internal node denotes (nonleaf node) denotes a test on an attribute
3. Branch represents an outcome of the test
4. Leaf nodes represent class labels or class distribution(Terminal node)
5. The topmost node in a tree is the root node.

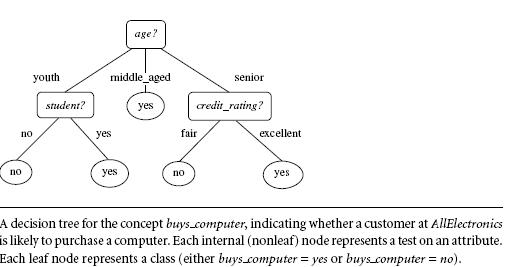
Decision tree generation consists of two phases

Tree construction

1. At start, all the training examples are at the root
2. Partition examples recursively based on selected attributes

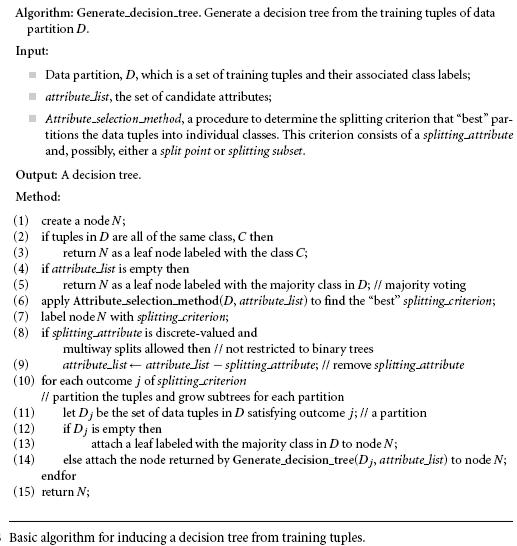
Tree pruning

1. Identify and remove branches that reflect noise or outliers



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.

Given a tuple,***X***, for which the associatedclass 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**

* The algorithm is called with three parameters: D, attribute list, and Attribute selection method.
* The parameter attribute list is a list of attributes describing the tuples.
* Attribute selection method specifies a heuristic procedure for selecting the attribute that “best” discriminates the given tuples according to class.
* This procedure employs an attribute selection measure such as information gain or the Gini index. Whether the tree is strictly binary is generally driven by the attribute selection measure.
* Some attribute selection measures, such as the Gini index, enforce the resulting tree to be binary.
* information gain, do not, therein allowing multiway splits (i.e., two or more branches to be grown from a node).

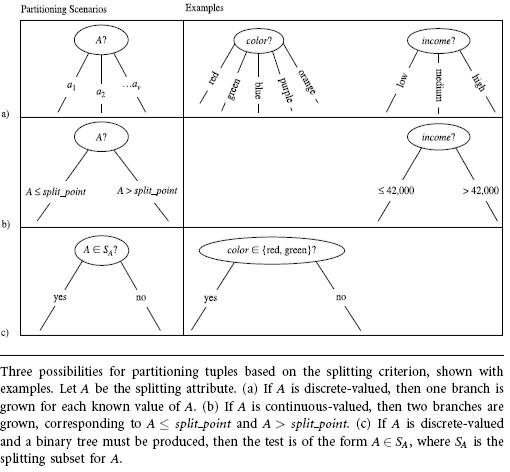
1. The tree starts as a single node, *N*, representing the training tuples in *D* (step 1)
2. If the tuples in *D* are all of the same class, then node *N* becomes a leaf and is labeled with that class (steps 2 and 3).

Note that steps 4 and 5 are terminating conditions.

1. 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 node *N* is labeled with the splitting criterion, which serves as a test at the node (step 7).

1. A branch is grown from node *N* for each of the outcomes of the splitting criterion.
2. The tuples in *D* are partitioned accordingly (steps 10 to 11).
3. There are three possible narios, as illustrated in Figure. Let *A* be the splitting attribute.
4. *A* has *v* distinct values, {*a*1, *a*2, : : : , *av}*, based on the training data.
5. A is discrete-valued: In this case, the outcomes of the test at node N correspond directly to the known values of A. A branch is created for each known value, aj , of A and labeled with that value (Figure a). Partition Dj is the subset of class-labeled tuples in D having value aj of A. Because all the tuples in a given partition have the same value for A, A need not be considered in any future partitioning of the tuples. Therefore, it is removed from attribute list
6. A is continuous-valued: In this case, the test at node N has two possible outcomes, corresponding to the conditions A ≤ split point and A > split point, respectively, where split point is the split-point returned by Attribute selection method as part of the splitting criterion.
7. A is discrete-valued and a binary tree must be produced (as dictated by the attribute selection measure or algorithm being used): The test at node N is of the form “A ∈ SA?,” where SA is the splitting subset for A, returned by Attribute selection method as part of the splitting criterion.



