

Unit 2 (cont.)

Data Cleaning

1. Ignore the tuple: This is usually done when the class label is missing (assuming the mining task involves classification). This method is not very effective, unless the tuple contains several attributes with missing values. It is especially poor when the percentage of missing values per attribute varies considerably.
2. Fill in the missing value manually: In general, this approach is time-consuming and may not be feasible given a large data set with many missing values.
3. Use a global constant to fill in the missing value: Replace all missing attribute values by the same constant, such as a label like “*Unknown*”. If missing values are replaced by, say, “*Unknown*,” then the mining program may mistakenly think that they form an interesting concept, since they all have a value in common—that of “*Unknown*.” Hence, although this method is simple, it is not good to use.
4. Use the attribute mean to fill in the missing value: For example, suppose that the average income of customers is \$56,000. Use this value to replace the missing value for *income*.
5. Use the attribute mean for all samples belonging to the same class as the given tuple: For example, if classifying customers according to *credit risk*, replace the missing value with the average *income* value for customers in the same credit risk category as that of the given tuple.
6. Use the most probable value to fill in the missing value: This may be determined with regression, inference-based tools using a Bayesian formalism, or decision tree induction. For example, using the other customer attributes in your data set, you may construct a decision tree to predict the missing values for *income*.

Noisy Data

- Binning: Binning methods smooth a sorted data value by consulting its “neighborhood,” that is, the values around it. The sorted values are distributed into a number of “buckets,” or *bins*. Because binning methods consult the neighborhood of values, they perform *local* smoothing.
- Regression: Data can be smoothed by fitting the data to a function, such as with regression. Linear regression involves finding the “best” line to fit two attributes (or variables), so that one attribute can be used to predict the other. Multiple linear regression is an extension of linear regression, where more than two attributes are involved and the data are fit to a multidimensional surface.

- Clustering: Outliers may be detected by clustering, where similar values are organized into groups, or “clusters.” Intuitively, values that fall outside of the set of clusters may be considered outliers

Many methods for data smoothing are also methods for data reduction involving discretization. For example, the binning techniques described above reduce the number of distinct values per attribute. This acts as a form of data reduction for logic-based data mining methods, such as decision tree induction, which repeatedly make value comparisons on sorted data. Concept hierarchies are a form of data discretization that can also be used for data smoothing.

Data Cleaning as a Process

The first step in data cleaning as a process is *discrepancy detection*. Discrepancies can be caused by several factors, including poorly designed data entry forms that have many optional fields, human error in data entry, deliberate errors (e.g., respondents not wanting to divulge information about themselves), and data decay (e.g., outdated addresses). Discrepancies may also arise from inconsistent data representations and the inconsistent use of codes. Errors in instrumentation devices that record data, and system errors, are another source of discrepancies. Errors can also occur when the data are (inadequately) used for purposes other than originally intended. There may also be inconsistencies due to data integration. As the starting point to find data discrepancy any known knowledge about the data is used for e.g. using Metadata.

There are a number of different commercial tools that can aid in the step of discrepancy detection. Data scrubbing tools use simple domain knowledge (e.g., knowledge of postal addresses, and spell-checking) to detect errors and make corrections in the data. These tools rely on parsing and fuzzy matching techniques when cleaning data from multiple sources. Data auditing tools find discrepancies by analyzing the data to discover rules and relationships, and detecting data that violate such conditions. Once we find discrepancies, we typically need to define and apply (a series of) transformations to correct them. Commercial tools can assist in the data transformation step. Data migration tools allow simple transformations to be specified, such as to replace the string “gender” by “sex”. ETL (extraction/transformation/loading) tools allow users to specify transforms through a graphical user interface (GUI). These tools typically support only a restricted set of transforms so that, often, we may also choose to write custom scripts for this step of the data cleaning process.

Data Integration

Data integration, combines data from multiple sources into a coherent data store, as in data warehousing. These sources may include multiple databases, data cubes, or flat files.

There are a number of issues to consider during data integration. *Schema integration* and *object matching* can be tricky. *Redundancy* is another important issue. An attribute (such as *annual revenue*, for instance) may be redundant if it can be “derived” from another attribute or set of attributes. Inconsistencies in attribute or dimension naming can also cause redundancies in the resulting data set. Some redundancies can be detected by correlation analysis. Given two attributes, such analysis can measure how strongly one attribute implies the other, based on the available data. For numerical attributes, we can evaluate the correlation between two attributes, A and B , by computing the correlation coefficient (also known as *Pearson’s product moment coefficient*, named after its inventor, Karl Pearson). This is

$$r_{A,B} = \frac{\sum_{i=1}^N (a_i - \bar{A})(b_i - \bar{B})}{N\sigma_A\sigma_B} = \frac{\sum_{i=1}^N (a_i b_i) - N\bar{A}\bar{B}}{N\sigma_A\sigma_B},$$

Where N is the number of tuples, a_i and b_i are the respective values of A and B in tuple i , \bar{A} and \bar{B} are the respective mean values of A and B , σ_A and σ_B are the respective standard deviations of A and B , and $\sum(a_i b_i)$ is the sum of the AB cross-product.

For categorical (discrete) data, a correlation relationship between two attributes, A and B , can be discovered by a χ^2 (chi-square) test.

$$\chi^2 = \sum_{i=1}^c \sum_{j=1}^r \frac{(o_{ij} - e_{ij})^2}{e_{ij}},$$

where o_{ij} is the *observed frequency* (i.e., actual count) of the joint event (A_i, B_j) and e_{ij} is the *expected frequency* of (A_i, B_j) , which can be computed as

$$e_{ij} = \frac{\text{count}(A = a_i) \times \text{count}(B = b_j)}{N},$$

Where N is the number of data tuples, $\text{count}(A=a_i)$ is the number of tuples having value a_i for A , and $\text{count}(B = b_j)$ is the number of tuples having value b_j for B . The sum in equation is computed over all of the $r \times c$ cells.

In addition to detecting redundancies between attributes, duplication should also be detected at the tuple level. The use of denormalized tables (often done to improve performance by avoiding joins) is another source of data redundancy. Inconsistencies often arise between various duplicates, due to inaccurate data entry or updating some but not all of the occurrences of the data. A third important issue in data integration is the *detection and resolution of data value conflicts*. This may be due to differences in representation, scaling, or encoding. For instance, a *weight* attribute may be stored in metric units in one system and British imperial units in another. For a hotel chain, the *price* of rooms in different cities may involve not only different currencies

but also different services (such as free breakfast) and taxes. When matching attributes from one database to another during integration, special attention must be paid to the *structure* of the data. This is to ensure that any attribute functional dependencies and referential constraints in the source system match those in the target system. Careful integration of the data from multiple sources can help reduce and avoid redundancies and inconsistencies in the resulting data set. This can help improve the accuracy and speed of the subsequent mining process.

Data Transformation

In *data transformation*, the data are transformed or consolidated into forms appropriate for mining. Data transformation can involve the following:

- Smoothing, which works to remove noise from the data. Such techniques include binning, regression, and clustering.
- Aggregation, where summary or aggregation operations are applied to the data. For example, the daily sales data may be aggregated so as to compute monthly and annual total amounts. This step is typically used in constructing a data cube for analysis of the data at multiple granularities.
- Generalization of the data, where low-level or “primitive” (raw) data are replaced by higher-level concepts through the use of concept hierarchies. For example, categorical attributes, like *street*, can be generalized to higher-level concepts, like *city* or *country*. Similarly, values for numerical attributes, like *age*, may be mapped to higher-level concepts, like *youth*, *middle-aged*, and *senior*.
- Normalization, where the attribute data are scaled so as to fall within a small specified range, such as [1:0 to 1:0, or 0:0 to 1:0].
- Attribute construction (or *feature construction*) where new attributes are constructed and added from the given set of attributes to help the mining process.

Data Reduction

Data reduction techniques can be applied to obtain a reduced representation of the data set that is much smaller in volume, yet closely maintains the integrity of the original data. That is, mining on the reduced data set should be more efficient yet produce the same (or almost the same) analytical results.

Strategies for data reduction include the following:

1. Data cube aggregation, where aggregation operations are applied to the data in the construction of a data cube.
2. Attribute subset selection, where irrelevant, weakly relevant, or redundant attributes or dimensions may be detected and removed.
3. Dimensionality reduction, where encoding mechanisms are used to reduce the data set size.
4. Numerosity reduction, where the data are replaced or estimated by alternative, smaller data representations such as parametric models (which need store only the model parameters

instead of the actual data) or nonparametric methods such as clustering, sampling, and the use of histograms.

5. Discretization and concept hierarchy generation, where raw data values for attributes are replaced by ranges or higher conceptual levels. Data discretization is a form of numerosity reduction that is very useful for the automatic generation of concept hierarchies.

Discretization and concept hierarchy generation are powerful tools for data mining, in that they allow the mining of data at multiple levels of abstraction.

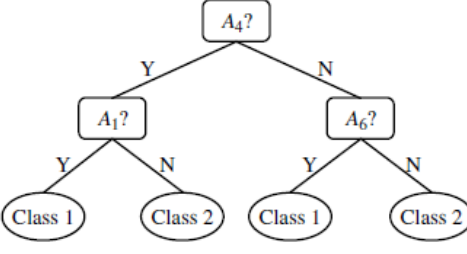
Data Cube Aggregation

Data cubes store multidimensional aggregated information. Each cell holds an aggregate data value, corresponding to the data point in multidimensional space. Concept hierarchies may exist for each attribute, allowing the analysis of data at multiple levels of abstraction. The cube created at the lowest level of abstraction is referred to as the *base cuboid*. The base cuboid should correspond to an individual entity of interest, such as *sales* or *customer*. In other words, the lowest level should be usable, or useful for the analysis. A cube at the highest level of abstraction is the *apex cuboid*.

Attribute Subset Selection

Attribute subset selection reduces the data set size by removing irrelevant or redundant attributes (or dimensions). The goal of attribute subset selection is to find a minimum set of attributes such that the resulting probability distribution of the data classes is as close as possible to the original distribution obtained using all attributes. Mining on a reduced set of attributes has an additional benefit. It reduces the number of attributes appearing in the discovered patterns, helping to make the patterns easier to understand.

For n attributes, there are 2^n possible subsets. An exhaustive search for the optimal subset of attributes can be prohibitively expensive, especially as n and the number of data classes increase. Therefore, heuristic methods that explore a reduced search space are commonly used for attribute subset selection. These methods are typically greedy in that, while searching through attribute space, they always make what looks to be the best choice at the time. Their strategy is to make a locally optimal choice in the hope that this will lead to a globally optimal solution. Such greedy methods are effective in practice and may come close to estimating an optimal solution. The “best” (and “worst”) attributes are typically determined using tests of statistical significance, which assume that the attributes are independent of one another. Many other attribute evaluation measures can be used, such as the *information gain* measure used in building decision trees for classification. Basic heuristic methods of attribute subset selection include the following techniques, some of which are illustrated in figure

Forward selection	Backward elimination	Decision tree induction
Initial attribute set: $\{A_1, A_2, A_3, A_4, A_5, A_6\}$ Initial reduced set: $\{\}$ $\Rightarrow \{A_1\}$ $\Rightarrow \{A_1, A_4\}$ \Rightarrow Reduced attribute set: $\{A_1, A_4, A_6\}$	Initial attribute set: $\{A_1, A_2, A_3, A_4, A_5, A_6\}$ $\Rightarrow \{A_1, A_3, A_4, A_5, A_6\}$ $\Rightarrow \{A_1, A_4, A_5, A_6\}$ \Rightarrow Reduced attribute set: $\{A_1, A_4, A_6\}$	Initial attribute set: $\{A_1, A_2, A_3, A_4, A_5, A_6\}$  \Rightarrow Reduced attribute set: $\{A_1, A_4, A_6\}$

1 .Stepwise forward selection: The procedure starts with an empty set of attributes as the reduced set. The best of the original attributes is determined and added to the reduced set. At each subsequent iteration or step, the best of the remaining original attributes is added to the set.

2. Stepwise backward elimination: The procedure starts with the full set of attributes. At each step, it removes the worst attribute remaining in the set.

3. Combination of forward selection and backward elimination: The stepwise forward selection and backward elimination methods can be combined so that, at each step, the procedure selects the best attribute and removes the worst from among the remaining attributes.

4. Decision tree induction: Decision tree algorithms, such as ID3, C4.5, and CART, were originally intended for classification. Decision tree induction constructs a flowchart like structure where each internal (non-leaf) node denotes a test on an attribute, each branch corresponds to an outcome of the test, and each external (leaf) node denotes a class prediction. At each node, the algorithm chooses the “best” attribute to partition the data into individual classes. When decision tree induction is used for attribute subset selection, a tree is constructed from the given data. All attributes that do not appear in the tree are assumed to be irrelevant. The set of attributes appearing in the tree form the reduced subset of attributes.

The stopping criteria for the methods may vary. The procedure may employ a threshold on the measure used to determine when to stop the attribute selection process.

Dimensionality Reduction

In *dimensionality reduction*, data encoding or transformations are applied so as to obtain a reduced or “compressed” representation of the original data. If the original data can be *reconstructed* from the compressed data without any loss of information, the data reduction is called lossless. If, instead, we can reconstruct only an approximation of the original data, then the data reduction is called lossy. There are several well-tuned algorithms for string compression. Although they are typically lossless, they allow only limited manipulation of the data. In this section, we instead focus on two popular and effective methods of lossy dimensionality reduction: *wavelet transforms* and *principal components analysis*.

Wavelet Transforms

The discrete wavelet transform (DWT) is a linear signal processing technique that, when applied to a data vector \mathbf{X} , transforms it to a numerically different vector, \mathbf{X}' , of wavelet coefficients. The

two vectors are of the same length. When applying this technique to data reduction, we consider each tuple as an n -dimensional data vector, that is, $\mathbf{X} = (x_1, x_2, \dots, x_n)$, depicting n measurements made on the tuple from n database attributes. The usefulness lies in the fact that the wavelet transformed data can be truncated. A compressed approximation of the data can be retained by storing only a small fraction of the strongest of the wavelet coefficients.

For example, all wavelet coefficients larger than some user-specified threshold can be retained. All other coefficients are set to 0. The resulting data representation is therefore very sparse, so that operations that can take advantage of data sparsity are computationally very fast if performed in wavelet space. The technique also works to remove noise without smoothing out the main features of the data, making it effective for data cleaning as well. Given a set of coefficients, an approximation of the original data can be constructed by applying the *inverse* of the DWT used. The DWT is closely related to the *discrete Fourier transform (DFT)*, a signal processing technique involving sines and cosines. In general, however, the DWT achieves better lossy compression. That is, if the same number of coefficients is retained for a DWT and a DFT of a given data vector, the DWT version will provide a more accurate approximation of the original data. Hence, for an equivalent approximation, the DWT requires less space than the DFT. Unlike the DFT, wavelets are quite localized in space, contributing to the conservation of local detail. The method is as follows:

1. The length, L , of the input data vector must be an integer power of 2. This condition can be met by padding the data vector with zeros as necessary ($L \geq n$).
2. Each transform involves applying two functions. The first applies some data smoothing, such as a sum or weighted average. The second performs a weighted difference, which acts to bring out the detailed features of the data.
3. The two functions are applied to pairs of data points in \mathbf{X} , that is, to all pairs of measurements (x_{2i}, x_{2i+1}) . This results in two sets of data of length $L/2$. In general, these represent a smoothed or low-frequency version of the input data and the high frequency content of it, respectively.
4. The two functions are recursively applied to the sets of data obtained in the previous loop, until the resulting data sets obtained are of length 2.
5. Selected values from the data sets obtained in the above iterations are designated the wavelet coefficients of the transformed data.

Equivalently, a matrix multiplication can be applied to the input data in order to obtain the wavelet coefficients, where the matrix used depends on the given DWT. The matrix must be orthonormal, meaning that the columns are unit vectors and are mutually orthogonal, so that the matrix inverse is just its transpose. Wavelet transforms can be applied to multidimensional data, such as a data cube.

This is done by first applying the transform to the first dimension, then to the second, and so on. The computational complexity involved is linear with respect to the number of cells in the cube. Wavelet transforms give good results on sparse or skewed data and on data with ordered attributes. Lossy compression by wavelets is reportedly better than JPEG compression, the current commercial standard. Wavelet transforms have many real-world applications, including the compression of fingerprint images, computer vision, analysis of time-series data, and data cleaning.

Principal Components Analysis

Principal components analysis, or PCA (also called the Karhunen-Loeve, or K-L, method), searches for k n -dimensional orthogonal vectors that can best be used to represent the data, where $k \leq n$. The original data are thus projected onto a much smaller space, resulting in dimensionality reduction. Unlike attribute subset selection, which reduces the attribute set size by retaining a subset of the initial set of attributes, PCA “combines” the essence of attributes by creating an alternative, smaller set of variables. The initial data can then be projected onto this smaller set. PCA often reveals relationships that were not previously suspected and thereby allows interpretations that would not ordinarily result.

