Unit 3: Classification

What is classification?

- Is a data mining technique used to predict the category of categorical data by building a model based on some predictor variables(to classify data).
- Predicator variable/ attribute is called class label attribute(predefined class)

Following are the **examples** of cases where the data analysis task is Classification:

- A bank loan officer wants to analyze the data in order to know which customers (loan applicant) are risky or which are safe.
- A marketing manager at a company needs to analyze a customer with a given profile, who will buy a new computer.

In both of the above examples, a model or classifier is constructed to predict the categorical labels. These labels are risky or safe for loan application data and yes or no for marketing data.

How does classification work?

It is a two-step process

1. Model Construction (learning step or training phase) -

   build a model to explain the target concept

   -model is represented as classification rules, decision trees, or mathematical formulae.

2. Model Usage

   -is used for classifying future or unknown cases

   -estimate the accuracy of the model

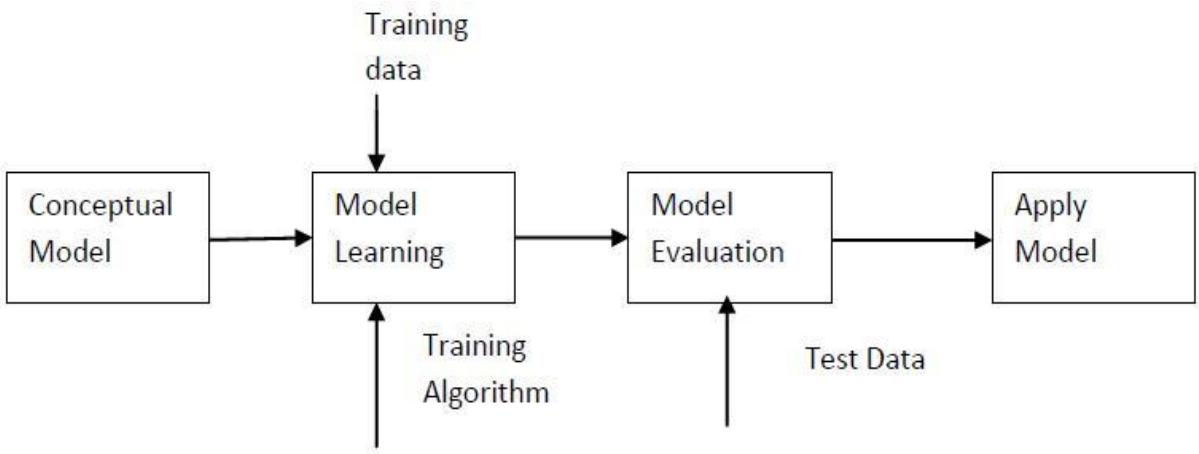


Fig: Stages in classification

Building the Classifier or Model

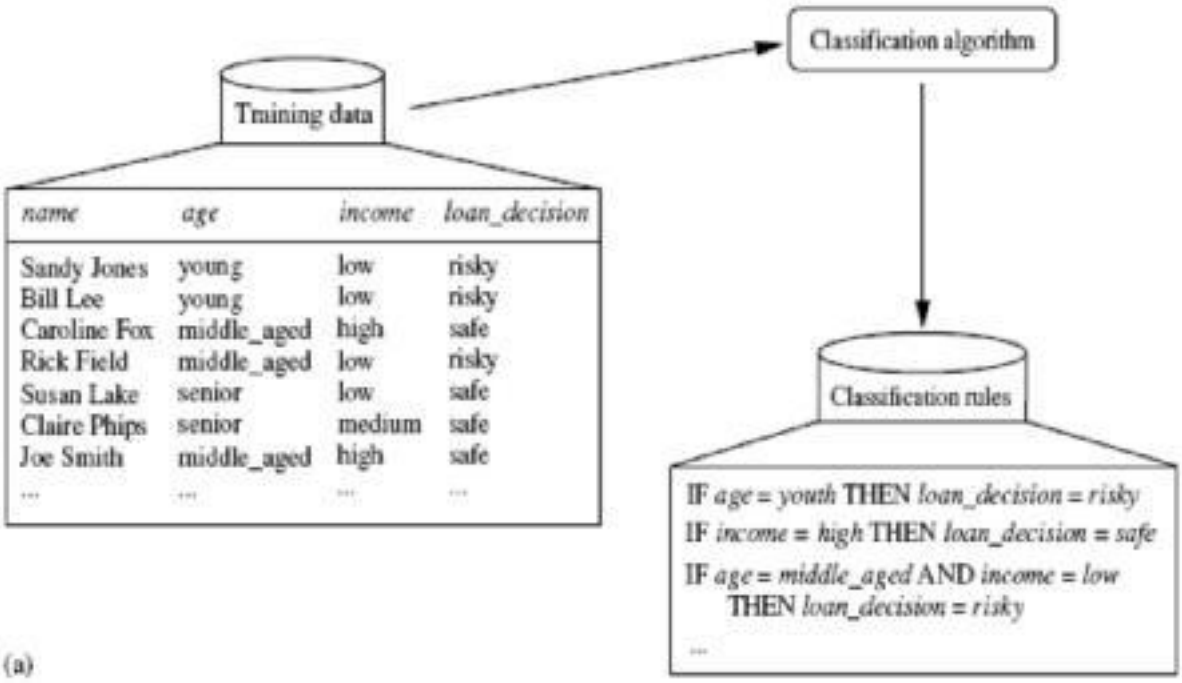
- This step is the learning step or the learning phase.
- In this step the classification algorithms build the classifier.

- The classifier is built from the training set made up of database tuples and their associated class labels.
- Each tuple that constitutes the training set is referred to as a category or class. These tuples can also be referred to as sample, object or data points.
- Because the class label of each training tuple is provided, this step is also known as supervised learning (i.e., the learning of the classifier is "supervised" in that it is told to which class each training tuple belongs). It contrasts with unsupervised learning (or clustering), in which the class label of each training tuple is not known, and the number or set of classes to be learned may not be known in advance.