**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 criterionThis

section describes three popular attribute selection measures—***information gain, gain ratio*,** and ***gini* *index***

**Information gain:**ID3(Iterative Dichotomiser) uses information gain as its attribute selection measure.



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,

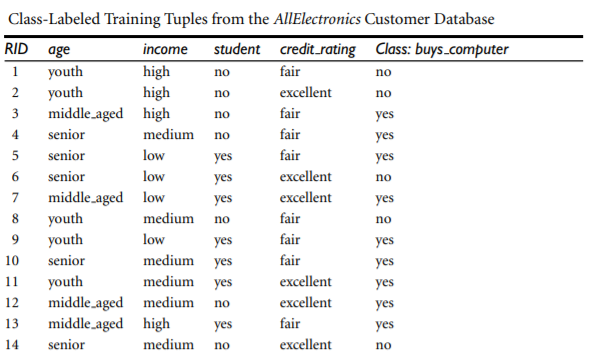


In other words, *Gain(A)* tells us how much would be gained by branching on *A*. It is the expected reduction in the information requirement caused by knowing the value of *A*. The attribute *A* with the highest information gain, (*Gain*(*A*)), is chosen as the splitting attribute at node *N*.

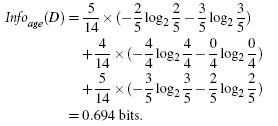
**Example** Induction of a decision tree using information gain.

Table presents a training set, *D*, of class-labeled tuples randomly selected from the *AllElectronics* customer database. 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*:





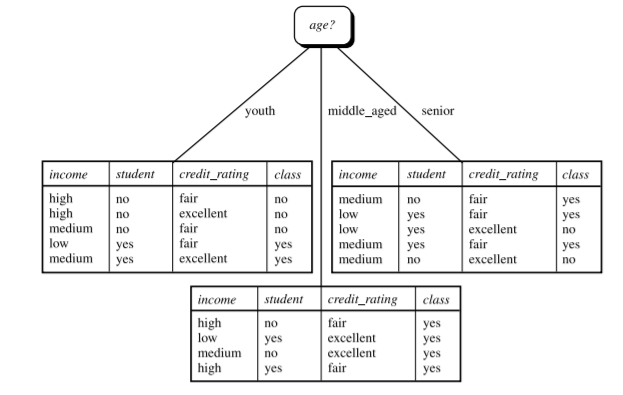
The expected information needed to classify a tuple in *D* if the tuples are partitioned according to *age* is



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

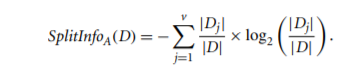


Similarly, we can compute *Gain*(*income*) = 0.029 bits, *Gain*(*student*) = 0.151 bits, and *Gain*(*credit rating*) = 0.048 bits. Because *age* has the highest information gain among the attributes,it is selected as the splitting attribute. Node *N* is labeled with *age*, and branches are grown for each of the attribute’s values. The tuples are then partitioned accordingly, as shown in Figure. 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



**Gain Ratio**

C4.5, a successor of ID3, uses an extension to information gain known as gain ratio.Information gain using a “split information” value defined analogously with Info(D) as

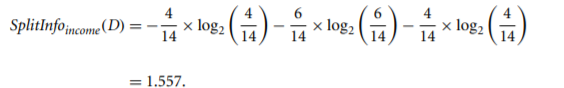


The gain ratio is defined as



Example:

Computation of gain ratio for the attribute income. A test on income splits the data of above Table into three partitions, namely low, medium, and high, containing four, six, and four tuples, respectively. To compute the gain ratio of income, we first use Eq. to obtain Split info



we have Gain(income) = 0.029. Therefore, GainRatio(income) = 0.029/1.557 = 0.019.

