UNIT-4

Classification: Basic Concepts – Basic concepts, Decision Tree Induction, Bayes Classification Methods, Rule based Classification, Model evaluation and Selection, Techniques to improve Classification Accuracy.

Classification: Basic Concepts

- What Is Classification?
- General Approach to Classification

What Is Classification?

Classification is to identify the category or the class label of a new observation. First, a set of data is used as training data. The set of input data and the corresponding outputs are given to the algorithm. So, the training data set includes the input data and their associated class labels.

Using the training dataset, the algorithm derives a model or the classifier. The derived model can be a decision tree, mathematical formula, or a neural network. In classification, when unlabeled data is given to the model, it should find the class to which it belongs.

The new data provided to the model is the test data set.

Classification is the process of classifying a record. One simple example of classification is to check whether it is raining or not. The answer can either be yes or no. So, there is a particular number of choices. Sometimes there can be more than two classes to classify. That is called *multiclass classification*.

The bank needs to analyze whether giving a loan to a particular customer is risky or not.

For example, based on observable data for multiple loan borrowers, a classification model may be established that forecasts credit risk. The data could track job records, homeownership or leasing, years of residency, number, type of deposits, historical credit ranking, etc. The goal would be credit ranking, the predictors would be the other characteristics, and the data would represent a case for each consumer. In this example, a model is constructed to find the categorical label. The labels are risky or safe.

General Approach to Classification:

"How does classification work?"

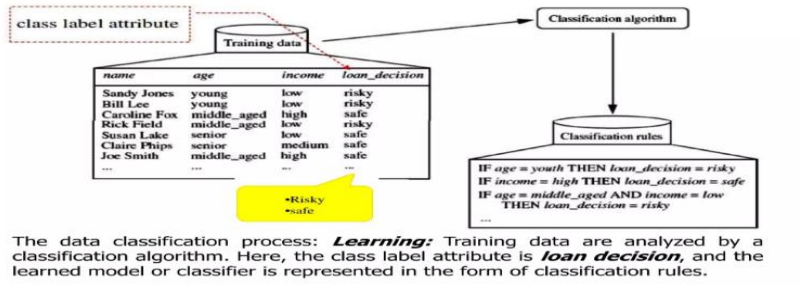
Data classification is a two-step process,

- Learning step (where a classification model is constructed)
- Classification step (where the model is used to predict class labels for given data).

The process is shown for the loan application data of Figure 8.1. (The data are simplified for illustrative purposes. In reality, we may expect many more attributes to be considered.

Classification Process (1): Model Construction

Example: Loan application



The data classification process: ***Learning:*** Training data are analyzed by a classification algorithm. Here, the class label attribute is ***loan decision,*** and the learned model or classifier is represented in the form of classification rules.

Learning step:

A classifier is built describing a predetermined set of data classes or concepts. In learning step (or training phase), where a classification algorithm builds the classifier by analyzing or "learning from" a training set made up of database tuples and their associated class labels.

A tuple, X, is represented by an n-dimensional attribute vector, X = (x1, x2,..., xn), depicting n measurements made on the tuple from n database attributes, respectively, A1, A2,..., An.

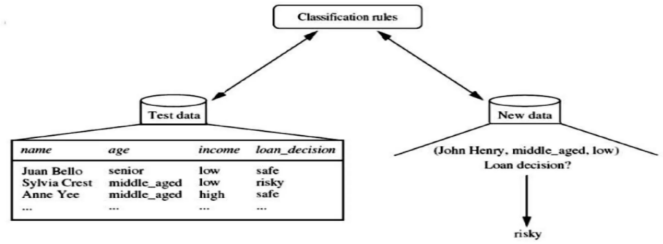
Each tuple, X, is assumed to belong to a predefined class as determined by another database attribute called the class label attribute.

In the context of classification, data tuples can be referred to as samples, examples, instances, data points, or objects.

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).

classification process can also be viewed as the learning of a mapping or function, y = f (X), that can predict the associated class label y of a given tuple X.

Classification Process (2): Use the Model in Prediction



Classification: Test data are used to estimate the accuracy of the classification rules. If the accuracy is considered acceptable, the rules can be applied to the classification of new data tuples.

In this view, we wish to learn a mapping or function that separates the data classes.

In our example, the mapping is represented as classification rules that identify loan applications as being either safe or risky.

The rules can be used to categorize future data tuples, as well as provide deeper insight into the data contents. They also provide a compressed data representation.

Classification:

"What about classification accuracy?"

In the second step (Figure 8.1b), the model is used for classification.

First, the predictive accuracy of the classifier is estimated. If we were to use the training set to measure the classifier's accuracy, this estimate would likely be optimistic, because the classifier tends to overfit the data (i.e., during learning it may incorporate some particular anomalies of the training data that are not present in the general data set overall).

Therefore, a test set is used, made up of test tuples and their associated class labels. They are independent of the training tuples, meaning that they were not used to construct the classifier**.**

The accuracy of a classifier on a given test set is the percentage of test set tuples that are correctly classified by the classifier.

The associated class label of each test tuple is compared with the learned classifier's class prediction for that tuple.

If the accuracy of the classifier is considered acceptable, the classifier can be used to classify future data tuples for which the class label is not known.

For example, the classification rules learned in Figure 8.1(a) from the analysis of data from previous loan applications can be used to approve or reject new or future loan applicants.

Decision Tree Induction:

- Attribute Selection Measures
- Other Attribute Selection Measures
- Tree Pruning
- Scalability and Decision Tree Induction

Decision Tree Induction:

Decision Tree:

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.

Decision Tree is a supervised learning method used in data mining for classification and regression methods. It is a tree that helps us in decision-making purposes.

Decision tree induction is the learning of decision trees from class-labeled training tuples.

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. Some decision tree algorithms produce only binary trees (where each internal node branches to exactly two other nodes), whereas others can produce nonbinary trees**.**

"How are 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 multidimensional data. The learning and classification steps of decision tree induction are simple and fast. In general, decision tree classifiers have good accuracy. Decision tree induction algorithms have been used for classification in many application areas such as medicine, manufacturing and production, financial analysis, astronomy, and molecular biology.

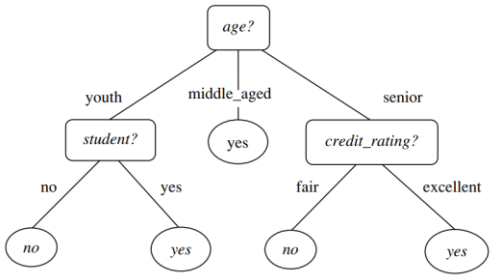


Fig: A decision tree for the concept buys computer

Decision Tree Algorithms:

- ID3 (Iterative Dichotomiser),
- C4.5, and
- Classification and Regression Trees (CART)

adopt a greedy (i.e., nonbacktracking) approach in which decision trees are constructed in a top-down recursive divide-and-conquer manner

Algorithm for learning decision trees:

The algorithm is called with three parameters:

- D, attribute list, and Attribute selection method.
- We refer to D as a data partition. Initially, it is the complete set of training tuples and their associated class labels.
- 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.

The tree starts as a single node, N, representing the training tuples in D (step 1)

Basic algorithm for inducing a decision tree from training tuples.

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).

Basic algorithm for inducing a decision tree from training tuples.:

Algorithm: Generate_decision_tree. Generate a decision tree from the training tuples of data partition, *D*.

Input:

- Data partition, *D*, which is a set of training tuples and their associated class labels;
- *attribute_list*, the set of candidate attributes;
- *Attribute_selection_method*, a procedure to determine the splitting criterion that "best" partitions the data tuples into individual classes. This criterion consists of a *splitting_attribute* and, possibly, either a *split-point* or *splitting subset*.

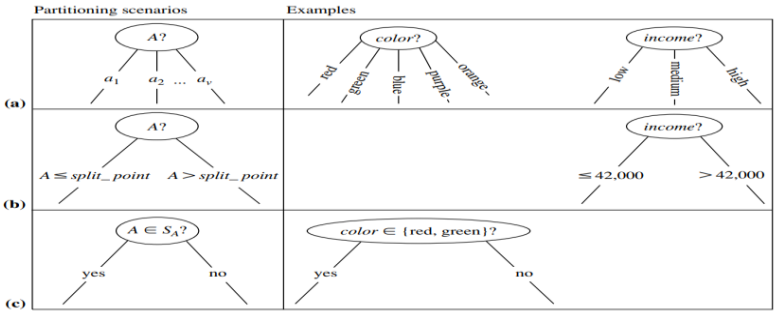
Output: A decision tree.

Method:

```
(1)  create a node N;
(2)  if tuples in D are all of the same class, C, then
(3)      return N as a leaf node labeled with the class C;
(4)  if attribute_list is empty then
(5)      return N as a leaf node labeled with the majority class in D; // majority voting
(6)  apply Attribute_selection_method(D, attribute_list) to find the "best" splitting_criterion;
(7)  label node N with splitting_criterion;
(8)  if splitting_attribute is discrete-valued and
         multiway splits allowed then // not restricted to binary trees
(9)      attribute_list ← attribute_list − splitting_attribute; // remove splitting_attribute
(10) for each outcome j of splitting_criterion
     // partition the tuples and grow subtrees for each partition
(11)     let Dj be the set of data tuples in D satisfying outcome j; // a partition
(12)     if Dj is empty then
(13)         attach a leaf labeled with the majority class in D to node N;
(14)     else attach the node returned by Generate_decision_tree(Dj, attribute_list) to node N;
     endfor
(15) return N;
```

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 splitting criterion also tells us which branches to grow from node N with respect to the outcomes of the chosen test.