For example, Consider the following set training data:

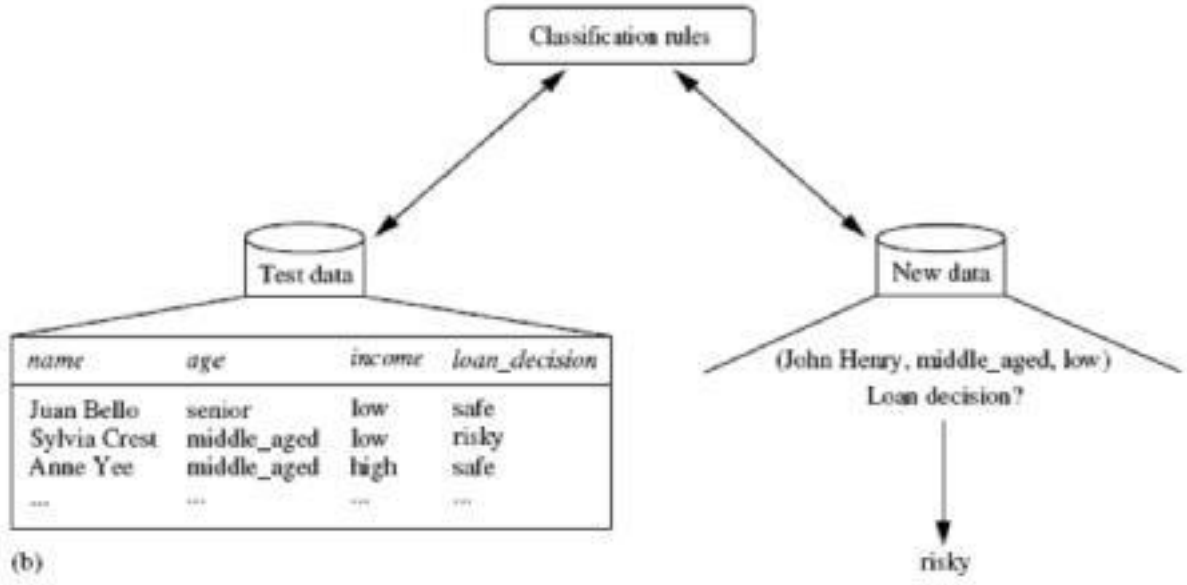
RID	age	income	student	credit_rating	Class: buys_computer
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middle_aged	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
6	senior	low	yes	excellent	no
7	middle_aged	low	yes	excellent	yes
8	youth	medium	no	fair	no
9	youth	low	yes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

Table 1: Training data set of AllElectronics



Using Classifier for Classification

In this step, the classifier is used for classification. Here the test data is used to estimate the accuracy of classification rules. The classification rules can be applied to the new data tuples if the accuracy is considered acceptable.



Preparing the Data for Classification

The major issue is preparing the data for Classification:

- Data cleaning

- Relevance Analysis
- Data Transformation and reduction

Comparing Classification Methods

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

- Accuracy: The accuracy of a classifier refers to the ability of a given classifier to correctly predict the class label of new or previously unseen data (i.e., tuples without class label information).
- Speed: This refers to the computational costs involved in generating and using the given classifier or predictor.
- Robustness: This is the ability of the classifier or predictor to make correct predictions given noisy data or data with missing values.
- Scalability: This refers to the ability to construct the classifier or predictor efficiently given large amounts of data.

Classification Types:

i. Decision Tree classifier
ii. Rule Based Classifier
iii. Nearest Neighbor Classifier
iv. Bayesian Classifier
v. Artificial Neural Network (ANN) Classifier

Decision Tree Classifier

A decision tree is a flowchart-like tree structure, where each internal node (nonleaf node) denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (or terminal node) holds a class label. The topmost node in a tree is the root node. For example:

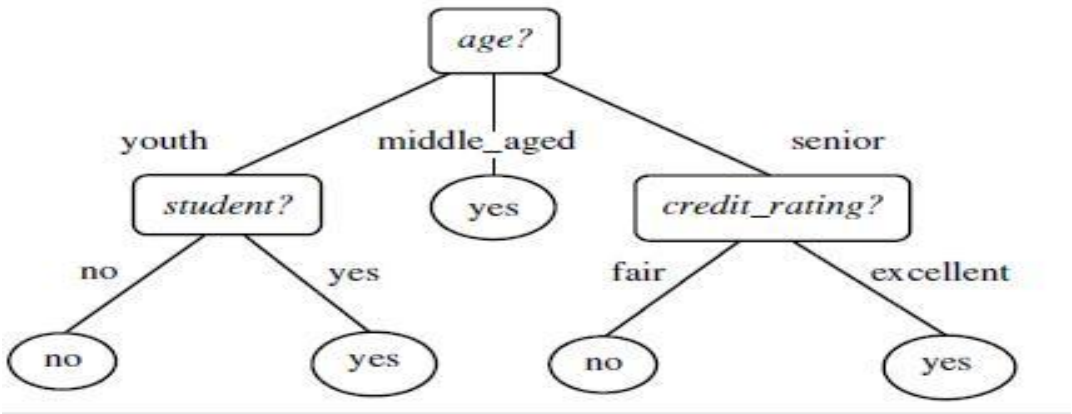


Fig 1: A decision tree for the concept buys computer, indicating whether a customer at AllElectronics is likely to purchase a computer. Each internal (nonleaf) node represents a test on an attribute. Each leaf node represents a class (either buys computer = yes or buys computer = no).

A typical decision tree is shown in Figure 1. 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.

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

"Why are decision tree classifiers so popular?" The construction of decision tree classifiers does not require any domain knowledge or parameter setting, and therefore is appropriate for exploratory knowledge discovery. Decision trees can handle high dimensional data.