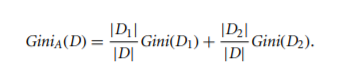
**Gini Index**

the Gini index measures the impurity of D, a data partition or set of training tuples, as



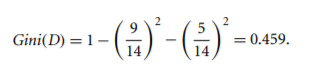
where pi is the probability that a tuple in D belongs to class Ci and is estimated by |C i,D|/|D|. The sum is computed over m classes.

if a binary split on A partitions D into D1 and D2, the Gini index of D given that partitioning is

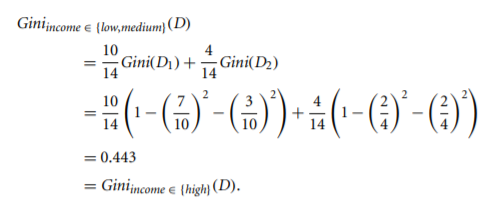


Example:

Induction of a decision tree using the Gini index. Let D be the training data shown earlier in above table , where there are nine tuples belonging to the class buys computer = yes and the remaining five tuples belong to the class buys computer = no. A (root) node N is created for the tuples in D. We first use Eq. for the Gini index to compute the impurity of D:



To find the splitting criterion for the tuples in D, we need to compute the Gini index for each attribute. Let’s start with the attribute income and consider each of the possible splitting subsets. Consider the subset {low, medium}. This would result in 10 tuples in partition D1 satisfying the condition “income ∈ {low, medium}.” The remaining four tuples of D would be assigned to partition D2. The Gini index value computed based on this partitioning is



## **Tree Pruning**

Tree pruning is performed in order to remove anomalies in the training data due to noise or outliers. The pruned trees are smaller and less complex.

### **Tree Pruning Approaches**

There are two approaches to prune a tree −

* **Pre-pruning** − The tree is pruned by halting its construction early.
* **Post-pruning** - This approach removes a sub-tree from a fully grown tree.

## **Cost Complexity**

The cost complexity is measured by the following two parameters −

* Number of leaves in the tree, and
* Error rate of the tree.

**Bayesian Classification**

Bayesian classification is based on Bayes' Theorem. Bayesian classifiers are the statistical classifiers. Bayesian classifiers can predict class membership probabilities such as the probability that a given tuple belongs to a particular class.

Bayesian classifiers have also exhibited high accuracy and speed when applied to large databases.

## **Baye's Theorem**

Bayes' Theorem is named after Thomas Bayes. There are two types of probabilities −

* Posterior Probability or a posteriori probability [P(H/X)] , of H conditioned on X.
* Prior Probability or a priori probability [P(H)],of H

where X is data tuple and H is some hypothesis.

The posterior probability, P(H|X), is based on more information (e.g., customer information) than the prior probability, P(H), which is independent of X.

Similarly, P(X|H) is the posterior probability of X conditioned on H.

P(X) is the prior probability of X

According to Bayes' Theorem,

P(H/X)= P(X/H)P(H) / P(X)

Na¨ıve Bayesian Classification

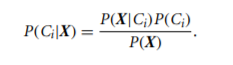
The na¨ıve Bayesian classifier, or simple Bayesian classifier, works as follows:

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. That is, the na¨ıve Bayesian classifier predicts that tuple X belongs to the class Ci if and only if

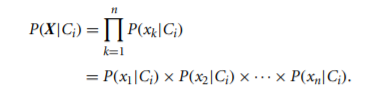


The class Ci for which P(Ci |X) is maximized is called the maximum posteriori hypothesis. By Bayes’ theorem



3. As P(X) is constant for all classes, only P(X|Ci)P(Ci) needs to be maximized. If the class prior probabilities are not known, then it is commonly assumed that the classes are 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). Tthe class prior probabilities may be estimated by P(Ci) = |Ci,D|/|D|, where |Ci,D| is the number of training tuples of class Ci in D.

4. Given data sets with many attributes, it would be extremely computationally expensive to compute P(X|Ci). To reduce computation in evaluating P(X|Ci), the na¨ıve assumption of class-conditional independence is made.



5. To predict the class label of X, P(X|Ci)P(Ci) is evaluated for each class Ci . The classifier predicts that the class label of tuple X is the class Ci if and only if



**Rule-Based Classification**

A rule-based classifier uses a set of IF-THEN rules for classification. An IF-THEN rule e is an expression of the form

IF condition THEN conclusion.

An example is rule R1,

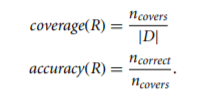
R1: IF age = youth AND student = yes THEN buys computer = yes

The “IF” part (or left side) of a rule is known **as the rule antecedent or precondition**. The “THEN” part (or right side) **is the rule consequent**. In the rule antecedent, the condition consists of one or more attribute tests (e.g., age = youth and student = yes) that are logically ANDed

R1 can also be written as

**R1: (age = youth) ∧ (student = yes) ⇒ (buys computer = yes).**

A rule R can be assessed by its coverage and accuracy. Given a tuple, X, from a classlabeled data set, D, let ncovers be the number of tuples covered by R; ncorrect be the number of tuples correctly classified by R; and |D| be the number of tuples in D. We can define the coverage and accuracy of R as



.