The basic procedure is as follows:

1. The input data are normalized, so that each attribute falls within the same range. This step helps ensure that attributes with large domains will not dominate attributes with smaller domains.
2. PCA computes k orthonormal vectors that provide a basis for the normalized input data. These are unit vectors that each point in a direction perpendicular to the others. These vectors are referred to as the *principal components*. The input data are a linear combination of the principal components.
3. The principal components are sorted in order of decreasing “significance” or strength. The principal components essentially serve as a new set of axes for the data, providing important information about variance. That is, the sorted axes are such that the first axis shows the most variance among the data, the second axis shows the next highest variance, and so on. For example, Figure shows the first two principal components, Y_1 and Y_2 , for the given set of data originally mapped to the axes X_1 and X_2 . This information helps identify groups or patterns within the data.
4. Because the components are sorted according to decreasing order of “significance,” the size of the data can be reduced by eliminating the weaker components, that is, those with low variance. Using the strongest principal components, it should be possible to reconstruct a good approximation of the original data. PCA is computationally inexpensive, can be applied to ordered and unordered attributes, and can handle sparse data and skewed data. Multidimensional data of more than two dimensions can be handled by reducing the problem to two dimensions. Principal components may be used as inputs to multiple regression and cluster analysis. In comparison with wavelet transforms, PCA tends to be better at handling sparse data, whereas wavelet transforms are more suitable for data of high dimensionality.

Numerosity Reduction – reduces the data volume by choosing alternative, ‘smaller’ forms of data representation

Regression and Log-Linear Models

Regression and log-linear models can be used to approximate the given data. In (simple) linear regression, the data are modeled to fit a straight line. For example, a random variable, y (called a *response variable*), can be modeled as a linear function of another random variable, x (called a *predictor variable*), with the equation

$$y = wx + b,$$

where the variance of y is assumed to be constant. In the context of data mining, x and y are numerical database attributes. The coefficients, w and b (called *regression coefficients*), specify the slope of the line and the Y -intercept, respectively. These coefficients can be solved for by the *method of least squares*, which minimizes the error between the actual line separating the data

and the estimate of the line. Multiple linear regression is an extension of (simple) linear regression, which allows a response variable, y , to be modelled as a linear function of two or more predictor variables.

Log-linear models approximate discrete multidimensional probability distributions.

Given a set of tuples in n dimensions (e.g., described by n attributes), we can consider each tuple as a point in an n -dimensional space. Log-linear model scan be used to estimate the probability of each point in a multidimensional space for a set of discretized attributes, based on a smaller subset of dimensional combinations.

This allows a higher-dimensional data space to be constructed from lower dimensional spaces. Log-linear models are therefore also useful for dimensionality reduction (since the lower-dimensional points together typically occupy less space than the original data points) and data smoothing (since aggregate estimates in the lower-dimensional space are less subject to sampling variations than the estimates in the higher-dimensional space).

Regression and log-linear models can both be used on sparse data, although their application may be limited. While both methods can handle skewed data, regression does exceptionally well. Regression can be computationally intensive when applied to high dimensional data, whereas log-linear models show good scalability for up to 10 or so dimensions.

Histograms

A histogram for an attribute, A , partitions the data distribution of A into disjoint subsets, or *buckets*. If each bucket represents only a single attribute-value/frequency pair, the buckets are called *singleton buckets*. Often, buckets instead represent continuous ranges for the given attribute. There are several partitioning rules, including the following:

- Equal-width: In an equal-width histogram, the width of each bucket range is uniform.
- Equal-frequency (or equidepth): In an equal-frequency histogram, the buckets are created so that, roughly, the frequency of each bucket is constant (that is, each bucket contains roughly the same number of contiguous data samples).
- V-Optimal: If we consider all of the possible histograms for a given number of buckets, the V-Optimal histogram is the one with the least variance. Histogram variance is a weighted sum of the original values that each bucket represents, where bucket weight is equal to the number of values in the bucket.
- MaxDiff: In a MaxDiff histogram, we consider the difference between each pair of adjacent values. A bucket boundary is established between each pair for pairs having the $b-1$ largest differences, where b is the user-specified number of buckets.

V-Optimal and MaxDiff histograms tend to be the most accurate and practical. Histograms are highly effective at approximating both sparse and dense data, as well as highly skewed and uniform data. The histograms described above for single attributes can be extended for multiple attributes. *Multidimensional histograms* can capture dependencies between attributes. Such histograms have been found effective in approximating data with up to five attributes. More studies are needed regarding the effectiveness of multidimensional histograms for very high dimensions. Singleton buckets are useful for storing outliers with high frequency.

Clustering

Clustering techniques consider data tuples as objects. They partition the objects into groups or *clusters*, so that objects within a cluster are “similar” to one another and “dissimilar” to objects in other clusters. Similarity is commonly defined in terms of how “close” the objects are in

space, based on a distance function. The “quality” of a cluster may be represented by its *diameter*, the maximum distance between any two objects in the cluster. *Centroid distance* is an alternative measure of cluster quality and is defined as the average distance of each cluster object from the cluster centroid. In data reduction, the cluster representations of the data are used to replace the actual data. The effectiveness of this technique depends on the nature of the data. It is much more effective for data that can be organized into distinct clusters than for smeared data.

In database systems, multidimensional index trees are primarily used for providing fast data access. They can also be used for hierarchical data reduction, providing a multi resolution clustering of the data. This can be used to provide approximate answers to queries. An index tree recursively partitions the multidimensional space for a given set of data objects, with the root node representing the entire space. Such trees are typically balanced, consisting of internal and leaf nodes. Each parent node contains keys and pointers to child nodes that, collectively, represent the space represented by the parent node. Each leaf node contains pointers to the data tuples they represent (or to the actual tuples).

An index tree can therefore store aggregate and detail data at varying levels of resolution or abstraction. It provides a hierarchy of clusterings of the data set, where each cluster has a label that holds for the data contained in the cluster. If we consider each child of a parent node as a bucket, then an index tree can be considered as a *hierarchical histogram*. The use of multidimensional index trees as a form of data reduction relies on an ordering of the attribute values in each dimension. Two-dimensional or multidimensional index trees include R-trees, quad-trees, and their variations. They are well suited for handling both sparse and skewed data.

Sampling

Sampling can be used as a data reduction technique because it allows a large data set to be represented by a much smaller random sample (or subset) of the data. Suppose that a large data set, D , contains N tuples. Let's look at the most common ways that we could sample D for data reduction.

- Simple random sample without replacement (SRSWOR) of size s : This is created by drawing s of the N tuples from D ($s < N$), where the probability of drawing any tuple in D is $1/N$, that is, all tuples are equally likely to be sampled.
- Simple random sample with replacement (SRSWR) of size s : This is similar to SRSWOR, except that each time a tuple is drawn from D , it is recorded and then *replaced*. That is, after a tuple is drawn, it is placed back in D so that it may be drawn again.
- Cluster sample: If the tuples in D are grouped into M mutually disjoint “clusters,” then an SRS of s clusters can be obtained, where $s < M$. For example, tuples in a database are usually retrieved a page at a time, so that each page can be considered a cluster. A reduced data representation can be obtained by applying, say, SRSWOR to the pages, resulting in a cluster sample of the tuples. Other clustering criteria conveying rich semantics can also be explored. For example, in a spatial database, we may choose to define clusters geographically based on how closely different areas are located.
- Stratified sample: If D is divided into mutually disjoint parts called *strata*, a stratified sample of D is generated by obtaining an SRS at each stratum. This helps ensure a representative sample, especially when the data are skewed. For example, a stratified sample may be obtained from customer data, where a stratum is created for each customer

age group. In this way, the age group having the smallest number of customers will be sure to be represented.

An advantage of sampling for data reduction is that the cost of obtaining a sample is *proportional to the size of the sample*, s , as opposed to N , the data set size. Hence, sampling complexity is potentially *sublinear* to the size of the data. Other data reduction techniques can require at least one complete pass through D . For a fixed sample size, sampling complexity increases only linearly as the number of data dimensions,

Data Discretization and Concept Hierarchy Generation

Data discretization techniques can be used to reduce the number of values for a given continuous attribute by dividing the range of the attribute into intervals. Interval label can then be used to replace actual data values. Replacing numerous values of a continuous attribute by a small number of interval labels thereby reduces and simplifies the original data. This leads to a concise, easy-to-use, knowledge-level representation of mining results. Discretization techniques can be categorized based on how the discretization is performed, such as whether it uses class information or which direction it proceeds (i.e., top-down vs. bottom-up). If the discretization process uses class information, then we say it is *supervised discretization*. Otherwise, it is *unsupervised*. If the process starts by first finding one or a few points (called *split points* or *cut points*) to split the entire attribute range, and then repeats this recursively on the resulting intervals, it is called *top-down discretization* or *splitting*. This contrasts with *bottom-up discretization* or *merging*, which starts by considering all of the continuous values as potential split-points, removes some by merging neighborhood values to form intervals, and then recursively applies this process to the resulting intervals. Discretization can be performed recursively on an attribute to provide a hierarchical or multi resolution partitioning of the attribute values, known as a concept hierarchy. Concept hierarchies are useful for mining at multiple levels of abstraction.

A concept hierarchy for a given numerical attribute defines a discretization of the attribute. Concept hierarchies can be used to reduce the data by collecting and replacing low-level concepts (such as numerical values for the attribute *age*) with higher-level concepts (such as *youth*, *middle-aged*, or *senior*). Although detail is lost by such data generalization, the generalized data may be more meaningful and easier to interpret. This contributes to a consistent representation of data mining results among multiple mining tasks, which is a common requirement. In addition, mining on a reduced data set requires fewer input/output operations and is more efficient than mining on a larger, ungeneralised data set. Because of these benefits, discretization techniques and concept hierarchies are typically applied before data mining as a preprocessing step, rather than during mining.

Discretization and Concept Hierarchy Generation for Numerical Data

It is difficult and laborious to specify concept hierarchies for numerical attributes because of the wide diversity of possible data ranges and the frequent updates of data values. Such manual specification can also be quite arbitrary.

Concept hierarchies for numerical attributes can be constructed automatically based on data discretization. We examine the following methods: *binning*, *histogram analysis*, *entropy-based discretization*, *c2-merging*, *cluster analysis*, and *discretization by intuitive partitioning*. In general, each method assumes that the values to be discretized are sorted in ascending order.

Binning

Binning is a top-down splitting technique based on a specified number of bins. These methods are also used as discretization methods for numerosity reduction and concept hierarchy generation. For example, attribute values can be discretized by applying equal-width or equal-frequency binning, and then replacing each bin value by the bin mean or median, as in *smoothing by bin means* or *smoothing by bin medians*, respectively. These techniques can be applied recursively to the resulting partitions in order to generate concept hierarchies. Binning does not use class information and is therefore an unsupervised discretization technique. It is sensitive to the user-specified number of bins, as well as the presence of outliers.

Histogram Analysis

Like binning, histogram analysis is an unsupervised discretization technique because it does not use class information. Histograms partition the values for an attribute, A , into disjoint ranges called *buckets*. In an *equal-width* histogram, for example, the values are partitioned into equal-sized partitions or ranges. With an *equal frequency* histogram, the values are partitioned so that, ideally, each partition contains the same number of data tuples. The histogram analysis algorithm can be applied recursively to each partition in order to automatically generate a multilevel concept hierarchy, with the procedure terminating once a prespecified number of concept levels has been reached. A *minimum interval size* can also be used per level to control the recursive procedure. This specifies the minimum width of a partition, or the minimum number of values for each partition at each level.

Entropy-Based Discretization

Entropy-based discretization is a supervised, top-down splitting technique. It explores class distribution information in its calculation and determination of split-points (data values for partitioning an attribute range). To discretize a numerical attribute, A , the method selects the value of A that has the minimum entropy as a split-point, and recursively partitions the resulting intervals to arrive at a hierarchical discretization. Such discretization forms a concept hierarchy for A . Let D consist of data tuples defined by a set of attributes and a class-label attribute.

The class-label attribute provides the class information per tuple. The basic method for entropy-based discretization of an attribute A within the set is as follows:

1. Each value of A can be considered as a potential interval boundary or split-point (denoted *split point*) to partition the range of A . That is, a split-point for A can partition the tuples in D into two subsets satisfying the conditions $A \leq \text{split point}$ and $A > \text{split point}$, respectively, thereby creating a binary discretization.
2. Entropy-based discretization, as mentioned above, uses information regarding the class label of tuples. To explain the intuition behind entropy-based discretization, we must take a glimpse at classification. Suppose we want to classify the tuples in D by partitioning on attribute A and some split-point. Ideally, we would like this partitioning to result in an exact classification of the tuples. For example, if we had two classes, we would hope that all of the tuples of, say, class $C1$ will fall into one partition, and all of the tuples of class $C2$ will fall into the other partition.

However, this is unlikely. For example, the first partition may contain many tuples of C_1 , but also some of C_2 . How much more information would we still need for a perfect classification, after this partitioning? This amount is called the *expected information requirement* for classifying a tuple in D based on partitioning by A . It is given by

$$Info_A(D) = \frac{|D_1|}{|D|} Entropy(D_1) + \frac{|D_2|}{|D|} Entropy(D_2),$$

Where D_1 and D_2 correspond to the tuples in D satisfying the conditions $A \leq \text{split point}$ and $A > \text{split point}$, respectively; $|D|$ is the number of tuples in D , and soon. The entropy function for a given set is calculated based on the class distribution of the tuples in the set. For example, given m classes, C_1, C_2, \dots, C_m , the entropy of D_1 is

$$Entropy(D_1) = - \sum_{i=1}^m p_i \log_2(p_i),$$

Where p_i is the probability of class C_i in D_1 , determined by dividing the number of tuples of class C_i in D_1 by $|D_1|$, the total number of tuples in D_1 . Therefore, when selecting a split-point for attribute A , we want to pick the attribute value that gives the minimum expected information requirement (i.e., $\min (Info_A(D))$). This would result in the minimum amount of expected information (still) required to perfectly classify the tuples after partitioning by $A \leq \text{split point}$ and $A > \text{split point}$. This is equivalent to the attribute-value pair with the maximum information gain.

3. The process of determining a split-point is recursively applied to each partition obtained, until some stopping criterion is met, such as when the minimum information requirement on all candidate split-points is less than a small threshold, ϵ , or when the number of intervals is greater than a threshold, *max interval*. Entropy-based discretization can reduce data size. Unlike the other methods mentioned here so far, entropy-based discretization uses class information. This makes it more likely that the interval boundaries (split-points) are defined to occur in places that may help improve classification accuracy. The entropy and information gain measures described here are also used for decision tree induction.

Interval Merging by χ^2 Analysis

Chi Merge is a χ^2 -based discretization method. The discretization methods that we have studied up to this point have all employed a top-down, splitting strategy. This contrasts with *Chi Merge*, which employs a bottom-up approach by finding the best neighboring intervals and then merging these to form larger intervals, recursively. The method is supervised in that it uses class information. The basic notion is that for accurate discretization, the relative class frequencies should be fairly consistent within an interval.

Therefore, if two adjacent intervals have a very similar distribution of classes, then the intervals can be merged. Otherwise, they should remain separate. *Chi Merge* proceeds as follows. Initially, each distinct value of a numerical attribute A is considered to be one interval. χ^2 tests are performed for every pair of adjacent intervals.

Adjacent intervals with the least χ^2 values are merged together, because low χ^2 values for a pair indicate similar class distributions. This merging process proceeds recursively until a predefined stopping criterion is met.

The χ^2 statistic tests the hypothesis that two adjacent intervals for a given attribute are independent of the class. The contingency table has two columns (representing the two adjacent intervals) and m rows, where m is the number of distinct classes.

The cell value o_{ij} is the count of tuples in the i th interval and j th class. Similarly, the expected frequency of o_{ij} is $e_{ij} = (\text{number of tuples in interval } i) \times (\text{number of tuples in class } j) / N$, where N is the total number of data tuples. Low χ^2 values for an interval pair indicate that the intervals are independent of the class and can, therefore, be merged.

Cluster Analysis

Cluster analysis is a popular data discretization method. A clustering algorithm can be applied to discretize a numerical attribute, A , by partitioning the values of A into clusters or groups. Clustering takes the distribution of A into consideration, as well as the closeness of data points, and therefore is able to produce high-quality discretization results.

Clustering can be used to generate a concept hierarchy for A by following either a top down splitting strategy or a bottom-up merging strategy, where each cluster forms a node of the concept hierarchy. In the former, each initial cluster or partition may be further decomposed into several sub clusters, forming a lower level of the hierarchy. In the latter, clusters are formed by repeatedly grouping neighboring clusters in order to form higher-level concepts.

Discretization by Intuitive Partitioning (for numerical refer notes)

Although the above discretization methods are useful in the generation of numerical hierarchies, many users would like to see numerical ranges partitioned into relatively uniform, easy-to-read intervals that appear intuitive or “natural.” For example, annual salaries broken into ranges like (\$50,000, \$60,000] are often more desirable than ranges like (\$51,263.98, \$60,872.34].

The 3-4-5 rule can be used to segment numerical data into relatively uniform, natural seeming intervals. In general, the rule partitions a given range of data into 3, 4, or 5 relatively equal-width intervals, recursively and level by level, based on the value range at the most significant digit.

The rule is as follows:

If an interval covers 3, 6, 7, or 9 distinct values at the most significant digit, then partition the range into 3 intervals (3 equal-width intervals for 3, 6, and 9; and 3 intervals in the grouping of 2-3-2 for 7).

If it covers 2, 4, or 8 distinct values at the most significant digit, then partition the range into 4 equal-width intervals.