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, ideally each partition would be pure (i.e., all of the tuples that fall into a given partition would belong to the same class). Conceptually, the "best" splitting criterion is the one that most closely results in such a scenario. Attribute selection measures are also known as splitting rules because they determine how the tuples at a given node are to be split. There are three popular attribute selection measures—information gain, gain ratio, and gini index.

Information gain

ID3 uses information gain as its attribute selection measure. This measure is based on pioneering work by Claude Shannon on information theory, which studied the value or "information content" of messages. Let node N represents or hold the tuples of partition D. The attribute with the highest information gain is chosen as the splitting attribute for node N. This attribute minimizes the information needed to classify the tuples in the resulting partitions and reflects the least randomness or "impurity" in these partitions.

The expected information needed to classify a tuple in D is given by

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

where pi is the probability that an arbitrary tuple in D belongs to class Ci and is estimated by |Ci,D|/|D|. A log function to the base 2 is used, because the information is encoded in bits. Info(D) is also known as the entropy of D.

Now, suppose we were to partition the tuples in D on some attribute A having v distinct values, {a1, a2, : : : , av} as observed from the training data. If A is discrete-valued, these values correspond directly to the v outcomes of a test on A. Attribute A can be used to split D into v partitions or subsets, {D1, D2, : : : , Dv}, where Dj contains those tuples in D that have outcome aj of A. How much more information would we still need (after the partitioning) in order to arrive at an exact classification? This amount is measured by

$$Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times Info(D_j).$$ ----------eq.(2)

The term |Dj|/ |D| acts as the weight of the jth partition. InfoA(D) is the expected information required to classify a tuple from D based on the partitioning by A. The smaller the expected information (still) required, the greater the purity of the partitions.

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

$$Gain(A) = Info(D) - Info_A(D).$$ ----------eq.(3)

The **attribute A with the highest information gain, (Gain(A)), is chosen as the splitting attrib**ute at node N.

Example 1: Decision tree using information gain.

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

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

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

Table: 1

In this example, 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 C1 correspond to yes and class C2 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 (1) to compute the expected information needed to classify a tuple in D:

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

Next, we need to compute the expected information requirement for each attribute.

Let's start with the attribute age. We need to look at the distribution of yes and no tuples for each category of age. For the age category youth, there are two yes tuples and three no tuples. For the category middle aged, there are four yes tuples and zero no tuples. For the category senior, there are three yes tuples and two no tuples.

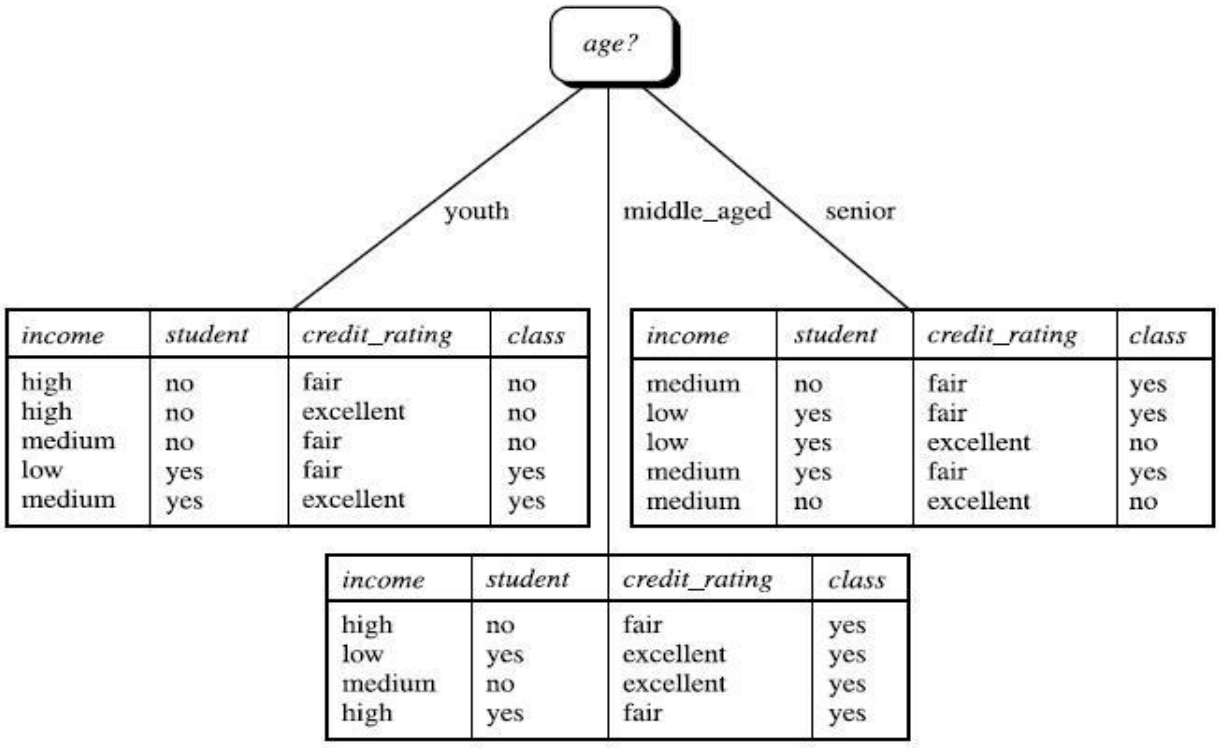
Using Equation (2) the expected information needed to classify a tuple in D if the tuples are partitioned according to age is,

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

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

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

Similarly, we can compute Gain(income) = 0.029 bits, Gain(student) = 0.151 bits, and Gain(credit rating) = 0.048 bits. Because age has the highest information gain among the attributes, it is selected as the splitting attribute. Node N is labeled with age, and branches are grown for each of the attribute's values. The tuples are then partitioned accordingly, as shown in Figure 2. 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 1.



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

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

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

Fig 2: The attribute age has the highest information gain and therefore becomes the splitting attribute at the root node of the decision tree. Branches are grown for each outcome of age. The tuples are shown partitioned accordingly.