## **Bayesian Belief Network**

Bayesian Belief Networks specify joint conditional probability distributions. They are also known as Belief Networks, Bayesian Networks, or Probabilistic Networks.

* A Belief Network allows class conditional independencies to be defined between subsets of variables.
* It provides a graphical model of causal relationship on which learning can be performed.
* We can use a trained Bayesian Network for classification.

There are two components that define a Bayesian Belief Network −

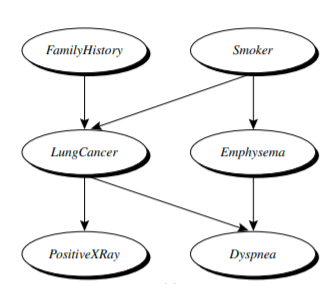
* Directed acyclic graph
* A set of conditional probability tables

## **Directed Acyclic Graph**

* Each node in a directed acyclic graph represents a random variable.
* These variable may be discrete or continuous valued.
* These variables may correspond to the actual attribute given in the data.

## **Directed Acyclic Graph Representation**

The following diagram shows a directed acyclic graph for six Boolean variables.



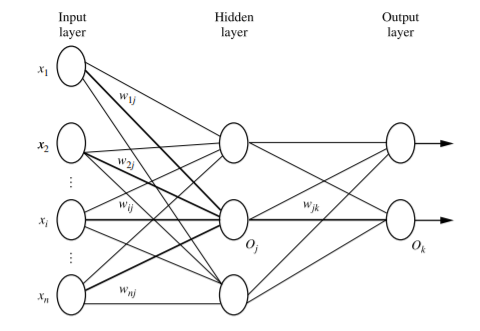
## **Conditional Probability**

The CPT for a variable Y specifies the conditional distribution P(Y|Parents(Y)), where Parents(Y) are the parents of Y.

CLASSIFICATION BY BACKPROPAGATION

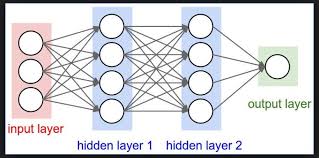
1.A MULTILAYER FEED-FORWARD NEURAL NETWORK

A multilayer feed-forward neural network consists of an input layer, one or more hidden layers, and an output layer



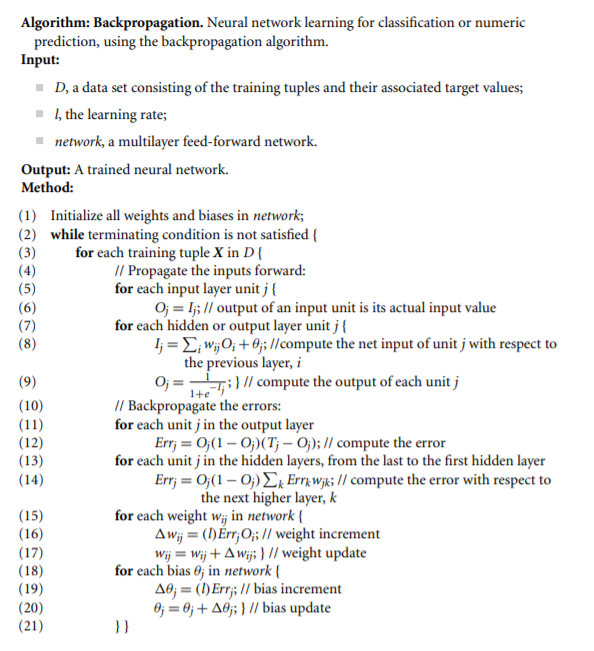
* Each layer is made up of units.
* The inputs to the network correspond to the attributes measured for each training tuple.
* The inputs are fed simultaneously into the units making up the input layer.
* These inputs pass through the input layer and are then weighted and fed simultaneously to a second layer of “neuronlike” units, known as a hidden layer.
* The outputs of the hidden layer units can be input to another hidden layer, and so on.
* The number of hidden layers is arbitrary, although in practice, usually only one is used.
* The weighted outputs of the last hidden layer are input to units making up the output layer, which emits the network’s prediction for given tuples.
* The units in the input layer are called input units.
* The units in the hidden layers and output layer are sometimes referred to as neurodes, due to their symbolic biological basis, or as output units.
* The multilayer neural network shown in Figure has two layers of output units. Therefore, we say that it is a **two-layer neural network**.