There are three possible scenarios, as illustrated in the following figure. Let A be the splitting attribute. A has v distinct values, {a1, a2,..., av }, based on the training data.



1. 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.
2. 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.
3. 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.

The algorithm uses the same process recursively to form a decision tree for the tuples at each resulting partition, Dj , of D (step 14).

The recursive partitioning stops only when any one of the following terminating conditions is true:

- All the tuples in partition D (represented at node N) belong to the same class (steps 2 and 3).
- There are no remaining attributes on which the tuples may be further partitioned (step 4). In this case, majority voting is employed (step 5). This involves converting node N into a leaf and labeling it with the most common class in D. Alternatively, the class distribution of the node tuples may be stored.
- There are no tuples for a given branch, that is, a partition Dj is empty (step 12). In this case, a leaf is created with the majority class in D (step 13). The resulting decision tree is returned (step 15)

Attribute Selection Measures:

Attribute selection measures are also known as splitting rules because they determine how the tuples at a given node are to be split.

- **Attribute selection measures**
  - Used to select the attribute that best partitions the tuples into distinct classes.
  - Information gain, Gain Ratio, Gini Index
- A decision tree algorithm is known as **ID3** (Iterative Dichotomiser). ← Info Gain
- **C4.5** algorithm (successor of ID3) benchmark to newer supervised learning algorithms ← Gain Ratio
- Classification and Regression Trees (**CART**) ← Gini Index
- Adopt a greedy (i.e., nonbacktracking) approach in which decision trees are constructed in a ***top-down*** recursive ***divide-and-conquer*** manner

Entropy:

Entropy refers to a common way to measure impurity. In the decision tree, it measures the randomness or impurity in data sets.

Information Gain:

Information gain or **IG** is a statistical property that measures how well a given attribute separates the training examples according to their target classification. Constructing a decision tree is all about finding an attribute that returns the highest information gain and the smallest entropy.

Let node N represent 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

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

where pi is the nonzero 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 just the average amount of information needed to identify the class label of a tuple in D. Note that, at

this point, the information we have is based on the proportions of tuples of each class. 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. These partitions would correspond to the branches grown from node N.

How much more information would we still need (after the partitioning) to arrive at an exact classification? This amount is measured by

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

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).

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.

"**But how can we compute the information gain of an attribute that is continuousvalued,** unlike in the example?" Suppose, instead, that we have an attribute A that is continuous-valued, rather than discrete-valued. (For example, suppose that instead of the discretized version of age from the example, we have the raw values for this attribute.) For such a scenario, we must determine the "best" split-point for A, where the split-point is a threshold on A.

We first sort the values of A in increasing order. Midpoint between each pair of adjacent values is considered as a possible split-point.

Therefore, given v values of A, then v − 1 possible splits are evaluated. For example, the midpoint between the values a_i and a_{i+1} of A is

$$\frac{a_i + a_{i+1}}{2}.$$

The point with the minimum expected information requirement for A is selected as the split point for A. D1 is the set of tuples in D satisfying A ≤ split point, and D2 is the set of tuples in D satisfying A > split point.

The information gain measure is biased toward tests with many outcomes.

Gain Ratio:

Information gain is biased towards choosing attributes with a large number of values as root nodes. It means it prefers the attribute with a large number of distinct values.

C4.5, an improvement of ID3, uses Gain ratio which is a modification of Information gain that reduces its bias and is usually the best option. Gain ratio overcomes the problem with information gain by taking into account the number of branches that would result before making the split. It corrects information gain by taking the intrinsic information of a split into account.

It applies a kind of normalization to information gain using a "split information" value defined analogously with Info(D) as

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

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

The gain ratio is defined as

$$GainRatio(A) = \frac{Gain(A)}{SplitInfo_A(D)}.$$

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

Gini Index: Gini index as a cost function used to evaluate splits in the dataset. It is calculated by subtracting the sum of the squared probabilities of each class from one. It favors larger partitions and easy to implement whereas information gain favors smaller partitions with distinct values.

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

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

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.
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

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

For each attribute, each of the possible binary splits is considered. For a discrete-valued attribute, the subset that gives the minimum Gini index for that attribute is selected as its splitting subset.
For continuous-valued attributes, each possible split-point must be considered. The strategy is similar to that described earlier for information gain, where the midpoint between each pair of (sorted) adjacent values is taken as a possible split-point. The point giving the minimum Gini index for a given (continuous-valued) attribute is taken as the split-point of that attribute. Recall that for a possible split-point of A, D1 is the set of tuples in D satisfying A ≤ split point, and D2 is the set of tuples in D satisfying A > split point.
The reduction in impurity that would be incurred by a binary split on a discrete- or continuous-valued attribute A is

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

The attribute that maximizes the reduction in impurity (or, equivalently, has the minimum Gini index) is selected as the splitting attribute. This attribute and either its splitting subset (for a discrete-valued splitting attribute) or split-point (for a continuous-valued splitting attribute) together form the splitting criterion.

Other Attribute Selection Measures:

Many other attribute selection measures have been proposed. CHAID, a decision tree algorithm that is popular in marketing, uses an attribute selection measure that is based on the statistical χ^2 test for independence. Other measures include C-SEP (which performs better than information gain and the Gini index in certain cases) and G-statistic (an information theoretic measure that is a close approximation to χ^2 distribution).
Attribute selection measures based on the Minimum Description Length (MDL) principle have the least bias toward multivalued attributes. MDL-based measures use encoding techniques to define the "best" decision tree as the one that requires the fewest number of bits to both (1) encode the tree and (2) encode the exceptions to the tree.
Other attribute selection measures consider multivariate splits (i.e., where the partitioning of tuples is based on a combination of attributes, rather than on a single attribute). The CART system, for example, can find multivariate splits based on a linear combination of attributes. Multivariate splits are a form of attribute (or feature) construction, where new attributes are created based on the existing ones.
"Which attribute selection measure is the best?" All measures have some bias. It has been shown that the time complexity of decision tree induction generally increases exponentially with tree height. Hence, measures that tend to produce shallower trees (e.g., with multiway rather than binary splits, and that favor more balanced splits) may be preferred.

Tree Pruning:

When a decision tree is built, many of the branches will reflect anomalies in the training data due to noise or outliers. Tree pruning methods address this problem of overfitting the data. Such methods typically use statistical measures to remove the least-reliable branches. They are usually faster and better at correctly classifying independent test data (i.e., of previously unseen tuples) than unpruned trees.

"How does tree pruning work?"

Overfitting is a practical problem while building a decision tree model. The model is having an issue of overfitting is considered when the algorithm continues to go deeper and deeper in the to reduce the training set error but results with an increased test set error i.e, Accuracy of prediction for our model goes down. It generally happens when it builds many branches due to outliers and irregularities in data. Two approaches which we can use to avoid overfitting are:

There are two common approaches to tree pruning: prepruning and postpruning.

prepruning :

In the prepruning approach, a tree is "pruned" by halting its construction early (e.g., by deciding not to further split or partition the subset of training tuples at a given node). Upon halting, the node becomes a leaf. The leaf may hold the most frequent class among the subset tuples or the probability distribution of those tuples.
When constructing a tree, measures such as statistical significance, information gain, Gini index, and so on, can be used to assess the goodness of a split. If partitioning the tuples at a node would result in a split that falls

below a pre specified threshold, then further partitioning of the given subset is halted. There are difficulties, however, in choosing an appropriate threshold. High thresholds could result in oversimplified trees, whereas low thresholds could result in very little simplification.

Postpruning:

postpruning, which removes subtrees from a "fully grown" tree. A subtree at a given node is pruned by removing its branches and replacing it with a leaf. The leaf is labeled with the most frequent class among the subtree being replaced. For example, notice the subtree at node "A3?" in the unpruned tree shown in following Figure. Suppose that the most common class within this subtree is "class B." In the pruned version of the tree, the subtree in question is pruned by replacing it with the leaf "class B."

The cost complexity pruning algorithm used in CART is an example of the postpruning approach. This approach considers the cost complexity of a tree to be a function of the number of leaves in the tree and the error rate of the tree (where the error rate is the percentage of tuples misclassified by the tree). It starts from the bottom of the tree. For each internal node, N, it computes the cost complexity of the subtree at N, and the cost complexity of the subtree at N if it were to be pruned (i.e., replaced by a leaf node). The two values are compared. If pruning the subtree at node N would result in a smaller cost complexity, then the subtree is pruned. Otherwise, it is kept..

A pruning set of class-labeled tuples is used to estimate cost complexity. This set is independent of the training set used to build the unpruned tree and of any test set used for accuracy estimation.

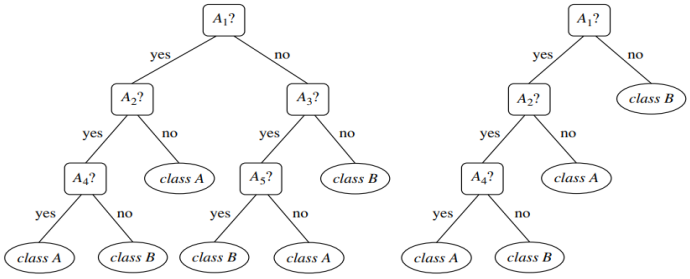


Figure : unpruned decision tree and a pruned version of it.