If it covers 1, 5, or 10 distinct values at the most significant digit, then partition the range into 5 equal-width intervals.

The rule can be recursively applied to each interval, creating a concept hierarchy for the given numerical attribute

Concept Hierarchy Generation for Categorical Data

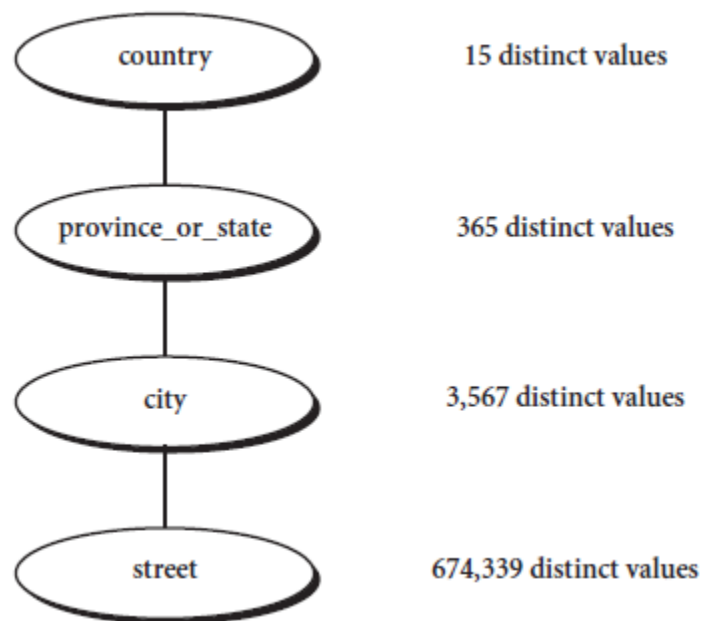
Categorical data are discrete data. Categorical attributes have a finite (but possibly large) number of distinct values, with no ordering among the values. Examples include *geographic location*, *job category*, and *item type*. There are several methods for the generation of concept hierarchies for categorical data.

Specification of a partial ordering of attributes explicitly at the schema level by users or experts: A user or expert can easily define a concept hierarchy by specifying a partial or total ordering of the attributes at the schema level. For example, a relational database or a dimension *location* of a data warehouse may contain the following group of attributes: *street*, *city*, *province* or *state*, and *country*.

Specification of a portion of a hierarchy by explicit data grouping: This is essentially the manual definition of a portion of a concept hierarchy. In a large database, it is unrealistic to define an entire concept hierarchy by explicit value enumeration. On the contrary, we can easily specify explicit groupings for a small portion of intermediate-level data. For example, after specifying that *province* and *country* form a hierarchy at the schema level, a user could define some intermediate levels.

Specification of a set of attributes, but not of their partial ordering: The system can then try to automatically generate the attribute ordering so as to construct a meaningful concept hierarchy.. “*Without knowledge of data semantics, how can a hierarchical ordering for an arbitrary set of categorical attributes be found?*”

Consider the following observation that since higher-level concepts generally cover several subordinate lower-level concepts, an attribute defining a high concept level (e.g., *country*) will usually contain a smaller number of distinct values than an attribute defining a lower concept level (e.g., *street*). Based on this observation, a concept hierarchy can be automatically generated based on the number of distinct values per attribute in the given attribute set. The attribute with the most distinct values is placed at the lowest level of the hierarchy. The lower the number of distinct values an attribute has, the higher it is in the generated concept hierarchy.



Specification of only a partial set of attributes: Sometimes a user can be sloppy when defining a hierarchy, or have only a vague idea about what should be included in a hierarchy. Consequently, the user may have included only a small subset of the relevant attributes in the hierarchy specification. For example, instead of including all of the hierarchically relevant attributes for *location*, the user may have specified only *street* and *city*. To handle such partially specified hierarchies, it is important to embed data semantics in the database schema so that attributes with tight semantic connections can be pinned together

UNIT 3: Association Analysis

Frequent patterns are patterns (such as item sets, subsequences, or substructures) that appear in a data set frequently. For example, a set of items, such as milk and bread, that appear frequently together in a transaction data set is a *frequent item set*. A subsequence, such as buying first a PC, then a digital camera, and then a memory card, if it occurs frequently in a shopping history database, is a (*frequent*) *sequential pattern*. A *substructure* can refer to different structural forms, such as subgraphs, subtrees, or sublattices, which may be combined with itemsets or subsequences. If a substructure occurs frequently, it is called a (*frequent*) *structured pattern*. Finding such frequent patterns plays an essential role in mining associations, correlations, and many other interesting relationships among data. Moreover, it helps in data classification, clustering, and other data mining tasks as well. Thus, frequent pattern mining has become an important data mining task and a focused theme in data mining research.

Frequent itemset mining leads to the discovery of associations and correlations among items in large transactional or relational data sets. With massive amounts of data continuously being collected and stored, many industries are becoming interested in mining such patterns from their

databases. The discovery of interesting correlation relationships among huge amounts of business transaction records can help in many business decision-making processes, such as catalog design, cross-marketing, and customer shopping behavior analysis.

A typical example of frequent itemset mining is market basket analysis. This process analyzes customer buying habits by finding associations between the different items that customers place in their “shopping baskets”. The discovery of such associations can help retailers develop marketing strategies by gaining insight into which items are frequently purchased together by customers.

If we think of the universe as the set of items available at the store, then each item has a Boolean variable representing the presence or absence of that item. Each basket can then be represented by a Boolean vector of values assigned to these variables. The Boolean vectors can be analyzed for buying patterns that reflect items that are frequently *associated* or purchased together. These patterns can be represented in the form of association rules. For example, the information that customers who purchase computers also tend to buy antivirus software at the same time is represented in Association Rule below:

$$\text{Computer} \Rightarrow \text{antivirus software} [\text{support} = 2\%, \text{confidence} = 60\%]$$

Rule support and confidence are two measures of rule interestingness. They respectively reflect the usefulness and certainty of discovered rules. A support of 2% for Association Rule means that 2% of all the transactions under analysis show that computer and antivirus software are purchased together. A confidence of 60% means that 60% of the customers who purchased a computer also bought the software. Typically, association rules are considered interesting if they satisfy both a minimum support threshold and a minimum confidence threshold. Such thresholds can be set by users or domain experts. Additional analysis can be performed to uncover interesting statistical correlations between associated items.

Frequent Itemsets, Closed Itemsets, and Association Rules

Let $I = \{I_1, I_2, \dots, I_m\}$ be a set of items. Let D , the task-relevant data, be a set of database transactions where each transaction T is a set of items such that $T \subseteq I$. Each transaction is associated with an identifier, called TID. Let A be a set of items. A transaction T is said to contain A if and only if $A \subseteq T$. An association rule is an implication of the form $A \Rightarrow B$, where $A \subset I$, $B \subset I$, and $A \cap B = \emptyset$. The rule $A \Rightarrow B$ holds in the transaction set D with support s , where s is the percentage of transactions in D that contain $A \cup B$ (i.e., the *union* of sets A and B , or say, both A and B). This is taken to be the probability, $P(A \cup B)$. The rule $A \Rightarrow B$ has confidence c in the transaction set D , where c is the percentage of transactions in D containing A that also contain B . This is taken to be the conditional probability, $P(B|A)$. That is,

$$\begin{aligned} \text{support}(A \Rightarrow B) &= P(A \cup B) \\ \text{confidence}(A \Rightarrow B) &= P(B|A). \end{aligned}$$

Rules that satisfy both a minimum support threshold (*min sup*) and a minimum confidence threshold (*min conf*) are called strong.

A set of items is referred to as an itemset. An itemset that contains k items is a k -itemset. The set $\{\text{computer}, \text{antivirus software}\}$ is a 2-itemset. The occurrence frequency of an itemset is the number of transactions that contain the itemset. This is also known, simply, as the frequency, support count, or count of the itemset. If the relative support of an itemset I satisfies a prespecified minimum support threshold, then I is a frequent itemset. The set of frequent k -itemsets is commonly denoted by L_k .

$$\text{confidence}(A \Rightarrow B) = P(B|A) = \frac{\text{support}(A \cup B)}{\text{support}(A)} = \frac{\text{support_count}(A \cup B)}{\text{support_count}(A)}.$$

the confidence of rule $A \Rightarrow B$ can be easily derived from the support counts of A and $A \cup B$. That is, once the support counts of A , B , and $A \cup B$ are found, it is straightforward to derive the corresponding association rules $A \Rightarrow B$ and $B \Rightarrow A$ and check whether they are strong. Thus the problem of mining association rules can be reduced to that of mining frequent itemsets.

In general, association rule mining can be viewed as a two-step process:

1. Find all frequent itemsets: By definition, each of these itemsets will occur at least as frequently as a predetermined minimum support count, *min sup*.
2. Generate strong association rules from the frequent itemsets: By definition, these rules must satisfy minimum support and minimum confidence.

An itemset X is closed in a data set S if there exists no proper super-itemset Y such that Y has the same support count as X in S . An itemset X is a closed frequent itemset in set S if X is both closed and frequent in S . An itemset X is a maximal frequent itemset (or max-itemset) in set S if X is frequent, and there exists no super-itemset Y such that $X \subset Y$ and Y is frequent in S .

Frequent Pattern Mining

Frequent pattern mining can be classified in various ways, based on the following criteria:

Based on the completeness of patterns to be mined: we can mine the complete set of frequent itemsets, the closed frequent itemsets, and the maximal frequent itemsets, given a minimum support threshold. We can also mine constrained frequent itemsets (i.e., those that satisfy a set of user-defined constraints), approximate frequent itemsets (i.e., those that derive only approximate support counts for the mined frequent itemsets), near-match frequent itemsets (i.e., those that tally the support count of the near or almost matching itemsets), top- k frequent itemsets (i.e., the k most frequent itemsets for a user-specified value, k), and so on. Different applications may have different requirements regarding the completeness of the patterns to be mined which in turn can lead to different evaluation and optimization methods.

Based on the levels of abstraction involved in the rule set: Some methods for association rule mining can find rules at differing levels of abstraction. For example, suppose that a set of association rules mined includes the following rules where X is a variable representing a customer:

$\text{buys}(X, \text{"computer"}) \Rightarrow \text{buys}(X, \text{"HP printer"})$

$buys(X, \text{"laptop computer"}) \Rightarrow buys(X, \text{"HP printer"})$.

the items bought are referenced at different levels of abstraction (e.g., “computer” is a higher-level abstraction of “laptop computer”). We refer to the rule set mined as consisting of multilevel association rules. The rules within a given set do not reference items or attributes at different levels of abstraction, then the set contains single-level association rules.

Based on the number of data dimensions involved in the rule: If the items or attributes in an association rule reference only one dimension, then it is a single-dimensional association rule. For example, could be rewritten as $buys(X, \text{"computer"}) \Rightarrow buys(X, \text{"antivirus software"})$.

If a rule references two or more dimensions, such as the dimensions *age*, *income*, and *buys*, then it is a multidimensional association rule.

$age(X, \text{"30. . .39"}) \wedge income(X, \text{"42K. . .48K"}) \Rightarrow buys(X, \text{"high resolution TV"})$.

Based on the types of values handled in the rule: If a rule involves associations between the presence or absence of items, it is a Boolean association rule. If a rule describes associations between quantitative items or attributes, then it is a quantitative association rule. Note that the quantitative attributes, *age* and *income*, have been discretized.

Based on the kinds of rules to be mined: Frequent pattern analysis can generate various kinds of rules and other interesting relationships. Association rules are the most popular kind of rules generated from frequent patterns. Typically, such mining can generate a large number of rules, many of which are redundant or do not indicate a correlation relationship among itemsets. Thus, the discovered associations can be further analyzed to uncover statistical correlations, leading to correlation rules.

We can also mine strong gradient relationships among itemsets, where a gradient is the ratio of the measure of an item when compared with that of its parent (a generalized itemset), its child (a specialized itemset), or its sibling (a comparable itemset). One such example is: “*The average sales from Sony Digital Camera increase over 16% when sold together with Sony Laptop Computer*”: both Sony Digital Camera and Sony Laptop Computer are siblings, where the parent itemset is Sony.

Based on the kinds of patterns to be mined:

Many kinds of frequent patterns can be mined from different kinds of data sets. For this chapter, our focus is on frequent itemset mining, that is, the mining of frequent itemsets (sets of items) from transactional or relational data sets. However, other kinds of frequent patterns can be found from other kinds of data sets. Sequential pattern mining searches for frequent subsequences in a sequence data set, where a sequence records an ordering of events. For example, with sequential pattern mining, we can study the order in which items are frequently purchased. For instance, customers may tend to first buy a PC, followed by a digital camera, and then a memory card. Structured pattern mining searches for frequent substructures in a structured data set. Notice that *structure* is a general concept that covers many different kinds of structural forms, such as graphs, lattices, trees, sequences, sets, single items, or combinations of such structures. Single

items are the simplest form of structure. Each element of an itemset may contain a subsequence, a subtree, and so on, and such containment relationships can be defined recursively. Therefore, structured pattern mining can be considered as the most general form of frequent pattern mining.

The Apriori Algorithm (For numerical kindly refer to your notes)

Algorithm uses *prior knowledge* of frequent itemset properties. Apriori employs an iterative approach known as a *level-wise* search, where k -itemsets are used to explore $(k+1)$ -itemsets. The set of frequent 1-itemsets is found by scanning the database to accumulate the count for each item, and collecting those items that satisfy minimum support. The resulting set is denoted L_1 . Next, L_1 is used to find L_2 , the set of frequent 2-itemsets, which is used to find L_3 , and so on, until no more frequent k -itemsets can be found. The finding of each L_k requires one full scan of the database. Apriori property: *All nonempty subsets of a frequent itemset must also be frequent.* This property belongs to a special category of properties called anti monotone in the Sense that *if a set cannot pass a test, all of its supersets will fail the same test as well.*

A two-step process is followed:

The join step: To find L_k , a set of candidate k -itemsets is generated by joining L_{k-1} with itself.

Prune Step: A scan of the database to determine the count of each candidate would result in the determination of L_k .

Generating Association Rules from Frequent Itemsets

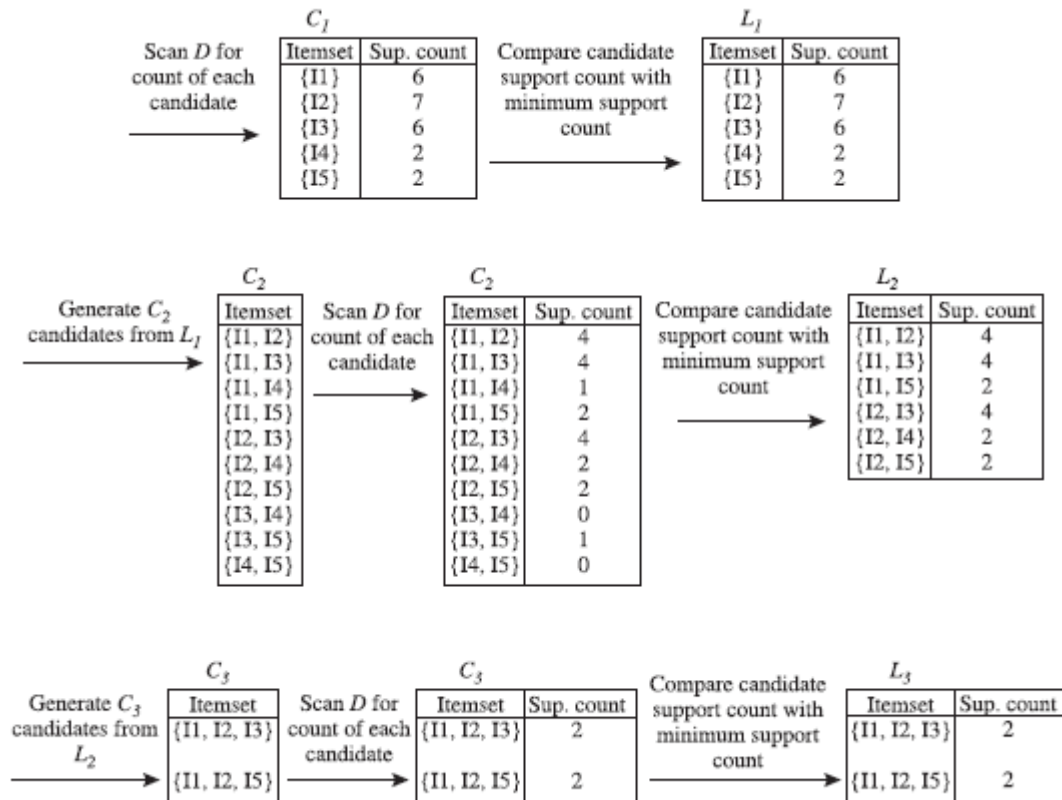
$$confidence(A \Rightarrow B) = P(B|A) = \frac{support_count(A \cup B)}{support_count(A)}.$$

The conditional probability is expressed in terms of itemset support count, where *support count*($A \cup B$) is the number of transactions containing the itemsets $A \cup B$, and *support count*(A) is the number of transactions containing the itemset A . Based on this equation, association rules can be generated

Q. Find the frequent itemset and association rules for the given data. Minimum support count is given as 2 and confidence is 70 %.