A network containing two hidden layers is called a **three-layer neural network**, and so on. It is a feed-forward network since none of the weights cycles back to an input unit or to a previous layer’s output unit. It is fully connected in that each unit provides input to each unit in the next forward layer.

****

**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 numeric prediction)



**Initialize the weights:** The weights in the network are initialized to small random numbers (e.g., ranging from −1.0 to 1.0, or −0.5 to 0.5). Each unit has a bias associated with it, as explained later. The biases are similarly initialized to small random numbers.

Each training tuple, X, is processed by the following steps.

**Propagate the inputs forward:** First, the training tuple is fed to the network’s input layer. The inputs pass through the input units, unchanged. That is, for an input unit, j, its output, Oj , is equal to its input value, Ij . Next, the net input and output of each unit in the hidden and output layers are computed.

To compute the net input to the unit, each input connected to the unit is multiplied by its corresponding weight, and this is summed. Given a unit, j in a hidden or output layer, the net input, Ij , to unit j is



where wij is the weight of the connection from unit i in the previous layer to unit j; Oi is the output of unit i from the previous layer; and θj is the bias of the unit. The bias acts as a threshold in that it serves to vary the activity of the unit.

Each unit in the hidden and output layers takes its net input and then applies an activation function to it, as illustrated in Figure. The function symbolizes the activation of the neuron represented by the unit. The logistic, or sigmoid, function is used. Given the net input Ij to unit j, then Oj , the output of unit j, is computed as



This function is also referred to as a squashing function, because it maps a large input domain onto the smaller range of 0 to 1.

Backpropagate the error: The error is propagated backward by updating the weights and biases to reflect the error of the network’s prediction. For a unit j in the output layer, the error Errj is computed by

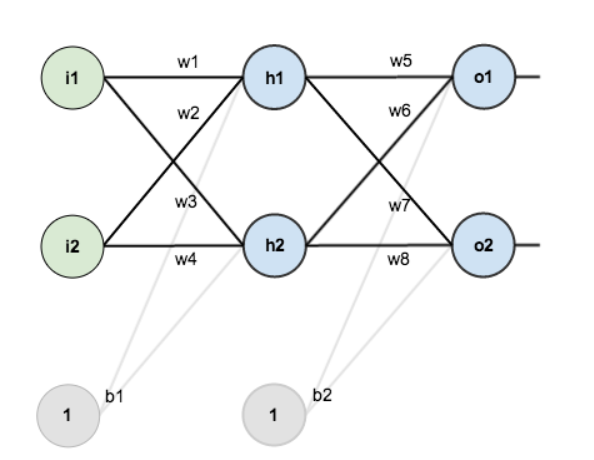


where Oj is the actual output of unit j, and Tj is the known target value of the given training tuple. Note that Oj(1 − Oj) is the derivative of the logistic function.

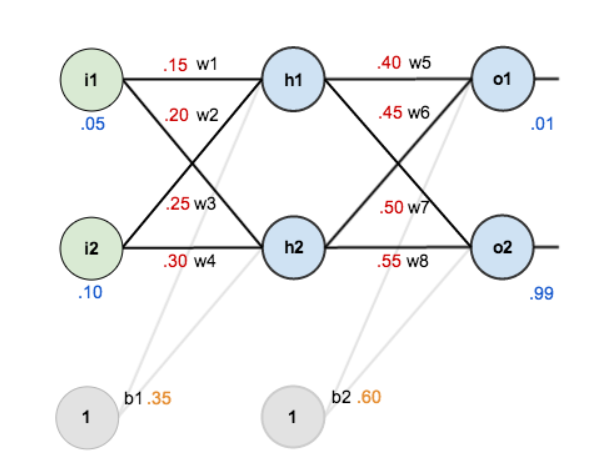
To compute the error of a hidden layer unit j, the weighted sum of the errors of the units connected to unit j in the next layer are considered. The error of a hidden layer unit j is



Example:



here are the initial weights, the biases, and training inputs/outputs:



The goal of backpropagation is to optimize the weights so that the neural network can learn how to correctly map arbitrary inputs to outputs.

For the rest of this tutorial we’re going to work with a single training set: given inputs 0.05 and 0.10, we want the neural network to output 0.01 and 0.99.