C4.5 uses a method called pessimistic pruning, which is similar to the cost complexity method in that it also uses error rate estimates to make decisions regarding subtree pruning. Pessimistic pruning, however, does not require the use of a prune set. Instead, it uses the training set to estimate error rates. Recall that an estimate of accuracy or error based on the training set is overly optimistic and, therefore, strongly biased. The pessimistic pruning method therefore adjusts the error rates obtained from the training set by adding a penalty, so as to counter the bias incurred.

Rather than pruning trees based on estimated error rates, we can prune trees based on the number of bits required to encode them. The "best" pruned tree is the one that minimizes the number of encoding bits. This method adopts the MDL principle.

Decision trees can suffer from repetition and replication making them to interpret. Repetition occurs when an attribute is repeatedly tested along a given branch of the tree (e.g., "age < 60?," followed by "age < 45?," and so on).

In replication, duplicate subtrees exist within the tree. The use of multivariate splits (splits based on a combination of attributes) can prevent these problems. Another approach is to use a different form of knowledge representation, such as rules, instead of decision trees.

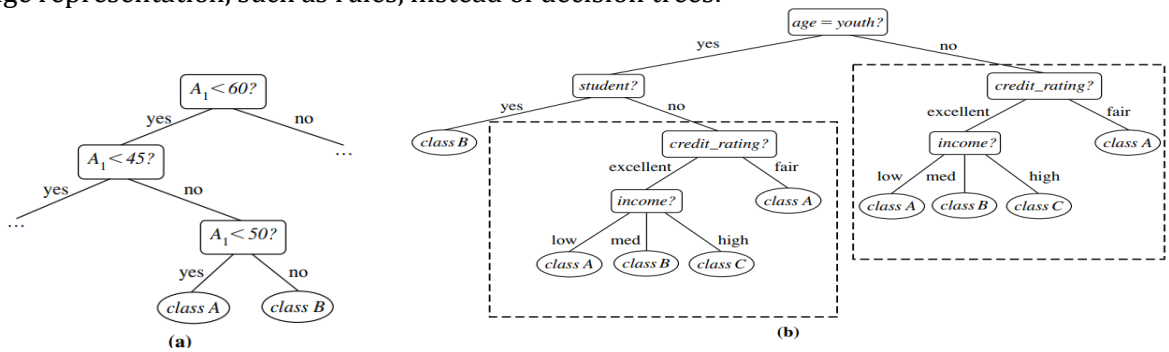


Fig: An example of: (a) subtree repetition, where an attribute is repeatedly tested along a given branch of the tree (e.g., age) and (b) subtree replication

Scalability and Decision Tree Induction:

The efficiency of existing decision tree algorithms, such as ID3, C4.5, and CART, has been well established for relatively small data sets. Efficiency becomes an issue of concern when these algorithms are applied to the

mining of very large real-world databases.
More scalable approaches, capable of handling training data that are too large to fit in memory, are required.
Several scalable decision tree induction methods have been introduced in recent studies:

- RainForest
- BOAT (Bootstrapped Optimistic Algorithm for Tree construction)

RainForest:

- The method maintains an AVC-set (where "AVC" stands for "Attribute-Value, Classlabel") for each attribute, at each tree node, describing the training tuples at the node.
- The AVC-set of an attribute A at node N gives the class label counts for each value of A for the tuples at N.
- The set of all AVC-sets at a node N is the AVC-group of N.
- The size of an AVC-set for attribute A at node N depends only on the number of distinct values of A and the number of classes in the set of tuples at N. Typically, this size should fit in memory, even for real-world data.
- RainForest also has techniques, however, for handling the case where the AVC-group does not fit in memory. Therefore, the method has high scalability for decision tree induction in very large data sets.

Fig: AVC-set for attribute

The use of data structures to hold aggregate information regarding the training data

	buys_computer	
age	yes	no
youth	2	3
middle_aged	4	0
senior	3	2

	buys_computer	
income	yes	no
low	3	1
medium	4	2
high	2	2

	buys_computer	
student	yes	no
yes	6	1
no	3	4

	buys_computer	
credit_ratting	yes	no
fair	6	2
excellent	3	3

for example, adapts to the amount of main memory available and applies to any decision tree induction algorithm.

BOAT (Bootstrapped Optimistic Algorithm for Tree construction):

- BOAT (Bootstrapped Optimistic Algorithm for Tree construction) is a decision tree algorithm that takes a completely different approach to scalability—it is not based on the use of any special data structures.
- Instead, it uses a statistical technique known as "bootstrapping" to create several smaller samples (or subsets) of the given training data, each of which fits in memory. Each subset is used to construct a tree, resulting in several trees.
- The trees are examined and used to construct a new tree, T 0 , that turns out to be "very close" to the tree that would have been generated if all the original training data had fit in memory.
- BOAT can use any attribute selection measure that selects binary splits and that is based on the notion of purity of partitions such as the Gini index.
- BOAT usually requires only two scans of D. This is quite an improvement, even in comparison to traditional decision tree algorithms which require one scan per tree level! BOAT was found to be two to three times faster than RainForest, while constructing exactly the same tree.

An additional advantage of BOAT is that it can be used for incremental updates. That is, BOAT can take new insertions and deletions for the training data and update the decision tree to reflect these changes, without having to reconstruct the tree from scratch.

Visual Mining for Decision Tree Induction:

- Perception-based classification (PBC) is an interactive approach based on multidimensional visualization techniques and allows the user to incorporate background knowledge about the data when building a decision tree.
- By visually interacting with the data, the user is also likely to develop a deeper understanding of the data. The resulting trees tend to be smaller than those built using traditional decision tree induction methods and so are easier to interpret, while achieving about the same accuracy.

"How can the data be visualized to support interactive decision tree construction?"

PBC uses a pixel-oriented approach to view multidimensional data with its class label information.
The circle segments approach is adapted, which maps d-dimensional data objects to a circle that is partitioned into d segments, each representing one attribute Here, an attribute value of a data object is mapped to one colored pixel, reflecting the object's class label. This mapping is done for each attribute–value pair of each data object. Sorting is done for each attribute to determine the arrangement order within a segment.

For example, attribute values within a given segment may be organized so as to display homogeneous (with respect to class label) regions within the same attribute value. The amount of training data that can be visualized. at one time is approximately determined by the product of the number of attributes and the number of data objects.

The PBC system displays a split screen, consisting of a Data Interaction window and a Knowledge Interaction window. A screenshot of PBC, a system for interactive decision tree construction.

The Data Interaction window displays the circle segments of the data under examination, while the Knowledge Interaction window displays the decision tree constructed so far. Initially, the complete training set is visualized in the Data Interaction window, while the Knowledge Interaction window displays an empty decision tree.

Traditional decision tree algorithms allow only binary splits for numeric attributes. PBC, however, allows the user to specify multiple split-points, resulting in multiple branches to be grown from a single tree node.

Bayes Classification Methods:

"What are Bayesian classifiers?" Bayesian classifiers are statistical classifiers. They can predict class membership probabilities such as the probability that a given tuple belongs to a particular class

- **Bayes' Theorem**
- **Na¨ıve Bayesian Classification**

Bayes' Theorem:

Bayes theorem helps to determine the probability of an event with random knowledge. It is used to calculate the probability of occurring one event while other one already occurred. It is a best method to relate the condition probability and marginal probability.

Let H be some hypothesis such as that the data tuple X belongs to a specified class C. For classification problems, we want to determine P(H|X), the probability that the hypothesis H holds given the "evidence" or observed data tuple X. In other words, we are looking for the probability that tuple X belongs to class C, given that we know the attribute description of X.

P(H|X) is the posterior probability, or a posteriori probability, of H conditioned on X.

For example, suppose our world of data tuples is confined to customers described by the attributes age and income, respectively, and that X is a 35-year-old customer with an income of $40,000. Suppose that H is the hypothesis that our customer will buy a computer. Then P(H|X) reflects the probability that customer X will buy a computer given that we know the customer's age and income.

In contrast, **P(H) is the prior probability**, or a priori probability, of H. For our example, this is the probability that any given customer will buy a computer, regardless of age, income, or any other information, for that matter. 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. That is, it is the probability that a customer, X, is 35 years old and earns $40,000, given that we know the customer will buy a computer.

P(X) is the prior probability of X. Using our example, it is the probability that a person from our set of customers is 35 years old and earns $40,000.

"How are these probabilities estimated?" P(H), P(X|H), and P(X) may be estimated from the given data, as we shall see next. Bayes' theorem is useful in that it provides a way of calculating the posterior probability, P(H|X), from P(H), P(X|H), and P(X).

Bayes' theorem is

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

Na¨ıve Bayesian Classification:

Na¨ıve Bayesian classifiers assume that the effect of an attribute value on a given class is independent of the values of the other attributes. This assumption is called class**conditional independence.**

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

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

Thus, we maximize P(Ci |X). The class Ci for which P(Ci |X) is maximized is called the maximum posteriori hypothesis. By Bayes' theorem

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

3. As P(X) is constant for all classes, only P(X|Ci)P(Ci) needs to be maximized. Note that the 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. Now, since there are no dependence relationships among the attributes.