Gain ratio

The information gain measure is biased toward tests with many outcomes. That is, it prefers to select attributes having a large number of values. For example, consider an attribute that acts as a unique identifier, such as product ID. A split on product ID would result in a large number of partitions (as many as there are values), each one containing just one tuple. Because each partition is pure, the information required to classify data set D based on this partitioning would be $Info_{product\_ID}(D) = 0$. Therefore, the information gained by partitioning on this attribute is maximal. Clearly, such a partitioning is useless for classification.

C4.5, a successor of ID3, uses an extension to information gain known as gain ratio, which attempts to overcome this bias. It applies a kind of normalization to information gain using a "split information" value defined analogously with Info(D) as

$$SplitInfo_A(D) = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} \times \log_2\left(\frac{|D_j|}{|D|}\right).$$ ----------eq.(4)

This value represents the potential information generated by splitting the training data set, D, into v partitions, corresponding to the v outcomes of a test on attribute A.

The gain ratio is defined as

$$GainRatio(A) = \frac{Gain(A)}{SplitInfo(A)}.$$ ----------eq.(5)

The attribute with the maximum gain ratio is selected as the splitting attribute.

Example 2: Computation of gain ratio for the attribute income.

A test on income splits the data of Table 1 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 Equation (4) to obtain

$$SplitInfo_A(D) = -\frac{4}{14} \times \log_2\left(\frac{4}{14}\right) - \frac{6}{14} \times \log_2\left(\frac{6}{14}\right) - \frac{4}{14} \times \log_2\left(\frac{4}{14}\right).$$

$$= 0.926.$$

From Example 1, we have Gain(income) = 0.029.

Therefore, GainRatio(income) =0.029/0.926 = 0.031.

Gini index

The Gini index is used in CART. Using the notation described above, the Gini index measures the impurity of D, a data partition or set of training tuples, as

$$Gini(D) = 1 - \sum_{i=1}^{m} p_i^2,$$

----------eq.(6)

where pi is the probability that a tuple in D belongs to class Ci and is estimated by |Ci,D|/|D|. The sum is computed over m classes. The Gini index considers a binary split for each attribute. Let's first consider the case where A is a discrete-valued attribute having v distinct values, {a1, a2, : : : , av}, occurring in D. To determine the best binary split on A, we examine all of the possible subsets that can be formed using known values of A If A has v possible values, then there are 2^v possible subsets. For example, if income has three possible values, namely {low, medium, high}, then the possible subsets are {low, medium, high}, {low, medium}, {low, high}, {medium, high}, {low}, {medium}, {high}, and {}. We exclude the power set, {low, medium, high}, and the empty set from consideration since, conceptually, they do not represent a split. Therefore, there are 2^v-2 possible ways to form two partitions of the data, D, based on a binary split on A. When considering a binary split, we compute a weighted sum of the impurity of each resulting partition. For example, if a binary split on A partitions D into D1 and D2, the gini index of D given that partitioning is

$$Gini_A(D) = \frac{|D_1|}{|D|} Gini(D_1) + \frac{|D_2|}{|D|} Gini(D_2).$$

----------eq.(7)

The reduction in impurity that would be incurred by a binary split on a discrete- or continuous-valued attribute A is

$$\Delta Gini(A) = Gini(D) - Gini_A(D).$$

----------eq.(8)

The attribute that maximizes the reduction in impurity (or, equivalently, has the minimum Gini index) is selected as the splitting attribute.

Example 3: Induction of a decision tree using gini index.

Let D be the training data of Table 1 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 Equation (6) for Gini index to compute the impurity of D:

$$Gini(D) = 1 - \left(\frac{9}{14}\right)^2 - \left(\frac{5}{14}\right)^2 = 0.459.$$

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 2 {low, medium}." The remaining four tuples of D would be assigned to partition D2. The Gini index value computed based on this partitioning is

$$\begin{aligned} &Gini_{income \in \{low,medium\}}(D) \\ &= \frac{10}{14} Gini(D_1) + \frac{4}{14} Gini(D_2) \\ &= \frac{10}{14}\left(1 - \left(\frac{6}{10}\right)^2 - \left(\frac{4}{10}\right)^2\right) + \frac{4}{14}\left(1 - \left(\frac{1}{4}\right)^2 - \left(\frac{3}{4}\right)^2\right) \\ &= 0.450 \\ &= Gini_{income \in \{high\}}(D). \end{aligned}$$

Similarly, the Gini index values for splits on the remaining subsets are: 0.315 (for the subsets {low, high} and {medium}) and 0.300 (for the subsets {medium, high} and {low}). Therefore, the best binary split for attribute income is on {medium, high} (or {low}) because it minimizes the gini index. Evaluating the attribute, we obtain {youth, senior} (or {middle aged}) as the best split for age with a Gini index of 0.375; the attributes {student} and {credit rating} are both binary, with Gini index values of 0.367and0.429, respectively.

The attribute income and splitting subset {medium, high} therefore give the minimum gini index overall, with a reduction in impurity of 0.459-0.300 = 0.159. The binary split "income 2 {medium, high}" results in the maximum reduction in impurity of the tuples in D and is returned as the splitting criterion. Hence, the Gini index has selected income instead of age at the root node.

Tree Pruning

- Tree Pruning is performed in order to remove anomalies in training data due to noise or outliers.
- The pruned trees are smaller and less complex.
- Pruning is a technique in machine learning that reduces the size of decision trees by removing sections of the tree that provide little power to classify instances.
- The dual goal of pruning is reduced complexity of the final classifier as well as better predictive accuracy by the reduction of overfitting and removal of sections of a classifier that may be based on noisy data.
- One of the questions that arise in a decision tree algorithm is the optimal size of the final tree.