## The Forward Pass

Here’s how we calculate the total net input for h_1:

net_{h1} = w_1 * i_1 + w_2 * i_2 + b_1 * 1

net_{h1} = 0.15 * 0.05 + 0.2 * 0.1 + 0.35 * 1 = 0.3775

INPUT OF h2

=i1\*w3+i2\*w4=1\*b1

0.05\*0.25+0.10\*.40+0.35\*1

We then squash it using the logistic function to get the output of h_1:

out_{h1} = \frac{1}{1+e^{-net_{h1}}} = \frac{1}{1+e^{-0.3775}} = 0.593269992

Carrying out the same process for h_2 we get:

out_{h2} = 0.596884378

### Calculating the Total Error

We can now calculate the error for each output neuron using the [squared error function](http://en.wikipedia.org/wiki/Backpropagation#Derivation) and sum them to get the total error:

E_{total} = \sum \frac{1}{2}(target - output)^{2}

For example, the target output for o_1 is 0.01 but the neural network output 0.75136507, therefore its error is:

E_{o1} = \frac{1}{2}(target_{o1} - out_{o1})^{2} = \frac{1}{2}(0.01 - 0.75136507)^{2} = 0.274811083

Repeating this process for o_2 (remembering that the target is 0.99) we get:

E_{o2} = 0.023560026

The total error for the neural network is the sum of these errors:

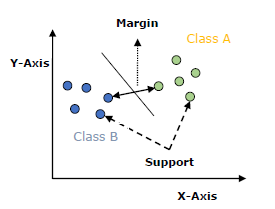
E_{total} = E_{o1} + E_{o2} = 0.274811083 + 0.023560026 = 0.298371109

**SUPPORT VECTOR MACHINES**

A method for the classification of both linear and nonlinear data

## **Working of SVM**

An SVM model is basically a representation of different classes in a hyperplane in multidimensional space. The hyperplane will be generated in an iterative manner by SVM so that the error can be minimized. The goal of SVM is to divide the datasets into classes to find a maximum marginal hyperplane (MMH).



The followings are important concepts in SVM −

* **Support Vectors** − Datapoints that are closest to the hyperplane is called support vectors. Separating line will be defined with the help of these data points.
* **Hyperplane** − As we can see in the above diagram, it is a decision plane or space which is divided between a set of objects having different classes.
* **Margin** − It may be defined as the gap between two lines on the closet data points of different classes. It can be calculated as the perpendicular distance from the line to the support vectors. Large margin is considered as a good margin and small margin is considered as a bad margin.

The main goal of SVM is to divide the datasets into classes to find a maximum marginal hyperplane (MMH) and it can be done in the following two steps −

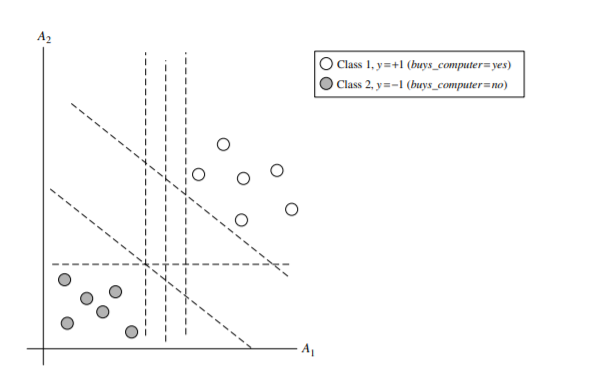
* First, SVM will generate hyperplanes iteratively that segregates the classes in best way.
* Then, it will choose the hyperplane that separates the classes correctly.

**1.The Case When the Data Are Linearly Separable**

Consider a two-class problem where the classes are linearly separable.

* Let the data set D be given as (X1, y1), (X2, y2), ... , (X|D| , y|D|), where Xi is the set of training tuples with associated class labels, yi .
* Each yi can take one of two values, either +1 or −1 (i.e., yi ∈ {+1, − 1}),

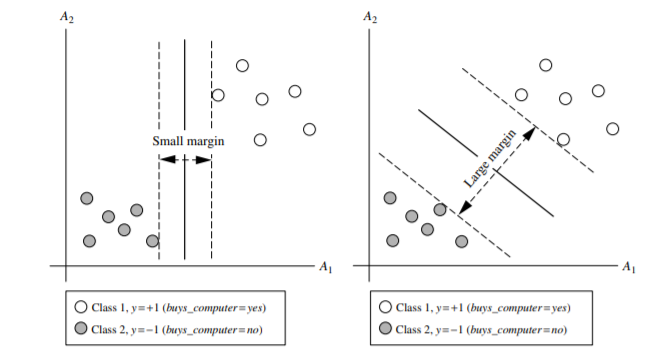
The graph shows the 2-D data are linearly separable (or “linear,”), because a straight line can be drawn to separate all the tuples of class +1 from all the tuples of class −1



An SVM approaches is used for searching the maximum marginal hyperplane. The Figures shows two possible separating hyperplanes and their associated margins.