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

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

For each attribute, we look at whether the attribute is **categorical or continuous-valued.**

For instance, to compute P(X|Ci), we consider the following:

(a) If Ak is categorical, then P(xk |Ci) is the number of tuples of class Ci in D having the value xk for Ak , divided by |Ci,D|, the number of tuples of class Ci in D.

(b) If Ak is continuous-valued, then we need to do a bit more work, but the calculation is pretty straightforward. A continuous-valued attribute is assumed to have a Gaussian distribution with a mean μ and standard deviation σ, defined by

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

So that $P(x_k|C_i) = g(x_k, \mu_{C_i}, \sigma_{C_i})$.

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

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

In other words, the predicted class label is the class Ci for which P(X|Ci)P(Ci) is the maximum.

Bayesian classifiers have the minimum error rate in comparison to all other classifiers.

A zero probability cancels the effects of all the other (posteriori) probabilities (on Ci) involved in the product.

There is a simple trick to avoid this problem. We can assume that our training database, D, is so large that adding one to each count that we need would only make a negligible difference in the estimated probability value, yet would conveniently avoid the case of probability values of zero. This technique for probability estimation is known as the **Laplacian correction or Laplace estimator**, If we have, say, q counts to which we each add one, then we must remember to add q to the corresponding denominator used in the probability calculation.

Example: Using the Laplacian correction to avoid computing probability values of zero. Suppose that for the class buys computer = yes in some training database, D, containing 1000 tuples, we have 0 tuples with income = low, 990 tuples with income = medium, and 10 tuples with income = high. The probabilities of these events, without the Laplacian correction, are 0, 0.990 (from 990/1000), and 0.010 (from 10/1000), respectively. Using the Laplacian correction for the three quantities, we pretend that we have 1 more tuple for each income-value pair.

In this way, we instead obtain the following probabilities (rounded up to three decimal places):

$$\frac{1}{1003} = 0.001, \frac{991}{1003} = 0.988, \text{ and } \frac{11}{1003} = 0.011,$$

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

Rule-Based Classification:

We first examine how such rules are used for classification We then study ways in which they can be generated, either from a decision tree or directly from the training data using a sequential covering algorithm.

- Using IF-THEN Rules for Classification
- Rule Extraction from a Decision Tree
- Rule Induction Using a Sequential Covering Algorithm
- Rule Quality Measures and Rule Pruning

Using IF-THEN Rules for 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 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. 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 (i.e., all 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 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

$$coverage(R) = \frac{n_{covers}}{|D|} \qquad accuracy(R) = \frac{n_{correct}}{n_{covers}}.$$

That is, a **rule's coverage** is the percentage of tuples that are covered by the rule (i.e., their 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: Rule accuracy and coverage These are classlabeled tuples from the AllElectronics customer database. Our task is to predict whether a customer will buy a computer.

Consider rule R1, 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%

Let's see how we can use rule-based classification to predict the class label of a given tuple, X. If a rule is satisfied by X, the rule is said to be triggered. For example, suppose we have

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

We would like to classify X according to buys computer. X satisfies R1, which triggers the rule. If R1 is the only rule satisfied, then the rule fires by returning the class prediction for X.

Note that triggering does not always mean firing because there may be more than one rule that is satisfied! If more than one rule is triggered, we have a potential problem.

What if they each specify a different class? Or what if no rule is satisfied by X?

We tackle the first question. If more than one rule is triggered, we need a conflict resolution strategy to figure out which rule gets to fire and assign its class prediction to X.

There are many possible strategies. We look at two, namely size ordering and rule ordering

The size ordering scheme assigns the highest priority to the triggering rule that has the "toughest" requirements, where toughness is measured by the rule antecedent size. That is, the triggering rule with the most attribute tests is fired.

The rule ordering scheme prioritizes the rules beforehand.

The ordering may be class-based or rule-based.

With class-based ordering, the classes are sorted in order of decreasing "importance" such as by decreasing order of prevalence. That is, all the rules for the most prevalent (or most frequent) class come first, the rules for the next prevalent class come next, and so on. Alternatively, they may be sorted based on the misclassification cost per class.

With rule-based ordering, the rules are organized into one long priority list, according to some measure of rule quality, such as accuracy, coverage, or size (number of attribute tests in the rule antecedent), or based on advice from domain experts. When rule ordering is used, the rule set is known as a decision list.

With rule ordering, the triggering rule that appears earliest in the list has the highest priority, and so it gets to fire its class prediction. Any other rule that satisfies X is ignored. Most rule-based classification systems use a class-based rule-ordering strategy.

Rule Extraction from a 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:

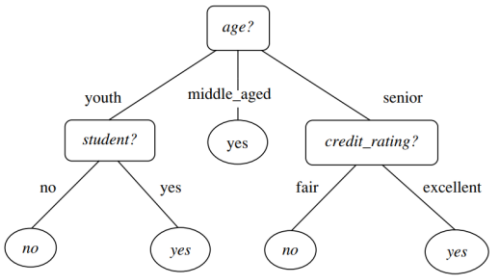
Extracting classification rules from a decision tree. The decision tree can be converted to classification IF-THEN rules by tracing the path from the root node to each leaf node in the tree. The rules extracted from Decision tree are as follows:

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

Mutually exclusive means that we cannot have rule conflicts here because no two rules will be triggered for the same tuple. (We have one rule per leaf, and any tuple can map to only one leaf.)

Exhaustive means there is one rule for each possible attribute–value combination, so that this set of rules does

not require a default rule. Therefore, the order of the rules does not matter—they are unordered.
"How can we prune the rule set?" For a given rule antecedent, any condition that does not improve the estimated accuracy of the rule can be pruned (i.e., removed), thereby generalizing the rule



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*

Rule Induction Using a Sequential Covering Algorithm:
IF-THEN rules can be extracted directly from the training data (i.e., without having to generate a decision tree first) using a sequential covering algorithm. The name comes from the notion that the rules are learned sequentially (one at a time), where each rule for a given class will ideally cover many of the class's tuples (and hopefully none of the tuples of other classes). Sequential covering algorithms are the most widely used approach to mining disjunctive sets of classification rules, and form the topic of this subsection.
There are many sequential covering algorithms. Popular variations include AQ, CN2, and the more recent RIPPER.
The general strategy is as follows:

- Rules are learned one at a time.
- Each time a rule is learned, the tuples covered by the rule are removed, and the process repeats on the remaining tuples.
- This sequential learning of rules is in contrast to decision tree induction. Because the path to each leaf in a decision tree corresponds to a rule, we can consider decision tree induction as learning a set of rules simultaneously**.**

Sequential covering algorithm:

Algorithm: Sequential covering. Learn a set of IF-THEN rules for classification.

Input:

- *D*, a data set of class-labeled tuples;
- *Att_vals*, the set of all attributes and their possible values.

Output: A set of IF-THEN rules.

Method:

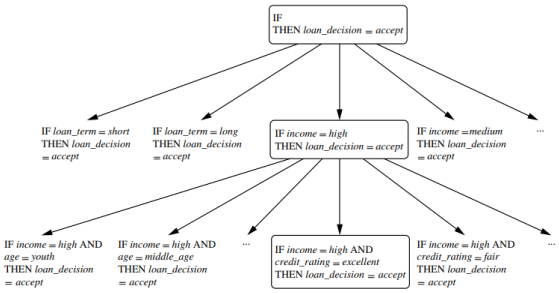
```
(1)  Rule_set = {}; // initial set of rules learned is empty
(2)  for each class c do
(3)      repeat
(4)          Rule = Learn_One_Rule(D, Att_vals, c);
(5)          remove tuples covered by Rule from D;
(6)          Rule_set = Rule_set + Rule; // add new rule to rule set
(7)      until terminating condition;
(8)  endfor
(9)  return Rule_Set;
```

The process continues until the terminating condition is met, such as when there are no more training tuples or the quality of a rule returned is below a user-specified threshold. The Learn One Rule procedure finds the "best" rule for the current class, given the current set of training tuples.
"How are rules learned?" rules are grown in a general-to-specific manner shown in the following figure.We can think of this as a beam search, where we start off with an empty rule and then gradually keep appending attribute tests to it. Suppose our training set, D, consists of loan application data.
Attributes regarding each applicant include their age, income, education level, residence, credit rating, and the term of the loan.
The classifying attribute is loan decision, which indicates whether a loan is accepted (considered safe) or rejected (considered risky). To learn a rule for the class "accept," we start off with the most general rule possible, that is, the condition of the rule antecedent is empty. The rule is

IF THEN loan decision = accept.

We then consider each possible attribute test that may be added to the rule. These can be derived from the parameter Att vals, which contains a list of attributes with their associated values. For example, for an attribute–value pair (att, val), we can consider attribute tests such as att = val, att ≤ val, att > val, and so on.
suppose Learn One Rule finds that the attribute test income = high best improves the accuracy of our current (empty) rule. We append it to the condition, so that the current rule becomes

IF income = high AND credit rating = excellent THEN loan decision = accept.



The process repeats, where at each step we continue to greedily grow rules until the resulting rule meets an acceptable quality level.