- A tree that is too large risks overfitting the training data and poorly generalizing to new samples.
- A small tree might not capture important structural information about the sample space.
- It is hard to tell when a tree algorithm should stop because it is impossible to tell if the addition of a single extra node will dramatically decrease error.
- A common strategy is to grow the tree until each node contains a small number of instances then use pruning to remove nodes that do not provide additional information.
- Pruning should reduce the size of a learning tree without reducing predictive accuracy as measured by a test set or using cross-validation.
- There are many techniques for tree pruning that differ in the measurement that is used to optimize performance.

Tree pruning approaches

i. Pre-pruning - The tree is pruned by halting its construction early.

ii. Post-pruning - This approach removes subtree form fully grown tree.

Pre-pruning

- Based on statistical significance test.
- Stop growing the tree when there is no statistically significant association between any attribute and the class at a particular node
- Most popular test: chi-squared test
- ID3 used chi-squared test in addition to information gain.
- Only statistically significant attributes were allowed to be selected by information gain procedure.
- Pre-pruning may stop the growth process prematurely: early stopping
- Pre-pruning faster than post-pruning

Post-pruning

- First, build full tree then, prune it.
- Fully-grown tree shows all attribute interactions
- Problem: some subtrees might be due to chance effects

Advantages of Decision Tree Classifier

– Inexpensive to construct

– Extremely fast at classifying unknown records

– Easy to interpret for small-sized trees

– Accuracy is comparable to other classification techniques for many simple data sets

Bayesian Classification

Bayesian classification is based on Baye's Theorem.

- It is a statistical classifier that predicts class membership probabilities such as the probability that a given tuple belongs to a particular class.

Baye's Law

P(A/B) = P(B/A) P(A)/ P(B)……..eq(8)

- Has high accuracy and speed for large databases.
- Has minimum error rate in comparison to all other classifier

Naïve Bayesian Classification

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

1. Let D be a training set of tuples and their associated class labels. As usual, each tuple is represented by an n-dimensional attribute vector, X = (x1, x2, : : : , xn), depicting n measurements made on the tuple from n attributes, respectively, A1, A2, : : : , An.

2. Suppose that there are m classes, C1, C2, : : : , Cm. Given a tuple, X, the classifier will predict that X belongs to the class having the highest posterior probability, conditioned on X. That is, the naïve Bayesian classifier predicts that tuple X belongs to the class Ci if and only if

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

Thus we maximize P(CijX). The classCi for which P(CijX) is maximized is called the maximum posteriori hypothesis. By Bayes' theorem (Equation (8)),

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

3. As P(X) is constant for all classes, only P(XjCi)P(Ci) need be maximized. If the class prior probabilities are not known, then it is commonly assumed that the classes are equally likely, that is, P(C1) = P(C2) = _ _ _ = P(Cm), and we would therefore maximize P(XjCi). Otherwise, we maximize P(XjCi)P(Ci). Note that the class prior probabilities may be estimated by P(Ci)=jCi,Dj/jDj,where jCi,Dj 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(XjCi). In order to reduce computation in evaluating P(XjCi), the naive assumption of class conditional independence is made. This presumes that the values of the attributes are conditionally independent of one another, given the class label of the tuple (i.e., that there are no dependence relationships among the attributes). Thus,

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

Example 4: Predicting a class label using naïve Bayesian classification.

We wish to predict the class label of a tuple using naïve Bayesian classification, given the same training data as in Example 1 for decision tree induction. The training data are in Table 1. The data tuples are described by the attributes age, income, student, and credit rating. The class label attribute, buys computer, has two distinct values (namely, {yes, no}). Let C1 correspond to the class buys computer = yes and C2 correspond to buys computer = no. The tuple we wish to classify is

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

We need to maximize P(XjCi)P(Ci), for i = 1, 2. P(Ci), the prior probability of each class, can be computed based on the training tuples:

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

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

To compute P(X|Ci), for i = 1, 2, we compute the following conditional probabilities:

P(age = youth | buys computer = yes) = 2/9 = 0.222

P(age = youth | buys computer = no) = 3/5 = 0.600

P(income = medium | buys computer = yes) = 4/9 = 0.444

P(income = medium | buys computer = no) = 2/5 = 0.400

P(student = yes | buys computer = yes) = 6/9 = 0.667

P(student = yes | buys computer = no) = 1/5 = 0.200

P(credit rating = fair | buys computer = yes) = 6/9 = 0.667

P(credit rating = fair | buys computer = no) = 2/5 = 0.400

Using the above probabilities, we obtain

P(X | buys computer = yes) = P(age = youth | buys computer = yes) * P(income = medium | buys computer = yes) * P(student = yes | buys computer = yes) * P(credit rating = fair | buys computer = yes)

= 0.222*0.444*0.667*0.667 = 0.44.

Similarly,

P(X | buys computer = no) = 0.600*0.400*0.200*0.400 = 0.019. To

find the class, Ci, that maximizes P(X | Ci)P(Ci), we compute

P(X | buys computer = yes) P(buys computer = yes) = 0.044*0.643 = 0.028

P(X | buys computer = no) P(buys computer = no) = 0.019*0.357 = 0.007

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

Rule-Based Classification

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

IF condition THEN conclusion.

An example is rule R1,

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

The "IF"-part (or left-hand side) of a rule is known as the rule antecedent or precondition. The "THEN"-part (or right-hand side) is the rule consequent. In the rule antecedent, the condition consists of one or more attribute tests (such as age = youth, and student = yes) that are logically ANDed. The rule's consequent contains a class prediction (in this case, we are predicting whether a customer will buy a computer). R1 can also be written as R1: (age = youth) ^ (student = yes))(buys computer = yes).

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

A rule R can be assessed by its coverage and accuracy. Given a tuple, X, from a class labeled data set,D, let n_{covers} be the number of tuples covered by R; $n_{correct}$ 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

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

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

That is, a rule's coverage is the percentage of tuples that are covered by the rule (i.e., whose attribute values hold true for the rule's antecedent). For a rule's accuracy, we look at the tuples that it covers and see what percentage of them the rule can correctly classify.