<i>TID</i>	<i>List of item IDs</i>
T100	I1, I2, I5
T200	I2, I4
T300	I2, I3
T400	I1, I2, I4
T500	I1, I3
T600	I2, I3
T700	I1, I3
T800	I1, I2, I3, I5
T900	I1, I2, I3

Solution:



Generating association rules. Suppose the data contain the frequent itemset $l = \{I1, I2, I5\}$. What are the association rules that can be generated from l ? The nonempty subsets of l are $\{I1, I2\}$, $\{I1, I5\}$, $\{I2, I5\}$, $\{I1\}$, $\{I2\}$, and $\{I5\}$. The resulting association rules are as shown below, each listed with its confidence

$I1 \wedge I2 \Rightarrow I5,$	$confidence = 2/4 = 50\%$
$I1 \wedge I5 \Rightarrow I2,$	$confidence = 2/2 = 100\%$
$I2 \wedge I5 \Rightarrow I1,$	$confidence = 2/2 = 100\%$
$I1 \Rightarrow I2 \wedge I5,$	$confidence = 2/6 = 33\%$
$I2 \Rightarrow I1 \wedge I5,$	$confidence = 2/7 = 29\%$
$I5 \Rightarrow I1 \wedge I2,$	$confidence = 2/2 = 100\%$

If the minimum confidence threshold is, say, 70%, then only the second, third, and last rules above are output, because these are the only ones generated that are strong.

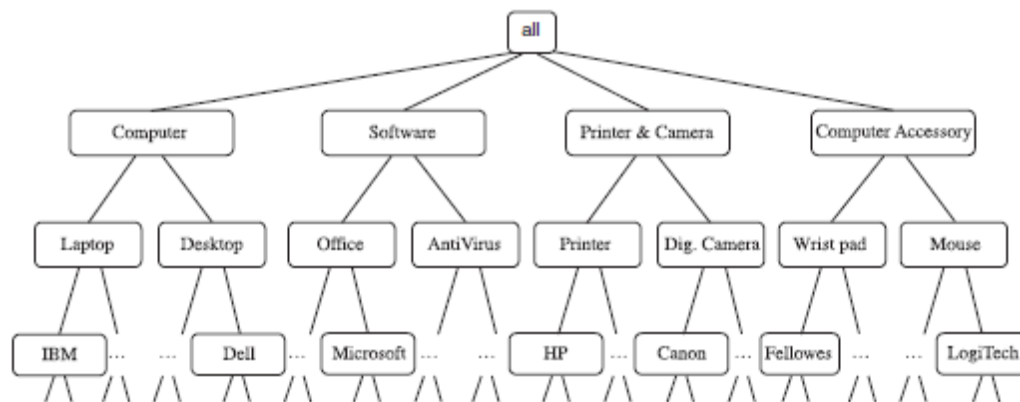
Similarly generate for itemset $I = \{I1, I2, I3\}$.

Mining Various Kinds of Association Rules

For many applications, it is difficult to find strong associations among data items at low or primitive levels of abstraction due to the sparsity of data at those levels. Strong associations discovered at high levels of abstraction may represent commonsense knowledge. Data mining systems should provide capabilities for mining association rules at multiple levels of abstraction, with sufficient flexibility for easy traversal among different abstraction spaces.

Mining multilevel association rules. A concept hierarchy defines a sequence of mappings from a set of low-level concepts to higher level, more general concepts. Data can be generalized by replacing low-level concepts within the data by their higher-level concepts, or *ancestors*, from a concept hierarchy.

<i>TID</i>	<i>Items Purchased</i>
T100	IBM-ThinkPad-T40/2373, HP-Photosmart-7660
T200	Microsoft-Office-Professional-2003, Microsoft-Plus!-Digital-Media
T300	Logitech-MX700-Cordless-Mouse, Fellowes-Wrist-Rest
T400	Dell-Dimension-XPS, Canon-PowerShot-S400
T500	IBM-ThinkPad-R40/P4M, Symantec-Norton-Antivirus-2003
...	...



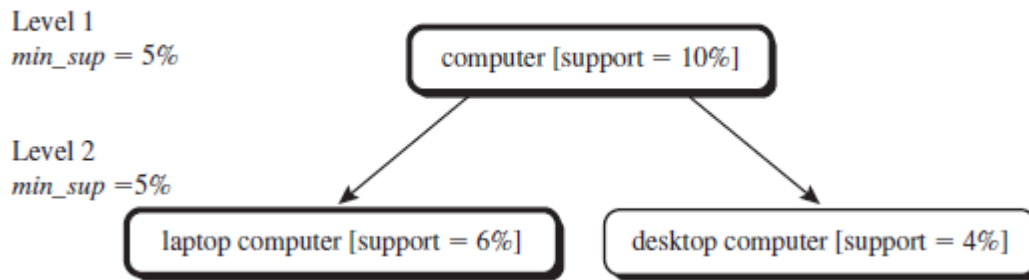
It is difficult to find interesting purchase patterns at such raw or primitive-level data. For instance “IBM-ThinkPad-R40 occurs in a very small fraction of the transactions, then it can be difficult to find strong associations involving these specific items. However, we would expect that it is easier to find strong associations between generalized abstractions of these items.

Association rules generated from mining data at multiple levels of abstraction are called multiple-level or multilevel association rules. Multilevel association rules can be mined efficiently using concept hierarchies under a support-confidence framework. In general, a top-down strategy is employed, where counts are accumulated for the calculation of frequent itemsets at each concept level, starting at the concept level

1 and working downward in the hierarchy toward the more specific concept levels, until no more frequent itemsets can be found. A number of variations to this approach are described below, where each variation involves “playing” with the support threshold in a slightly different way.

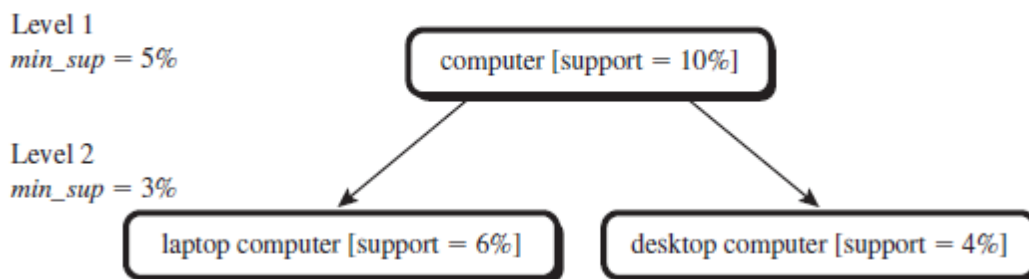
Using uniform minimum support for all levels (referred to as uniform support): The same minimum support threshold is used when mining at each level of abstraction. When a uniform minimum support threshold is used, the search procedure is simplified. The method is also simple in that users are required to specify only one minimum support threshold. For example, in Figure a minimum support threshold of 5% is used throughout (e.g., for mining from “computer” down to “laptop computer”). Both “computer” and “laptop computer” are found to be frequent, while “desktop computer” is not. Nodes with thick borders indicate that an examined item or frequent itemset.

Disadvantages: If the minimum support threshold is set too high, it could miss some meaningful associations occurring at low abstraction levels. If the threshold is set too low, it may generate many uninteresting associations occurring at high abstraction levels.



Using reduced minimum support at lower levels (referred to as reduced support): Each level of abstraction has its own minimum support threshold. The deeper the level of abstraction, the smaller the corresponding threshold is. Nodes with thick borders indicate that an examined item or frequent itemset.

the minimum support thresholds for levels 1 and 2 are 5% and 3%, respectively. In this way, “computer,” “laptop computer,” and “desktop computer” are all considered frequent.



Using item or group-based minimum support (referred to as group-based support): Users or experts often have insight as to which groups are more important than others, it is sometimes more desirable to set up user-specific, item, or group based minimal support thresholds when mining multilevel rules. For example, a user could set up the minimum support thresholds based on product price or on items of interest, such as by setting particularly low support thresholds for *laptop computers* and *flash drives* in order to pay particular attention to the association patterns containing items in these categories.

A serious side effect of mining multilevel association rules is its generation of many redundant rules across multiple levels of abstraction due to the “ancestor” relationships among items. For example, consider the following rules where “laptop computer” is an ancestor of “IBM laptop computer” based on the concept hierarchy.

buys(X, “laptop computer”)⇒*buys*(X, “HP printer”) [*support* = 8%, *confidence* = 70%] rule 1
buys(X, “IBM laptop computer”)⇒*buys*(X, “HP printer”) [*support* = 2%, *confidence* = 72%]
 rule 2

“If both rules are mined, then how useful is the latter rule?” If the latter, less general rule does not provide new information, then it should be removed. Let’s look at how this may be determined. A rule $R1$ is an ancestor of a rule $R2$, if $R1$ can be obtained by replacing the items in $R2$ by their ancestors in a concept hierarchy. For example, Rule 1 is an ancestor of Rule 2 because *“laptop computer”* is an ancestor of *“IBM laptop computer.”* Based on this definition, a rule can be considered redundant if its support and confidence are close to their “expected” values, based on an ancestor of the rule. Suppose that Rule 1 has a 70% confidence and 8% support, and that about one-quarter of all *“laptop computer”* sales are for *“IBM laptop computers.”* We may expect Rule 2 to have a confidence of around 70% (since all data samples of *“IBM laptop computer”* are also samples of *“laptop computer”*) and a support of around 2%.

Mining Multidimensional Association Rules

Single dimension or intradimensional association rule: association rules that imply a single predicate

$buys(X, \text{“digital camera”}) \Rightarrow buys(X, \text{“HP printer”})$.

It is considered as a singledimensional or intradimensional association rule because it contains a single distinct predicate (e.g., *buys*) with multiple occurrences (i.e., the predicate occurs more than once within the rule).

Multidimensional association Rule: Suppose, however, that rather than using a transactional database, sales and related information are stored in a relational database or data warehouse. Such data stores are multidimensional, by definition. For instance, in addition to keeping track of the items purchased in sales transactions, a relational database may record other attributes associated with the items, such as the quantity purchased or the price, or the branch location of the sale. Additional relational information regarding the customers who purchased the items, such as customer age, occupation, credit rating, income, and address, may also be stored. Considering each database attribute or warehouse dimension as a predicate, we can therefore mine association rules containing *multiple* predicates, such as

$$age(X, \text{“20...29”}) \wedge occupation(X, \text{“student”}) \Rightarrow buys(X, \text{“laptop”})$$

Association rules that involve two or more dimensions or predicates can be referred to as multidimensional association rules. It contains three predicates (*age*, *occupation*, and *buys*), each of which occurs *only once* in the rule. Hence, we say that it has no repeated predicates. Multidimensional association rules with no repeated predicates are called inter dimensional association rules.

Hybrid association rule: We can also mine multidimensional association rules with repeated predicates, which contain multiple occurrences of some predicates. These rules are called hybrid-dimensional association rules. An example of such a rule is the following, where the predicate *buys* is repeated:

$$age(X, "20...29") \wedge buys(X, "laptop") \Rightarrow buys(X, "HP printer")$$

Note that database attributes can be categorical or quantitative. Categorical attributes have a finite number of possible values, with no ordering among the values (e.g., *occupation*, *brand*, *color*). Categorical attributes are also called nominal attributes, because their values are “names of things.” Quantitative attributes are numeric and have an implicit ordering among values (e.g., *age*, *income*, *price*). Techniques for mining multidimensional association rules can be categorized into two basic approaches regarding the treatment of quantitative attributes.

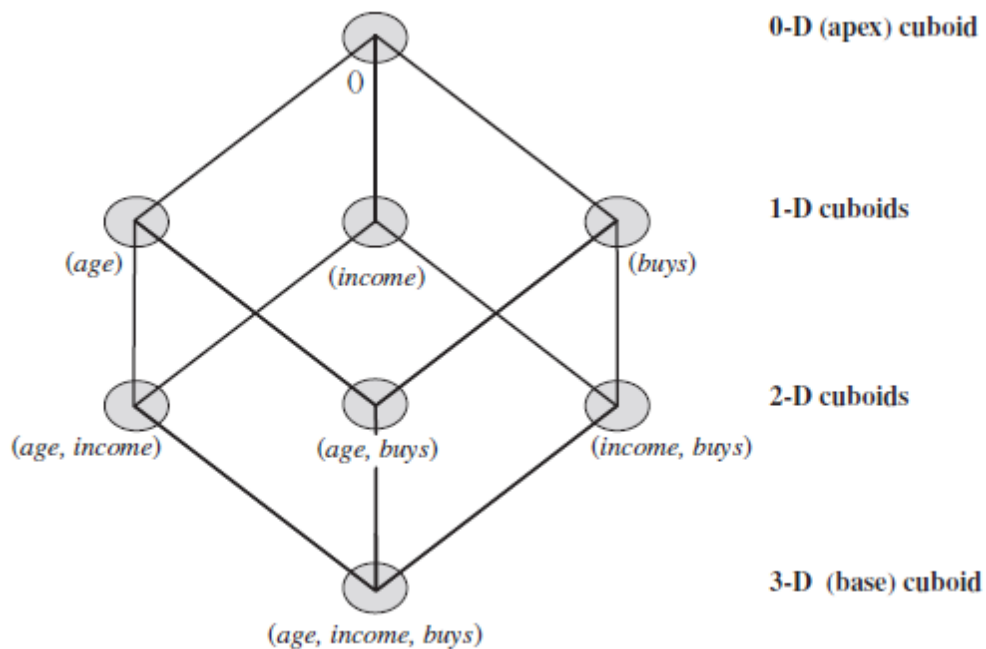
In the first approach, *quantitative attributes are discretized using predefined concept Hierarchies*. This discretization occurs before mining. For instance, a concept hierarchy for *income* may be used to replace the original numeric values of this attribute by interval labels, such as “0 . . . 20K”, “21K . . . 30K”, “31K . . . 40K”, and so on. Here discretization is *static* and predetermined. We refer to this as mining multidimensional association rules using static discretization of quantitative attributes.

In the second approach, *quantitative attributes are discretized or clustered into “bins” based on the distribution of the data*. These bins may be further combined during the mining process. The discretization process is *dynamic* and established so as to satisfy some mining criteria, such as maximizing the confidence of the rules mined. Because this strategy treats the numeric attribute values as quantities rather than as predefined ranges or categories, association rules mined from this approach are also referred to as (dynamic) quantitative association rules.

Mining Multidimensional Association Rules Using Static Discretization of Quantitative Attributes

Quantitative attributes, are discretized before mining using predefined concept hierarchies or data discretization techniques, where numeric values are replaced by interval labels. Categorical attributes may also be generalized to higher conceptual levels if desired. If the resulting task-relevant data are stored in a relational table, then any of the frequent itemset mining algorithms we have discussed can be modified easily so as to find all frequent predicate sets rather than frequent itemsets. In particular, instead of searching on only one attribute like *buys*, we need to search through all of the relevant attributes, treating each attribute-value pair as an itemset.

Alternatively, the transformed multidimensional data may be used to construct a *data cube*. Data cubes are well suited for the mining of multidimensional association rules: They store aggregates (such as counts), in multidimensional space, which is essential for computing the support and confidence of multidimensional association rules. Figure below shows the lattice of cuboids defining a data cube for the dimensions *age*, *income*, and *buys*. The cells of an *n*-dimensional cuboid can be used to store the support counts of the corresponding *n*-predicate sets. The base cuboid aggregates the task-relevant data by *age*, *income*, and *buys*; the 2-D cuboid, (*age*, *income*), aggregates by *age* and *income*, and so on; the 0-D (apex) cuboid contains the total number of transactions in the task-relevant data.



Mining Quantitative Association Rules (*dynamically* discretized)

Quantitative association rules are multidimensional association rules in which the numeric attributes are *dynamically* discretized during the mining process so as to satisfy some mining criteria, such as maximizing the confidence or compactness of the rules mined. In this section, we focus specifically on how to mine quantitative association rules having two quantitative attributes on the left-hand side of the rule and one categorical attribute on the right-hand side of the rule.

$$Aquan1 \wedge Aquan2 \Rightarrow Acat$$

where *Aquan1* and *Aquan2* are tests on quantitative attribute intervals (where the intervals are dynamically determined), and *Acat* tests a categorical attribute from the task-relevant data. Such rules have been referred to as two-dimensional quantitative association rules, because they contain two quantitative dimensions. For instance, suppose you are curious about the association relationship between pairs of quantitative attributes, like customer age and income, and the type of television (such as *high-definition TV*, i.e., *HDTV*) that customers like to buy. An example of such a 2-D quantitative association rule is

$$age(X, "30...39") \wedge income(X, "42K...48K") \Rightarrow buys(X, "HDTV")$$

How can we find such rules? An approach used in a system called ARCS (Association Rule Clustering System) which borrows ideas from image processing. Essentially, this approach maps pairs of quantitative attributes onto a 2-D grid for tuples satisfying a given categorical attribute

condition. The grid is then searched for clusters of points from which the association rules are generated. The following steps are involved in ARCS:

Binning: Quantitative attributes can have a very wide range of values defining their domain. Just think about how big a 2-D grid would be if we plotted *age* and *income* as axes, where each possible value of *age* was assigned a unique position on one axis, and similarly, each possible value of *income* was assigned a unique position on the other axis! To keep grids down to a manageable size, we instead partition the ranges of quantitative attributes into intervals. These intervals are dynamic in that they may later be further combined during the mining process. The partitioning process is referred to as binning, that is, where the intervals are considered “bins”. Three common binning strategies are as follows:

- Equal-width binning
- Equal-frequency binning
- Clustering-based binning

ARCS uses equal-width binning, where the bin size for each quantitative attribute is input by the user. A 2-D array for each possible bin combination involving both quantitative attributes is created. Each array cell holds the corresponding count distribution for each possible class of the categorical attribute of the rule right-hand side. By creating this data structure, the task-relevant data need only be scanned once. The same 2-D array can be used to generate rules for any value of the categorical attribute, based on the same two quantitative attributes.