Rule Quality Measures:

Learn One Rule needs a measure of rule quality. Every time it considers an attribute test, it must check to see if appending such a test to the current rule's condition will result in an improved rule**.**

Another measure is based on information gain and was proposed in FOIL (First Order Inductive Learner), a sequential covering algorithm that learns first-order logic rules. Learning first-order rules is more complex because such rules contain variables, whereas the rules we are concerned with in this section are propositional (i.e., variablefree). In machine learning, the tuples of the class for which we are learning rules are called positive tuples, while the remaining tuples are negative.

Let pos (neg) be the number of positive (negative) tuples covered by R. Let pos' (neg') be the number of positive (negative) tuples covered by R ' .

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

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

We want to assess whether any observed differences between these two distributions may be attributed to chance. We can use the likelihood ratio statistic,

$$Likelihood\_Ratio = 2 \sum_{i=1}^{m} f_i \log \left(\frac{f_i}{e_i} \right),$$

Rule Pruning :Learn One Rule does not employ a test set when evaluating rules.

Assessments of rule quality as described previously are made with tuples from the original training data. These assessments are optimistic because the rules will likely overfit the data. That is, the rules may perform well on the training data, but less well on subsequent data.

To compensate for this, we can prune the rules. A rule is pruned by removing a conjunct (attribute test).

We choose to prune a rule, R, if the pruned version of R has greater quality, as assessed on an independent set of tuples. As in decision tree pruning, we refer to this set as a pruning set.

Various pruning strategies can be used such as the pessimistic pruning approach described in the previous section.

FOIL uses a simple yet effective method. Given a rule, R

$$FOIL\_Prune(R) = \frac{pos - neg}{pos + neg},$$

where pos and neg are the number of positive and negative tuples covered by R, respectively. This value will increase with the accuracy of R on a pruning set. Therefore, if the FOIL Prune value is higher for the pruned version of R, then we prune R.

Model Evaluation and Selection:

- Metrics for Evaluating Classifier Performance
- Holdout Method and Random Subsampling
- Cross-Validation
- Bootstrap
- Model Selection Using Statistical Tests of Significance
- Comparing Classifiers Based on Cost–Benefit and ROC Curves

Metrics for Evaluating Classifier Performance: Measures for assessing how good or how "accurate" your classifier is at predicting the class label of tuples.

We will consider the case of where the class tuples are more or less evenly distributed, as well as the case where classes are unbalanced (e.g., where an important class of interest is rare such as in medical tests). The classifier evaluation measures presented in this section are summarized as follows:

They include accuracy (also known as recognition rate), sensitivity (or recall), specificity, precision, F1, and Fβ. Using training data to derive a classifier and then estimate the accuracy of the resulting learned model can result in misleading overoptimistic estimates due to overspecialization of the learning algorithm to the data.Instead, it is better to measure the classifier's accuracy on a test set consisting of class-labeled tuples that were not used to train the model.

Measure	*Formula*
accuracy, recognition rate	$\frac{TP+TN}{P+N}$
error rate, misclassification rate	$\frac{FP+FN}{P+N}$
sensitivity, true positive rate, recall	$\frac{TP}{P}$
specificity, true negative rate	$\frac{TN}{N}$
precision	$\frac{TP}{TP+FP}$
F, F_1, F-score, harmonic mean of precision and recall	$\frac{2 \times precision \times recall}{precision + recall}$
F_β, where β is a non-negative real number	$\frac{(1+\beta^2) \times precision \times recall}{\beta^2 \times precision + recall}$

Evaluation measures.

There are four additional terms we need to know that are the "building blocks" used in computing many evaluation measures:

True positives (TP): These refer to the positive tuples that were correctly labeled by the classifier. Let TP be the number of true positives.

True negatives(TN): These are the negative tuples that were correctly labeled by the classifier. Let TN be the number of true negatives.

False positives (FP): These are the negative tuples that were incorrectly labeled as positive (e.g., tuples of class buys computer = no for which the classifier predicted buys computer = yes). Let FP be the number of false positives.

False negatives (FN): These are the positive tuples that were mislabeled as negative (e.g., tuples of class buys computer = yes for which the classifier predicted buys computer = no). Let FN be the number of false negatives.

These terms are summarized in the confusion matrix of the following Figure:

		Predicted class		
		yes	*no*	Total
Actual class	*yes*	*TP*	*FN*	*P*
	no	*FP*	*TN*	*N*
	Total	P'	N'	$P + N$

The table may have additional rows or columns to provide totals. For example, in the confusion matrix of above Figure, P and N are shown. In addition, P' is the number of tuples that were labeled as positive (TP + FP) and N0 is the number of tuples that were labeled as negative (TN + FN).

The total number of tuples is TP + TN + FP + TN, or P + N, or P' + N' .

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. That is,

$$accuracy = \frac{TP + TN}{P + N}.$$

Error rate or misclassification rate:

Error rate or misclassification rate of a classifier, M, which is simply 1 − accuracy(M), where accuracy(M) is the accuracy of M. This also can be computed as

$$error\ rate = \frac{FP + FN}{P + N}.$$

Sensitivity and Specificity:

The sensitivity and specificity measures can be used, respectively, for this purpose. Sensitivity is also referred to as the true positive (recognition) rate (i.e., the proportion of positive tuples that are correctly identified), while specificity is the true negative rate (i.e., the proportion of negative tuples that are correctly identified). These measures are defined as

$$sensitivity = \frac{TP}{P}$$

$$specificity = \frac{TN}{N}$$

It can be shown that accuracy is a function of sensitivity and specificity

$$accuracy = sensitivity\frac{P}{(P+N)} + specificity\frac{N}{(P+N)}.$$

Example: Sensitivity and specificity. Figure 8.16 shows a confusion matrix for medical data where the class values are yes and no for a class label attribute, cancer.

Classes	*yes*	*no*	*Total*	*Recognition (%)*
yes	**90**	**210**	300	30.00
no	**140**	**9560**	9700	98.56
Total	230	9770	10,000	96.40

Figure 8.16 Confusion matrix for the classes cancer = yes and cancer = no.

The sensitivity of the classifier is 90 / 300 = 30.00%.

The specificity is 9560 / 9700 = 98.56%.

The classifier's overall accuracy is 9650 10,000 = 96.50%. Thus, we note that although the classifier has a high accuracy, it's ability to correctly label the positive (rare) class is poor given its low sensitivity. It has high specificity, meaning that it can accurately recognize negative tuples.

Precision and Recall Measures:

The precision and recall measures are also widely used in classification.

Precision can be thought of as a measure of exactness (i.e., what percentage of tuples labeled as positive are actually such), whereas recall is a measure of completeness (what percentage of positive tuples are labeled as such). If recall seems familiar, that's because it is the same as sensitivity (or the true positive rate).

These measures can be computed as

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN} = \frac{TP}{P}.$$

Example : Precision and recall.

The precision of the classifier in Figure 8.16 for the yes class is 90 /230 = 39.13%.

The recall is 90 300 = 30.00%, which is the same calculation for sensitivity.

A perfect precision score of 1.0 for a class C means that every tuple that the classifier labeled as belonging to class C does indeed belong to class C

An alternative way to use precision and recall is to combine them into a single measure. This is the approach of the F measure (also known as the F1 score or F-score) an the Fβ measure. They are defined as

$$F = \frac{2 \times precision \times recall}{precision + recall} \qquad F_\beta = \frac{(1+\beta^2) \times precision \times recall}{\beta^2 \times precision + recall},$$

where β is a non-negative real number.

The F measure is the harmonic mean of precision and recall (the proof of which is left as an exercise). It gives equal weight to precision and recall.

The Fβ measure is a weighted measure of precision and recall. It assigns β times as much weight to recall as to precision.

Commonly used Fβ measures are F2 (which weights recall twice as much as precision) and F0.5 (which weights precision twice as much as recall).

In addition to accuracy-based measures, classifiers can also be compared with respect to the following additional aspects:

Speed: This refers to the computational costs involved in generating and using the given classifier.

Robustness: This is the ability of the classifier to make correct predictions given noisy data or data with missing values. Robustness is typically assessed with a series of synthetic data sets representing increasing degrees of noise and missing values.

Scalability: This refers to the ability to construct the classifier efficiently given large amounts of data. Scalability is typically assessed with a series of data sets of increasing size.

Interpretability: This refers to the level of understanding and insight that is provided by the classifier or predictor. Interpretability is subjective and therefore more difficult to assess. Decision trees and classification rules can be easy to interpret, yet their interpretability may diminish the more they become complex

Holdout Method and Random Subsampling:

Holdout Method:

In this method, the given data are randomly partitioned into two independent sets, a training set and a test set. two-thirds of the data are allocated to the training set, and the remaining one-third is allocated to the test set. The training set is used to derive the model. The model's accuracy is then estimated with the test set .The estimate is pessimistic because only a portion of the initial data is used to derive the model.

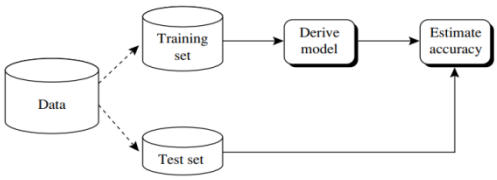


Fig: Estimating accuracy with the holdout method

Random subsampling is a variation of the holdout method in which the holdout method is repeated k times. The overall accuracy estimate is taken as the average of the accuracies obtained from each iteration.