Example 5: Rule accuracy and coverage.

Let's go back to our data of Table 1. These are class-labeled tuples from the AllElectronics customer database. Our task is to predict whether a customer will buy a computer. Consider

rule R1 above, which covers 2 of the 14 tuples. It can correctly classify both tuples. Therefore, coverage(R1) = 2/14 = 14.28% and accuracy (R1) = 2/2 = 100%.

How does Rule-Based Classifier work?

- If a rule is satisfied by a tuple, the rule is said to be triggered. Triggering doesn't always mean firing because there may be more than one rules that can be satisfied.
- Three different cases occur for classification.

Case-I: If only one rule is satisfied

- When any instances is covered by only one rule then the rule fires by returning the class prediction for the tuple defined by the rule.

Case-II: If more than one rules are satisfied

- If more than one rules are triggered, we need a conflict resolution strategy to find which rule is fired.
- Rule ordering or rule ranking or rule priority can be set in case of rules conflict. A rule ordering may be class-based or rule-based.
- Rule-based ordering: Individual rules are ranked based on their quality.
- Class-based ordering: Rules that belong to the same class appear together
- When rule-based ordering is used, the rule set is known as a decision list.

- If any instance not triggered by any rule, use default class for classification. Mostly most frequent class is assigned as default class.

Eg:

S.No.	Name	Blood Type	Give Birth	Can fly	Live in water	Class
1	Lemur	Warm	Yes	No	No	?
2	Turtle	Cold	No	No	Sometimes	?
3	Shark	Cold	Yes	No	Yes	?

Rule base

R1: (Give Birth = No) ^ (Can fly = Yes) => Birds

R2: (Give Birth = No) ^ (Live in Water = Yes) => Fishes

R3: (Give Birth = Yes) ^ (Blood Type = Warm) => Mammals

R4: (Give Birth = No) ^ (Can fly = No) => Reptiles

R5: (Live in Water = Sometimes) => Amphibians

- In above example, R1 and R2 don't have any coverage. R3, R4 & R5 have coverage.

- Instance 1 is triggered by R3, instance 2 is triggered by R4 & R5 and instance 3 is not triggered by any instances.
- Since instance 1 is triggered by only one rule (R3) so it is fired as a class mammal, instance 2 is triggered by more than two rules (R4 & R5) and hence conflict occurs. To resolve the conflict the class can be identified using priority (rule priority or class priority). Instance 3 is not triggered by any rules, to resolve this conflict default class can be used.

Rule Extraction from Decision Tree

To extract rules from a decision tree, one rule is created for each path from the root to a leaf node. Each splitting criterion along a given path is logically ANDed to form the rule antecedent ("IF" part). The leaf node holds the class prediction, forming the rule consequent ("THEN" part).

Example 6: Extracting classification rules from a decision tree.

The decision tree of Figure 1 can be converted to classification IF-THEN rules by tracing the path from the root node to each leaf node in the tree. The rules extracted from Figure 1 are

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

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

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

R4: IF age = senior AND credit rating = excellent THEN buys computer = yes

R5: IF age = senior AND credit rating = fair THEN buys computer = no

Advantages of Rule-Based Classifiers

- As highly expressive as decision trees
- Easy to interpret
- Easy to generate
- Can classify new instances rapidly
- Performance comparable to decision trees

Nearest neighbor Classifier

- Uses k "closest" points (nearest neighbors) for performing classification. K-closet neighbor of a record 'X' are data points that have the K-smallest distance of 'X'.
- Classification based on learning by analogy i.e. by comparing a given test tuple with training tuple that are similar to it.
- Training tuples are described by n-attributes.
- When given an unknown tuple, a k-nearest- neighbor classifier searches the pattern space for the k-training tuples that are closest to the unknown tuple.
- Nearest neighbor classifier requires:

- Set of stored records

- Distance matric to compute distance between records. For distance calculation any standard approach can be used such as Euclidean distance.

- The value of 'K', the number of nearest neighbor to retrieve.

- To classify the unknown records

- Compute distance to other training records.

- Identify the k-nearest neighbor.

-Use class label nearest neighbors to determine the class label of unknown record.

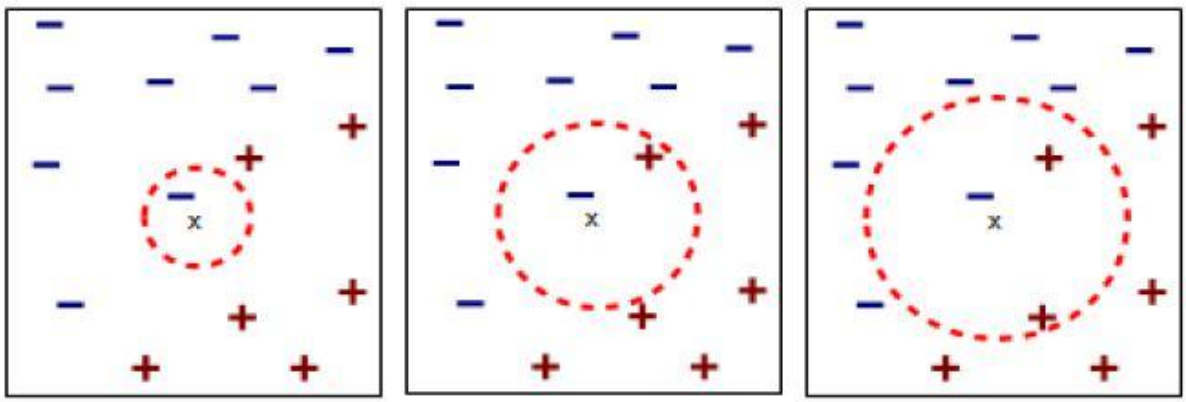
In case of conflict, use majority vote for classification.