Both hyperplanes classify all the given data tuples correctly.

However, we expect the hyperplane with the larger margin to be more accurate at classifying future data tuples than the hyperplane with the smaller margin.



A separating hyperplane can be written as

W ·X + b = 0 ------------------(1)

where **W** is a weight vector, namely, W = {w1, w2,..., wn};

**n** is the number of attributes;

**b** is a scalar, often referred to as a bias. To aid in visualization, let’s consider two input attributes, A1 and A2, as in Figure . Training tuples are 2-D (e.g., X = (x1, x2)), where x1 and x2 are the values of attributes A1 and A2, respectively, for X. If we add of **b** as an additional weight, w0,,then the above equation can be written as

w0 + w1x1 + w2x2 = 0 ------------------(2)

Thus, any point that lies above the separating hyperplane satisfies

w0 + w1x1 + w2x2 > 0. ----------------------(3)

Similarly, any point that lies below the separating hyperplane satisfies

w0 + w1x1 + w2x2 < 0.------------------------(4)

The weights can be adjusted so that the hyperplanes defining the “sides” of the margin can be written as

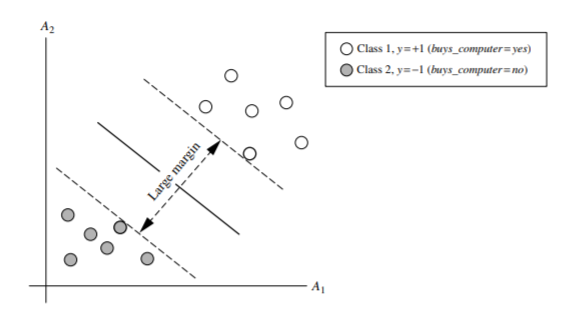
H1 : w0 + w1x1 + w2x2 ≥ 1 for yi = +1, ------------(5)

H2 : w0 + w1x1 + w2x2 ≤ −1 for yi = −1. -----------(6)

That is, any tuple that falls on or above H1 belongs to class +1, and any tuple that falls on or below H2 belongs to class −1. Combining the two inequalities of Eqs.(5) &(6)

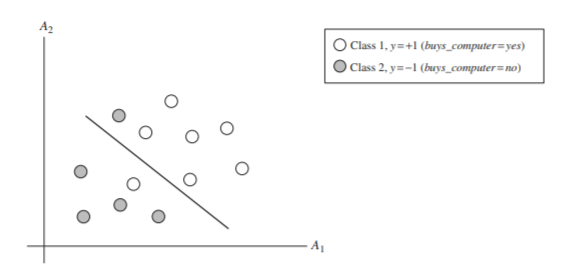
yi(w0 + w1x1 + w2x2) ≥ 1, ∀i. ---------------(7)

Any training tuples that fall on hyperplanes H1 or H2 (i.e., the “sides” defining the margin) satisfy Eq. (7) and are called support vectors. That is, they are equally close to the (separating) MMH



The distance from the separating hyperplane to any point on H1 is 1 ||W|| , where ||W|| is the Euclidean norm of W, that is, √ W · W

**2.The Case When the Data Are Linearly Inseparable**



**LAZY LEARNERS**

**Lazy learning** (e.g., instance-based learning): Simply stores training data and waits until it is given a test tuple.

**Eager learning** : Given a set of training tuples, constructs a classification model before receiving new (e.g., test) data to classify

two examples of lazy learners:

**k-nearest-neighbor classifiers**

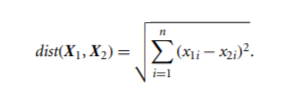
**case-based reasoning classifiers**

k-Nearest-Neighbor Classifiers

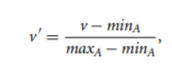
nearest neighbors is a simple algorithm that stores all available cases and classifies new cases based on a similarity measure (e.g., distance functions).