Cross-Validation:

In k-fold cross-validation, the initial data are randomly partitioned into k mutually exclusive subsets or "folds," D1, D2,..., Dk , each of approximately equal size.

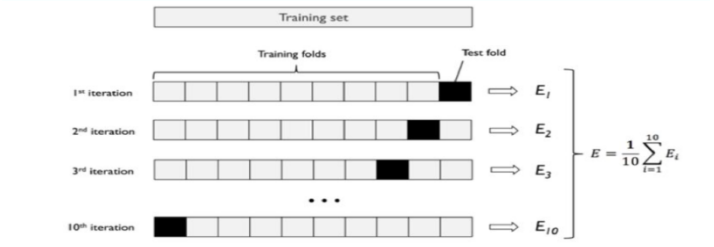
Training and testing is performed k times. In iteration i, partition Di is reserved as the test set, and the remaining partitions are collectively used to train the model. That is, in the first iteration, subsets D2,..., Dk collectively serve as the training set to obtain a first model, which is tested on D1; the second iteration is trained on subsets D1, D3,..., Dk and tested on D2; and so on.

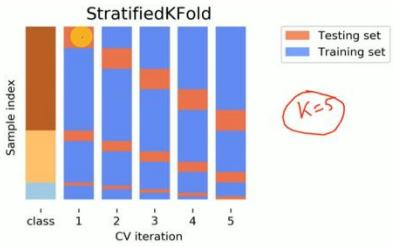
Unlike the holdout and random subsampling methods, here each sample is used the same number of times for training and once for testing.

- Randomly split the training dataset into *k folds without* replacement
  - *k-1 folds are used for the model training*
  - *one fold is used for performance evaluation.*
- Procedure is repeated *k times so that we obtain k* models and performance estimates.

For classification, the accuracy estimate is the overall number of correct classifications from the k iterations, divided by the total number of tuples in the initial data.

Leave-one-out is a special case of k-fold cross-validation where k is set to the number of initial tuples. That is, only one sample is "left out" at a time for the test set. In **stratified cross-validation**, the folds are stratified so that the class distribution of the tuples in each fold is approximately the same as that in the initial data. In general, stratified 10-fold cross-validation is recommended for estimating accuracy due to its relatively low bias and variance.





Bootstrap: Unlike the accuracy estimation methods just mentioned, the bootstrap method samples the given training tuples uniformly with replacement.

That is, each time a tuple is selected, it is equally likely to be selected again and re-added to the training set.

For instance, imagine a machine that randomly selects tuples for our training set.

In sampling with replacement, the machine is allowed to select the same tuple more than once.

There are several bootstrap methods.

A commonly used one is the .632 bootstrap

We can repeat the sampling procedure k times, where in each iteration, we use the current test set to obtain an accuracy estimate of the model obtained from the current bootstrap sample. The overall accuracy of the model, M, is then estimated as

$$Acc(M) = \frac{1}{k}\sum_{i=1}^{k}(0.632 \times Acc(M_i)_{test\_set} + 0.368 \times Acc(M_i)_{train\_set}),$$

where $Acc(Mi)_{test\ set}$ is the accuracy of the model obtained with bootstrap sample i when it is applied to test set i. $Acc(Mi)_{train\ set}$ is the accuracy of the model obtained with bootstrap sample i when it is applied to the original set of data tuples. Bootstrapping tends to be overly optimistic. It works best with small data sets.

Model Selection Using Statistical Tests of Significance:

- Significance tests and ROC curves are useful tools for model selection.
- Significance tests can be used to assess whether the difference in accuracy between two classifiers is due to chance.
- ROC curves plot the true positive rate (or sensitivity) versus the false positive rate (or 1 – specificity) of

one or more classifiers.

To determine if there is any "real" difference in the mean error rates of two models, we need to employ a test of statistical significance. In addition, we want to obtain some confidence limits for our mean error rates so that we can make statements like, "Any observed mean will not vary by ± two standard errors 95% of the time for future samples" or "One model is better than the other by a margin of error of ± 4%."

What do we need to perform the statistical test?

Suppose that for each model, we did 10-fold cross-validation, say, 10 times, each time using a different 10-fold data partitioning.

Each partitioning is independently drawn. We can average the 10 error rates obtained each for M1 and M2, respectively, to obtain the mean error rate for each model.

For a given model, the individual error rates calculated in the cross-validations may be considered as different, independent samples from a probability distribution. In general, they follow a t-distribution with k − 1 degrees of freedom where, here, k = 10. (This distribution looks very similar to a normal, or Gaussian, distribution even though the functions defining the two are quite different.

Both are unimodal, symmetric, and bellshaped.)

t-test:

This allows us to do hypothesis testing where the significance test used is the t-test, or Student'st-test.

A t-test is a type of inferential statistic test used to determine if there is a significant difference between the means of two groups. It is often used when data is normally distributed and population variance is unknown. The t-test is used in hypothesis testing to assess whether the observed difference between the means of the two groups is statistically significant or just due to random variation.

The most used key terms in T-test are as follows:

- **T-statistic:** The t-statistic is a measure of the difference between the means of two groups relative to the variability within each group. It is calculated as the difference between the sample means divided by the standard error of the difference. It is also known as the t-value or t-score
    - If the t-value is large => the two groups belong to different groups.
    - If the t-value is small => the two groups belong to the same group.
- **T-Distribution:** The t-distribution, commonly known as the Student's t-distribution.It is employed in statistical inference when working with small sample sizes and population standard deviations are unknown. The t-distribution gets closer to the normal distribution as the sample size rises. It plays a crucial role in hypothesis testing and estimating population parameters with limited data.
- **Degree of freedom (df):** The degree of freedom represents the number of values in a calculation that is free to vary. The degree of freedom (df) tells us the number of independent variables used for calculating the estimate between 2 sample groups.
  In a t-test, the degree of freedom is calculated as the total sample size minus 1 i.e

$$df = \sum n_s - 1$$

where "ns" is the number of observations in the sample. It reflects the number of values in the sample that are free to vary after estimating the sample mean.

Suppose, we have 2 samples A and B. The df would be calculated as **df = (nA-1) + (nB -1)**

Significance level (α): It is the probability of rejecting the null hypothesis when it is true. In simpler terms, it tells us about the percentage of risk involved in saying that a difference exists between two groups when in reality it does not.

In data mining practice, we may often employ a single test set, that is, the same test set can be used for both M1 and M2. In such cases, we do a pairwise comparison of the two models for each 10-fold cross-validation round.

Let err(M1)i (or err(M2)i) be the error rate of model M1 (or M2) on round i. The error rates for M1 are averaged to obtain a mean error rate for M1, denoted err(M1). Similarly, we can obtain err(M2). The variance of the difference between the two models is denoted var(M1 − M2). The t-test computes the t-statistic with k − 1 degrees of freedom for k samples. In our example we have k = 10 since, here, the k samples are our error rates obtained from ten 10-fold cross-validations for each model.

The t-statistic for pairwise comparison is computed as follows:

$$t = \frac{\overline{err}(M_1) - \overline{err}(M_2)}{\sqrt{var(M_1 - M_2)/k}},$$

Comparing Classifiers Based on Cost–Benefit and ROC Curves:

Significance tests and ROC curves are useful tools for model selection. Significance tests can be used to assess whether the difference in accuracy between two classifiers is due to chance. ROC curves plot the true positive rate (or sensitivity) versus the false positive rate (or 1 − specificity) of one or more classifiers.

The ROC Curve:

The receiver operating characteristic (ROC) curve is frequently used for evaluating the performance of binary classification algorithms. It provides a graphical representation of a classifier's performance, rather than a single value like most other metrics.

Receiver operating characteristic curves are a useful visual tool for comparing two classification models. Given a test set and a model, TPR is the proportion of positive (or "yes") tuples that are correctly labeled by the model; FPR is the proportion of negative (or "no") tuples that are mislabeled as positive.

Receiver Operating Characteristics (ROC) graphs are a useful technique for organizing classifiers and visualizing their performance. An ROC graph is a technique for visualizing, organizing and selecting classifiers based on their performance.

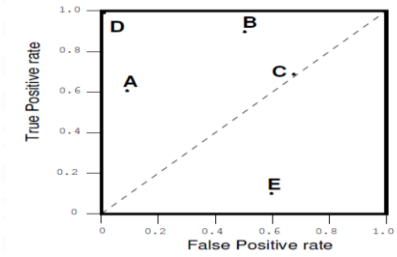
ROC graphs are two-dimensional graphs in which TP rate is plotted on the Y axis and FP rate is plotted on the X axis.

A ROC graph depicts relative trade-off between benefits (true positives) and costs (false positives).

A discrete classier is one that outputs only a class label. Each discrete classifier produces an (FP rate, TP rate) pair, which corresponds to a single point in ROC space.

The following figure shows the ROC curves of two classification models. The diagonal line representing random guessing is also shown. Thus, the closer the ROC curve of a model is to the diagonal line, the less accurate the model.