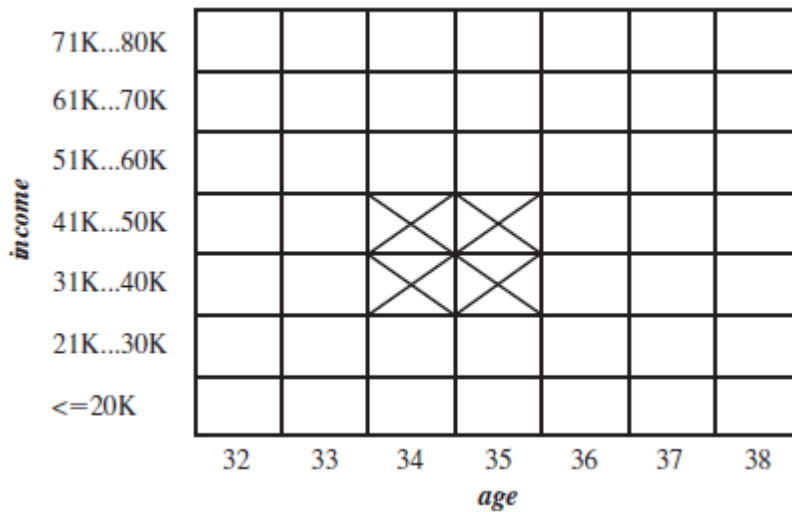
Finding frequent predicate sets: Once the 2-D array containing the count distribution for each category is set up, it can be scanned to find the frequent predicate sets (those satisfying minimum support) that also satisfy minimum confidence. Strong association rules can then be generated from these predicate sets, using a rule generation algorithm.

Clustering the association rules: The strong association rules obtained in the previous step are then mapped to a 2-D grid. Figure shows a 2-D grid for 2-D quantitative association rules predicting the condition *buys*(*X*, “HDTV”) on the rule right-hand side, given the quantitative attributes *age* and *income*.

$age(X, 34) \wedge income(X, “31K...40K”) \Rightarrow buys(X, “HDTV”)$
 $age(X, 34) \wedge income(X, “31K...40K”) \Rightarrow buys(X, “HDTV”)$
 $age(X, 34) \wedge income(X, “41K...50K”) \Rightarrow buys(X, “HDTV”)$
 $age(X, 35) \wedge income(X, “41K...50K”) \Rightarrow buys(X, “HDTV”).$

Notice that these rules are quite “close” to one another, forming a rule cluster on the grid. Indeed, the four rules can be combined or “clustered” together to form the following simpler rule, which subsumes and replaces the above four rules:

$age(X, “34...35”) \wedge income(X, “31K...50K”) \Rightarrow buys(X, “HDTV”)$



ARCS employs a clustering algorithm for this purpose. The algorithm scans the grid, searching for rectangular clusters of rules. In this way, bins of the quantitative attributes occurring within a rule cluster may be further combined, and hence further dynamic discretization of the quantitative attributes occurs.

The grid-based technique described here assumes that the initial association rules can be clustered into rectangular regions. Before performing the clustering, smoothing techniques can be used to help remove noise and outliers from the data. Rectangular clusters may oversimplify the data. Alternative approaches have been proposed, based on other shapes of regions that tend to better fit the data, yet require greater computation effort.

A non-grid-based technique has been proposed to find quantitative association rules that are more general, where any number of quantitative and categorical attributes can appear on either side of the rules. In this technique, quantitative attributes are dynamically partitioned using equal-frequency binning, and the partitions are combined based on a measure of *partial completeness*, which quantifies the information lost due to partitioning.

Unit 4 : Classification and Prediction

Classification and prediction are two forms of data analysis that can be used to extract models describing important data classes or to predict future data trends. Such analysis can help provide us with a better understanding of the data at large. Whereas *classification* predicts categorical (discrete, unordered) labels, *prediction* models continuous valued functions. For example, we can build a classification model to categorize bank loan applications as either safe or risky, or a prediction model to predict the expenditures in dollars of potential customers on computer equipment given their income and occupation.

A bank loans officer needs analysis of her data in order to learn which loan applicants are “safe” and which are “risky” for the bank. A marketing manager at *AllElectronics* needs data analysis to help guess whether a customer with a given profile will buy a new computer. A medical researcher wants to analyze breast cancer data in order to predict which one of three specific treatments a patient should receive. In each of these examples, the data analysis task is classification, where a model or classifier is constructed to predict *categorical labels*, such as “safe” or “risky” for the loan application data; “yes” or “no” for the marketing data; or “treatment A,” “treatment B,” or “treatment C” for the medical data. These categories can be represented by discrete values, where the ordering among values has no meaning.

Data classification is a two-step process. In the first step, a classifier is built describing a predetermined set of data classes or concepts. This is the 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, \mathbf{X} , is represented by an n -dimensional attribute vector, $\mathbf{X} = (x_1, x_2, \dots, x_n)$, depicting n measurements made on the tuple from n database attributes, respectively, A_1, A_2, \dots, A_n .¹ Each tuple, \mathbf{X} , is assumed to belong to a predefined class as determined by another database attribute called the class label attribute. The class label attribute is discrete-valued and unordered. It is *categorical* in that each value serves as a category or class. The individual tuples making up the training set are referred to as training tuples and are selected from the database under analysis. 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). 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.

This first step of the classification process can also be viewed as the learning of a mapping or function, $y = f(\mathbf{X})$, that can predict the associated class label y of a given tuple \mathbf{X} . In this view, we wish to learn a mapping or function that separates the data classes. Typically, this mapping is represented in the form of classification rules, decision trees, or mathematical formulae.

In the second step 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 accuracy of the classifier, a test set is used, made up of test tuples and their associated class labels. These tuples are randomly selected from the general data set. They are independent of the training tuples, meaning that they are 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. (Such data are also referred to in the machine learning literature as “*unknown*” or “*previously unseen*” data).

Prediction

“How is (numeric) prediction different from classification?” Data prediction is a two step process, similar to that of data classification. However, for prediction, we lose the terminology of “class label attribute” because the attribute for which values are being predicted is continuous-valued (ordered) rather than categorical (discrete-valued and unordered). The attribute can be referred to simply as the predicted attribute.³ Suppose that, in our example, we instead wanted to predict the amount (in dollars) that would be “safe” for the bank to loan an applicant. The data mining task becomes prediction, rather than

classification. We would replace the categorical attribute, *loan decision*, with the continuous-valued *loan amount* as the predicted attribute, and build a predictor for our task.

Note that prediction can also be viewed as a mapping or function, $y = f(X)$, where X is the input (e.g., a tuple describing a loan applicant), and the output y is a continuous or ordered value (such as the predicted amount that the bank can safely loan the applicant); That is, we wish to learn a mapping or function that models the relationship between X and y .

Preparing the Data for Classification and Prediction

The following preprocessing steps may be applied to the data to help improve the accuracy, efficiency, and scalability of the classification or prediction process:

Data cleaning: This refers to the preprocessing of data in order to remove or reduce *noise* (by applying smoothing techniques, for example) and the treatment of *missing values* (e.g., by replacing a missing value with the most commonly occurring value for that attribute, or with the most probable value based on statistics). Although most classification algorithms have some mechanisms for handling noisy or missing data, this step can help reduce confusion during learning.

Relevance analysis: Many of the attributes in the data may be *redundant*. Correlation analysis can be used to identify whether any two given attributes are statistically related. A strong correlation between attributes A_1 and A_2 would suggest that one of the two could be removed from further analysis. A database may also contain *irrelevant* attributes. Attribute subset selection⁴ can be used in these cases to find a reduced set of attributes such that the resulting probability distribution of the data classes is as close as possible to the original distribution obtained using all attributes. Ideally, the time spent on relevance analysis, when added to the time spent on learning from the resulting “reduced” attribute (or feature) subset, should be less than the time that would have been spent on learning from the original set of attributes. Hence, such analysis can help improve classification efficiency and scalability.

Data transformation and reduction: The data may be transformed by normalization, particularly when neural networks or methods involving distance measurements are used in the learning step. Normalization involves scaling all values for a given attribute so that they fall within a small specified range, such as -1.0 to 1.0 , or 0.0 to 1.0 . In methods that use distance measurements, for example, this would prevent attributes with initially large ranges (like, say, *income*) from outweighing attributes with initially smaller ranges (such as binary attributes). The data can also be transformed by *generalizing* it to higher-level concepts. Concept hierarchies may be used for this purpose. This is particularly useful for continuous valued attributes. For example, numeric values for the attribute *income* can be generalized to discrete ranges, such as *low*, *medium*, and *high*. Similarly, categorical attributes, like *street*, can be generalized to higher-level concepts, like *city*. Because generalization compresses the original training data, fewer input/output operations may be involved during learning.

Characteristics of Classification and prediction methods

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). Similarly, the accuracy of a predictor refers to how well a given predictor can guess the value of the predicted attribute for new or previously unseen data. Accuracy can be estimated using one or more test sets that are independent of the training set.

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

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.

Classification by Decision Tree Induction

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

“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.

Advantages:

- 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.
- Their representation of acquired knowledge in tree form is intuitive and generally easy to assimilate by humans.
- The learning and classification steps of decision tree induction are simple and fast.
- 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.

Most algorithms for decision tree induction also follow such a top-down approach, which starts with a training set of tuples and their associated class labels. The training set is recursively partitioned into smaller subsets as the tree is being built.

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

Input:

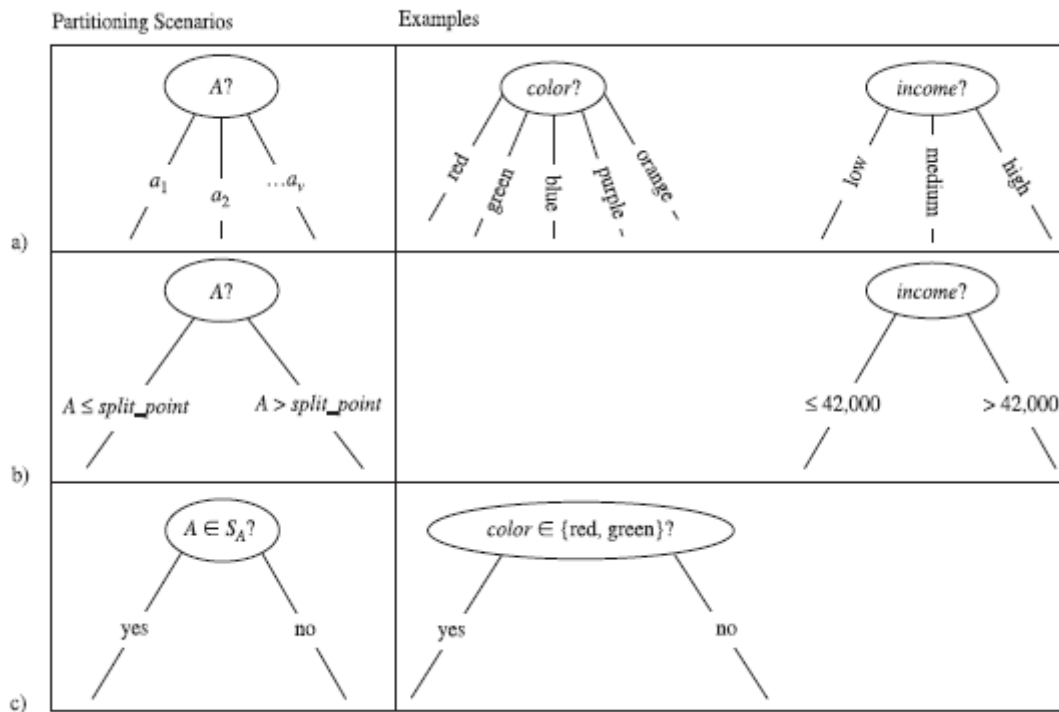
- Data partition, D , which is a set of training tuples and their associated class labels;
- *attribute_list*, the set of candidate attributes;
- *Attribute_selection_method*, a procedure to determine the splitting criterion that “best” partitions the data tuples into individual classes. This criterion 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* \leftarrow *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 D_j be the set of data tuples in D satisfying outcome j ; // a partition
- (12) **if** D_j 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*(D_j , *attribute_list*) to node N ;
- endfor**
- (15) return N ;

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. The splitting criterion also tells us which branches to grow from node N with respect to the outcomes of the chosen test. More specifically, the splitting criterion indicates the splitting attribute and may also indicate either a split-point or a splitting subset. The splitting criterion is determined so that, ideally, the resulting partitions at each branch are as “pure” as possible. A partition is pure if all of the tuples in it belong to the same class. In other words, if we were to split up the tuples in D according to the mutually exclusive outcomes of the splitting criterion, we hope for the resulting partitions to be as pure as possible. There are three possible scenarios, as illustrated in Figure Let A be the splitting attribute.



- (a) If A is discrete-valued, then one branch is grown for each known value of A .
- (b) If A is continuous-valued, then two branches are grown, corresponding to $A \leq \text{split_point}$ and $A > \text{split_point}$.
- (c) If A is discrete-valued and a binary tree must be produced, then the test is of the form $A \in S_A$, where S_A is the splitting subset for A

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. The attribute selection measure provides a ranking for each attribute describing the given training tuples. The attribute having the best score for the measure is chosen as the *splitting attribute* for the given tuples.

Information gain

information gain is an attribute selection measure. The attribute with the highest information gain is chosen as the splitting attribute for node N . Let node N represent or hold the tuples of partition D . $Info(D)$ is also known as the entropy of D .

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

where p_i is the probability that an arbitrary tuple in D belongs to class C_i . Attribute A can be used to split D into v partitions or subsets, $\{D_1, D_2, \dots, D_v\}$, where D_j contains those tuples in D that have outcome a_j of A . This amount is measured by

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

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).

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

$Gain(A)$ tells us how much would be gained by branching on A. The attribute A with the highest information gain, ($Gain(A)$), is chosen as the splitting attribute at node N.

Q

<i>RID</i>	<i>age</i>	<i>income</i>	<i>student</i>	<i>credit_rating</i>	<i>Class: buys_computer</i>
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

Solution: In this example, each attribute is discrete-valued. Continuous-valued attributes have been generalized.) The class label attribute, *buys computer*, has two distinct values (namely, {*yes*, *no*}); To find the splitting criterion for these tuples, we must compute the information gain of each attribute.

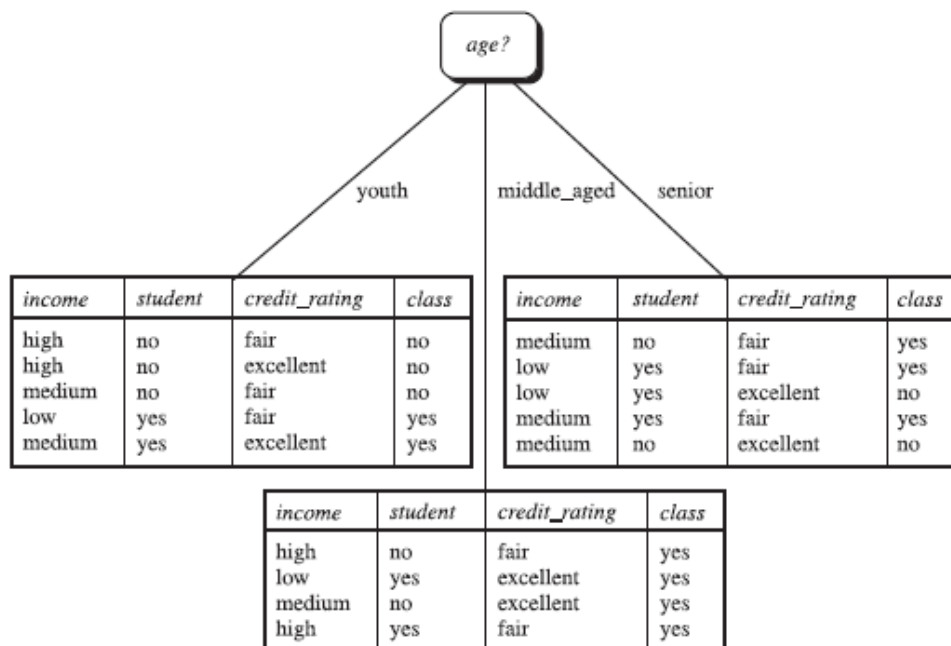
$$Info(D) = -\frac{9}{14} \log_2\left(\frac{9}{14}\right) - \frac{5}{14} \log_2\left(\frac{5}{14}\right) = 0.940 \text{ bits.}$$

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.