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, X1 = (x11, x12,..., x1n) and X2 = (x21, x22,..., x2n), is



Min-max normalization, for example, can be used to transform a value v of a numeric attribute A to v1 in the range [0, 1] by computing



Case-Based Reasoning

**CBR**: Uses a database of problem solutions to solve new problems

* Store symbolic description (tuples or cases)—not points in a Euclidean space
* Applications: Customer-service (product-related diagnosis), legal ruling
* Methodology
* Instances represented by rich symbolic descriptions (e.g., function graphs)
* Search for similar cases, multiple retrieved cases may be combined
* Tight coupling between case retrieval, knowledge-based reasoning, and problem solving
* Challenges
  + Find a good similarity metric
  + Indexing based on syntactic similarity measure, and when failure, backtracking, and adapting to additional cases

Other Classification Methods

* **Genetic Algorithms**
* **Rough Set Approach**
* **Fuzzy Set Approaches**

**Genetic Algorithmis based on an analogy to biological evolution**

**Genetic algorithms** are excellent for searching through large and complex data sets.

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

Each rule is represented by a string of bits

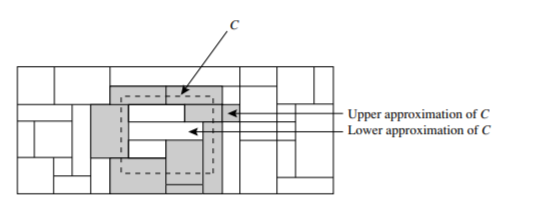
Example

suppose that samples in a given training set are described by two Boolean attributes, A1 and A2, and that there are two classes, C1 and C2. The rule “IF A1 AND NOT A2 THEN C2” can be encoded as the bit string “100,” where the two leftmost bits represent attributes A1 and A2, respectively, and the rightmost bit represents the class. Similarly, the rule “IF NOT A1 AND NOT A2 THEN C1” can be encoded as “001.” If an attribute has k values, where k > 2, then k bits may be used to encode the attribute’s values.

* Based on the notion of survival of the **fittest**, a new population is formed to consist of the fittest rules and their offspring
* The *fitness of a rule* is represented by its classification accuracy on a set of training examples
* Offspring are generated by *crossover* and *mutation*
* Crossover, substrings from pairs of rules are swapped to form new pairs of rules.
* In mutation, randomly selected bits in a rule’s string are inverted.
* The process continues until a population P evolves *when each rule in P satisfies a prespecified threshold*

**Rough Set Approach**

* 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

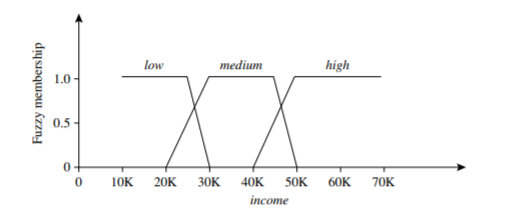


**Fuzzy Set Approaches**

* Fuzzy logic uses truth values between 0.0 and 1.0 to represent the degree of membership (such as in a *fuzzy membership graph*)
* Attribute values are converted to fuzzy values. Ex.:



* + Income, *x*, is assigned a fuzzy membership value to each of the discrete categories {low, medium, high}, e.g. $49K belongs to “medium income” with fuzzy value
* Each applicable rule contributes a vote for membership in the categories



Unit V

Cluster Analysis

* Cluster: A collection of data objects
* similar (or related) to one another within the same group
* dissimilar (or unrelated) to the objects in other groups
* Cluster analysis (or *clustering*, *data segmentation, …*)
  + Finding similarities between data according to the characteristics found in the data and grouping similar data objects into clusters
* Unsupervised learning: no predefined classes (i.e., *learning by observations* vs. learning by examples: supervised)

Requirements for Cluster Analysis

* Scalability
* Ability to deal with different types of attributes
* Discovery of clusters with arbitrary shape
* Requirements for domain knowledge to determine input parameters
* Ability to deal with noisy data
* Incremental clustering and insensitivity to input order
* Capability of clustering high-dimensionality data
* Constraint-based clustering
* Interpretability and usability

Overview of Basic Clustering Methods

Partitioning methods: Given a set of n objects, a partitioning method constructs k partitions of the data, where each partition represents a cluster and k ≤ n. That is, it divides the data into k groups such that each group must contain at least one object.

Density-based methods can divide a set of objects into multiple exclusive clusters, or a hierarchy of clusters

Grid-based methods: Grid-based methods quantize the object space into a finite number of cells that form a grid structure

Hierarchical methods: A hierarchical method creates a hierarchical decomposition of the given set of data objects.