An example of a ROC graph



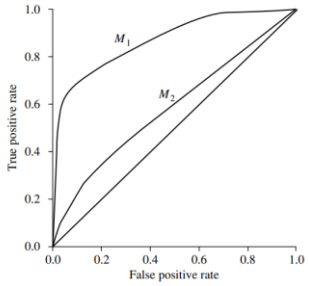


Figure: A basic ROC graph showing five discrete classiers. ROC curves of two classification models, M1 and M2

Techniques to Improve Classification Accuracy:

Ensemble Methods:

Ensemble methods can be used to increase overall accuracy by learning and combining a series of individual (base) classifier models.

Examples of ensemble methods:

- Bagging (Decrease variance)
- Boosting (Decrease bias)
- Random forests

An ensemble for classification is a composite model, made up of a combination of classifiers. The individual classifiers vote, and a class label prediction is returned by the ensemble based on the collection of votes. Ensembles tend to be more accurate than their component classifiers

An ensemble combines a series of k learned models (or base classifiers), M1, M2,..., Mk , with the aim of creating an improved composite classification model, M∗. A given data set, D, is used to create k training sets, D1, D2,..., Dk , where Di (1 ≤ i ≤ k − 1) is used to generate classifier Mi .

Given a new data tuple to classify, the base classifiers each vote by returning a class prediction. The ensemble returns a class prediction based on the votes of the base classifiers.

An ensemble method is a technique that combines the predictions from multiple models to make more accurate predictions than any individual model. A model comprised of many models is called ensemble model.

Bagging, boosting, and random forests are popular ensemble methods.

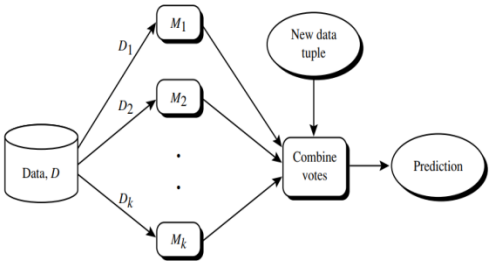


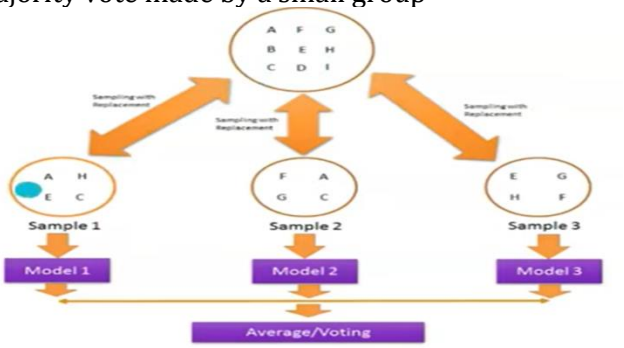
Fig: Ensemble methods generate a set of classification models, M1, M2,..., Mk .

Bagging(Bootstrap aggregation:

In bagging we take different subsets of dataset randomly and combine them with the help of bootstrap sampling. In detail given a training data set containing the n number of training examples, a sample of m training examples will be generated by sampling with replacement. In bagging we used the most popular strategies for aggregating the outputs of the base learners, find out the majority vote in a classification task and finding the mean in the regression task.

There are two main key ingredients of bagging are bootstrap and aggregation.

Example: Suppose that you are a patient and would like to have a diagnosis made based on your symptoms. Instead of asking one doctor, you may choose to ask several. If a certain diagnosis occurs more than any other, you may choose this as the final or best diagnosis. That is, the final diagnosis is made based on a majority vote, where each doctor gets an equal vote. Now replace each doctor by a classifier, and you have the basic idea behind bagging. Intuitively, a majority vote made by a large group of doctors may be more reliable than a majority vote made by a small group



Given a set, D, of d tuples, bagging works as follows. For iteration i(i = 1, 2,..., k), a training set, Di , of d tuples is sampled with replacement from the original set of tuples, D.

Note that the term bagging stands for bootstrap aggregation. A classifier model, Mi , is learned for each training set, Di . To classify an unknown tuple, X, each classifier, Mi , returns its class prediction, which counts as one vote. The bagged classifier, M∗, counts the votes and assigns the class with the most votes to X. Bagging can be applied to the prediction of continuous values by taking the average value of each prediction for a given test tuple. **The algorithm is summarized as**

Algorithm: Bagging. The bagging algorithm—create an ensemble of classification models for a learning scheme where each model gives an equally weighted prediction.

Input:

- D, a set of d training tuples;
- k, the number of models in the ensemble;
- a classification learning scheme (decision tree algorithm, naïve Bayesian, etc.).

Output: The ensemble—a composite model, $M*$.

Method:

(1) **for** $i = 1$ to k **do** // create k models:
(2) create bootstrap sample, D_i, by sampling D with replacement;
(3) use D_i and the learning scheme to derive a model, M_i;
(4) **endfor**

To use the ensemble to classify a tuple, X:

let each of the k models classify X and return the majority vote;

Boosting and AdaBoost

Boosting is a kind of algorithms that is able to convert weal learners to strong learners. In a boosting technique each algorithm

i.e base learners are trained sequentially and at every time the next learner is trying to reduce the error by updating

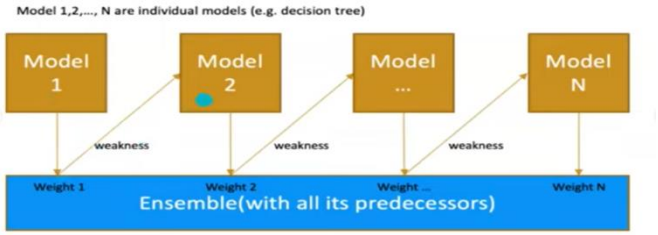
the parameters and perform better in comparison to the previous learner.

Boosting works by training a different set of learners sequentially and combining them for prediction, where the learners

focus more on the mistakes of the earlier learners**.**

Example: suppose that as a patient, you have certain symptoms. Instead of consulting one doctor, you choose to consult several. Suppose you assign weights to the value or worth of each doctor's diagnosis, based on the accuracies of previous diagnoses they have made. The final diagnosis is then a combination of the weighted diagnoses. This is the essence behind boosting.

In boosting, weights are also assigned to each training tuple. A series of k classifiers is iteratively learned. After a classifier, Mi , is learned, the weights are updated to allow the subsequent classifier, Mi+1, to "pay more attention" to the training tuples that were misclassified by Mi . The final boosted classifier, M∗, combines the votes of each individual classifier, where the weight of each classifier's vote is a function of its accuracy



AdaBoost (short for Adaptive Boosting):

AdaBoost is a popular boosting algorithm. Suppose we want to boost the accuracy of a learning method. We are given D, a data set of d class-labeled tuples, (X1, y1),(X2, y2),...,(Xd, yd), where yi is the class label of tuple Xi . Initially, AdaBoost assigns each training tuple an equal weight of 1/d.

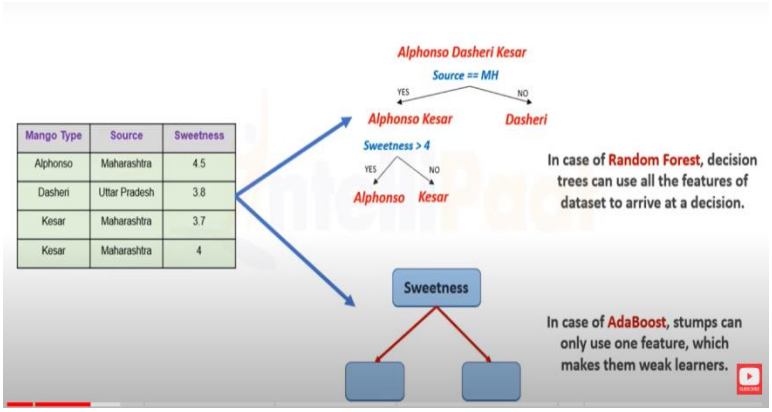
Generating k classifiers for the ensemble requires k rounds through the rest of the algorithm. In round i, the tuples from D are sampled to form a training set, Di , of size d.

Sampling with replacement is used—the same tuple may be selected more than once. Each tuple's chance of being selected is based on its weight. A classifier model, Mi , is derived from the training tuples of Di . Its error is then calculated using Di as a test set. The weights of the training tuples are then adjusted according to how they were classified.

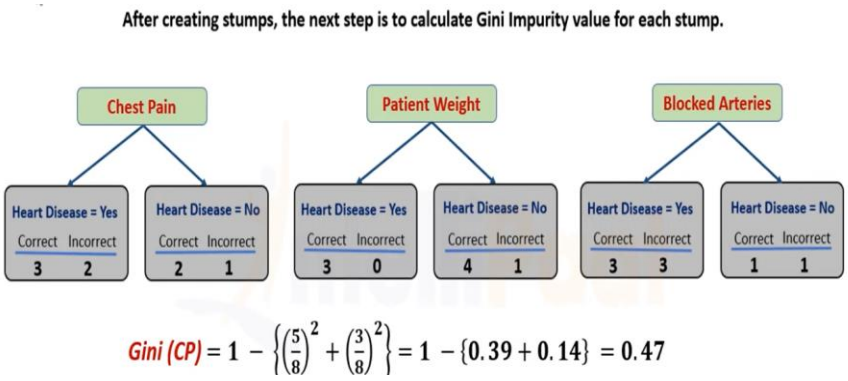
If a tuple was incorrectly classified, its weight is increased. If a tuple was correctly classified, its weight is decreased. A tuple's weight reflects how difficult it is to classify— the higher the weight, the more often it has been misclassified. These weights will be used to generate the training samples for the classifier of the next round. The basic idea is that when we build a classifier, we want it to focus more on the misclassified tuples of the previous round. Some classifiers may be better at classifying some "difficult" tuples than others. In this way, we build a series of classifiers that complement each other. **"Once boosting is complete, how is the ensemble of classifiers used to predict the class label of a tuple, X?"** Unlike bagging, where each classifier was assigned an equal vote, boosting assigns a weight to each classifier's vote, based on how well the classifier performed. The lower a classifier's error rate, the more accurate it is, and therefore, the higher its weight for voting should be.