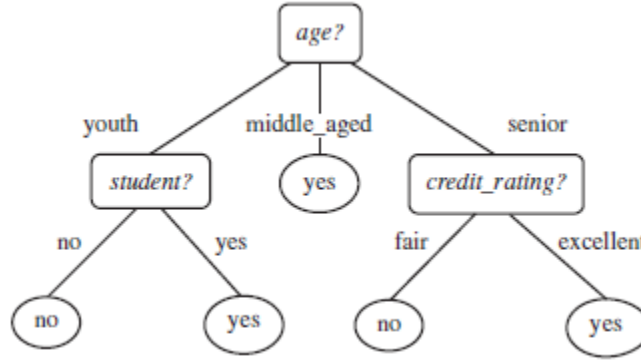
$$\begin{aligned}
 Info_{age}(D) &= \frac{5}{14} \times \left(-\frac{2}{5} \log_2 \frac{2}{5} - \frac{3}{5} \log_2 \frac{3}{5} \right) \\
 &\quad + \frac{4}{14} \times \left(-\frac{4}{4} \log_2 \frac{4}{4} - \frac{0}{4} \log_2 \frac{0}{4} \right) \\
 &\quad + \frac{5}{14} \times \left(-\frac{3}{5} \log_2 \frac{3}{5} - \frac{2}{5} \log_2 \frac{2}{5} \right) \\
 &= 0.694 \text{ bits.}
 \end{aligned}$$

Hence, the gain in information from such a partitioning would be
 $Gain(age) = Info(D) - Info_{age}(D) = 0.940 - 0.694 = 0.246$ 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.



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 is as follows.



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. The information gained by partitioning on this attribute is maximal. Clearly, such a partitioning is useless for classification.

An extension to information gain known as *gain ratio*, which attempts to overcome this bias. A kind of normalization to information gain using a “split information” value.

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

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 A . Note that, for each outcome, it considers the number of tuples having that outcome with respect to the total number of tuples in D . It differs from information gain, which measures the information with respect to classification that is acquired based on the same partitioning. The gain ratio is defined as

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

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

Example: Computation of gain ratio for the attribute *income*. A test on *income* splits the data of Table into three partitions, namely *low*, *medium*, and *high*, containing four, six, and four tuples, respectively. To compute the gain ratio of *income*,

$$\begin{aligned}
 SplitInfo_A(D) &= -\frac{4}{14} \times \log_2 \left(\frac{4}{14} \right) - \frac{6}{14} \times \log_2 \left(\frac{6}{14} \right) - \frac{4}{14} \times \log_2 \left(\frac{4}{14} \right). \\
 &= 0.926.
 \end{aligned}$$

we have $Gain(income) = 0.029$. Therefore, $GainRatio(income) = 0.029/0.926 = 0.031$

Gini index

The Gini index measures the impurity of D . The Gini index considers a binary split for each attribute.

A is a discrete-valued attribute having v distinct values, $\{a_1, a_2, \dots, a_v\}$, 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 . Each subset, SA , can be considered as a binary test for attribute A . 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 $2v-2$ possible ways to form two partitions of the data, D , based on a binary split on A .

if a binary split on A partitions D into D_1 and D_2 , 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.

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

where p_i is the probability that a tuple in D belongs to class C_i

Induction of a decision tree using gini index.:

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 D_1 satisfying the condition "*income* $\{low, medium\}$." The remaining four tuples of D would be assigned to partition D_2 . 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 0.367 and 0.429, respectively. The attribute *income* and splitting subset $\{medium, high\}$ therefore give the minimum gini index overall. Hence, the Gini index has selected *income* instead of *age* at the root node, unlike the (nonbinary) tree created by information gain.

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. Pruned trees tend to be smaller and less complex and, thus, easier to comprehend. They are usually faster and better at correctly classifying independent test data (i.e., of previously unseen tuples) than unpruned trees.

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

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.

The second and more common approach is 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.

Bayesian Classification

Bayesian classifiers are statistical classifiers. They can predict class membership probabilities, such as the probability that a given tuple belongs to a particular class. Bayesian classifiers have also exhibited high accuracy and speed when applied to large databases. Bayesian classification is based on Bayes’ theorem,

Predicting a class label using naïve Bayesian classification.

Example: 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*}).

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

We need to maximize $P(X|C_i)P(C_i)$

$$P(buys_computer = yes) = 9/14 = 0.643$$

$$P(buys_computer = no) = 5/14 = 0.357$$

To compute $PX|C_i$, for $i = 1, 2$, we compute the following conditional probabilities:

$$P(age = youth \mid buys_computer = yes) = 2/9 = 0.222$$

$$P(age = youth \mid buys_computer = no) = 3/5 = 0.600$$

$$P(income = medium \mid buys_computer = yes) = 4/9 = 0.444$$

$$P(income = medium \mid buys_computer = no) = 2/5 = 0.400$$

$$P(student = yes \mid buys_computer = yes) = 6/9 = 0.667$$

$$P(student = yes \mid buys_computer = no) = 1/5 = 0.200$$

$$P(credit_rating = fair \mid buys_computer = yes) = 6/9 = 0.667$$

$$P(credit_rating = fair \mid buys_computer = no) = 2/5 = 0.400$$

Using the above probabilities, we obtain

$$\begin{aligned}P(X|buys_computer = yes) &= P(age = youth \mid buys_computer = yes) \times \\&\quad P(income = medium \mid buys_computer = yes) \times \\&\quad P(student = yes \mid buys_computer = yes) \times \\&\quad P(credit_rating = fair \mid buys_computer = yes) \\&= 0.222 \times 0.444 \times 0.667 \times 0.667 = 0.044.\end{aligned}$$

Similarly,

$$P(X|buys_computer = no) = 0.600 \times 0.400 \times 0.200 \times 0.400 = 0.019.$$

To find the class, C_i , that maximizes $P(X|C_i)P(C_i)$, we compute

$$P(X|buys_computer = yes)P(buys_computer = yes) = 0.044 \times 0.643 = 0.028$$

$$P(X|buys_computer = no)P(buys_computer = no) = 0.019 \times 0.357 = 0.007$$

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

Classification by Backpropagation

Backpropagation learns by iteratively processing a data set of training tuples, comparing the network's prediction for each tuple with the actual known *target* value. The target value may be the known class label of the training tuple (for classification problems) or a continuous value (for prediction). For each training tuple, the weights are modified so as to minimize the mean squared error between the network's prediction and the actual target value. These modifications are made in the "backwards" direction, that is, from the output layer, through each hidden layer down to the first hidden layer (hence the name *backpropagation*).

Backpropagation is a neural network learning algorithm. Neural network is a set of connected input/output 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 the input tuples. Neural networks involve long training times and are therefore more suitable for applications where this is feasible.

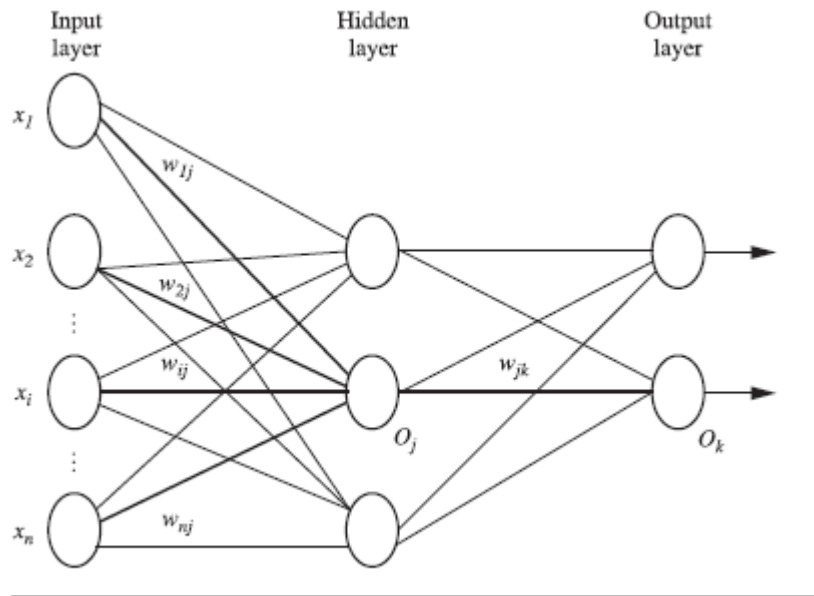
Disadvantage: Neural networks have been criticized for their poor interpretability.

Advantages: high tolerance of noisy data as well as their ability to classify patterns on which they have not been trained.

They can be used when you may have little knowledge of the relationships between attributes and classes. They have been successful on a wide array of real-world data, including handwritten character recognition, pathology and laboratory medicine, and training a computer to pronounce English text.

Multilayer Feed-Forward Neural Network

A multilayer feed-forward neural network consists of an *input layer*, one or more *hidden layers*, and an *output layer*. The units in the input layer are called input units. The units in the hidden layers and output layer are sometimes referred to as neurodes, due to their symbolic biological basis, or as output units



A multilayer feed-forward neural network.

Defining a Network Topology

Before training can begin, the user must decide on the network topology by specifying the number of units in the input layer, the number of hidden layers (if more than one), the number of units in each hidden layer, and the number of units in the output layer. Normalizing the input values for each attribute measured in the training tuples will help speed up the learning phase.

Algorithm: Backpropagation. Neural network learning for classification or prediction, using the backpropagation algorithm.

Input:

- D , a data set consisting of the training tuples and their associated target values;
- l , the learning rate;
- $network$, a multilayer feed-forward network.

Output: A trained neural network.

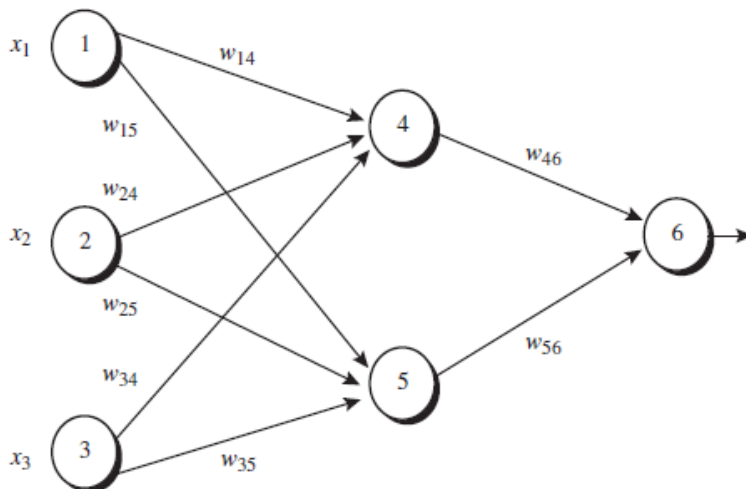
Method:

```

(1) Initialize all weights and biases in  $network$ ;
(2) while terminating condition is not satisfied {
(3)   for each training tuple  $X$  in  $D$  {
(4)     // Propagate the inputs forward:
(5)     for each input layer unit  $j$  {
(6)        $O_j = I_j$ ; // output of an input unit is its actual input value
(7)     for each hidden or output layer unit  $j$  {
(8)        $I_j = \sum_i w_{ij} O_i + \theta_j$ ; // compute the net input of unit  $j$  with respect to the
        previous layer,  $i$ 
(9)        $O_j = \frac{1}{1+e^{-I_j}}$ ; // compute the output of each unit  $j$ 
(10)    // Backpropagate the errors:
(11)    for each unit  $j$  in the output layer
(12)       $Err_j = O_j(1 - O_j)(T_j - O_j)$ ; // compute the error
(13)    for each unit  $j$  in the hidden layers, from the last to the first hidden layer
(14)       $Err_j = O_j(1 - O_j) \sum_k Err_k w_{jk}$ ; // compute the error with respect to the
        next higher layer,  $k$ 
(15)    for each weight  $w_{ij}$  in  $network$  {
(16)       $\Delta w_{ij} = (l) Err_j O_i$ ; // weight increment
(17)       $w_{ij} = w_{ij} + \Delta w_{ij}$ ; // weight update
(18)    for each bias  $\theta_j$  in  $network$  {
(19)       $\Delta \theta_j = (l) Err_j$ ; // bias increment
(20)       $\theta_j = \theta_j + \Delta \theta_j$ ; // bias update
(21)    } }

```

Example:



Initial input, weight, and bias values.

x_1	x_2	x_3	w_{14}	w_{15}	w_{24}	w_{25}	w_{34}	w_{35}	w_{46}	w_{56}	θ_4	θ_5	θ_6
1	0	1	0.2	-0.3	0.4	0.1	-0.5	0.2	-0.3	-0.2	-0.4	0.2	0.1

To compute the net input to the unit, each input connected to the unit is multiplied by its corresponding weight, and this is summed

$$I_j = \sum_i w_{ij} O_i + \theta_j,$$

The logistic, or sigmoid, function is used. Given the net input I_j to unit j , then O_j , the output of unit j , is computed as

$$O_j = \frac{1}{1 + e^{-I_j}}.$$

The net input and output calculations.

Unit j	Net input, I_j	Output, O_j
4	$0.2 + 0 - 0.5 - 0.4 = -0.7$	$1/(1 + e^{0.7}) = 0.332$
5	$-0.3 + 0 + 0.2 + 0.2 = 0.1$	$1/(1 + e^{-0.1}) = 0.525$
6	$(-0.3)(0.332) - (0.2)(0.525) + 0.1 = -0.105$	$1/(1 + e^{0.105}) = 0.474$

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

$$Err_j = O_j(1 - O_j)(T_j - O_j),$$

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

$$Err_j = O_j(1 - O_j) \sum_k Err_k w_{jk},$$

Calculation of the error at each node.

Unit j	Err_j
6	$(0.474)(1 - 0.474)(1 - 0.474) = 0.1311$
5	$(0.525)(1 - 0.525)(0.1311)(-0.2) = -0.0065$
4	$(0.332)(1 - 0.332)(0.1311)(-0.3) = -0.0087$

The weights and biases are updated to reflect the propagated errors. Weights are updated by the following equations, where Δw_{ij} is the change in weight w_{ij} :

$$\Delta w_{ij} = (l)Err_j O_i$$

$$w_{ij} = w_{ij} + \Delta w_{ij}$$

Biases are updated by the following equations below, where $\Delta \theta_j$ is the change in bias θ_j :

$$\Delta \theta_j = (l)Err_j$$

$$\theta_j = \theta_j + \Delta \theta_j$$

Calculations for weight and bias updating.

Weight or bias	New value
w_{46}	$-0.3 + (0.9)(0.1311)(0.332) = -0.261$
w_{56}	$-0.2 + (0.9)(0.1311)(0.525) = -0.138$
w_{14}	$0.2 + (0.9)(-0.0087)(1) = 0.192$
w_{15}	$-0.3 + (0.9)(-0.0065)(1) = -0.306$
w_{24}	$0.4 + (0.9)(-0.0087)(0) = 0.4$
w_{25}	$0.1 + (0.9)(-0.0065)(0) = 0.1$
w_{34}	$-0.5 + (0.9)(-0.0087)(1) = -0.508$
w_{35}	$0.2 + (0.9)(-0.0065)(1) = 0.194$
θ_6	$0.1 + (0.9)(0.1311) = 0.218$
θ_5	$0.2 + (0.9)(-0.0065) = 0.194$
θ_4	$-0.4 + (0.9)(-0.0087) = -0.408$

Prediction

“What if we would like to predict a continuous value, rather than a categorical label?” Numeric prediction is the task of predicting continuous (or ordered) values for given input. For example, we may wish to predict the salary of college graduates with 10 years of work experience, or the potential sales of a new product given its price. Most widely used approach for numeric prediction is regression. Regression analysis is a good choice when all of the predictor variables are continuous valued as well. Many problems can be solved by *linear regression*, and even more can be tackled by applying transformations to the variables so that a nonlinear problem can be converted to a linear one.

Linear Regression

Straight-line regression analysis involves a response variable, y , and a single predictor variable, x . It is the simplest form of regression, and models y as a linear function of x .

$$y = w_0 + w_1 x.$$

where the variance of y is assumed to be constant, and w_0 and w_1 are regression coefficients specifying the Y-intercept and slope of the line, respectively. w_1 can be calculated as

$$w_1 = \frac{\sum_{i=1}^{|D|} (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{|D|} (x_i - \bar{x})^2}$$

where \bar{x} is the mean value of $x_1, x_2, \dots, x_{|D|}$, and \bar{y} is the mean value of $y_1, y_2, \dots, y_{|D|}$. The coefficients w_0 and w_1 often provide good approximations to otherwise complicated regression equations

$$w_0 = \bar{y} - w_1 \bar{x}$$

UNIT 5: Cluster Analysis

Cluster Analysis

The process of grouping a set of physical or abstract objects into classes of similar objects is called clustering. A cluster is a collection of data objects that are similar to one another within the same cluster and are dissimilar to the objects in other clusters.

Although classification is an effective means for distinguishing groups or classes of objects, it requires the often costly collection and labeling of a large set of training tuples or patterns, which the classifier uses to model each group. It is often more desirable to proceed in the reverse direction: First partition the set of data into groups based on data similarity (e.g., using clustering), and then assign labels to the relatively small number of groups. Additional advantages of such a clustering-based process are that it is adaptable to changes and helps single out useful features that distinguish different groups.