Issues of classification using k-nearest neighbor

classification i. Choosing the value of K

- One of challenge in classification is to choose the appropriate value of K. If K is too small, it is sensitive to noise points. If K is too large, neighbor may include points from other classes.

- With the change of value of K, the classification result may vary.



ii. Scaling Issue

- Attribute may have to be scaled to prevent distance measure from being dominated by one of attributes. Eg. Height, Temperature etc.

iii. Distance computing for non-numeric data.

- Use Distance as 0 for the same data and maximum possible distance for different data.

iv. Missing values

- Use maximum possible distance

Disadvantages:

- Poor accuracy when data have noise and irrelevant attributes.

- Slow when classifying test tuples.
- Classifying unknown records are relatively expensive.

Artificial Neural Network (ANN) Classifier

- It is set of connected i/o units in which each connection has a weight associated with it.
- During the learning phase the network learns by adjusting the weights so as to be able to predict the correct class label of i/p labels.
- It also referred as connectionist learning due to connection between units.
- It has long training time and poor interpretability but has tolerance to noisy data.
- It can classify pattern on which they have not been trained.
- Well suited for continuous valued i/ps.
- It has parallel topology and processing.
- Before training the network topology must be designed by:

i. Specifying number of i/p nodes/units: Depends upon number of independent variable in data set.

ii. Number of hidden layers: Generally only layer is considered in most of the problem. Two layers can be designed for complex problem. Number of nodes in the hidden layer can be adjusted iteratively.

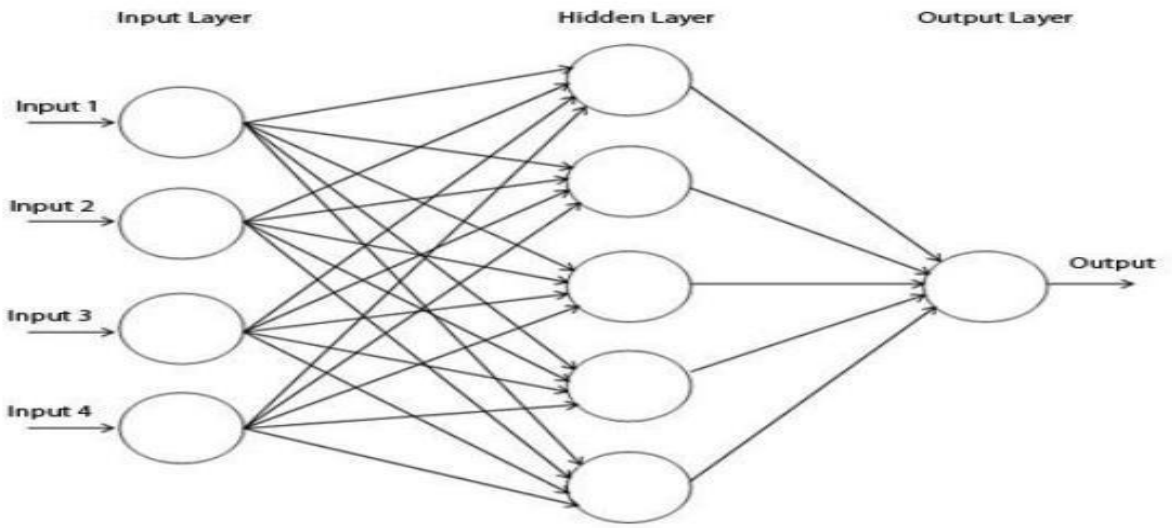
iii. Number of output nodes/units: Depends upon number of class labels of the data set.

iv. Learning rate: Can be adjusted iteratively.

v. Learning algorithm: Any appropriate learning algorithm can be selected during training phase.

vi. Bias value: Can be adjusted iteratively.

- During training the connection weights must be adjusted to fit i/p values with the o/p values.



Back propagation algorithm

Step 1: Initialization: Set all the weights and thresholds levels of the network to random numbers uniformly distributed inside a small range.

Step 2: Activation: Activate the back propagation neural network by applying i/ps and desired o/ps.

i. Calculate the actual o/ps of the neurons in the hidden layers.

ii. Calculate the actual o/ps of the neurons in the o/p layers.

Step 3: Weight training:

i. Updates weights in the back propagation network by propagating backwards the errors associated with the o/p neurons.

ii. Calculate error gradient of o/p layer and hence of neurons in the hidden layer.

Step 4: Iteration: Increase iteration by repeating steps 2 and 3 until selected error criteria is satisfied.

Issues:

Overfitting:

- Overfitting occurs when a statistical model describes random error or noise instead of the underlying relationship.
- Overfitting refers to a model that models the training data too well.
- Overfitting happens when a model learns the detail and noise in the training data to the extent that it negatively impacts the performance of the model on new data. This means that the noise or random fluctuations in the training data is picked up and

learned as concepts by the model. The problem is that these concepts do not apply to new data and negatively impact the models ability to generalize.

- Overfitting generally occurs when a model is excessively complex, such as having too many parameters relative to the number of observations.
- A model which has been overfit will generally have poor predictive performance.
- In order to avoid Overfitting, it is necessary to use additional techniques (e.g. cross validation, pruning (Pre or Post), model comparison.

- Noise in training data.
- Incomplete training data.
- Flaw in assumed theory.

Underfitting:

- It refers to a model that can neither model the training data nor generalize to new data.
- An underfit machine learning model is not a suitable model and will be obvious as it will have poor performance on the training data.
- Underfitting is often not discussed as it is easy to detect given a good performance metric. The remedy is to move on and try alternate machine learning algorithms.

How To Limit Overfitting

Both Overfitting and underfitting can lead to poor model performance. But by far the most common problem in applied machine learning is overfitting.

Overfitting is such a problem because the evaluation of machine learning algorithms on training data is different from the evaluation we actually care the most about, namely how well the algorithm performs on unseen data.