The weight of classifier Mi 's vote is

$$log\frac{1-error(M_i)}{error(M_i)}.$$



For each class, c, we sum the weights of each classifier that assigned class c to X. The class with the highest sum is the "winner" and is returned as the class prediction for tuple X.



$$Gini\ (CP) = 1 - \left\{\left(\frac{5}{8}\right)^2 + \left(\frac{3}{8}\right)^2\right\} = 1 - \{0.39 + 0.14\} = 0.47$$

The algorithm is summarized following:

Algorithm: AdaBoost. A boosting algorithm—create an ensemble of classifiers. Each one gives a weighted vote.

Input:

- D, a set of d class-labeled training tuples;
- k, the number of rounds (one classifier is generated per round);
- a classification learning scheme.

Output: A composite model.

Method:

(1) initialize the weight of each tuple in D to $1/d$;
(2) **for** $i = 1$ to k **do** // for each round:
(3) sample D with replacement according to the tuple weights to obtain D_i;
(4) use training set D_i to derive a model, M_i;
(5) compute $error(M_i)$, the error rate of M_i (Eq. 8.34)
(6) **if** $error(M_i) > 0.5$ **then**
(7) go back to step 3 and try again;
(8) **endif**
(9) **for** each tuple in D_i that was correctly classified **do**
(10) multiply the weight of the tuple by $error(M_i)/(1 - error(M_i))$; // update weights
(11) normalize the weight of each tuple;
(12) **endfor**

To use the ensemble to classify tuple, X:

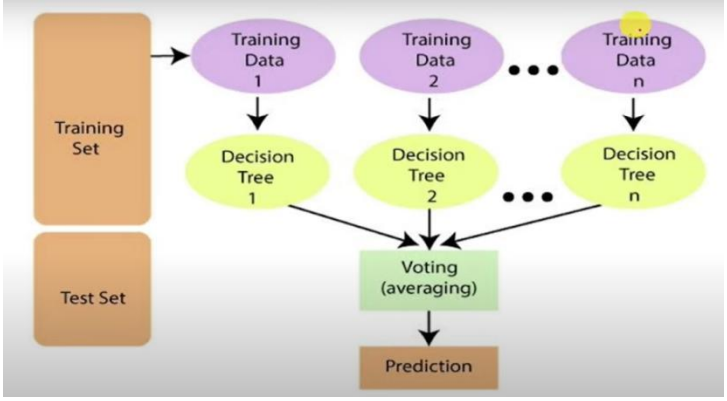
(1) initialize weight of each class to 0;
(2) **for** $i = 1$ to k **do** // for each classifier:
(3) $w_i = log \frac{1-error(M_i)}{error(M_i)}$; // weight of the classifier's vote
(4) $c = M_i(\boldsymbol{X})$; // get class prediction for $\boldsymbol{X}$ from M_i
(5) add w_i to weight for class c
(6) **endfor**
(7) return the class with the largest weight;

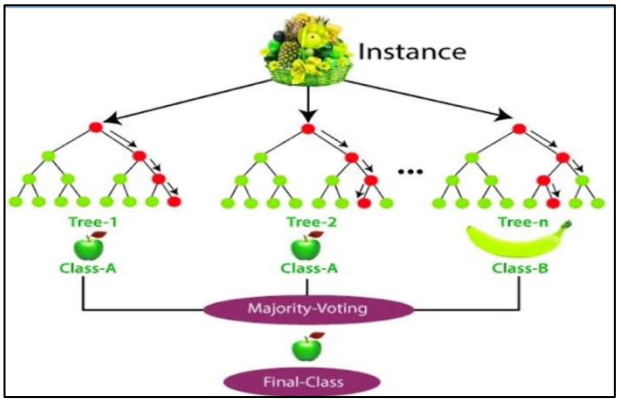
Random Forests:

We now present another ensemble method called random forests. Imagine that each of the classifiers in the ensemble is a decision tree classifier so that the collection of classifiers is a "forest." The individual decision trees are generated using a random selection of attributes at each node to determine the split. More formally, each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest. During classification, each tree votes and the most popular class is returned.

1. Build random forests :
   a) If the number of examples in the training set is $\boldsymbol{N}$, take a sample of $\boldsymbol{n}$ examples at random - but with replacement, from the original data. This sample will be the training set for generating the tree.
   b) If there are $\boldsymbol{M}$ input variables, $\boldsymbol{m}$ variables are selected at random out of the $\boldsymbol{M}$ and the best split on these $\boldsymbol{m}$ is used to split the node. The value of $\boldsymbol{m}$ is held constant during the generation of the various trees in the forest.
   c) Each tree is grown to the largest extent possible.
2. For new data points, find the predictions of each decision tree, and assign the new data points to the category that wins the **majority votes**.

Example:





The generalization error for a forest converges as long as the number of trees in the forest is large. Thus, overfitting is not a problem. The accuracy of a random forest depends on the strength of the individual classifiers and a measure of the dependence between them.

Improving Classification Accuracy of Class-Imbalanced Data:

The class imbalance problem occurs when the main class of interest is represented by only a few tuples.

A classification data set with skewed class proportions is called imbalanced. Classes that make up a large proportion of the data set are called majority classes. Those that make up a smaller proportion are minority classes.

Strategies to address this problem : include general approaches for improving the classification accuracy of class-imbalanced data.

These approaches include

(1) oversampling,

(2) undersampling,

(3) threshold moving, and (4) ensemble techniques.

The first three do not involve any changes to the construction of the classification model.

Both oversampling and under sampling change the training data distribution so that the rare (positive) class is well represented. Oversampling works by resampling the positive tuples so that the resulting training set contains an equal number of positive and negative tuples. Under sampling works by decreasing the number of negative tuples. It randomly eliminates tuples from the majority (negative) class until there are an equal number of positive and negative tuples.

Example: Oversampling and undersampling. Suppose the original training set contains 100 positive and 1000 negative tuples. In oversampling, we replicate tuples of the rarer class to form a new training set containing 1000 positive tuples and 1000 negative tuples. In undersampling, we randomly eliminate negative tuples so that the new training set contains 100 positive tuples and 100 negative tuples.

UNIT WISE IMPORTANT QUESTIONS:

1.Define information gain and Explain its importance in decision tree induction

department	*status*	*age*	*salary*	*count*
sales	senior	31...35	46K...50K	30
sales	junior	26...30	26K...30K	40
sales	junior	31...35	31K...35K	40
systems	junior	21...25	46K...50K	20
systems	senior	31...35	66K...70K	5
systems	junior	26...30	46K...50K	3
systems	senior	41...45	66K...70K	3
marketing	senior	36...40	46K...50K	10
marketing	junior	31...35	41K...45K	4
secretary	senior	46...50	36K...40K	4
secretary	junior	26...30	26K...30K	6

The following table consists of training data from an employee database. The data have been generalized. For example, "31 ... 35" for age represents the age range of 31 to 35. For a given row entry, count represents the number of data tuples having the values for department, status, age, and salary given in that row.

Let status be the class label attribute.

(a) How would you modify the basic decision tree algorithm to take into consideration the count of each generalized data tuple (i.e., of each row entry)?

(b) Use your algorithm to construct a decision tree from the given data.- 4M

(c) Given a data tuple having the values "systems," "26 . . . 30," and "46–50K" for the attributes department, age, and salary, respectively, what would a na¨ıve Bayesian classification of the status for the tuple be?

2. Consider a school with a total population of 100 persons. These 100 persons can be seen either as 'Students' and 'Teachers' or as a population of 'Males' and 'Females'.

With below tabulation of the 100 people, what is the conditional probability that a certain number of the school is a 'Teacher' given that he is a 'Man'?

	Female	Male	Total
Teacher	8	12	20
Student	32	48	80
Total	40	60	100

3. Why is tree pruning useful in decision tree induction? What is the drawback of using a separate set of tuples to evaluate pruning?

4. Why is naïve bayesian classification called naïve? Breifly outline the major ideas of naïve bayesian classification. Explain Naïve bayes classfication.

5. Compare the advantages and disadvantages of eager classifications versus lazy classification.

6. Briefly outline major steps of decision tree classification.

7. Explain decision tree induction with example.

8. Explain Bayes theorem .

9. Explain rule based classification.
10. Explain the methods used to improve the accuracy of a classifier.
11. Explain Attribute selection measures.
12. What are ensemble methods explain in detail.
13. Explain the following:

a. Bagging
b. Boosting
c. Adaboost Algorithm
d. Cross validation(k-fold cross validation)
e. Sequential algorithm
e. Overfitting and Underfitting
f. Evaluation Measures