Clustering is a challenging field of research in which its potential applications pose their own special requirements. The following are typical requirements of clustering in data mining:

Scalability: Many clustering algorithms work well on small data sets containing fewer than several hundred data objects; however, a large database may contain millions of objects. Clustering on a sample of a given large data set may lead to biased results. Highly scalable clustering algorithms are needed.

Ability to deal with different types of attributes: Many algorithms are designed to cluster interval-based (numerical) data. However, applications may require clustering other types of data, such as binary, categorical (nominal), and ordinal data, or mixtures of these data types.

Discovery of clusters with arbitrary shape: Many clustering algorithms determine clusters based on Euclidean or Manhattan distance measures. Algorithms based on such distance measures tend to find spherical clusters with similar size and density. However, a cluster could be of any shape. It is important to develop algorithms that can detect clusters of arbitrary shape.

Minimal requirements for domain knowledge to determine input parameters: Many clustering algorithms require users to input certain parameters in cluster analysis (such as the number of desired clusters). The clustering results can be quite sensitive to input parameters. Parameters are often difficult to determine, especially for data sets containing high-dimensional objects. This not only burdens users, but it also makes the quality of clustering difficult to control.

Ability to deal with noisy data: Most real-world databases contain outliers or missing, unknown, or erroneous data. Some clustering algorithms are sensitive to such data and may lead to clusters of poor quality.

Incremental clustering and insensitivity to the order of input records: Some clustering algorithms cannot incorporate newly inserted data (i.e., database updates) into existing clustering structures and, instead, must determine a new clustering from scratch. Some clustering algorithms are sensitive to the order of input data. That is, given a set of data objects, such an algorithm may return dramatically different clustering depending on the order of presentation of the input objects. It is important to develop incremental clustering algorithms and algorithms that are insensitive to the order of input.

High dimensionality: A database or a data warehouse can contain several dimensions or attributes. Many clustering algorithms are good at handling low-dimensional data, involving only two to three dimensions. Human eyes are good at judging the quality of clustering for up to three dimensions. Finding clusters of data objects in high dimensional space is challenging, especially considering that such data can be sparse and highly skewed.

Main memory-based clustering algorithms typically operate on either of the following two data structures.

Data matrix (or object-by-variable structure): This represents n objects, such as persons, with p variables (also called measurements or attributes), such as age, height, weight, gender, and so on. The structure is in the form of a relational table, or n -by- p matrix (n objects \times p variables):

$$\begin{bmatrix} x_{11} & \cdots & x_{1f} & \cdots & x_{1p} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ x_{i1} & \cdots & x_{if} & \cdots & x_{ip} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ x_{n1} & \cdots & x_{nf} & \cdots & x_{np} \end{bmatrix}$$

Dissimilarity matrix (or object-by-object structure): This stores a collection of proximities that are available for all pairs of n objects. It is often represented by an n -by- n table:

$$\begin{bmatrix} 0 & & & & & \\ d(2,1) & 0 & & & & \\ d(3,1) & d(3,2) & 0 & & & \\ \vdots & \vdots & \vdots & & & \\ d(n,1) & d(n,2) & \dots & \dots & 0 \end{bmatrix}$$

where $d(i, j)$ is the measured difference or dissimilarity between objects i and j . In general, $d(i, j)$ is a nonnegative number that is close to 0 when objects i and j are highly similar or “near” each other, and becomes larger the more they differ.

Interval-Scaled Variables

Interval-scaled variables are continuous measurements of a roughly linear scale. Typical examples include weight and height, latitude and longitude coordinates and weather temperature. The measurement unit used can affect the clustering analysis. For example, changing measurement units from meters to inches for height, or from kilograms to pounds for weight, may lead to a very different clustering structure. In general, expressing a variable in smaller units will lead to a larger range for that variable, and thus a larger effect on the resulting clustering structure. To help avoid dependence on the choice of measurement units, the data should be standardized. Standardizing measurements attempts to give all variables an equal weight.

To standardize measurements, one choice is to convert the original measurements to unitless variables. Given measurements for a variable f , this can be performed as follows.

1. Calculate the mean absolute deviation,

$$s_f = \frac{1}{n}(|x_{1f} - m_f| + |x_{2f} - m_f| + \dots + |x_{nf} - m_f|),$$

where x_{1f}, \dots, x_{nf} are n measurements of f , and m_f is the *mean* value of f , that is,

$$m_f = \frac{1}{n}(x_{1f} + x_{2f} + \dots + x_{nf}).$$

2. Calculate the standardized measurement, or z-score:

$$z_{if} = \frac{x_{if} - m_f}{s_f}.$$

The mean absolute deviation, s_f is more robust to outliers than the standard deviation, σ_f . When computing the mean absolute deviation $|x_{if} - m_f|$, the deviations from the mean are not squared; hence, the effect of outliers is somewhat reduced. There are more robust measures of dispersion, such as the *median absolute deviation*. However, the advantage of using the mean absolute deviation is that the z-scores of outliers do not become too small; hence, the outliers remain detectable.

After standardization, or without standardization in certain applications, the dissimilarity (or similarity) between the objects described by interval-scaled variables is typically computed based on the distance between each pair of objects. The most popular distance measure is Euclidean distance, which is defined as

$$d(i, j) = \sqrt{(x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2 + \cdots + (x_{in} - x_{jn})^2},$$

where $i = (x_{i1}, x_{i2}, \dots, x_{in})$ and $j = (x_{j1}, x_{j2}, \dots, x_{jn})$ are two n -dimensional data objects.

Binary Variables

A binary variable has only two states: 0 or 1, where 0 means that the variable is absent, and 1 means that it is present. If all binary variables are thought of as having the same weight, we have the 2-by-2 contingency table of where q is the number of variables that equal 1 for both objects i and j , r is the number of variables that equal 1 for object i but that are 0 for object j , s is the number of variables that equal 0 for object i but equal 1 for object j , and t is the number of variables that equal 0 for both objects i and j . The total number of variables is p ,

where $p = q + r + s + t$.

A contingency table for binary variables.

		object j		
		1	0	sum
object i	1	q	r	$q + r$
	0	s	t	$s + t$
	sum	$q + s$	$r + t$	p

A binary variable is symmetric if both of its states are equally valuable and carry the same weight; that is, there is no preference on which outcome should be coded as 0 or 1. One such example could be the attribute *gender* having the states *male* and *female*. Dissimilarity that is based on symmetric binary variables is called symmetric binary dissimilarity. Its dissimilarity (or

distance) measure, defined in equation below, can be used to assess the dissimilarity between objects i and j .

$$d(i, j) = \frac{r + s}{q + r + s + t}.$$

binary variable is asymmetric if the outcomes of the states are not equally important, such as the *positive* and *negative* outcomes of a disease *test*. By convention, we shall code the most important outcome, which is usually the rarest one, by 1 (e.g., *HIV positive*) and the other by 0 (e.g., *HIV negative*). Given two asymmetric binary variables, the agreement of two 1s (a positive match) is then considered more significant than that of two 0s (a negative match). Therefore, such binary variables are often considered “monary” (as if having one state). The dissimilarity based on such variables is called asymmetric binary dissimilarity, where the number of negative matches, t , is considered unimportant and thus is ignored in the computation, as shown in Equation

$$d(i, j) = \frac{r + s}{q + r + s}.$$

Categorical Variables

A categorical variable is a generalization of the binary variable in that it can take on more than two states. For example, *map color* is a categorical variable that may have, say, five states: *red*, *yellow*, *green*, *pink*, and *blue*. Let the number of states of a categorical variable be M . The states can be denoted by letters, symbols, or a set of integers, such as 1, 2, : : : , M . Notice that such integers are used just for data handling and do not represent any specific ordering.

The dissimilarity between two objects i and j can be computed based on the ratio of mismatches:

$$d(i, j) = \frac{p - m}{p},$$

where m is the number of *matches* (i.e., the number of variables for which i and j are in the same state), and p is the total number of variables. Weights can be assigned to increase the effect of m or to assign greater weight to the matches in variables having a larger number of states.

Ordinal Variables

A discrete ordinal variable resembles a categorical variable, except that the M states of the ordinal value are ordered in a meaningful sequence. Ordinal variables are very useful for registering subjective assessments of qualities that cannot be measured objectively. For example, professional ranks are often enumerated in a sequential order, such as *assistant*, *associate*, and *full* for professors. A continuous ordinal variable looks like a set of continuous data of an unknown scale; that is, the relative ordering of the values is essential but their actual magnitude is not. For example, the relative ranking in a particular sport (e.g., gold, silver, bronze) is often more essential than the actual values of a particular measure. Ordinal variables may also be obtained from the discretization of interval-scaled quantities by splitting the value range into a finite number of classes. The values of an ordinal variable can be mapped to *ranks*. For example, suppose that an ordinal variable f has M_f states. These ordered states define the ranking $1, 2, \dots, M_f$.

The treatment of ordinal variables is quite similar to that of interval-scaled variables when computing the dissimilarity between objects. Suppose that f is a variable from a set of ordinal variables describing n objects. The dissimilarity computation with respect to f involves the following steps:

1. The value of f for the i th object is x_{if} , and f has M_f ordered states, representing the ranking $1, 2, \dots, M_f$. Replace each x_{if} by its corresponding rank, r_{if} , $1, 2, \dots, M_f$.
2. Since each ordinal variable can have a different number of states, it is often necessary to map the range of each variable onto $[0.0, 1.0]$ so that each variable has equal weight. This can be achieved by replacing the rank r_{if} of the i th object in the f th variable by

$$z_{if} = \frac{r_{if} - 1}{M_f - 1}.$$

Dissimilarity can then be computed using any of the distance measures for interval-scaled variables, using z_{if} to represent the f value for the i th object.

Ratio-Scaled Variables

A ratio-scaled variable makes a positive measurement on a nonlinear scale, such as an exponential scale, approximately following the formula

$$Ae^{Bt} \quad \text{or} \quad Ae^{-Bt}$$

where A and B are positive constants, and t typically represents time.

[for example refer class notes]

Partitioning Methods

Partitioning methods: Given a database of n objects or data tuples, a partitioning method constructs k partitions of the data, where each partition represents a cluster and $k \leq n$. That is, it classifies the data into k groups, which together satisfy the following requirements: (1) each group must contain at least one object, and (2) each object must belong to exactly one group. Notice that the second requirement can be relaxed in some fuzzy partitioning techniques. Given k , the number of partitions to construct, a partitioning method creates an initial partitioning. It then uses an iterative relocation technique that attempts to improve the partitioning by moving objects from one group to another. The general criterion of a good partitioning is that objects in the same cluster are “close” or related to each other, whereas objects of different clusters are “far apart” or very different.

Centroid-Based Technique: The k -Means Method

The k -means algorithm takes the input parameter, k , and partitions a set of n objects into k clusters so that the resulting intracenter similarity is high but the intercluster similarity is low. Cluster similarity is measured in regard to the mean value of the objects in a cluster, which can be viewed as the cluster’s centroid or center of gravity.

The k -means algorithm proceeds as follows.

First, it randomly selects k of the objects, each of which initially represents a cluster mean or center. For each of the remaining objects, an object is assigned to the cluster to which it is the most similar, based on the distance between the object and the cluster mean. It then computes the new mean for each cluster. This process iterates until the criterion function converges. Typically, the square-error criterion is used, defined as

$$E = \sum_{i=1}^k \sum_{p \in C_i} |p - m_i|^2,$$

where E is the sum of the square error for all objects in the data set; p is the point in space representing a given object; and m_i is the mean of cluster C_i . In other words, for each object in each cluster, the distance from the object to its cluster center is squared, and the distances are summed. The algorithm attempts to determine k partitions that minimize the square-error function. It works well when the clusters are compact clouds that are rather well separated from one another. The method is relatively scalable and efficient in processing large data sets because the computational complexity of the algorithm is $O(nkt)$, where n is the total number of objects, k is the number of clusters, and t is the number of iterations. Normally, $k \ll n$ and $t \ll n$. The method often terminates at a local optimum. The k -means method, however, can be applied only when the mean of a cluster is defined. This may not be the case in some applications, such as when data with categorical attributes are involved. The necessity for users to specify k , the number of clusters, in advance can be seen as a disadvantage. The k -means method is not suitable for discovering clusters with nonconvex shapes or clusters of very different size. Moreover, it is sensitive to noise and outlier data points because a small number of such data can substantially influence the mean value.

Algorithm: k-means. The k -means algorithm for partitioning, where each cluster's center is represented by the mean value of the objects in the cluster.

Input: k : the number of clusters, D : a data set containing n objects.

Output: A set of k clusters.

Method:

- (1) arbitrarily choose k objects from D as the initial cluster centers;
- (2) repeat
- (3) (re)assign each object to the cluster to which the object is the most similar, based on the mean value of the objects in the cluster;
- (4) update the cluster means, i.e., calculate the mean value of the objects for each cluster;
- (5) until no change;

Representative Object-Based Technique: The k-Medoids Method

The k -means algorithm is sensitive to outliers because an object with an extremely large value may substantially distort the distribution of data. This effect is particularly exacerbated due to the use of the square-error function. Instead of taking

the mean value of the objects in a cluster as a reference point, we can pick actual

objects to represent the clusters, using one representative object per cluster. Each remaining object is clustered with the representative object to which it is the most similar. The

partitioning method is then performed based on the principle of minimizing the sum of the dissimilarities between each object and its corresponding reference point. That is, an

absolute-error criterion is used, defined as

$$E = \sum_{j=1}^k \sum_{p \in C_j} |p - o_j|,$$

where E is the sum of the absolute error for all objects in the data set; p is the point in space representing a given object in cluster C_j ; and o_j is the representative object of C_j . In general, the algorithm iterates until, eventually, each representative object is actually the medoid, or most centrally located object, of its cluster. This is the basis of the k -medoids method for grouping n objects into k clusters. Let's look closer at k -medoids clustering. The initial representative objects (or seeds) are chosen arbitrarily. The iterative process of replacing representative objects by nonrepresentative objects continues as long as the quality of the resulting clustering is improved. This quality is estimated using a cost function that measures the average dissimilarity between an object and the representative object of its cluster. To determine whether a nonrepresentative object, o_{random} , is a good replacement for a current representative object, o_j , the following four cases are examined for each of the nonrepresentative objects, p , as illustrated in Figure .

Case 1: p currently belongs to representative object, o_j . If o_j is replaced by o_{random} as a representative object and p is closest to one of the other representative objects, o_i , $i \neq j$, then p is reassigned to o_i .

Case 2: p currently belongs to representative object, o_j . If o_j is replaced by o_{random} as a representative object and p is closest to o_{random} , then p is reassigned to o_{random} .

Case 3: p currently belongs to representative object, o_i , $i \neq j$. If o_j is replaced by o_{random} as a representative object and p is still closest to o_i , then the assignment does not change.

Case 4: p currently belongs to representative object, o_i , $i \neq j$. If o_j is replaced by o_{random} as a representative object and p is closest to o_{random} , then p is reassigned to o_{random} .

Each time a reassignment occurs, a difference in absolute error, E , is contributed to the cost function. Therefore, the cost function calculates the difference in absolute-error value if a current representative object is replaced by a nonrepresentative object. The total cost of swapping is the sum of costs incurred by all nonrepresentative objects. If the total cost is negative, then o_i is replaced or swapped with o_{random} since the actual absolute error E would be reduced. If the total

cost is positive, the current representative object, o_j , is considered acceptable, and nothing is changed in the iteration.

PAM(Partitioning Around Medoids) was one of the first k -medoids algorithms introduced. It attempts to determine k partitions for n objects. After an initial random selection of k representative objects, the algorithm repeatedly tries to make a better choice of cluster representatives. All of the possible pairs of objects are analysed, where one object in each pair is considered a representative object and the other is not. The quality of the resulting clustering is calculated for each such combination. An object, o_j , is replaced with the object causing the greatest reduction in error. The set of best objects for each cluster in one iteration forms the representative objects for the next iteration. The final set of representative objects are the respective medoids of the clusters.

Algorithm: k -medoids. PAM, a k -medoids algorithm for partitioning based on medoid or central objects.

Input: k : the number of clusters, D : a data set containing n objects.

Output: A set of k clusters.

Method:

- (1) arbitrarily choose k objects in D as the initial representative objects or seeds;
- (2) repeat
- (3) assign each remaining object to the cluster with the nearest representative object;
- (4) randomly select a nonrepresentative object, o_{random} ;
- (5) compute the total cost, S , of swapping representative object, o_j , with o_{random} ;
- (6) if $S < 0$ then swap o_j with o_{random} to form the new set of k representative objects;
- (7) until no change;

The k -medoids method is more robust than k -means in the presence of noise and outliers, because a medoid is less influenced by outliers or other extreme values than a mean. However, its processing is more costly than the k -means method. Both methods require the user to specify k , the number of clusters.

Partitioning Methods in Large Databases: From k -Medoids to CLARANS

A typical k -medoids partitioning algorithm like PAM works effectively for small data sets, but does not scale well for large data sets. To deal with larger data sets, a *sampling*-based method, called CLARA (Clustering LARge Applications), can be used.

The idea behind CLARA is as follows: Instead of taking the whole set of data into consideration, a small portion of the actual data is chosen as a representative of the data. Medoids are then chosen from this sample using PAM. If the sample is selected in a fairly random manner, it should closely represent the original data set. The representative objects (medoids) chosen will likely be similar to those that would have been chosen from the whole data set. CLARA draws multiple samples of the data set, applies PAM on each sample, and returns its best clustering as the output. As expected, CLARA can deal with larger data sets than PAM.