There are two important techniques that you can use when evaluating machine learning algorithms to limit overfitting:

1. Use a resampling technique to estimate model accuracy.
2. Hold back a validation dataset.

The most popular resampling technique is k-fold cross validation. It allows you to train and test your model k-times on different subsets of training data and build up an estimate of the performance of a machine learning model on unseen data.

A validation dataset is simply a subset of your training data that you hold back from your machine learning algorithms until the very end of your project. After you have selected and tuned your machine learning algorithms on your training dataset you can evaluate the learned

models on the validation dataset to get a final objective idea of how the models might perform on unseen data.

Using cross validation is a gold standard in applied machine learning for estimating model accuracy on unseen data. If you have the data, using a validation dataset is also an excellent practice.

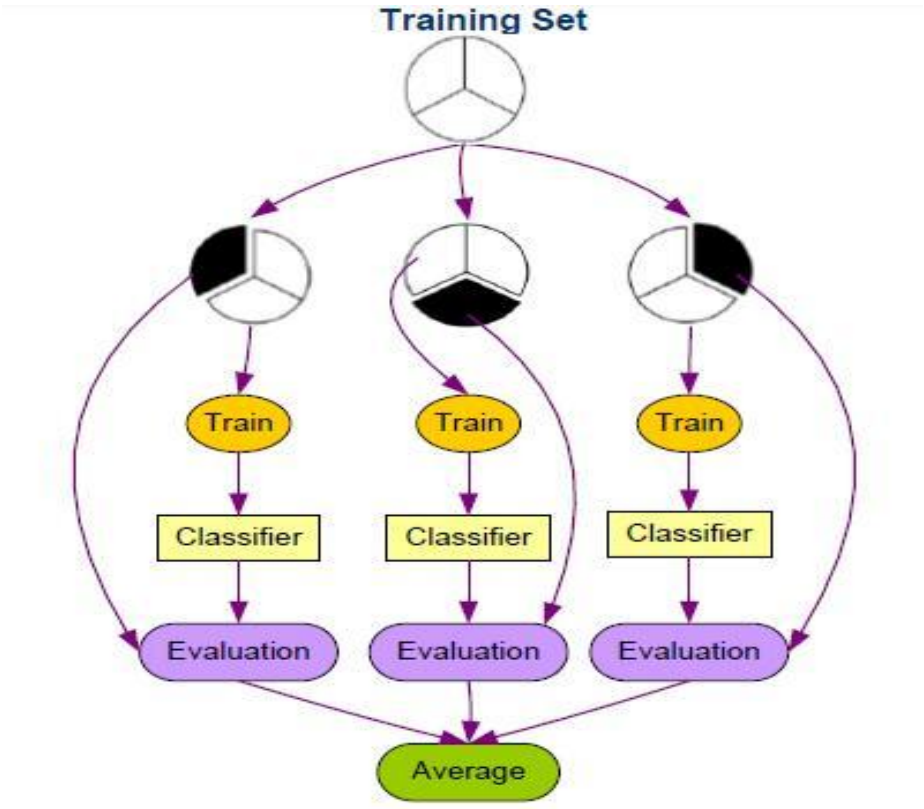
Cross Validation (The holdout method)

Data set divided into two groups. Training set: used to train the classifier and Test set: used to estimate the error rate of the trained classifier

Total number of examples = Training Set +Test

Set **K-Fold Cross-Validation**

- K-Fold Cross validation is similar to Random Sub sampling.
- Create a K-fold partition of the dataset, For each of K experiments, use K-1 folds for training and the remaining one for testing.
- The advantage of K-Fold Cross validation is that all the examples in the dataset are eventually used for both training and testing.
- The true error is estimated as the average error rate



Leave-one-out Cross-Validation

- Leave-one-out is the degenerate case of K-Fold Cross Validation, where K is chosen as the total number of examples where one sample is left out at each experiment.

Model Comparison:

- Models can be evaluated based on the output using different method

: i. Confusion Matrix

ii. ROC Analysis

Confusion Matrix (Contigency Table):

- A confusion matrix contains information about actual and predicted classifications done by classifier.
- Performance of such system is commonly evaluated using data in the matrix.
- It is also known as a contingency table or an error matrix, is a specific table layout that allows visualization of the performance of an algorithm.
- Each column of the matrix represents the instances in a predicted class, while each row represents the instances in an actual class.

	Predicted Positive	**Predicted Negative**
Positive Examples	True Positive (TP)	False Negative (FN)
Negative Examples	False Positive (FP)	True Negative (TN)

Accuracy: The accuracy of a classifier on a given test set is the percentage of test set tuples that are correctly classified by the classifier. This is also referred to as the overall recognition rate of the classifier, that is, it reflects how well the classifier recognizes tuples of the various classes.

Accuracy: (TP + TN) / Total data count (P)

Error rate: error rate or misclassification rate of a classifier, M, which is simply 1-Acc(M), where Acc(M) is the accuracy of M.

Sensitivity (Recall or True positive rate): Sensitivity (SN) is calculated as the number of correct positive predictions divided by the total number of positives. It is also called recall (REC) or true positive rate (TPR). The best sensitivity is 1.0, whereas the worst is 0.0.

Sensitivity=TP/TP+FN

Specificity (True negative rate): Specificity (SP) is calculated as the number of correct negative predictions divided by the total number of negatives. It is also called true negative rate (TNR). The best specificity is 1.0, whereas the worst is 0.0.

=TN/FP+TN

Precision (Positive predictive value): Precision (PREC) is calculated as the number of correct positive predictions divided by the total number of positive predictions. It is also called positive predictive value (PPV). The best precision is 1.0, whereas the worst is 0.0.

Precision: TP / (TP + FP) or TN/ (TN + FN)

ROC Analysis

- Receiver Operating Characteristic (ROC), or ROC curve, is a graphical plot that illustrates the performance of a binary classifier system as its discrimination threshold is varied.
- The curve is created by plotting the true positive rate against the false positive rate at various threshold settings.