The effectiveness of CLARA depends on the sample size. Notice that PAM searches for the best k medoids among a given data set, whereas CLARA searches for the best k medoids among the *selected sample* of the data set. CLARA cannot find the best clustering if any of the best sampled medoids is not among the best k medoids. That is, if an object o_i is one of the best k medoids but is not selected during sampling, CLARA will never find the best clustering. This is, therefore, a trade-off for efficiency. A good clustering based on sampling will not necessarily represent a good clustering of the whole data set if the sample is biased.

A k -medoids type algorithm called CLARANS (Clustering Large Applications based upon RANdomizedSearch) was proposed, which combines the sampling technique with PAM. However, unlike CLARA, CLARANS does not confine itself to any sample at any given time.

While CLARA has a fixed sample at each stage of the search, CLARANS draws a sample with some randomness in each step of the search. Conceptually, the clustering process can be viewed as a search through a graph, where each node is a potential solution (a set of k medoids). Two nodes are *neighbors* (that is, connected by an arc in the graph) if their sets differ by only one object. Each node can be assigned a cost that is defined by the total dissimilarity between every object and the medoid of its cluster. At each step, PAM examines all of the neighbors of the current node in its search for a minimum cost solution. The current node is then replaced by the neighbor with the largest descent in costs. Because CLARA works on a sample of the entire dataset, it examines fewer neighbors and restricts the search to subgraphs that are smaller than the original graph. While CLARA draws a sample of nodes at the beginning of a search, CLARANS dynamically draws a random sample of neighbors in each step of a search. The number of neighbors to be randomly sampled is restricted by a user-specified parameter. In this way, CLARANS does not confine the search to a localized area. If a better neighbor is found (i.e., having a lower error), CLARANS moves to the neighbor's node and the process starts again; otherwise, the current clustering produces a local minimum. If a local minimum is found, CLARANS starts with new randomly selected nodes in search for a new local minimum. Once a user-specified number of local minima has been found, the algorithm outputs, as a solution, the best local minimum, that is, the local minimum having the lowest cost.

Chameleon: A Hierarchical Clustering Algorithm Using Dynamic Modeling

Chameleon is a hierarchical clustering algorithm that uses dynamic modeling to determine the similarity between pairs of clusters. It was derived based on the observed weaknesses of two

hierarchical clustering algorithms: ROCK and CURE. ROCK and related schemes emphasize cluster interconnectivity while ignoring information regarding cluster proximity. CURE and related schemes consider cluster proximity yet ignore cluster interconnectivity. In Chameleon, cluster similarity is assessed based on how well-connected objects are within a cluster and on the proximity of clusters. That is, two clusters are merged if their interconnectivity is high and they are close together. Thus, Chameleon does not depend on a static, user-supplied model and can automatically adapt to the internal characteristics of the clusters being merged. The merge process facilitates the discovery of natural and homogeneous clusters and applies to all types of data as long as a similarity function can be specified. Chameleon uses a k -nearest-neighbor graph approach to construct a sparse graph, where each vertex of the graph represents a data object, and there exists an edge between two vertices (objects) if one object is among the k -most-similar objects of the other. The edges are weighted to reflect the similarity between objects. Chameleon uses a graph partitioning algorithm to partition the k -nearest-neighbor graph into a large number of relatively small subclusters. It then uses an agglomerative hierarchical clustering algorithm that repeatedly merges subclusters based on their similarity. To determine the pairs of most similar subclusters, it takes into account both the interconnectivity as well as the closeness of the clusters. The k -nearest-neighbor graph captures the concept of neighborhood dynamically: the neighborhood radius of an object is determined by the density of the region in which the object resides. In a dense region, the neighborhood is defined narrowly; in a sparse region, it is defined more widely.

The graph-partitioning algorithm partitions the k -nearest-neighbor graph such that it minimizes the edge cut. That is, a cluster C is partitioned into subclusters C_i and C_j so as to minimize the *weight of the edges* that would be cut should C be bisected into C_i and C_j . Edge cut is denoted $EC(C_i, C_j)$ and assesses the *absolute* interconnectivity between clusters C_i and C_j .

Chameleon determines the similarity between each pair of clusters C_i and C_j according to their *relative interconnectivity*, $RI(C_i, C_j)$, and their *relative closeness*, $RC(C_i, C_j)$:

The relative interconnectivity, $RI(C_i, C_j)$, between two clusters, C_i and C_j , is defined as the absolute interconnectivity between C_i and C_j , normalized with respect to the internal interconnectivity of the two clusters, C_i and C_j . That is,

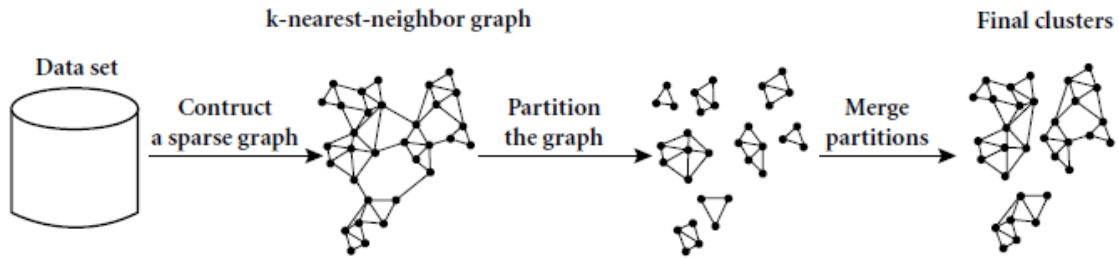
$$RI(C_i, C_j) = \frac{|EC_{\{C_i, C_j\}}|}{\frac{1}{2}(|EC_{C_i}| + |EC_{C_j}|)},$$

where $EC_{\{C_i, C_j\}}$ is the edge cut, defined as above, for a cluster containing both C_i and C_j . Similarly, EC_{C_i} (or EC_{C_j}) is the minimum sum of the cut edges that partition C_i (or C_j) into two roughly equal parts.

The relative closeness, $RC(C_i, C_j)$, between a pair of clusters, C_i and C_j , is the absolute closeness between C_i and C_j , normalized with respect to the internal closeness of the two clusters, C_i and C_j . It is defined as

$$RC(C_i, C_j) = \frac{\bar{S}_{EC\{C_i, C_j\}}}{\frac{|C_i|}{|C_i|+|C_j|}\bar{S}_{EC_{C_i}} + \frac{|C_j|}{|C_i|+|C_j|}\bar{S}_{EC_{C_j}}},$$

where $\bar{S}_{EC\{C_i, C_j\}}$ is the average weight of the edges that connect vertices in C_i to vertices in C_j , and $\bar{S}_{EC_{C_i}}$ (or $\bar{S}_{EC_{C_j}}$) is the average weight of the edges that belong to the mincut bisector of cluster C_i (or C_j).



DBSCAN: A Density-Based Clustering Method Based on Connected Regions with Sufficiently High Density

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a density based clustering algorithm. The algorithm grows regions with sufficiently high density into clusters and discovers clusters of arbitrary shape in spatial databases with noise.

It defines a cluster as a maximal set of density-connected points. The basic ideas of density-based clustering involve a number of new definitions. We intuitively present these definitions, and then follow up with an example.

- The neighborhood within a radius ε of a given object is called the ε -neighborhood of the object.
- If the ε -neighborhood of an object contains at least minimum number, MinPts, of objects, then the object is called a core object.
- Given a set of objects, D, we say that an object p is directly density-reachable from object q if p is within the ε -neighborhood of q , and q is a core object.
- An object p is density-reachable from object q with respect to ε and MinPts in a set of objects, D, if there is a chain of objects p_1, \dots, p_n , where $p_1 = q$ and $p_n = p$ such that p_{i+1} is directly density-reachable from p_i with respect to ε and MinPts, for $1 \leq i \leq n$, $p_i \in D$.

- An object p is density-connected to object q with respect to ϵ and MinPts in a set of objects, D , if there is an object $o \in D$ such that both p and q are density-reachable from o with respect to ϵ and MinPts .

Density reachability is the transitive closure of direct density reachability, and this relationship is asymmetric. Only core objects are mutually density reachable. Density connectivity, however, is a symmetric relation. A density-based cluster is a set of density-connected objects that is maximal with respect to density-reachability. Every object not contained in any cluster is considered to be noise.

DBSCAN searches for clusters by checking the ϵ -neighborhood of each point in the database. If the ϵ -neighborhood of a point p contains more than MinPts , a new cluster with p as a core object is created. DBSCAN then iteratively collects directly density-reachable objects from these core objects, which may involve the merge of a few density-reachable clusters. The process terminates when no new point can be added to any cluster.

OPTICS: Ordering Points to Identify the Clustering Structure

Although DBSCAN can cluster objects given input parameters such as ϵ and MinPts , it still leaves the user with the responsibility of selecting parameter values that will lead to the discovery of acceptable clusters. Actually, this is a problem associated with many other clustering algorithms. Such parameter settings are usually empirically set and difficult to determine, especially for real-world, high-dimensional data sets. Most algorithms are very sensitive to such parameter values: slightly different settings may lead to very different clusterings of the data. Moreover, high-dimensional real data sets often have very skewed distributions, such that their intrinsic clustering structure may not be characterized by global density parameters. To help overcome this difficulty, a cluster analysis method called OPTICS was proposed. Rather than produce a data set clustering explicitly, OPTICS computes an augmented cluster ordering for automatic and interactive cluster analysis. This ordering represents the density-based clustering structure of the data. It contains information that is equivalent to density-based clustering obtained from a wide range of parameter settings. The cluster ordering can be used to extract basic clustering information (such as cluster centers or arbitrary-shaped clusters) as well as provide the intrinsic clustering structure.

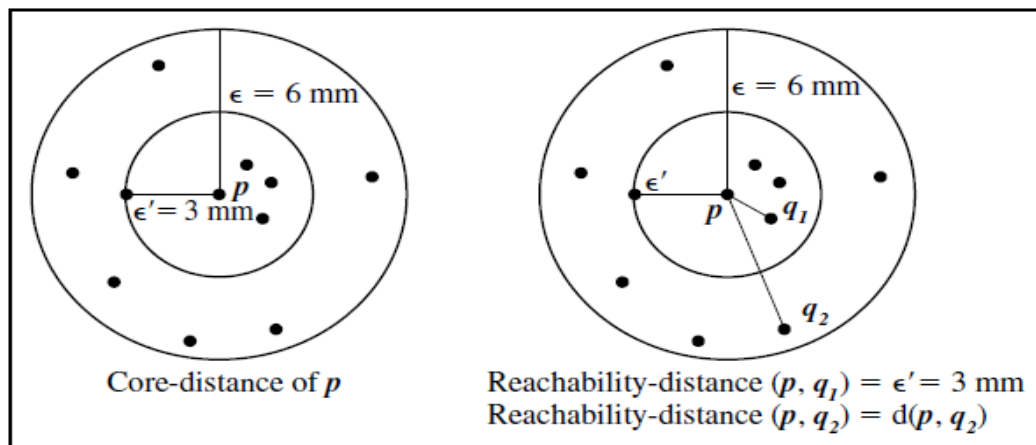
By examining DBSCAN, we can easily see that for a constant MinPts value, density based clusters with respect to a higher density (i.e., a lower value for ϵ) are completely contained in density-connected sets obtained with respect to a lower density. Recall that the parameter ϵ is a distance—it is the neighborhood radius. Therefore, in order to produce a set or ordering of density-based clusters, we can extend the DBSCAN algorithm to process a set of distance parameter values at the same time. To construct the different clusterings simultaneously, the objects should be processed in a specific order. This order selects an object that is density-reachable with respect to the lowest ϵ value so that clusters with higher density (lower ϵ) will be finished first.

Based on this idea, two values need to be stored for each object—core-distance and reachability-distance:

The core-distance of an object p is the smallest ϵ' value that makes $\{p\}$ a core object. If p is not a core object, the core-distance of p is undefined.

The reachability-distance of an object q with respect to another object p is the greater value of the core-distance of p and the Euclidean distance between p and q . If p is not a core object, the reachability-distance between p and q is undefined.

The OPTICS algorithm creates an ordering of the objects in a database, additionally storing the core-distance and a suitable reachability distance for each object. An algorithm was proposed to extract clusters based on the ordering information produced by OPTICS. Such information is sufficient for the extraction of all density-based clusterings with respect to any distance ϵ_0 that is smaller than the distance ϵ used in generating the order.



STING: STatisticalINformation Grid

STING is a grid-based multiresolution clustering technique in which the spatial area is divided into rectangular cells. There are usually several levels of such rectangular cells corresponding to different levels of resolution, and these cells form a hierarchical structure: each cell at a high level is partitioned to form a number of cells at the next lower level. Statistical information regarding the attributes in each grid cell (such as the mean, maximum, and minimum values) is precomputed and stored. These statistical parameters are useful for query processing, as described below.

Statistical parameters of higher-level cells can easily be computed from the parameters of the lower-level cells. These parameters include the following: the attribute-independent parameter, *count*; the attribute-dependent parameters, *mean*, *stdev* (standard deviation), *min* (minimum), *max* (maximum); and the type of *distribution* that the attribute value in the cell follows, such as *normal*, *uniform*, *exponential*, or *none* (if the distribution is unknown). When the data are loaded into the database, the parameters *count*, *mean*, *stdev*, *min*, and *max* of the bottom-level cells are calculated directly from the data. The value of *distribution* may either be assigned by the user if the distribution type is known beforehand or obtained by hypothesis tests such as the χ^2 test. The type of distribution of a higher-level cell can be computed based on the majority of distribution types of its corresponding lower-level cells in conjunction with a threshold filtering process. If

the distributions of the lower level cells disagree with each other and fail the threshold test, the distribution type of the high-level cell is set to *none*.

The statistical parameters can be used in a top-down, grid-based method as follows. First, a layer within the hierarchical structure is determined from which the query-answering process is to start. This layer typically contains a small number of cells. For each cell in the current layer, we compute the confidence interval (or estimated range of probability) reflecting the cell's relevancy to the given query. The irrelevant cells are removed from further consideration.

Processing of the next lower level examines only the remaining relevant cells. This process is repeated until the bottom layer is reached. At this time, if the query specification is met, the regions of relevant cells that satisfy the query are returned. Otherwise, the data that fall into the relevant cells are retrieved and further processed until they meet the requirements of the query.

STING offers several advantages: (1) the grid-based computation is *query-independent*, because the statistical information stored in each cell represents the summary information of the data in the grid cell, independent of the query; (2) the grid structure facilitates parallel processing and incremental updating; and (3) the method's efficiency is a major advantage:

STING goes through the database once to compute the statistical parameters of the cells, and hence the time complexity of generating clusters is $O(n)$, where n is the total number of objects. After generating the hierarchical structure, the query processing time is $O(g)$, where g is the total number of grid cells at the lowest level, which is usually much smaller than n .

CLIQUE: A Dimension-Growth Subspace Clustering Method

CLIQUE (CLustering In QUEst) was the first algorithm proposed for dimension-growth subspace clustering in high-dimensional space. In dimension-growth subspace clustering, the clustering process starts at single-dimensional subspaces and grows upward to higher-dimensional ones. Because CLIQUE partitions each dimension like a grid structure and determines whether a cell is dense based on the number of points it contains, it can also be viewed as an integration of density-based and grid-based clustering methods.

However, its overall approach is typical of subspace clustering for high-dimensional space, and so it is introduced in this section.

The ideas of the CLIQUE clustering algorithm are outlined as follows. Given a large set of multidimensional data points, the data space is usually not uniformly occupied by the data points. CLIQUE's clustering identifies the sparse and the "crowded" areas in space (or units), thereby discovering the overall distribution patterns of the data set.

A unit is dense if the fraction of total data points contained in it exceeds an input model parameter. In CLIQUE, a cluster is defined as a maximal set of connected dense units.

The ideas of the CLIQUE clustering algorithm are outlined as follows.

Given a large set of multidimensional data points, the data space is usually not uniformly occupied by the data points. CLIQUE's clustering identifies the sparse and the "crowded" *areas in space* (or units), thereby discovering the overall distribution patterns of the data set.

A unit is dense if the fraction of total data points contained in it exceeds an input model parameter. In CLIQUE, a cluster is defined as a maximal set of *connected dense units*.

The identification of the candidate search space is based on the Apriori property used in association rule mining.⁸ In general, the property employs prior knowledge of items in the search space so that portions of the space can be pruned. The property, adapted for CLIQUE, states the following: If a k -dimensional unit is dense, then so are its projections in $(k-1)$ -dimensional space. That is, given a k -dimensional candidate dense unit, if we check its $(k-1)$ th projection units and find any that are not dense, then we know that the k th dimensional unit cannot be dense either. Therefore, we can generate potential or candidate dense units in k -dimensional space from the dense units found in $(k-1)$ -dimensional space. In general, the resulting space searched is much smaller than the original space. The dense units are then examined in order to determine the clusters.

In the second step, CLIQUE generates a minimal description for each cluster as follows. For each cluster, it determines the maximal region that covers the cluster of connected dense units. It then determines a minimal cover (logic description) for each cluster.